FACULTAD DE CIENCIAS

Simulación de la disociación de la molécula de H_2 bajo el efecto de un campo eléctrico usando un modelo sencillo

(Simulation of the dissociation of the H_2 molecule under an electric field using a simple quantum mechanical model)

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: Álvaro Soza Marañón
Director: Pablo García Fernández

Septiembre 2016

Índice general

Resumen

El objetivo de este trabajo consiste en estudiar un sistema cuántico sencillo, pero con una gran fenomenología como es la molécula de hidrógeno. Mediante la simulación de primeros principios, se ha encontrado la curva de disociación, las bandas y el hamiltoniano para dos configuraciones distintas. Se ha comprobado que el modelo sencillo a dos niveles no es el adecuado para entender todos los fenómenos presentes en la molécula, siendo necesario considerar más niveles.

En un principio se pretendía estudiar la dinámica temporal de la molécula tras interaccionar con un campo eléctrico, así como explorar posibles estados excitados, poder llegar a ver la disociación molecular y los mecanismos implicados en la misma mediante la implementación de un código desarrollado en Python. Sin embargo, debido a dificultades técnicas en la parametrización del modelo esto no se ha llevado acabo, aunque el código implementado, se ha testeado con resultados satisfactorios, permitiendo realizar una continuación de este trabajo.

Palabras clave: Estructura electrónica, disociación, matriz densidad

Abstract

The main goal of this project is to study a simple quantum system with interesting and varied phenomenology, as it is the hydrogen molecule. By simulating it at the first principles level the hydrogen molecule's dissociation curve, its bands and its Hamiltonian were found at two different simulation levels. It has been confirmed that the simple two-level model is not suitable to understand all the physical phenomenons of the molecule, therefore a larger number of levels it is needed.

The initial goal of this work was to study the time dependent dynamics of the molecule after interacting with an electric field, as well as to explore excited states and be able to observe the molecular dissociation and the mechanisms involved by means of the implementation of a Python code. However, this has not been possible due to technical problems. The Python code has been tested with satisfactory results that will allow us to continue working on this project.

Key words: Electronic structure, dissociation, density matrix

Capítulo 1

Introducción

1.1. Importancia de la física computacional

Los modelos sencillos se emplean frecuentemente en física y química con el fin de entender cualitativamente las diferentes interacciones que se dan en el sistema. En la química, su uso está muy extendido debido a que estos modelos sencillos se emplean para determinar como será el enlace formado o como será el crecimiento de un cristal, si estudiamos el estado sólido. La predicción de la formación del enlace es fundamental, ya que la geometría molecular determina en gran manera las propiedades que mostrará el sistema.

Muchos de estos modelos de enlace empleados en química, tienen en cuenta de forma simplificada la estructura electrónica, cómo la teoría de Lewis basada en la idea de conseguir la estabilidad mediante el llenado de la capa de valencia, y que consiguen determinar, en algunos casos, el ángulo del enlace y es capaz de explicar enlaces múltiples. Existen mejoras a este tipo de modelos, introduciendo el concepto de orbital y buscando la menor repulsión entre los electrones.

Otros modelos solo consideran las interacciones electrostáticas entre los iones o átomos y la repulsión entre las nubes electrónicas, como ocurre en el modelo de Born-Mayer. Sin embargo, este modelo solo es aplicable a sólidos iónicos debido a que la carga se encuentra localizada, mientras que en el caso del enlace covalente y metálico no funciona, debido a que en ambos tipos de enlace, las nubes electrónicas se encuentran mas difundidas por el sistema.

Todos estos modelos no son aplicables a todas las moléculas, no explican adecuadamente el enlace y tampoco son capaces de explicar ningún estado excitado. Además de no ser capaces de predecir cuantitativamente ningún experimento. Para poder obtener una comprensión adecuada de la naturaleza del enlace, distancias de equilibrio, distribución de las nubes de carga, poder predecir con fiabilidad la geometría molecular, así como la dinámica bajo la acción de campos externos, es necesario la resolución de la ecuación de Schrödinger.

Aquí entra en juego física computacional, dado que la ecuación de Schrödinger no es resoluble de forma analítica para sistemas con mas de dos cuerpos, de forma que tiene que ser resuelta por métodos numéricos. La resolución nos proporcionará datos precisos acerca de lo que ocurre en el sistema y, en muchos casos, llegar allí donde es difícil llegar con la experimentación.

A pesar de la precisión de los cálculos obtenidos mediante la simulación, siempre harán falta modelos, sencillos o no, pero que sean adecuados para la correcta interpretación y comprensión de los fenómenos que tienen lugar en el sistema de estudio.

1.2. Método para grandes sistemas con estructura fija

En la actualidad los métodos computacionales en la simulación de sistemas se ven limitados, entre otras cosas, por el número de átomos que conforman el sistema.

Los métodos que se emplean para sistemas grandes son, en general, útiles para tener una idea de como el sistema se comporta en el espacio de fases. Pero en muchos casos, estos métodos no reproducen de forma fiel el sistema. Esto es debido a la necesidad de introducir importantes aproximaciones para resolver las ecuaciones. Algunos ejemplos de este tipo de métodos son: enlace fuerte, hamiltonianos modelo, hamiltoniano de Ising y Heisenberg. Durante los últimos años en la Universidad de Cantabria se han estado desarrollando métodos de simulación de estructura electrónica para grandes sistemas que van más allá que estos modelos comúnmente aplicados. Aún así tienen limitaciones como la necesidad de una estructura de enlaces fija. en este trabajo vamos a comprobar los límites del método aplicándolo a la disociación de una molécula sencilla. En el presente trabajo, se pretende utilizar un método aplicable a sistema con gran número de elementos, siempre que se mantengan la estructura y existan pequeñas variaciones en la densidad electrónica del mismo.

1.3. La molécula de H_2

En el presente trabajo se ha elegido la molécula de hidrógeno por un sistema cuántico muy sencillo, pero que posee una gran cantidad de fenómenos físicos interesantes. El esquema de niveles enlazante-antienlazante permite entender experimentos de fotoquímica, mientras que la curva de disociación nos lleva de un estado diamagnético a tener dos átomos disociados, que se corresponde con un estado paramagnético, pasando por fases intermedias, antiferromagnéticas.

En la literatura donde es habitual encontrar el modelo sencillo a dos niveles para explicar el enlace en esta molécula, muchos autores asumen que los dos orbitales moleculares tienen un carácter perfectamente definido. El objetivo principal de este trabajo es estudiar la validez de este modelo con el rigor permitidos por las modernas técnicas de simulación de moléculas y sólidos.

Capítulo 2

Teoría

2.1. Introducción

En este capítulo comenzaremos introduciendo los métodos de primeros principios, en particular la Teoría del funcional de la densidad, empleado para la resolución de la ecuación de Schrödinger. Se justificará, a partir de las limitaciones de este método, las técnicas de segundos principios, que permite escalar en el número de átomos del sistema sin un gran coste computacional. Ambos métodos tienen como magnitud central la densidad del sistema y veremos, con la evolución temporal de esta magnitud, cómo se pueden conocer propiedades del sistema fuera del equilibrio.

Cómo en todo problema de mecánica cuántica, necesitamos una base en la que poder trabajar. En este caso se ha optado por una base de funciones de Wannier que, cómo veremos, tiene propiedades que las hacen idóneas para nuestro problema.

Por último, aplicando el teorema de Hellmann-Feynman dentro del esquema de segundos principios, encontraremos una expresión para la fuerza a la que se ve sometida el sistema en un estado excitado con respecto a uno de referencia.

2.2. Introducción a la Teoría del funcional de la densidad

Para describir la estructura electrónica de un sistema formado por electrones y núcleos hay que trabajar dentro del marco de la mecánica cuántica. La ecuación que rige la mecánica del sistema descrito en un régimen no relativista es la ecuación de Schrödinger,

$$i\hbar\frac{d}{dt}\Psi\left(\vec{r},t\right) = \hat{\mathcal{H}}_{\mathrm{TOTAL}}\Psi\left(\vec{r},t\right) \tag{2.1}$$

donde Ψ es la función de onda, que representa el estado en el que se encuentra el sistema. Aunque el objetivo del trabajo es resolver la ecuación de Schrödinger dependiente del tiempo ahora nos centraremos en la ecuación de Schrödinger para el estado estacionario (Ec.2.2) que es donde mas trabajos existen para encontrar la solución molecular.

$$\hat{\mathcal{H}}_{\mathrm{TOTAL}}\Psi\left(\vec{r},t\right) = E\Psi\left(\vec{r},t\right) \tag{2.2}$$

La resolución de la ecuación de Schrödinger no resulta sencilla y su dificultad reside en el número de partículas que conformen el sistema y de la forma del potencial. En el sistema planteado existen interacciones entre muchos cuerpos lo que hace irresoluble (2.1) de forma analítica, por lo que hay que emplear aproximaciones. La más importante, que se emplea de forma habitual en la literatura, es la aproximación de Born-Oppenheimer, que nos permite desacoplar el movimiento nuclear y electrónico. La separación de la dinámica electrónica de la

nuclear tiene su justificación en la gran diferencia de masa que existe entre los nucleones y los electrones y en cómo responden estos últimos a cambios en la configuración nuclear. Para una explicación detallada de dicha aproximación ver las siguientes referencias [1, 7]. Dentro de esta aproximación, el hamiltoniano que recoge todas las interacciones del sistema constituido por N_e electrones y N_n nucleones, para una configuración en la que los núcleos permanecen fijos, es el siguiente

$$\begin{aligned}\hat{\mathcal{H}} &= \hat{T}_e + \hat{V_{ne}} + \hat{V_{ee}} + \hat{V_{nn}} = \\ &= -\sum_i^{N_e} \frac{1}{2}\vec{\nabla_i^2} - \sum_a^{N_n}\sum_i^{N_e} \frac{Z_a}{|\vec{R_a} - \vec{r_i}|} + \sum_i^{N_e}\sum_{i>j}^{N_e} \frac{1}{|r_i \vec{-} \vec{r_j}|} + \sum_a^{N_n}\sum_{b>a}^{N_n} \frac{Z_a Z_b}{|\vec{R_a} - \vec{R_b}|}\end{aligned} \tag{2.3}$$

En esta expresión, escrita en unidades atómicas, el primer término representa la energía cinética de los electrones, el segundo recoge las interacciones de cada electrón con el resto de los núcleos, el tercer término representa la interacción de cada electrón con el resto y por último tenemos la interacciones entre los núcleos del sistema. El problema planteado con el hamiltoniano Ec. (2.3) de nuevo no es resoluble analíticamente por lo que hay que recurrir a métodos numéricos para su resolución.

Los métodos numéricos que tratan de resolver Ec.(2.1) son variados y dependen del marco teórico usado para describir el problema de forma que se pueda resolver computacionalmente. Estos métodos se denominan de primeros principios debido a que sólo emplean el uso de constantes fundamentales. Los métodos denominados de función de onda, como Hartree-Fock, se basan en aproximar la función de onda del sistema, realizando el calculo con el hamiltoniano exacto. Sin embargo, una manera mas eficiente de afrontar el problema es crear métodos que realizan aproximaciones sobre el hamiltoniano, en vez sobre la función de onda, que consideran exacta. El método más representativo, dentro de este último grupo, es la Teoría del funcional de la densidad, conocida como DFT, por sus siglas en inglés. Es el método computacional más usado para la resolución de la estructura electrónica en física y química [7]. La idea fundamental de la DFT reside en expresar las distintas magnitudes en la Ec.(2.1) como funcionales de la densidad electrónica, es decir, que todas las magnitudes del sistema se pueden expresar como funciones cuyo argumento sea la densidad electrónica Ec.(2.4)

$$\rho(\vec{r}) = \int \mid \Psi(\vec{r_1}, \vec{r_2}, \ldots, \vec{r_N}) \mid^2 d\vec{r_2} \ldots \vec{r_N} \tag{2.4}$$

donde $\Psi(\vec{r_1}, \vec{r_2}, \ldots, \vec{r_N})$ es la función de onda que representa el estado del sistema. La forma de dicha función es complicada, dependiendo de las coordenadas espaciales de los N_e electrones que conforman el sistema. Como se verá mas adelante, (Sec. 2.2.1), los teoremas de Hohenberg y Kohn demuestran que la densidad electrónica, $\rho(\vec{r})$, contiene la misma información que la función de onda, $\Psi(\vec{r_1}, \vec{r_2}, \ldots, \vec{r_N})$.

Las ventajas de desarrollar un marco teórico basado en la densidad electrónica son las siguientes:

- La densidad electrónica es un observable, por lo que se puede realizar una comparación directa con datos experimentales.
- Es una función real y solo depende de tres coordenadas espaciales, mientras que la función de onda que representa al sistema es una función compleja y depende de $3N_e$ variables espaciales, lo que hace que su tratamiento no sea sencillo.
- Juega un papel similar a la función de onda, ψ, ya que nos permite conocer el número de electrones, N_e, las posiciones nucleares y las cargas de estas ultimas.

De forma histórica existen intentos de describir el sistema en términos de la densidad, la primera aproximación de $T[\rho]$ fue realizada en 1928 por Thomas y Fermi considerando un gas homogéneo de electrones, sin embargo resultó ser una mala aproximación. Posteriormente Dirac añadió el funcional relacionado con el intercambio de electrones, pero tampoco se alcanzó un resultado satisfactorio.

La idea de usar la densidad electrónica para describir la energía del sistema se abandonó debido a la mala representación de la energía cinética en términos de la densidad. En 1962 se demostraron dos teoremas que sustentarán el posterior desarrollo y aplicación de la DFT moderna. El primero de ellos garantiza la existencia de una correspondencia unívoca entre la densidad electrónica y la energía del sistema. El segundo de ellos es un teorema variacional.

2.2.1. Teoremas de Hohenberg y Kohn

- **Teorema 1** *En el estado fundamental del sistema, el potencial externo es determinado, salvo una constante, por la densidad electrónica* $\rho(\vec{r})$

  De este primer teorema concluimos que el conocimiento de la densidad electrónica nos permite determinar de forma unívoca el potencial externo y viceversa. Esto nos indica que la densidad electrónica juega el mismo papel que la función de onda en la ecuación de Schrödinger. Sin embargo este teorema no nos ofrece una forma de determinar la densidad electrónica.

  La expresión de la energía, explicitando la dependencia con el potencial, es la siguiente:

  $$E_v[\rho] = F_{HK}[\rho] + \int \rho(\vec{r})v(\vec{r})d\vec{r} \tag{2.5}$$

  Donde $F_{HK}[\rho]$ es el funcional universal de Hohenberg-Kohn, que es válido para todos los sistemas ya que es independiente del potencial $v(\vec{r})$

- **Teorema 2** *Sea* $\tilde{\rho}(\vec{r})$ *una densidad de un estado fundamental de prueba, tal que* $\tilde{\rho}(\vec{r}) \geq 0$ *y* $\int \tilde{\rho}(\vec{r})d\vec{r} = N$, $E_{exacta} \leq E_v[\tilde{\rho}]$, *donde* $E_v[\rho]$ *es el funcional de la energía de prueba y* E_{exacta} *es la energía exacta obtenida de la resolución de la ecuación de Schrödinger para el estado fundamental*

Esto permite desarrollar métodos variacionales para encontrar computacionalmente la densidad electrónica.

2.2.2. Método de Kohn-Sham

En el momento en el que se enunciaron los teoremas de Hohenberg-Kohn no se conocía la forma de obtener un funcional de la energía cinética con precisión suficiente como para predecir, por ejemplo el enlace químico. En el modelo desarrollado por Thomas y Fermi no se conseguía una buena aproximación de la energía cinética, incluso con las posteriores correcciones realizadas por Dirac. En 1965 Kohn y Sham (KS) idearon una forma de obtener la contribución de la energía cinética. Recurrieron a un sistema ficticio formado por N_e electrones no interactuantes que se mueven en un potencial efectivo, $v_{\text{efectivo}}(\vec{r})$, de forma que tenga la misma densidad y energía que un sistema real de N_e electrones. Al tener un sistema de electrones que no interaccionan, la función de onda que describe dicho sistema puede ser descrita mediante un determinante de Slater y la energía cinética se puede escribir como:

$$T_S = -\sum_{i=1}^{N} o_i \left\langle \phi_i(t) \right| \frac{1}{2}\nabla^2 \left| \phi_i(t) \right\rangle \tag{2.6}$$

donde ϕ_i son los orbitales del sistema de electrones no interactuantes y o_i las ocupaciones de los mismos. El uso de esta expresión arrojará buenos resultados si la energía cinética exacta, $T[\rho]$, es muy parecida a la del sistema no interactuante, T_S.

La energía del sistema real viene dada por la expresión 2.5 y puede ser reescrita en términos de los funcionales del sistema ficticio

$$E_v[\rho] = T_s[\rho] + J[\rho] + E_{xc}[\rho] + \int \rho(\vec{r})v(\vec{r})d\vec{r} \tag{2.7}$$

donde $J[\rho]$ es la energía electrostática clásica debida a la interacción culombiana entre los electrones, $E_{xc}[\rho]$ es el funcional de intercambio y correlación del sistema y representa todas las interacciones de los electrones que no se describan de forma clásica, además de la energía cinética de correlación. Puede expresarse de la siguiente forma

$$E_{xc}[\rho] = T[\rho] - T_s[\rho] + V_{ee}[\rho] - J[\rho] \tag{2.8}$$

Este término agrupa los funcionales que no tienen una expresión sencilla. La energía de intercambio-correlación incluye la diferencia entre la energía cinética entre el sistema real y el ficticio. Esta diferencia se espera que sea pequeña, ya que la energía cinética del sistema ficticio se describe mediante orbitales de HF, método que produce predicciones fiables.

A partir de la energía de intercambio-correlación podemos definir el potencial de intercambio-correlación, $v_{xc}(\vec{r})$, de la siguiente forma

$$v_{xc}(\vec{r}) = \frac{\delta E_{xc}[\rho]}{\delta \rho} \tag{2.9}$$

de forma que el potencial efectivo en el que se mueven los cuasi-electrones será

$$v_{\text{efectivo}}(\vec{r}) = v(\vec{r}) + \frac{\delta J[\rho]}{\delta \rho} + \frac{\delta E_{xc}[\rho]}{\delta \rho} \tag{2.10}$$

El sistema real puede ser descrito por un conjunto de N ecuaciones a un electrón, similares a las ecuaciones de Hartree-Fock pero usando un potencial efectivo.

$$\hat{h_S}\ \phi_i = \left[-\frac{1}{2}\nabla^2 + v_{eff}\right]\phi_i = \varepsilon_i \phi_i \tag{2.11}$$

Las ecuaciones de Kohn-Sham son exactas, ya que no se ha realizado ninguna aproximación, sin embargo al no conocer con exactitud el potencial de intercambio-correlación, la bondad de los cálculos reside en una buena elección de este potencial.

De los funcionales que contribuyen a la energía conocemos completamente $J[\rho]$. La contribución de intercambio y correlación no se conoce de forma exacta, pero su contribución a la energía total es aproximadamente del 1 %. Sin embargo, la energía cinética, $T[\rho]$, es el término más problemático ya que se corresponde, aproximadamente, con la mitad de la energía total y los funcionales encontrados son más difíciles de modelar.

2.3. Teoría del funcional de la densidad de segundos principios

Los métodos de primeros principios, como es la DFT, son eficientes siempre que el número de partículas no sea muy grande. En tal caso el cálculo del hamiltoniano $\hat{h_S}$ en la ecuación 2.11 y su diagonalización resulta enormemente costoso. Un posible solución a este problema es recurrir a métodos semi-empíricos como el modelo de enlace fuerte, que permite reducir

el número de elementos significativos en la matriz $\hat{h_S}$, volviéndola dispersa (sparce en inglés) permitiendo nuevas técnicas de diagonalización. Del mismo modo el movimiento de los átomos alrededor de una posición de equilibrio puede representarse mediante hamiltonianos modelo y el magnetismo asimilarse a un modelo de Ising.

En esta sección introducimos el método denominado de segundos principios, Ref. [5]. Recibe este nombre porque solo requiere datos obtenidos de simulaciones de primeros principios, por lo tanto se puede considerar un método con plena capacidad predictiva. A continuación se describen los puntos mas importantes de la teoría que sustenta al método y que se detalla en Ref. [5].

Asumimos que estamos en condiciones de aplicar la aproximación de Born-Oppenheimer. En primer lugar, se elige una configuración de los núcleos, que servirá de referencia para describir otra configuración del sistema. Además, para cada configuración atómica definimos una densidad electrónica de referencia, $\rho_0(\vec{r})$, para cada una de las posibles configuraciones atómicas. La densidad electrónica, $\rho_0(\vec{r})$, además se puede describir en términos de la configuración de referencia de la siguiente forma

$$\rho(\vec{r}) \;=\; \rho_0(\vec{r}) \;+\; \delta\rho(\vec{r}) \tag{2.12}$$

En la expresión anterior, $\delta\rho(\vec{r})$ es la deformación de la densidad, la cual describe los cambios en las propiedades del sistema con respecto a la configuración de referencia. Esta forma de describir la densidad del sistema permite reescribir la energía DFT, de manera aproximada, como la suma de tres términos mediante la expansión de la energía DFT (Ec.2.7) en potencias de $\delta\rho$

$$E_{DFT} \;\approx\; E \;=\; E^{(0)} \;+\; E^{(1)} \;+\; E^{(2)} \tag{2.13}$$

El término de orden cero, $E^{(0)}$, corresponde a la energía exacta DFT para la densidad de referencia, $\rho_0(\vec{r})$. El término de primer orden, $E^{(1)}$, recoge la diferencia de la energía a un electrón entre las configuraciones excitadas y la de referencia y por último, el término de segundo orden contiene la contribución a dos electrones debida al intercambio y la correlación y las interacciones clásicas. La expresión de los términos descritos es la siguiente

$$E^{(0)} \;=\; \sum_{j\vec{k}} o^{(0)}_{j\vec{k}} \left\langle \psi^{(0)}_{j\vec{k}} \right| \hat{t} + v_{\text{ext}} \left| \psi^{(0)}_{j\vec{k}} \right\rangle + \frac{1}{2} \iint \frac{\rho_0(\vec{r})\,\rho_0\left(\vec{r'}\right)}{|\vec{r} \;-\; \vec{r'}|} d^3r d^3r' \;+\; E_{xc}[\rho_0] + E_{nn} \tag{2.14}$$

$$E^{(1)} \;=\; \sum_{j\vec{k}} \left[o_{j\vec{k}} \left\langle \psi_{j\vec{k}} \right| \hat{h_0} \left| \psi_{j\vec{k}} \right\rangle - o^{(0)}_{j\vec{k}} \left\langle \psi^{(0)}_{j\vec{k}} \right| \hat{h_0} \left| \psi^{(0)}_{j\vec{k}} \right\rangle \right] \tag{2.15}$$

$$E^{(2)} \;=\; \frac{1}{2} \int d^3r \int d^3r' g\left(\vec{r}, \vec{r'}\right) \delta\rho(\vec{r})\, \delta\rho\left(\vec{r'}\right) \tag{2.16}$$

donde $o_{j\vec{k}}$ es la ocupación del estado caracterizado por el vector de onda $\vec{k}$ y el índice de banda j, mientras que el operador, $\hat{h_0}$, es el hamiltoniano de Kohn-Sham para un electrón definido para la configuración de referencia [5], $\hat{h_0} \;=\; \hat{h}_{\text{eff}}[\rho_0]$. En el término de segundo orden, $g\left(\vec{r}, \vec{r'}\right)$ es el operador de la interacción electrón-electrón, su expresión puede verse en la Ref. [5]

En la siguiente sección (Sec.2.4) se detallarán las características de la base elegida y cómo son las expresiones de los términos de la energía en la base de funciones de Wannier. Además, se detallarán las ecuaciones autoconsistentes a resolver, análogas a las ecuaciones de Kohn-Sham, y se escribirá el hamiltoniano efectivo, equivalente al hamiltoniano de Hartree-Fock. [5]

2.4. Funciones de Wannier

La formulación usada en el tratamiento de nuestro problema requiere del cálculo de elementos de matriz del hamiltoniano de Kohn-Sham para un electrón. Para dicho cálculo hace falta una base sobre la que trabajar, en este caso se ha optado por una base formada por funciones de Wannier [12]. Estas funciones se construyen mediante una transformación unitaria de los orbitales moleculares y, por lo tanto, son muy eficientes a la hora de describir el sistema ya que contienen toda la información del mismo. Además, tienen otras ventajas que ayudan a la simulación:

- Las funciones de Wannier se pueden elegir de forma que sean ortogonales entre sí, evitando el cálculo de integrales de solapamiento y, por lo tanto, reduciendo el coste computacional.
- Las funciones de Wannier se pueden escoger de forma que, espacialmente, estén muy localizadas sobre los núcleos, decayendo rápidamente si nos alejamos de los mismos. Esto convierte la matriz que representa el hamiltoniano en dispersa y permite la aplicación de métodos eficientes de diagonalización como Lanczos [6]

La expresión matemática de una función de Wannier es la siguiente, ver Ref. [12]

$$|\chi_a\rangle \;=\; \frac{V}{(2\pi)^3}\int_{BZ} d\vec{k}\; e^{-i\vec{k}\vec{R}_a}\;\sum_{m=1}^{J}\; T_{ma}^{(\vec{k})}\left|\psi_{m\vec{k}}^{(0)}\right\rangle \tag{2.17}$$

donde V es el volumen de la celda primitiva, la integral se extiende sobre toda la zona de Brillouin, el índice m recorre las J bandas, las matrices $T_{ma}^{(\vec{k})}$ representan las transformaciones unitarias entre las funciones de Bloch para un vector de onda, $\vec{k}$, dado y por último, $\left|\psi_{m\vec{k}}^{(0)}\right\rangle$, representan el conjunto inicial de estados propios del hamiltoniano que definen las bandas.

Veamos como las funciones de Wannier pueden usarse para describir los estados no estacionarios. Supongamos que los posibles estados que describen el sistema varían con el tiempo y se representan, en notación de dirac, $|\phi_i(t)\rangle$. De forma que se puede reescribir la expresión 2.4 para la densidad de la siguiente forma

$$\hat{\rho}(t) = \sum_{i=1}^{N} o_i\,|\phi_i(\vec{r},t)\rangle\,\langle\phi_i(\vec{r},t)| \tag{2.18}$$

donde o_i representa la ocupación de los estados ϕ_i. Suponemos que el estado sistema en un instante de tiempo t es:

$$\phi_i(\vec{r},t) = \sum_{\alpha=1}^{M} c_{i\alpha}(\vec{r},t)\;\chi_\alpha(\vec{r},t) \tag{2.19}$$

donde $\chi_\alpha(\vec{r},t)$ es un elemento de la base $\{\chi_i\}_{i=1}^{M}$ usada para describir los estados del sistema, dicha base será un conjunto de funciones de Wannier. Sustituyendo Eq. (2.19) en la expresión de la densidad, Eq.(2.18), se obtiene

$$\rho(\vec{r},t) = \sum_{\alpha=1}^{M}\sum_{\beta=1}^{M}\left(\sum_{i=1}^{N} o_i c_{i\alpha}^{*} c_{i\beta}\right)\chi_\alpha^{*}(\vec{r},t)\chi_\beta(\vec{r},t) = \sum_{\alpha=1}^{M}\sum_{\beta=1}^{M} d_{\alpha\beta}\;\chi_\alpha^{*}(\vec{r},t)\chi_\beta(\vec{r},t) \tag{2.20}$$

donde $d_{\alpha\beta}$ representan los elementos de matriz de la matriz densidad en la base $\{\chi_i\}_{i=1}^{M}$

Se destacan dos propiedades de la matriz densidad:

- El hecho de expresar la matriz densidad en una base de funciones ortogonales, como pueden ser las funciones de Wanniers, permite escribir el número de electrones del sistema de la siguiente forma.

$$N \;=\; \int \rho(\vec{r})\; d^3\vec{r} \;=\; \sum_{\alpha=1}^{M}\sum_{\beta=1}^{M} d_{\alpha\beta} \int \chi^*_\alpha \chi_\beta \; d^3\vec{r} \;=\; \sum_{\alpha=1}^{M}\sum_{\beta=1}^{M} d_{\alpha\beta}\delta_{\alpha\beta} \;=\; \sum_{\alpha} d_{\alpha\alpha} \;=\; Tr(d) \tag{2.21}$$

donde $Tr(d)$ es la traza de la matriz densidad, esto es la suma de la diagonal de la misma.

- La matriz densidad es un observable, por lo tanto se trata de una matriz hermítica

$$d = d^\dagger \tag{2.22}$$

Como ya se menciono en la sección anterior, una vez conocida la base en la que trabajamos, estamos en condiciones de escribir las expresiones de las términos de energía, 2.15 y 2.16, en una base de funciones de Wannier.

$$E^{(1)} \;=\; \left[\sum_{\alpha\beta} d_{\alpha\beta} \left\langle \chi_\alpha \right| \hat{h}_0 \left| \chi_\beta \right\rangle \;-\; \sum_{\alpha\beta} d^{(0)}_{\alpha\beta} \left\langle \chi_\alpha \right| \hat{h}_0 \left| \chi_\beta \right\rangle\right] \;=\; \sum_{\alpha\beta} D_{\alpha\beta}\gamma_{\alpha\beta} \tag{2.23}$$

$$E^{(2)} \;=\; \frac{1}{2}\sum_{\alpha\beta}\sum_{\alpha'\beta'} D_{\alpha\beta}D_{\alpha'\beta'}U_{\alpha\beta\alpha'\beta'} \tag{2.24}$$

Donde $d^{(0)}_{\alpha\beta}$ es la matriz de densidad en el estado de referencia, $\hat{h}_0$ es el hamiltoniano efectivo en la configuración de referencia y $D_{\alpha\beta}$ es la matriz de deformación. A continuación escribiremos las ecuaciones autoconsistentes del método de segundos principios, análogas a las de Kohn-Sham

$$\sum_{b} \; h^s_{ab,\vec{k}} c^s_{jb\vec{k}} \;=\; \epsilon^s_{j\vec{k}} c^s_{ja\vec{k}} \tag{2.25}$$

donde $\epsilon^s_{j\vec{k}}$ es la energía de la j-ésima banda con vector de onda $\vec{k}$ para el canal s de spin y el elemento de matriz del hamiltoniano, $h^s_{ab,\vec{k}}$, es [5]

$$h^s_{ab,\vec{k}} \;=\; \sum_{\vec{R}_B \;-\; \vec{R}_A} e^{i\vec{k}\left(\vec{R}_B \;-\; \vec{R}_A\right)} h^s_{ab} \tag{2.26}$$

$$h^s_{ab} \;=\; \gamma_{ab} + \sum_{a'b'} \left[\left(D^s_{a'b'} \;+\; D^{-s}_{a'b'}\right) U_{aba'b'} \;+\; \left(D^{-s}_{a'b'} \;-\; D^s_{a'b'}\right) I_{aba'b'}\right] \tag{2.27}$$

Conociendo un valor de la matriz de deformación, $D_{\alpha\beta}$, podemos calcular el hamiltoniano efectivo y de la diagonalización del hamiltoniano podemos obtener una nueva matriz de deformación, continuando con el proceso hasta llegar a la autoconsistencia [5]. Los parámetros γ y U recogen gran parte de las contribuciones electrostáticas, mientras que el parámetro I recoge las contribuciones no clásicas.

2.5. Propagación de la matriz densidad

En el apartado anterior se ha visto la importancia de la densidad en el marco de la DFT y cómo ρ está unívocamente determinada por la matriz de densidad. Sin embargo, no se ha

explicado como evoluciona esta magnitud con el tiempo. Para hacerlo partimos del esquema de Heisenberg de la mecánica cuántica

$$i\hbar\frac{d}{dt}\rho(t) = [h(t), \rho(t)] \tag{2.28}$$

donde $\mathcal{H}$ es el hamiltoniano del sistema y $[h(t), \rho(t)]$ es el conmutador del hamiltoniano con la matriz densidad. La importancia de esta expresión es vital, ya que a partir del hamiltoniano efectivo, $\hat{h}_{\text{eff}}$, que depende de la densidad, se puede realizar la propagación de la densidad, obteniendo un juego cerrado de ecuaciones que nos permiten conocer propiedades del sistema fuera del equilibrio.

2.6. Fuerzas

Para el estudio de la dinámica es necesario conocer las fuerzas a las que se ve sometidas el sistema para cambiar la geometría de forma adecuada. El teorema de Hellmann-Feynman [2], permite encontrar una expresión dentro del esquema de segundos principios. Comencemos por la expresión matemática del teorema

$$\frac{dE\left(Q\right)}{dQ} \;=\; \left\langle \frac{\partial\mathcal{H}\left(Q\right)}{\partial Q} \right\rangle \tag{2.29}$$

Esta teorema supone que la energía, $E\left(Q\right)$, depende de un parámetro, que en nuestro caso será la distancia internuclear. Por otro lado, supone que el hamiltoniano, $\mathcal{H}\left(Q\right)$, tiene una dependencia con el mismo parámetro. El valor medio de la Ec.2.29 se puede reescribir, teniendo en cuenta que los estados del sistema puede expresarse en una base de Wannier (Ec. 2.19)

$$\begin{aligned}
\left\langle \frac{\partial\hat{\mathcal{H}}\left(Q\right)}{\partial Q} \right\rangle \;&=\; \sum_{n} o_n \left\langle \phi_n \right| \frac{\partial\hat{\mathcal{H}}\left(Q\right)}{\partial Q} \left| \phi_n \right\rangle \\
&= \sum_{\alpha}\sum_{\beta}\left(\sum_{n} o_n c^{*}_{n\alpha} c^{*}_{n\beta}\right) \left\langle \chi_\alpha \right| \frac{\partial\hat{\mathcal{H}}\left(Q\right)}{\partial Q} \left| \chi_\beta \right\rangle \\
&= \sum_{\alpha}\sum_{\beta} d_{\alpha\beta} \left\langle \chi_\alpha \right| \frac{\partial\hat{\mathcal{H}}\left(Q\right)}{\partial Q} \left| \chi_\beta \right\rangle
\end{aligned} \tag{2.30}$$

Usando esta expresión y la relación $\vec{F} \;=\; -\dfrac{dE}{dQ}$, estamos en condiciones de reescribir la expresión 2.29 como sigue

$$\vec{F} \;=\; -\sum_{\alpha}\sum_{\beta} d_{\alpha\beta} \left\langle \chi_\alpha \right| \frac{\partial\hat{\mathcal{H}}\left(Q\right)}{\partial Q} \left| \chi_\beta \right\rangle \tag{2.31}$$

Uno de los pilares del esquema de segundos principios es describir la densidad electrónica de un estado del sistema, $\rho\left(\vec{r}\right)$, en términos de la densidad de una configuración de referencia, $\rho_0\left(\vec{r}\right)$ del mismo sistema

$$\rho\left(\vec{r}\right) \;=\; \rho_0\left(\vec{r}\right) \;+\; \delta\rho\left(\vec{r}\right) \tag{2.32}$$

Esta expresión tiene su expresión equivalente, si consideramos el operador matricial

$$d_{\alpha\beta} \;=\; d^{0}_{\alpha\beta} \;+\; D_{\alpha\beta} \tag{2.33}$$

por lo tanto, se puede describir la fuerza, (Ec.2.31), en términos de la matriz densidad en la configuración de referencia, $d^0_{\alpha\beta}$, más un término de corrección asociado al cambio en la densidad y que se recoge en la matriz de deformación, $D_{\alpha\beta}$

$$\begin{aligned}\vec{F} &= -\sum_{\alpha}\sum_{\beta} d^0_{\alpha\beta} \langle\chi_\alpha| \frac{\partial\hat{\mathcal{H}}(Q)}{\partial Q} |\chi_\beta\rangle - \sum_{\alpha}\sum_{\beta} D_{\alpha\beta} \langle\chi_\alpha| \frac{d\mathcal{H}(Q)}{dQ} |\chi_\beta\rangle \\ &= \vec{F}_0 - \sum_{\alpha}\sum_{\beta} D_{\alpha\beta} \langle\chi_\alpha| \frac{d\mathcal{H}(Q)}{dQ} |\chi_\beta\rangle\end{aligned} \tag{2.34}$$

En la expresión anterior, $\vec{F}_0$, representa la fuerza en el sistema que posee la densidad de referencia. Esta expresión permite conocer la fuerza a la que se ve sometido el sistema en cualquier configuración que no sea la de referencia, con tan solo conocer el valor de la matriz de deformación, es decir, como es el cambio de la densidad entre ambas configuraciones.

Esta es la idea principal del trabajo. Con el conocimiento de $\vec{F}_0$ para el estado fundamental y el hamiltoniano electrónico llevar a cabo simulaciones de la molécula de H_2 en el estado excitado y poder estudiar la disociación.

Capítulo 3

Simulaciones de H_2 a primeros principios

3.1. Molécula de H_2

La molécula de hidrógeno es el sistema neutro mas sencillo, está formado por dos protones y dos electrones. Para describir este sistema de una manera sencilla utilizamos el método de Hartree-Fock donde se pueden definir niveles a un electrón. En este caso la función de onda se escribe como un determinante de Slater [2, 7]

$$\Psi_{HF}\left(1,2,\ldots,N_e\right) \;=\; \frac{1}{\sqrt{N_e\,!}}\begin{vmatrix}\phi_1(1) & \phi_2(1)\\ \phi_1(2) & \phi_2(2)\end{vmatrix} \;=\; |\phi_1,\ \phi_2| \tag{3.1}$$

En la expresión anterior, $\phi_i\left(j\right)$ es el i-ésimo orbital molecular ocupado por el j-ésimo electrón.

Estos orbitales moleculares pueden describirse como combinaciones lineales de orbitales atómicos, facilitando la resolución de las ecuaciones de Hartree-Fock [2, 7]. Para la molécula de hidrógeno, el esquema de los orbitales moleculares formado por la combinación lineal de orbitales atómicos $1s$ es el que se muestra en la Fig.(3.1), el nivel designado con la etiqueta $1\sigma_g$ es el nivel enlazante, mientras que el nivel $1\sigma_u$ es el nivel antienlazante. Como se puede observar en la Fig.(3.1) la separación de los niveles enlazante y antienlazante depende de la distancia internuclear, cuanto mayor sea el acercamiento mayor sera la diferencia de energía entre los orbitales moleculares. El punto de equilibrio de la molécula se encuentra de forma experimental, cuando la distancia internuclear es de $R \;=\; 0{,}74\ \mathring{A}$ en ese punto la energía de disociación de la molécula es de $E \;=\; 4{,}52\ eV$.

La forma en la que se ocupen estos orbitales moleculares dan lugar a distintos estados de la molécula, estos se representan en la Fig.(3.4) y expanden tanto el estado fundamental como los estados excitados de la molécula. Supongamos que la molécula se encuentra en el estado fundamental, ${}^1\Sigma_g$, esto es, con los dos electrones ocupando el orbital molecular $1\sigma_g$. En dicho estado, los electrones se acomodan de forma que la probabilidad de encontrarlos en la zona internuclear aumente Fig.(3.2), permitiendo al sistema mantenerse en un estado de equilibrio. Además, en esta configuración la molécula de hidrógeno se encuentra en un estado singlete de spin. Se puede comprobar aplicando los operadores $\hat{S}^2$ y $\vec{\hat{S}}$ a la función Hartree-Fock de este estado.

Ahora, si colocamos los dos electrones en el nivel antienlazante estaremos en un estado ${}^1\Sigma_g$, que es también de tipo singlete de spin. La ocupación de este orbital molecular implica que las densidades electrónicas se localicen sobre los núcleos, dejando la zona internuclear libre de carga, Fig.(3.3), por lo tanto este estado conduce a una situación inestable que desemboca en la disociación de la molécula.

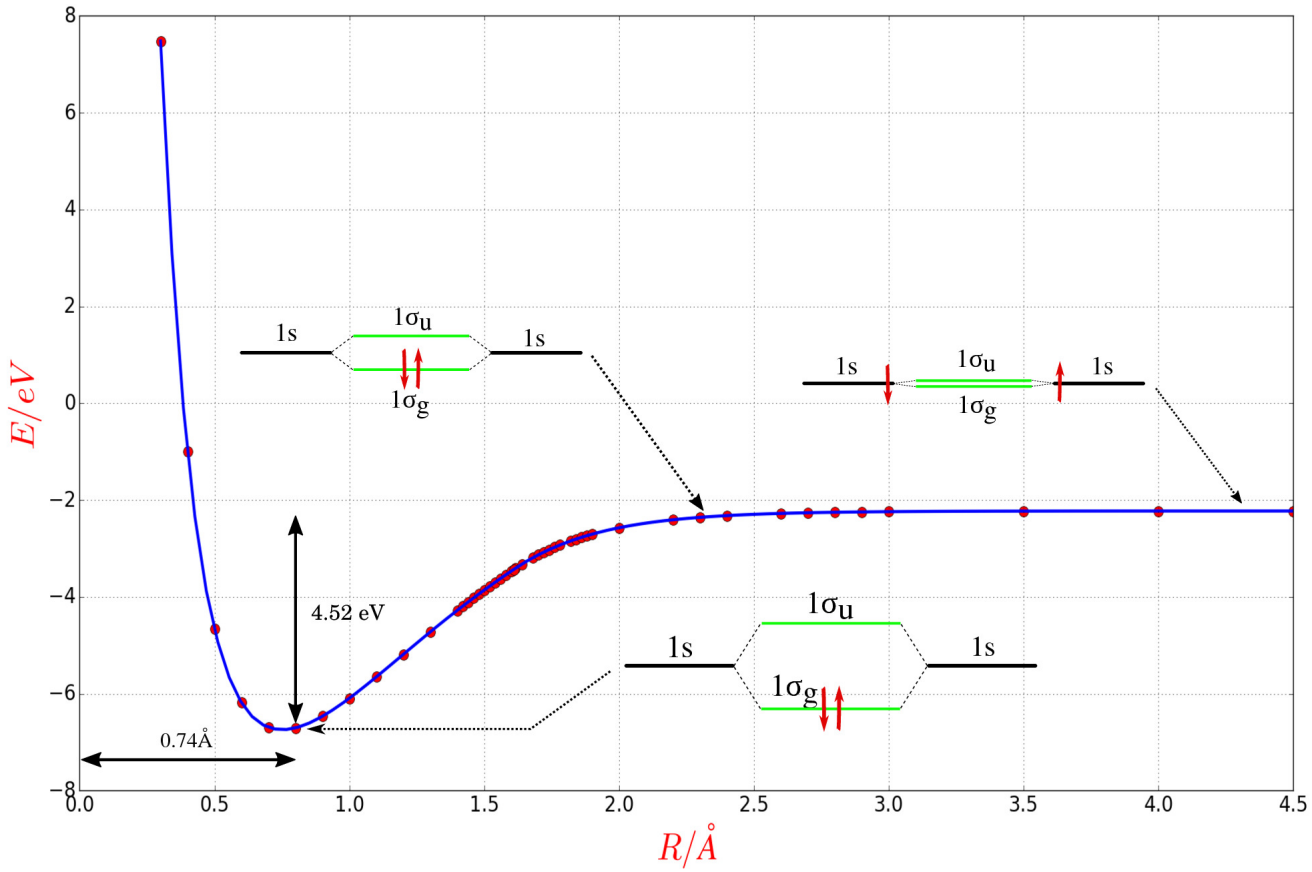

Figura 3.1: Se representa la curva de energía en función de la distancia internuclear. Como se puede observar en los esquemas de niveles, cuando la distancia internuclear es grande los orbitales moleculares tienen carácter pseudoatómico, de forma que los electrones se alojan en cada uno de ellos. Sin embargo, cuando la distancia va reduciéndose los orbitales pierden su carácter atómico y son orbitales moleculares completos, alojándose los electrones en el nivel enlazante, cuya energía esta por debajo de los orbitales atómicos originales.

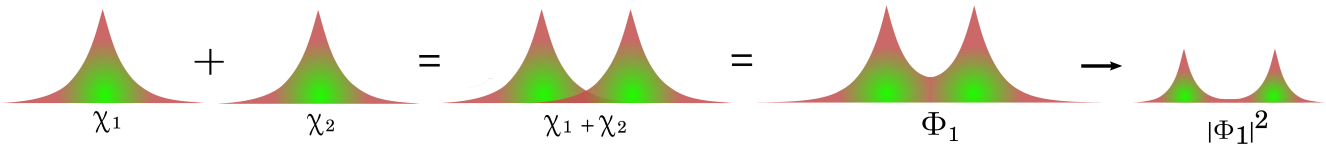

Figura 3.2: Función de onda del orbital molecular enlazante, formada por la combinación de orbitales $1s$.

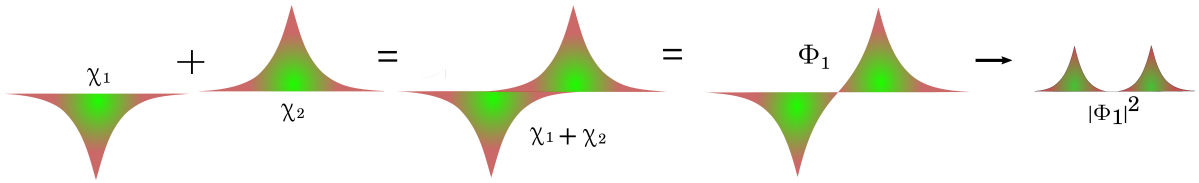

Figura 3.3: Descripción de los orbitales moleculares de la molécula de hidrógeno, construidos a partir de orbitales atómicos $1s$

Si ahora suponemos que tenemos un electrón en cada uno de los niveles tendremos dos posibles estados: $^1\Sigma_u$ y $^3\Sigma_u$, que expanden casos intermedios. El primero de ellos se corresponde

a un estado singlete de spin y como se puede ver en Fig.(3.4) cuando se realizan detallados cálculos de primeros principios posee un pequeño mínimo de forma que estamos ante un estado estable. El segundo estado se corresponde con un triplete de spin, es inestable y acaba con la disociación de la molécula.

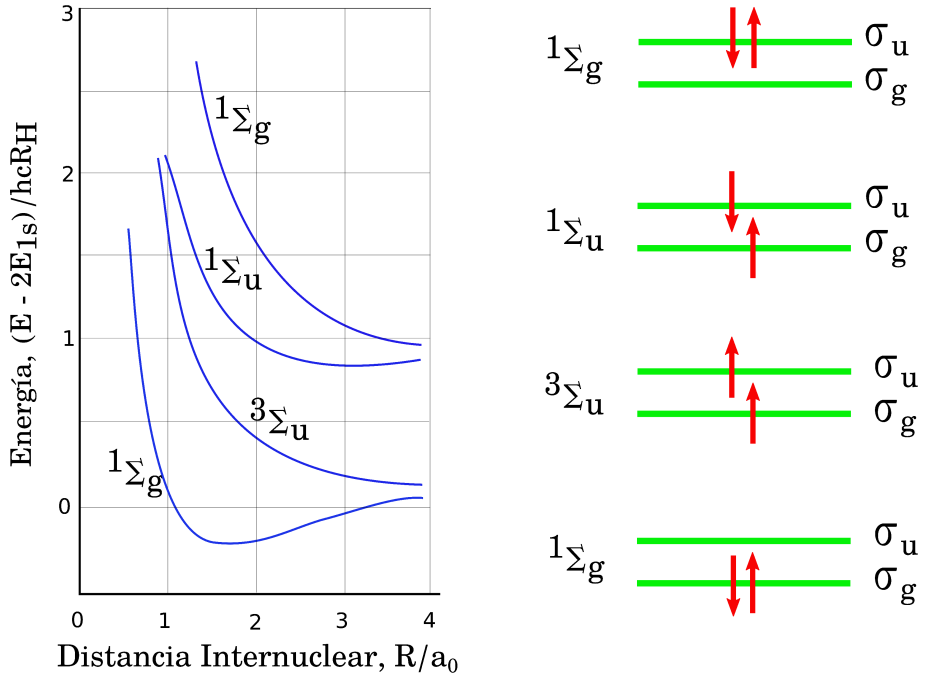

Figura 3.4: Descripción de los diferentes estados de la molécula de hidrógeno para un esquema de dos orbitales moleculares. A la derecha se puede observar como es la configuración de los electrones en los orbitales moleculares para cada estado excitado de la molécula. [2]

3.2. Cálculos de primeros principios

Los cálculos de primeros principios se han realizado con el código VASP [9–11] usando una aproximación GGA para el funcional de intercambio y correlación [13]. Las dos configuraciones resueltas son: RHF y UHF, por sus siglas en inglés. La configuración RHF, Fig.(3.5 b), aloja los dos electrones en el mismo orbital espacial de forma que incluso, cuando los átomos estén separados, los electrones con spin arriba y abajo tienen la misma parte espacial. Sin embargo, la configuración UHF, Fig.(3.5 a), considera orbitales espaciales distintos para cada componente de spin.

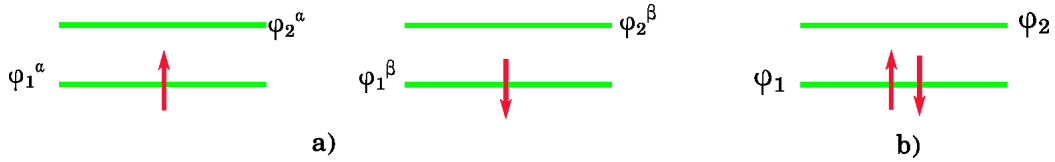

Figura 3.5: Esquema de la configuración UHF, a la izquierda, donde cada canal de spin, arriba y abajo denotados, respectivamente, con la letra α y β , tiene orbitales espaciales distintos, mientras que en la configuración RHF, a la derecha, para cada spin el orbital espacial es el mismo.

Como se ha descrito en la sección anterior, la energía de disociación experimental de la molécula es de $E = 4.52\ eV$ [2]. La configuración RHF ($E = 6.51\ eV$), Fig.(3.6), no reproduce

de forma adecuada los datos experimentales, mientras que la configuración UHF ($E = 4.53\ eV$) reproduce los datos experimentales de forma satisfactoria.

En la configuración RHF los electrones están obligados a compartir el mismo orbital espacial [7], de forma que, incluso a grandes distancias, los electrones se encuentran apareados. Esto significa que cuando se observa las contribuciones intraatómicas los electrones no pueden aprovechar de forma efectiva el intercambio y observan una repulsión electrón-electrón mas alta que cuando se permite el desapareamiento de los espines. Esta contribución extra puede observarse al comparar los elementos diagonales del hamiltoniano en RHF y UHF. Las ocupaciones de estos orbitales vienen representados por los elementos de la diagonal de la matriz densidad Fig.(3.14). Por otro lado, los elementos fuera de la diagonal, que representan la contribución al enlace debido a la interferencias de las funciones pseudoatómicas, es decir, representan la contribución covalente del enlace. Al considerar orbitales UHF proporcionamos a los electrones un grado de libertad adicional, a que electrones se alojan en orbitales espaciales distintos atendiendo a su spin, de esta forma los electrones se evitan bajando su repulsión mutua intraatómica, como puede verse en la Fig.(3.16) [7].

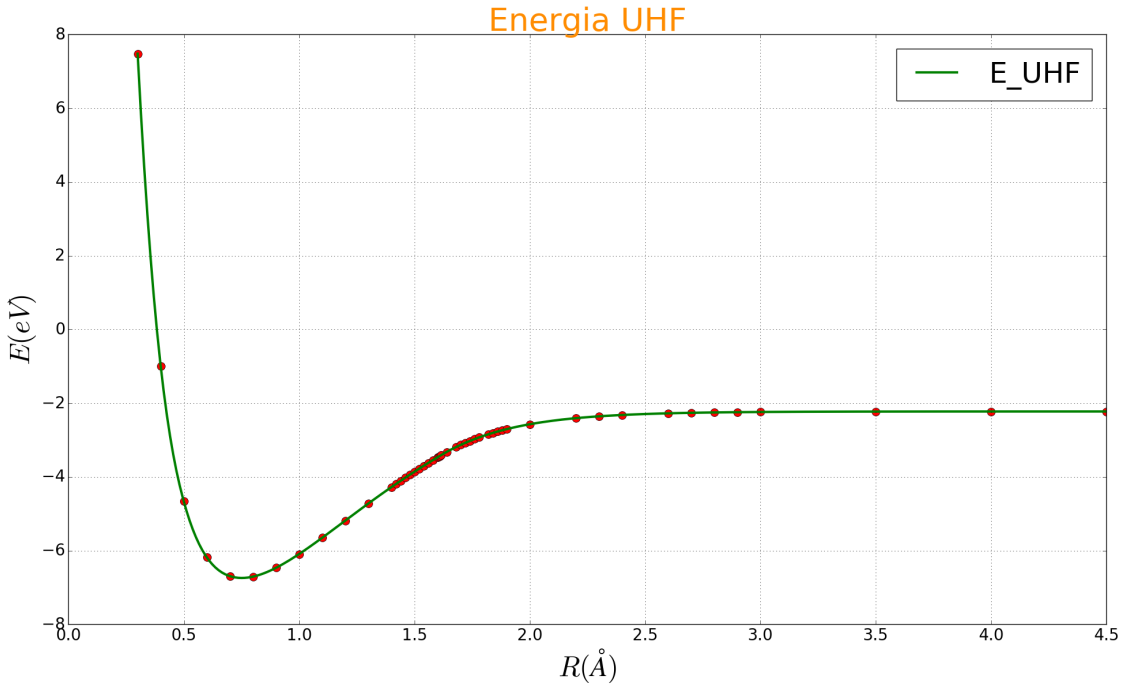

Figura 3.6: Curva de energía de la molécula de hidrógeno en función de la distancia internuclear para la configuración UHF

En las Fig.(3.8) y Fig.(3.9) se representan la energía de los orbitales moleculares de las configuraciones RHF y UHF respectivamente.

En un cálculo inicial, solo se consideró cuatro orbitales moleculares. Sin embargo, se observó la necesidad de incluir un mayor número de niveles, ya que en cuando se intentó obtener el modelo usual a 2 orbitales no se puedo encontrar el comportamiento esperado. Una vez repetido el cálculo, en este caso considerando 40 orbitales moleculares, se observa un gran conjunto de orbitales, a energías $\epsilon > 0$, cuya energía es independiente de la distancia internuclear, (Fig.(3.8) , Fig.(3.9)), estos orbitales representan estados libre de electrones en la molécula. También puede observarse dos orbitales moleculares, cuya energía depende de la distancia entre los núcleos denominados orbital enlazante (curva roja) y antienlazante (curva verde).

A grandes distancias, en el caso RHF, (Fig.3.8), estos dos orbitales acaban degenerados, mientras que, en el caso UHF, (Fig.(3.9), estos orbitales se diferencian claramente. A cortas

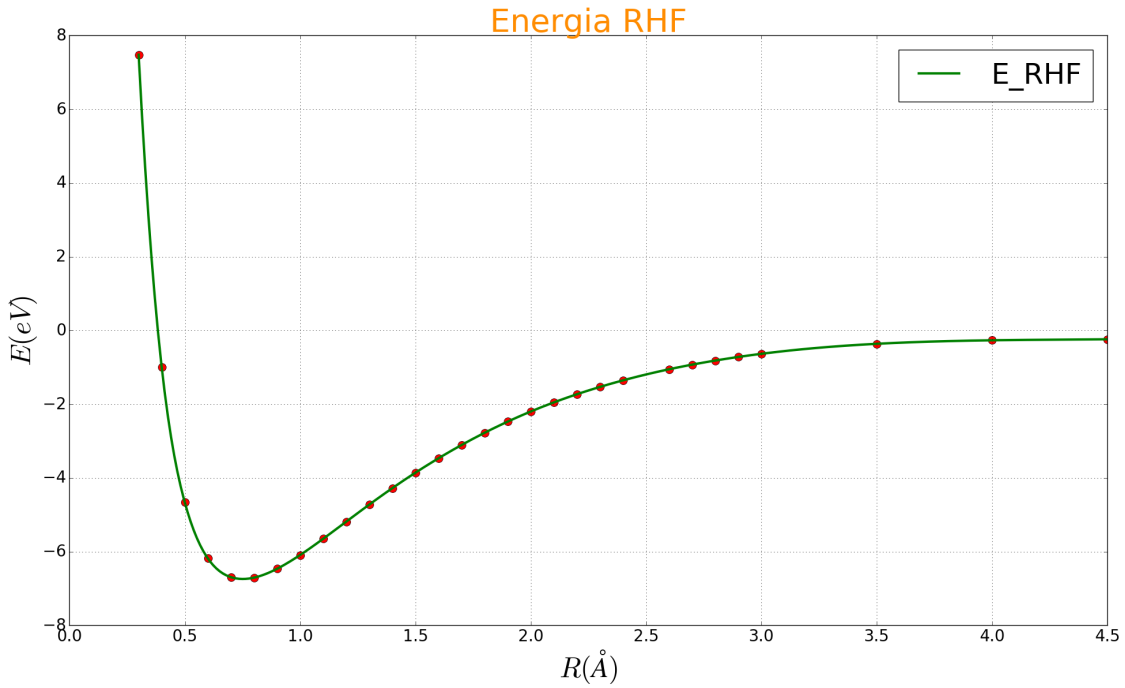

Figura 3.7: Curva de energía de la molécula de hidrógeno en función de la distancia internuclear para la configuración RHF

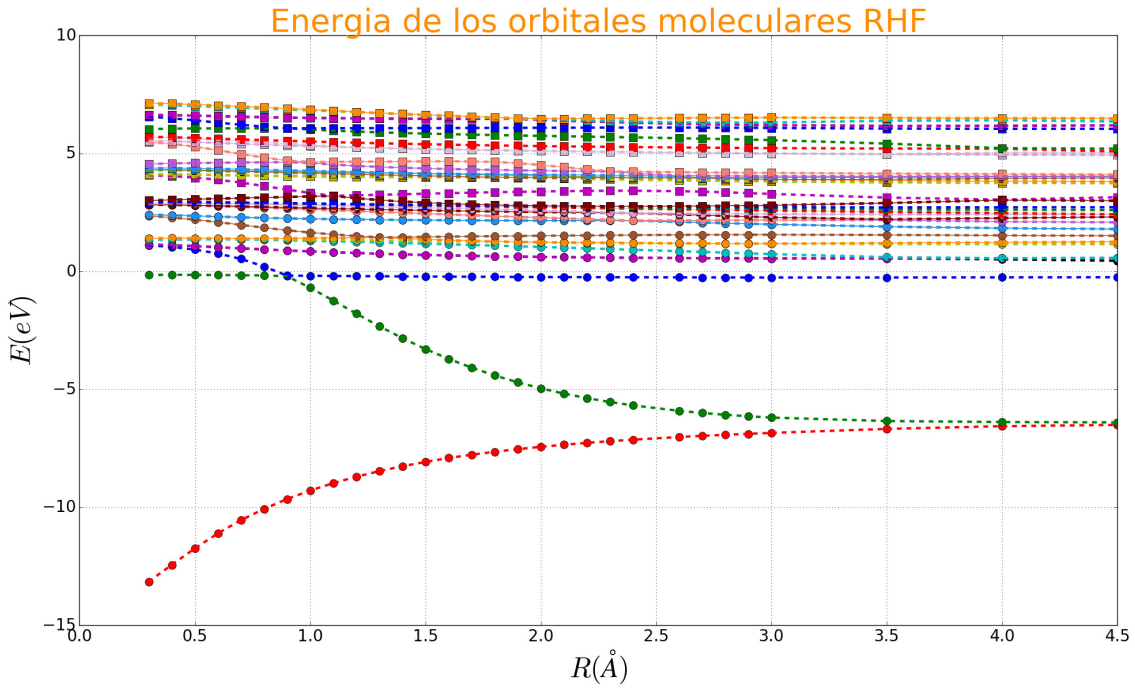

Figura 3.8: Representación de la variación de la energía de los orbitales moleculares en función de la distancia internuclear.

distancias el comportamiento del orbital enlazante y antienlazante en RHF es igual al de sus homólogos en la configuración UHF. En ambas configuraciones se puede observar que el orbital antienlazante sufre múltiples cruces con los orbitales que conforman el continuo. Por este motivo las energías de estos orbitales crecen levemente, porque el carácter del orbital antienlazante se

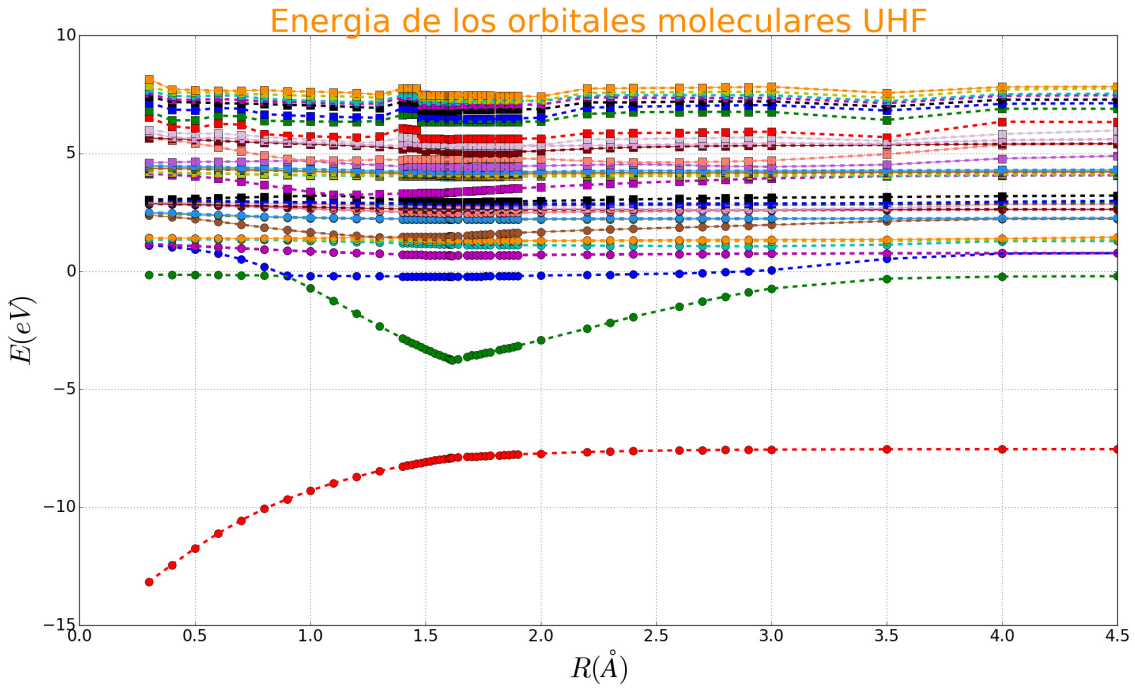

Figura 3.9: Representación de la variación de la energía de los orbitales moleculares en función de la distancia internuclear para la configuración UHF.

mezcla con el de los orbitales del continuo.

3.3. Transformación a una base de funciones de Wannier

Para comprender el comportamiento de las bandas se procederá a realizar una transformación a orbitales localizados de Wannier que permiten recuperar la imagen de la combinación lineal de orbitales atómicos. En la Sección 2.4 se describió como las funciones de Wannier son útiles como base para describir nuestro problema, en las Fig.(3.10), Fig.(3.11), Fig.(3.12) se representan, respectivamente, el hamiltoniano para la configuración RHF y UHF, para ambos canales de spin, en una base de funciones de Wannier.

Los elementos del hamiltoniano se han obtenido de primeros principios. El cálculo se ha realizado para un intervalo de distancias de $R = 0.3$ $\mathring{A}$ hasta $R = 4.5$ $\mathring{A}$. Sin embargo, se ha prescindido de los puntos por debajo de la distancia de equilibrio, $R_{eq} = 0.74$ $\mathring{A}$, ya que el cálculo no es fiable debido a la multitud de cruces de los orbitales del continuo con el orbital antienlazante. Además, a esas distancias la energía de la molécula es muy superior a la del mínimo y eso hace que la molécula no visite estas configuraciones (Fig.(3.7)).

En la configuración RHF, Fig.(3.10), se puede observar como los elementos de la diagonal, que representan las energías de las funciones localizadas, crecen a distancias cortas, cuando las interacciones electrostáticas entre las funciones localizadas son mayores. Por otro lado, los elementos fuera de la diagonal, que representan el acoplamiento de las funciones pseudoatómicas y que controlan su mezcla en el orbital molecular, y por tanto representan el enlace, van disminuyendo a medida que la distancia internuclear crece. Sin embargo, a cortas distancias el acople entre las funciones pseudoatómicas es importante, ya que el enlace es estable.

En la configuración UHF, Fig.(3.12) y Fig.(3.11), se puede observar como el acoplamiento entre las funciones localizadas es similar al de la configuración RHF hasta $R \simeq 1.5$ $\mathring{A}$. En este intervalo existe un fuerte acoplamiento, mientras que a grandes distancias, el acoplamiento

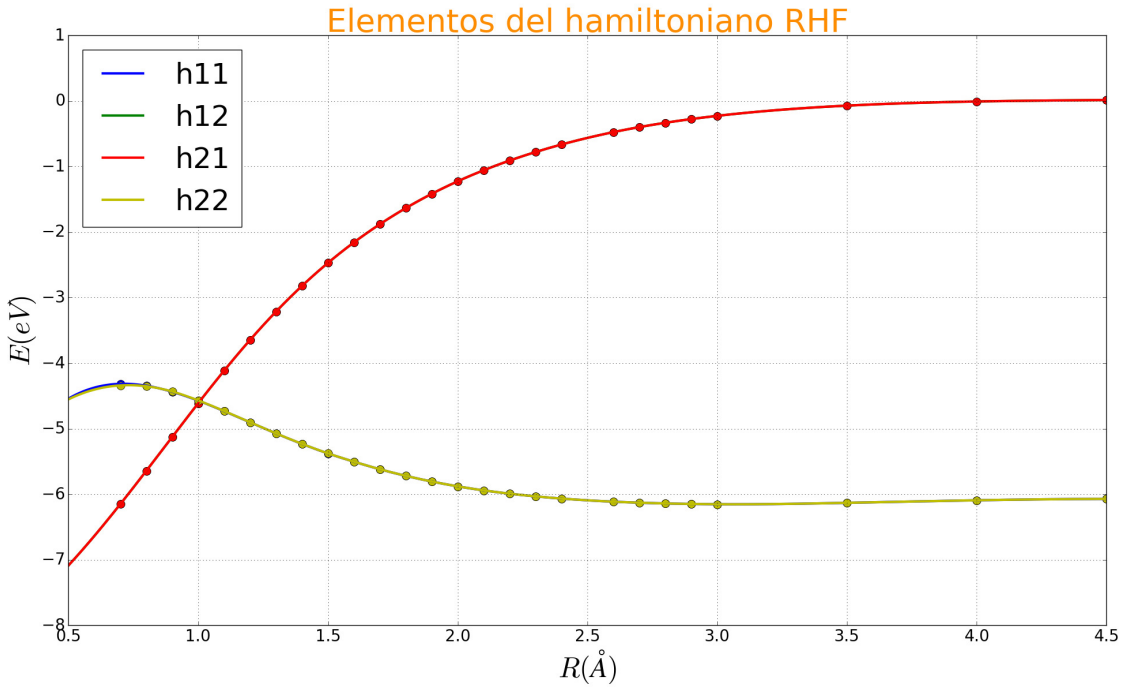

Figura 3.10: Representación de los valores de los elementos del hamiltoniano para la configuración RHF en función de la distancia internuclear. Los marcas representan los valores obtenidos de los cálculos de primeros principios, a partir de estos se ha procedido a realizar una interpolación mediante spline cúbicos, que nos permiten ver la tendencia de los elementos del hamiltoniano a cortas distancias.

(h_{12},h_{21}) se anula, dejando de existir el enlace. A distancias intermedias y largas podemos observar una transición de un comportamiento diamagnético a uno antiferromagnético. Esto se puede observar en UHF debido a que permitimos el desacoplamiento de los 2 canales de spin. Además desde una imagen física lo que se observa es una competición entre el magnetismo intra-atómico que favorece la aparición de un momento magnético localizado en cada uno de los átomos de hidrógeno y la fuerza del enlace favorece el apareamiento de los espines en el orbital enlazante.

De la diagonalización del hamiltoniano se puede hallar la matriz densidad para cada una de la configuraciones, de acuerdo con la Ec.(2.20). Para la configuración de referencia, Fig.(3.14), se observa que los elementos de la matriz densidad son constantes con la distancia internuclear. Los elementos de la diagonal representan la ocupación de cada una de las Wannier, mientras que los elementos fuera de la diagonal representan la interferencia de las Wannier, que refuerzan el enlace. Por otro lado, en las Fig.(3.12) y Fig.(3.11) tenemos las matrices densidad para cada canal de spin en la configuración UHF. En esta configuración la matriz densidad se mantiene constante cuando la distancia internuclear es pequeña, de forma que tenemos las Wannier parcialmente ocupadas con cargas de spin opuesto. Cuando la distancia internuclear crece, las Wannier comienzan a poblarse únicamente con carga del mismo spin, de forma que a grandes distancias las funciones localizadas albergan carga de un único spin, es decir alojan un solo electrón. En ambos canales, el término de interferencia responsable de reforzar el enlace, va disminuyendo conforme aumenta la distancia entre los núcleos, hasta llegar a la disociación neutra.

Una vez determinada la energía y el hamiltoniano para ambas configuraciones del sistema, se puede comprobar la validez de la expresión de la fuerza, Ec.(2.34). Como se puede ver en la Fig.(3.17) a cortas distancias la fuerza en ambos esquemas (RHF y UHF) es la misma, ya

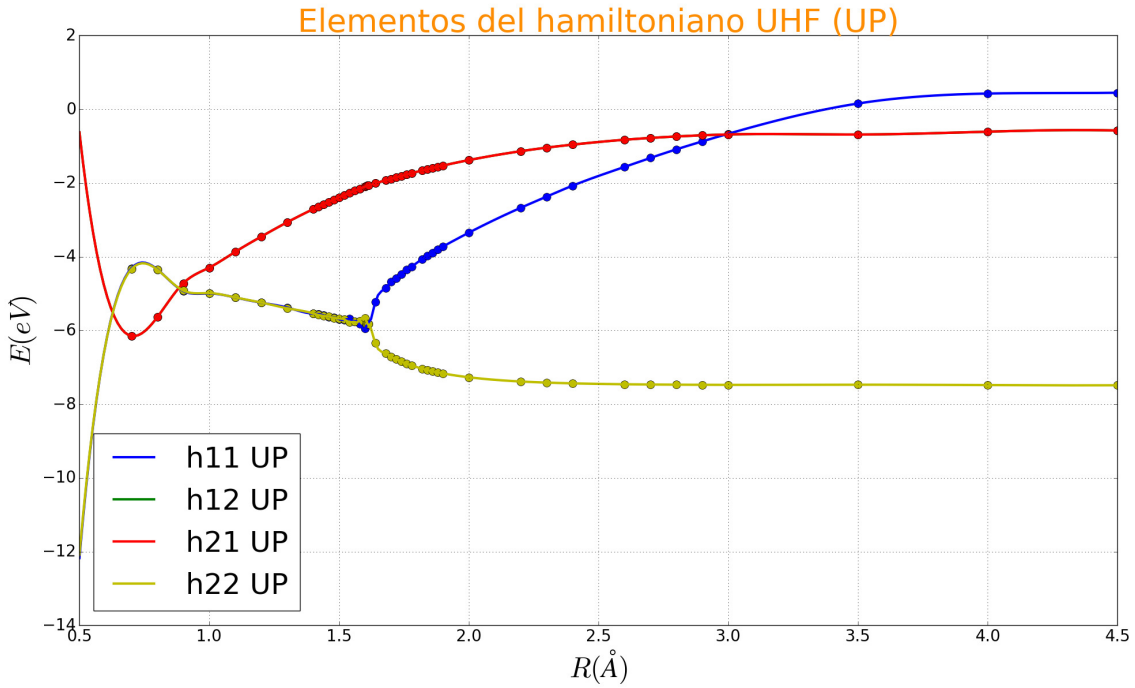

Figura 3.11: Representación de los valores de los elementos del hamiltoniano para la configuración UHF en función de la distancia internuclear, para el canal de spin "up". Los marcas representan los valores obtenidos de los cálculos de primeros principios, a partir de estos se ha procedido a realizar una interpolación mediante spline cúbicos, que nos permiten ver la tendencia de los elementos del hamiltoniano a cortas distancias.

que cómo se ha descrito, ambas configuraciones son equivalentes en ese régimen. Sin embargo, a mediada que la distancia internuclear crece, se puede ver como la fuerza en ambas configuraciones cambia. En virtud de lo expuesto en la sección Sec.2.6 la diferencia de la fuerza entre dos estados del sistema se recoge en el término de la deformación de la densidad y como se puede ver en la Fig.3.17, a partir de la configuración de referencia, y conociendo como cambia la densidad entre ambos esquemas, se reproduce correctamente la fuerza UHF.

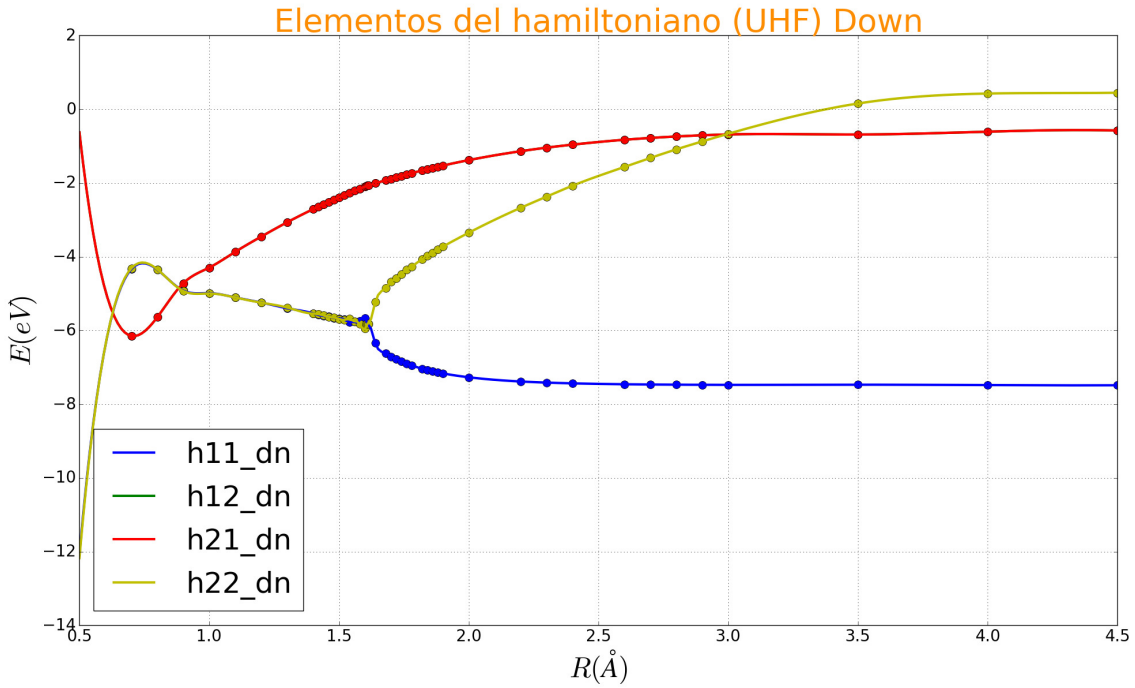

Figura 3.12: Representación de los valores de los elementos del hamiltoniano para la configuración UHF en función de la distancia internuclear, para el canal "down". Los marcas representan los valores obtenidos de los cálculos de primeros principios, a partir de estos se ha procedido a realizar una interpolación mediante spline cúbicos, que nos permiten ver la tendencia de los elementos del hamiltoniano a cortas distancias.

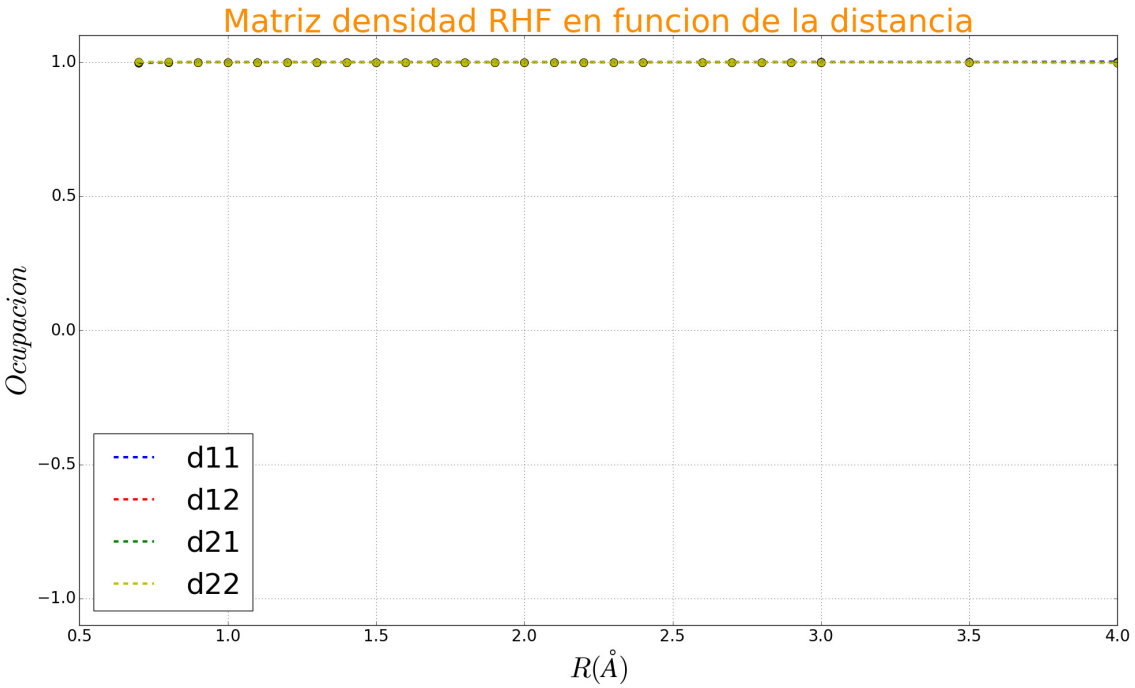

Figura 3.13: Representación de los valores de los elementos del hamiltoniano para la configuración RHF en función de la distancia internuclear. La matriz densidad permanece constante con la distancia, manteniéndose ambas Wannier ocupadas.

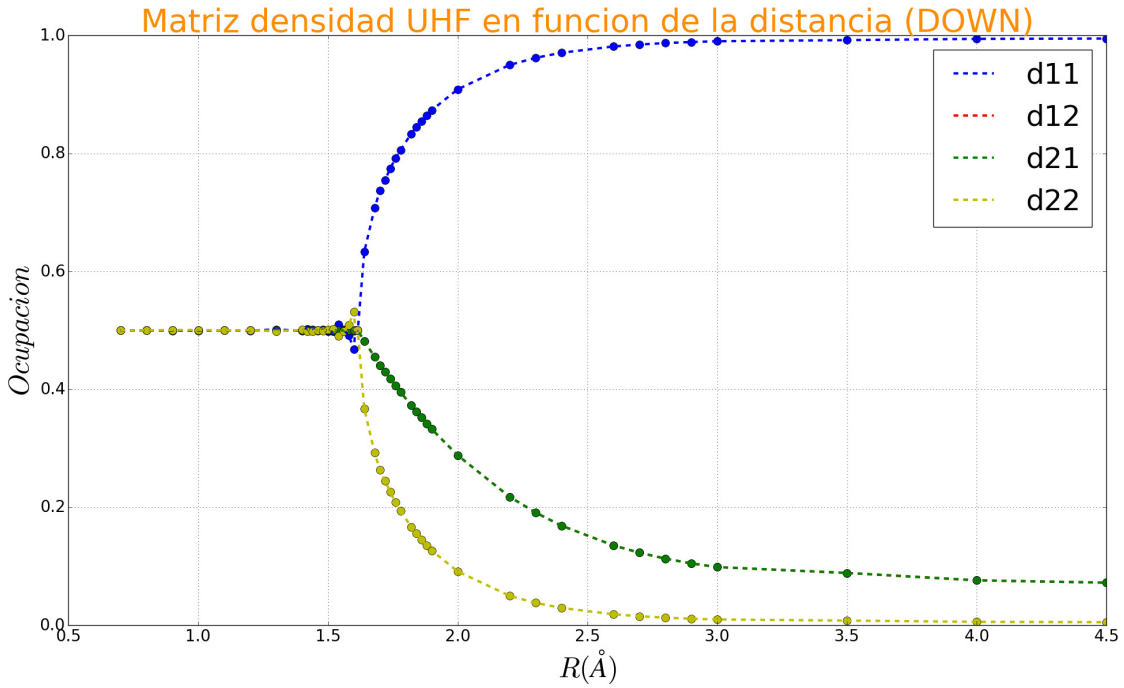

Figura 3.14: Representación de los valores de los elementos del hamiltoniano para la configuración UHF en función de la distancia internuclear. A cortas distancia la matriz densidad permanece constante, con las Wannier parcialmente ocupadas. Cuando la distancia crece, una de las Wannier se va poblando, a costa de la despoblación de la otra.

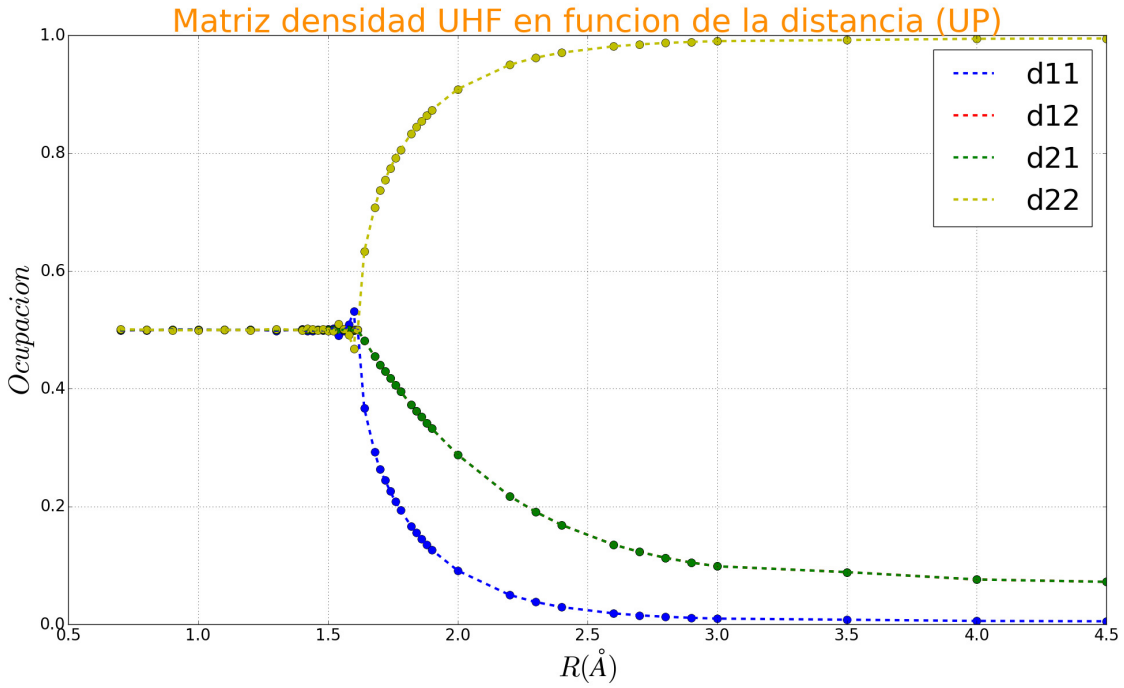

Figura 3.15: Representación de los valores de los elementos del hamiltoniano para la configuración UHF en función de la distancia internuclear. A cortas distancias la matriz densidad permanece constante, con las Wannier ocupadas parcialmente, al igual que en caso del canal "down" de spin. Cuando la distancia crece, la Wannier que se despueble en el caso "down" se puebla, por carga del canal "up".

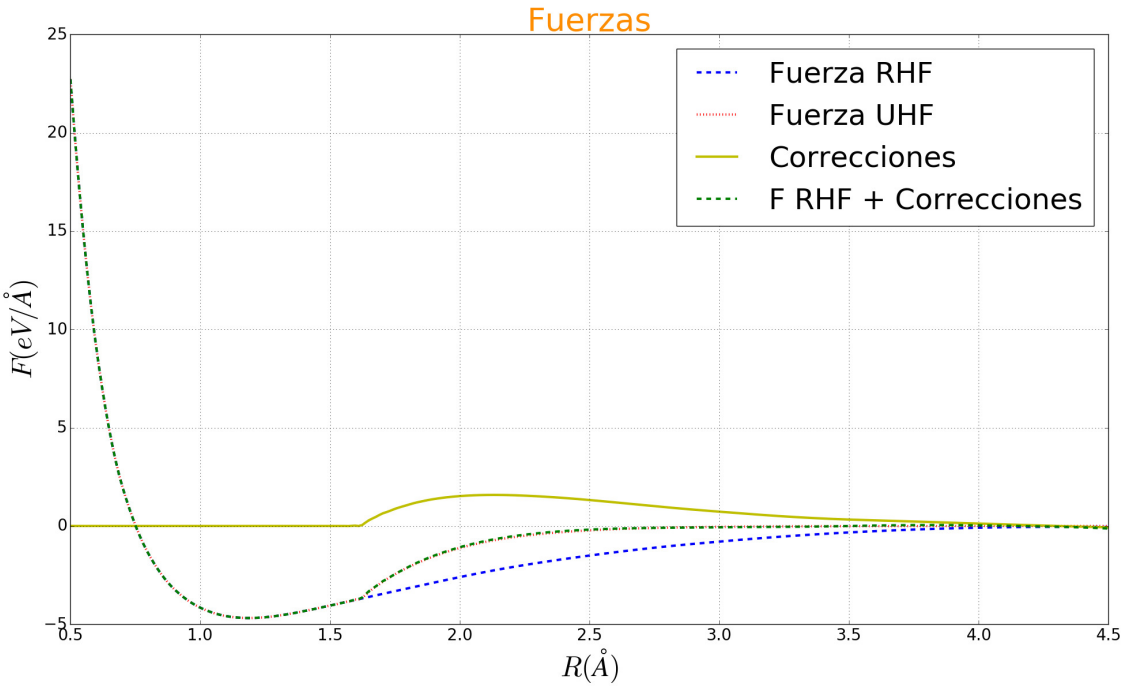

Figura 3.16: Representación de la fuerza en cada uno de los configuraciones del sistema. Estas curvas se obtienen de la expresión que relaciona la fuerza con el potencial. Mientras que las correcciones se han calculado de acuerdo al segundo término de la Ec.(2.34).

Capítulo 4

Simulaciones de H_2 a segundos principios

4.1. Introducción

El objetivo de este capítulo es detallar los aspectos técnicos del código de segundos principios, que tiene tres propósitos, concretamente la realización del cálculo de la evolución temporal de la matriz densidad, que viene descrita por la siguiente expresión

$$i\hbar\frac{d}{dt}\rho(t) = [\mathcal{H}(t), \rho(t)] \tag{4.1}$$

en segundo lugar, el cálculo del dipolo eléctrico

$$\vec{p}(t) \;=\; \int \rho\left(\vec{r}, t\right) \vec{r}\; d^3r \tag{4.2}$$

que nos permitirá observar como responde el sistema a la aplicación de un pulso de un campo eléctrico y como se alcanzan estados excitados de la molécula y por último, se implementa la integración de Verlet [14] para la resolución de la ecuación del movimiento de Newton, con el fin de obtener la dinámica del sistema.

4.2. Construcción del código

El código, recogido en el apéndice C, se ha implementado en Python (que se ha aprendido para la ocasión). En lo que sigue se describen los aspectos técnicos relacionados con los métodos numéricos implementados.

- Para la resolución de la evolución de la matriz densidad 4.1 se han implementado varios métodos con el fin de elegir aquel que tuviese menor error numérico. El método elegido aplica el operador de evolución temporal, que transforma el estado del sistema en el instante t_0 en un estado en el instante de tiempo t. Esta transformación se realiza de la siguiente forma

  $$|\phi_i(t)\rangle = \mathcal{U}(t, t_0)\,|\phi_i(t_0)\rangle \tag{4.3}$$

  donde $\mathcal{U}(t, t_0)$ es el operador unitario, denominado operador de evolución temporal que asumiendo que el hamiltoniano es constante durante el tiempo de evolución es, en unidades atómicas, [4]

  $$\mathcal{U}(t, t_0) \;=\; e^{-ih(t-t_0)} \tag{4.4}$$

Esta forma de describir los estados, permite reescribir la definición de la densidad, Ec.2.18, como sigue

$$\rho(t) = \mathcal{U}(t,t_0)\left(\sum_{i=1}^{N} o_i \ |\phi_i(t_0)\rangle\langle\phi_i(t_0)|\right)\mathcal{U}^{\dagger}(t,t_0) \tag{4.5}$$

$$\rho(t) = \mathcal{U}(t,t_0)\ \rho(t_0)\ \mathcal{U}^{\dagger}(t,t_0) \tag{4.6}$$

donde $\rho\,(t_0)$ es el valor inicial de la matriz densidad, obtenido de un cálculo estacionario de primeros principios. El cálculo del operador de evolución temporal se realiza mediante el desarrollo en serie de la exponencial con cuatro términos. La expresión 4.6 es la implementada en el código (Apéndice C). El paso temporal elegido es $h = 0.01\ fs$, mientras que la evolución se realiza hasta los 10 fs.

Los otros métodos implementados han sido Runge-Kutta 4 y el esquema de diferencias finitas [3, 8]. La comparativa de estos tres métodos para la resolución de la evolución temporal se presentan en la Fig.4.1.

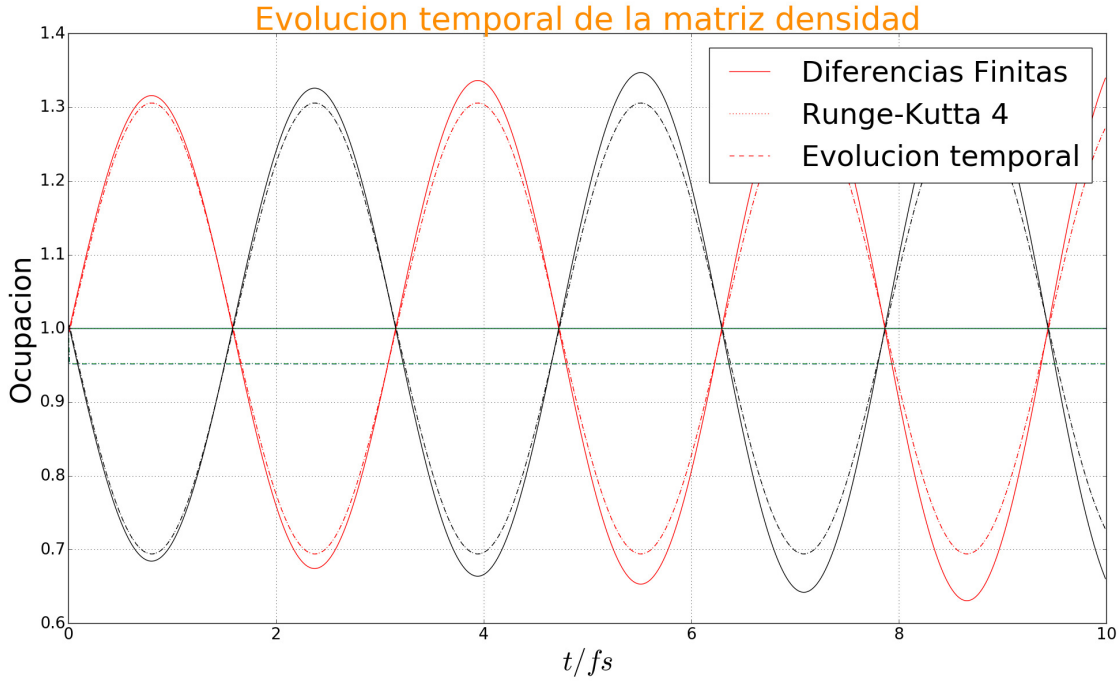

Figura 4.1: Representación de la evolución de los elementos de la matriz densidad, utilizando los tres métodos nombrados. La matriz densidad inicial usada para la prueba, ha sido la identidad. Se puede observar como los elementos de la diagonal oscilan. Se puede observar como estas oscilaciones crecen con el calculo por diferencias finitas, mientras que el uso de los otros dos métodos no conllevan este error.

- Como ya se ha mencionado para comprender como el sistema responde a la aplicación de un campo eléctrico calculamos el momento dipolar, 4.2. Como estamos desarrollando un esquema basado en el uso de la densidad electrónica, escribimos esta magnitud en términos del operador que representa a la densidad, la matriz densidad

$$\vec{p}\,(t) \ = \ \int \rho\,(\vec{r})\,\vec{r}\ d^3\vec{r} \ = \ \int \sum_{\alpha\beta} d_{\alpha\beta}\chi_\alpha\chi_\beta\ \vec{r}\ d^3\vec{r} \ = \ \sum_{\alpha\beta} d_{\alpha\beta}\ \vec{r}_{\alpha\beta} \tag{4.7}$$

En la expresión anterior, $d_{\alpha\beta}$, representa los elementos de de la matriz densidad, que se han obtenido a partir de la evolución de un estado del sistema, caracterizado por un densidad inicial, ρ_0. Las distancias $\vec{r}_{\alpha\beta}$ representa los centroides de las funciones de Wannier y los elementos dipolos eléctricos entre ellas, esta magnitud se obtiene del cálculo de primeros principios realizado con VASP.

La expresión obtenida es dependiente del tiempo. Para calcular el espectro óptico de absorción se ha procedido al cálculo de esta magnitud en el espacio de frecuencias mediante una transformación real de fourier.

- El estudio de la dinámica de un sistema pasa por la integración de la ecuación del movimiento de Newton, que es una ecuación diferencial de segundo orden

$$\ddot{\vec{x}}(t) = \frac{\vec{F}[\vec{x}(t)]}{M} \tag{4.8}$$

Donde M es la masa del sistema y $\vec{F}$ es la fuerza a la que se ve sometida el sistema. Para la resolución de 4.8 se recurre al algoritmo de Verlet, en su versión sin velocidades [14].

- El algoritmo de Verlet tiene un error de orden cuarto, $\mathcal{O}(h^4)$, con el paso de tiempo, h, elegido. En este caso se ha optado por un paso temporal de h =0.01 fs.
- No se precisa del cálculo de las velocidades para hallar las posiciones con el tiempo. El cálculo de las mismas puede realizarse una vez se han integrado las ecuaciones y obtenido las posiciones, $\vec{x}_n$, mediante la siguiente expresión iterativa

$$\begin{cases} \ddot{\vec{x}}_{n+1}(t) = 2\vec{x}_n(t) - \vec{x}_{n-1}(t) + \frac{\vec{F}[\vec{x}_n(t)]}{M}h^2 \\ \vec{x}_0(t) \\ \vec{x}_1(t) = \vec{x}_0(t) + \vec{v}_0(t)\,h + \frac{\vec{F}[\vec{x}_n(t)]}{M}h^2 \end{cases} \tag{4.9}$$

donde el índice comienza en $n = 1$. La posición inicial, $\vec{x}_0(t)$ es la distancia internuclear del sistema, mientras que en la primera iteración, $\vec{x}_1(t)$, requiere de la velocidad inicial. Si consideramos que sistema está a cierta temperatura, y se encuentra en algún nivel vibracional, se tomará la velocidad correspondiente, en caso de que nos encontremos en el estado de equilibrio la velocidad se tomará nula. Una vez integrada la ecuación diferencial se dibujan los valores obtenidos en función del tiempo.

Capítulo 5

Conclusiones

Para finalizar el trabajo se comentará en lineas generales la conclusiones extraídas de la realización del mismo.

Tras ver el comportamiento de los orbitales moleculares, se puede concluir que suponer que en el enlace de la molécula de hidrógeno solo intervienen dos orbitales moleculares es insuficiente. El cruce del orbital antienlazante con el resto de orbitales moleculares difumina su carácter, al mezclarse con el resto. Por lo tanto, para un correcto entendimiento de la molécula es necesario considerar un mayor número de orbitales moleculares, para no perder el carácter del nivel antienlazante entre el cuasi-continuo de niveles superiores. Sobre todo teniendo en cuenta que una transición vertical desde el punto de equilibrio del estado fundamental nos llevará directamente al centro de este enjambre de estados.

Otro de los objetivos del trabajo era comprobar como, dentro del formalismo de segundos principios [5], se podía reproducir la fuerza de una configuración del sistema, a partir de cálculos de primeros principios de una configuración de referencia y conociendo únicamente como varia la densidad entre las dos configuraciones. Es decir, los métodos de segundos principios se podrían extender eventualmente a situaciones donde se rompieran los enlaces. El resultado fue satisfactorio, ya que se comprobó la validez de la expresión 2.34, tal y como se muestra en la Fig. 3.17.

Por otro lado, debido a complicaciones técnicas derivadas del tratamiento y comprensión de los cálculos obtenidos de primeros principios, no se pudo concluir con todos los objetivos del trabajo. En particular, los pasos finales incluyendo la parametrización del hamiltoniano, y el consecutivo estudio de la dinámica que nos dará el acceso a estados excitados mediante la aplicación de un campo eléctrico no han podido realizarse. Sin embargo, el código que permite el estudio de la dinámica así como la respuesta del sistema a la aplicación de un pulso eléctrico está implementado (Apéndice A). Por lo tanto, el paso natural al término de este trabajo es la aplicación del código de segundos principios para el estudio de la dinámica de la molécula, ver estados excitados y ver como el alcance de algunos de estos estados desemboca en la disociación de la molécula.

Bibliografía

[1] Ashcroft, N. W., and Mermin, D. *Solid State Physics*. Harcourt College Publishers, 1976.

[2] Atkins, P., and Friedman, R. *Molecular Quantum Mechanics*. Oxford University Press, 2011.

[3] Atkinson, K., and Han, W. *Elementary numerical analysis*. John Wiley & Sons, 2003.

[4] Cohen-Tannoudji, C., Diu, B., and Laloë, F. *Quantum mechanics*. Quantum Mechanics. Wiley, 1977.

[5] García-Fernández, P., Wojdeł, J. C., Íñiguez, J., and Junquera, J. Second-principles method for materials simulations including electron and lattice degrees of freedom. *Phys. Rev. B 93* (May 2016), 195137.

[6] Golub, G., and Van Loan, C. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996.

[7] Jensen, F. *Introduction to Computational Chemistry*. John Wiley & Sons, 1999.

[8] Klein, A., and Gudonov, A. *Introductory Computational Physics*. Cambridge University Press, 2006.

[9] Kresse, G., and Furthmüller, J. Efficient iterative schemes for *ab initio* total-energy calculations using a plane-wave basis set. *Phys. Rev. B 54* (Oct 1996), 11169–11186.

[10] Kresse, G., and Hafner, J. *Ab initio* molecular dynamics for liquid metals. *Phys. Rev. B 47* (Jan 1993), 558–561.

[11] Kresse, G., and Hafner, J. *Ab initio* molecular-dynamics simulation of the liquid-metal˘amorphous-semiconductor transition in germanium. *Phys. Rev. B 49* (May 1994), 14251–14269.

[12] Marzari, N., Mostofi, A. A., Yates, J. R., Souza, I., and Vanderbilt, D. Maximally localized wannier functions: Theory and applications. *Rev. Mod. Phys. 84* (2012), 1419–1475.

[13] Perdew, J. P., Chevary, J. A., Vosko, S. H., Jackson, K. A., Pederson, M. R., Singh, D. J., and Fiolhais, C. Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation. *Phys. Rev. B 48* (Aug 1993), 4978–4978.

[14] Verlet, L. Computer experiments on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev. 159* (Jul 1967), 98–103.

Apéndice A

Código 1: Diagonalización del hamiltoniano

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

#Importamos la librería numpy y matplotlib.
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import interpolate

#########################################################################
#Diagonaliza el hamiltoniano UHF y RHF. A partir de los vectores        #
#propios obtenidos, se forman las matrices densidad para las dos        #
# configuraciones.                                                      #
#INPUT: Ficheros elementos del hamiltoniano RHF y UHF(DOWN Y UP)        #
#                                                                       #
#OUTPUT: -Graficas con los elementos de matriz de la densidad           #
#         para RHF, UHF DOWN y UHF UP                                   #
#        -Ficheros que guardan los elementos de la matriz               #
#         densidad para cada configuracion                              #
#########################################################################

################################################## MATRIZ DENSIDAD RHF

#Leemos los ficheros del hamiltoniano RHF, previamente calculados.

r_rhf, h_11_rhf = np.loadtxt('h_11_rhf.txt', usecols=(0,1), unpack=True)
h_12_rhf = np.loadtxt('h_12_rhf.txt', usecols=(1,), unpack=True)
h_21_rhf = np.loadtxt('h_21_rhf.txt', usecols=(1,), unpack=True)
h_22_rhf = np.loadtxt('h_22_rhf.txt', usecols=(1,), unpack=True)


h_rhf = []
#Creamos la matriz hamiltoniano para cada punto.
for i in range(1,len(h_11_rhf)+1):
        h_rhf.append( np.matrix( [[ h_11_rhf[i-1],h_12_rhf[i-1]], [h_21_rhf[i
            -1],h_22_rhf[i-1] ]]) )


h_rhf_diagonal = []
#Diagonalizamos la matriz para cada punto.
for i in range(1,len(h_11_rhf)+1):
        h_rhf_diagonal.append( np.linalg.eig(h_rhf[i-1]) )
```

```
c11 = []
c12 = []
c21 = []
c22 = []
#Sacamos los coeficientes
for i in range(1,len(h_11_rhf)+1):

        if h_rhf_diagonal[i-1][0].item(0) < h_rhf_diagonal[i-1][0].item(1) :
                c11.append(h_rhf_diagonal[i-1][1][:,0].item(0))
                c12.append(h_rhf_diagonal[i-1][1][:,0].item(1))
                c21.append(h_rhf_diagonal[i-1][1][:,1].item(0))
                c22.append(h_rhf_diagonal[i-1][1][:,1].item(1))
        elif h_rhf_diagonal[i-1][0].item(0) > h_rhf_diagonal[i-1][0].item(1) :
                c11.append(h_rhf_diagonal[i-1][1][:,1].item(0))
                c12.append(h_rhf_diagonal[i-1][1][:,1].item(1))
                c21.append(h_rhf_diagonal[i-1][1][:,0].item(0))
                c22.append(h_rhf_diagonal[i-1][1][:,0].item(1))



d_11 = []
d_12 = []
d_21 = []
d_22 = []
d_11_rhf_up = []
d_12_rhf_up = []
d_21_rhf_up = []
d_22_rhf_up = []
#Elementos de la matriz densidad RHF para cada punto R.
for i in range(1, len(h_11_rhf)+1):
        d_11.append( 2*c11[i-1]*c11[i-1] +  0*c21[i-1]*c21[i-1])
        d_12.append( 2*c11[i-1]*c12[i-1] +  0*c21[i-1]*c22[i-1])
        d_21.append( 2*c12[i-1]*c11[i-1] +  0*c22[i-1]*c21[i-1])
        d_22.append( 2*c12[i-1]*c12[i-1] +  0*c22[i-1]*c22[i-1])

#Matriz densidad RHF para cada canal.
for i in range(1, len(h_11_rhf)+1):
        d_11_rhf_up.append( 1*c11[i-1]*c11[i-1] +  0*c21[i-1]*c21[i-1])
        d_12_rhf_up.append( 1*c11[i-1]*c12[i-1] +  0*c21[i-1]*c22[i-1])
        d_21_rhf_up.append( 1*c12[i-1]*c11[i-1] +  0*c22[i-1]*c21[i-1])
        d_22_rhf_up.append( 1*c12[i-1]*c12[i-1] +  0*c22[i-1]*c22[i-1])


#Escrbimos los ficheros de  la matriz densidad.
f1=open("d_11_rhf.txt","w")
f2=open("d_12_rhf.txt","w")
f3=open("d_21_rhf.txt","w")
f4=open("d_22_rhf.txt","w")
f5=open("d_11_rhf_up.txt","w")
f6=open("d_12_rhf_up.txt","w")
f7=open("d_21_rhf_up.txt","w")
f8=open("d_22_rhf_up.txt","w")
for i in range(1,len(d_11)+1):
        f1.write(str(r_rhf[i-1]) + '\t'+ str(d_11[i-1]))
        f1.write('\n')
        f2.write(str(r_rhf[i-1]) + '\t'+ str(d_12[i-1]))
        f2.write('\n')
        f3.write(str(r_rhf[i-1]) + '\t'+ str(d_21[i-1]))
```

```
        f3.write('\n')
        f4.write(str(r_rhf[i-1]) + '\t'+ str(d_22[i-1]))
        f4.write('\n')
        f5.write(str(r_rhf[i-1]) + '\t'+ str(d_11_rhf_up[i-1]))
        f5.write('\n')
        f6.write(str(r_rhf[i-1]) + '\t'+ str(d_12_rhf_up[i-1]))
        f6.write('\n')
        f7.write(str(r_rhf[i-1]) + '\t'+ str(d_21_rhf_up[i-1]))
        f7.write('\n')
        f8.write(str(r_rhf[i-1]) + '\t'+ str(d_22_rhf_up[i-1]))
        f8.write('\n')

f1.close()
f2.close()
f3.close()
f4.close()
f5.close()
f6.close()
f7.close()
f8.close()

#Dibujamo la matriz densidad para la configuración RHF.

plt.figure('Matriz densidad RHF en funcion de R')
del d_11[-1]
del d_12[-1]
del d_21[-1]
del d_22[-1]
r_rhf_ = []
for i in range(0,len(r_rhf)-1):
        r_rhf_.append(r_rhf[i])

plt.plot(r_rhf_,d_11,'bo',ms=10)
plt.plot(r_rhf_,d_12,'ro',ms=10)
plt.plot(r_rhf_,d_21,'go',ms=10)
plt.plot(r_rhf_,d_22,'yo',ms=10)
plt.plot(r_rhf_,d_11,'b—', linewidth = 3, label = 'd11' )
plt.plot(r_rhf_,d_12,'r—', linewidth = 3, label = 'd12' )
plt.plot(r_rhf_,d_21,'g—', linewidth = 3, label = 'd21' )
plt.plot(r_rhf_,d_22,'y—', linewidth = 3, label = 'd22' )

#Colocamos la leyenda.
plt.legend(loc=0,prop={'size':36})

#Ponemos la rejilla.
plt.grid(True)

#Ponemos el título.
plt.title("Matriz densidad RHF en funcion de la distancia", fontsize = 40, color
    = 'darkorange', verticalalignment = 'baseline', horizontalalignment = '
   center' )

#Ponemos nombre a los ejes.
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))

#Ponemos nombre a los ejes.
plt.ylabel('$Ocupacion $', fontsize = 35, color = (0,0,0))

#Ponemos limites eje Y.
plt.ylim(-1.10,1.10)
```

```
#Agrandamos los valores de los ejes.
plt.yticks(size=20)
plt.xticks(size=20)

################################################## MATRIZ DENSIDAD UHF

#------------------------------------------------- Matriz densidad UHF UP

#Leemos los ficheros del hamiltoniano UHF UP, previamente calculados.
r_uhf, h_11_uhf = np.loadtxt('h_11_uhf.txt', usecols=(0,1), unpack=True)
h_12_uhf = np.loadtxt('h_12_uhf.txt', usecols=(1,), unpack=True)
h_21_uhf = np.loadtxt('h_21_uhf.txt', usecols=(1,), unpack=True)
h_22_uhf = np.loadtxt('h_22_uhf.txt', usecols=(1,), unpack=True)

o1_uhf = np.loadtxt('o1_uhf.txt', usecols=(1,), unpack=True)
o2_uhf = np.loadtxt('o2_uhf.txt', usecols=(1,), unpack=True)
#Rehacemos la lista de los elementos h11 y h22.
for i in range(1,3) :
        h_11_uhf.put(i-1, h_11_rhf[i-1] )
        h_22_uhf.put(i-1, h_22_rhf[i-1] )
        h_12_uhf.put(i-1, h_12_rhf[i-1])
        h_21_uhf.put(i-1, h_21_rhf[i-1])

h_uhf = []
#Creamos la matriz hamiltoniano para cada punto.
for i in range(1,len(h_11_uhf)+1):
        h_uhf.append( np.matrix( [[ h_11_uhf[i-1],h_12_uhf[i-1]], [ h_21_uhf[i
            -1],h_22_uhf[i-1] ]]) )

h_uhf_diagonal = []
#Diagonalizamos la matriz para cada punto.
for i in range(1,len(h_11_uhf)+1):
        h_uhf_diagonal.append( np.linalg.eig(h_uhf[i-1]) )


c11_uhf = []
c12_uhf = []
c21_uhf = []
c22_uhf = []
#Sacamos los coeficientes.
for i in range(1,len(h_11_uhf)+1):

        if h_uhf_diagonal[i-1][0].item(0) < h_uhf_diagonal[i-1][0].item(1) :
                c11_uhf.append(h_uhf_diagonal[i-1][1][:,0].item(0))
                c12_uhf.append(h_uhf_diagonal[i-1][1][:,0].item(1))
                c21_uhf.append(h_uhf_diagonal[i-1][1][:,1].item(0))
                c22_uhf.append(h_uhf_diagonal[i-1][1][:,1].item(1))
        elif h_uhf_diagonal[i-1][0].item(0) > h_uhf_diagonal[i-1][0].item(1) :
                c11_uhf.append(h_uhf_diagonal[i-1][1][:,1].item(0))
                c12_uhf.append(h_uhf_diagonal[i-1][1][:,1].item(1))
                c21_uhf.append(h_uhf_diagonal[i-1][1][:,0].item(0))
                c22_uhf.append(h_uhf_diagonal[i-1][1][:,0].item(1))

d_11_uhf = []
d_12_uhf = []
d_21_uhf = []
d_22_uhf = []

#Elementos de la matriz densidad para cada punto R.
```

```
for i in range(1, len(h_11_uhf)+1):
        d_11_uhf.append(o1_uhf[i-1]*c11_uhf[i-1]*c11_uhf[i-1] + o2_uhf[i-1]*
            c21_uhf[i-1]*c21_uhf[i-1])
        d_12_uhf.append(o1_uhf[i-1]*c11_uhf[i-1]*c12_uhf[i-1] + o2_uhf[i-1]*
            c22_uhf[i-1]*c21_uhf[i-1])
        d_21_uhf.append(o1_uhf[i-1]*c12_uhf[i-1]*c11_uhf[i-1] + o2_uhf[i-1]*
            c21_uhf[i-1]*c22_uhf[i-1])
        d_22_uhf.append(o1_uhf[i-1]*c12_uhf[i-1]*c12_uhf[i-1] + o2_uhf[i-1]*
            c22_uhf[i-1]*c22_uhf[i-1])

#Escrbimos los ficheros con la matriz densidad.
f1=open("d_11_uhf.txt","w")
f2=open("d_12_uhf.txt","w")
f3=open("d_21_uhf.txt","w")
f4=open("d_22_uhf.txt","w")
for i in range(1,len(d_11_uhf)+1):
        f1.write(str(r_uhf[i-1]) + '\t'+ str(d_11_uhf[i-1]))
        f1.write('\n')
        f2.write(str(r_uhf[i-1]) + '\t'+ str(d_12_uhf[i-1]))
        f2.write('\n')
        f3.write(str(r_uhf[i-1]) + '\t'+ str(d_21_uhf[i-1]))
        f3.write('\n')
        f4.write(str(r_uhf[i-1]) + '\t'+ str(d_22_uhf[i-1]))
        f4.write('\n')

f1.close()
f2.close()
f3.close()
f4.close()

#Dibujamos la matriz densidad para la configuración UHF.

plt.figure('Matriz densidad UHF en funcion de R (UP)')

plt.plot(r_uhf,d_11_uhf,'bo',ms=10)
plt.plot(r_uhf,d_12_uhf,'ro',ms=10)
plt.plot(r_uhf,d_21_uhf,'go',ms=10)
plt.plot(r_uhf,d_22_uhf,'yo',ms=10)
plt.plot(r_uhf,d_11_uhf,'b—', linewidth = 3, label = 'd11' )
plt.plot(r_uhf,d_12_uhf,'r—', linewidth = 3, label = 'd12' )
plt.plot(r_uhf,d_21_uhf,'g—', linewidth = 3, label = 'd21' )
plt.plot(r_uhf,d_22_uhf,'y—', linewidth = 3, label = 'd22' )

#Colocamos la leyenda.
plt.legend(loc=0,prop={'size':36})

#Ponemos la rejilla.
plt.grid(True)

#Ponemos el título.
plt.title("Matriz densidad UHF en funcion de la distancia (UP)", fontsize = 40,
    color = 'darkorange', verticalalignment = 'baseline', horizontalalignment = '
    center' )

#Ponemos nombre a los ejes.
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))

#Ponemos nombre a los ejes.
plt.ylabel('$Ocupacion $', fontsize = 35, color = (0,0,0))
```

```
#Agrandamos los valores de los ejes.
plt.yticks(size=20)
plt.xticks(size=20)



#------------------------------------------- Matriz densidad UHF DOWN

#Leemos los ficheros del hamiltoniano UHF DOWN.

h_11_uhf_d = np.loadtxt('h_22_uhf.txt', usecols=(1,), unpack=True)
h_12_uhf_d = np.loadtxt('h_12_uhf.txt', usecols=(1,), unpack=True)
h_21_uhf_d = np.loadtxt('h_21_uhf.txt', usecols=(1,), unpack=True)
h_22_uhf_d = np.loadtxt('h_11_uhf.txt', usecols=(1,), unpack=True)


#Rehacemos la lista de los elementos h11 y h22
for i in range(1,3) :
        h_11_uhf_d.put(i-1, h_22_rhf[i-1])
        h_22_uhf_d.put(i-1, h_22_rhf[i-1])
        h_12_uhf_d.put(i-1, h_12_rhf[i-1])
        h_21_uhf_d.put(i-1, h_21_rhf[i-1])


h_uhf_d = []
#Creamos la matriz hamiltoniano para cada punto.
for i in range(1,len(h_11_uhf_d)+1):
        h_uhf_d.append( np.matrix( [[ h_11_uhf_d[i-1],h_12_uhf_d[i-1]], [
            h_21_uhf_d[i-1],h_22_uhf_d[i-1] ]]) )

h_uhf_diagonal_d = []
#Diagonalizamos la matriz para cada punto.
for i in range(1,len(h_11_uhf_d)+1):
        h_uhf_diagonal_d.append( np.linalg.eig(h_uhf_d[i-1]) )


c11_uhf_d = []
c12_uhf_d = []
c21_uhf_d = []
c22_uhf_d = []
#Sacamos los coeficientes.
for i in range(1,len(h_11_uhf_d)+1):
        if h_uhf_diagonal_d[i-1][0].item(0) < h_uhf_diagonal_d[i-1][0].item(1) :
                c11_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,0].item(0))
                c12_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,0].item(1))
                c21_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,1].item(0))
                c22_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,1].item(1))
        elif h_uhf_diagonal_d[i-1][0].item(0) > h_uhf_diagonal_d[i-1][0].item(1)
            :
                c11_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,1].item(0))
                c12_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,1].item(1))
                c21_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,0].item(0))
                c22_uhf_d.append(h_uhf_diagonal_d[i-1][1][:,0].item(1))



d_11_uhf_d = []
d_12_uhf_d = []
d_21_uhf_d = []
d_22_uhf_d = []
```

```
#Elementos de la matriz densidad para cada punto R.
for i in range(1, len(h_11_uhf_d)+1):
        d_11_uhf_d.append(o1_uhf[i-1]*c11_uhf_d[i-1]*c11_uhf_d[i-1] + o2_uhf[i
            -1]*c21_uhf_d[i-1]*c21_uhf_d[i-1])
        d_12_uhf_d.append(o1_uhf[i-1]*c11_uhf_d[i-1]*c12_uhf_d[i-1] + o2_uhf[i
            -1]*c22_uhf_d[i-1]*c21_uhf_d[i-1])
        d_21_uhf_d.append(o1_uhf[i-1]*c12_uhf_d[i-1]*c11_uhf_d[i-1] + o2_uhf[i
            -1]*c21_uhf_d[i-1]*c22_uhf_d[i-1])
        d_22_uhf_d.append(o1_uhf[i-1]*c12_uhf_d[i-1]*c12_uhf_d[i-1] + o2_uhf[i
            -1]*c22_uhf_d[i-1]*c22_uhf_d[i-1])


#Escrbimos los ficheros de la matriz densidad.
f1=open("d_11_uhf_d.txt","w")
f2=open("d_12_uhf_d.txt","w")
f3=open("d_21_uhf_d.txt","w")
f4=open("d_22_uhf_d.txt","w")
for i in range(1,len(d_11_uhf_d)+1):
        f1.write(str(r_uhf[i-1]) + '\t'+ str(d_11_uhf_d[i-1]))
        f1.write('\n')
        f2.write(str(r_uhf[i-1]) + '\t'+ str(d_12_uhf_d[i-1]))
        f2.write('\n')
        f3.write(str(r_uhf[i-1]) + '\t'+ str(d_21_uhf_d[i-1]))
        f3.write('\n')
        f4.write(str(r_uhf[i-1]) + '\t'+ str(d_22_uhf_d[i-1]))
        f4.write('\n')

f1.close()
f2.close()
f3.close()
f4.close()

#Dibujamos la matriz densidad para la configuración UHF DOWN.

plt.figure('Matriz densidad UHF en funcion de R (DOWN)')

plt.plot(r_uhf,d_11_uhf_d,'bo',ms=10)
plt.plot(r_uhf,d_12_uhf_d,'ro',ms=10)
plt.plot(r_uhf,d_21_uhf_d,'go',ms=10)
plt.plot(r_uhf,d_22_uhf_d,'yo',ms=10)
plt.plot(r_uhf,d_11_uhf_d,'b—', linewidth = 3, label = 'd11' )
plt.plot(r_uhf,d_12_uhf_d,'r—', linewidth = 3, label = 'd12' )
plt.plot(r_uhf,d_21_uhf_d,'g—', linewidth = 3, label = 'd21' )
plt.plot(r_uhf,d_22_uhf_d,'y—', linewidth = 3, label = 'd22' )

#Colocamos la leyenda.
plt.legend(loc=0,prop={'size':36})

#Ponemos la rejilla.
plt.grid(True)

#Ponemos el título.
plt.title("Matriz densidad UHF en funcion de la distancia (DOWN)", fontsize =
    40, color = 'darkorange', verticalalignment = 'baseline', horizontalalignment
     = 'center' )

#Ponemos nombre a los ejes.
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
```

```
#Ponemos nombre a los ejes.
plt.ylabel('$Ocupacion $', fontsize = 35, color = (0,0,0))

#Agrandamos los valores de los ejes.
plt.yticks(size=20)
plt.xticks(size=20)

#Mostramos la RHF Y RHF.
plt.show()
```

Apéndice B

Código 2: Representación y cálculo de fuerzas

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-


import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate

#########################################################################
#Este código lee los ficheros que contiene los valores de              #
# primeros principios                                                  #
#INPUT : Energía RHF, Energia UHF, Elementos hamiltoniano RHF,         #
#        UHF (UP Y DOWN)                                              #
#        Energía de la bandas RHF y UHF                               #
#                                                                     #
#Se interpolan los datos de la energía y de los elementos             #
#del hamiltoniano                                                     #
#OUTPUT: Gráfico de los datos leeidos y también de las fuerzas        #
#        UHF y RHF, asi como las correciones.                         #
#########################################################################

########################################################## Energia RHF
plt.figure('Energia RHF')

#Sacamos los datos
r_rhf, e_rhf = np.loadtxt('energiaRHF.txt', usecols=(0,1), unpack=True)
xnew = np.linspace(0.50,4.50,num=300)
xnew1 = np.linspace(0.30,4.50, num=300)

#Puntos de Primeros principios
plt.plot(r_rhf,e_rhf,'ro',ms=10)

#Spline
tck = interpolate.splrep(r_rhf,e_rhf,s=0)
e_RHF = interpolate.splev(xnew1,tck,der=0)

#Derivada del spline (FUERZA RHF)
F_RHF = interpolate.splev(xnew,tck,der=1)

#Dibujamos la energia
plt.plot(xnew1, e_RHF,'g', linewidth = 3, label='E_RHF')
```

```
#Ponemos la rejilla
plt.grid(True)

#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = 'black')
plt.ylabel('$E/ eV $', fontsize = 35, color = 'black')

#Titulo
plt.title('Energia RHF', fontsize = 40, color = 'darkorange', verticalalignment
    = 'baseline', horizontalalignment = 'center')

#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)

#Leyenda
plt.legend(loc =0,prop={'size':36})

############################################################ FUERZA RHF

plt.figure('Fuerzas')

#Dibujamos fuerza UHF
plt.plot(xnew, -F_RHF, 'b--', linewidth = 3, label = 'Fuerza RHF')

#Ponemos la rejilla
plt.grid(True)

#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$F/ eV/\AA $', fontsize = 35, color = (0,0,0))

#Titulo
plt.title('Fuerzas', fontsize = 40, color = 'darkorange', verticalalignment = '
    baseline', horizontalalignment = 'center')

#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)

#Leyenda
plt.legend(loc =0,prop={'size':36})

################################### Elementos del hamiltoniano (RHF)

R_RHF, h11_rhf = np.loadtxt('h_11_rhf.txt', usecols=(0,1), unpack=True)
h12_rhf = np.loadtxt('h_12_rhf.txt', usecols=(1,), unpack=True)
h21_rhf = np.loadtxt('h_21_rhf.txt', usecols=(1,), unpack=True)
h22_rhf = np.loadtxt('h_22_rhf.txt', usecols=(1,), unpack=True)

plt.figure('Elementos del hamiltoniano RHF')

#Spline
tck11 = interpolate.splrep(R_RHF,h11_rhf,s=0)
tck12 = interpolate.splrep(R_RHF,h12_rhf,s=0)
tck21 = interpolate.splrep(R_RHF,h21_rhf,s=0)
tck22 = interpolate.splrep(R_RHF,h22_rhf,s=0)
h11 = interpolate.splev(xnew,tck11,der=0)
h12 = interpolate.splev(xnew,tck12,der=0)
h21 = interpolate.splev(xnew,tck21,der=0)
```

```
h22 = interpolate.splev(xnew,tck22,der=0)
der_h11 = interpolate.splev(xnew,tck11,der=1)
der_h12 = interpolate.splev(xnew,tck12,der=1)
der_h21 = interpolate.splev(xnew,tck21,der=1)
der_h22 = interpolate.splev(xnew,tck22,der=1)

#Dibujamos los elementos del Hamiltoniano
plt.plot(R_RHF,h11_rhf,'bo',ms=10)
plt.hold(True)
plt.plot(R_RHF,h12_rhf,'go',ms=10)
plt.plot(R_RHF,h21_rhf,'ro',ms=10)
plt.plot(R_RHF,h22_rhf,'yo',ms=10)
plt.plot(xnew,h11,'b',linewidth = 3, label='h11')
plt.hold(True)
plt.plot(xnew,h12,'g',linewidth = 3, label='h12')
plt.plot(xnew,h21,'r',linewidth = 3, label='h21')
plt.plot(xnew,h22,'y',linewidth = 3, label='h22')

#Leyenda
plt.legend(loc=0,prop={'size':36})

#Titulo
plt.title('Elementos del hamiltoniano RHF', fontsize = 40, color = 'darkorange',
     verticalalignment = 'baseline', horizontalalignment = 'center')

#Etiquetas de los ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/ eV $', fontsize = 35, color = (0,0,0))

#Tamaño de los ejes
plt.yticks(size=20)
plt.xticks(size=20)

#Ponemos la rejilla
plt.grid(True)

############################################################ Bandas RHF
plt.figure('Energia de las bandas (RHF)')
#Obtenemos datos
r_RHF, orb1_RHF = np.loadtxt('orb1RHF.txt', usecols=(0,1), unpack=True)
orb2_RHF = np.loadtxt('orb2RHF.txt', usecols=(1,), unpack=True)
orb3_RHF = np.loadtxt('orb3RHF.txt', usecols=(1,), unpack=True)
orb4_RHF = np.loadtxt('orb4RHF.txt', usecols=(1,), unpack=True)
orb5_RHF = np.loadtxt('orb5RHF.txt', usecols=(1,), unpack=True)
orb6_RHF = np.loadtxt('orb6RHF.txt', usecols=(1,), unpack=True)
orb7_RHF = np.loadtxt('orb7RHF.txt', usecols=(1,), unpack=True)
orb8_RHF = np.loadtxt('orb8RHF.txt', usecols=(1,), unpack=True)
orb9_RHF = np.loadtxt('orb9RHF.txt', usecols=(1,), unpack=True)
orb10_RHF = np.loadtxt('orb10RHF.txt', usecols=(1,), unpack=True)
orb11_RHF = np.loadtxt('orb11RHF.txt', usecols=(1,), unpack=True)
orb12_RHF = np.loadtxt('orb12RHF.txt', usecols=(1,), unpack=True)
orb13_RHF = np.loadtxt('orb13RHF.txt', usecols=(1,), unpack=True)
orb14_RHF = np.loadtxt('orb14RHF.txt', usecols=(1,), unpack=True)
orb15_RHF = np.loadtxt('orb15RHF.txt', usecols=(1,), unpack=True)
orb16_RHF = np.loadtxt('orb16RHF.txt', usecols=(1,), unpack=True)
orb17_RHF = np.loadtxt('orb17RHF.txt', usecols=(1,), unpack=True)
orb18_RHF = np.loadtxt('orb18RHF.txt', usecols=(1,), unpack=True)
orb19_RHF = np.loadtxt('orb19RHF.txt', usecols=(1,), unpack=True)
orb20_RHF = np.loadtxt('orb20RHF.txt', usecols=(1,), unpack=True)
orb21_RHF = np.loadtxt('orb21RHF.txt', usecols=(1,), unpack=True)
```

```
orb22_RHF = np.loadtxt('orb22RHF.txt', usecols=(1,), unpack=True)
orb23_RHF = np.loadtxt('orb23RHF.txt', usecols=(1,), unpack=True)
orb24_RHF = np.loadtxt('orb24RHF.txt', usecols=(1,), unpack=True)
orb25_RHF = np.loadtxt('orb25RHF.txt', usecols=(1,), unpack=True)
orb26_RHF = np.loadtxt('orb26RHF.txt', usecols=(1,), unpack=True)
orb27_RHF = np.loadtxt('orb27RHF.txt', usecols=(1,), unpack=True)
orb28_RHF = np.loadtxt('orb28RHF.txt', usecols=(1,), unpack=True)
orb29_RHF = np.loadtxt('orb29RHF.txt', usecols=(1,), unpack=True)
orb30_RHF = np.loadtxt('orb20RHF.txt', usecols=(1,), unpack=True)
orb31_RHF = np.loadtxt('orb31RHF.txt', usecols=(1,), unpack=True)
orb32_RHF = np.loadtxt('orb32RHF.txt', usecols=(1,), unpack=True)
orb33_RHF = np.loadtxt('orb33RHF.txt', usecols=(1,), unpack=True)
orb34_RHF = np.loadtxt('orb34RHF.txt', usecols=(1,), unpack=True)
orb35_RHF = np.loadtxt('orb35RHF.txt', usecols=(1,), unpack=True)
orb36_RHF = np.loadtxt('orb36RHF.txt', usecols=(1,), unpack=True)
orb37_RHF = np.loadtxt('orb37RHF.txt', usecols=(1,), unpack=True)
orb38_RHF = np.loadtxt('orb38RHF.txt', usecols=(1,), unpack=True)
orb39_RHF = np.loadtxt('orb39RHF.txt', usecols=(1,), unpack=True)
orb40_RHF = np.loadtxt('orb40RHF.txt', usecols=(1,), unpack=True)

plt.plot(r_RHF, orb1_RHF, 'ro', ms=10, )
plt.plot(r_RHF, orb2_RHF, 'go', ms=10, )
plt.plot(r_RHF, orb3_RHF, 'bo', ms=10, )
plt.plot(r_RHF, orb4_RHF, 'ko', ms=10, )
plt.plot(r_RHF, orb5_RHF, 'mo', ms=10, )
plt.plot(r_RHF, orb6_RHF, 'co', ms=10, )
plt.plot(r_RHF, orb7_RHF, 'yo', ms=10, )
plt.plot(r_RHF, orb8_RHF,  color = 'darkorange', marker = 'o', ms =10)
plt.plot(r_RHF, orb9_RHF,  color = 'sienna', marker = 'o', ms =10)
plt.plot(r_RHF, orb10_RHF, color = 'olivedrab', marker = 'o', ms =10)
plt.plot(r_RHF, orb11_RHF, color = 'dodgerblue', marker = 'o', ms =10)
plt.plot(r_RHF, orb12_RHF, color = 'mediumorchid', marker = 'o', ms =10)
plt.plot(r_RHF, orb13_RHF, color = 'salmon', marker = 'o', ms =10)
plt.plot(r_RHF, orb14_RHF, color = 'maroon', marker = 'o', ms =10)
plt.plot(r_RHF, orb15_RHF, color = 'plum', marker = 'o', ms =10)
plt.plot(r_RHF, orb16_RHF, color = 'thistle', marker = 'o', ms =10)
plt.plot(r_RHF, orb17_RHF, 'rs', ms=10, )
plt.plot(r_RHF, orb18_RHF, 'gs', ms=10, )
plt.plot(r_RHF, orb19_RHF, 'bs', ms=10, )
plt.plot(r_RHF, orb20_RHF, 'ks', ms=10, )
plt.plot(r_RHF, orb21_RHF, 'ms', ms=10, )
plt.plot(r_RHF, orb22_RHF, 'cs', ms=10, )
plt.plot(r_RHF, orb23_RHF, 'ys', ms=10, )
plt.plot(r_RHF, orb24_RHF, color = 'darkorange', marker = 's', ms =10)
plt.plot(r_RHF, orb25_RHF, color = 'sienna', marker = 's', ms =10)
plt.plot(r_RHF, orb26_RHF, color = 'olivedrab', marker = 's', ms =10)
plt.plot(r_RHF, orb27_RHF, color = 'dodgerblue', marker = 's', ms =10)
plt.plot(r_RHF, orb28_RHF, color = 'mediumorchid', marker = 's', ms =10)
plt.plot(r_RHF, orb29_RHF, color = 'salmon', marker = 's', ms =10)
plt.plot(r_RHF, orb30_RHF, color = 'maroon', marker = 's', ms =10)
plt.plot(r_RHF, orb31_RHF, color = 'plum', marker = 's', ms =10)
plt.plot(r_RHF, orb32_RHF, color = 'thistle', marker = 's', ms =10)
plt.plot(r_RHF, orb33_RHF, 'rs', ms=10, )
plt.plot(r_RHF, orb34_RHF, 'gs', ms=10, )
plt.plot(r_RHF, orb35_RHF, 'bs', ms=10, )
plt.plot(r_RHF, orb36_RHF, 'ks', ms=10, )
plt.plot(r_RHF, orb37_RHF, 'ms', ms=10, )
plt.plot(r_RHF, orb38_RHF, 'cs', ms=10, )
plt.plot(r_RHF, orb39_RHF, 'ys', ms=10, )
plt.plot(r_RHF, orb40_RHF, color = 'darkorange', marker = 's', ms =10)
```

```
plt.plot(r_RHF, orb1_RHF, 'r—', linewidth = 3, label = 'orb1')
plt.plot(r_RHF, orb2_RHF, 'g—', linewidth = 3, label = 'orb2')
plt.plot(r_RHF, orb3_RHF, 'b—', linewidth = 3, label = 'orb3')
plt.plot(r_RHF, orb4_RHF, 'k—', linewidth = 3, label = 'orb4')
plt.plot(r_RHF, orb5_RHF, 'm—', linewidth = 3, label = 'orb5')
plt.plot(r_RHF, orb6_RHF, 'c—', linewidth = 3, label = 'orb6')
plt.plot(r_RHF, orb7_RHF, 'y—', linewidth = 3, label = 'orb7')
plt.plot(r_RHF, orb8_RHF, color = 'darkorange', linestyle = '—' , linewidth =
    3, label = 'orb8')
plt.plot(r_RHF, orb9_RHF, color = 'sienna', linestyle = '—' , linewidth = 3,
    label = 'orb9')
plt.plot(r_RHF, orb10_RHF, color = 'olivedrab', linestyle = '—' , linewidth =
    3, label = 'orb10')
plt.plot(r_RHF, orb11_RHF, color = 'dodgerblue', linestyle = '—' , linewidth =
    3, label = 'orb11')
plt.plot(r_RHF, orb12_RHF, color = 'mediumorchid', linestyle = '—' , linewidth
    = 3, label = 'orb12')
plt.plot(r_RHF, orb13_RHF, color = 'salmon', linestyle = '—' , linewidth = 3,
    label = 'orb13')
plt.plot(r_RHF, orb14_RHF, color = 'maroon', linestyle = '—' , linewidth = 3,
    label = 'orb14')
plt.plot(r_RHF, orb15_RHF, color = 'plum', linestyle = '—' , linewidth = 3,
    label = 'orb15')
plt.plot(r_RHF, orb16_RHF, color = 'thistle', linestyle = '—' , linewidth = 3,
    label = 'orb16')
plt.plot(r_RHF, orb17_RHF, 'r—', linewidth = 3, label = 'orb17')
plt.plot(r_RHF, orb18_RHF, 'g—', linewidth = 3, label = 'orb18')
plt.plot(r_RHF, orb19_RHF, 'b—', linewidth = 3, label = 'orb19')
plt.plot(r_RHF, orb20_RHF, 'k—', linewidth = 3, label = 'orb20')
plt.plot(r_RHF, orb21_RHF, 'm—', linewidth = 3, label = 'orb21')
plt.plot(r_RHF, orb22_RHF, 'c—', linewidth = 3, label = 'orb22')
plt.plot(r_RHF, orb23_RHF, 'y—', linewidth = 3, label = 'orb23')
plt.plot(r_RHF, orb24_RHF, color = 'darkorange', linestyle = '—' , linewidth =
    3, label = 'orb24')
plt.plot(r_RHF, orb25_RHF, color = 'sienna', linestyle = '—' , linewidth = 3,
    label = 'orb25')
plt.plot(r_RHF, orb26_RHF, color = 'olivedrab', linestyle = '—' , linewidth =
    3, label = 'orb26')
plt.plot(r_RHF, orb27_RHF, color = 'dodgerblue', linestyle = '—' , linewidth =
    3, label = 'orb27')
plt.plot(r_RHF, orb28_RHF, color = 'mediumorchid', linestyle = '—' , linewidth
    = 3, label = 'orb28')
plt.plot(r_RHF, orb29_RHF, color = 'salmon', linestyle = '—' , linewidth = 3,
    label = 'orb29')
plt.plot(r_RHF, orb30_RHF, color = 'maroon', linestyle = '—' , linewidth = 3,
    label = 'orb30')
plt.plot(r_RHF, orb31_RHF, color = 'plum', linestyle = '—' , linewidth = 3,
    label = 'orb31')
plt.plot(r_RHF, orb32_RHF, color = 'thistle', linestyle = '—' , linewidth = 3,
    label = 'orb32')
plt.plot(r_RHF, orb33_RHF, 'r—', linewidth = 3, label = 'orb33')
plt.plot(r_RHF, orb34_RHF, 'g—', linewidth = 3, label = 'orb34')
plt.plot(r_RHF, orb35_RHF, 'b—', linewidth = 3, label = 'orb35')
plt.plot(r_RHF, orb36_RHF, 'k—', linewidth = 3, label = 'orb36')
plt.plot(r_RHF, orb37_RHF, 'm—', linewidth = 3, label = 'orb37')
plt.plot(r_RHF, orb38_RHF, 'c—', linewidth = 3, label = 'orb38')
plt.plot(r_RHF, orb39_RHF, 'y—', linewidth = 3, label = 'orb39')
plt.plot(r_RHF, orb40_RHF, color = 'darkorange', linestyle = '—' , linewidth =
    3, label = 'orb40')
```

```
#Ponemos la rejilla
plt.grid(True)

#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/eV$', fontsize = 35, color = (0,0,0))

#Titulo
plt.title('Energia de la bandas RHF', fontsize = 40, color = 'darkorange',
    verticalalignment = 'baseline', horizontalalignment = 'center')

#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)

################################### Elementos del hamiltoniano (UHF)

#-------------------------------------------------------Hamiltoniano UP
R_UHF, h11_uhf_0 = np.loadtxt('h_11_uhf.txt', usecols=(0,1), unpack=True)
h11_uhf = np.loadtxt('h_11_uhf.txt', usecols=(1,), unpack=True)
h12_uhf = np.loadtxt('h_12_uhf.txt', usecols=(1,), unpack=True)
h21_uhf = np.loadtxt('h_21_uhf.txt', usecols=(1,), unpack=True)
h22_uhf = np.loadtxt('h_22_uhf.txt', usecols=(1,), unpack=True)


#Rehacemos la lista de los elementos h11 y h22
for i in range(1,3) :
        h11_uhf.put(i-1, h11_rhf[i-1])
        h22_uhf.put(i-1, h22_rhf[i-1])
        h12_uhf.put(i-1, h12_rhf[i-1])
        h21_uhf.put(i-1, h21_rhf[i-1])

plt.figure('Elementos del hamiltoniano UHF (UP)')

#Spline
tck11_up = interpolate.splrep(R_UHF,h11_uhf,s=0)
tck12_up = interpolate.splrep(R_UHF,h12_uhf,s=0)
tck21_up = interpolate.splrep(R_UHF,h21_uhf,s=0)
tck22_up = interpolate.splrep(R_UHF,h22_uhf,s=0)
h11_up = interpolate.splev(xnew,tck11_up,der=0)
h12_up = interpolate.splev(xnew,tck12_up,der=0)
h21_up = interpolate.splev(xnew,tck21_up,der=0)
h22_up = interpolate.splev(xnew,tck22_up,der=0)

#Dibujamos los elementos del Hamiltoniano
plt.plot(R_UHF,h11_uhf,'bo',ms=10)
plt.hold(True)
plt.plot(R_UHF,h12_uhf,'go',ms=10)
plt.plot(R_UHF,h21_uhf,'ro',ms=10)
plt.plot(R_UHF,h22_uhf,'yo',ms=10)
plt.plot(xnew,h11_up,'b',linewidth = 3, label='h11 UP')
plt.hold(True)
plt.plot(xnew,h12_up,'g',linewidth = 3, label='h12 UP')
plt.plot(xnew,h21_up,'r',linewidth = 3, label='h21 UP')
plt.plot(xnew,h22_up,'y',linewidth = 3, label='h22 UP')

#Leyenda
plt.legend(loc=0,prop={'size':36})
```

```
#Titulo
plt.title('Elementos del hamiltoniano UHF (UP)', fontsize = 40, color = '
    darkorange', verticalalignment = 'baseline', horizontalalignment = 'center')

#Etiquetas de los ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/ eV $', fontsize = 35, color = (0,0,0))

#Tamaño de los ejes
plt.yticks(size=20)
plt.xticks(size=20)

#Ponemos la rejilla
plt.grid(True)

#------------------------------------------------- Hamiltoniano down
plt.figure('Elementos del hamiltoniano UHF (DOWN)')

R_UHF, h11_uhf_dn = np.loadtxt('h_22_uhf.txt', usecols=(0,1), unpack=True)
h12_uhf_dn = np.loadtxt('h_12_uhf.txt', usecols=(1,), unpack=True)
h21_uhf_dn = np.loadtxt('h_21_uhf.txt', usecols=(1,), unpack=True)
h22_uhf_dn = np.loadtxt('h_11_uhf.txt', usecols=(1,), unpack=True)

#Rehacemos la lista de los elementos h11 y h22
for i in range(1,3) :
        h11_uhf_dn.put(i-1, h22_rhf[i-1])
        h22_uhf_dn.put(i-1, h11_rhf[i-1])
        h12_uhf_dn.put(i-1, h12_rhf[i-1])
        h21_uhf_dn.put(i-1, h21_rhf[i-1])


#Spline
tck11_dn = interpolate.splrep(R_UHF,h11_uhf_dn,s=0)
tck12_dn = interpolate.splrep(R_UHF,h12_uhf_dn,s=0)
tck21_dn = interpolate.splrep(R_UHF,h21_uhf_dn,s=0)
tck22_dn = interpolate.splrep(R_UHF,h22_uhf_dn,s=0)
h11_dn = interpolate.splev(xnew,tck11_dn,der=0)
h12_dn = interpolate.splev(xnew,tck12_dn,der=0)
h21_dn = interpolate.splev(xnew,tck21_dn,der=0)
h22_dn = interpolate.splev(xnew,tck22_dn,der=0)

#Dibujamos los elementos del Hamiltoniano
plt.plot(R_UHF,h11_uhf_dn,'bo',ms=10)
plt.hold(True)
plt.plot(R_UHF,h12_uhf_dn,'go',ms=10)
plt.plot(R_UHF,h21_uhf_dn,'ro',ms=10)
plt.plot(R_UHF,h22_uhf_dn,'yo',ms=10)
plt.plot(xnew,h11_dn,'b',linewidth = 3, label='h11_dn')
plt.hold(True)
plt.plot(xnew,h12_dn,'g',linewidth = 3, label='h12_dn')
plt.plot(xnew,h21_dn,'r',linewidth = 3, label='h21_dn')
plt.plot(xnew,h22_dn,'y',linewidth = 3, label='h22_dn')

#Leyenda
plt.legend(loc=0,prop={'size':36})

#Titulo
plt.title('Elementos del hamiltoniano (UHF) Down', fontsize = 40, color = '
    darkorange', verticalalignment = 'baseline', horizontalalignment = 'center')
```

```
#Etiquetas de los ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/ eV $', fontsize = 35, color = (0,0,0))

#Tamaño de los ejes
plt.yticks(size=20)
plt.xticks(size=20)

#Ponemos la rejilla
plt.grid(True)

############################################################ Energia UHF
plt.figure('Energia UHF')

#Sacamos los datos
r_uhf, e_uhf = np.loadtxt('energiaUHF.txt', usecols=(0,1), unpack=True)
#Puntos de Primeros principios
plt.plot(r_uhf,e_uhf,'ro',ms=10)
#Spline
tck1 = interpolate.splrep(r_uhf,e_uhf,s=0)
e_UHF = interpolate.splev(xnew1,tck1,der=0)
#Derivada del spline (FUERZA RHF)
F_UHF = interpolate.splev(xnew,tck1,der=1)
#Dibujamos la energia
plt.plot(xnew1, e_UHF,'g', linewidth = 3, label='E_UHF')
#Ponemos la rejilla
plt.grid(True)
#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/ eV $', fontsize = 35, color = (0,0,0))
#Titulo
plt.title('Energia UHF', fontsize = 40, color = 'darkorange', verticalalignment
    = 'baseline', horizontalalignment = 'center')
#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)
#Leyenda
plt.legend(loc =0,prop={'size':36})

############################################################ FUERZA UHF
plt.figure('Fuerzas')

#Dibujamos fuerza UHF
plt.plot(xnew, -F_UHF, 'r:', linewidth = 3, label = 'Fuerza UHF')
#Ponemos la rejilla
plt.grid(True)
#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$F/ eV/\AA $', fontsize = 35, color = (0,0,0))
#Titulo
plt.title('Fuerzas', fontsize = 40, color = 'darkorange', verticalalignment = '
    baseline', horizontalalignment = 'center')
#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)
#Leyenda
plt.legend(loc =0,prop={'size':36})

############################################## FUERZA + CORRECCION
```

```
#Leemos los valores de la matriz densidad UHF Y RHF
r_r, d11_rhf = np.loadtxt('d_11_rhf_up.txt', usecols=(0,1), unpack=True)
d12_rhf = np.loadtxt('d_12_rhf_up.txt', usecols=(1,), unpack=True)
d21_rhf = np.loadtxt('d_21_rhf_up.txt', usecols=(1,), unpack=True)
d22_rhf = np.loadtxt('d_22_rhf_up.txt', usecols=(1,), unpack=True)

r_u, d11_uhf_up = np.loadtxt('d_11_uhf.txt', usecols=(0,1), unpack=True)
d12_uhf_up = np.loadtxt('d_12_uhf.txt', usecols=(1,), unpack=True)
d21_uhf_up = np.loadtxt('d_21_uhf.txt', usecols=(1,), unpack=True)
d22_uhf_up = np.loadtxt('d_22_uhf.txt', usecols=(1,), unpack=True)

d11_uhf_dn = np.loadtxt('d_11_uhf_d.txt', usecols=(1,), unpack=True)
d12_uhf_dn = np.loadtxt('d_12_uhf_d.txt', usecols=(1,), unpack=True)
d21_uhf_dn = np.loadtxt('d_21_uhf_d.txt', usecols=(1,), unpack=True)
d22_uhf_dn = np.loadtxt('d_22_uhf_d.txt', usecols=(1,), unpack=True)

#Interpolamos los valores de la matriz densidad
tck11r = interpolate.splrep(r_r,d11_rhf,s=0)
tck12r = interpolate.splrep(r_r,d12_rhf,s=0)
tck21r = interpolate.splrep(r_r,d21_rhf,s=0)
tck22r = interpolate.splrep(r_r,d22_rhf,s=0)

tck11up = interpolate.splrep(r_u,d11_uhf_up,s=0)
tck12up = interpolate.splrep(r_u,d12_uhf_up,s=0)
tck21up = interpolate.splrep(r_u,d21_uhf_up,s=0)
tck22up = interpolate.splrep(r_u,d22_uhf_up,s=0)

tck11dn = interpolate.splrep(r_u,d11_uhf_dn,s=0)
tck12dn = interpolate.splrep(r_u,d12_uhf_dn,s=0)
tck21dn = interpolate.splrep(r_u,d21_uhf_dn,s=0)
tck22dn = interpolate.splrep(r_u,d22_uhf_dn,s=0)

d_11_rhf = interpolate.splev(xnew,tck11r,der=0)
d_12_rhf = interpolate.splev(xnew,tck12r,der=0)
d_21_rhf = interpolate.splev(xnew,tck21r,der=0)
d_22_rhf = interpolate.splev(xnew,tck22r,der=0)

d_11_uhf_up = interpolate.splev(xnew,tck11up,der=0)
d_12_uhf_up = interpolate.splev(xnew,tck12up,der=0)
d_21_uhf_up = interpolate.splev(xnew,tck21up,der=0)
d_22_uhf_up = interpolate.splev(xnew,tck22up,der=0)

d_11_uhf_dn = interpolate.splev(xnew,tck11dn,der=0)
d_12_uhf_dn = interpolate.splev(xnew,tck12dn,der=0)
d_21_uhf_dn = interpolate.splev(xnew,tck21dn,der=0)
d_22_uhf_dn = interpolate.splev(xnew,tck22dn,der=0)

D_11_up = []
D_12_up = []
D_21_up = []
D_22_up = []
D_11_dn = []
D_12_dn = []
D_21_dn = []
D_22_dn = []
#Calculamos los valores de las matrices de deformacion
for i in range (1,len(xnew)+1):
        D_11_up.append(d_11_uhf_up[i-1] - d_11_rhf[i-1])
        D_12_up.append(d_12_uhf_up[i-1] - d_12_rhf[i-1])
        D_21_up.append(d_21_uhf_up[i-1] - d_21_rhf[i-1])
```

```
        D_22_up.append(d_22_uhf_up[i-1] - d_22_rhf[i-1])

        D_11_dn.append(d_11_uhf_dn[i-1] - d_11_rhf[i-1])
        D_12_dn.append(d_12_uhf_dn[i-1] - d_12_rhf[i-1])
        D_21_dn.append(d_21_uhf_dn[i-1] - d_21_rhf[i-1])
        D_22_dn.append(d_22_uhf_dn[i-1] - d_22_rhf[i-1])


total = []
correcciones = []
for i in range(1,len(xnew)+1):
        up = D_11_up[i-1]*der_h11[i-1] + D_12_up[i-1]*der_h12[i-1] + D_21_up[i
            -1]*der_h21[i-1] + D_22_up[i-1]*der_h22[i-1]
        dn = D_11_dn[i-1]*der_h11[i-1] + D_12_dn[i-1]*der_h12[i-1] + D_21_dn[i
            -1]*der_h21[i-1] + D_22_dn[i-1]*der_h22[i-1]
        correcciones.append(up + dn)
        total.append(-np.array(correcciones[i-1]) - F_RHF[i-1])

plt.figure('Fuerzas')

#Dibujamos correcciones
plt.plot(xnew, -np.array(correcciones), 'y-', linewidth = 3, label = '
    Correcciones')
plt.plot(xnew, total, 'g--', linewidth = 3, label = 'F RHF + Correcciones' )
#Ponemos la rejilla
plt.grid(True)
#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$F/ eV/\AA $', fontsize = 35, color = (0,0,0))
#Titulo
plt.title('Fuerzas', fontsize = 40, color = 'darkorange', verticalalignment = '
    baseline', horizontalalignment = 'center')
#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)
#Leyenda
plt.legend(loc =0,prop={'size':36})

########################################################## Bandas UHF UP

#Obtenemos datos
r_U, orb1_UHF = np.loadtxt('orb1UHF.txt', usecols=(0,1), unpack=True)
orb2_UHF = np.loadtxt('orb2UHF.txt', usecols=(1,), unpack=True)
orb3_UHF = np.loadtxt('orb3UHF.txt', usecols=(1,), unpack=True)
orb4_UHF = np.loadtxt('orb4UHF.txt', usecols=(1,), unpack=True)
orb5_UHF = np.loadtxt('orb5UHF.txt', usecols=(1,), unpack=True)
orb6_UHF = np.loadtxt('orb6UHF.txt', usecols=(1,), unpack=True)
orb7_UHF = np.loadtxt('orb7UHF.txt', usecols=(1,), unpack=True)
orb8_UHF = np.loadtxt('orb8UHF.txt', usecols=(1,), unpack=True)
orb9_UHF = np.loadtxt('orb9UHF.txt', usecols=(1,), unpack=True)
orb10_UHF = np.loadtxt('orb10UHF.txt', usecols=(1,), unpack=True)
orb11_UHF = np.loadtxt('orb11UHF.txt', usecols=(1,), unpack=True)
orb12_UHF = np.loadtxt('orb12UHF.txt', usecols=(1,), unpack=True)
orb13_UHF = np.loadtxt('orb13UHF.txt', usecols=(1,), unpack=True)
orb14_UHF = np.loadtxt('orb14UHF.txt', usecols=(1,), unpack=True)
orb15_UHF = np.loadtxt('orb15UHF.txt', usecols=(1,), unpack=True)
orb16_UHF = np.loadtxt('orb16UHF.txt', usecols=(1,), unpack=True)
orb17_UHF = np.loadtxt('orb17UHF.txt', usecols=(1,), unpack=True)
orb18_UHF = np.loadtxt('orb18UHF.txt', usecols=(1,), unpack=True)
orb19_UHF = np.loadtxt('orb19UHF.txt', usecols=(1,), unpack=True)
```

```
orb20_UHF = np.loadtxt('orb20UHF.txt', usecols=(1,), unpack=True)
orb21_UHF = np.loadtxt('orb21UHF.txt', usecols=(1,), unpack=True)
orb22_UHF = np.loadtxt('orb22UHF.txt', usecols=(1,), unpack=True)
orb23_UHF = np.loadtxt('orb23UHF.txt', usecols=(1,), unpack=True)
orb24_UHF = np.loadtxt('orb24UHF.txt', usecols=(1,), unpack=True)
orb25_UHF = np.loadtxt('orb25UHF.txt', usecols=(1,), unpack=True)
orb26_UHF = np.loadtxt('orb26UHF.txt', usecols=(1,), unpack=True)
orb27_UHF = np.loadtxt('orb27UHF.txt', usecols=(1,), unpack=True)
orb28_UHF = np.loadtxt('orb28UHF.txt', usecols=(1,), unpack=True)
orb29_UHF = np.loadtxt('orb29UHF.txt', usecols=(1,), unpack=True)
orb30_UHF = np.loadtxt('orb30UHF.txt', usecols=(1,), unpack=True)
orb31_UHF = np.loadtxt('orb31UHF.txt', usecols=(1,), unpack=True)
orb32_UHF = np.loadtxt('orb32UHF.txt', usecols=(1,), unpack=True)
orb33_UHF = np.loadtxt('orb33UHF.txt', usecols=(1,), unpack=True)
orb34_UHF = np.loadtxt('orb34UHF.txt', usecols=(1,), unpack=True)
orb35_UHF = np.loadtxt('orb35UHF.txt', usecols=(1,), unpack=True)
orb36_UHF = np.loadtxt('orb36UHF.txt', usecols=(1,), unpack=True)
orb37_UHF = np.loadtxt('orb37UHF.txt', usecols=(1,), unpack=True)
orb38_UHF = np.loadtxt('orb38UHF.txt', usecols=(1,), unpack=True)
orb39_UHF = np.loadtxt('orb39UHF.txt', usecols=(1,), unpack=True)
orb40_UHF = np.loadtxt('orb40UHF.txt', usecols=(1,), unpack=True)


orb1_UHF_dn = []
orb2_UHF_dn = []
orb3_UHF_dn = []
orb4_UHF_dn = []
orb5_UHF_dn = []
orb6_UHF_dn = []
orb7_UHF_dn = []
orb8_UHF_dn = []
orb9_UHF_dn = []
orb10_UHF_dn = []
orb11_UHF_dn = []
orb12_UHF_dn = []
orb13_UHF_dn = []
orb14_UHF_dn = []
orb15_UHF_dn = []
orb16_UHF_dn = []
orb17_UHF_dn = []
orb18_UHF_dn = []
orb19_UHF_dn = []
orb20_UHF_dn = []
orb21_UHF_dn = []
orb22_UHF_dn = []
orb23_UHF_dn = []
orb24_UHF_dn = []
orb25_UHF_dn = []
orb26_UHF_dn = []
orb27_UHF_dn = []
orb28_UHF_dn = []
orb29_UHF_dn = []
orb30_UHF_dn = []
orb31_UHF_dn = []
orb32_UHF_dn = []
orb33_UHF_dn = []
orb34_UHF_dn = []
orb35_UHF_dn = []
orb36_UHF_dn = []
orb37_UHF_dn = []
```

```
orb38_UHF_dn = []
orb39_UHF_dn = []
orb40_UHF_dn = []

orb1_UHF_up = []
orb2_UHF_up = []
orb3_UHF_up = []
orb4_UHF_up = []
orb5_UHF_up = []
orb6_UHF_up = []
orb7_UHF_up = []
orb8_UHF_up = []
orb9_UHF_up = []
orb10_UHF_up = []
orb11_UHF_up = []
orb12_UHF_up = []
orb13_UHF_up = []
orb14_UHF_up = []
orb15_UHF_up = []
orb16_UHF_up = []
orb17_UHF_up = []
orb18_UHF_up = []
orb19_UHF_up = []
orb20_UHF_up = []
orb21_UHF_up = []
orb22_UHF_up = []
orb23_UHF_up = []
orb24_UHF_up = []
orb25_UHF_up = []
orb26_UHF_up = []
orb27_UHF_up = []
orb28_UHF_up = []
orb29_UHF_up = []
orb30_UHF_up = []
orb31_UHF_up = []
orb32_UHF_up = []
orb33_UHF_up = []
orb34_UHF_up = []
orb35_UHF_up = []
orb36_UHF_up = []
orb37_UHF_up = []
orb38_UHF_up = []
orb39_UHF_up = []
orb40_UHF_up = []

r_UHF =[]
#Separamos las componentes de spin
#Sacamos los up
for i in range(1,len(r_U)+1):
        if i%2 == 0:
                orb1_UHF_dn.append( orb1_UHF[i-1] )
                orb2_UHF_dn.append( orb2_UHF[i-1] )
                orb3_UHF_dn.append( orb3_UHF[i-1] )
                orb4_UHF_dn.append( orb4_UHF[i-1] )
                orb5_UHF_dn.append( orb5_UHF[i-1] )
                orb6_UHF_dn.append( orb6_UHF[i-1] )
                orb7_UHF_dn.append( orb7_UHF[i-1] )
                orb8_UHF_dn.append( orb8_UHF[i-1] )
                orb9_UHF_dn.append( orb9_UHF[i-1] )
                orb10_UHF_dn.append( orb10_UHF[i-1] )
```

```
            orb11_UHF_dn.append( orb11_UHF[i-1] )
            orb12_UHF_dn.append( orb12_UHF[i-1] )
            orb13_UHF_dn.append( orb13_UHF[i-1] )
            orb14_UHF_dn.append( orb14_UHF[i-1] )
            orb15_UHF_dn.append( orb15_UHF[i-1] )
            orb16_UHF_dn.append( orb16_UHF[i-1] )
            orb17_UHF_dn.append( orb17_UHF[i-1] )
            orb18_UHF_dn.append( orb18_UHF[i-1] )
            orb19_UHF_dn.append( orb19_UHF[i-1] )
            orb20_UHF_dn.append( orb20_UHF[i-1] )
            orb21_UHF_dn.append( orb21_UHF[i-1] )
            orb22_UHF_dn.append( orb22_UHF[i-1] )
            orb23_UHF_dn.append( orb23_UHF[i-1] )
            orb24_UHF_dn.append( orb24_UHF[i-1] )
            orb25_UHF_dn.append( orb25_UHF[i-1] )
            orb26_UHF_dn.append( orb26_UHF[i-1] )
            orb27_UHF_dn.append( orb27_UHF[i-1] )
            orb28_UHF_dn.append( orb28_UHF[i-1] )
            orb29_UHF_dn.append( orb29_UHF[i-1] )
            orb30_UHF_dn.append( orb30_UHF[i-1] )
            orb31_UHF_dn.append( orb31_UHF[i-1] )
            orb32_UHF_dn.append( orb32_UHF[i-1] )
            orb33_UHF_dn.append( orb33_UHF[i-1] )
            orb34_UHF_dn.append( orb34_UHF[i-1] )
            orb35_UHF_dn.append( orb35_UHF[i-1] )
            orb36_UHF_dn.append( orb36_UHF[i-1] )
            orb37_UHF_dn.append( orb37_UHF[i-1] )
            orb38_UHF_dn.append( orb38_UHF[i-1] )
            orb39_UHF_dn.append( orb39_UHF[i-1] )
            orb40_UHF_dn.append( orb40_UHF[i-1] )

            r_UHF.append( r_U[i-1] )
    else :
            orb1_UHF_up.append( orb1_UHF[i-1] )
            orb2_UHF_up.append( orb2_UHF[i-1] )
            orb3_UHF_up.append( orb3_UHF[i-1] )
            orb4_UHF_up.append( orb4_UHF[i-1] )
            orb5_UHF_up.append( orb5_UHF[i-1] )
            orb6_UHF_up.append( orb6_UHF[i-1] )
            orb7_UHF_up.append( orb7_UHF[i-1] )
            orb8_UHF_up.append( orb8_UHF[i-1] )
            orb9_UHF_up.append( orb9_UHF[i-1] )
            orb10_UHF_up.append( orb10_UHF[i-1] )
            orb11_UHF_up.append( orb11_UHF[i-1] )
            orb12_UHF_up.append( orb12_UHF[i-1] )
            orb13_UHF_up.append( orb13_UHF[i-1] )
            orb14_UHF_up.append( orb14_UHF[i-1] )
            orb15_UHF_up.append( orb15_UHF[i-1] )
            orb16_UHF_up.append( orb16_UHF[i-1] )
            orb17_UHF_up.append( orb17_UHF[i-1] )
            orb18_UHF_up.append( orb18_UHF[i-1] )
            orb19_UHF_up.append( orb19_UHF[i-1] )
            orb20_UHF_up.append( orb20_UHF[i-1] )
            orb21_UHF_up.append( orb21_UHF[i-1] )
            orb22_UHF_up.append( orb22_UHF[i-1] )
            orb23_UHF_up.append( orb23_UHF[i-1] )
            orb24_UHF_up.append( orb24_UHF[i-1] )
            orb25_UHF_up.append( orb25_UHF[i-1] )
            orb26_UHF_up.append( orb26_UHF[i-1] )
            orb27_UHF_up.append( orb27_UHF[i-1] )
```

```
                orb28_UHF_up.append( orb28_UHF[i-1] )
                orb29_UHF_up.append( orb29_UHF[i-1] )
                orb30_UHF_up.append( orb30_UHF[i-1] )
                orb31_UHF_up.append( orb31_UHF[i-1] )
                orb32_UHF_up.append( orb32_UHF[i-1] )
                orb33_UHF_up.append( orb33_UHF[i-1] )
                orb34_UHF_up.append( orb34_UHF[i-1] )
                orb35_UHF_up.append( orb35_UHF[i-1] )
                orb36_UHF_up.append( orb36_UHF[i-1] )
                orb37_UHF_up.append( orb37_UHF[i-1] )
                orb38_UHF_up.append( orb38_UHF[i-1] )
                orb39_UHF_up.append( orb39_UHF[i-1] )
                orb40_UHF_up.append( orb40_UHF[i-1] )


plt.figure('Energia de las bandas (UHF) UP')
plt.plot(r_UHF, orb1_UHF_up, 'ro', ms=10, )
plt.plot(r_UHF, orb2_UHF_up, 'go', ms=10, )
plt.plot(r_UHF, orb3_UHF_up, 'bo', ms=10, )
plt.plot(r_UHF, orb4_UHF_up, 'ko', ms=10, )
plt.plot(r_UHF, orb5_UHF_up, 'mo', ms=10, )
plt.plot(r_UHF, orb6_UHF_up, 'co', ms=10, )
plt.plot(r_UHF, orb7_UHF_up, 'yo', ms=10, )
plt.plot(r_UHF, orb8_UHF_up,  color = 'darkorange', marker = 'o', ms =10)
plt.plot(r_UHF, orb9_UHF_up,  color = 'sienna', marker = 'o', ms =10)
plt.plot(r_UHF, orb10_UHF_up, color = 'olivedrab', marker = 'o', ms =10)
plt.plot(r_UHF, orb11_UHF_up, color = 'dodgerblue', marker = 'o', ms =10)
plt.plot(r_UHF, orb12_UHF_up, color = 'mediumorchid', marker = 'o', ms =10)
plt.plot(r_UHF, orb13_UHF_up, color = 'salmon', marker = 'o', ms =10)
plt.plot(r_UHF, orb14_UHF_up, color = 'maroon', marker = 'o', ms =10)
plt.plot(r_UHF, orb15_UHF_up, color = 'plum', marker = 'o', ms =10)
plt.plot(r_UHF, orb16_UHF_up, color = 'thistle', marker = 'o', ms =10)
plt.plot(r_UHF, orb17_UHF_up, 'rs', ms=10, )
plt.plot(r_UHF, orb18_UHF_up, 'gs', ms=10, )
plt.plot(r_UHF, orb19_UHF_up, 'bs', ms=10, )
plt.plot(r_UHF, orb20_UHF_up, 'ks', ms=10, )
plt.plot(r_UHF, orb21_UHF_up, 'ms', ms=10, )
plt.plot(r_UHF, orb22_UHF_up, 'cs', ms=10, )
plt.plot(r_UHF, orb23_UHF_up, 'ys', ms=10, )
plt.plot(r_UHF, orb24_UHF_up, color = 'darkorange', marker = 's', ms =10)
plt.plot(r_UHF, orb25_UHF_up, color = 'sienna', marker = 's', ms =10)
plt.plot(r_UHF, orb26_UHF_up, color = 'olivedrab', marker = 's', ms =10)
plt.plot(r_UHF, orb27_UHF_up, color = 'dodgerblue', marker = 's', ms =10)
plt.plot(r_UHF, orb28_UHF_up, color = 'mediumorchid', marker = 's', ms =10)
plt.plot(r_UHF, orb29_UHF_up, color = 'salmon', marker = 's', ms =10)
plt.plot(r_UHF, orb30_UHF_up, color = 'maroon', marker = 's', ms =10)
plt.plot(r_UHF, orb31_UHF_up, color = 'plum', marker = 's', ms =10)
plt.plot(r_UHF, orb32_UHF_up, color = 'thistle', marker = 's', ms =10)
plt.plot(r_UHF, orb33_UHF_up, 'rs', ms=10, )
plt.plot(r_UHF, orb34_UHF_up, 'gs', ms=10, )
plt.plot(r_UHF, orb35_UHF_up, 'bs', ms=10, )
plt.plot(r_UHF, orb36_UHF_up, 'ks', ms=10, )
plt.plot(r_UHF, orb37_UHF_up, 'ms', ms=10, )
plt.plot(r_UHF, orb38_UHF_up, 'cs', ms=10, )
plt.plot(r_UHF, orb39_UHF_up, 'ys', ms=10, )
plt.plot(r_UHF, orb40_UHF_up, color = 'darkorange', marker = 's', ms =10)


plt.plot(r_UHF, orb1_UHF_up, 'r--', linewidth = 3, label = 'orb1')
plt.plot(r_UHF, orb2_UHF_up, 'g--', linewidth = 3, label = 'orb2')
```

```
plt.plot(r_UHF, orb3_UHF_up, 'b—', linewidth = 3, label = 'orb3')
plt.plot(r_UHF, orb4_UHF_up, 'k—', linewidth = 3, label = 'orb4')
plt.plot(r_UHF, orb5_UHF_up, 'm—', linewidth = 3, label = 'orb5')
plt.plot(r_UHF, orb6_UHF_up, 'c—', linewidth = 3, label = 'orb6')
plt.plot(r_UHF, orb7_UHF_up, 'y—', linewidth = 3, label = 'orb7')
plt.plot(r_UHF, orb8_UHF_up, color = 'darkorange', linestyle = '—' , linewidth
    = 3, label = 'orb8')
plt.plot(r_UHF, orb9_UHF_up, color = 'sienna', linestyle = '—' , linewidth = 3,
     label = 'orb9')
plt.plot(r_UHF, orb10_UHF_up, color = 'olivedrab', linestyle = '—' , linewidth
    = 3, label = 'orb10')
plt.plot(r_UHF, orb11_UHF_up, color = 'dodgerblue', linestyle = '—' , linewidth
     = 3, label = 'orb11')
plt.plot(r_UHF, orb12_UHF_up, color = 'mediumorchid', linestyle = '—' ,
    linewidth = 3, label = 'orb12')
plt.plot(r_UHF, orb13_UHF_up, color = 'salmon', linestyle = '—' , linewidth =
    3, label = 'orb13')
plt.plot(r_UHF, orb14_UHF_up, color = 'maroon', linestyle = '—' , linewidth =
    3, label = 'orb14')
plt.plot(r_UHF, orb15_UHF_up, color = 'plum', linestyle = '—' , linewidth = 3,
    label = 'orb15')
plt.plot(r_UHF, orb16_UHF_up, color = 'thistle', linestyle = '—' , linewidth =
    3, label = 'orb16')
plt.plot(r_UHF, orb17_UHF_up, 'r—', linewidth = 3, label = 'orb17')
plt.plot(r_UHF, orb18_UHF_up, 'g—', linewidth = 3, label = 'orb18')
plt.plot(r_UHF, orb19_UHF_up, 'b—', linewidth = 3, label = 'orb19')
plt.plot(r_UHF, orb20_UHF_up, 'k—', linewidth = 3, label = 'orb20')
plt.plot(r_UHF, orb21_UHF_up, 'm—', linewidth = 3, label = 'orb21')
plt.plot(r_UHF, orb22_UHF_up, 'c—', linewidth = 3, label = 'orb22')
plt.plot(r_UHF, orb23_UHF_up, 'y—', linewidth = 3, label = 'orb23')
plt.plot(r_UHF, orb24_UHF_up, color = 'darkorange', linestyle = '—' , linewidth
     = 3, label = 'orb24')
plt.plot(r_UHF, orb25_UHF_up, color = 'sienna', linestyle = '—' , linewidth =
    3, label = 'orb25')
plt.plot(r_UHF, orb26_UHF_up, color = 'olivedrab', linestyle = '—' , linewidth
    = 3, label = 'orb26')
plt.plot(r_UHF, orb27_UHF_up, color = 'dodgerblue', linestyle = '—' , linewidth
     = 3, label = 'orb27')
plt.plot(r_UHF, orb28_UHF_up, color = 'mediumorchid', linestyle = '—' ,
    linewidth = 3, label = 'orb28')
plt.plot(r_UHF, orb29_UHF_up, color = 'salmon', linestyle = '—' , linewidth =
    3, label = 'orb29')
plt.plot(r_UHF, orb30_UHF_up, color = 'maroon', linestyle = '—' , linewidth =
    3, label = 'orb30')
plt.plot(r_UHF, orb31_UHF_up, color = 'plum', linestyle = '—' , linewidth = 3,
    label = 'orb31')
plt.plot(r_UHF, orb32_UHF_up, color = 'thistle', linestyle = '—' , linewidth =
    3, label = 'orb32')
plt.plot(r_UHF, orb33_UHF_up, 'r—', linewidth = 3, label = 'orb33')
plt.plot(r_UHF, orb34_UHF_up, 'g—', linewidth = 3, label = 'orb34')
plt.plot(r_UHF, orb35_UHF_up, 'b—', linewidth = 3, label = 'orb35')
plt.plot(r_UHF, orb36_UHF_up, 'k—', linewidth = 3, label = 'orb36')
plt.plot(r_UHF, orb37_UHF_up, 'm—', linewidth = 3, label = 'orb37')
plt.plot(r_UHF, orb38_UHF_up, 'c—', linewidth = 3, label = 'orb38')
plt.plot(r_UHF, orb39_UHF_up, 'y—', linewidth = 3, label = 'orb39')
plt.plot(r_UHF, orb40_UHF_up, color = 'darkorange', linestyle = '—' , linewidth
     = 3, label = 'orb40')


#Ponemos la rejilla
```

```
plt.grid(True)
#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/eV$', fontsize = 35, color = (0,0,0))
#Titulo
plt.title('Energia de la bandas UHF', fontsize = 40, color = 'darkorange',
    verticalalignment = 'baseline', horizontalalignment = 'center')
#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)


plt.figure('Energia de las bandas (UHF) DOWN')
plt.plot(r_UHF, orb1_UHF_dn, 'r—', linewidth = 3, label = 'orb1')
plt.plot(r_UHF, orb2_UHF_dn, 'g—', linewidth = 3, label = 'orb2')
plt.plot(r_UHF, orb3_UHF_dn, 'b—', linewidth = 3, label = 'orb3')
plt.plot(r_UHF, orb4_UHF_dn, 'k—', linewidth = 3, label = 'orb4')
plt.plot(r_UHF, orb5_UHF_dn, 'm—', linewidth = 3, label = 'orb5')
plt.plot(r_UHF, orb6_UHF_dn, 'c—', linewidth = 3, label = 'orb6')
plt.plot(r_UHF, orb7_UHF_dn, 'y—', linewidth = 3, label = 'orb7')
plt.plot(r_UHF, orb8_UHF_dn, color = 'darkorange', linestyle = '—' , linewidth
    = 3, label = 'orb8')
plt.plot(r_UHF, orb9_UHF_dn, color = 'sienna', linestyle = '—' , linewidth = 3,
     label = 'orb9')
plt.plot(r_UHF, orb10_UHF_dn, color = 'olivedrab', linestyle = '—' , linewidth
    = 3, label = 'orb10')
plt.plot(r_UHF, orb11_UHF_dn, color = 'dodgerblue', linestyle = '—' , linewidth
     = 3, label = 'orb11')
plt.plot(r_UHF, orb12_UHF_dn, color = 'mediumorchid', linestyle = '—' ,
    linewidth = 3, label = 'orb12')
plt.plot(r_UHF, orb13_UHF_dn, color = 'salmon', linestyle = '—' , linewidth =
    3, label = 'orb13')
plt.plot(r_UHF, orb14_UHF_dn, color = 'maroon', linestyle = '—' , linewidth =
    3, label = 'orb14')
plt.plot(r_UHF, orb15_UHF_dn, color = 'plum', linestyle = '—' , linewidth = 3,
    label = 'orb15')
plt.plot(r_UHF, orb16_UHF_dn, color = 'thistle', linestyle = '—' , linewidth =
    3, label = 'orb16')
plt.plot(r_UHF, orb17_UHF_dn, 'r—', linewidth = 3, label = 'orb17')
plt.plot(r_UHF, orb18_UHF_dn, 'g—', linewidth = 3, label = 'orb18')
plt.plot(r_UHF, orb19_UHF_dn, 'b—', linewidth = 3, label = 'orb19')
plt.plot(r_UHF, orb20_UHF_dn, 'k—', linewidth = 3, label = 'orb20')
plt.plot(r_UHF, orb21_UHF_dn, 'm—', linewidth = 3, label = 'orb21')
plt.plot(r_UHF, orb22_UHF_dn, 'c—', linewidth = 3, label = 'orb22')
plt.plot(r_UHF, orb23_UHF_dn, 'y—', linewidth = 3, label = 'orb23')
plt.plot(r_UHF, orb24_UHF_dn, color = 'darkorange', linestyle = '—' , linewidth
     = 3, label = 'orb24')
plt.plot(r_UHF, orb25_UHF_dn, color = 'sienna', linestyle = '—' , linewidth =
    3, label = 'orb25')
plt.plot(r_UHF, orb26_UHF_dn, color = 'olivedrab', linestyle = '—' , linewidth
    = 3, label = 'orb26')
plt.plot(r_UHF, orb27_UHF_dn, color = 'dodgerblue', linestyle = '—' , linewidth
     = 3, label = 'orb27')
plt.plot(r_UHF, orb28_UHF_dn, color = 'mediumorchid', linestyle = '—' ,
    linewidth = 3, label = 'orb28')
plt.plot(r_UHF, orb29_UHF_dn, color = 'salmon', linestyle = '—' , linewidth =
    3, label = 'orb29')
plt.plot(r_UHF, orb30_UHF_dn, color = 'maroon', linestyle = '—' , linewidth =
    3, label = 'orb30')
plt.plot(r_UHF, orb31_UHF_dn, color = 'plum', linestyle = '—' , linewidth = 3,
```

```
    label = 'orb31')
plt.plot(r_UHF, orb32_UHF_dn, color = 'thistle', linestyle = '--' , linewidth =
    3, label = 'orb32')
plt.plot(r_UHF, orb33_UHF_dn, 'r--', linewidth = 3, label = 'orb33')
plt.plot(r_UHF, orb34_UHF_dn, 'g--', linewidth = 3, label = 'orb34')
plt.plot(r_UHF, orb35_UHF_dn, 'b--', linewidth = 3, label = 'orb35')
plt.plot(r_UHF, orb36_UHF_dn, 'k--', linewidth = 3, label = 'orb36')
plt.plot(r_UHF, orb37_UHF_dn, 'm--', linewidth = 3, label = 'orb37')
plt.plot(r_UHF, orb38_UHF_dn, 'c--', linewidth = 3, label = 'orb38')
plt.plot(r_UHF, orb39_UHF_dn, 'y--', linewidth = 3, label = 'orb39')
plt.plot(r_UHF, orb40_UHF_dn, color = 'darkorange', linestyle = '--' , linewidth
    = 3, label = 'orb40')


#Ponemos la rejilla
plt.grid(True)
#Etiquetas ejes
plt.xlabel('$R/ \AA $', fontsize = 35, color = (0,0,0))
plt.ylabel('$E/eV$', fontsize = 35, color = (0,0,0))
#Titulo
plt.title('Energia de la bandas UHF', fontsize = 40, color = 'darkorange',
    verticalalignment = 'baseline', horizontalalignment = 'center')
#Tamaño de ejes
plt.yticks(size=20)
plt.xticks(size=20)

plt.show()
```

Apéndice C

Código 3: Segundos principios

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

#Importamos la librería numpy y matplotlib.
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import interpolate


############ DATOS ##################

#Número de iteraciones.
n = 1000
#Paso temporal en la evolución de la matriz densidad.
t_i = 0.0
t_f = 10.0
#Paso temporal
h = (t_f - t_i)/n
#Vector que contiene los tiempos en los que evaluamos
#la matriz densidad.
t = np.arange(t_i,t_f,h)
#Variable que guardara el hamiltoniano
global h_t
#--Intensidad del campo
e_0 = 30
#--Instante de tiempo en el que se da el pulso
t_0 = 0
#Valor de la posicion de la distancia internuclear en el array
#que las contiene.
z=7
#FIN DE DATOS ------------------------------------------------


########## FUNCIONES ################

################## RECOGIDA DE VALORES DE LA MATRIZ DENSIDAD (OK)

def getDensidad():
        #Variables que guardan los valores de la matriz densidad (RHF)
        global d110_rhf, d120_rhf, d220_rhf, d210_rhf

        #Leemos los ficheros del hamiltoniano, previamente calculados
        d110_rhf = np.loadtxt('d_11_rhf.txt', usecols=(1,), unpack=True)
        d120_rhf = np.loadtxt('d_12_rhf.txt', usecols=(1,), unpack=True)
```

```
        d210_rhf = np.loadtxt('d_22_rhf.txt', usecols=(1,), unpack=True)
        d220_rhf = np.loadtxt('d_21_rhf.txt', usecols=(1,), unpack=True)
        #Variables que guardan la matriz densidad UHF (DOWN Y UP)
        global d110_uhf_u, d120_uhf_u, d220_uhf_u, d210_uhf_u, d110_uhf_d,
            d120_uhf_d, d220_uhf_d, d210_uhf_d

        #Leemos los ficheros del hamiltoniano, previamente calculados
        d110_uhf_u = np.loadtxt('d_11_uhf.txt', usecols=(1,), unpack=True)
        d120_uhf_u = np.loadtxt('d_12_uhf.txt', usecols=(1,), unpack=True)
        d210_uhf_u = np.loadtxt('d_22_uhf.txt', usecols=(1,), unpack=True)
        d220_uhf_u = np.loadtxt('d_21_uhf.txt', usecols=(1,), unpack=True)

        d110_uhf_d = np.loadtxt('d_11_uhf_d.txt', usecols=(1,), unpack=True)
        d120_uhf_d = np.loadtxt('d_12_uhf_d.txt', usecols=(1,), unpack=True)
        d210_uhf_d = np.loadtxt('d_22_uhf_d.txt', usecols=(1,), unpack=True)
        d220_uhf_d = np.loadtxt('d_21_uhf_d.txt', usecols=(1,), unpack=True)
#FIN DE RECOGIDA DE VALORES DE LA MATRIZ DENSIDAD-----------------

######################## RECOGIDA DE VALORES DEL HAMILTONIANO (OK)

def getHamil():
        #Variables que guardan los valores del hamiltoniano (RHF)
        global h_11_rhf, h_12_rhf, h_22_rhf, h_21_rhf, r_rhf

        #Leemos los ficheros del hamiltoniano, previamente calculados
        r_rhf, h_11_rhf = np.loadtxt('h_11_rhf.txt', usecols=(0,1), unpack=True)
        h_12_rhf = np.loadtxt('h_12_rhf.txt', usecols=(1,), unpack=True)
        h_22_rhf = np.loadtxt('h_22_rhf.txt', usecols=(1,), unpack=True)
        h_21_rhf = np.loadtxt('h_21_rhf.txt', usecols=(1,), unpack=True)
        #Variables que guardan los valores del hamiltoniano (UHF)
        global h_11_uhf, h_12_uhf, h_22_uhf, r_21_uhf

        #Leemos los ficheros del hamiltoniano, previamente calculados
        r_uhf, h_11_uhf = np.loadtxt('h_11_uhf.txt', usecols=(0,1), unpack=True)
        h_12_uhf = np.loadtxt('h_12_uhf.txt', usecols=(1,), unpack=True)
        h_22_uhf = np.loadtxt('h_22_uhf.txt', usecols=(1,), unpack=True)
        h_21_uhf = np.loadtxt('h_21_uhf.txt', usecols=(1,), unpack=True)

        hamil_0_rhf = []
        #Formamos la matriz con las posiciones
        for i in range(1,len(r_rhf)+1) :
                hamil_0_rhf.append( np.matrix( [[ h_11_rhf[i-1],h_12_rhf[i-1] ],
                    [ h_21_rhf[i-1],h_22_rhf[i-1] ]] ) )

        hamil_0_uhf = []
        #Formamos la matriz con las posiciones
        for i in range(1,len(r_uhf)+1) :
                hamil_0_uhf.append( np.matrix( [[ h_11_uhf[i-1],h_12_uhf[i-1] ],
                    [ h_21_uhf[i-1],h_22_uhf[i-1] ]] ) )

#FIN DE RECOGIDA DE VALORES DEL HAMILTONIANO-----------------------

#################################### RECOGIDA DE POSICIONES (OK!)

def getPos():

        #Variable que guardará las posiciones
        global r_11_uhf, r_12_uhf, r_21_uhf, r_22_uhf, r_ab_uhf, R_uhf, r_11_rhf
            , r_12_rhf, r_21_rhf, r_22_rhf, r_ab_rhf, R_rhf
```

```
        #Leemos el fichero que contiene las posiciones UHF
        r_11_uhf = np.loadtxt('r_11_uhf.txt', usecols=(1,), unpack=True)
        r_12_uhf = np.loadtxt('r_12_uhf.txt', usecols=(1,), unpack=True)
        r_21_uhf = np.loadtxt('r_21_uhf.txt', usecols=(1,), unpack=True)
        r_22_uhf = np.loadtxt('r_22_uhf.txt', usecols=(1,), unpack=True)

        r_ab_uhf = []
        #Formamos la matriz con las posiciones
        for i in range(1,len(r_11_uhf)+1) :
                r_ab_uhf.append( np.matrix( [[ r_11_uhf[i-1],r_12_uhf[i-1] ], [
                    r_21_uhf[i-1],r_22_uhf[i-1] ]] ) )


        #Leemos el fichero que contiene las posiciones RHF
        r_11_rhf = np.loadtxt('r_11_rhf.txt', usecols=(1,), unpack=True)
        r_12_rhf = np.loadtxt('r_12_rhf.txt', usecols=(1,), unpack=True)
        r_21_rhf = np.loadtxt('r_21_rhf.txt', usecols=(1,), unpack=True)
        r_22_rhf = np.loadtxt('r_22_rhf.txt', usecols=(1,), unpack=True)

        r_ab_rhf = []
        #Formamos la matriz con las posiciones
        for i in range(1,len(r_11_rhf)+1) :
                r_ab_rhf.append( np.matrix( [[ r_11_rhf[i-1],r_12_rhf[i-1] ], [
                    r_21_rhf[i-1],r_22_rhf[i-1] ]] ) )

#FIN DE RECOGIDA DE POSICIONES-----------------------------------

### MÉTODOS PARA LA EVOLUCIÓN TEMPORAL DE LA MATRIZ DENSIDAD ###

################################ MÉTODO DE DIFERENCIAS FINITAS (OK!)

def diferencias_finitas(dens_0, hamil_0):

        #Vector que contiene los resultados de diferencias finitas
        d = []

        #Valor inicial de la matriz densidad
        d.insert(0,dens_0)

        #Hacemos el cálculo del valor de la matriz densidad
        #en un instante posterior
        for i in np.arange(1,len(t)+1) :
                #Realizamos el pulso
                if ( t[i-1] == t_0 ) :
                        e = e_0
                        h_e = -e*r_ab_uhf[z]
                else :
                        e = 0
                        h_e = -e*r_ab_uhf[z]

                h_t = hamil_0 + h_e

                d.append( ( h_t*d[i-1] - d[i-1]*h_t )*h*(-1j) + d[i-1] )

        #Definimos las listas que contienen los elementos de la
        #matriz densidad.
        global d11, d12, d21, d22, d11
        d11 = []
        d12 = []
        d21 = []
```

```
        d22 = []

        #Dibujamos la matriz densidad en función del tiempo.
        for k in range(0,n) :
                d11.append(d[k][0,0])
                d12.append(d[k][0,1])
                d21.append(d[k][1,0])
                d22.append(d[k][1,1])

        d11 = np.array(d11)
        d12 = np.array(d12)
        d21 = np.array(d21)
        d22 = np.array(d22)

        #Dibujamos la matriz densidad en función del tiempo.

        plt.figure('Evolucion temporal de la matriz densidad - Diferencias
            Finitas')

        plt.plot(t, d11, 'r',linewidth = 3, label='d11')
        plt.hold(True)
        plt.plot(t, d12, 'b',linewidth = 3, label='d12')
        plt.plot(t, d21, 'g',linewidth = 3, label='d21')
        plt.plot(t, d22, 'k',linewidth = 3, label='d22')

        #Ponemos nombre a los ejes
        plt.xlabel('$t/ fs $', fontsize = 35, color = 'black')
        plt.ylabel('Ocupacion', fontsize = 35, color = 'black')
        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos el título
        plt.title("Evolucion de la matriz densidad - Diferencias Finitas",
            fontsize = 40, color = 'darkorange', verticalalignment = 'baseline',
            horizontalalignment = 'center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#FIN DEL MÉTODO DE LAS DIFERENCIAS FINITAS ----------------------

########################## CÁLCULO DEL MÉTODO RUNGE-KUTTA 4 (OK!)

def runge_kutta(dens_0,hamil_0):
        #Vector que contiene los resultados de RK4
        d_rk4 = []

        #Valor inicial de la matriz densidad
        d_rk4.insert(0,dens_0)

        for i in range(1,len(t)+1) :
        #Calculamos los pesos w1, w2, w3, w4.
                #Realizamos el pulso
                if ( t[i-1] == t_0 ) :
                        e = e_0
                        h_e = -e*r_ab_uhf[z]
```

```
            else :
                    e = 0
                    h_e = -e*r_ab_uhf[z]

            h_t = hamil_0 + h_e

            #Calculamos los pesos para Runge-Kutta 4
            w1 = h*( h_t*d_rk4[i-1] - d_rk4[i-1]*h_t )*(-1j)
            w2 = h*( h_t*(d_rk4[i-1]+w1/2) - (d_rk4[i-1]+w1/2)*h_t )*(-1j)
            w3 = h*( h_t*(d_rk4[i-1]+w2/2) - (d_rk4[i-1]+w2/2)*h_t )*(-1j)
            w4 = h*( h_t*(d_rk4[i-1]+w3) - (d_rk4[i-1]+w3)*h_t )*(-1j)

            #Calculamos el valor de la matriz densidad
            #en el siguiente punto.
            d_rk4.insert(i+1-1, d_rk4[i-1] + ( w1 + 2*w2 + 2*w3 + w4 )/6 )

    #Definimos las listas que contienen los elementos de
    #la matriz densidad.
    global d_rk4_11, d_rk4_12, d_rk4_21, d_rk4_22
    d_rk4_11 = []
    d_rk4_12 = []
    d_rk4_21 = []
    d_rk4_22 = []

    #Llenamos las listas anteriores con los elementos de matriz.
    for k in range(0,n) :
            d_rk4_11.append(d_rk4[k][0,0])
            d_rk4_12.append(d_rk4[k][0,1])
            d_rk4_21.append(d_rk4[k][1,0])
            d_rk4_22.append(d_rk4[k][1,1])

    d_rk4_11 = np.array(d_rk4_11)
    d_rk4_12 = np.array(d_rk4_12)
    d_rk4_21 = np.array(d_rk4_21)
    d_rk4_22 = np.array(d_rk4_22)


    #Dibujamos el resultado del método de Runge-Kutta 4.
    plt.figure('Evolucion temporal de la matriz densidad - Runge-Kutta 4')

    plt.plot(t,d_rk4_11,'r:', linewidth = 3, label='d11')
    plt.hold(True)
    plt.plot(t,d_rk4_12,'b:', linewidth = 3, label='d12')
    plt.plot(t,d_rk4_21,'g:', linewidth = 3, label='d21')
    plt.plot(t,d_rk4_22,'k:', linewidth = 3, label='d22')

    #Ponemos nombre a los ejes
    plt.xlabel('$t/ fs $', fontsize = 35, color = 'black')
    plt.ylabel('Ocupacion', fontsize = 35, color = ' black')
    #Ponemos la rejilla
    plt.grid(True)

    #Ponemos el título
    plt.title("Evolucion de la matriz densidad - Runge-Kutta 4", fontsize =
        40, color = 'darkorange', verticalalignment = 'baseline',
        horizontalalignment = 'center' )

    #Colocamos la leyenda
    plt.legend(loc=0,prop={'size':36})
```

```
        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#FIN DEL RUNGE-KUTTA 4 ----------------------------------------------

### PROPAGACIÓN MEDIANTE OPERADOR DE EVOLUCIÓN TEMPORAL U(T,T0)

def evolucion_temporal(dens_0, hamil_0):
        #Vector que contiene los resultados de ET.
        d_et = []
        d_et.insert(0, dens_0)

        #Realizamos la suma con 4 términos y para cada
        #instante de tiempo.
        for m in range(1,len(t)):
                #Realizamos el pulso
                if ( t[m-1] == t_0 ) :
                        e = e_0
                        h_e = -e*r_ab_uhf[z]
                else :
                        e = 0
                        h_e = -e*r_ab_uhf[z]

                h_t = hamil_0 + h_e

                #Primer término del desarrollo de taylor ( i = 0 )
                suma = np.identity(2)

                for i in range(1,4) :
                        suma = suma + ( ( (t[m]-t[m-1])*h_t*(-1)*(1j) )**i ) /
                            math.factorial(i)

                #Calculamos el valor de la matriz densidad
                d_et.append(suma*d_et[m-1]*(suma.H))

        #Definimos las listas que contienen los elementos de la
        #matriz densidad.
        global d_et11, d_et12, d_et21, d_et22
        d_et11 = []
        d_et12 = []
        d_et21 = []
        d_et22 = []


        #Llenamos las listas anteriores con los elementos de matriz.
        for k in range(0,n) :
                d_et11.append(d_et[k][0,0])
                d_et12.append(d_et[k][0,1])
                d_et21.append(d_et[k][1,0])
                d_et22.append(d_et[k][1,1])

        d_et11 = np.array(d_et11)
        d_et12 = np.array(d_et12)
        d_et21 = np.array(d_et21)
        d_et22 = np.array(d_et22)

        #Dibujamos el resultado del método del operador de evolución temporal.
        plt.figure('Evolucion temporal de la matriz densidad - Operador
            evolucion')
```

```
        plt.plot(t,d_et11, 'r--', linewidth = 3, label='d11')
        plt.hold(True)
        plt.plot(t,d_et12, 'b--', linewidth = 3, label='d12')
        plt.plot(t,d_et21, 'g--', linewidth = 3, label='d21')
        plt.plot(t,d_et22, 'k--', linewidth = 3, label='d22')

        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos nombre a los ejes
        plt.xlabel('$t/ fs$', fontsize = 35, color = (0,0,0))
        plt.ylabel('Ocupacion', fontsize = 35, color = (0,0,0))
        #Ponemos el título
        plt.title("Evolucion temporal de la matriz densidad - Operador evolucion
            ", fontsize = 40, color = 'darkorange', verticalalignment =     '
            baseline', horizontalalignment = 'center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#FIN PROPAGACIÓN OPERADOR DE EVOLUCIÓN TEMPORAL U(T,T0) ----------

################################### CÁLCULO DEL DIPOLO ELÉCTRICO

#Dipolo eléctrico obtenido mediante la propagacion
#por diferencias finitas
def dipolo_df():
        #Vector que guarda el valor del dipolo en cada
        #instante de tiempo
        global p_df_time
        p_df_time = []

        for i in range (0,len(t)) :
                p_df_time.append( d11[i]*r_11_uhf[z] + d12[i]*r_12_uhf[z] + d21[
                    i]*r_21_uhf[z] + d22[i]*r_22_uhf[z] )

        #Realizamos la TF real para pasar al espacio
        #recíproco ( frecuencias ) del dipolo
        global p_df_freq
        p_df_freq = []
        p_df_freq = np.fft.rfft(p_df_time)
        p_df_freq = np.array(p_df_freq)

        #Dibujamos el dipolo en el espacio de frecuencias
        plt.figure('Dipolo en el espacio de frecuencias - Diferencias Finitas')
        ax1 = plt.subplot(111)

        #Pasamos los valores del tiempo a frecuencias
        freq = np.array( range( len( p_df_freq ) ) ) / float( len( p_df_time ) )
             / h

        #Dibujamos el resultado del método del operador
        # de evolución temporal.
        plt.plot(freq,np.real(p_df_freq), 'r--', linewidth = 3, label='Dipolo
            parte real')
```

```
        plt.plot(freq,np.imag(p_df_freq), 'b—', linewidth = 3, label='Dipolo
            parte imaginaria')

        #Ponemos nombre a los ejes
        plt.xlabel('$f/ Hz $', fontsize = 35, color = (0,0,0))

        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos el título
        plt.title("Dipolo en el espacio de frecuencias-Diferencias Finitas",
            fontsize = 40, color = 'darkorange', verticalalignment = 'baseline',
            horizontalalignment = 'center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

        #Dibujamos el dipolo en el espacio real
        plt.figure('Dipolo en el espacio temporal-Diferencias Finitas')
        plt.plot(t,p_df_time,'r—', linewidth = 3, label='Dipolo')

        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos nombre a los ejes
        plt.xlabel('$t/s? $', fontsize = 35, color = (0,0,0))

        #Ponemos el título
        plt.title("Dipolo en el espacio temporal - Diferencias Finitas",
            fontsize = 40, color = 'darkorange', verticalalignment = 'baseline',
            horizontalalignment = 'center' )
        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#Dipolo eléctrico obtenido mediante la propagación por
#Runge-Kutta 4
def dipolo_rk4():
        #Vector que guarda el valor del dipolo en cada
        #instante de tiempo
        global p_rk4_time
        p_rk4_time = []

        for i in range (0,len(t)) :
                p_rk4_time.append( d_rk4_11[i]*r_11_uhf[z] +d_rk4_12[i]*r_12_uhf
                    [z] + d_rk4_21[i]*r_21_uhf[z] + d_rk4_22[i]*r_22_uhf[z] )

        #Realizamos la TF real para pasar al espacio
        #recíproco ( frecuencias )
        global p_rk4_freq
        p_rk4_freq = []
        p_rk4_freq = np.fft.rfft(p_rk4_time)
        p_rk4_freq = np.array(p_rk4_freq)
```

```
        #Dibujamos el dipolo en el espacio de frecuencias
        plt.figure('Dipolo en el espacio de frecuencias - Runge Kutta 4')
        ax1 = plt.subplot(111)

        #Pasamos los valores del tiempo a frecuencias
        freq = np.array( range( len( p_rk4_freq ) ) ) / float( len( p_rk4_time )
            ) / h

        #Dibujamos el resultado del método del operador
        #de evolución temporal.
        plt.plot(freq,np.real(p_rk4_freq), 'r--', linewidth = 3, label='Dipolo
            parte real')
        plt.plot(freq,np.imag(p_rk4_freq), 'b--', linewidth = 3, label='Dipolo
            parte imaginaria')

        #Ponemos nombre a los ejes
        plt.xlabel('$f/ Hz $', fontsize = 35, color = (0,0,0))

        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos el título
        plt.title("Dipolo en el espacio de frecuencias - Runge-Kutta 4",
            fontsize = 40, color = 'darkorange', verticalalignment = 'baseline',
            horizontalalignment = 'center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

        #Dibujamos el dipolo en el espacio real
        plt.figure('Dipolo en el espacio temporal - Runge Kutta 4')
        plt.plot(t,p_rk4_time,'r--', linewidth = 3, label='Dipolo')

        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos nombre a los ejes
        plt.xlabel('$t/s? $', fontsize = 35, color = (0,0,0))

        #Ponemos el título
        plt.title("Dipolo en el espacio temporal - Runge-Kutta 4", fontsize =
            40, color = 'darkorange', verticalalignment = 'baseline',
            horizontalalignment = 'center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#Dipolo eléctrico obtenido mediante la propagación mediante
#el uso del operador de evolución
def dipolo_et():
        #Vector que guarda el valor del dipolo en cada
```

```
#instante de tiempo
global p_et_time
p_et_time = []

for i in range (0,len(t)) :
        p_et_time.append( d_et11[i]*r_11_uhf[z] + d_et12[i]*r_12_uhf[z]
           + d_et21[i]*r_21_uhf[z] + d_et22[i]*r_22_uhf[z] )

#Realizamos la TF real para pasar al espacio
#recíproco ( frecuencias )
global p_et_freq
p_et_freq = []
p_et_freq = np.fft.rfft(p_et_time)
p_et_freq = np.array(p_et_freq)

#Dibujamos el dipolo en el espacio de frecuencias
plt.figure('Dipolo en el espacio de frecuencias - Operador evolucion')
ax1 = plt.subplot(111)

#Pasamos los valores del tiempo a frecuencias
freq = np.array( range( len( p_et_freq ) ) ) / float( len( p_et_time ) )
    / h

#Dibujamos el resultado del método del operador
#de evolución temporal.
plt.plot(freq,np.real(p_et_freq), 'r--', linewlidth = 3, label='Dipolo
   parte real')
plt.plot(freq,np.imag(p_et_freq), 'b--', linewidth = 3, label='Dipolo
   parte imaginaria')

#Ponemos nombre a los ejes
plt.xlabel('$f/ Hz $', fontsize = 35, color = (0,0,0))

#Ponemos la rejilla
plt.grid(True)

#Ponemos el título
plt.title("Dipolo en el espacio de frecuencias - Operador evolución",
   fontsize = 40, color = 'darkorange', verticalalignment = 'baseline',
   horizontalalignment = 'center' )

#Colocamos la leyenda
plt.legend(loc=0,prop={'size':36})

#Agrandamos los valores de los ejes
plt.yticks(size=20)
plt.xticks(size=20)

#Dibujamos el dipolo en el espacio real
plt.figure('Dipolo en el espacio temporal - Operador evolucion')
plt.plot(t,p_et_time,'r--', linewidth = 3, label='Dipolo')

#Ponemos la rejilla
plt.grid(True)

#Ponemos nombre a los ejes
plt.xlabel('$t/s? $', fontsize = 35, color = (0,0,0))

#Ponemos el título
plt.title("Dipolo en el espacio temporal - Operador evolucion", fontsize
```

```
            = 40, color = 'darkorange', verticalalignment = 'baseline',
            horizontalalignment = 'center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#FIN CÁLCULO DEL DIPOLO ELÉCTRICO ----------------------------------

################################################ VERLET INTEGRATION

def verletIntegration(pos_inicial, vel_inicial) :
        #Masa del sistema en unidades átomicas
        m =
        #Leemos los valores de las energias
        e_uhf = np.loadtxt('energiaUHF.txt', usecols=(1,), unpack=True)
        #Spline
        tck = interpolate.splrep(r_uhf,e_uhf,s=0)
        #Variable que guarda las posiciones
        x = []

        #Valores iniciales
        x.append( pos_inical )
        x.append( pos_inicial + vel_inicial*h + 0.5 * (interpolate.splev(
            pos_inicial,tck,der=1)/m) * h**2 )

        #Calculamos los valores de la posicion mediante Verlet, iterando
        for i in range(2, len(t)) :
                x.append( 2*x[i-1] - x[i-2] + (interpolate.splev(x[i-1],tck,der
                    =1)/m) * h**2 )


        #Dibujamos los valores de la posición con el tiempo.
        plt.figure('Posición en función del tiempo')
        plt.plot(t,x,'r--', linewidth = 3, label = 'Posicion')

        #Ponemos nombre a los ejes
        plt.xlabel('$f/ Hz $', fontsize = 35, color = (0,0,0))

        #Ponemos la rejilla
        plt.grid(True)

        #Ponemos el título
        plt.title("Posiciones en funcion del tiempo", fontsize = 40, color = '
            darkorange', verticalalignment = 'baseline', horizontalalignment = '
            center' )

        #Colocamos la leyenda
        plt.legend(loc=0,prop={'size':36})

        #Agrandamos los valores de los ejes
        plt.yticks(size=20)
        plt.xticks(size=20)

#FIN VERLET INTEGRATION ----------------------------------------------

#-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_- FIN FUNCIONES
```

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

############################## 1. Estructura general del programa

#Corriendo  previamente el código en "diag_hamiltoniano.py" se
#obtienen archivos con los valores de la matriz densidad.

#Leemos los archivos que contienen la densidad, posiciones
#y hamiltoniano. Formamos las matrices de la densidad y
#del hamiltoniano
getDensidad()
getPos()
getHamil()

#Una vez tenemos todos los datos, vemos como evoluciona la
#matriz densidad con el tiempo. Hay que elegir una configuración
#de los núcleos, para elegir adecuadamente la matriz densidad
#inicial y la distancia r_ab.

evolucion_temporal( dens_0, hamil_0 )

#Calculamos el dipolo.
dipolo_et()

#Realizamos la dinámica. Hay que elegir la posicion inicial
#( pos_inicial ) en la que comenzamos la dinámica y también
#hay que elegir la velocidad inicial ( vel_inicial ).
verletIntegration(pos_inicial, vel_inicial)

#Mostramos las figuras.
plt.show()
```