水素原子のシュレディンガー方程式を最も単純なやり方で解く

田浦

1

水素原子中の電子の波動関数 $\phi(\boldsymbol{x})$ が満たす，時間に依存しない方程式は,

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + V(\boldsymbol{x})\right)\phi(\boldsymbol{x}) = E\phi(\boldsymbol{x}) \tag{1}$$

ここで,

$$V(\boldsymbol{x}) = -\frac{e^2}{4\pi\epsilon_0|\boldsymbol{x}|} \tag{2}$$

なので結局,

$$\left(-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right) - \frac{e^2}{4\pi\epsilon_0\sqrt{x^2+y^2+z^2}}\right)\phi(x,y,z) = E\phi(x,y,z) \tag{3}$$

が解くべき方程式.

2　離散化

解を求める空間を

$$[-a,a]^3 \equiv \{(x,y,z) \mid -a \le x,y,z \le a\} \tag{4}$$

とし, 一辺を $(n+1)$ 等分して離散化する $(x_{-1} = -a, x_{i+1} = x_i + 2a/(n+1), x_n = a)$.

$$\phi_{i,j,k} \equiv \phi(x_i, y_j, z_k) \tag{5}$$

と定義し, 式 (3) を離散化する. 例えば

$$\frac{\partial^2}{\partial x^2}\phi(x,y,z) \approx \frac{\phi(x+d,y,z) - 2\phi(x,y,z) + \phi(x-d,y,z)}{\Delta x^2} \tag{6}$$

したがって,

$$-\frac{\hbar^2}{2m\Delta x^2}\left(\phi_{i+1,j,k}+\phi_{i-1,j,k}+\phi_{i,j+1,k}+\phi_{i,j-1,k}+\phi_{i,j,k+1}+\phi_{i,j,k-1}-6\phi_{i,j,k}\right) - \frac{e^2}{4\pi\epsilon_0\sqrt{x_i^2+y_j^2+z_k^2}}\phi_{i,j,k} = E\phi_{i,j,k} \tag{7}$$

ただし $\Delta x = 2a/(n+1)$.

ただしい境界条件は無限遠で $\phi = 0$ というものだが，考えている領域の境界で 0 とする．

3　Python 化

$\phi_{i,j,k}$ を辞書順に並べたベクトルを ϕ とする．

$$\phi = {}^t(\phi_{0,0,0}\ \phi_{0,0,1}\ \cdots)$$

その約束のもと，左辺を

$$H\phi$$

と書いた時, H に相当する行列を作ることだけが仕事で，あとは固有値を求める scipy の関数に放り込めば良い.

まず

$$\phi_{i+1,j,k}+\phi_{i-1,j,k}+\phi_{i,j+1,k}+\phi_{i,j-1,k}+\phi_{i,j,k+1}+\phi_{i,j,k-1}-6\phi_{i,j,k}$$

を $L\phi$ と書いた時の L は，対角成分に -6, あとは非対角各成分に 1 を持つような帯が 6 つ並ぶだけの行列で，scipy では, diags を用いて簡単に書ける．

```
L = scipy.sparse.diags([-6.0,  1.0,  1.0,   1.0,  1.0,  1.0,  1.0 ],
                       [  0,    1,   -1,    n,   -n,  n*n, -n*n ],
                       shape=(n**3, n**3))
```

次に，

$$\frac{1}{\sqrt{x_i^2+y_j^2+z_k^2}}\phi_{i,j,k}$$

を $D\phi$ と書いたときの D に相当する行列は，単なる対角行列で，$\phi_{i,j,k}$ に対する添字が $1/\sqrt{x_i^2+y_j^2+z_k^2}$ であるというもの．$\phi_{i,j,k}$ は $n^2i+nj+k$ 番目の成分だからそれを用いてその行列を作ることもできるが，もっと簡単なのはまず 3 次元配列として作ってから 1 次元に reshape するというやり方．

```
l = np.linspace(-a, a, n+2)[1:-1]  # [-a,a]を (n+1)等分した内点n 個
x,y,z = np.meshgrid(l, l, l)       # x,y,z それぞれが n x n x n 配列
V = 1.0 / np.sqrt(x*x+y*y+z*z)     # V も n x n x n (universal 関数)
D = scipy.sparse.diags([ V.flatten() ], [ 0 ]) # flatten で 1 次元配列にする
```

上記のようにして L, D を得たら，

$$H=-\frac{\hbar^2}{2m\Delta x^2}L-\frac{e^2}{4.0\pi\epsilon_0}D$$

とすればよい．

ただし行列を目視で確認するときに絶対値の小さい値ばかりだと見難いので，L の係数が 1 になるようにスケールさせておく．

$$H'=-L-\frac{m\Delta x^2e^2}{2\pi\epsilon_0\hbar^2}D$$

付録のコードで計算しているのはこの H' である．

4 固有値ソルバ

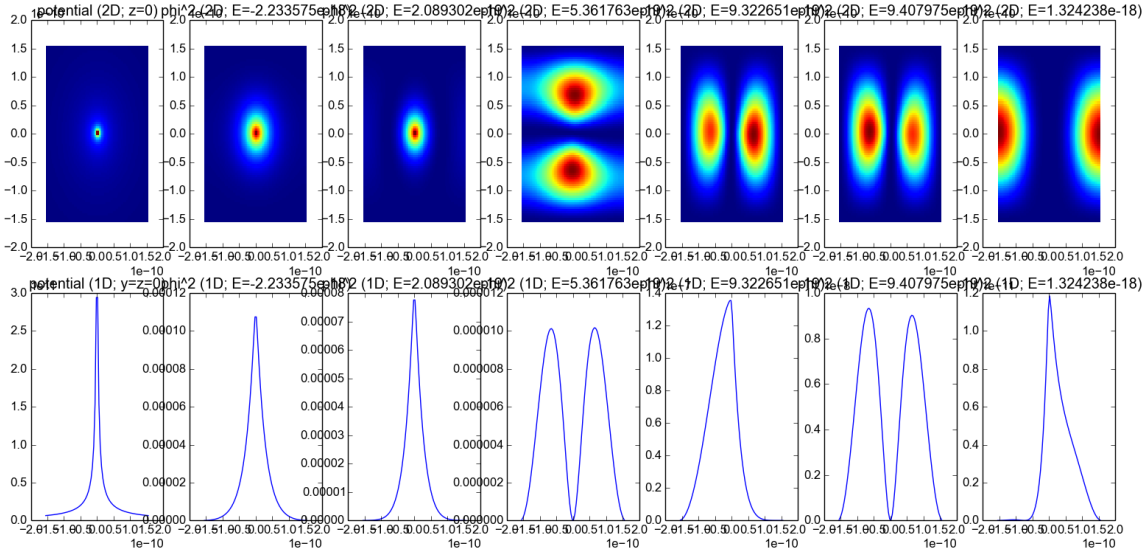
`scipy.sparse.linalg` に，`eigs` (一般の行列用), `eigsh` (対称 (半?) 正定値行列用) の 2 種類がある．対称は間違いないのだが，正定値かどうか確信のないまま結果オーライで後者を使っている．

多数の固有値固有ベクトルのうち，どれを返すか (`which`)，いくつ返すか (`k`) というオプションがあり，`which` が重要．algebraically smallest (つまり絶対値ではなく普通に符号を含めて小さいものを選ぶ)`which='SA'` を用いる．`k` は表示したいだけ選べばよく，とりあえず 6 としている．

```
scipy.sparse.linalg.eigsh(H_, k=6, which='SA', tol=tol)
```

5 結果

なんとなくそれっぽいものが得られている気が．左端がポテンシャル．あとは左からエネルギーが小さい順に並べたもの．



なお，最小の固有値は，$-2.23308690 \times 10^{-18}$ [J] と出ており，これは「水素原子 基底状態 エネルギー」とかでググって出てくる値と近いもので，いいのではないかという気がしています．

A　7/1 時点のコード

実質的な仕事をしているのは make_lhs_matrix という 10 文ほどの関数．

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import math,random
import numpy as np
import scipy.sparse
import scipy.sparse.linalg
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import time

#
# solve time-independent shrodinger equation:
#
# (-h'^2/2m △ + V(x)) f = E f
#
# where V(x) = -e * e/(4.0 * pi * epsilon |x|)
#
# h' = h/(2 pi) (h is a plank constant)
# m  = mass of electron
# e  = charge of ploton
# epsilon = permittivity of vacuum
#

eps =  8.854187817e-12
m   =  9.10938291e-31
e   = -1.602176565e-19
h   =  6.62606957e-34
h_  =  h / (2.0 * math.pi)
r   =  5.3e-11

#
# make the lefthand side matrix of
#
# ((-h_**2 /(2.0 * m)) △ - e**2 /(4.0 * pi * eps) * 1.0/|x|) f = E f
#
# in space [-a,a] x [-a,a] x [-a,a],
# with a mesh of n x n x n interim points
# (i.e., dx = (2a)/(n+1))
#

def make_lhs_matrix_3d(n, a):
    # 3D laplacian matrix
    L = scipy.sparse.diags([-6.0,  1.0,  1.0,   1.0,  1.0,  1.0,  1.0 ],
                           [   0,    1,   -1,    n,   -n,  n*n, -n*n ],
                          shape=(n**3, n**3))
    # D = matrix representing the potential
    l = np.linspace(-a, a, n+2)[1:-1]
    x,y,z = np.meshgrid(l, l, l)
    V = 1.0 / np.sqrt(x*x+y*y+z*z)
    D = scipy.sparse.diags([ V.flatten() ], [ 0 ])
    dx = (2.0 * a) / (n + 1)
    a = -h_ * h_ / (2.0 * m * dx * dx)
    b = -e * e / (4.0 * math.pi * eps)
    # here, the "true" matrix is a * L + b * D, but
    # to maintain human-readability of the matrix,
    # we divide it by (-a).
    # we return the triple (scale, potential, matrix)
    H_ = -L - (b/a) * D
    return -a,V,H_


def visualize_3d(a, V, E, phis):
    """
    visialize the result
     a : half of the side length of the cube in which
         we simulated. that is, we calculated solutions
         in the domain [-a,a]^3
     V : the n x n x n 3D array describing the potential
         in the above domain. n is the number of points
         along each axis.
     E : a 1D array describing eigenvalues we obtained
     phis : a (n**3) x nf 2D array describing eigenfunctions
         we obtained. n is the number of points
         along each axis and nf the number of eigenvalues
         we obtained.

    this function shows them in a tile of graphs like this.

```

```
     +-----+-----+-----+-----+-----+---------+
     |  1  |  2  |  3  | ..  |     |nf+1     |
     |     |     |     |     |     |         |
     | V   |phi0 |phi1 |     |     |phi[nf-1]|
     |(2D) |(2D) |(2D) |     |     |(2D)     |
     +-----+-----+-----+-----+-----+---------+
     |nf+2 |nf+3 |nf+4 | ..  |     |2(nf+1)  |
     |     |     |     |     |     |         |
     | V   |phi0 |phi1 |     |     |phi[nf-1]|
     |(1D) |(1D) |(1D) |     |     |(1D)     |
     +-----+-----+-----+-----+-----+---------+
    """

    #
    # n  : number of lattice points
    # nf : the number of eigenvalues/functions we found
    print "Eigenvalues = %s" % E
    n3,nf = phis.shape
    n = int(math.pow(n3+0.01, 1.0/3.0))
    assert(n ** 3 == n3), (n, n3)
    l = np.linspace(-a, a, n+2)[1:-1]
    x,y = np.meshgrid(l,l)
    # number of rows/colums of the tile above
    row = 2
    col = nf + 1
    # really start working
    fig = plt.figure()
    # show V, both in 1D and 2D
    ax2d = fig.add_subplot(row,col,1)
    ax2d.set_title("potential (2D; z=0)")
    ax2d.pcolor(x, y, V[:,:,n/2])
    ax1d = fig.add_subplot(row,col,nf+2)
    ax1d.set_title("potential (1D; y=z=0)")
    ax1d.plot(l, V[:,n/2,n/2])
    # show nf eigenvalues/functions
    for k in range(nf):
        # extract k-th column of phis
        phi = phis[:,k].reshape(n,n,n)
        # show in 2D
        ax2 = fig.add_subplot(row,col,k+2)
        ax2.set_title("phi^2 (2D; E=%e)" % E[k])
        f = phi[:,:,n/2] * phi[:,:,n/2].conjugate()
        ax2.pcolor(x, y, f)
        # show in 1D
        ax1 = fig.add_subplot(row,col,k+nf+3)
        ax1.set_title("phi^2 (1D; E=%e)" % E[k])
        f = phi[:,n/2,n/2] * phi[:,n/2,n/2].conjugate()
        ax1.plot(l, f)
    plt.show()

def hydrogen(n):
    """
    solve shrodinger equation of a hydrogen
    atom (or a hydrogen-like ion that has
    only one electron) in a cube, using
    n x n x n lattice points.

    """
    # simulate in [-a,a]x[-a,a]x[-a,a]
    a = 3 * r
    # we get a trouble if n is odd, as an origin
    # will be included in the lattice, at which
    # the potential is infinite
    if n % 2 != 0:
        sys.stderr.write("avoid using an odd n to avoid "
                         "zero division near the origin, "
                         "fixed it to %d\n" % (n + 1))
        n = n + 1
    # get H of Hx = Ex, along with potential V
    # for visualization purpose.
    # the returned H_ is actually a matrix such
    # that scale * H_ is the real matrix.
    # so we need to scale returned eivenvalues
    scale,V,H_ = make_lhs_matrix_3d(n, a)
    # scale MUST BE > 0.0 so that finding
    # algebraically smallest eigenvalues returns
    # eigenvalues we want; otherwise we colud fix
    # that by finding largest ones, but do not bother
    assert(scale > 0.0)
    # error tolerance
    tol = 1e-6
    t0 = time.time()
    # find algebraically smallest 6 eigenvalues of H
    E,phis = scipy.sparse.linalg.eigsh(H_, k=6, which='SA', tol=tol)
    t1 = time.time()
    print "%f sec" % (t1 - t0)
    visualize_3d(a, V, scale * E, phis)
```