10.009 The Digital World

Term 3. 2017

Problem Set 7 (for Chemistry Project)

Last update: February 22, 2017

Due dates:

- **Problems**: Monday, Week 9, 5:00pm.

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 a function to calculate the energy level of a given principal quantum number. This function should take 1 integer argument and return the energy level in eV. Round to 5 decimal places. (Pre-requisite: Part 2a) Hint: You can use import `scipy.constants` as c to get the necessary constants.

   To test:

   ```
   print 'n = 1'
   ans= energy_n(1)
   print ans

   print 'n = 2'
   ans= energy_n(2)
   print ans

   print 'n = 3'
   ans= energy_n(3)
   print ans
   ```

   The output should be:

   ```
   n = 1
   -13.60569
   n = 2
   -3.40142
   n = 3
   -1.51174
   ```

2. **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
```

3. **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. (Pre-requisite: Part 2b) The convention is shown below.

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

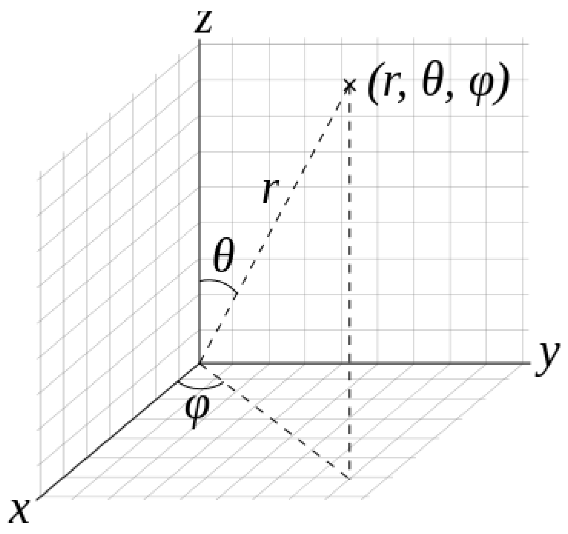

Figure 1: Spherical Coordinate System.

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)
```

4. **Week 3:** Create a function using the while loop that calculates the factorial of the input integer. For example, 4! = 4 x 3 x 2 x1 = 24. This function should take 1 integer argument and return the calculated factorial. The return value should be an integer.

   To test:

```
print 'fact(3)'
ans=fact(3)
print ans


print 'fact(5)'
ans=fact(5)
print ans


print 'fact(4)'
ans=fact(4)
print ans


print 'fact(1)'
```

```
ans=fact(1)
print ans
```

The output should be:

```
fact(3)
6
fact(5)
120
fact(4)
24
fact(1)
1
```

5. **Week 3:** Create a function that calculates the associated Legendre function. This function should take in 2 integer arguments and return the function that computes the associated Legendre with a variable $\cos(\theta)$. You should be able to obtain the function for any combination of m=0,1,2,3 and n=0,1,2,3 (Pre-requisite: Part 2e).

   Hint: You may want to create a function that acts as a reference table to a set of recalculated associated Legendre function. For example,

   ```
   def p00(theta):
     return 1

   def assoc_legendre(m,l):
     if m==0 and l==0:
       return p00
   ```

   To test:

   ```
   print 'f=assoc_legendre(0,0)'
   print 'f(1)'
   f=assoc_legendre(0,0)
   ans=f(1)
   print ans

   print 'f=assoc_legendre(1,1)'
   print 'f(1)'
   f=assoc_legendre(1,1)
   ans=f(1)
   print ans

   print 'f=assoc_legendre(2,3)'
   print 'f(1)'
   f=assoc_legendre(2,3)
   ans=f(1)
   print ans

   print 'f=assoc_legendre(2,3)'
   print 'f(0)'
   f=assoc_legendre(2,3)
   ans=f(0)
   print ans
   ```

   The output should be:

```
f=assoc_legendre(0,0)
f(1)
1
f=assoc_legendre(1,1)
f(1)
0.841470984808
f=assoc_legendre(2,3)
f(1)
5.73860550926
f=assoc_legendre(2,3)
f(0)
0.0
```

6. **Week 3:** Create a function that calculates the associated Laguerre function. This function should take in 2 integer arguments and return the function that computes the associated Laguerre with a variable x. You should be able to obtain the function for any combination of $p = 0, 1, 2, 3$ and $q - p = 0, 1, 2, 3$ (Pre-requisite: Part 2f)

   Hint: You may want to create a function that acts as a reference table to a set of recalculated associated Laguerre function. For example

```
def l00(x):
  return 1

def assoc_laguerre(p,qmp):
  if p==0 and qmp==0:
    return l00
```

   Note that `qmp` is the argument for $q - p$.

   To test:

```
print 'f=assoc_laguerre(0,0)'
print 'f(1)'
f=assoc_laguerre(0,0)
ans=f(1)
print ans

print 'f=assoc_laguerre(1,1)'
print 'f(1)'
f=assoc_laguerre(1,1)
ans=f(1)
print ans

print 'f=assoc_laguerre(2,2)'
print 'f(1)'
f=assoc_laguerre(2,2)
ans=f(1)
print ans

print 'f=assoc_laguerre(2,2)'
print 'f(0)'
f=assoc_laguerre(2,2)
ans=f(0)
print ans
```

   The output should be:

```
f=assoc_laguerre(0,0)
f(1)
1
f=assoc_laguerre(1,1)
f(1)
2
f=assoc_laguerre(2,2)
f(1)
60
f=assoc_laguerre(2,2)
f(0)
144
```

7. **Week 4:** 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. (Pre-requisite: Part 2e). 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. You can use the previous function `assoc_legendre(m,l)`, in the following way:

```
pfunc=assoc_legendre(m,l)
y=pfunc(theta)
```

where m and l are the respective quantum numbers and theta is the angle θ. This means that `assoc_legendre(m,l)` must return a function with one argument θ. See the test cases on the `assoc_legendre(m,l)` question.

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)
```

8. **Week 4:** 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. (Pre-requisite: Part 2f). Hint: You may want to use `np.round()` function to round the return value to 5 decimal places. You can use the previous function `assoc_laguerre(p, qmp)`, in the following way:

```
lfunc=assoc_laguerre(p, qmp)
y=lfunc(x)
```

   where the argument p and qmp refers to p and $q - p$ in the associated Laguerre. This means that `assoc_laguerre(p, qmp)` must return a function with one argument. See the test cases on the `assoc_laguerre(p, qmp)` question.

   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
```

9. **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 positive x axis.
- N_y: Number of points in the positive y axis.
- N_z: Number of points in the positive 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. $|\Psi|^2$

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.meshgrid(x,y,z)`: This function takes in 1D arrays and returns its 3D arrays to conform to a 3D grid.
- `m=numpy.absolute(c)`: This function takes in a complex number and returns its absolute value or its magnitude.

Hint: You also 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(2,1,1,8,2,2,2)
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. -8.]
  [-8. -8.]]

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

 [[-8.  8.]
  [-8.  8.]]]
mag:
[[[ 0.  0.]
  [ 0.  0.]]

 [[ 0.  0.]
  [ 0.  0.]]]
Test 2
x, y, z:
[[[-5. -5.]
  [ 0.  0.]
  [ 5.  5.]]

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

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

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

 [[-1.66667 -1.66667]
  [-1.66667 -1.66667]
  [-1.66667 -1.66667]]

 [[ 1.66667  1.66667]
  [ 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.]
  [-5.  5.]
  [-5.  5.]]]
mag:
[[[  4.00000000e-05   4.00000000e-05]
  [  1.10000000e-04   1.10000000e-04]
  [  4.00000000e-05   4.00000000e-05]]

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

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

 [[  4.00000000e-05   4.00000000e-05]
  [  1.10000000e-04   1.10000000e-04]
  [  4.00000000e-05   4.00000000e-05]]]
Test 3
x, y, z:
[[[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]

 [[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]

 [[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]

 [[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]] [[[-3. -3. -3.]
  [-3. -3. -3.]
  [-3. -3. -3.]
  [-3. -3. -3.]
  [-3. -3. -3.]]

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

 [[ 1.  1.  1.]
  [ 1.  1.  1.]
  [ 1.  1.  1.]
  [ 1.  1.  1.]
  [ 1.  1.  1.]]

 [[ 3.  3.  3.]
  [ 3.  3.  3.]
  [ 3.  3.  3.]
  [ 3.  3.  3.]
  [ 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]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  7.20000000e-04   5.00000000e-04   7.20000000e-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]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  5.70000000e-04   3.66000000e-03   5.70000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]]

 [[  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  5.70000000e-04   3.66000000e-03   5.70000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-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]
  [  7.20000000e-04   5.00000000e-04   7.20000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  5.60000000e-04   7.20000000e-04   5.60000000e-04]]]
```

10. **Week 8: Plots of Real Orbitals:** Write a program to plot the real orbitals of your assigned hydrogen function. The real orbitals will be a linear combination of your complex

wave functions:

$$\begin{aligned}\Psi_{px} &= \frac{1}{\sqrt{2}}(\Psi_1^{-1} - \Psi_1^1) \\ \Psi_{py} &= \frac{i}{\sqrt{2}}(\Psi_1^{-1} + \Psi_1^1)\end{aligned}$$

Note that this notation may be slightly different from `https://en.wikipedia.org/wiki/Atomic_orbital#Real_orbitals` and `https://en.wikipedia.org/wiki/Spherical_harmonics#Real_form`.

The function takes in several arguments:

- n: quantum number n
- l: quantum number l, i.e. 's', 'p', or 'd'.
- *orbital*: the choice of orbitals
- *roa*: maximum distance to plot from the centre, normalized to Bohr radius, i.e. r/a.
- N_x: Number of points in the positive x axis.
- N_y: Number of points in the positive y axis.
- N_z: Number of points in the positive 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. $|\Psi|^2$

You need not submit this to Tutor.

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

11. **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 Tutor.

12. **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 Tutor.

13. **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 Tutor.

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_real(3,'p', 'x',10,20,20,20)

x.dump('xdata3px.dat')
y.dump('ydata3px.dat')
z.dump('zdata3px.dat')
mag.dump('density3px.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('xdata3px.dat')
y = np.load('ydata3px.dat')
z = np.load('zdata3px.dat')

mag = np.load('density3px.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('xdata3px.dat')
y = np.load('ydata3px.dat')
z = np.load('zdata3px.dat')

density = np.load('density3px.dat')

figure = mlab.figure('DensityPlot')
mag=density/np.amax(density)

pts = mlab.points3d(mag,opacity=0.5, transparent=True)
# or pts = mlab.contour3d(mag, opacity=0.5)

mlab.colorbar(orientation='vertical')
mlab.axes()
mlab.show()
```

Plots

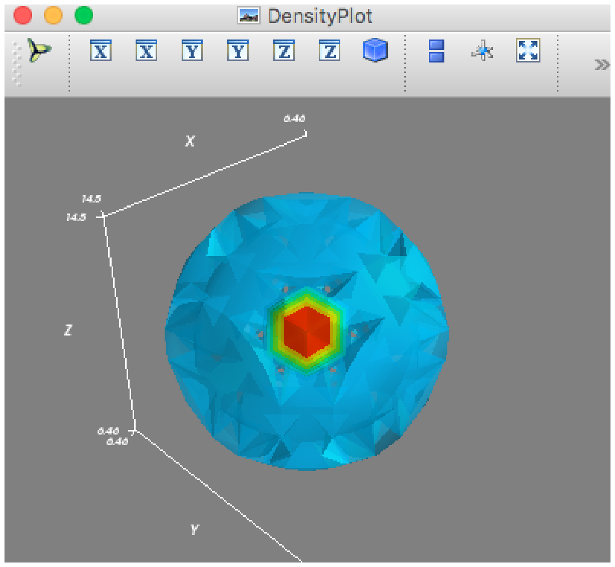

Figure 2: Magnitude plot for 2s using contour3d from mlab Mayavi package.

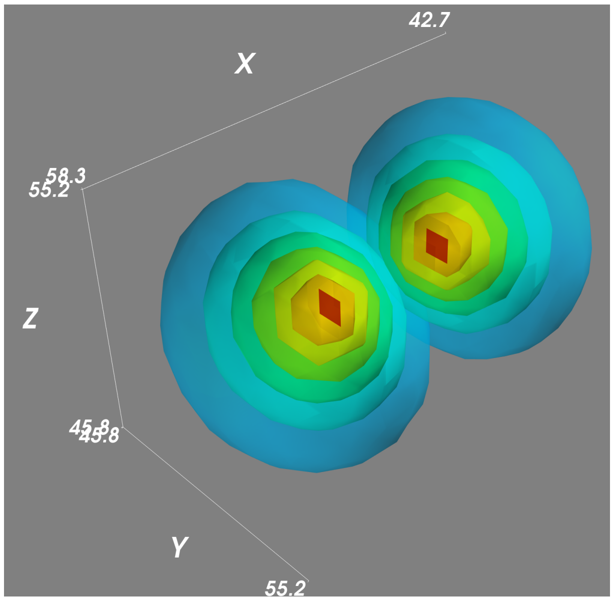

Figure 3: Magnitude plot for $2p_x$ using contour3d from mlab Mayavi package.

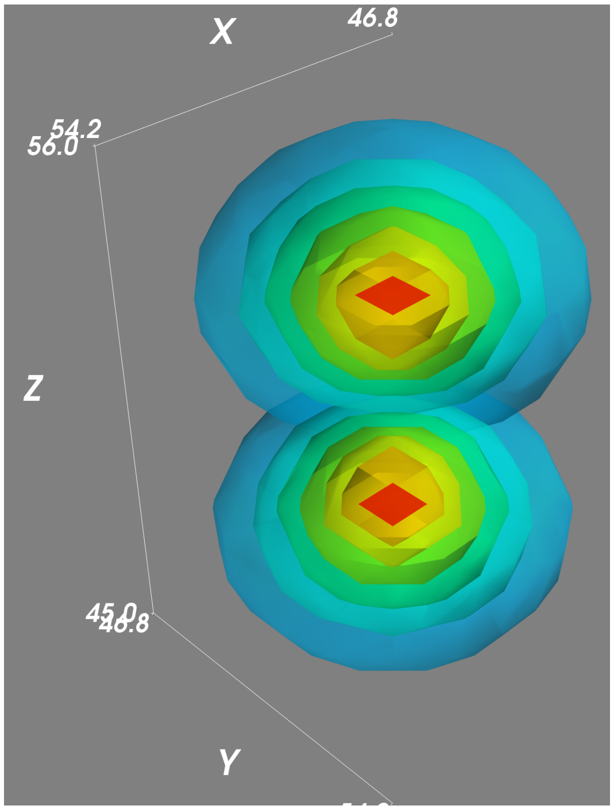

Figure 4: Magnitude plot for $3p_z$ using contour3d from mlab Mayavi package.

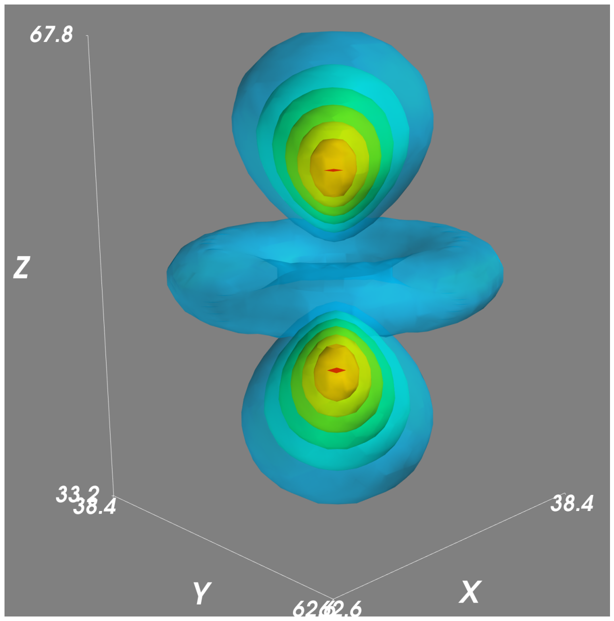

Figure 5: Magnitude plot for $3d_{z^2}$ using contour3d from mlab Mayavi package.

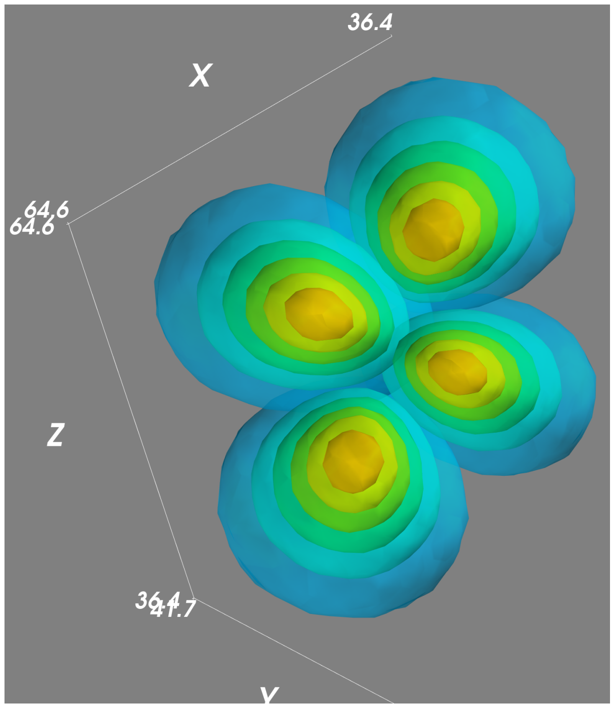

Figure 6: Magnitude plot for $3d_{xz}$ using contour3d from mlab Mayavi package.