# Práctica 1 - JUAN MANUEL RUIZ MUÑOZ

March 26, 2024

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import cmath as cm
```

Para comenzar, definimos las solución de la parte polar:

[3]:
```
def phi2(m,x):
    return np.exp(1j*m*x)
```

[4]:
```
alpha=np.linspace(0,2*np.pi,360)
func=phi2(2,alpha)
real=func.real
compleja=func.imag
modulo=abs(func)
```

La solución a la ecuación de Schrödinger para el átomo de hidrógeno se puede separar en partes radial, azimutal (o angular) y temporal. La parte temporal de la solución es una función de onda que describe cómo cambia el estado del sistema con el tiempo.

Si añadimos la parte temporal a la solución polar, obtendremos una solución completa a la ecuación de Schrödinger dependiente del tiempo. Esta solución describe cómo cambia el estado del átomo de hidrógeno tanto en el espacio como en el tiempo.

La parte temporal de la solución es una función exponencial compleja de la forma `exp(-iEt/h)`, donde `E` es la energía del estado, `h` es la constante de Planck y `t` es el tiempo. Esta función describe cómo oscila la fase de la función de onda con el tiempo.

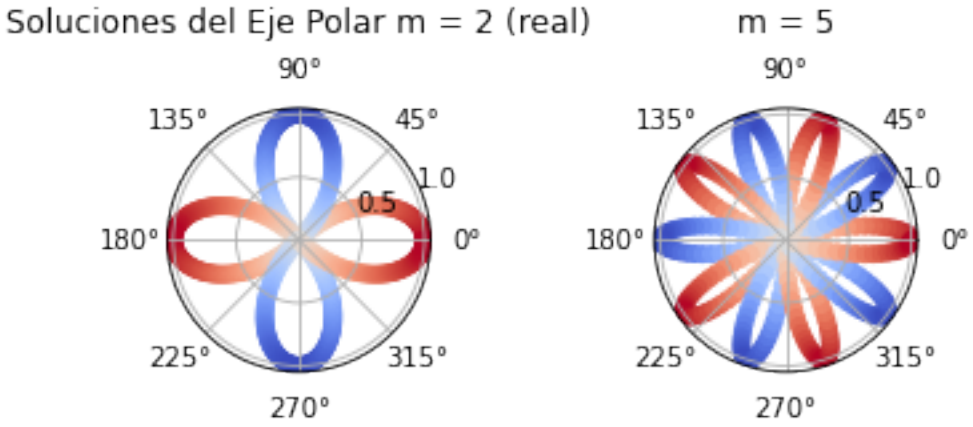
Es importante tener en cuenta que, aunque la parte temporal añade una dimensión adicional a la solución, no cambia la forma de los orbitales atómicos, que están determinados por las partes radial y azimutal de la solución. Sin embargo, puede afectar a las propiedades observables del sistema, como la densidad de probabilidad del electrón.

En el caso de implementar la parte temporal a las soluciones que obtenemos más abajo, lo que se observaría sería que las soluciones rotan, como si de unas aspas de avión se tratara.

Representamos primero la parte real de la solución polar:

[5]:
```
#Hacemos la representación en coordenadas polares
ax=plt.subplot(221,projection="polar")
#Parte real
r=real
```

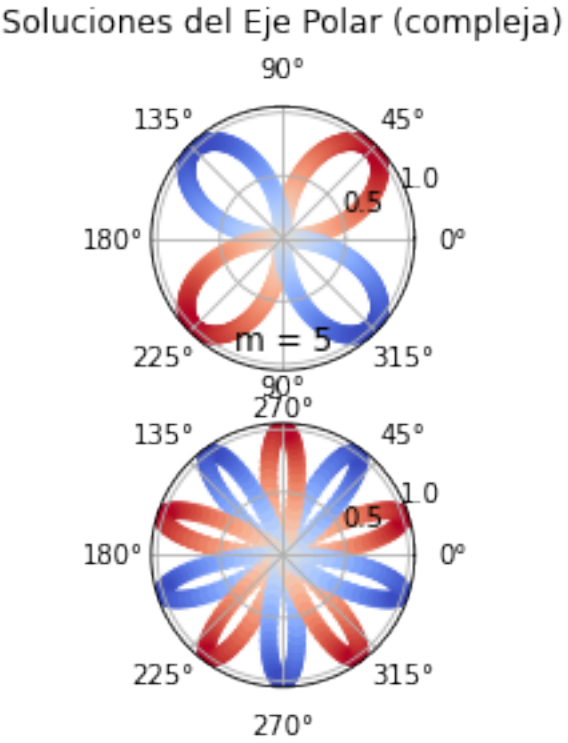
```
scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)

plt.scatter(alpha,np.abs(r),c=colors)
ax.grid(True)
ax.set_title("Soluciones del Eje Polar m = 2 (real)", va="bottom")

#pintamos para otro valor de m
func=phi2(5,alpha)
real=func.real
r=real
scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)
ax=plt.subplot(222,projection="polar")
plt.scatter(alpha,np.abs(r),c=colors)
ax.set_title("m = 5", va="bottom")
ax.grid(True)
```




Ahora la parte compleja:

[6]:
```
#Parte compleja
r=compleja

scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)

ax=plt.subplot(222,projection="polar")
plt.scatter(alpha,np.abs(r),c=colors)
ax.grid(True)
ax.set_title("Soluciones del Eje Polar (compleja)", va="bottom")
```

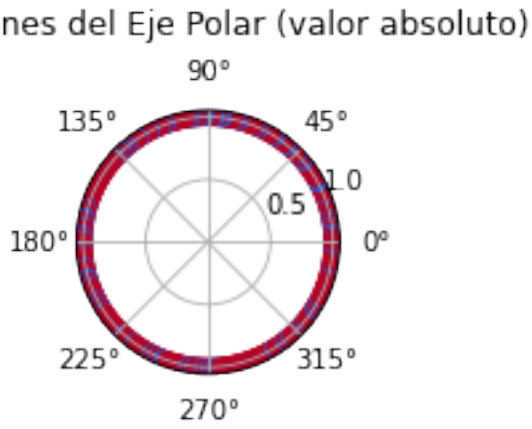
```
#pintamos para otro valor de m
func=phi2(5,alpha)
compleja=func.imag
r=compleja
scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)
ax=plt.subplot(224,projection="polar")
plt.scatter(alpha,np.abs(r),c=colors)
ax.set_title("m = 5", va="bottom")
ax.grid(True)
```




Y por último, representamos el valor absoluto:

[7]:
```
#Módulo
r=modulo

scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)

ax=plt.subplot(223,projection="polar")
plt.scatter(alpha,np.abs(r),c=colors)
ax.grid(True)
ax.set_title("Soluciones del Eje Polar (valor absoluto)", va="bottom")
```

[7]: `Text(0.5, 1.0, 'Soluciones del Eje Polar (valor absoluto)')`




La parte azimutal de la solución de la ecuación de Schrödinger para el átomo de hidrógeno está representada por las funciones armónicas esféricas. Estas funciones dependen de los números cuánticos `l` y `m`, y determinan la forma y orientación de los orbitales atómicos. El número cuántico `l` determina la forma general del orbital (por ejemplo, `s`, `p`, `d`, etc.), mientras que `m` determina su orientación en el espacio.

[9]:
```
#Parte de Legendre

def recursive_deriv(f,k):
    def compute(x,dx=0.01):
        return 0.5*(f(x+dx)-f(x-dx))/dx
    if k==0:
        return f
    if k==1:
        return compute
    else:
        return recursive_deriv(compute,k-1)
```

[10]:
```
def legendre(x,l):
    def f(x):
        a=(x**2-1)**l
        return a
    df = recursive_deriv(f,l)
    p_l= 1.0/(2**l*np.math.factorial(l))*df(x)
    return p_l
```

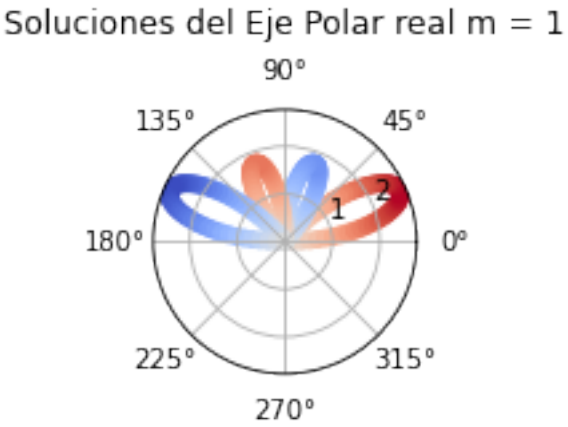
[11]:
```
def asociados_legendre(m,l,theta):
    def g(x):
        return legendre(x,l)
    dg=recursive_deriv(g,abs(m))
```

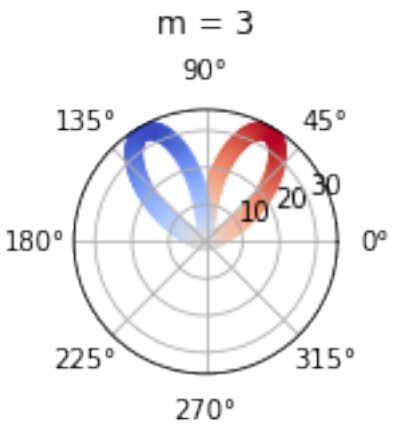
```
    p_lm=(1-theta**2)**(0.5*abs(m))*dg(theta)
    return p_lm
```

Una vez tenemos definidos los polinomios de Legendre y la derivada recursiva construimos la función azimutal que llamará a éstas a su vez.

[12]:
```
def azimutal(m,l,theta):
    sol=asociados_legendre(m,l,np.cos(theta))
    return sol
```

[13]:
```
theta=np.linspace(0,np.pi,180)
r=azimutal(1,4,theta)
scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)
ax=plt.subplot(224,projection="polar")
plt.scatter(theta,np.abs(r),c=colors)
ax.grid(True)
ax.set_title("Soluciones del Eje Polar real m = 1", va="bottom")
plt.show()

#pintamos para otro valor de m
r=azimutal(3,4,theta)
scaled_r=(r-r.min())/r.ptp()
colors=plt.cm.coolwarm(scaled_r)
ax=plt.subplot(224,projection="polar")
plt.scatter(theta,np.abs(r),c=colors)
ax.set_title("m = 3", va="bottom")
ax.grid(True)
plt.show()
```






Cambiar los valores de los números cuánticos `l` y `m` afectará a las soluciones de la ecuación de Schrödinger para el átomo de hidrógeno de las siguientes maneras:

- `l` (número cuántico azimutal o angular): Este número cuántico determina la forma del orbital atómico. Por ejemplo, si `l` es 0, el orbital es una esfera (orbital s). Si `l` es 1, el orbital tiene forma de mancuerna (orbital p). Si `l` es 2, el orbital tiene forma de rosquilla con un lóbulo en cada extremo (orbital d). Así, cambiar `l` cambiará la forma del orbital atómico.
- `m` (número cuántico magnético): Este número cuántico determina la orientación del orbital en el espacio. Por ejemplo, para un orbital p (`l` = 1), `m` puede ser -1, 0 o 1, correspondiendo a orientaciones a lo largo de los ejes x, y, z respectivamente. Así, cambiar `m` cambiará la orientación del orbital atómico.

Es importante tener en cuenta que estos cambios en `l` y `m` no afectarán a la energía del estado en el átomo de hidrógeno (en ausencia de un campo magnético), ya que la energía sólo depende del número cuántico principal `n`. Sin embargo, en presencia de un campo magnético, los estados con diferentes valores de `m` tendrán diferentes energías debido al efecto Zeeman.

[14]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.colors as mcolors
```

[15]:
```
theta, phi = np.linspace(0,np.pi, 200), np.linspace(0, 2*np.pi, 40)
THETA, PHI = np.meshgrid(theta, phi)
```

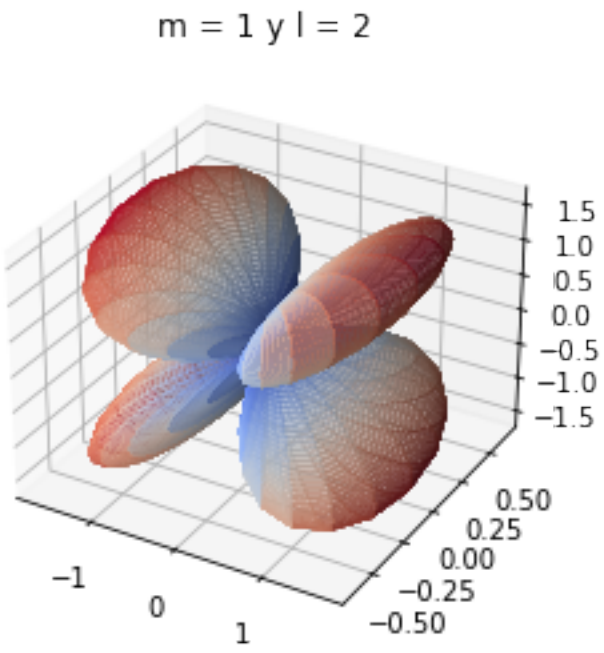
Definimos la parte armónica como el producto de la parte polar y la parte azimutal:

[16]:
```
def armonicos(alpha,theta,l,m):
    return azimutal(m,l,theta)*phi2(m,alpha)
```

[21]:
```
m=1
l=2

#R = (np.absolute(Y_l_m(THETA,PHI,l,m)))**2
```

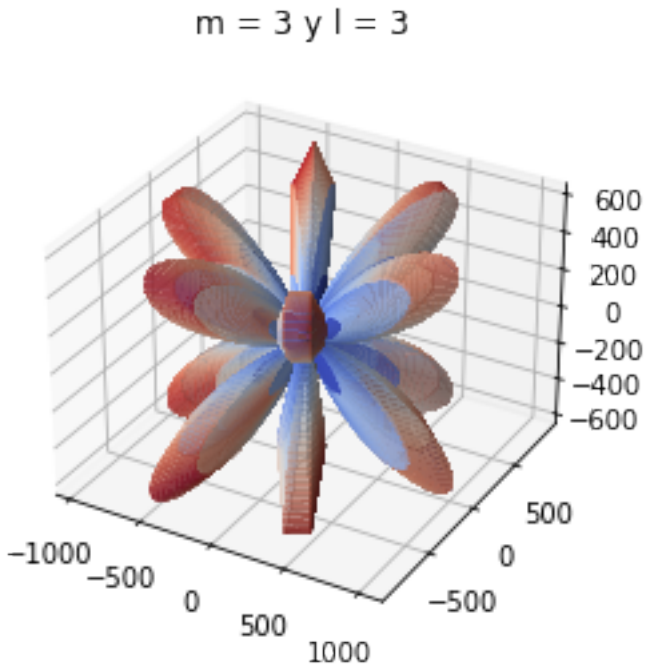
```
R = (np.real(armonicos(PHI,THETA,l,m)))**2#+(np.
 ↪imag(armonicos(PHI,THETA,l,m)))**2#Pasamos a coordenadas Cartesinas
X = R*np.sin(THETA) * np.cos(PHI)
Y = R*np.sin(THETA) * np.sin(PHI)
Z = R * np.cos(THETA)
```

[22]:
```
#Representamos. El módulo de la Función de onda aparece cómo R y un color
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cmap = plt.get_cmap('coolwarm')
norm = mcolors.Normalize(vmin=R.min(), vmax=R.max())
plot = ax.plot_surface(
X, Y, Z, rstride=1, cstride=1, facecolors=cmap(norm(R)),
linewidth=0, antialiased=False, alpha=.4)
ax.set_title("m = 1 y l = 2", va="bottom")
plt.show()
```



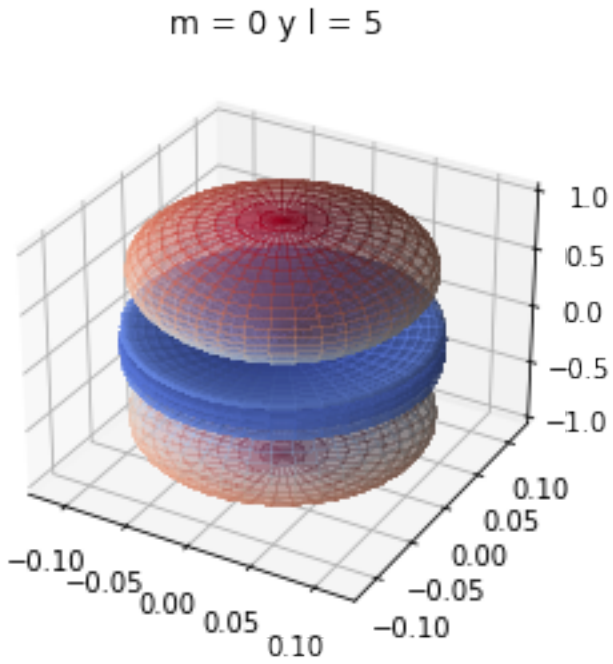

[23]:
```
#Definamos ahora otros valores de m y l

m=3
l=4

#R = (np.absolute(Y_l_m(THETA,PHI,l,m)))**2
R = (np.real(armonicos(PHI,THETA,l,m)))**2#+(np.
 ↪imag(armonicos(PHI,THETA,l,m)))**2#Pasamos a coordenadas Cartesinas
```

```
X = R*np.sin(THETA) * np.cos(PHI)
Y = R*np.sin(THETA) * np.sin(PHI)
Z = R * np.cos(THETA)

#Representamos. El módulo de la Función de onda aparece cómo R y un color
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cmap = plt.get_cmap('coolwarm')
norm = mcolors.Normalize(vmin=R.min(), vmax=R.max())
plot = ax.plot_surface(
X, Y, Z, rstride=1, cstride=1, facecolors=cmap(norm(R)),
linewidth=0, antialiased=False, alpha=.4)
ax.set_title("m = 3 y l = 3", va="bottom")
plt.show()
```




[24]:
```
#probamos con otros valores
m=0
l=5

#R = (np.absolute(Y_l_m(THETA,PHI,l,m)))**2
R = (np.real(armonicos(PHI,THETA,l,m)))**2#+(np.
 ↪imag(armonicos(PHI,THETA,l,m)))**2#Pasamos a coordenadas Cartesinas
X = R*np.sin(THETA) * np.cos(PHI)
Y = R*np.sin(THETA) * np.sin(PHI)
```

```
Z = R * np.cos(THETA)

#Representamos. El módulo de la Función de onda aparece cómo R y un color
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cmap = plt.get_cmap('coolwarm')
norm = mcolors.Normalize(vmin=R.min(), vmax=R.max())
plot = ax.plot_surface(
X, Y, Z, rstride=1, cstride=1, facecolors=cmap(norm(R)),
linewidth=0, antialiased=False, alpha=.4)
ax.set_title("m = 0 y l = 5", va="bottom")
plt.show()
```

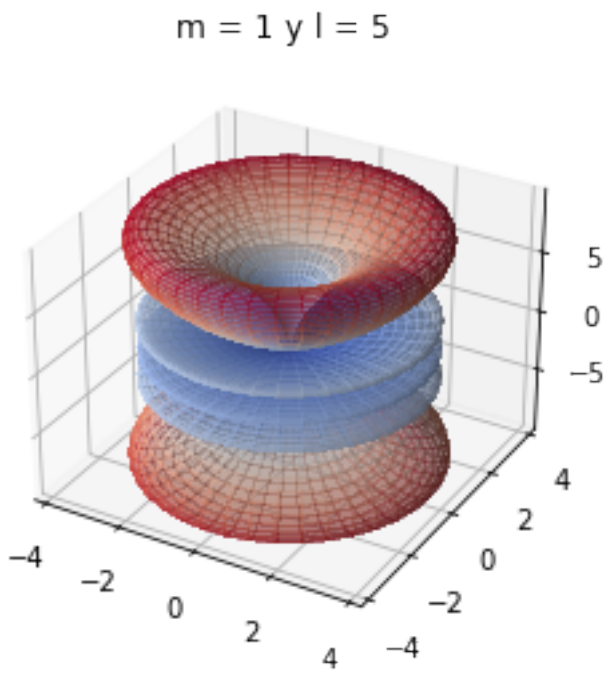



Al multiplicar la solución polar y la solución azimutal, se obtiene una descripción completa de cómo la función de onda del electrón varía con los ángulos  y .

Podemos representar el módulo de R:

[48]:
```
#para m=1 y l=1

m=1
l=5

R = (np.absolute(armonicos(PHI,THETA,l,m)))**2
X = R*np.sin(THETA) * np.cos(PHI)
Y = R*np.sin(THETA) * np.sin(PHI)
```

```
Z = R * np.cos(THETA)

#Representamos. El módulo de la Función de onda aparece cómo R y un color

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cmap = plt.get_cmap('coolwarm')
norm = mcolors.Normalize(vmin=R.min(), vmax=R.max())
plot = ax.plot_surface(
X, Y, Z, rstride=1, cstride=1, facecolors=cmap(norm(R)),
linewidth=0, antialiased=False, alpha=.4)
ax.set_title("m = 1 y l = 5", va="bottom")
plt.show()
```




Vamos ahora con la solución radial. Definimos las funciones que nos serán necesarias para ello:

[31]:
```
def laguerre(x,q):
    def h(x):
        return np.exp(-x)*x**q
    dh=recursive_deriv(h,q)
    L_q=np.exp(x)*dh(x)
    return L_q
```

[32]:
```
def asociados_laguerre(x,q,p):
    def i(x):
```

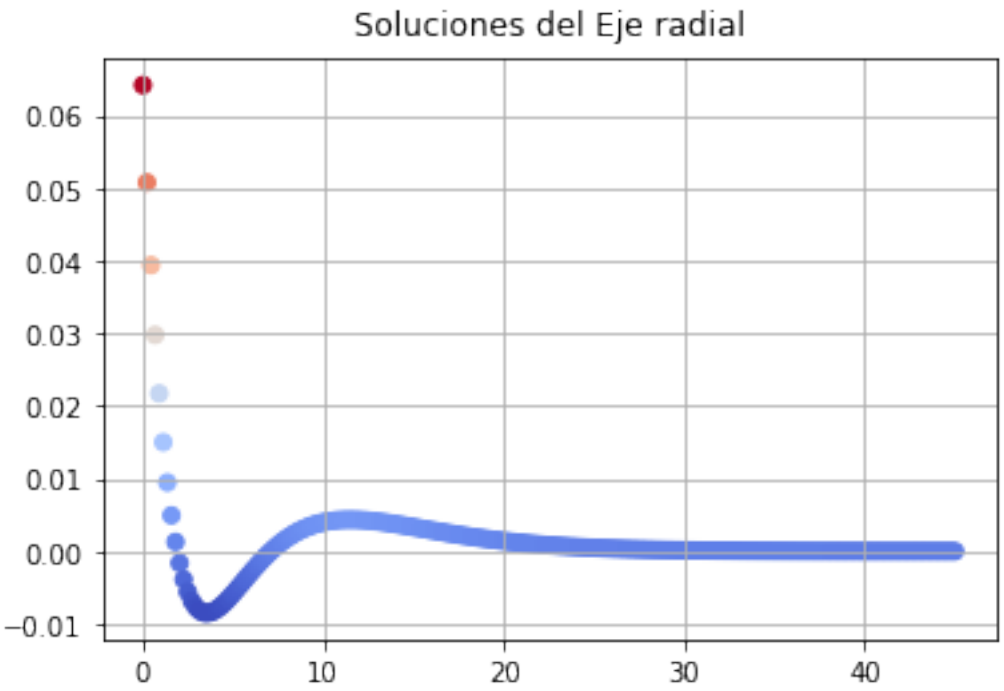
```
        return laguerre(x,q)
    di=recursive_deriv(i,p)
    L_qp=(-1)**p*di(x)
    return L_qp
```

[33]:
```
def upsilon(rho,n,l):
    return asociados_laguerre(2*rho,n+l,2*l+1)
```

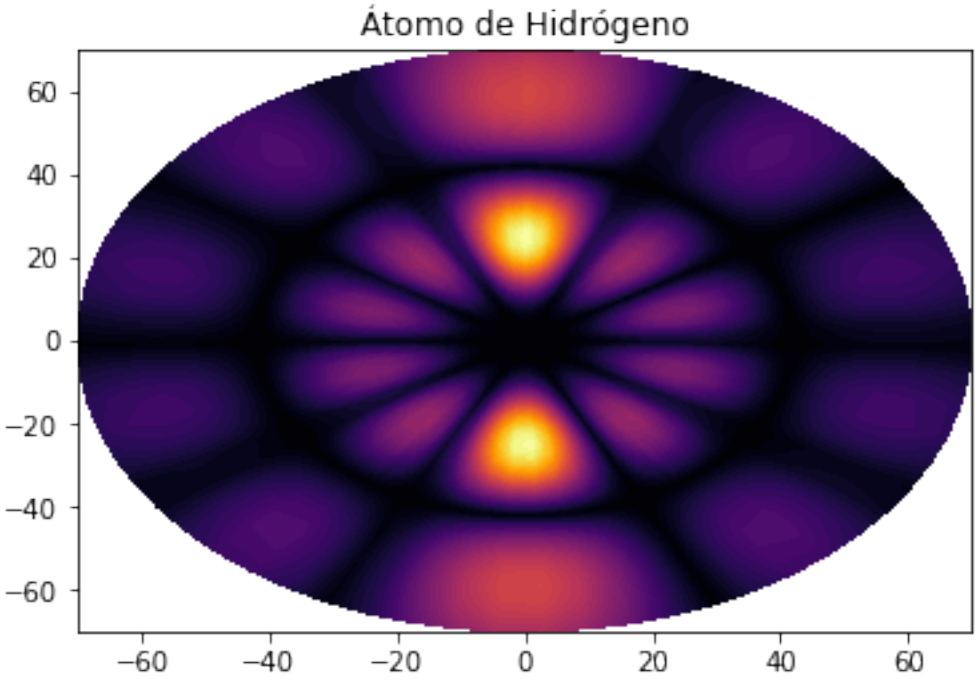
[34]:
```
def funcradial(rho,n,l):
    radial=(1/(rho*n))*rho**(l+1)*np.exp(-rho)*upsilon(rho,n,l)
    return radial
```

[35]:
```
from scipy.special import assoc_laguerre
```

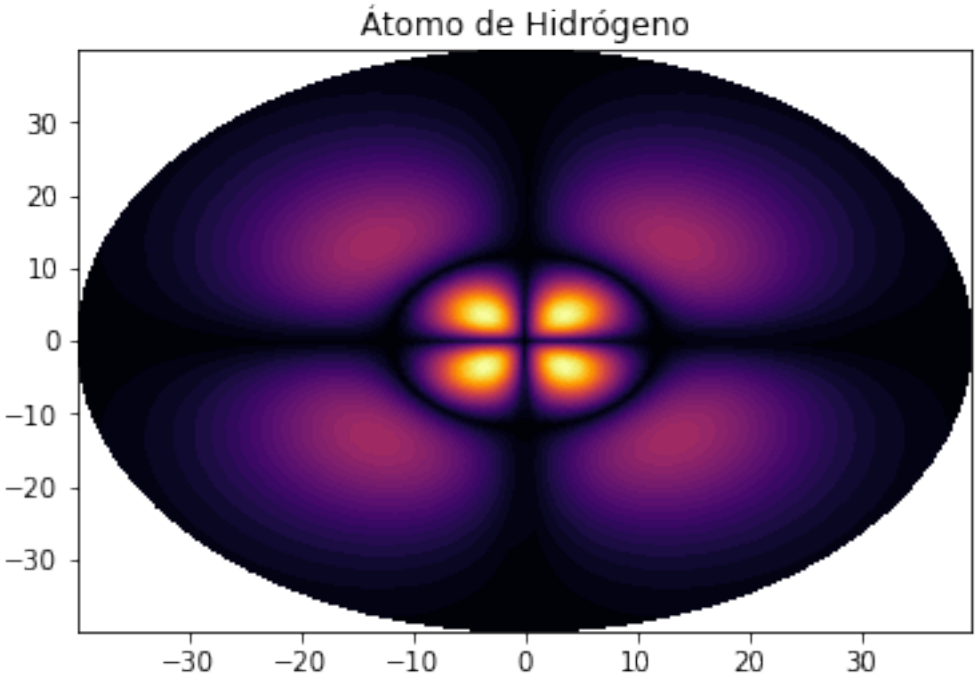
[39]:
```
def radial(x, l, n):
    raiz = np.sqrt((2/(n))**3*(np.math.factorial(n-l-1)/(2*n*(np.math.
 ↪factorial(n+l))**3)))
    return raiz*np.exp(-x/n)*(2*x/n)**l*assoc_laguerre(2*x/n, n-l-1, 2*l+1)

x=np.linspace(0.0001,45,200)
n=3
l=0
f_r=radial(x,l,n)
#Hacemos la representación en coordenadas cartesianas
ax = plt.subplot(111)

#Escalamos la gama de colores
scaled_fr = (f_r - f_r.min()) / f_r.ptp()
colors = plt.cm.coolwarm(scaled_fr)

#Representamos
plt.scatter(x,f_r, c=colors)
ax.grid(True)
ax.set_title("Soluciones del Eje radial", va='bottom')
plt.show()
```




[43]:
```
#Unión de ambas partes
m=0
n=7
l=5
r = np.arange(0, 10*n, 0.1)
th= np.arange(0, 2.1*np.pi, 0.1)
R, TH = np.meshgrid(r, th)
VAL=(np.abs(radial(r,l,n)*armonicos(0,TH,l,m)))
X = R*np.cos(TH)
Y = R*np.sin(TH)

# Set colour interpolation and colour map
colorinterpolation = 50
colourMap = plt.cm.inferno

fig = plt.figure()
ax = fig.add_subplot(111)

# Configure the contour
plt.contourf(Y, X, VAL, colorinterpolation, cmap=colourMap)
plt.title("Átomo de Hidrógeno")
plt.show()
```




[47]:
```
#Unión de ambas partes
m=1
n=4
l=2
r = np.arange(0, 10*n, 0.1)
th= np.arange(0, 2.1*np.pi, 0.1)
R, TH = np.meshgrid(r, th)
VAL=(np.abs(radial(r,l,n)*armonicos(0,TH,l,m)))
X = R*np.cos(TH)
Y = R*np.sin(TH)

# Set colour interpolation and colour map
colorinterpolation = 50
colourMap = plt.cm.inferno

fig = plt.figure()
ax = fig.add_subplot(111)

# Configure the contour
plt.contourf(Y, X, VAL, colorinterpolation, cmap=colourMap)
plt.title("Átomo de Hidrógeno")
plt.show()
```


Átomo de Hidrógeno
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30