QUPLOTS

Documentación

Autor: R.G José Adrián
R.V. Maria Isabel

Tutor: Dr. Castellanos Águila Jesús Eduardo

8 de agosto de 2025

Índice

1. Introducción

La química es una herramienta esencial para entender el funcionamiento del mundo que nos rodea. Cuando trasladamos este estudio a una escala más pequeña y compleja, nos adentramos en el campo de la química cuántica. Comprender estos fenómenos resulta desafiante, ya que suelen ser intangibles o demasiado pequeños para ser observados directamente.
Actualmente, existen diversos recursos que facilitan la comprensión de estos conceptos. Los modelos gráficos que representan fenómenos naturales han demostrado ser de gran utilidad para el aprendizaje, ya que permiten visualizar procesos que de otro modo serían abstractos. Un ejemplo claro es el caso de las partículas, cuya dinámica puede ser difícil de imaginar sin herramientas de visualización.
Esta librería, desarrollada en el lenguaje de programación Python, tiene como objetivo mostrar gráficamente fenómenos de la química cuántica y ayudar a los estudiantes a comprender estos conceptos a través de simulaciones visuales. En esta documentación, exploraremos cómo utilizar cada módulo de la librería "Quplots".

2. Instalación

Página oficial de la librería en pypi `https://pypi.org/project/quplots/`
Se recomienda crear un entorno virtual.

Entorno de conda

```
conda create  -n nombre_entorno
#Activar
\texttt{conda activate nombre\_entorno}
#Desactivar
conda deactivate
#Instalar quplots
pip install quplots
```

Entorno de venv

```
python -m venv nombre_entorno
# Windows
nombre_entorno\Scripts\activate
# macOS / Linux
source nombre_entorno/bin/activate

# Desactivar
deactivate
#Instalar quplots
pip install quplots
```

3. Estructura de QuPlots

Quplots es una librería que posee 3 ramificaciones principales

- *quplots.electron*
- *quplots.hybridization*
- *quplots.plots*

4. quplots.electron

Esta clase hace los cálculos necesarios para describir las propiedades geométricas del electrón. Esta clase posee los siguientes atributos

- n, el número cuántico principal, define los niveles de energía del electrón.
- l, el número cuántico azimutal, define los subniveles o "subcapas"de energía del electrón.
- m, el número cuántico magnético, define la orientación de la dirección de movimiento del electrón.
- $radius$, Se define un radio atómico.

Los números cuánticos deben ser enteros y poseen las siguientes reglas

$$n \in \mathbb{N} = \{1, 2, 3, ...\} \tag{1}$$
$$l \in \mathbb{Z}^+ = \{0 \leq l < n\} \tag{2}$$
$$m \in \mathbb{Z} = \{-l \leq m \leq l\} \tag{3}$$

El radio puede ser $r \in \mathbb{R}^+$

El siguiente bloque de código muestra como instanciar un objeto del tipo electron, el objeto tendrá los números $n = 1$, $l = 0$, $m = 0$, radio $= 1$

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
```

La clase posee las siguientes funciones.

1. `get_cartesian_grid()`
2. `get_spherical_grid(x,y,z)`
3. `compute_radial()`
4. `compute_real_spherical()`
5. `compute_imaginary_spherical()`
6. `compute_wavefunction_2D(d)`
7. `compute_wavefunction_3D()`
8. `compute_wavefunction_3D_complex()`
9. `getN()`
10. `getL()`
11. `getM()`
12. `getRadius()`

4.1. `get_cartesian_grid()`

La función no requiere ningún parámetro de entrada,devuelve una tupla de 3 elementos, x, y, z. La función genera una malla de tres dimensiones, cada elemento es un array de numpy, útil para cálculos posteriores. Esta malla por defecto está definida en un rango de $x, y, z \in [-10, 10]$ con 51 divisiones

`get_cartesian_grid()`

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
x,y,z=e.get_cartesian_grid()
print(x)
```

```
array([[[-10. , -10. , -10. , ..., -10. , -10. , -10. ], [ -9.6, -9.6, -9.6, ..., -9.6, -9.6, -9.6], [ -9.2, -9.2, -9.2,
..., -9.2, -9.2, -9.2], ..., [ 9.2, 9.2, 9.2, ..., 9.2, 9.2, 9.2], [ 9.6, 9.6, 9.6, ..., 9.6, 9.6, 9.6], [ 10. , 10. , 10.
, ..., 10. , 10. , 10. ]],
[[-10. , -10. , -10. , ..., -10. , -10. , -10. ], [ -9.6, -9.6, -9.6, ..., -9.6, -9.6, -9.6], [ -9.2, -9.2, -9.2, ..., -9.2,
-9.2, -9.2], ..., [ 9.2, 9.2, 9.2, ..., 9.2, 9.2, 9.2], [ 9.6, 9.6, 9.6, ..., 9.6, 9.6, 9.6], [ 10. , 10. , 10. , ..., 10.
, 10. , 10. ]],
[[-10. , -10. , -10. , ..., -10. , -10. , -10. ], [ -9.6, -9.6, -9.6, ..., -9.6, -9.6, -9.6], [ -9.2, -9.2, -9.2, ..., -9.2,
-9.2, -9.2], ..., [ 9.2, 9.2, 9.2, ..., 9.2, 9.2, 9.2], [ 9.6, 9.6, 9.6, ..., 9.6, 9.6, 9.6], [ 10. , 10. , 10. , ..., 10.
, 10. , 10. ]],
```

4.2. get_spherical_grid(x,y,z)

La función recibe como parámetros 3 variables; x,y,z. Las variables tienen que ser o una lista, tupla, un arreglo de la librería numpy. Devuelve una tupla de tres elementos, cada elemento es un array de la librería numpy. La función convierte las coordenadas cartesianas entregadas en la entrada a coordenadas esférica, devueltas en al salida de la función.

Las coordenadas r, θ, ϕ, son de gran utilidad en la solución de la ecuación de Schrödinger, ya que es más clara y a su vez, no llega a ser complejo, el describir a la partícula en coordenadas esféricas que en coordenadas cartesianas.

get_spherical_grid(x,y,z)

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
x,y,z=e.get_cartesian_grid()
r,theta,phi=e.get_spherical_grid(x,y,z)
r
```

La conversión de coordenadas reales a esféricas se realiza como lo describe la (Eq.4,5,6) [2]

$$r = \sqrt{x^2 + y^2 + z^2} \tag{4}$$

$$\phi = \tan^{-1}\left(\frac{y}{x}\right) \tag{5}$$

$$\theta = \cos^{-1}\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right) \tag{6}$$

4.3. compute_radial()

Calcula la función radial de la ecuación de Erwin Schrödinger. No recibe argumentos y devuelve un escalar para la función radial. La función radial se define como [1]

$$R_{nl}(r) = \sqrt{\left(\frac{2}{n\alpha_0}\right)^3 \frac{(n-l-1)!}{2n(n+l)!}} \left(e^{\frac{-r}{n\alpha_o}}\right) \left(\frac{2r}{n\alpha_0}\right)^l \cdot L^{2l+1}_{n-l-1}\left(\frac{2r}{n\alpha_0}\right) \tag{7}$$

Siendo n, l, los números cuánticos principal y azimutal. α_0 el radio de Bohr y $L^{2l+1}_{n-l-1}\left(\frac{2r}{n\alpha_0}\right)$, las funciones asociadas de Laguerre.

compute_radial()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
e.compute_radial()
```

```
0.7357588823428847
```

4.4. compute_real_spherical()

No requiere parámetros de entrada, y devuelve una tupla de 5 elementos. El valor de la función de los armónicos esféricos, el valor de las coordenadas θ, ϕ, los rangos de colores definidos por la distribución de probabilidad y los números cuánticos l,m. Calcula la función real de los armónicos esféricos La función de armónicos esféricos reales se define por

$$Y_{lm}(\theta,\phi) = (-1)^m \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} e^{im\phi} \cdot P_l^m(\cos\theta) \tag{8}$$

Donde l, m,son los números cuánticos azimutal y magnético. y $P_l^m(\cos\theta)$ Las funciones asociadas de Legendre

compute_real_spherical()

```
from quplots import electron
e=electron(1,0,0,1)
Y,theta,phi,colors,l,m=e.compute_real_spherical()
print(f'Y={Y}')
print(f'theta={theta}')
print(f'phi={phi}')
print(f'Probability Colors={colors}')
print(f'Number L={l}')
print(f'Number m={m}')
```

```
Y=[[0.28209479 0.28209479 0.28209479 ... 0.28209479 0.28209479 0.28209479] [0.28209479
0.28209479 0.28209479 ... 0.28209479 0.28209479 0.28209479] [0.28209479 0.28209479 0.28209479 ...
0.28209479 0.28209479 0.28209479] ... [0.28209479 0.28209479 0.28209479 ... 0.28209479 0.28209479
0.28209479] [0.28209479 0.28209479 0.28209479 ... 0.28209479 0.28209479 0.28209479] [0.28209479
0.28209479 0.28209479 ... 0.28209479 0.28209479 0.28209479]]
theta=[[0. 0.03173326 0.06346652 ... 3.07812614 3.10985939 3.14159265] [0. 0.03173326 0.06346652 ...
3.07812614 3.10985939 3.14159265] [0. 0.03173326 0.06346652 ... 3.07812614 3.10985939 3.14159265]
... [0. 0.03173326 0.06346652 ... 3.07812614 3.10985939 3.14159265] [0. 0.03173326 0.06346652 ...
3.07812614 3.10985939 3.14159265] [0. 0.03173326 0.06346652 ... 3.07812614 3.10985939 3.14159265]]
phi=[[0. 0. 0. ... 0. 0. 0. ] [0.06346652 0.06346652 0.06346652 ... 0.06346652 0.06346652 0.06346652]
[0.12693304 0.12693304 0.12693304 ... 0.12693304 0.12693304 0.12693304] ... [6.15625227 6.15625227
6.15625227 ... 6.15625227 6.15625227 6.15625227] [6.21971879 6.21971879 6.21971879 ... 6.21971879
6.21971879 6.21971879] [6.28318531 6.28318531 6.28318531 ... 6.28318531 6.28318531 6.28318531]]
Probability Colors=[[nan nan nan ... nan nan nan] [nan nan nan ... nan nan nan] [nan nan nan ...
nan nan nan] ... [nan nan nan ... nan nan nan] [nan nan nan ... nan nan nan] [nan nan nan ... nan
nan nan]]
Number L=0
Number m=0
```

4.5. compute_imaginary_spherical()

Función que no recibe parámetros y que devuelve una tupla de 5 elementos. El valor de la función de los armónicos esféricos, el valor de las coordenadas θ, ϕ, los rangos de colores definidos por la distribución de probabilidad y los números cuánticos l,m. Calcula la parte compleja de los armónicos esféricos

compute_imaginary_spherical()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
Y,theta,phi,colors,l,m=e.compute_imaginary_spherical()
print(f'Y={Y}')
print(f'theta={theta}')
print(f'phi={phi}')
print(f'Probability Colors={colors}')
print(f'Number L={l}')
print(f'Number m={m}')
```

4.6. compute_wavefunction_2D(d=20)

La función recibe un parámetro de amplitud, por defecto es de 20. Devuelve el valor de psi(ψ),r,n,l,m Esta función realiza la proyección de la función ψ de onda para un plano de dos dimensiones.

$$|\psi_{n,l,m}|^2 = |R_n(r) \cdot Y_{l,m}(\theta, \phi)|^2 \tag{9}$$

compute_wavefunction_2D(d)

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
psi,r,n,l,m=e.compute_wavefunction_2D()
print(f'Psi={psi}')
print(f'rmax={r}')
print(f'n={n}')
print(f'Number L={l}')
print(f'Number m={m}')
```

```
Psi=[[4.32069605e-05 4.39825656e-05 4.47704891e-05 ... 4.47704891e-05 4.39825656e-05 4.32069605e-05] [4.39825656e-05 4.47736914e-05 4.55774175e-05 ... 4.55774175e-05 4.47736914e-05 4.39825656e-05] [4.47704891e-05 4.55774175e-05 4.63972337e-05 ... 4.63972337e-05 4.55774175e-05 4.47704891e-05] ... [4.47704891e-05 4.55774175e-05 4.63972337e-05 ... 4.63972337e-05 4.55774175e-05 4.47704891e-05] [4.39825656e-05 4.47736914e-05 4.55774175e-05 ... 4.55774175e-05 4.47736914e-05 4.39825656e-05] [4.32069605e-05 4.39825656e-05 4.47704891e-05 ... 4.47704891e-05 4.39825656e-05 4.32069605e-05]]
rmax=3.148314831483148
n=1
Number L=0
Number m=0
```

4.7. compute_wavefunction_3D()

La función no requiere ningún parámetro de entrada, devuelve un arreglo de la librería numpy de 3 dimensiones(tensor). La función calcula la función de onda de probabilidad en 3 dimensiones. Esta

función es útil cuando se gráfica en 3 dimensiones.

compute_wavefunction_3D()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
psi=e.compute_wavefunction_3D()
print(f"Psi={psi}")
```

```
Psi=[[[1.69521186e-08 2.12894660e-08 2.65657695e-08 ... 2.65657695e-08 2.12894660e-08
1.69521186e-08] [2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08
2.12894660e-08] [2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08
2.65657695e-08] ... [2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08
2.65657695e-08] [2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08
2.12894660e-08] [1.69521186e-08 2.12894660e-08 2.65657695e-08 ... 2.65657695e-08 2.12894660e-08
1.69521186e-08]]
[[2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08 2.12894660e-08]
[2.68189779e-08 3.38931572e-08 4.25594062e-08 ... 4.25594062e-08 3.38931572e-08 2.68189779e-08]
[3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
... [3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
[2.68189779e-08 3.38931572e-08 4.25594062e-08 ... 4.25594062e-08 3.38931572e-08 2.68189779e-08]
[2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08 2.12894660e-08]]
[[2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08 2.65657695e-08]
[3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
[4.21463336e-08 5.36131594e-08 6.77641615e-08 ... 6.77641615e-08 5.36131594e-08 4.21463336e-08]
... [4.21463336e-08 5.36131594e-08 6.77641615e-08 ... 6.77641615e-08 5.36131594e-08 4.21463336e-08]
[3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
[2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08 2.65657695e-08]]
...
[[2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08 2.65657695e-08]
[3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
[4.21463336e-08 5.36131594e-08 6.77641615e-08 ... 6.77641615e-08 5.36131594e-08 4.21463336e-08]
... [4.21463336e-08 5.36131594e-08 6.77641615e-08 ... 6.77641615e-08 5.36131594e-08 4.21463336e-08]
[3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
[2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08 2.65657695e-08]]
[[2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08 2.12894660e-08]
[2.68189779e-08 3.38931572e-08 4.25594062e-08 ... 4.25594062e-08 3.38931572e-08 2.68189779e-08]
[3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
... [3.35686777e-08 4.25594062e-08 5.36131594e-08 ... 5.36131594e-08 4.25594062e-08 3.35686777e-08]
[2.68189779e-08 3.38931572e-08 4.25594062e-08 ... 4.25594062e-08 3.38931572e-08 2.68189779e-08]
[2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08 2.12894660e-08]]
[[1.69521186e-08 2.12894660e-08 2.65657695e-08 ... 2.65657695e-08 2.12894660e-08 1.69521186e-08]
[2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08 2.12894660e-08]
[2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08 2.65657695e-08]
... [2.65657695e-08 3.35686777e-08 4.21463336e-08 ... 4.21463336e-08 3.35686777e-08 2.65657695e-08]
[2.12894660e-08 2.68189779e-08 3.35686777e-08 ... 3.35686777e-08 2.68189779e-08 2.12894660e-08]
[1.69521186e-08 2.12894660e-08 2.65657695e-08 ... 2.65657695e-08 2.12894660e-08 1.69521186e-08]]]
```

4.8. compute_wavefunction_3D_complex()

Función que no recibe ningún parámetro y que devuelve un tensor de 3-D. Calcula la parte compleja de la función de onda.

$$|\psi_{n,l,m}|^2 = |R_{n,l}(r)\Re\left(Y_{l,m}(\theta,\phi)\right)\Im\left(Y_{l,m}(\theta,\phi)\right)|^2 \tag{10}$$

compute_wavefunction_3D_complex()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
psi=e.compute_wavefunction_3D_complex()
print(f"Psi={psi}")
```

4.9. getN()

Función que no recibe valores y devuelve el atributo n (número cuántico principal) de nuestro objeto la clase electron.

getN()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
n=e.getN()
print(n)
```

```
1
```

4.10. getL()

Función que no recibe valores y devuelve el atributo l (número cuántico azimutal) de nuestro objeto de la clase electron

getL()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
l=e.getL()
print(l)
```

```
0
```

4.11. getM()

Función que no recibe valores y devuelve el atributo m (número cuántico magnético) de nuestro objeto la clase electron.

getM()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
```

```
m=e.getM()
print(n)
```

```
0
```

4.12. getRadius()

Función que no recibe valores y devuelve el atributo radius de nuestro objeto de la clase electron

getRadius()

```
from quplots import electron
# n,l, m, y radio
e=electron(1,0,0,1)
radius=e.getRadius()
print(l)
```

```
1
```

5. quplots.hybridization

La hibridación posee los siguientes métodos

1. generate_sp()
2. generate_sp2()
3. generate_sp3()
4. generate_sp2d()
5. generate_sp3d()
6. generate_sp3d2()

```
from quplots import hybridization
h=hybridization()
```

5.1. generate_sp()

La función no recibe valores y devuelvea una tupla de dos tensores

generate_sp()

```
from quplots import hybridization
h=hybridization()
psi1,psi2=h.generate_sp()
print(psi1)
```

La hibridación es la combinación lineal de funciones ψ de onda, cumpliendo la ortonormalidad. La hibridación sp (linear) se define por las siguientes ecuaciones [3]

$$\psi_{sp_1} = \frac{1}{\sqrt{2}}(\psi_s - \psi p_z) \tag{11}$$

$$\psi_{sp_1} = \frac{1}{\sqrt{2}}(\psi_s + \psi p_z) \tag{12}$$

5.2. generate_sp2()

Función que no recibe valores de entrada y que devuelve una tupla de 3 tensores.

generate_sp2()

```
from quplots import hybridization
h=hybridization()
psi1,psi2,psi3=h.generate_sp2()
print(psi1)
```

La hibridación sp^2 (trigonal planar) está descrita por las siguientes ecuaciones [3]

$$\psi_{sp_1^2} = \frac{1}{\sqrt{3}}\left(\psi_s + \sqrt{2}\psi_{p_z}\right) \tag{13}$$

$$\psi_{sp_2^2} = \frac{1}{\sqrt{3}}\left(\psi_s - \sqrt{\frac{1}{2}}\psi_{p_x} - \sqrt{\frac{3}{2}}\psi_{p_z}\right) \tag{14}$$

$$\psi_{sp_3^2} = \frac{1}{\sqrt{3}}\left(\psi_s - \sqrt{\frac{1}{2}}\psi_{p_x} + \sqrt{\frac{3}{2}}\psi_{p_y}\right) \tag{15}$$

5.3. generate_sp3()

generate_sp3()

```
from quplots import hybridization
h=hybridization()
psi1,psi2,psi3,psi4=h.generate_sp3()
print(psi1)
```

La hibridación sp^3 (tetrahedral) está descrita por las siguientes ecuaciones [3]

$$\psi_{sp_1^3} = \frac{1}{2}\left(\psi_s + \psi_{p_x} + \psi_{p_y} + \psi_{p_z}\right) \tag{16}$$

$$\psi_{sp_2^3} = \frac{1}{2}\left(\psi_s + \psi_{p_x} - \psi_{p_y} - \psi_{p_z}\right) \tag{17}$$

$$\psi_{sp_3^3} = \frac{1}{2}\left(\psi_s - \psi_{p_x} - \psi_{p_y} + \psi_{p_z}\right) \tag{18}$$

$$\psi_{sp_4^3} = \frac{1}{2}\left(\psi_s - \psi_{p_x} + \psi_{p_y} - \psi_{p_z}\right) \tag{19}$$

5.4. generate_sp2d()

Función que no se ingresan valores y retorna una tupla de 4 tensores

generate_sp2d()

```
from quplots import hybridization
h=hybridization()
psi1,psi2,psi3,psi4=h.generate_sp2d()
print(psi1)
```

La hibridación sp^2d (square planar) se describe con las siguientes ecuaciones [3]

$$\psi_{sp^2d_1} = \frac{1}{2}\psi_s + \frac{1}{\sqrt{2}}\psi_{p_x} + \frac{1}{2}\psi_{d_{x^2-y^2}} \tag{20}$$

$$\psi_{sp^2d_2} = \frac{1}{2}\psi_s - \frac{1}{\sqrt{2}}\psi_{p_x} + \frac{1}{2}\psi_{d_{x^2-y^2}} \tag{21}$$

$$\psi_{sp^2d_3} = \frac{1}{2}\psi_s + \frac{1}{\sqrt{2}}\psi_{p_y} - \frac{1}{2}\psi_{d_{x^2-y^2}} \tag{22}$$

$$\psi_{sp^2d_4} = \frac{1}{2}\psi_s - \frac{1}{\sqrt{2}}\psi_{p_y} - \frac{1}{2}\psi_{d_{x^2-y^2}} \tag{23}$$

5.5. generate_sp3d()

Función que no se le pasan valores y retorna una tupla de 5 tensores.

generate_sp3d()

```
from quplots import hybridization
h=hybridization()
psi1,psi2,psi3,psi4,psi5=h.generate_sp3d()
print(psi1)
```

La hibridación sp^3d (trigonal bipyramidal) se describe como [3] :

$$\psi_{sp^3d_1} = \sqrt{\frac{1}{3}}\left(\psi_s + \sqrt{2}\psi_{p_z}\right) \tag{24}$$

$$\psi_{sp^3d_2} = \sqrt{\frac{1}{3}}\left(\psi_s - \sqrt{\frac{1}{2}}\psi_{p_x} + \sqrt{\frac{3}{2}}\psi_{p_y}\right) \tag{25}$$

$$\psi_{sp^3d_3} = \sqrt{\frac{1}{3}}\left(\psi_s - \sqrt{\frac{1}{2}}\psi_{p_x} - \sqrt{\frac{3}{2}}\psi_{p_z}\right) \tag{26}$$

$$\psi_{sp^3d_4} = \sqrt{\frac{1}{2}}\left(\psi_{d_{z^2}} + \psi_{p_z}\right) \tag{27}$$

$$\psi_{sp^3d_5} = \sqrt{\frac{1}{2}}\left(-\psi_{d_{z^2}} + \psi_{p_z}\right) \tag{28}$$

5.6. generate_sp3d2()

La función no recibe valores y retorna una tupla de 6 tensores

generate_sp3d2()

```
from quplots import hybridization
h=hybridization()
psi1,psi2,psi3,psi4,psi5,psi6=h.generate_sp3d2()
print(psi1)
```

La hibridación sp^3d^2 (trigonal bipyramidal), se describe como [3]:

$$\psi_{sp^3d^2_1} = \frac{1}{\sqrt{6}}\psi_s + \frac{1}{\sqrt{2}}\psi_{p_z} + \frac{1}{\sqrt{3}}\psi_{d_{z^2}}$$
$$\psi_{sp^3d^2_2} = \frac{1}{\sqrt{6}}\psi_s + \frac{1}{\sqrt{2}}\psi_{p_x} - \frac{1}{\sqrt{12}}\psi_{d_{z^2}} + \frac{1}{2}\psi_{d_{x^2-y^2}}$$
$$\psi_{sp^3d^2_3} = \frac{1}{\sqrt{6}}\psi_s + \frac{1}{\sqrt{2}}\psi_{p_y} - \frac{1}{\sqrt{12}}\psi_{d_{z^2}} - \frac{1}{2}\psi_{d_{x^2-y^2}}$$
$$\psi_{sp^3d^2_4} = \frac{1}{\sqrt{6}}\psi_s - \frac{1}{\sqrt{2}}\psi_{p_x} - \frac{1}{\sqrt{12}}\psi_{d_{z^2}} + \frac{1}{2}\psi_{d_{x^2-y^2}}$$
$$\psi_{sp^3d^2_5} = \frac{1}{\sqrt{6}}\psi_s - \frac{1}{\sqrt{2}}\psi_{p_y} - \frac{1}{\sqrt{12}}\psi_{d_{z^2}} - \frac{1}{2}\psi_{d_{x^2-y^2}}$$
$$\psi_{sp^3d^2_6} = \frac{1}{\sqrt{6}}\psi_s - \frac{1}{\sqrt{2}}\psi_{p_z} + \frac{1}{\sqrt{3}}\psi_{d_{z^2}}$$

6. quplots.plots

La clase plots, posee los siguientes métodos

1. `plot_radial_(elect=None,d=None,n=None,l=None,**plot_kwargs)`
2. `plot_spherical_real( elect=None, R=None, theta=None, phi=None, fcolors=None, l=None, m=None, **plots_kwargs)`
3. `plot_spherical_imaginary( elect=None, R=None, theta=None, phi=None, fcolors=None, l=None, m=None, **plots_kwargs)`
4. `plot_wf_2d(elect=None,psi_sq=None,max_r=None,n=None,l=None,m=None, **plot_kwargs):`
5. `plot_wf_3d(elect,**kwargs_plot)`
6. `plot_sp(**kwargs)`
7. `plot_sp2(**kwargs)`
8. `plot_sp3(**kwargs)`
9. `plot_sp2d(**kwargs)`
10. `plot_sp3d(**kwargs)`
11. `plot_sp3d2(**kwargs)`

Para usar todas las funciones de esta librería se debe de instanciar un objeto de la clase plots

```
from quplots import plots
p=plots()
```

Ninguna de las funciones retorna algo, solamente realizan gráficos con matplotlib y plotly

6.1. plot_radial_(elect=None,d=None,n=None,l=None,**plot_kwargs)

La función tiene varios parámetros que pueden ser opcionales. Él parámetro elect, debe ser un objeto de la clase electrón que se encuentren definidos sus valores de n, l, m, r. Si colocamos este parámetro no es necesario colocar d,n,l. Sin embargo, si se prefieren colocar manualmente cada uno de los parámetros. elect debe ser none, el valor de d, es el radio atómico que puede contener valores de $\mathbb{R}^+$, n debe ser $\mathbb{N}$, ya que es el número cuántico principal y l es el número azimutal que es $\mathbb{Z}^+, l < n$. El parámetro **plot_kwargs, permite añadir personalizar los estilos de los gráficos. Puede encontrar un anexo de todos esos estilos en `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib-pyplot-plot`.

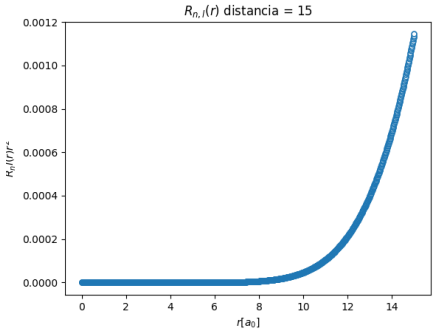

Figura 1: Gráfico con el objeto electron

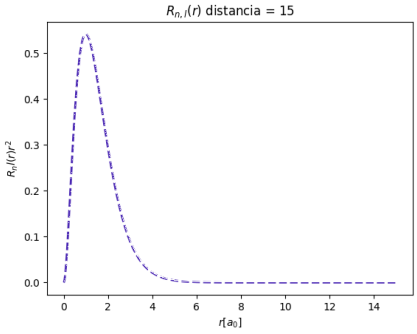

Figura 2: Gráfico sin el objeto

plot_radial_(elect=None,d=None,n=None,l=None,plot_kwargs) con objeto**

```
from quplots import electron,plots

e1=electron(1,0,0,15)
#Metodo con el objeto electron
p=plots()
p.plot_radial_(
    e1,
    color='#1f77b4',           # Azul elegante (paleta estándar de matplotlib)
    linestyle='-',            # Línea continua
    linewidth=2.0,            # Línea moderadamente gruesa
    marker='o',               # Marcador circular
    markersize=5,             # Tamaño de marcador moderado
    markerfacecolor='white', # Interior blanco para contraste
    markeredgecolor='#1f77b4',# Borde del marcador a juego con la línea
    markeredgewidth=1.2,      # Grosor del borde
    alpha=1.0                # Sin transparencia
)
```

plot_radial_(elect=None,d=None,n=None,l=None,plot_kwargs) sin objeto**

```
from quplots import plots
p=plots()
#metodo con la forma manual de agregar elementos
p.plot_radial_(None,15,1,0,    color='#4a27b4',           # Azul elegante (paleta
↪  estándar de matplotlib)
    linestyle='--',            # Línea continua
    linewidth=2.0,            # Línea moderadamente gruesa
    marker=',',               # Marcador circular
    markersize=5,             # Tamaño de marcador moderado
    markerfacecolor='white', # Interior blanco para contraste
    markeredgecolor='#1f77b4',# Borde del marcador a juego con la línea
    markeredgewidth=1.2,      # Grosor del borde
    alpha=1.0                # Sin transparencia
)
```

6.2. plot_spherical_real(elect=None, R=None, theta=None, phi=None, fcolors=None, l=None, m=None, **plots_kwargs)

La función requiere varios parámetros. Ya sea pasarle el objeto de electrón o colocar manualmente los valores. Estos valores son resultados de la función del objeto `e.compute_real_spherical()`. Recordando la sección 3.4, la función `compute_real_spherical()` devuelve una tupla de de 6 elementos, los primeros 4 son arreglos de la librería numpy de 2 dimensiones. Los otros dos elementos son 2 escalares.
Nota: La librería en la versión 0.0.1 tiene un bug al momento de correr el código 'plot_spherical_real sin el objeto' en un cuaderno Google Colab, esto se debe ya que Google Colab no puede renderizarlo adecuadamente, debido a cuestiones numéricas de θ, ϕ , por lo que se recomienda que se utilice el código 'plot_spherical_real con el objeto' en un cuaderno Google Colab.

plot_spherical_real() con el objeto

```
from quplots import electron,plots
p=plots()
# n,l, m, y radio
e1=electron(6,5,0,15)
R,theta,phi,colors,l,m=e1.compute_real_spherical()
p.plot_spherical_real(e1,colorscale='Inferno', showscale=False)
```

plot_spherical_real() sin el objeto

```
from quplots import electron,plots
p=plots()
# n,l, m, y radio
e1=electron(6,5,0,15)
R,theta,phi,colors,l,m=e1.compute_real_spherical()
p.plot_spherical_real(None,R,phi,theta,colors,l,m,colorscale='Icefire',
↪  showscale=False)
```

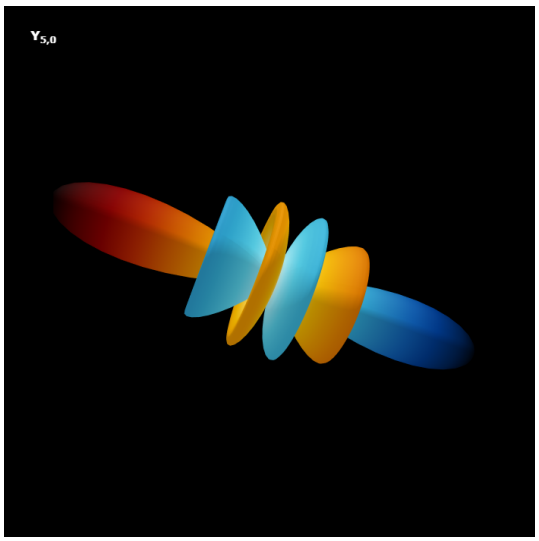

Figura 3: Gráfico con el objeto `electrón`

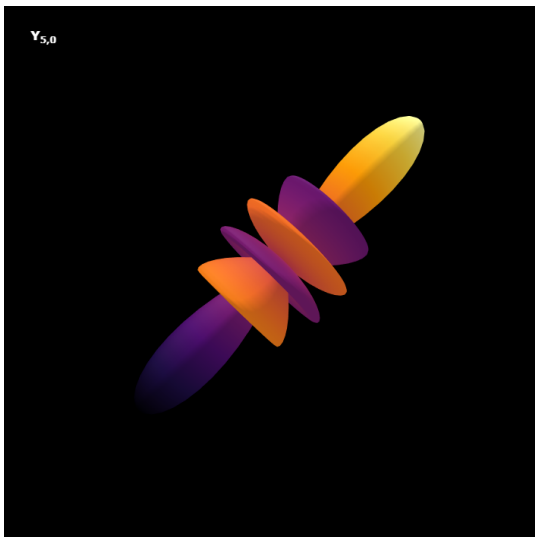

Figura 4: Gráfico sin el objeto

6.3. plot_spherical_imaginary(elect=None, R=None, theta=None, phi=None, fcolors=None, l=None, m=None, **plots_kwargs)

La función requiere varios parámetros. Ya sea pasarle el objeto de electron o colocarle manualmente los valores. Estos valores son resultados de la función del objeto `e.compute_imaginary_spherical()`. Recordando la sección 3.4, la función `compute_imaginary_spherical()` devuelve una tupla de de 6 elementos, los primeros 4 son arreglos de la librería numpy de 2 dimensiones. Los otros dos elementos son 2 escalares.

plot_spherical_imaginary con objeto y sin objeto

```
from quplots import electron,plots
p=plots()
# n,l, m, y radio
e1=electron(6,5,-1,15)
R,theta,phi,colors,l,m=e1.compute_imaginary_spherical()
p.plot_spherical_imaginary(e1,colorscale='Inferno')
p.plot_spherical_imaginary(None,R,phi,theta,colors,l,m,colorscale='Icefire')
```

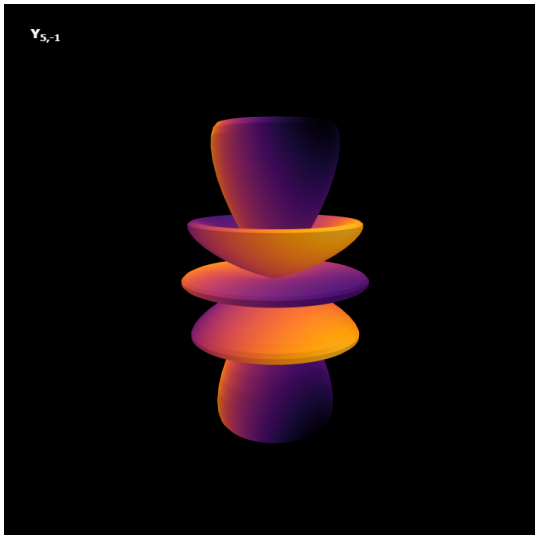

Figura 5: Gráfico con el objeto `electron`

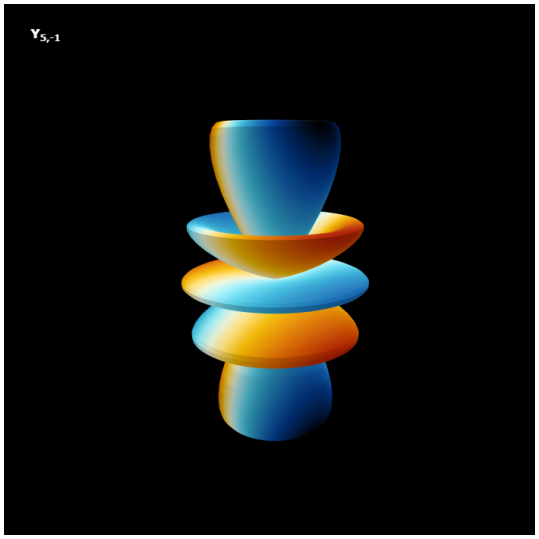

Figura 6: Gráfico sin el objeto

6.4. plot_wf_2d(elect=None,psi_sq=None,max_r=None,n=None,l=None,m=None, **plot_kwargs)

La función requiere el objeto electro de la clase electron o requiere los parámetros psi_sq,max_r,n,l,m. Recordando la sección del 3.6 devuelve una tupla con los parámetros de psi,que es una arreglo de numpy, el r, que es un arreglo de numpy, el n,l,m son escalares enteros.

plot_wf_2d() con objeto

```
from quplots import electron,plots
p=plots()
e1=electron(3,1,0,5)
```

```
p.plot_wf_2d(e1)
```

plot_wf_2d() sin objeto

```
from quplots import electron, plots
p=plots()
e1=electron(3,1,0,5)
y,r,n,l,m=e1.compute\_wavefunction\_2D()
p.plot_wf_2d(None, y, r, n, l, m,
         levels=10,
         cmap='viridis',
         linewidths=1.5,
         alpha=0.8)
```

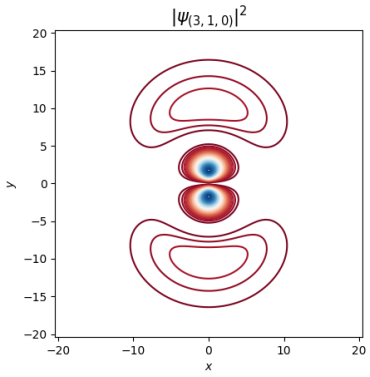

Figura 7: Gráfico con el objeto `electron`

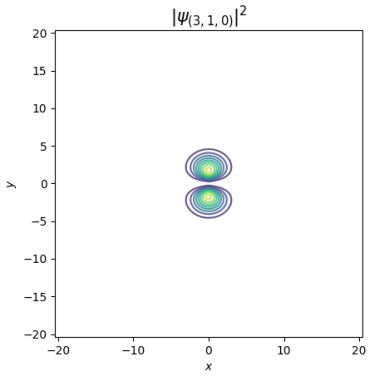

Figura 8: Gráfico sin el objeto

6.5. plot_wf_3d(elect,**kwargs_plot)

La función recibe el objeto elect de la clase electron, y estilos personalizables para gráficas isosurfaces de plotly. Puede encontrar un anexo de estos estilos en `https://plotly.com/python/reference/isosurface/`

plot_wf_3d()

```
    from quplots import electron, plots
    p=plots()
    e1=electron(3,1,0,5)
    p.plot_wf_3d(
    e1,
    colorscale='Plotly3',
    opacity=0.7,
    surface_count=4,
    reversescale=True,
    caps=dict(x_show=False, y_show=False, z_show=False),
    showscale=False,
    lighting=dict(
```

```
        ambient=0.1,
        diffuse=0.4,
        specular=1.0,   # alto brillo
        roughness=0.05, # muy pulido
        fresnel=0.2
    ),
    lightposition=dict(
        x=200,
        y=100,
        z=0
    ),
    width=700,   # ancho
    height=700   # alto
)
```

Figura 9: Función de onda en 3 dimensiones con estilos de plotly

6.6. plot_sp(**kwargs)

Función que solamente requiere los estilos personalizados de plotly, `https://plotly.com/python/reference/isosurface/`, los puedes visualizar aquí

Hibridación sp

```
from quplots import plots
p = plots()
p.plot_sp(
    colorscale='Viridis',
    reversescale=True,
    opacity=0.6,
    surface_count=5,
    caps=dict(x_show=False, y_show=False, z_show=False),
    lighting=dict(ambient=0.4, diffuse=0.9, specular=0.5, roughness=0.9,
    ↪  fresnel=0.1),
    lightposition=dict(x=100, y=200, z=0),
```

```
    showscale=True,
    name="Orbital"
)
```

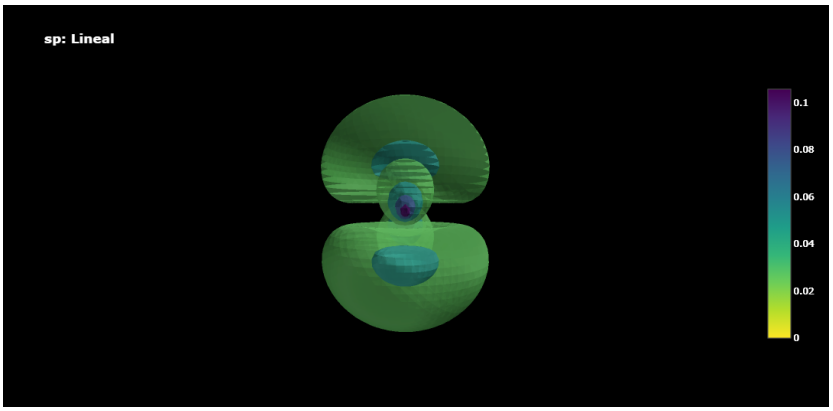

Figura 10: Gráfica hibridación sp (lineal)

6.7. `plot_sp2(**kwargs)`

Función que solamente requiere los estilos personalizados de plotly.

Hibridación sp^2

```
from quplots import plots
p = plots()
p.plot_sp2(colorscale='Plasma',
           reversescale=False,
           opacity=0.6,
           surface_count=5)
```

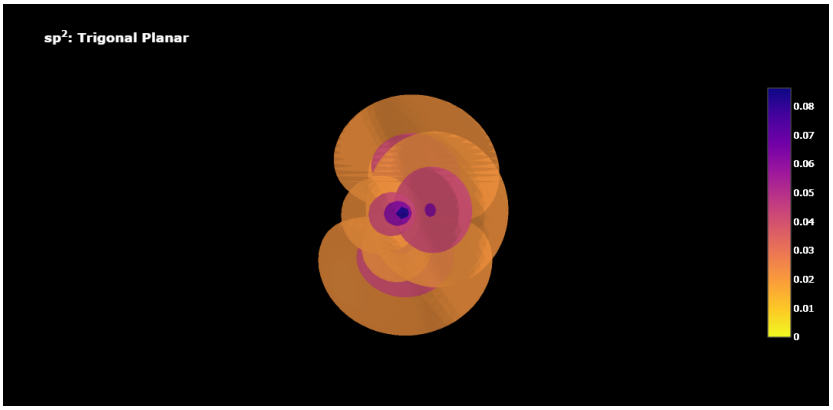

Figura 11: Gráfica hibridación sp^2 (trigonal planar)

6.8. `plot_sp3(**kwargs)`

Función que solamente requiere los estilos personalizados de plotly.

Hibridación sp^3

```
from quplots import plots
p = plots()
p.plot_sp3(colorscale='Cividis',
           reversescale=False,
           opacity=0.6,
           surface_count=5)
```

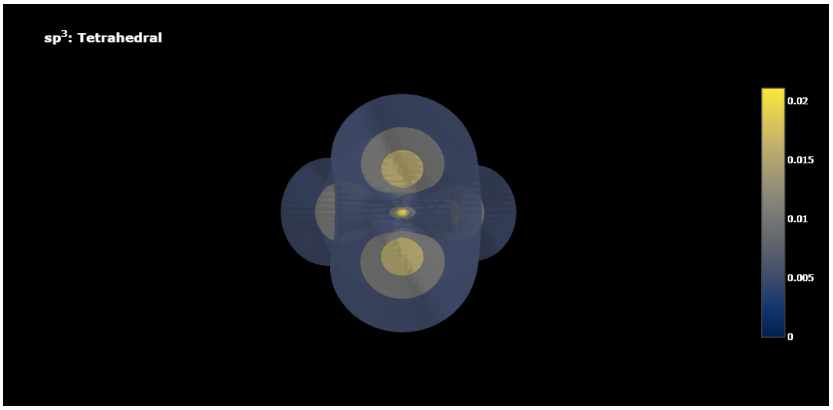

Figura 12: Gráfica hibridación sp^3 (tetrahedral)

6.9. `plot_sp2d(**kwargs)`

Función que solamente requiere los estilos personalizados de plotly.

Hibridación sp^2d

```
from quplots import plots
p = plots()
p.plot_sp2d(colorscale='Inferno',
            reversescale=False,
            opacity=0.6,
            surface_count=5)
```

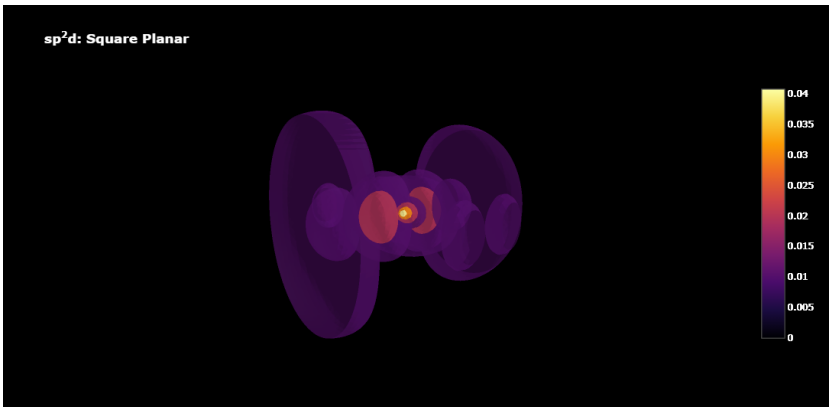

Figura 13: Gráfica hibridación sp^2d (square planar)

6.10. `plot_sp3d(**kwargs)`

Función que solamente requiere los estilos personalizados de plotly.

Hibridación sp^3d

```
from quplots import plots
p = plots()
p.plot_sp3d(colorscale='Magma',
            reversescale=False,
            opacity=0.6,
            surface_count=5)
```

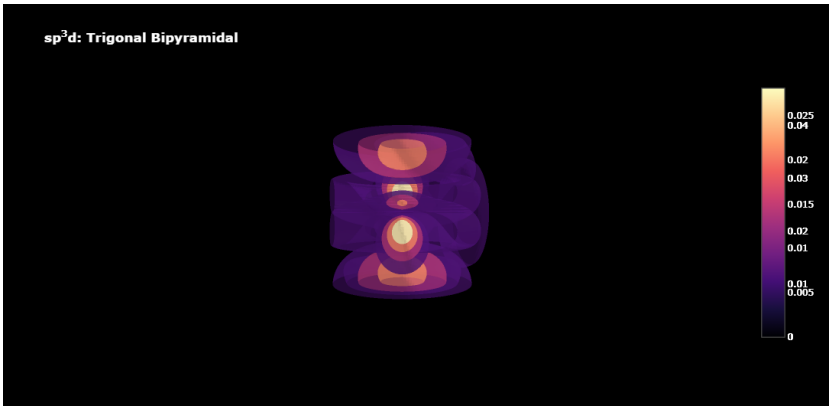

Figura 14: Gráfica hibridación sp^3d (trigonal bipyramidal)

6.11. `plot_sp3d2(**kwargs)`

Función que solamente requiere los estilos personalizados de plotly. Dentro de todas las funciones podemos un código de ejemplo

Hibridación sp^3d^2

```
from quplots import plots
p = plots()
p.plot_sp3d2(colorscale='Turbo',
             reversescale=False,
             opacity=0.6,
             surface_count=5)
```

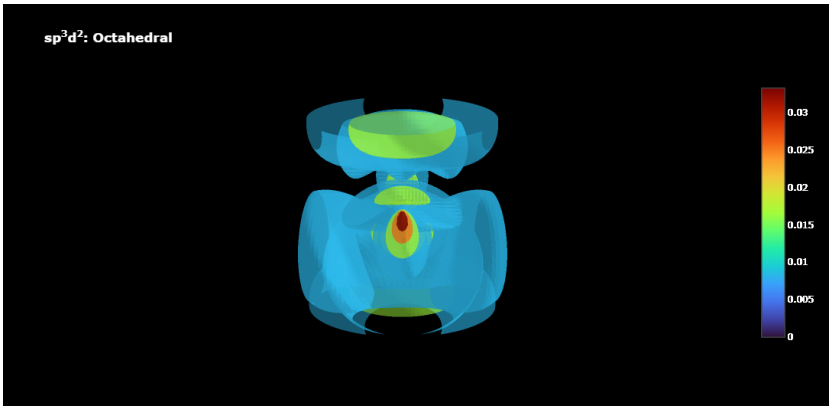

Figura 15: Gráfica hibridación sp^3d^2 (octahedral)

7. Material didáctico

7.1. Notebooks en Colab

Solo haz click para probar la librería sin instalar nada en tu computadora.
`https://drive.google.com/drive/folders/1unziXZhdp8siNKy9K0BNCC4TV_Gu60RW?usp=sharing`

7.2. Sitio Web

Próximamente...

7.3. Repositorio del proyecto

`https://github.com/JoseAdrianRodriguezGonzalez/quplots`

7.4. Videos

`https://youtube.com/@quplots?si=KterFMrhEb8fjEYS`

Referencias

[1] Peter William Atkins and Ronald S. Friedman. *Molecular Quantum Mechanics*. Oxford University Press, Oxford, 5th edition, 2011.

[2] LibreTexts. Coordenadas esféricas. https://espanol.libretexts.org/Quimica/Qun.d.

[3] A. Saputra, R. C. Lorentz, F. Suyono, Noor, D. M. S. Chansyanah, and K. Budi. Visualizing three dimensional hybrid atomic orbitals using winplot: An application for student self instruction. pp. 1–2.