PRÁCTICA 1

MIREIA SERRANO BELTRÁ

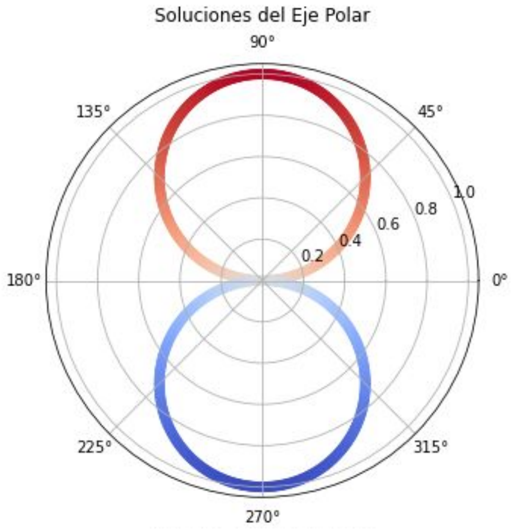
Solución de la Ecuación de Schrödinger para el Átomo de Hidrógeno

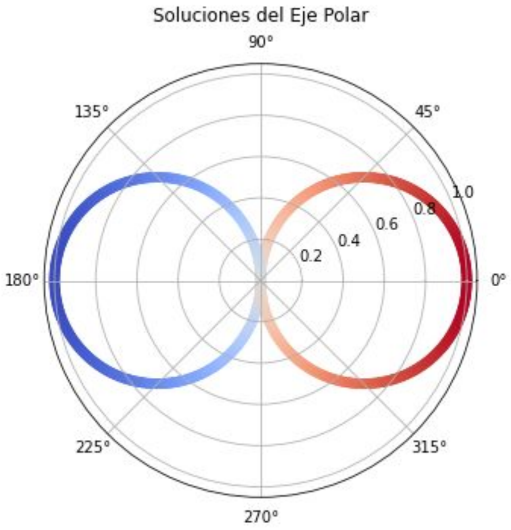
1. REPRESENTACIÓN SOLUCIÓN POLAR

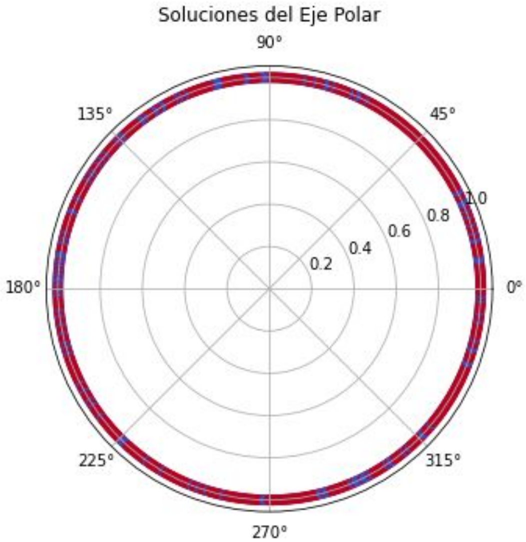
Damos valores a theta y representamos la parte imaginaria, real y el módulo.

```
m = 1

theta=np.arange(0,2*np.pi,0.01)

Phi = np. arange (0, 2*np.pi , 0.01)

f_th = np.exp (1j*m*Phi)

ri = f_th.imag

ra = abs(f_th)

rr = f_th.real
```

1. REPRESENTACIÓN SOLUCIÓN POLAR







2. REPRESENTACIÓN PARTE AZIMUTAL

Definimos una función recursiva para calcular los polinomios de legendre.

```
m=1
l=2
#funcion para calcular derivadas de orden n
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)
```

```
#polinomios de legendre
def Pl(x,l):
    def f(x):
        return (x**2-1)**l
    df =  recursive_deriv(f,l)
    return 1.0/(2**l*np.math.factorial(l))*df(x)
def Plm(x,l,m):
    def f(x):
        return Pl(x,l)
    df = recursive_deriv(f, np.abs(m))
    return (1-x**2)**( np.abs(m)/2)*df(x)
```

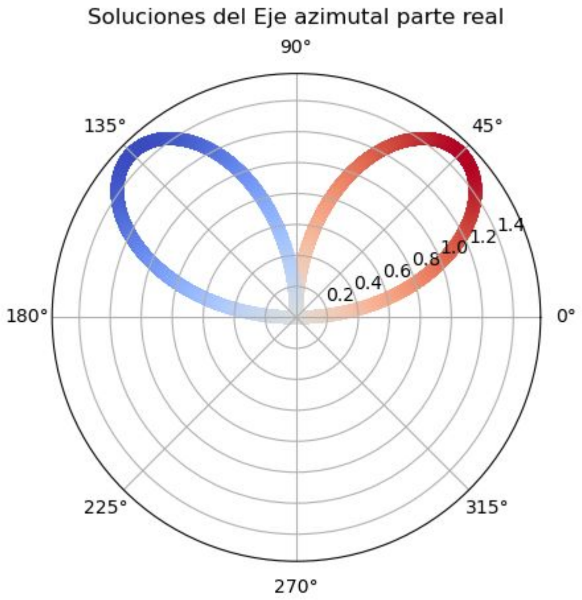
2. REPRESENTACIÓN PARTE AZIMUTAL

Le damos valores a theta y representamos la parte real.

```
theta = np.arange(0, np.pi , 0.0001)
#Solucion generica en complejos
f_ph = Plm(np.cos(theta), l, m)
#Grafica parte real
r = f_ph.real
#Hacemos la representacion en coordenadas Polares
ax = plt.subplot(111,projection='polar')
#Escalamos la gama de colores
scaled_r = (r - r.min()) / r.ptp()
colors = plt.cm.coolwarm(scaled_r)
#Representamos
plt.scatter(theta,np.abs(r), c=colors)
ax.grid(True)
ax.set_title("Soluciones del Eje azimutal parte real", va='bottom')
plt.show()
```

2. REPRESENTACIÓN PARTE AZIMUTAL

Solución obtenida para m = 1 y l = 2



3. ARMÓNICOS ESFÉRICOS 3D

Las funciones de onda angulares ya normalizadas se denominan armónicos esféricos :

$$\boxed{Y_l^m(\theta,\phi) = (-1)^m \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}\, e^{im\phi} P_l^m(\cos\theta)}$$

Implementamos las siguientes funciones en python para la representación en 3D :

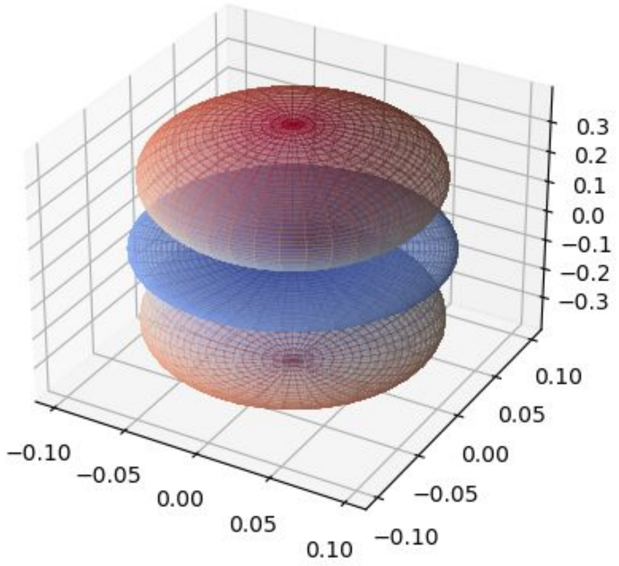
```
def normalizacion(l,m):
    a=(-1)**m
    s=(2*l+1)*np.math.factorial(l-np.abs(m))
    t=4*np.pi*np.math.factorial(l+np.abs(m))
    b=np.sqrt(s/t)
    return a*b
```

```
def armonicos(th,ph,l,m):
    norma=normalizacion(l,m)
    f_th=np.exp(1j*m*th)
    f_ph=Plm(np.cos(ph),l,m)
    return norma*f_th*f_ph
```

3. ARMÓNICOS ESFÉRICOS 3D

Con este código obtenemos la siguiente gráfica para l=2 y m=0 :

```
import matplotlib.colors as mcolors
#Ahora Vamos a Calcular para todos los ángulos el valor de la función de onda angular.
theta, phi = np.linspace(0,np.pi, 200), np.linspace(0, 2*np.pi, 40)
THETA, PHI = np.meshgrid(theta, phi)
 #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)
plt.show()
```



4. RESOLUCIÓN ECUACIÓN RADIAL

Debemos calcular el polinomio asociado de Laguerre:

$$L_{q-p}^{p}(x) \equiv (-1)^p \left(\frac{d}{dx}\right)^p L_q(x)$$

Y el polinomio de Laguerre :

$$L_q(x) \equiv e^x \left(\frac{d}{dx}\right)^q \left(e^{-x} x^q\right)$$

Función radial del átomo de hidrógeno :

$$\boxed{\psi_{nlm} = \sqrt{\left(\frac{2}{na}\right)^3 \frac{(n-l-1)!}{2n[(n+l)!]^3}}\, e^{-r/na} \left(\frac{2r}{na}\right)^l L_{n-l-1}^{2l+1}\left(\frac{2r}{na}\right) Y_l^m(\theta, \phi).}$$

4. RESOLUCIÓN ECUACIÓN RADIAL

Implementamos el siguiente código :

```
#polinomio de Laguerre
def Lq(x, q):
    def f(x):
        return np.exp(-x)*x**q
    df = recursive_deriv (f,q)
    return np.exp(x)*df(x)

#polinomio asociado de Laguerre
def Lqp(x, p, q_p):
    def f(x):
        return Lq(x, p+q_p)
    df = recursive_deriv(f,p)
    return ((-1)**p)*df(x)
```

```
def R(n,l,m,r):
    def f(x):
        return Lqp(x, 2*l+1, n-l-1)
    rho = 2*r/n
    a = np.math.factorial(n-l-1)
    b = 2*n*(( np.math.factorial(n+l))**3)
    c = (2/n)**3
    norm = np.sqrt(c*a/b)
    return norm*np.exp(-rho/2)*((rho)**l)*f(rho)
```

4. RESOLUCIÓN ECUACIÓN RADIAL

Implementamos el siguiente código :

```
r = np. arange (0, 8*n**1.5 , 0.001)
f_r = R(n,l,m,r)

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

4. RESOLUCIÓN ECUACIÓN RADIAL

Obtenemos las siguientes gráficas: