Example of manipulating phimap in the cube-format
Technical details:

This example shows how to use python-script to extract data from phimap/espmap in cube format. Suppose that Delphi generates a phimap file “phimap.txt” and you want to get a potential at some points with coordinates (x1,y1,z1), (x2,y2,z2) and etc.

Let’s consider two python’s scripts “cube.py” and “interpolation.py”. The scripts can be found in
the Appendix. The first script, “cube.py”, reads file phimap and recognizes the following parameters: N –number of points along one dimension, initial point (X0,Y0,Z0), endpoint (X1,Y1,Z1) and scale. Then the script calls “f2=interpolation.Fi(x,y,z,Phi,coord)”, where “coord” is array with the coordinates (x1,y1,z1), (x2,y2,z2) where the potential will be outputted and array f2 contains value of potential in these points. Since these points may not match any grid point, the potential must be interpolated. It is done by the script “interpolation.py” which uses trilinear interpolation (http://en.wikipedia.org/wiki/Trilinear_interpolation).

Formulation so the problem: Consider a spherical molecule with total
charge *q* homogeneously distributed on the molecule surface, immersed in solvent containing mobile univalent ions, as depicted in Figure 1. We have the next analytical solution for potential in kBT/e_0 units (see M. J. Holst - The Poisson-Boltzmann Equation. Analysis and Multilevel Numerical Solution. 1994).

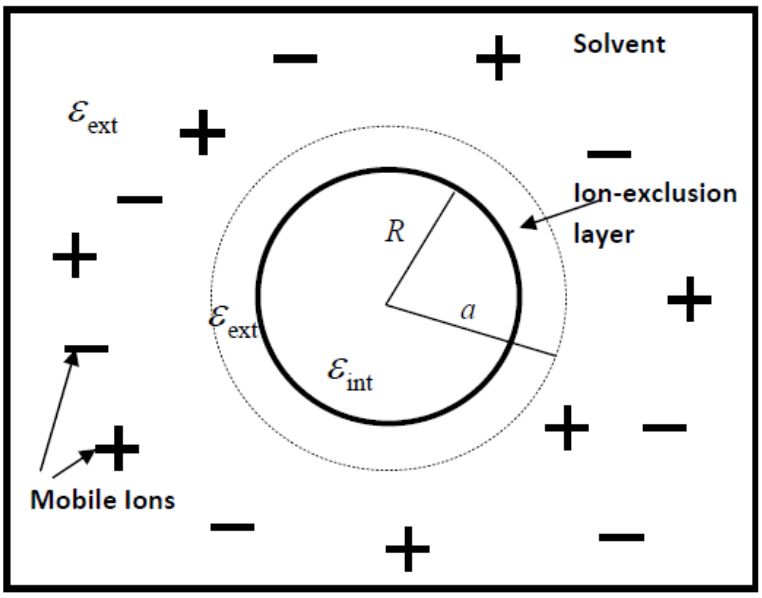

Figure 1. A spherical molecule with spherically symmetric

$$\Phi(r) = \frac{q \cdot e^{k\cdot(a-r)}}{4 \cdot \pi \cdot \varepsilon_0 \cdot \varepsilon_{ext} \cdot \frac{e_0}{k_B T} \cdot (1 + k \cdot a) \cdot r}, \quad r > a \tag{1}$$

where *k* is Debye-Huckel parameter, can be found in the output file from Delphi as inverse Debye length.

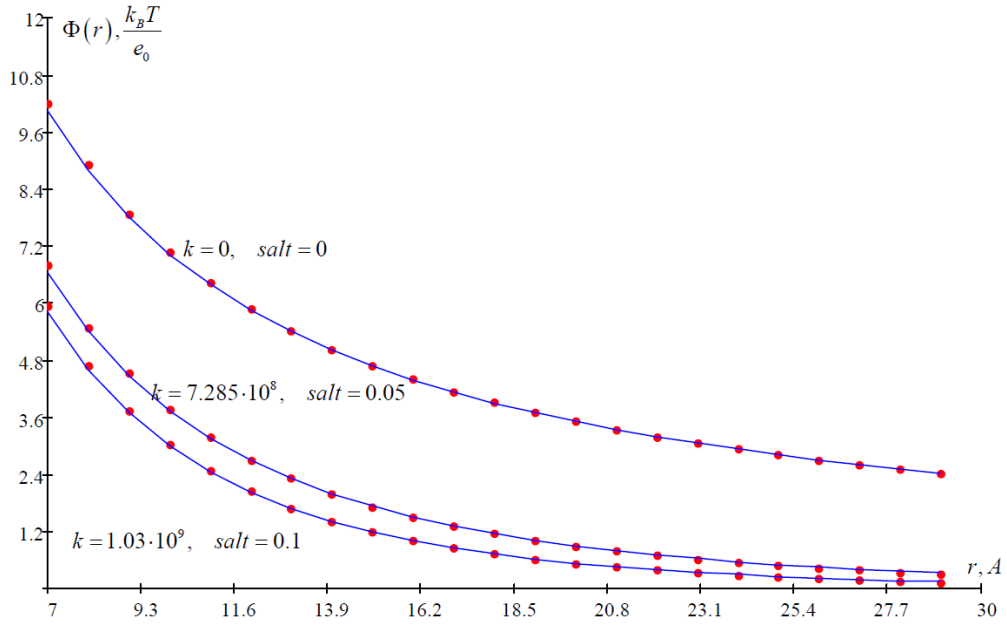

Figure 2. Dependence of potential from distance for different *k* . Solid line is eq. (1), while the points are the potential calculated with DelPhi. εext =80 , εint = 4 , R= 5Å, a=7Å.

Figure 2 shows a very good agreement between calculations with DelPhi and analytical result (1).

Appendix

cube.py:

```
import numpy as np
import interpolation
fphi='phimap.txt'
f=open(fphi,'r')
CUBE=f.readlines()
f.close()
RBohr=0.529177
N=int(CUBE[3].split()[0])
X0=float(CUBE[2].split()[1])*RBohr
Y0=float(CUBE[2].split()[2])*RBohr
Z0=float(CUBE[2].split()[3])*RBohr
scale=float(CUBE[3].split()[1])*RBohr
X1=X0+(N-1)*scale
Y1=Y0+(N-1)*scale
Z1=Z0+(N-1)*scale
Phi=np.zeros((N,N,N))
Phi1=np.zeros(N*N*N)
t=0
for ind in CUBE[7:]:
q=ind.split()
number=[float(P) for P in q]
Phi1[t:t+len(number)]=number
t=t+len(number)
Phi=Phi1.reshape(N,N,N)
ind=np.arange(N)
x=X0+ind*scale
y=Y0+ind*scale
z=Z0+ind*scale
n=len(np.arange(5.0,20.0,0.5))
coord=np.zeros((3,n))
coord[0,:]=np.arange(5.0,20.0,0.5)
coord[1,:]=[0]*n
coord[2,:]=[0]*n
f2=interpolation.Fi(x,y,z,Phi,coord)
```

interpolation.py

```
def Fi(x,y,z,Phi,Points):
import numpy as np
N=len(Points[0,:])
Fi=np.zeros((N,1))
ind_x = (Points[0,:]-x[0])//(x[1]-x[0])
ind_y = (Points[1,:]-y[0])//(y[1]-y[0])
ind_z = (Points[2,:]-z[0])//(z[1]-z[0])
for i in range(N):
C000=Phi[ind_z[i],ind_y[i],ind_x[i]]
```

```
C100=Phi[ind_z[i],ind_y[i],ind_x[i]+1]
C010=Phi[ind_z[i],ind_y[i]+1,ind_x[i]]
C110=Phi[ind_z[i],ind_y[i]+1,ind_x[i]+1]
C001=Phi[ind_z[i]+1,ind_y[i],ind_x[i]]
C101=Phi[ind_z[i]+1,ind_y[i],ind_x[i]+1]
C011=Phi[ind_z[i]+1,ind_y[i]+1,ind_x[i]]
C111=Phi[ind_z[i]+1,ind_y[i]+1,ind_x[i]+1]
a=Points[0,i]-x[ind_x[i]]
b=x[ind_x[i]+1]-Points[0,i]
C00=(C000*b+C100*a)/(a+b)
C10=(C010*b+C110*a)/(a+b)
C01=(C001*b+C101*a)/(a+b)
C11=(C011*b+C111*a)/(a+b)
a=Points[1,i]-y[ind_y[i]]
b=y[ind_y[i]+1]-Points[1,i]
C0=(C00*b+C10*a)/(a+b)
C1=(C01*b+C11*a)/(a+b)
a=Points[2,i]-z[ind_z[i]]
b=z[ind_z[i]+1]-Points[2,i]
Fi[i]=(C0*b+C1*a)/(a+b)
return Fi
```