Computing Schrödinger's Equations

Due Date: Week 9 Monday, 19 March, 5 pm

Name: ______________________________ Principal Quantum Number: _______

____

____

Objectives

- Calculate the complete wavefunction solution from the radial and angular solution.
- Plot and visualize the probability density functions and the shapes of these orbitals.

Instructions and

For this assignment, you are required to work in assigned groups of 2 or 3 and **submit one set of answers per group**. The assignment is split into three sections:
Part A: Theory (Chemistry) Segment
Part B: Coding (Digital World) Segment
Part C: In-Depth Theory Segment (Bonus)

In this assignment, choose the principal quantum number *n* (3 or 4) that your team will be working on. You will need to complete this assignment for all *l* and *m* (*m* is written as m_l in your Chemistry notes) values for the chosen *n* i.e. if your group chooses to do this assignment for $n = 3$, you are expected to solve for $l = 0, 1, 2$ and $m = 0$, $m = -1, 0, 1$ and $m = -2, -1, 0, 1, 2$.

Weightage

This assignment is worth 5% of the final Chemistry grade and 2% of the final Digital World grade. A bonus of up to 4%, using the base 5% of the final Chemistry grade as an example, is available:

- The highest score attainable is 5% if you attempt $n = 3$.
- If your attempt in solving for all orbitals in $n = 4$ is completely correct, you get an additional 2% on top of the base 5%, such that the highest score attainable is 7%. If a mistake is found, you are entitled to 0% of the bonus score and you will be graded out of the base 5%.

The scoring system, using the base 5% of the final Chemistry grade as an example, is as follows:

	$n = 3$ is chosen	$n = 4$ is chosen
Theory (Part A) and Plots (Part B)	5%	7% (bonus: 2%)
In-Depth Theory (Part C)	7% (bonus: 2%)	9% (bonus: 4%)

Assignment Submission

You are required to notify your Chemistry instructors of your choice of principal quantum number *n* by the end of Week 6 Friday, 5 pm.

You are required to submit the following to your Chemistry instructors on Week 9 Monday, 5 pm for your chosen principal quantum number *n*.

(i) Answers to the Theory Section (Part A; Chemistry) – step-by-step handwritten working is expected.

(ii) 3D plots and cross-section plots (Part B; Digital World)

Note that:

(iii) If your team chooses $n = 3$, you are required to submit the above items (i) and (ii) for all *m* numbers in $l = 0, 1, 2$.

(iv) If your team chooses $n = 4$, you are required to submit the above items (i) and (ii) for all *m* numbers in $l = 0, 1, 2, 3$ in order to be considered for the bonus points of 2%

(v) If your team chooses to attempt Part C (In-Depth Theory), you are required to submit step-by-step handwritten working and the code for your chosen principal quantum number *n* and all *l* and *m* numbers for that chosen principal quantum number *n*.

References

It is highly recommended that you read References (b) to (g) as it may prove to be very useful for this assignment.

a. Introduction to Quantum Mechanics (Second Edition), David J. Griffths
b. From the time independent Schrödinger's equation in 3D to spherical harmonics, https://www.youtube.com/watch?v=bqYMUDxD3WQ
c. Time independent Schrödinger's equation in 3D radial behavior, https://www.youtube.com/watch?v=Nq72tUEtbfM
d. Matplotlib 2D/3D plotting tutorials, http://matplotlib.org/users/pyplot_tutorial.html, http://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html
e. Python scripting for 3D plotting, http://docs.enthought.com/mayavi/mayavi/mlab.html
f. Hydrogen atom viewer, http://falstad.com/qmatom/
g. Atom in a box simulation tool, http://daugerresearch.com/orbitals/index.shtml#download

Part A: Theory (Chemistry) Segment

Coordinate System

A reference coordinate system is needed to describe position. The spherical coordinate system is used here as we assume that an atom is spherical. The Cartesian and spherical coordinate systems are related by the following mathematical relations:

$$x = r\sin\theta\cos\phi$$
$$y = r\sin\theta\sin\phi$$
$$z = r\cos\theta$$
$$x^2 + y^2 + z^2 = r^2$$

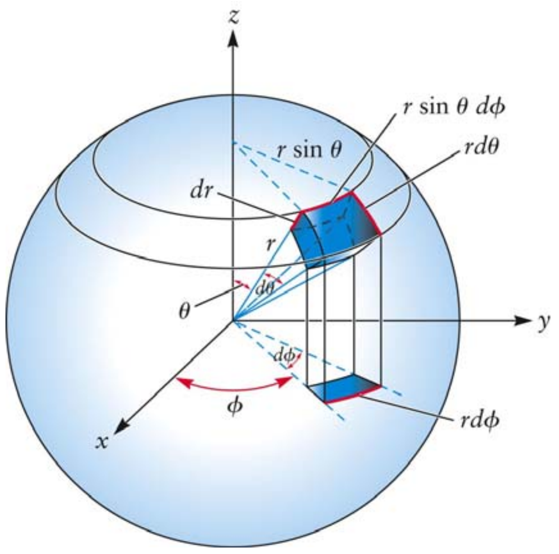


Image: Figure 5-3 of Oxtoby (2012)

Normalized Radial and Normalized Angular Solutions

The solutions of the Schrodinger's wave equation provide important information on the probability distributions for the position of the electron and hence, the shape of atomic orbitals. In this assignment, we will consider the hydrogen atom, a single electron system.

The Schrodinger equation is:

$$-\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi = E\Psi$$

For each combination of quantum numbers (*n*, *l*, *m*), the solution of the Schrodinger equation is a product of a radial part, or the normalized radial solution *R*(*r*), and an angular part, or the normalized angular solution *Y*(*θ*, *ϕ*), of the form:

$$\Psi(r,\theta,\phi) = R(r)\cdot Y(\theta,\phi)$$

The above solution allows for separate examination of the radial and angular contributions to the wavefunction. The normalized radial solution R_l^n, in terms of *r* and *a* (= Bohr radius for hydrogen atom), for different quantum numbers are as follows:

n	*l*	R_l^n
1	0	$R_0^1 = \frac{2}{\sqrt{a^3}}\exp\left(-\frac{r}{a}\right)$
2	0	$R_0^2 = \frac{1}{\sqrt{2}}a^{-3/2}\left(1-\frac{r}{2a}\right)\exp\left(-\frac{r}{2a}\right)$
2	1	$R_1^2 = \frac{1}{\sqrt{24}}a^{-3/2}\left(\frac{r}{a}\right)\exp\left(-\frac{r}{2a}\right)$
3	0	$R_0^3 = \frac{2}{81\sqrt{3}}a^{-3/2}\left[27-18\left(\frac{r}{a}\right)+2\left(\frac{r}{a}\right)^2\right]\exp\left(-\frac{r}{3a}\right)$
3	1	$R_1^3 = \frac{8}{27\sqrt{6}}a^{-3/2}\left(1-\frac{r}{6a}\right)\left(\frac{r}{a}\right)\exp\left(-\frac{r}{3a}\right)$
3	2	$R_2^3 = \frac{4}{81\sqrt{30}}a^{-3/2}\left(\frac{r}{a}\right)^2\exp\left(-\frac{r}{3a}\right)$
4	0	$R_0^4 = \frac{1}{4}a^{-3/2}\left[1-\frac{3}{4}\left(\frac{r}{a}\right)+\frac{1}{8}\left(\frac{r}{a}\right)^2-\frac{1}{192}\left(\frac{r}{a}\right)^3\right]\exp\left(-\frac{r}{4a}\right)$
4	1	$R_1^4 = \frac{\sqrt{5}}{16\sqrt{3}}a^{-3/2}\left(\frac{r}{a}\right)\left[1-\frac{1}{4}\left(\frac{r}{a}\right)+\frac{1}{80}\left(\frac{r}{a}\right)^2\right]\exp\left(-\frac{r}{4a}\right)$
4	2	$R_2^4 = \frac{1}{64\sqrt{5}}a^{-3/2}\left(\frac{r}{a}\right)^2\left[1-\frac{1}{12}\left(\frac{r}{a}\right)\right]\exp\left(-\frac{r}{4a}\right)$
4	3	$R_3^4 = \frac{1}{768\sqrt{35}}a^{-3/2}\left(\frac{r}{a}\right)^3\exp\left(-\frac{r}{4a}\right)$

The normalized angular solution Y_l^m, in terms of *θ* and *ϕ*, for different quantum numbers are as follows:

l	m	Y_l^m
0	0	$Y_0^0 = \sqrt{\frac{1}{4\pi}}$
1	+1	$Y_1^{+1} = -\sqrt{\frac{3}{8\pi}} \sin\theta \exp(i\phi)$
1	0	$Y_1^0 = \sqrt{\frac{3}{4\pi}} \cos\theta$
1	−1	$Y_1^{-1} = \sqrt{\frac{3}{8\pi}} \sin\theta \exp(-i\phi)$
2	+2	$Y_2^{+2} = \sqrt{\frac{15}{32\pi}} \sin^2\theta \exp(i2\phi)$
2	+1	$Y_2^{+1} = -\sqrt{\frac{15}{8\pi}} \cos\theta \sin\theta \exp(i\phi)$
2	0	$Y_2^0 = \sqrt{\frac{5}{16\pi}} (3\cos^2\theta - 1)$
2	−1	$Y_2^{-1} = \sqrt{\frac{15}{8\pi}} \cos\theta \sin\theta \exp(-i\phi)$
2	−2	$Y_2^{-2} = \sqrt{\frac{15}{32\pi}} \sin^2\theta \exp(-i2\phi)$
3	+3	$Y_3^{+3} = -\sqrt{\frac{35}{64\pi}} \sin^3\theta \exp(i3\phi)$
3	+2	$Y_3^{+2} = \sqrt{\frac{105}{32\pi}} \cos\theta \sin^2\theta \exp(i2\phi)$
3	+1	$Y_3^{+1} = -\sqrt{\frac{21}{64\pi}} (\sin\theta)(5\cos^2\theta - 1) \exp(i\phi)$
3	0	$Y_3^0 = \sqrt{\frac{7}{16\pi}} (5\cos^3\theta - 3\cos\theta)$
3	−1	$Y_3^{-1} = \sqrt{\frac{21}{64\pi}} (\sin\theta)(5\cos^2\theta - 1) \exp(-i\phi)$
3	−2	$Y_3^{-2} = \sqrt{\frac{105}{32\pi}} \cos\theta \sin^2\theta \exp(-i2\phi)$
3	−3	$Y_3^{-3} = \sqrt{\frac{35}{64\pi}} \sin^3\theta \exp(-i3\phi)$

Linear Combination

For $l = 0$ or *s* orbitals, the angular portion Y is a constant which means that the wavefunction does not depend on either θ or ϕ. All *s* orbitals are thus spherically symmetric about the nucleus (Figure 1) and the probability of finding an electron depends only on its distance r and not its direction in space.

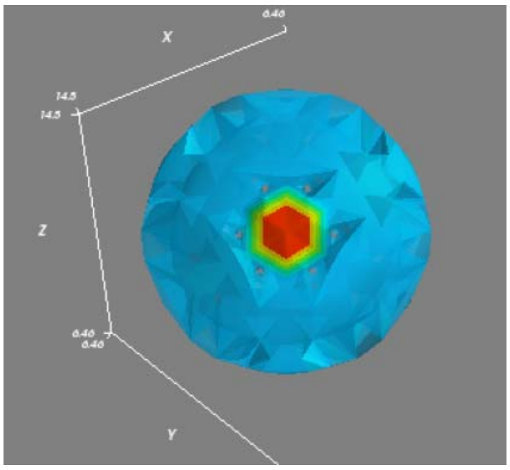


Figure 1: Plot of a *s* orbital

Orbitals with angular quantum numbers that are not equal to 0 are not spherically symmetric. For $l = 1$ or *p* orbitals, there are three different angular wavefunctions which lead to three orbitals with the same shape but different orientation in space ($m = +1, 0, -1$). For $l = 1$, $m = 0$, the wavefunction Y is proportional to cos θ. This orbital has its maximum amplitude along the z-axis where $\theta = 0$ or π and a node in the *xy*-plane, giving a resultant dumbbell-shaped orbital plot. Since the orientation of the *p* orbital is oriented along the *z*-axis, this is thus known as the p_z orbital.

The angular wavefunctions for $l = 1$, $m = +1$ and $l = 1$, $m = -1$, on the other hand, do not have simple geometrical interpretation due to the complex component in the wavefunction; plotting this complex orbital gives a doughnut-shaped orbital (Figure 2). The familiar dumbbell-shaped orbital plots (also known as the plots of real orbitals; Figure 3) for p_x and p_y orbitals require the linear combination (addition and subtraction) of the normalized angular solution of these complex wavefunctions such that:

$$Y_{p_x} = \frac{1}{\sqrt{2}}(Y_1^{-1} - Y_1^{+1})$$
$$Y_{p_y} = \frac{i}{\sqrt{2}}(Y_1^{-1} + Y_1^{+1})$$

To revise complex functions, refer to Math 1 Week 5 Cohort 1 slides.

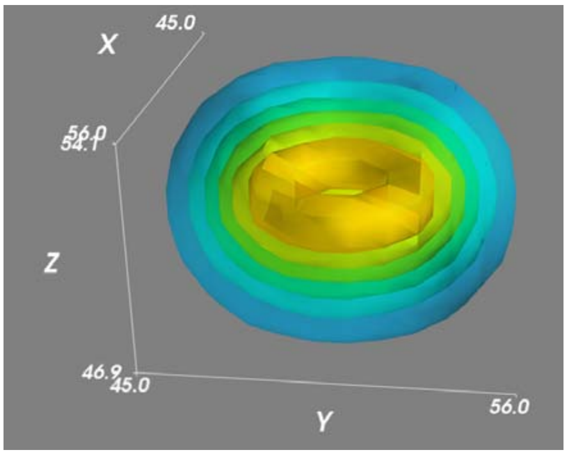


Figure 2: Plot of complex $2p_x$ orbital

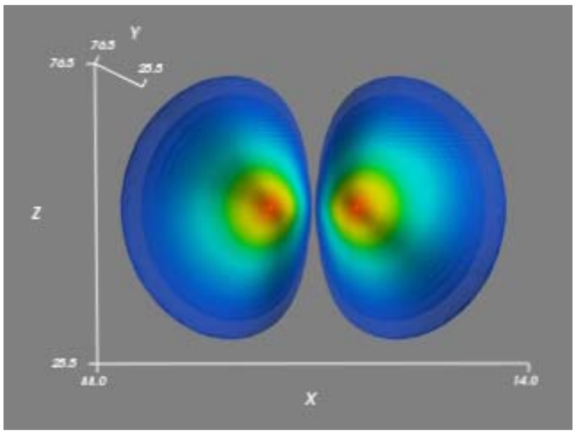


Figure 3: Plot of real $2p_x$ orbital

For $l = 2$ or the *d* orbitals, the wavefunction corresponding to $m = 0$ does not have a complex component while linear combinations need to be performed for $m = \pm1$ and $m = \pm2$ to give real orbital plots. Similarly, this is to be applied to $l = 3$ or *f* orbitals.

In Part A or the Chemistry section of this assignment, you are tasked to derive the equation $\Psi(r, \theta, \phi)$ for each combination of quantum numbers for your chosen principal quantum number *n*. You are also required to determine the linear combination of the complex wavefunctions to obtain the form required for real orbital plots. In addition, you will need to assign the correct naming convention to each of the orbital i.e. the quantum numbers $n = 1$, $l = 0$, $m = \pm1$ give the $2p_x$ and $2p_y$ orbitals for the respective sum and difference of the angular portion of the complex wavefunction.

Part B: Coding (Digital World) Segment

Kindly refer to Problem 7 of 10.009 The Digital World. This is also attached at the end of this document.

Part C (Optional): In-Depth Theory Segment (BONUS)

The Schrödinger equation is:

$$-\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi = E\Psi$$

For a system in spherical coordinates (*r*, *θ*, *ϕ*), the Laplacian operator in spherical coordinates is:

$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\left[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right]$$

The Laplaican operator is used to transform the above Schrödinger equation from Cartesian to spherical coordinates to give Equation 1:

$$\frac{\partial}{\partial r}\left(r^2\frac{\partial\Psi}{\partial r}\right) + \frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Psi}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2\Psi}{\partial\phi^2} = (E - V)\left(-\frac{2mr^2}{\hbar^2}\right)\Psi$$

(a) In Equation 1 above, there are terms that depend only on *r* and terms that depend only on angles θ and ϕ. To solve the equation, apply separation of variables and derive the expressions for both the angular component (an equation written in terms of *θ* and *ϕ* only) and radial component (an equation written in terms of *r* only). This can be done through the use of a separation constant, $l(l+1)$, so that the LHS of the equation adds up to the RHS of Equation 1. Do not substitute the chosen quantum numbers into the equation.

Hint for (a): We are looking at solutions that are separable into products in the form of:

$$\Psi(r,\theta,\phi) = R(r)\cdot Y(\theta,\phi)$$

You will need to use:

$$l(l+1) = \frac{1}{R}\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) + \frac{2mr^2}{\hbar^2}\left(E + \frac{e^2}{4\pi\varepsilon_0 r}\right) \text{ and } -V = \frac{e^2}{4\pi\varepsilon_0 r}$$

Now that we have separated Equation 1 to two equations in (a), we can solve for each component individually. With the chosen principal quantum number *n*, solve for the angular component first in (b) and then the radial component in (c) for all combinations of quantum numbers for the chosen principal quantum number *n*.

(b) Given the general expressions for the normalized angular solution for *Y*(*θ*, *ϕ*), associated Legendre function and the Legendre polynomial, find the Legendre polynomial, associated Legendre function and the normalized angular solution for all combinations of quantum numbers for your chosen principal quantum number *n*. Handwritten, step-by-step working is to be shown.

$$\text{Legendre Polynomial, } P_l(x) = \frac{1}{2^l l!}\left(\frac{\partial}{\partial x}\right)^l (x^2-1)^l$$

$$\text{Associated Legendre Function, } P_l^m(x) = (1-x^2)^{\frac{|m|}{2}}\left(\frac{\partial}{\partial x}\right)^{|m|} P_l(x)$$

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

$$\text{where } \epsilon = \begin{cases}(-1)^m \text{ for } m > 0\\ 1 \text{ for } m \le 0\end{cases}$$

Note: $P_l^m(\cos\theta)$ is a function of $\cos\theta$ while $P_l^m(x)$ is a function of x.

(c) Given the general expressions for the normalized radial solution for R(*r*), associated Laguerre function and the Laguerre polynomial, find the Laguerre polynomial, associated Laguerre function and the normalized radial solution, in terms of *r* and *a*, for all combinations of quantum numbers for your chosen principal quantum number *n*. Handwritten, step-by-step working is to be shown. Do not substitute the value of *a* into the solution.

$$\text{Laguerre Polynomial, } L_q(x) = e^x \left(\frac{\partial}{\partial x}\right)^q \left(e^{-x} x^q\right)$$

$$\text{Associated Laguerre Function, } L_{q-p}^{p}(x) = (-1)^p \left(\frac{\partial}{\partial x}\right)^p L_q(x)$$

$$\text{Normalized Radial Solution, } R_l^n(r) = \sqrt{\left(\frac{2}{na}\right)^3 \frac{(n-l-1)!}{2n[(n+l)!]^3}}\, e^{-\frac{r}{na}} \left(\frac{2r}{na}\right)^l \left[L_{q-p}^{p}\left(\frac{2r}{na}\right)\right]$$

where $p = 2l + 1$ and $q - p = n - l - 1$

Note: $L_{q-p}^{p}\left(\frac{2r}{na}\right)$ is a function of $\frac{2r}{na}$ while $L_{q-p}^{p}(x)$ is a function of x.

10.009 The Digital World

Term 3. 2018

Problem Set 7 (for Chemistry Project)

Last update: February 21, 2018

Due dates:

- **Problems**: Check Vocareum on individual submission dates.

Objectives:

1. Learn to use Numpy for numerical computations

Note: Solve the programming problems listed below using your favourite editor and test it. Make sure you save your programs in files with suitably chosen names and in an newly created directory. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems

1. **Week 2:** Create two functions to convert degrees to radian and radian to degrees respectively. These functions should take 1 float argument and return the respective conversions each. Round to 5 decimal places.

   To Test:

   ```
   print('deg_to_rad(90)')
   ans=deg_to_rad(90)
   print(ans)

   print('deg_to_rad(180)')
   ans=deg_to_rad(180)
   print(ans)

   print('deg_to_rad(270)')
   ans=deg_to_rad(270)
   print(ans)

   print('rad_to_deg(3.14)')
   ans=rad_to_deg(3.14)
   print(ans)

   print('rad_to_deg(3.14/2.0)')
   ans=rad_to_deg(3.14/2.0)
   print(ans)

   print('rad_to_deg(3.14*3/4)')
   ans=rad_to_deg(3.14*3/4)
   print(ans)
   ```

   The output should be:

   ```
   deg_to_rad(90)
   1.5708
   deg_to_rad(180)
   3.14159
   deg_to_rad(270)
   4.71239
   rad_to_deg(3.14)
   179.90875
   rad_to_deg(3.14/2.0)
   89.95437
   rad_to_deg(3.14*3/4)
   134.93156
   ```

2. **Week 2:** Create two functions to convert spherical to cartesian coordinates and cartesian to spherical coordinates. These functions should take 3 float arguments and return the 3 respective conversions. Round to 5 decimal places. The convention is shown below.

   Hint: you can use Numpy trigonometric function by doing `import numpy as np`.

   The input and output of these functions are as follows:

   - `spherical_to_cartesian(r, theta, phi)`: Returns `(x,y,z)`

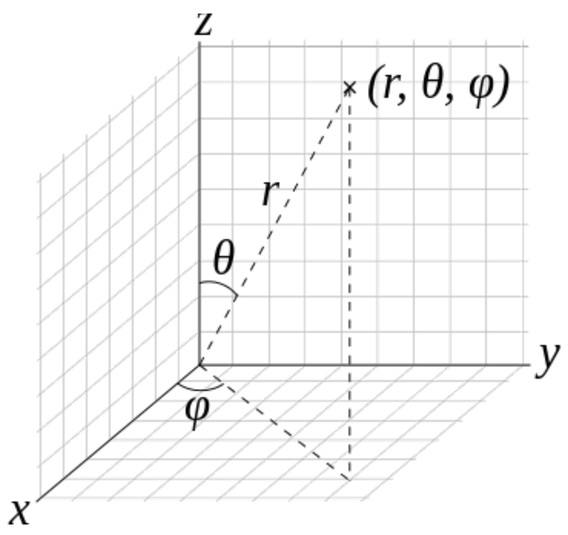


Figure 1: Spherical Coordinate System.

- `cartesian_to_spherical(x, y, z)`: Returns `(r, theta, phi)`

To test:

```
print('spherical_to_cartesian(3,0,np.pi)')
ans=spherical_to_cartesian(3,0,np.pi)
print(ans)

print('spherical_to_cartesian(3,np.pi/2.0,np.pi/2.0)')
ans=spherical_to_cartesian(3,np.pi/2.0,np.pi/2.0)
print(ans)

print('spherical_to_cartesian(3,np.pi, 0)')
ans=spherical_to_cartesian(3,np.pi,0)
print(ans)

print('cartesian_to_spherical(3,0,0)')
ans=cartesian_to_spherical(3,0,0)
print(ans)

print('cartesian_to_spherical(1,3,0)')
ans=cartesian_to_spherical(1,3,0)
print(ans)

print('cartesian_to_spherical(1,0,3)')
ans=cartesian_to_spherical(1,0,3)
print(ans)

print('cartesian_to_spherical(1,1,1)')
ans=cartesian_to_spherical(1,1,1)
print(ans)
```

The output should be:

```
spherical_to_cartesian(3,0,np.pi)
(-0.0, 0.0, 3.0)
spherical_to_cartesian(3,np.pi/2.0,np.pi/2.0)
(0.0, 3.0, 0.0)
```

```
spherical_to_cartesian(3,np.pi, 0)
(0.0, 0.0, -3.0)
cartesian_to_spherical(3,0,0)
(3.0, 1.5708, 0.0)
cartesian_to_spherical(1,3,0)
(3.16228, 1.5708, 1.24905)
cartesian_to_spherical(1,0,3)
(3.16228, 0.32174999999999998, 0.0)
cartesian_to_spherical(1,-3,0)
(3.16228, 1.5708, -1.24905)
cartesian_to_spherical(1,1,1)
(1.7320500000000001, 0.95531999999999995, 0.78539999999999999)
```

3. **Week 2:** Create a function to get the magnitude of a complex number. This function should take in a complex number and return a float as its magnitude. You not allowed to use `absolute` or similar built-in function.

   To test:

```
print('absolute(1+2j)')
ans=absolute(1+2j)
print(ans)

print('absolute(3+4j)')
ans=absolute(3+4j)
print(ans)

print('absolute(1+0j)')
ans=absolute(1+0j)
print(ans)

print('absolute(0+1j)')
ans=absolute(1+0j)
print(ans)
```

   The output should be:

```
absolute(1+2j)
2.2360679775
absolute(3+4j)
5.0
absolute(1+0j)
1.0
absolute(0+1j)
1.0
```

4. **Week 3:** Create a function that calculates the normalized angular solution. This function should take 4 float arguments and return the value of the normalized angular solution for the specific m, l, θ and ϕ. The return value is a complex number rounded to 5 decimal places for both the real and the imaginary parts. Hint: You may want to use `np.round()` function to round the return value to 5 decimal places.

   To test:

```
print('angular_wave_func(0,0,0,0)')
```

```
ans=angular_wave_func(0,0,0,0)
print(ans)

print('angular_wave_func(0,1,c.pi,0)')
ans=angular_wave_func(0,1,c.pi,0)
print(ans)

print('angular_wave_func(1,1,c.pi/2,c.pi)')
ans=angular_wave_func(1,1,c.pi/2,c.pi)
print(ans)

print('angular_wave_func(0,2,c.pi,0)')
ans=angular_wave_func(0,2,c.pi,0)
print(ans)
```

The output should be:

```
angular_wave_func(0,0,0,0)
(0.28209+0j)
angular_wave_func(0,1,c.pi,0)
(-0.4886+0j)
angular_wave_func(1,1,c.pi/2,c.pi)
(0.34549+0j)
angular_wave_func(0,2,c.pi,0)
(0.63078+0j)
```

5. **Week 3:** Create a function that calculates the normalized radial solution. This function should take 3 float arguments and return the value of the normalized radial solution. The return value should be normalized to $a^{-3/2}$, where a is the Bohr's radius, and rounded to 5 decimal places.

To test:

```
a=c.physical_constants['Bohr radius'][0]
print('radial_wave_func(1,0,a)')
ans=radial_wave_func(1,0,a)
print(ans)

print('radial_wave_func(2,1,a)')
ans=radial_wave_func(2,1,a)
print(ans)

print('radial_wave_func(2,1,2*a)')
ans=radial_wave_func(2,1,2*a)
print(ans)

print('radial_wave_func(3,1,2*a)')
ans=radial_wave_func(3,1,2*a)
print(ans)
```

The output should be:

```
radial_wave_func(1,0,a)
0.73576
radial_wave_func(2,1,a)
0.12381
radial_wave_func(2,1,2*a)
0.15019
```

```
radial_wave_func(3,1,2*a)
0.08281
```

6. **Week 4:** Create a function called `linspace` that takes in three arguments: start, stop, and number of points. The function should return a list of points from start up to and including stop. The number of points specifies the number of elements in the list and if not specified will be 50 points. Round each element to five decimal place. You can check `numpy.linspace` for reference. You are not allowed to use `numpy.linspace` or any other built-in function. However, you can use `numpy.linspace` to test your own function and compare the result.

   For example,

```
print('linspace(2.0, 3.0, num=3)')
ans=linspace(2.0, 3.0, num=3)
print(ans)
print('linspace(2.0, 3.0, num=5)')
ans=linspace(2.0, 3.0, num=5)
print(ans)
print('linspace(2.0, 3.0)')
ans=linspace(2.0, 3.0)
print(ans)
```

   The output should be:

```
linspace(2.0, 3.0, num=3)
[2.0, 2.5, 3.0]
linspace(2.0, 3.0, num=5)
[2.0, 2.25, 2.5, 2.75, 3.0]
linspace(2.0, 3.0)
[2.0, 2.02041, 2.04082, 2.06122, 2.08163, 2.10204, 2.12245, 2.14286,
     2.16327, 2.18367, 2.20408, 2.22449, 2.2449, 2.26531, 2.28571,
    2.30612, 2.32653, 2.34694, 2.36735, 2.38776, 2.40816, 2.42857,
    2.44898, 2.46939, 2.4898, 2.5102, 2.53061, 2.55102, 2.57143,
    2.59184, 2.61224, 2.63265, 2.65306, 2.67347, 2.69388, 2.71429,
    2.73469, 2.7551, 2.77551, 2.79592, 2.81633, 2.83673, 2.85714,
    2.87755, 2.89796, 2.91837, 2.93878, 2.95918, 2.97959, 3.0]
```

7. **Week 4:** Create a function called `meshgrid` that takes in three arguments `x, y, z`. The three input arguments are Python lists for the x, y, and z points in each one dimension. The function should return a list of lists as described in `numpy.meshgrid`. You are not allowed to use `numpy.meshgrid` or any other built-in function. However, you can use `numpy.meshgrid` to test your own function and compare the result. The following page explains the output of meshgrid in 2D: `https://plot.ly/numpy/meshgrid/`. Basically, if you have three arrays:

$$\begin{aligned} x &= [a1, a2, a3] \\ y &= [b1, b2, b3, b4] \\ z &= [c1, c2] \end{aligned}$$

The output of `meshgrid(x,y,z)` gives you:

```
[[[a1, a1], [a2, a2] , [a3, a3]], [[a1, a1], [a2, a2], [a3, a3]],
  [[a1, a1], [a2, a2], [a3, a3]], [[a1, a1], [a2, a2], [a3, a3]]],
[[[b1, b1], [b1, b1], [b1, b1]], [[b2, b2], [b2, b2], [b2, b2]],
  [[b3, b3], [b3, b3], [b3, b3]], [[b4, b4], [b4, b4], [b4, b4]]],
[[[c1, c2], [c1, c2], [c1, c2]], [[c1, c2], [c1, c2], [c1, c2]],
  [[c1, c2], [c1, c2], [c1, c2]], [[c1, c2], [c1, c2], [c1, c2]]]
```

Each array here contains arrays with an array full of the first item, the next filled with all the next item in the original array, etc. Note the following on the dimension:

- The dimension of the inner most list is set by the dimension of `z`.
- The dimension of the second list is set by the dimension of `x`.
- The dimension of the outer most list is set by the dimension of `y`.

For example,

```
x=[1,2,3]
y=[4,5,6,7]
z=[8,9]
print('test 1')
ans=meshgrid(x,y,z)
print(ans)

x=[0,0.5,1]
y=[2,2.5,3.0,3.5]
z=[4.0,4.5]
print('test 2')
ans=meshgrid(x,y,z)
print(ans)
```

The output should be:

```
test 1
([[[1.0, 1.0], [2.0, 2.0], [3.0, 3.0]], [[1.0, 1.0], [2.0, 2.0],
    [3.0, 3.0]], [[1.0, 1.0], [2.0, 2.0], [3.0, 3.0]], [[1.0, 1.0],
    [2.0, 2.0], [3.0, 3.0]]], [[[4.0, 4.0], [4.0, 4.0], [4.0, 4.0]],
    [[5.0, 5.0], [5.0, 5.0], [5.0, 5.0]], [[6.0, 6.0], [6.0, 6.0],
    [6.0, 6.0]], [[7.0, 7.0], [7.0, 7.0], [7.0, 7.0]]], [[[8.0, 9.0],
     [8.0, 9.0], [8.0, 9.0]], [[8.0, 9.0], [8.0, 9.0], [8.0, 9.0]],
    [[8.0, 9.0], [8.0, 9.0], [8.0, 9.0]], [[8.0, 9.0], [8.0, 9.0],
    [8.0, 9.0]]]])
test 2
([[[0.0, 0.0], [0.5, 0.5], [1.0, 1.0]], [[0.0, 0.0], [0.5, 0.5],
    [1.0,
1.0]], [[0.0, 0.0], [0.5, 0.5], [1.0, 1.0]], [[0.0, 0.0], [0.5,
    0.5],
[1.0, 1.0]]], [[[2.0, 2.0], [2.0, 2.0], [2.0, 2.0]], [[2.5, 2.5],
[2.5, 2.5], [2.5, 2.5]], [[3.0, 3.0], [3.0, 3.0], [3.0, 3.0]],
    [[3.5,
3.5], [3.5, 3.5], [3.5, 3.5]]], [[[4.0, 4.5], [4.0, 4.5], [4.0,
    4.5]],
[[4.0, 4.5], [4.0, 4.5], [4.0, 4.5]], [[4.0, 4.5], [4.0, 4.5], [4.0,
4.5]], [[4.0, 4.5], [4.0, 4.5], [4.0, 4.5]]])
```

8. **Week 5:** Create a function that calculates the square of the magnitude of the real wave function. The function takes in several arguments:

   - n: quantum number n
   - l: quantum number l
   - m: quantum number m
   - roa: maximum distance to plot from the centre, normalized to Bohr radius, i.e. r/a.
   - N_x: Number of points in the x axis.
   - N_y: Number of points in the y axis.
   - N_z: Number of points in the z axis.

   The function should returns:

   - xx: x location of all the points in a 3D Numpy array.
   - yy: y location of all the points in a 3D Numpy array.
   - zz: z location of all the points in a 3D Numpy array.
   - $density$: The square of the magnitude of the real wave function, i.e. $|\Psi|^2$

   Note that: the real wavefunction is be a linear combination of your complex wave functions. The real angular wavefunction can be computed from:

$$Y_{lm} = \begin{cases} \frac{i}{\sqrt{2}}(Y_l^m - (-1)^m Y_l^{-m}), & \text{if } m < 0 \\ Y_l^0, & \text{if } m == 0 \\ \frac{1}{\sqrt{2}}(Y_l^{-m} + (-1)^m Y_l^m), & \text{if } m > 0. \end{cases}$$

   You can refer to: `https://en.wikipedia.org/wiki/Spherical_harmonics#Real_form` for more detail.

   Hint: You may find the following functions to be useful:

   - `fvec=numpy.vectorize(f)`: This function takes in a function and return its vectorized version of the function.
   - `xx,yy,zz=numpy.mgrid[]`: This function takes in 1D arrays and returns its 3D arrays to conform to a 3D grid. **If you use meshgrid, you will need to swap the x and the y to conform to mgrid.**
   - `m=absolute(c)`: This function takes in a complex number and returns its absolute value or its magnitude. Use your own function rather than numpy's built-in function.
   - `ar=numpy.array(x)`: This function takes in a list and returns a numpy Array. Numpy array is faster to process than Python's list.

Hint: You will need to use all the previous functions you have done. Note that some of those functions may round the output to 5 decimal places and the final magnitude output from this function should also be rounded to 5 decimal places.

To test:

```
print('Test 1')
x,y,z,mag=hydrogen_wave_func(2,1,1,8,3,3,3)
print('x, y, z:')
print(x, y, z)
print('mag:')
print(mag)

print('Test 2')
x,y,z,mag=hydrogen_wave_func(2,1,1,5,3,4,2)
print('x, y, z:')
print(x, y, z)
print('mag:')
print(mag)

print('Test 3')
x,y,z,mag=hydrogen_wave_func(2,0,0,3,5,4,3)
print('x, y, z:')
print(x, y, z)
print('mag:')
print(mag)
```

The output should be:

```
Test 1
x, y, z:
[[[-8. -8. -8.]
  [-8. -8. -8.]
  [-8. -8. -8.]]

 [[ 0.  0.  0.]
  [ 0.  0.  0.]
  [ 0.  0.  0.]]

 [[ 8.  8.  8.]
  [ 8.  8.  8.]
  [ 8.  8.  8.]]] [[[-8. -8. -8.]
  [ 0.  0.  0.]
  [ 8.  8.  8.]]

 [[-8. -8. -8.]
  [ 0.  0.  0.]
  [ 8.  8.  8.]]

 [[-8. -8. -8.]
  [ 0.  0.  0.]
  [ 8.  8.  8.]]] [[[-8.  0.  8.]
  [-8.  0.  8.]
  [-8.  0.  8.]]

 [[-8.  0.  8.]
  [-8.  0.  8.]
  [-8.  0.  8.]]

 [[-8.  0.  8.]
```

```
  [-8.  0.  8.]
  [-8.  0.  8.]]]
mag:
[[[  0.00000000e+00   1.00000000e-05   0.00000000e+00]
  [  1.00000000e-05   2.10000000e-04   1.00000000e-05]
  [  0.00000000e+00   1.00000000e-05   0.00000000e+00]]

 [[  0.00000000e+00   0.00000000e+00   0.00000000e+00]
  [  0.00000000e+00              nan   0.00000000e+00]
  [  0.00000000e+00   0.00000000e+00   0.00000000e+00]]

 [[  0.00000000e+00   1.00000000e-05   0.00000000e+00]
  [  1.00000000e-05   2.10000000e-04   1.00000000e-05]
  [  0.00000000e+00   1.00000000e-05   0.00000000e+00]]]
Test 2
x, y, z:
[[[-5. -5.]
  [-5. -5.]
  [-5. -5.]
  [-5. -5.]]

 [[ 0.  0.]
  [ 0.  0.]
  [ 0.  0.]
  [ 0.  0.]]

 [[ 5.  5.]
  [ 5.  5.]
  [ 5.  5.]
  [ 5.  5.]]] [[[-5.       -5.     ]
  [-1.66667 -1.66667]
  [ 1.66667  1.66667]
  [ 5.       5.     ]]

 [[-5.       -5.     ]
  [-1.66667 -1.66667]
  [ 1.66667  1.66667]
  [ 5.       5.     ]]

 [[-5.       -5.     ]
  [-1.66667 -1.66667]
  [ 1.66667  1.66667]
  [ 5.       5.     ]]] [[[-5.  5.]
  [-5.  5.]
  [-5.  5.]
  [-5.  5.]]

 [[-5.  5.]
  [-5.  5.]
  [-5.  5.]
  [-5.  5.]]

 [[-5.  5.]
  [-5.  5.]
  [-5.  5.]
  [-5.  5.]]]
mag:
[[[  4.00000000e-05   4.00000000e-05]
  [  1.70000000e-04   1.70000000e-04]
  [  1.70000000e-04   1.70000000e-04]
  [  4.00000000e-05   4.00000000e-05]]
 [[  0.00000000e+00   0.00000000e+00]
  [  0.00000000e+00   0.00000000e+00]
  [  0.00000000e+00   0.00000000e+00]
  [  0.00000000e+00   0.00000000e+00]]

 [[  4.00000000e-05   4.00000000e-05]
  [  1.70000000e-04   1.70000000e-04]
  [  1.70000000e-04   1.70000000e-04]
  [  4.00000000e-05   4.00000000e-05]]]
Test 3
x, y, z:
[[[-3.  -3.  -3. ]
  [-3.  -3.  -3. ]
  [-3.  -3.  -3. ]
  [-3.  -3.  -3. ]]

 [[-1.5 -1.5 -1.5]
  [-1.5 -1.5 -1.5]
  [-1.5 -1.5 -1.5]
  [-1.5 -1.5 -1.5]]

 [[ 0.   0.   0. ]
  [ 0.   0.   0. ]
  [ 0.   0.   0. ]
  [ 0.   0.   0. ]]

 [[ 1.5  1.5  1.5]
  [ 1.5  1.5  1.5]
  [ 1.5  1.5  1.5]
  [ 1.5  1.5  1.5]]

 [[ 3.   3.   3. ]
  [ 3.   3.   3. ]
  [ 3.   3.   3. ]
  [ 3.   3.   3. ]]] [[[-3. -3. -3.]
  [-1. -1. -1.]
  [ 1.  1.  1.]
  [ 3.  3.  3.]]

 [[-3. -3. -3.]
  [-1. -1. -1.]
  [ 1.  1.  1.]
  [ 3.  3.  3.]]

 [[-3. -3. -3.]
  [-1. -1. -1.]
  [ 1.  1.  1.]
  [ 3.  3.  3.]]

 [[-3. -3. -3.]
  [-1. -1. -1.]
  [ 1.  1.  1.]
  [ 3.  3.  3.]]

 [[-3. -3. -3.]
  [-1. -1. -1.]
  [ 1.  1.  1.]
  [ 3.  3.  3.]]] [[[-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.   0.   3.]]

 [[-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]]

 [[-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]]

 [[-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]]

 [[-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]
  [-3.   0.   3.]]]
mag:
[[[  5.60000000e-04   7.20000000e-04   5.60000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  5.60000000e-04   7.20000000e-04   5.60000000e-04]]

 [[  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]]

 [[  7.20000000e-04   5.00000000e-04   7.20000000e-04]
  [  5.70000000e-04   3.66000000e-03   5.70000000e-04]
  [  5.70000000e-04   3.66000000e-03   5.70000000e-04]
  [  7.20000000e-04   5.00000000e-04   7.20000000e-04]]

 [[  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]]

 [[  5.60000000e-04   7.20000000e-04   5.60000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  5.60000000e-04   7.20000000e-04   5.60000000e-04]]]
```

9. **Week 8: Bonus Part: Plots of Real Orbitals:**

   Submit your plot for your assigned quantum numbers to your Chemistry instructors to get a point for this item.

10. **Week 8:** In the final function to calculate the hydrogen wave function, you are to use the other previous functions you have calculated. However, some of those functions rounds the result to 5 decimal places. The error on the final wave function magnitude is called ____ due to ____.

(a) floating point error, rounding error.

(b) propagation error, rounding error.

(c) propagation error, floating point error.

(d) rounding error, propagation error.

Submit your answer on eDimension.

11. **Week 8:** What is the effect when you increase the number of points Nx, Ny, Nz, while maintaining the values the other parameters?

    (a) increase of accuracy, decrease of computational time.

    (b) decrease of accuracy, increase of computational time.

    (c) increalse of accuracy, increase of computational time.

    (d) decrease of accuracy, decrease of computational time.

    Submit your answer on eDimension.

12. **Week 8:** What is the effect of increasing the distance r/a, while maintaining the values of the other paramters?

    (a) increase of accuracy, no change in computational time.

    (b) decrease of accuracy, change in computational time.

    (c) increalse of accuracy, change in computational time.

    (d) decrease of accuracy, no change in computational time.

    Submit your answer on eDimension.

Plotting sample codes:

- You can use the following code to save the Python data to a file:

```
import numpy as np

#######
# write all your function definitions here
#######

x,y,z,mag=hydrogen_wave_func(3,1,0,10,20,20,20)

x.dump('xdata.dat')
y.dump('ydata.dat')
z.dump('zdata.dat')
mag.dump('density.dat')
```

- You can use the following code to plot using matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.load('xdata.dat')
y = np.load('ydata.dat')
z = np.load('zdata.dat')

mag = np.load('density.dat')

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

for a in range(0,len(mag)):
    for b in range(0,len(mag)):
        for c in range(0,len(mag)):
            ax.scatter(x[a][b][c],y[a][b][c],z[a][b][c], marker='o',
                alpha=(mag[a][b][c]/np.amax(mag)))
plt.show()
```

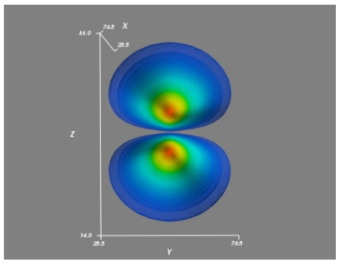
- You can use the following code to plot using mlab Mayavi package:

```
import numpy as np
from mayavi import mlab

x = np.load('xdata.dat')
y = np.load('ydata.dat')
z = np.load('zdata.dat')

density = np.load('density.dat')

figure = mlab.figure('DensityPlot')
# you should modify the parameters
pts = mlab.contour3d(density,contours=20,opacity=0.5)
mlab.axes()
mlab.show()
```

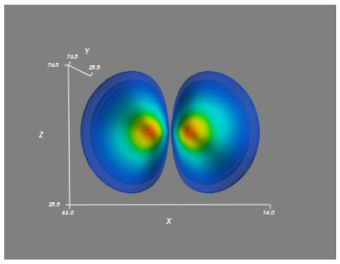
- You can check VolumeSlicer example from this website: http://docs.enthought.com/mayavi/mayavi/auto/example_volume_slicer.html.

Plots

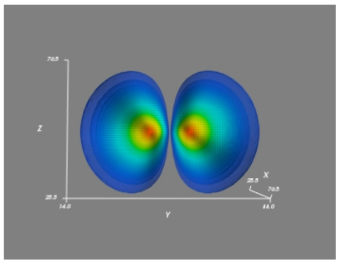
Sample plots



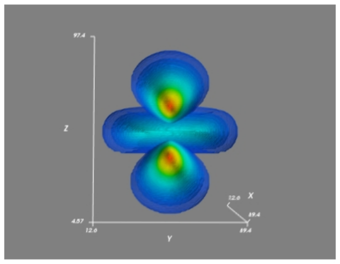
$2p_z$



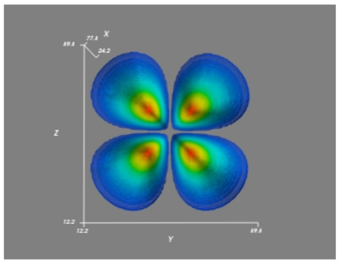
$2p_x$



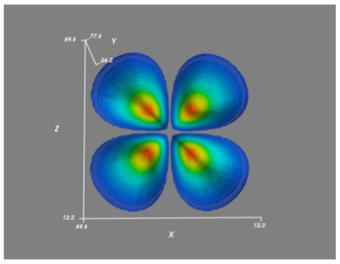
$2p_y$



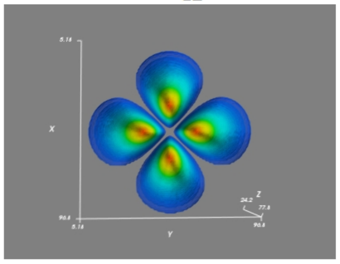
$3d_{z2}$



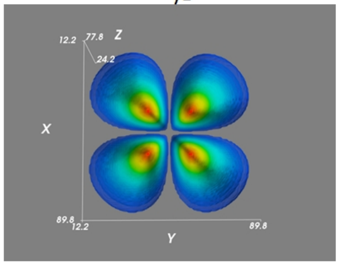
$3d_{yz}$



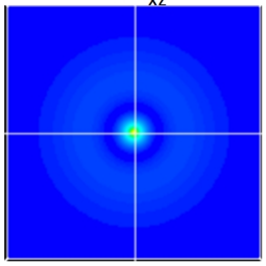
$3d_{xz}$



$3d_{x2-y2}$



$3d_{xy}$



2s orbital (volume slicing)