Time Independent 3D Schrödinger Equation in Python - the Hydrogen Atom

Nanometric System Simulation - Nanoscience and Nanotechnology UAB 2022/23

Xabier Oianguren Asua

1 The 3D Hydrogen Atom and Suitable Units

Given the Coulomb interaction potential for an electron at spatial coordinates $\vec{x} \equiv (x_1, x_2, x_3) \in \mathbb{R}^3$ relative to the proton,

$$V(\vec{x}) = -\frac{e^2}{4\pi\varepsilon_0\sqrt{x_1^2+x_2^2+x_3^2}}, \tag{1}$$

where e is the fundamental charge and ε_0 the vacuum permittivity, the time independent Schrödinger equation for the Hydrogen atom (around its center of mass) is [1]

$$\left(-\frac{\hbar^2}{2m}\sum_{k=1}^{3}\frac{\partial^2}{\partial x_k^2} + V(\vec{x})\right)\psi(\vec{x}) = E\psi(\vec{x}), \tag{2}$$

where $m_k = m$ is the mass of the electron $\forall k$, $\hbar$ is the Planck constant and $\psi(x_1, x_2, x_3) : \mathbb{R}^3 \to \mathbb{C}$ is the wavefunction of the electron in the $\vec{x}$ position.

If we define a constant with spatial units, so-called Bohr radious,

$$a_0 := \frac{4\pi\varepsilon\hbar^2}{me^2} \tag{3}$$

and a constant with energy units, called the Hartree energy,

$$E_H = \frac{me^4}{(4\pi\varepsilon_0)^2\hbar^2}, \tag{4}$$

we can define adimensional coordinate and energy variables, $\vec{q} \equiv (q_1, q_2, q_3)$ and E' respectively, related to the ones in equation (2) as

$$q_j = \frac{x_j}{a_0} \quad j \in \{1,2,3\}; \quad E' = \frac{E}{E_H}. \tag{5}$$

With this variable change, defining the wavefunction $\phi(\vec{q}) := \psi(\vec{x}(\vec{q}))$, the Schrödinger equation (2) is left in the following "adimensionalized" shape suitable for numerical computation

$$\left(-\frac{1}{2}\sum_{k=1}^{3}\frac{\partial^2}{\partial {q_k}^2} - \frac{1}{\sqrt{{q_1}^2+{q_2}^2+{q_3}^2}}\right)\phi(\vec{q}) = E'\phi(\vec{q}). \tag{6}$$

where we define the potential $U(\vec{q}) := -1/\sqrt{{q_1}^2+{q_2}^2+{q_3}^2}$.

This is the same result as if we had employed the well-known atomic units [2].

2 The Computational Method

In what follows, we explain a way to generalize the method originally provided by Ref. [3] to numerically solve a Schrödinger-like eigenstate equation.

Consider a spatial grid of nodes $(q_1^{(i)}, q_2^{(j)}, q_3^{(k)}) \in \Omega \subset \mathbb{R}^3$, on the computational domain $\Omega := \prod_{\alpha=1}^{3}(q_\alpha^{min}, q_\alpha^{max}) \subset \mathbb{R}^3$ for all $i \in \{0, ..., N_1 - 1\}$, $j \in \{0, ..., N_2 - 1\}$ and $j \in \{0, ..., N_3 - 1\}$, where the α-th degree of freedom has N_α equi-spaced points

$$q_\alpha^{(l)} := q_\alpha^{min} + l\Delta q_\alpha; \quad \Delta q_\alpha := \frac{q_\alpha^{max} - q_\alpha^{min}}{N_\alpha - 1}. \tag{7}$$

We consider the discretized wavefunction over these nodes as $\phi_{ijk} := \phi(q_1^{(i)}, q_2^{(j)}, q_3^{(k)})$. Then, if we use a centered-difference scheme for the second derivative as

$$\frac{\partial^2 f(q)}{\partial q^2} \simeq \frac{f(q+\Delta q) - 2f(q) + f(q-\Delta q)}{\Delta q^2}, \tag{8}$$

we have that

$$\sum_{\alpha=1}^{3} \frac{\partial^2}{\partial q_\alpha^2}\phi_{ijk} \simeq \tag{9}$$

$$\simeq -2\left(\frac{1}{\Delta q_1^2} + \frac{1}{\Delta q_2^2} + \frac{1}{\Delta q_3^2}\right)\phi_{ijk} + \frac{1}{\Delta q_1^2}(\phi_{i+1jk} + \phi_{i-1jk}) + \frac{1}{\Delta q_2^2}(\phi_{ij+1k} + \phi_{ij-1k}) + \frac{1}{\Delta q_3^2}(\phi_{ijk+1} + \phi_{ijk-1}).$$

This leaves a discretized approximation of equation (2) as

$$\left(\frac{1}{\Delta q_1^2} + \frac{1}{\Delta q_2^2} + \frac{1}{\Delta q_3^2} + V_{ijk}\right)\phi_{ijk} - \tag{10}$$

$$-\frac{1}{2\Delta q_1^2}(\phi_{i+1jk} + \phi_{i-1jk}) - \frac{1}{2\Delta q_2^2}(\phi_{ij+1k} + \phi_{ij-1k}) - \frac{1}{2\Delta q_3^2}(\phi_{ijk+1} + \phi_{ijk-1}) = E\phi_{ijk}$$

where we defined $V_{ijk} := V\left(q_1^{(i)}, q_2^{(j)}, q_3^{(k)}\right)$.

If we then use the boundary condition bounding the eigenstates inside the simulation domain

$$\phi(\vec{q})\Big|_{\vec{q}\notin\Omega} = 0 \implies \quad \phi_{ijk} = 0 \text{ if } i \in \{-1, N_1\} \text{ or } j \in \{-1, N_2\} \text{ or } k \in \{-1, N_3\}, \tag{11}$$

the discretized time independent Schrödinger Equation (10) can be written as a matrix eigenvector problem. For this, we first define the vectorized form of the array ϕ_{ijk} over the simulation domain as ϕ_l, where its index $l \in 1, ..., N_1N_2N_3$ obeys

$$l = iN_3N_2 + jN_3 + k. \tag{12}$$

Then, we note that if we define a sparse square matrix H_{sl} of dimensions $N_1N_2N_3 \times N_1N_2N_3$, such that the following rules are obeyed:

- It is symmetric.
- The main diagonal is formed by the vector $\alpha_l = \frac{1}{\Delta q_1^2} + \frac{1}{\Delta q_2^2} + \frac{1}{\Delta q_3^2} + V_l$ of $N_1N_2N_3$ elements.
- The diagonal with offset 1 is a vector β_n of $N_1N_2N_3 - 1$ elements, such that

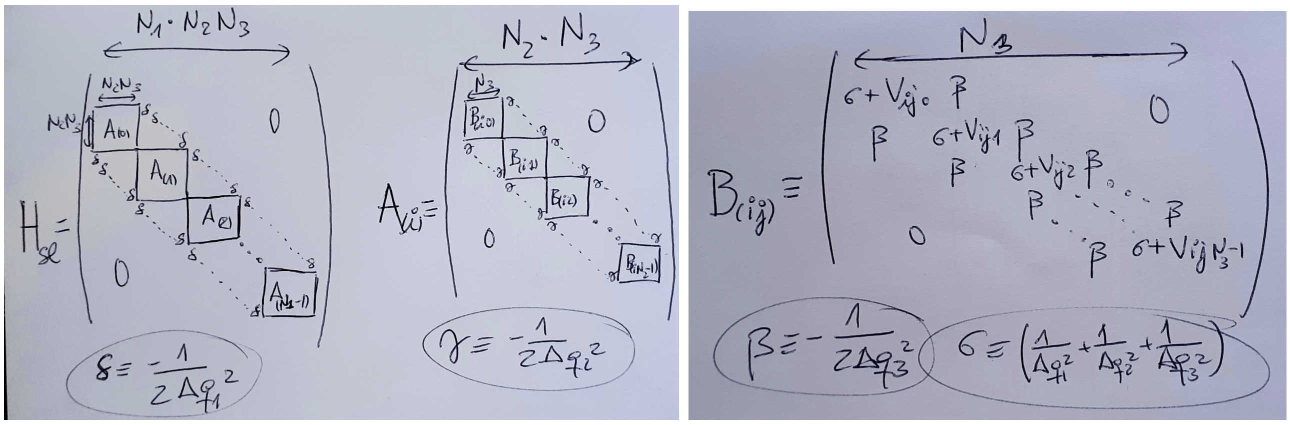
$$\beta_n = \begin{cases} 0 & \text{if } l = (j+1)N_3 - 1 \text{ for } j \in \{0, N_2 - 1\} \\ -\frac{1}{2\Delta q_3^2} & \text{else} \end{cases}$$

- The diagonal with offset N_3 is a vector γ_n of $N_1N_2N_3 - N_3$ elements, such that

$$\gamma_n = \begin{cases} 0 & \text{if } n \in \{(i+1)N_3N_2 - N_3, (i+1)N_3N_2\} \text{ for } i \in \{0, N_1 - 1\} \\ -\frac{1}{2\Delta q_2^2} & \text{else} \end{cases}$$

- The diagonal with offset N_3N_2 is a vector $\delta_n = -\frac{1}{2\Delta q_1^2}$ of $N_1N_2N_3 - N_2N_3$ elements.

These rules shape the matrix that we have more intuitively drawn in Figure 1, where one can even abstract the trend that the construction of the matrix for a higher dimensional quantum system would follow.



$N_1 \cdot N_2 N_3$
$H_{sl} =$
$\delta = -\frac{1}{2\Delta q_1^2}$
$N_2 \cdot N_3$
$A_{(i)} =$
$\gamma = -\frac{1}{2\Delta q_2^2}$
N_3
$B_{(ij)} =$
$\beta = -\frac{1}{2\Delta q_3^2}$
$\sigma = \left(\frac{1}{\Delta q_1^2} + \frac{1}{\Delta q_2^2} + \frac{1}{\Delta q_3^2}\right)$

Figure 1: Schematic depiction of the matrix H_{sl}.

With this we claim that the matrix-eigenvector equation

$$H_{sl}\phi_l = E\phi_s \tag{13}$$

is identical to the discretized time independent Schrödinger equation (10), together with the boundary condition (11).

3 The Implementation

We implemented the solver in Python, as can be found in our Github repository [4]. The `scipy` library [5] is employed to build the sparse matrix H_{sl} and find its eigenstates and eigenvectors, `numpy` [6] is employed for the numerical core, while `matplotlib` [7] is employed for the generation of the plots. The full Python code can be found in the repository mentioned above, but we show in Listings 1, 2, and 3, the routines that respectively build the matrix H_{sl}, that build the grid compute the eigenstates and normalize them, and the one written for the plotting.

Listing 1: Part of the script designed as the solver that builds the discretized Hamiltonian.

```python
import numpy as np
import scipy.sparse as sp
#Discretized Hamiltonian as a Sparse Matrix
def get_discrete_H(Nx, Ny, Nz, dx, dy, dz, xs, ys, zs):
    main_diagonal = -0.5*(-2.0)*(1/dx**2+1/dy**2+1/dz**2)*np.ones(Nx*Ny*Nz)
    for i in range(Nx):
        for j in range(Ny):
            for k in range(Nz):
                main_diagonal[i*Nz*Ny+j*Nz+k] += V(xs[i], ys[j], zs[k])
    z_diagonals = -0.5*1.0/dz**2*np.ones(Nx*Ny*Nz-1)
    y_diagonals = -0.5*1.0/dy**2*np.ones(Nx*Ny*Nz-Nz)
    x_diagonals = -0.5*1.0/dx**2*np.ones(Nx*Ny*Nz-Nz*Ny)
    # There are some zeros we need to place in these diagonals
    for j in range(Ny-1):
        z_diagonals[(j+1)*Nz-1] = 0
    for i in range(Nx-1):
        y_diagonals[ (i+1)*Nz*Ny-Nz:(i+1)*Nz*Ny ] = 0

    return sp.diags( diagonals=
        [main_diagonal, z_diagonals, z_diagonals,y_diagonals, y_diagonals, x_diagonals,
    x_diagonals],
        offsets=[0, 1, -1, Nz, -Nz, Nz*Ny, -Nz*Ny] )
```

Listing 2: Part of the script designed as the solver that defines the grid and solves the eigenvector problem, to then normalize the eigenstates.

```python
import numpy as np
import scipy.sparse.linalg as lg

Ls = np.array([ 18,18,18 ]) # (Lx, Ly, Lz) Bohr Radii
xlowers = -Ls/2.0
xuppers = Ls/2.0

# Number of points to be used per dimension
Ns = [100, 100, 100] # (Nx, Ny, Nz)

# Increments to be used per dimension
dxs = [(xuppers[j]-xlowers[j])/(Ns[j]-1) for j in range(3)] # (dx, dy, dz)


#Create coordinates at which the solution will be calculated
nodes = [np.linspace(xlowers[j], xuppers[j], Ns[j]) for j in range(3)] # (xs, ys, zs)

#Calculation of discrete form of Schrodinger Equation
H=get_discrete_H(*Ns, *dxs, *nodes)


# Diagonalize the matrix H
eigenValues, eigenVectors = lg.eigsh(H, k=num_eig, which='SM', maxiter=200, tol=0.01)

# We will normalise the states "the lazy way":
# instead of integrating each dimension with its own discretization, we will use the average
    one
dx_av = np.mean(dxs)
for k in range(0, num_eig):
    eigenVectors[:,k] /= np.sqrt((np.dot(eigenVectors[:,k], eigenVectors[:,k])*dx_av**3))

# Reshape the eigen-vectors to a 3D array each (in their natural indexing)

eigenStates=eigenVectors.T.reshape((num_eig, *Ns))
```

Listing 3: Part of the script designed as the solver that plots the results.

```python
import numpy as np
import matplotlib.pyplot as plt

every=2 # Only take one data point every this number in each axis to plot
grid = np.array(np.meshgrid(*nodes))[:,::every, ::every, ::every]
print(grid.shape)
for j in range(0, num_eig):
    fig = plt.figure( figsize=(7,7))
    ax = fig.add_subplot(111, projection='3d')

    colormap = ax.scatter3D(*grid, c=eigenStates[j, ::every, ::every, ::every],
          cmap='seismic',  s=0.003, alpha=0.4 ) #, antialiased=True)
    fig.colorbar(colormap, fraction=0.04, location='left')
    ax.set_xlabel("x (Bohr Radii)")
    ax.set_ylabel("y (Bohr Radii)")
    ax.set_zlabel("z (Bohr Radii)")
    ax.set_title(f"Real Part of the {j}-th Energy Eigenstate")
    plt.savefig(f"Re_Eig_{j}_Ener_{eigenValues[j]:.4}_N_{Ns[0]}_L_{Ls[0]}.png")
    #plt.show()

    fig = plt.figure( figsize=(7,7))
    ax = fig.add_subplot(111, projection='3d')

    colormap = ax.scatter3D(*grid, c=np.abs(eigenStates[j, ::every, ::every, ::every])**2,
           cmap='hot', s=0.003, alpha=0.4 ) #, antialiased=True)
    fig.colorbar(colormap, fraction=0.04, location='left')
    ax.set_xlabel("x (Bohr Radii)")
    ax.set_ylabel("y (Bohr Radii)")
    ax.set_zlabel("z (Bohr Radii)")
    ax.set_title(f"Magnitude Squared of the {j}-th Energy Eigenstate")
    plt.savefig(f"pdf_Eig_{j}_Ener_{eigenValues[j]:.4}_N_{Ns[0]}_L_{Ls[0]}.png")
    #plt.show()
    print(f"{j}-th done!")
```

4 Results

Using the script with $N_1, N_2, N_3 = 160$, we obtain the first 9 eigenstates and eigen-energies of the Hydrogen atom, using a simulation box Ω of $6a_0$ unit sides for the ground state (the s orbital), $14a_0$ unit sides for the next three states (the p orbitals) and $25a_0$ unit sides for the last five computed ones (the d orbitals). In average, the time required for the result output was of 0.86 h per eigenstate, using a computer with 16 Gb RAM and a i7-8565U CPU 1.80GHz.

The eigen-energies obtained can be found in Table 1 together with the analytic solutions and relative errors. All the obtained eigenstates were real, so we plotted the wavefunction, in addition to its squared magnitude (representing the probability density for the localization of the electron) in Figures 3-??.

Table 1: Table of obtained results. The analytical energies were obtained with the well-known equation $E_n = -\frac{1}{2n^2}$ (Hartree) for the Hydrogen atom. The relative error is computed as the absolute value of the simulation result minus the analytic one, divided by the analytic one.

Eigenstate Number	**Quantum Number n**	**Simulation Eigen-Energies (Hartree)**	**Analytic Eigen-Energies (Hartree)**	**Absolute Error (Hartree)**
1	1	-0.4963	-0.5	0.0037
2	2	-0.1094	-0.125	0.0156
3	2	-0.1094	-0.125	0.0156
4	2	-0.1056	-0.125	0.0194
5	3	-0.01497	-0.056	0.0410
6	3	-0.005051	-0.056	0.0509

We can see that even though the eigen-energies do not exactly meet the analytic ones for the higher eigen-states, at least for the ground-state (orbital s with $n = 1$) and the p orbitals ($n = 2$), which are the next three, the computation appears to be quite accurate (with an absolute error of 0.01 and 0.015 respectively). Also, for the p orbitals, even if they do not give the exact same energies, at least they give the same energy within the first two significant numbers. This was expectable since the wavefunctions of the p orbitals are to be simply rotated versions of each other. Finally, regarding the supposedly d ($n = 3$) orbitals, we clearly see that the obtained results are not correct, since they are off with a five times bigger error than the previous ones and even if their eigenergies should in theory be the same they are not even in the same order of magnitude.

We could obtain more accurate results with a bigger simulation domain and a higher number of spatial nodes, but since the size of the matrix scales as $O((N_1 N_2 N_3)^2)$ we were not able to do so.

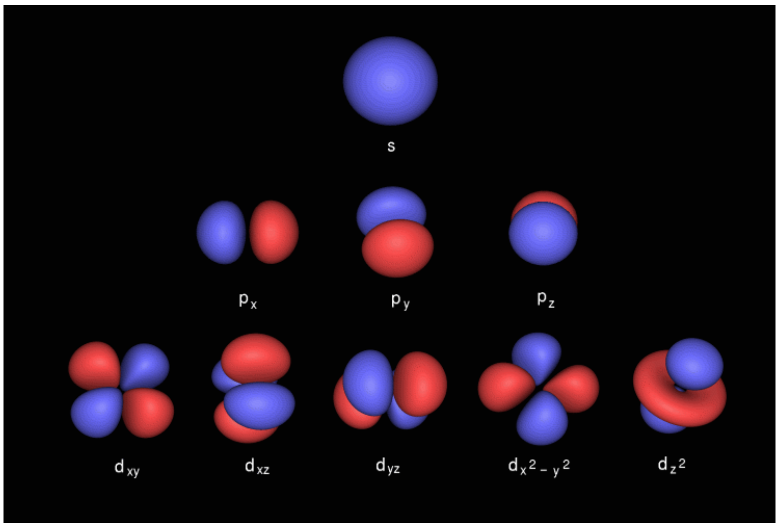


Figure 2: Surface levels of atomic orbitals of Hydrogen taken from Ref. [1].

Regarding the wavefunctions, we see that the ground-state (s orbital) has precisely the radially symmetric shape one expects for the Hydrogen atom, where the core of the probability density is gathered within the ball of radious $1a_0$, just as predicted by the Bohr model. Thus, compared with the analytical shapes of Figure 2, the prediction appears to be fine. For the next three wavefunctions (p orbitals), we obtain the expected shape with a nodal plane that goes normal to the three axes (one for each eigenstate), with the wavefunction having symmetric positive and negative values in each side of the nodal plane. These also match the expected ones in Figure 2. Finally, regarding the last computed eigenstates, we no longer find a correspondence with the expected eigenfunctions of Figure 2. Instead, we find what the f (n=4) orbiatls should look like, which could explain the errors of their eigen-energies. This is probably due to the fact that we employed a too big simulation domain with too few simulation nodes, but the computational time we had did not allow us to improve the result without an expensive overhead.

References

[1] S. M. Blinder, *Quantum Chemistry (Blinder). 1.7. Hydrogen Atom.* LibreTexts Chemistry.

[2] "Wikipedia entry on atomic units." `https://en.wikipedia.org/wiki/Hartree_atomic_units`.

[3] D. G. Truhlar, "Finite difference boundary value method for solving one-dimensional eigenvalue equations," *Journal of Computational Physics*, vol. 10, no. 1, pp. 123–132, 1972.

[4] "Github repository with the python script generated for the report." `https://github.com/Oiangu9/_Miscellaneous/tree/main/SSN`.

[5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[6] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[7] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

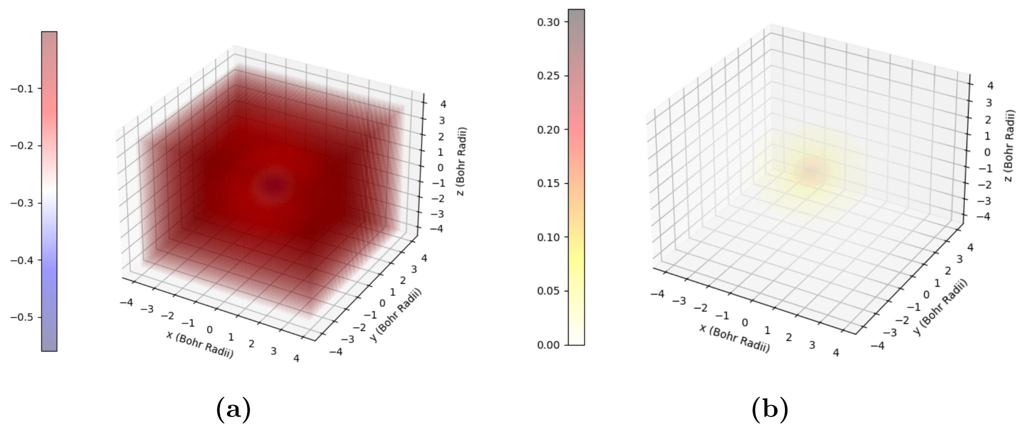


Figure 3: The first eigenestate obtained (from lowest to highest eigen-energy). It clearly shows an s orbital. In (a) the real part of the wavefunction is plotted, while in (b) the absolute value squared (the probability density for the electron presence).

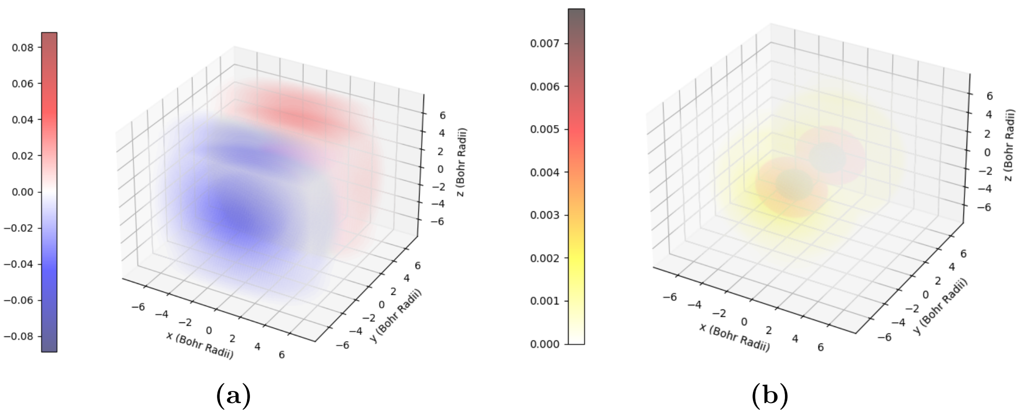


Figure 4: The second eigenestate obtained (from lowest to highest eigen-energy). It clearly shows a p orbital. In (a) the real part of the wavefunction is plotted, while in (b) the absolute value squared (the probability density for the electron presence).

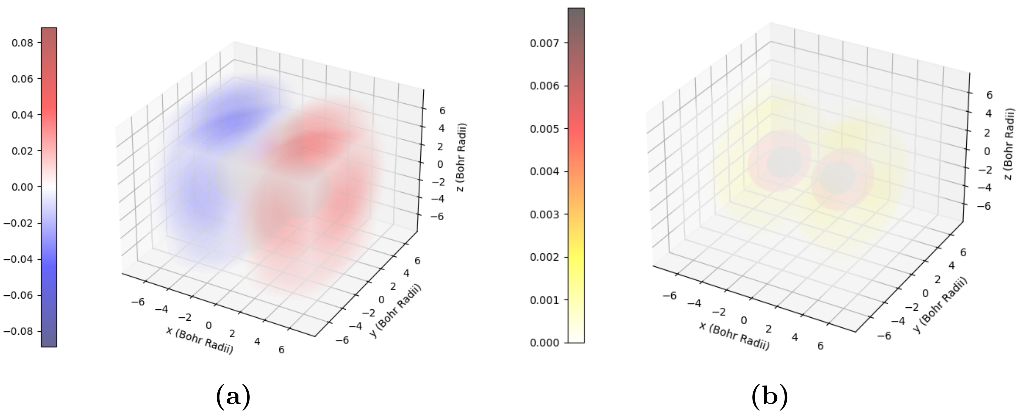


Figure 5: The third eigenestate obtained (from lowest to highest eigen-energy). It clearly shows a p orbital. In (a) the real part of the wavefunction is plotted, while in (b) the absolute value squared (the probability density for the electron presence).

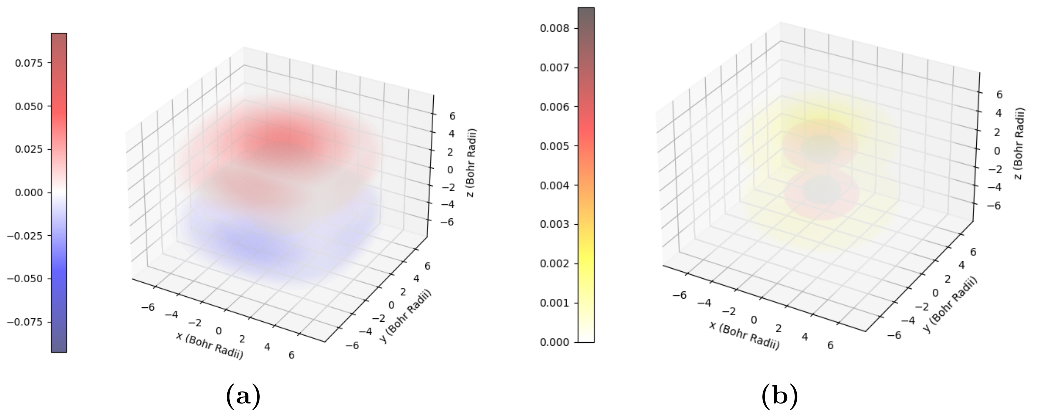


Figure 6: The fourth eigenestate obtained (from lowest to highest eigen-energy). It clearly shows a *p* orbital. In (a) the real part of the wavefunction is plotted, while in (b) the absolute value squared (the probability density for the electron presence).

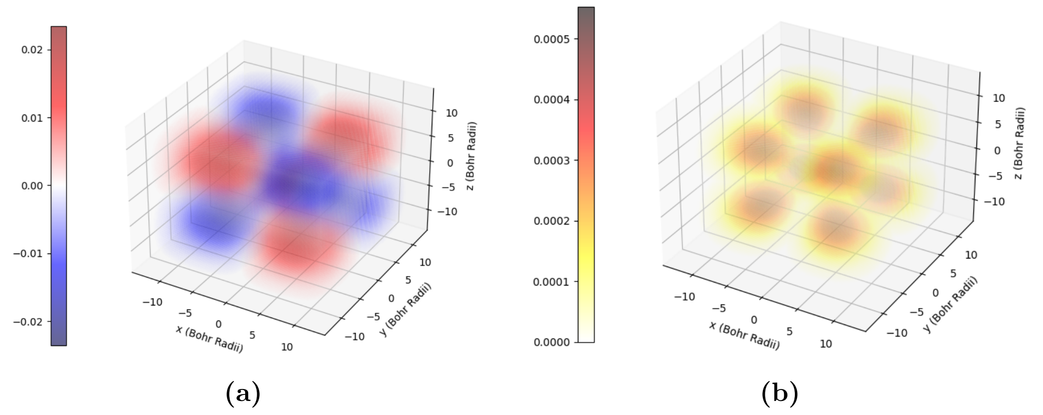


Figure 7: The fifth eigenestate obtained (from lowest to highest eigen-energy). It clearly shows a *d* orbital. In (a) the real part of the wavefunction is plotted, while in (b) the absolute value squared (the probability density for the electron presence).

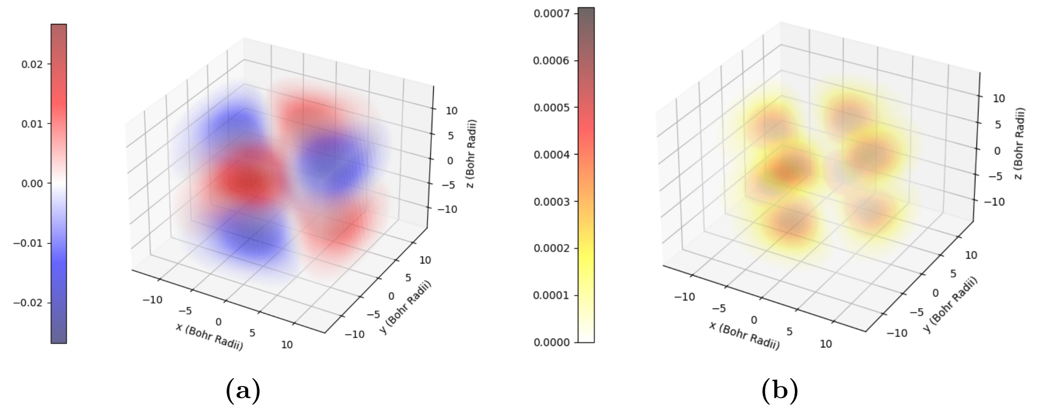


Figure 8: The sixth eigenestate obtained (from lowest to highest eigen-energy). It clearly shows a *d* orbital. In (a) the real part of the wavefunction is plotted, while in (b) the absolute value squared (the probability density for the electron presence).