10.009 The Digital World

Term 3. 2018

Problem Set 7 (for Chemistry Project)

Last update: January 22, 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)`

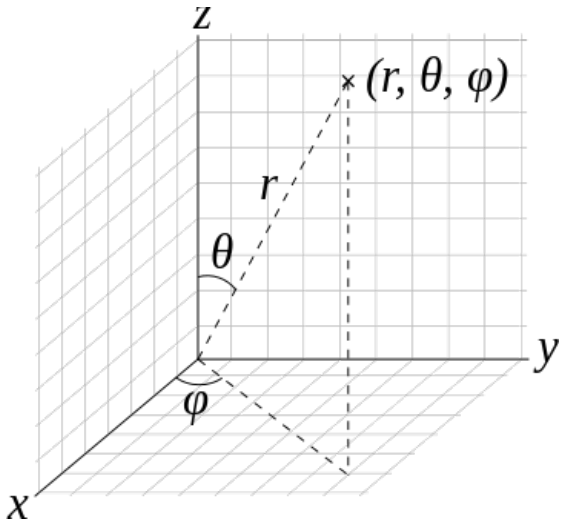


Figure 1: Spherical Coordinate System.

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

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

print('cartesian_to_spherical(0,-3,0)')
ans=cartesian_to_spherical(0,-3,0)
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(0,3,0)
(3.0, 1.5708, 1.5708)
cartesian_to_spherical(0,0,3)
(3.0, 0.0, 0.0)
cartesian_to_spherical(0,-3,0)
(3.0, 1.5708, -1.5708)
```

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

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(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)
```

   The output should be:

```
radial_wave_func(1,0,a)
0.73576
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:

$$x = [a1, a2, a3]$$

$$y = [b1, b2, b3, b4]$$

$$z = [c1, c2]$$

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], [b2, b2], [b3, b3]], [[b1, b1], [b2, b2], [b3, b3]],
  [[b1, b1], [b2, b2], [b3, b3]], [[b1, b1], [b2, b2], [b3, b3]]],
[[[c1, c1], [c2, c2], [c3, c3]], [[c1, c1], [c2, c2], [c3, c3]],
  [[c1, c1], [c2, c2], [c3, c3]], [[c1, c1], [c2, c2], [c3, c3]]]
```

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=meshgrid(x,y,z)`: This function takes in 1D arrays and returns its 3D arrays to conform to a 3D grid. Use your own meshgrid function rather than numpy's one.
   - `m=mag(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:

```
Test 1
x, y, z:
(array([[[-8.     , -8.     , -8.     , -8.     ],
        [-2.66667, -2.66667, -2.66667, -2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 8.     ,  8.     ,  8.     ,  8.     ]],

       [[-8.     , -8.     , -8.     , -8.     ],
        [-2.66667, -2.66667, -2.66667, -2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 8.     ,  8.     ,  8.     ,  8.     ]],

       [[-8.     , -8.     , -8.     , -8.     ],
        [-2.66667, -2.66667, -2.66667, -2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 8.     ,  8.     ,  8.     ,  8.     ]],

       [[-8.     , -8.     , -8.     , -8.     ],
        [-2.66667, -2.66667, -2.66667, -2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 8.     ,  8.     ,  8.     ,  8.     ]]]), array([[[-8.
      , -8.     , -8.     , -8.     ],
        [-8.     , -8.     , -8.     , -8.     ],
        [-8.     , -8.     , -8.     , -8.     ],
        [-8.     , -8.     , -8.     , -8.     ]],

       [[-2.66667, -2.66667, -2.66667, -2.66667],
        [-2.66667, -2.66667, -2.66667, -2.66667],
        [-2.66667, -2.66667, -2.66667, -2.66667],
        [-2.66667, -2.66667, -2.66667, -2.66667]],

       [[ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667],
        [ 2.66667,  2.66667,  2.66667,  2.66667]],

       [[ 8.     ,  8.     ,  8.     ,  8.     ],
        [ 8.     ,  8.     ,  8.     ,  8.     ],
        [ 8.     ,  8.     ,  8.     ,  8.     ],
        [ 8.     ,  8.     ,  8.     ,  8.     ]]]), array([[[-8.
      , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ]],

       [[-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ]],

       [[-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ],
        [-8.     , -2.66667,  2.66667,  8.     ]],
```

```
       [[-8.      , -2.66667,  2.66667,  8.      ],
        [-8.      , -2.66667,  2.66667,  8.      ],
        [-8.      , -2.66667,  2.66667,  8.      ],
        [-8.      , -2.66667,  2.66667,  8.      ]]]))
mag:
[[[  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
    +00]
  [  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
     +00]
  [  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
     +00]
  [  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
     +00]]

 [[  1.00000000e-05   9.00000000e-05   9.00000000e-05   1.00000000e
    -05]
  [  1.00000000e-05   7.00000000e-04   7.00000000e-04   1.00000000e
     -05]
  [  1.00000000e-05   7.00000000e-04   7.00000000e-04   1.00000000e
     -05]
  [  1.00000000e-05   9.00000000e-05   9.00000000e-05   1.00000000e
     -05]]

 [[  1.00000000e-05   9.00000000e-05   9.00000000e-05   1.00000000e
    -05]
  [  1.00000000e-05   7.00000000e-04   7.00000000e-04   1.00000000e
     -05]
  [  1.00000000e-05   7.00000000e-04   7.00000000e-04   1.00000000e
     -05]
  [  1.00000000e-05   9.00000000e-05   9.00000000e-05   1.00000000e
     -05]]

 [[  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
    +00]
  [  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
     +00]
  [  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
     +00]
  [  0.00000000e+00   1.00000000e-05   1.00000000e-05   0.00000000e
     +00]]]
Test 2
x, y, z:
(array([[[-5.      , -5.      , -5.      ],
         [-1.66667, -1.66667, -1.66667],
         [ 1.66667,  1.66667,  1.66667],
         [ 5.      ,  5.      ,  5.      ]],

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

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

        [[-5.      , -5.      , -5.      ],
         [-1.66667, -1.66667, -1.66667],
         [ 1.66667,  1.66667,  1.66667],
```

```
         [ 5.      ,  5.      ,  5.      ]],

        [[-5.      , -5.      , -5.      ],
         [-1.66667, -1.66667, -1.66667],
         [ 1.66667,  1.66667,  1.66667],
         [ 5.      ,  5.      ,  5.      ]]]), array([[[-5. , -5. , -5.
            ],
         [-5. , -5. , -5. ],
         [-5. , -5. , -5. ],
         [-5. , -5. , -5. ]],

        [[-2.5, -2.5, -2.5],
         [-2.5, -2.5, -2.5],
         [-2.5, -2.5, -2.5],
         [-2.5, -2.5, -2.5]],

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

        [[ 2.5,  2.5,  2.5],
         [ 2.5,  2.5,  2.5],
         [ 2.5,  2.5,  2.5],
         [ 2.5,  2.5,  2.5]],

        [[ 5. ,  5. ,  5. ],
         [ 5. ,  5. ,  5. ],
         [ 5. ,  5. ,  5. ],
         [ 5. ,  5. ,  5. ]]]), array([[[-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.]],

        [[-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.]],

        [[-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.]],

        [[-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.]],

        [[-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.],
         [-5.,  0.,  5.]]]))
mag:
[[[  4.00000000e-05   2.10000000e-04   4.00000000e-05]
  [  2.00000000e-05   1.40000000e-04   2.00000000e-05]
  [  2.00000000e-05   1.40000000e-04   2.00000000e-05]
  [  4.00000000e-05   2.10000000e-04   4.00000000e-05]]

 [[  1.40000000e-04   9.30000000e-04   1.40000000e-04]
  [  8.00000000e-05   1.37000000e-03   8.00000000e-05]
```

```
  [  8.00000000e-05   1.37000000e-03   8.00000000e-05]
  [  1.40000000e-04   9.30000000e-04   1.40000000e-04]]

 [[  2.10000000e-04   1.68000000e-03   2.10000000e-04]
  [  1.40000000e-04   5.22000000e-03   1.40000000e-04]
  [  1.40000000e-04   5.22000000e-03   1.40000000e-04]
  [  2.10000000e-04   1.68000000e-03   2.10000000e-04]]

 [[  1.40000000e-04   9.30000000e-04   1.40000000e-04]
  [  8.00000000e-05   1.37000000e-03   8.00000000e-05]
  [  8.00000000e-05   1.37000000e-03   8.00000000e-05]
  [  1.40000000e-04   9.30000000e-04   1.40000000e-04]]

 [[  4.00000000e-05   2.10000000e-04   4.00000000e-05]
  [  2.00000000e-05   1.40000000e-04   2.00000000e-05]
  [  2.00000000e-05   1.40000000e-04   2.00000000e-05]
  [  4.00000000e-05   2.10000000e-04   4.00000000e-05]]]
Test 3
x, y, z:
(array([[[-3. , -3. , -3. , -3. ],
        [-1.8, -1.8, -1.8, -1.8],
        [-0.6, -0.6, -0.6, -0.6],
        [ 0.6,  0.6,  0.6,  0.6],
        [ 1.8,  1.8,  1.8,  1.8],
        [ 3. ,  3. ,  3. ,  3. ]],

       [[-3. , -3. , -3. , -3. ],
        [-1.8, -1.8, -1.8, -1.8],
        [-0.6, -0.6, -0.6, -0.6],
        [ 0.6,  0.6,  0.6,  0.6],
        [ 1.8,  1.8,  1.8,  1.8],
        [ 3. ,  3. ,  3. ,  3. ]],

       [[-3. , -3. , -3. , -3. ],
        [-1.8, -1.8, -1.8, -1.8],
        [-0.6, -0.6, -0.6, -0.6],
        [ 0.6,  0.6,  0.6,  0.6],
        [ 1.8,  1.8,  1.8,  1.8],
        [ 3. ,  3. ,  3. ,  3. ]],

       [[-3. , -3. , -3. , -3. ],
        [-1.8, -1.8, -1.8, -1.8],
        [-0.6, -0.6, -0.6, -0.6],
        [ 0.6,  0.6,  0.6,  0.6],
        [ 1.8,  1.8,  1.8,  1.8],
        [ 3. ,  3. ,  3. ,  3. ]],

       [[-3. , -3. , -3. , -3. ],
        [-1.8, -1.8, -1.8, -1.8],
        [-0.6, -0.6, -0.6, -0.6],
        [ 0.6,  0.6,  0.6,  0.6],
        [ 1.8,  1.8,  1.8,  1.8],
        [ 3. ,  3. ,  3. ,  3. ]]]), array([[[-3. , -3. , -3. , -3.
           ],
        [-3. , -3. , -3. , -3. ],
        [-3. , -3. , -3. , -3. ],
        [-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],
        [-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. ],
        [ 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],
        [ 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. ,  3. ],
        [ 3. ,  3. ,  3. ,  3. ],
        [ 3. ,  3. ,  3. ,  3. ]]]), array([[[-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.]],

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

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

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

       [[-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.],
        [-3., -1.,  1.,  3.]]]))
mag:
[[[  5.60000000e-04   7.10000000e-04   7.10000000e-04   5.60000000e
```

```
  -04]
 [  6.70000000e-04   7.00000000e-04   7.00000000e-04   6.70000000e
    -04]
 [  7.20000000e-04   5.90000000e-04   5.90000000e-04   7.20000000e
    -04]
 [  7.20000000e-04   5.90000000e-04   5.90000000e-04   7.20000000e
    -04]
 [  6.70000000e-04   7.00000000e-04   7.00000000e-04   6.70000000e
    -04]
 [  5.60000000e-04   7.10000000e-04   7.10000000e-04   5.60000000e
    -04]]

[[  6.90000000e-04   6.80000000e-04   6.80000000e-04   6.90000000e
   -04]
 [  7.20000000e-04   2.30000000e-04   2.30000000e-04   7.20000000e
    -04]
 [  6.50000000e-04   1.00000000e-05   1.00000000e-05   6.50000000e
    -04]
 [  6.50000000e-04   1.00000000e-05   1.00000000e-05   6.50000000e
    -04]
 [  7.20000000e-04   2.30000000e-04   2.30000000e-04   7.20000000e
    -04]
 [  6.90000000e-04   6.80000000e-04   6.80000000e-04   6.90000000e
    -04]]

[[  7.20000000e-04   5.70000000e-04   5.70000000e-04   7.20000000e
   -04]
 [  6.80000000e-04   0.00000000e+00   0.00000000e+00   6.80000000e
    -04]
 [  5.20000000e-04   2.15000000e-03   2.15000000e-03   5.20000000e
    -04]
 [  5.20000000e-04   2.15000000e-03   2.15000000e-03   5.20000000e
    -04]
 [  6.80000000e-04   0.00000000e+00   0.00000000e+00   6.80000000e
    -04]
 [  7.20000000e-04   5.70000000e-04   5.70000000e-04   7.20000000e
    -04]]

[[  6.90000000e-04   6.80000000e-04   6.80000000e-04   6.90000000e
   -04]
 [  7.20000000e-04   2.30000000e-04   2.30000000e-04   7.20000000e
    -04]
 [  6.50000000e-04   1.00000000e-05   1.00000000e-05   6.50000000e
    -04]
 [  6.50000000e-04   1.00000000e-05   1.00000000e-05   6.50000000e
    -04]
 [  7.20000000e-04   2.30000000e-04   2.30000000e-04   7.20000000e
    -04]
 [  6.90000000e-04   6.80000000e-04   6.80000000e-04   6.90000000e
    -04]]

[[  5.60000000e-04   7.10000000e-04   7.10000000e-04   5.60000000e
   -04]
 [  6.70000000e-04   7.00000000e-04   7.00000000e-04   6.70000000e
    -04]
 [  7.20000000e-04   5.90000000e-04   5.90000000e-04   7.20000000e
    -04]
 [  7.20000000e-04   5.90000000e-04   5.90000000e-04   7.20000000e
    -04]
 [  6.70000000e-04   7.00000000e-04   7.00000000e-04   6.70000000e
    -04]
```

```
[  5.60000000e-04   7.10000000e-04   7.10000000e-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()
```

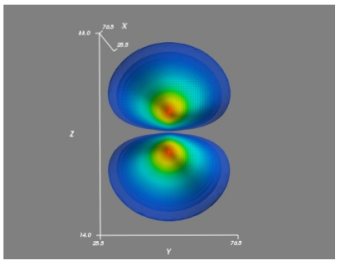
- You can use the following code to plot using mlab Mayavi package:

```
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()
```

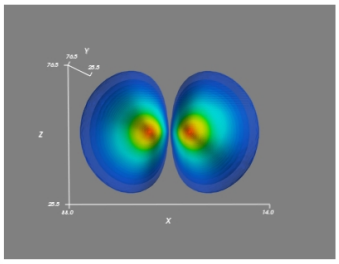
- You can check VolumeSlicer example from this website: http://docs.enthought.com/mayavi/mayavi/auto/example_volume_slicer.html.

Plots

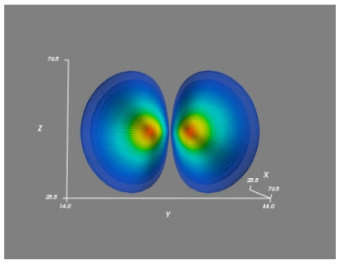
Sample plots



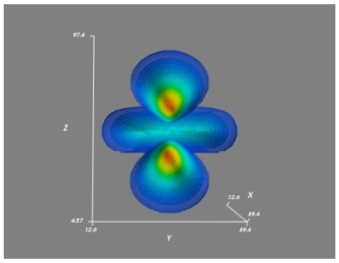
$2p_z$



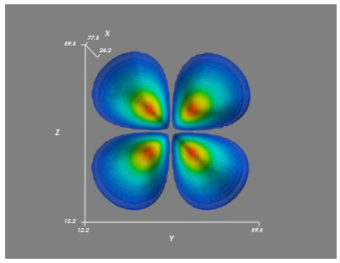
$2p_x$



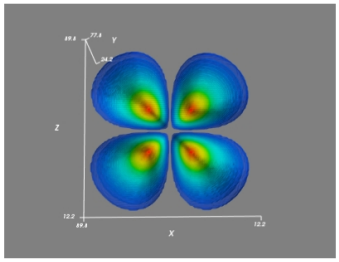
$2p_y$



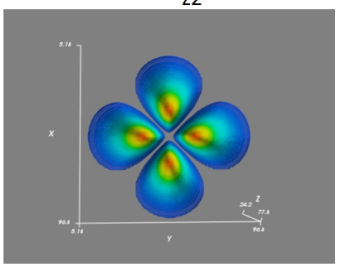
$3d_{z2}$



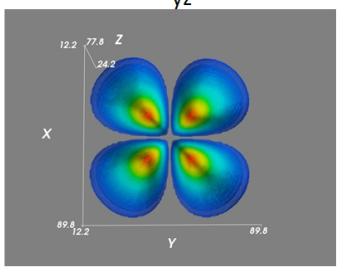
$3d_{yz}$



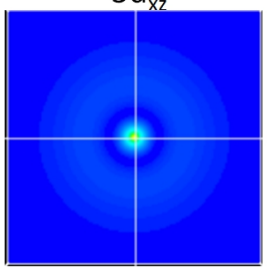
$3d_{xz}$



$3d_{x2-y2}$



$3d_{xy}$



2s orbital (volume slicing)