计算物理第四次作业

梁旭民*

Cuiying Hornors College, Lanzhou University
liangxm15@lzu.edu.cn

Abstract

本次计算物理作业有两个问题：第一个问题是学习*Sherman-Morrison*公式及*Sherman-Morrison-Woodbury*公式，并利用公式将对于循环三对角线性方程组的求解问题转化为对于三对角线性方程组的求解，从而使得问题得到简化。第二个问题则是利用有限差分的方法求解一个具体的物理问题：第一类边界条件下的*Poisson-Boltzman*方程从而求解出等离子体孤波的电势分布情况。

I. Sherman-Morrison公式

A. 问题描述

利用Sherman-Morrison公式作为Gauss消元法的修正，求解循环三对角矩阵的逆，从而求解循环三对角线性方程组$Ax = r$并且在以上基础上了解woodbury公式

B. 问题分析

1. Sherman-Morrison公式表述

设$A \in \mathbb{R}^{n\times n}$是一个可逆方阵并且$u, v \in \mathbb{R}^n$是列向量。当且仅当$I + v^T A^{-1} u \neq 0$,则有$A + uv^T$是可逆的，且它的逆可以写成

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{I + v^T A^{-1} u}$$

其中，uv^T是两个向量u, v的外积。

2. Sherman-Morrison公式证明

证明必要性。当且仅当$XY = YX = I$，矩阵Y(Sherman-Morrison公式的右边)是矩阵X$(A + uv^T)$的逆矩阵。

我们先验证矩阵Y满足$XY = I$，则有

$$\begin{aligned}
XY &= (A + uv^T)\left(A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u}\right) \\
&= AA^{-1} + uv^T A^{-1} \\
&\quad - \frac{AA^{-1}uv^T A^{-1} + uv^T A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u} \\
&= I + uv^T A^{-1} \\
&\quad - \frac{uv^T A^{-1} + uv^T A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u} \\
&= I + uv^T A^{-1} \\
&\quad - \frac{u(1 + v^T A^{-1} u)v^T A^{-1}}{1 + v^T A^{-1} u} \\
&= I + uv^T A^{-1} - uv^T A^{-1} \\
&= I
\end{aligned}$$

类似的，可以验证

$$YX = \left(A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u}\right)(A + uv^T) = I.$$

证明充分性。假设$u \neq 0$，则有

$$(A+uv^T)A^{-1}u = u+uv^T A^{-1}u = (I+v^T A^{-1}u)u$$

因此，假设$A + uv^T$是可逆矩阵，则$(A + uv^T)A^{-1}$作为可逆矩阵的乘积，也是可逆矩阵。因此，通过假设$u \neq 0$，我们可以得到$(A + uv^T)A^{-1}u \neq 0$。

3. Woodbury公式表述

设$A \in \mathbb{R}^{n\times n}$是一个可逆方阵并且$u, v \in \mathbb{R}^n$是列向量。当且仅当$I + v^T A^{-1} u \neq 0$,则

*指导老师：齐新老师

有$A+uv^T$是可逆的。如果$A+uv^T$是可逆的，且它的逆可以写成

$$(A+uv^T)^{-1} = A^{-1} - A^{-1}u(I+v^TA^{-1}u)v^TA^{-1}$$

4. Sherman-Morrison-Woodbury公式证明

设$(A+uv^T)x = r$，则有

$$A^{-1}(A+uv^T)x = x + A^{-1}uv^Tx = A^{-1}r$$

若假设

$$\begin{cases} A^{-1}x = z \\ A^{-1}r = y \end{cases}$$

则可以得到

$$x + zv^Tx = y$$

左右两边同时左乘v^T，并令$\alpha = v^Tx$，则有

$$\alpha + v^Tz\alpha = v^Ty$$

可以解得

$$\alpha = \frac{v^Ty}{I+v^Tz}$$

代入$x+\alpha z = y$，则有

$$\begin{aligned} x &= y - \alpha z \\ &= A^{-1}r - A^{-1}u(I+v^TA^{-1}u)^{-1}v^Ty \\ &= A^{-1}r - A^{-1}u(I+v^TA^{-1}u)^{-1}v^TA^{-1}r \\ &= [A^{-1} - A^{-1}u(I+v^TA^{-1}u)^{-1}v^TA^{-1}]r \end{aligned}$$

因此证明

$$(A+uv^T) = A^{-1} - A^{-1}u(I+v^TA^{-1}u)^{-1}v^TA^{-1}$$

5. 利用Sherman-Morrison公式求解循环三对角线性方程组

求解的循环三对角线性方程组为

$$Ax = r$$

其中$A \in \mathbb{R}^{n\times n}$为可逆矩阵，其具体形式如下

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 & a_1 \\ a_2 & b_2 & c_2 & 0 & \vdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \vdots & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\ 0 & \cdots & \cdots & a_{n-1} & b_{n-1} & c_{n-1} \\ c_n & 0 & \cdots & 0 & a_n & b_n \end{bmatrix}$$

而矢量$r = (r_1, r_2, \cdots, r_n)^T$为已知，其中$x = (x_1, x_2, \cdots, x_n)^T$为待求未知数。

若构造三对角矩阵B为

$$\begin{bmatrix} b_1-\gamma & c_1 & 0 & \cdots & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \vdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \vdots & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\ 0 & \cdots & \cdots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & \cdots & 0 & a_n & b_n - \frac{a_1c_n}{\gamma} \end{bmatrix}$$

使其满足$A = B + uv^T$,且有$1 + v^TB^{-1}u \neq 0$。其中$u, v \in \mathbb{R}^n$为列向量，可以写成

$$\begin{cases} u = (\gamma, 0, 0, \cdots, c_n)^T \\ v = (1, 0, 0, \cdots, \frac{a_1}{\gamma})^T \end{cases}$$

则原线性方程组变成

$$(B+uv^T)x = r$$

可以将该线性方程组分解成以下两个三对角线性方程组，通过求解以下两个三对角线性方程组

$$\begin{cases} By = r \\ Bz = u \end{cases}$$

利用Sherman-Morrison公式，便可以得到原线性方程组的解为

$$x = y - \frac{v^Ty}{1+v^Tz}z$$

6. 验证原方程解

将$x = y - \frac{v^Ty}{1+v^Tz}z$代入原线性方程组，则有

$$\begin{aligned} &(B+uv^T)x \\ &= (B+uv^T)(y - \frac{v^Ty}{1+v^Tz}z) \\ &= By + uv^Ty - \frac{v^Ty}{1+v^Tz}Bz - \frac{v^Ty}{1+v^Tz}uv^Tz \\ &= r + uv^Ty - \frac{v^Tyu + v^Tyuv^Tz}{1+v^Tz} \\ &= r + uv^Ty - \frac{(1+v^Tz)v^Tyu}{1+v^Tz} \\ &= r + uv^Ty - uv^Ty \\ &= r \end{aligned}$$

II. 一维Poisson-Bolzman方程

A. 问题描述

求解第一类边界条件下（边界$\phi = 0$）的一维Poisson-Boltzman方程，计算等离子体孤波的电势。

$$\frac{\partial^2\phi}{\partial x^2} = \mu \mathrm{e}^{\phi} - Zn_i$$

其中，$\mu = 1$，$Z = 1$，离子密度$n_i(x)$为

$$n_i(x) = 1 + 3\varepsilon^2 u_0 \mathrm{sech}^2 \frac{\varepsilon\sqrt{\mu_0}[x - x_0]}{2}$$

其形状大致见FIG. 1.(纵坐标为电势ϕ，但在$\varepsilon^2 \ll 1$时，$\phi = n_i - 1$)

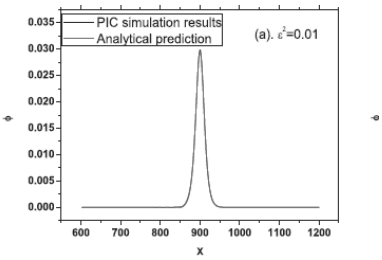

Figure 1: 离子浓度分布

其中，$u_0 = 1$。

B. 问题分析

1. 简介

Poisson- Boltzmann方程是用来计算电解质溶液中离子浓度和电荷密度分布的一个微分方程。其基本形式为(单位为高斯单位制)

$$\nabla^2\phi(\mathbf{r}) = -\frac{4\pi}{\epsilon}\sum_i c_i^0 z_i q e^{-\beta z_i q\phi(\mathbf{r})}$$

其中，ϕ是体系的电势，ϵ是溶液的介电常数，c_i^0和z_i分别为第i种离子的体相浓度和电荷，$\beta = \frac{1}{k_B T}$，其中k_B是Boltzman常数。该方程的雏形最早出现于双电层理论的Gouy-Chapman模型中，在这个模型中离子在电极表面附近的分布被认为是遵从玻尔兹曼分布。

如今该方程被广泛运用于各种电解质溶液体系性质的计算和分子模拟中，特别是生物体系中各种大分子（例如核酸和蛋白质）在溶液中电荷分布和溶解自由能的计算。

2. 原理

Poisson-Boltzman方程实际上是通过对体系的平均力势能(Potential of Mean Force, PMF)作平均场近似而得到。从电解质溶液体系的Poisson方程出发

$$\nabla^2\phi(\mathbf{r}) = -\frac{4\pi}{\epsilon}\sum_i z_i q c_i(\mathbf{r})$$

而第i种离子的浓度函数$c_i(\mathbf{r})$可以写成

$$c_i(\mathbf{r}) = c_i^0 e^{-\beta w_i(\mathbf{r})}$$

其中$w_i(\mathbf{r})$即为第i种离子的平均力势能。

在平均场近似中，忽略离子间的关联，令平均力势能近似等于该离子的电势能

$$w_i(\mathbf{r}) \simeq z_i q\phi(\mathbf{r}),$$

即得到Poisson-Boltzman方程。

3. 求解

Poisson-Boltzman方程是一个非线性偏微分方程，除了在特定简化体系（如Gouy-Chapman模型）中能求得解析解外，一般采用数值解法，例如有限差分法或者有限元方法。

当离子的电势能绝对值较小时，即$\beta z_i q\phi(\mathbf{r}) << 1$时，可以把泊松-玻尔兹曼方程中的指数项仅展开到一阶

$$\mathrm{e}^{-\beta z_i q\phi(\mathbf{r})} \simeq 1 - \beta z_i q\phi(\mathbf{r})$$

即可得到Debye-Hückel方程

$$\nabla^2\phi(\mathbf{r}) + \kappa^2\phi(\mathbf{r}) = 0$$

其中$\kappa^2 = \frac{4\pi z_i^2 q^2 c_i^0}{\epsilon k_B T}$。Debye-Hückel方程是一个线性偏微分方程，易于求解。在稀溶液中，德拜－休克尔方程对于泊松－玻尔兹曼方程而言是很好的近似。

4. 应用与局限

Poisson-Boltzman方程的优势在于将溶液中的水简化为具有均一介电常数的电介质，这种隐式溶剂（Implicit Solvent）的处理方法极大地简化了生物大分子溶液体系中的模拟和计算。例如，在生物大分子溶液的分子动力学模拟中，体系可以只包含生物大分子，而忽略水分子和其他离子，并采用Poisson-Boltzman方程来获得大分子的受力。类似地，对于溶解

自由能的计算，来自溶剂的贡献可以使用广义Born模型来处理，而离子的贡献则可以采用Poisson-Boltzman方程。

Poisson-Boltzman方程的缺点在于其所使用的平均场近似，当溶液中出现一定浓度高价离子导致离子间相互作用和关联增强，Poisson-Boltzman方程的解将无法解释一些由关联所产生的现象，比如带相同电荷的物体在高价盐溶液中相互吸引，以及带电胶体在高价盐溶液中的电泳呈现电荷反转，这些现象必须考虑离子间的关联才能得到合理解释。

C. 算法设计

选取$N = 30000$，$dx = 0.01$，使得求解范围为$0 - 300$:

1. 首先利用二阶微商差分公式

$$\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = \frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{\mathrm{d}x^2}$$

将Poisson-Boltzman方程左端化为一个$N \times N$的三对角行列式。再将密度n_i离散化，得到右端$\rho(1:30000)$数组。

2. 将Poisson-Boltzman方程右端指数项做一阶泰勒展开

$$\mathrm{e}^{\phi} = 1 + \phi$$

并将其代入Poisson-Boltzman方程，利用三对角矩阵求解，得到一阶线性近似的电势

$$\phi_n^{(1)} \quad n = 1, 2, ..., 30000$$

3. 得到$\phi_n^{(1)}$后，利用

$$\begin{aligned}\mathrm{e}^{\phi^{(2)}} &= \mathrm{e}^{\phi^{(1)} + (\phi^{(2)} - \phi^{(1)})} \\ &= \mathrm{e}^{\phi^{(1)}} \mathrm{e}^{(\phi^{(2)} - \phi^{(1)})} \\ &= \mathrm{e}^{\phi^{(1)}} [1 + (\phi^{(2)} - \phi^{(1)})]\end{aligned}$$

再次将Poisson-Boltzman方程右端线性化，再解新的对三角矩阵得到$\phi_n^{(2)}$，如此往复

$$\mathrm{e}^{\phi^{(n+1)}} = \mathrm{e}^{\phi^{(n)}} [1 + (\phi^{(n+1)} - \phi^{(n)})]$$

数次迭代后即可线性逼近非线性Poisson-Boltzman方程解。

4. 初步校验程序正确性。ε^2为小量，分别取值$0.01, 0.1, 0.3$，在极小时候$\phi = n_i - 1$，电势和密度形状完全相同，随着ε^2增大，电势密度开始出现差别。

D. 计算结果

我们现将$\varepsilon^2 = 0.01, 0.1, 0.3$的数据存入excel中，然后使用Python的Matplotlib包将离子浓度分布图像及Poisson-Boltzman解的图像输出。离子浓度分布随ε^2图像绘制见Figure 2.

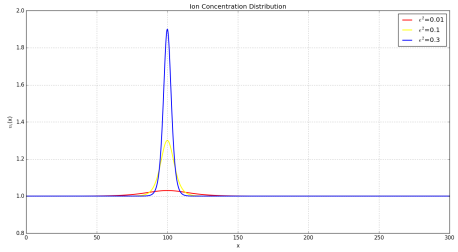

Figure 2: 离子浓度分布图

Poisson-Boltzman方程求解结果见Figure 3. 通过图像我们可以发现，随着ε^2的增大，Poisson-Boltzman方程的解趋势基本不变，但极小值越来越小。

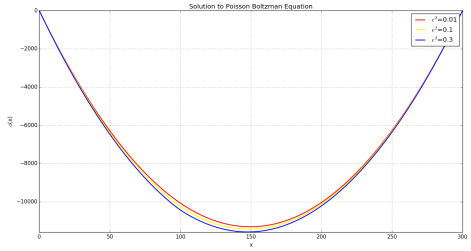

Figure 3: *Poisson-Boltzman*方程解

Appendices

A. 求解循环三对角矩阵的逆

Here is the program to solve the linear equations.

Input Python source:

```
'''
use Sherman—Morrison Formula and Thomas Method to solve cyclic tridiagonal linear
equation
'''
import numpy as np
# Thomas Method for soling tridiagonal linear equation Ax=r
# parameter: a,b,c,d are list—like of same length
# b: main diagonal of matrix A
# a: main diagonal below of matrix A
# c: main diagonal upper of matrix A
# d: Ax=d
# return: x(type=list), the solution of Ax=r
def TDMA(a,b,c,d):
    try:
        n = len(d)    # order of tridiagonal square matrix
# use a,b,c to create matrix A, which is not necessary in the algorithm
        A = np.array([[0]*n]*n, dtype='float64')
        for i in range(n):
            A[i,i] = b[i]
            if i > 0:
                A[i, i—1] = a[i]
            if i < n—1:
                A[i, i+1] = c[i]
        c_1 = [0]*n# new list of modified coefficients
        d_1 = [0]*n
        for i in range(n):
            if not i:
                c_1[i] = c[i]/b[i]
                d_1[i] = d[i]/b[i]
            else:
                c_1[i] = c[i]/(b[i]—c_1[i—1]*a[i])
                d_1[i] = (d[i]—d_1[i—1]*a[i])/(b[i]—c_1[i—1] * a[i])
        x = [0]*n # x: solution of Ax=r
        for i in range(n—1, —1, —1):
            if i == n—1:
                x[i] = d_1[i]
            else:
                x[i] = d_1[i]—c_1[i]*x[i+1]
        x = [round(_, 4) for _ in x]
        return(x)
    except Exception as e:
        return(e)
def Cyclic_Tridiagnoal_Linear_Equation(a,b,c,d):
    try:
        n = len(d)# use a,b,c to create cyclic tridiagonal matrix A
        A = np.array([[0] * n] * n, dtype='float64')
        for i in range(n):
            A[i, i] = b[i]
            if i > 0:
                A[i, i — 1] = a[i]
            if i < n — 1:
                A[i, i + 1] = c[i]
        A[0, n — 1] = a[0]
        A[n — 1, 0] = c[n — 1]
        gamma = 1 # gamma can be set freely
```

```
        u = [gamma] + [0] * (n - 2) + [c[n - 1]]
        v = [1] + [0] * (n - 2) + [a[0] / gamma]
        # modify the coefficient to form A'
        b[0] -= gamma
        b[n-1] -= a[0] * c[n-1] / gamma
        a[0] = 0
        c[n - 1] = 0
        # solve A'y=d, A'z=u by using Thomas Method
        y = np.array(TDMA(a, b, c, d))
        z = np.array(TDMA(a, b, c, u))
        x = y - (np.dot(np.array(v), y)) / (1 + np.dot(np.array(v), z)) * z
        x = [round(_,3) for _ in x]
        return(x)
    except Exception as e:
        return(e)
def main():
    a = [2, 1, 1, 1, 1]
    b = [4, 4, 4, 4, 4]
    c = [1, 1, 1, 1, 3]
    d = [7, 6, 6, 6, 8]
    x = Cyclic_Tridiagnoal_Linear_Equation(a,b,c,d)
    print('The solution is %s'%x)
main()
```

B. 求解Poisson-Boltzman方程

Here is the program to solve the Poisson-Boltzman equations.

Input Python source:

```
'''Use iterative method to find the solution of Possion—Boltzman
equation with the first type of boundary Condition'''
from numpy import *
import numpy as np
import matplotlib.pyplot as plt
import sympy
'''Input Parameter'''
N=30000
a=0;b=300
epsilon2=0.1
Z=1.0
mu=1.0
dx=(b—a)/N
'''Functions'''
def sech(x):
    return(sympy.cosh(x)**(—1))
'''Two order difference formula'''
#tridiagonal matrices
a=[];b=[];c=[]
d=np.ones(N,dtype='float64')
for i in range(N+1):
    a.append(d[i—1]/(dx*dx))
    b.append(—2*d[i—1]/(dx*dx))
    c.append(d[i—1]/(dx*dx))
a[0]=0;c[N]=0
#discrete of right side
Ni=[]
def ni(x):
    u0=1.0
    x0=100
    epsilon=double(epsilon2**(0.5))
    return(double(1+3*epsilon**(2)*u0*sech(epsilon*u0**(0.5)*(x—x0)/2)**(2)))
for i in range(N+1):
    Ni.append(ni((i+1)*dx))
'''Iterative'''
gam=np.zeros(N+1,dtype='float64')
phi=np.zeros(N+1,dtype='float64')
f=np.zeros(N+1,dtype='float64')
tolx=0.0001
while tolx>=0.0001:
    tolx=0
    p=double(b[0])—dx*dx*exp(double(f[0]))
    phi[0]=dx*dx*(exp(f[0])*(1—f[0])—Ni[0])/p
    j=0
    while j<N:
        gam[j]=c[j]/p
        p=double(b[j+1])—dx*dx*exp(double(f[j+1]))—double(a[j])*double(gam[j])
        phi[j+1]=(dx*dx*(exp(f[j+1])*(1—f[j+1])—Ni[j+1])—a[j]*phi[j]/p)
        j=j+1
    j=N
    while j>0:
        err=f[j]
        phi[j—1]=(phi[j—1]—phi[j]*gam[j])
        f[j]=phi[j]
        err=abs(err—f[j])
        if err>=tolx:
            tolx=err
        j=j—1
```

```
    f[0]=phi[0]
x=[]
for i in range(N+1):
    x.append(i*dx)
fig=plt.figure()
ax=fig.add_subplot(111)
ax.grid()
plt.plot(x,phi)
plt.xlabel('x')
plt.ylabel('$\psi(x) $',fontsize=18)
plt.title('Solution to Poisson Boltzman Equation')
plt.show()
```

C. 离子浓度分布作图部分

Here is the program to draw the figure.

Input Python source:

```
from numpy import *
import numpy as np
import matplotlib.pyplot as plt
import openpyxl
from openpyxl import Workbook
wb=openpyxl.load_workbook('Ion_concentration_distribution.xlsx')
sheet=wb.get_sheet_by_name('Sheet')
x=[];y1=[];y2=[];y3=[]
for cell in list(sheet.columns)[0]:
    x.append(float(cell.value))
for cell in list(sheet.columns)[1]:
    y1.append(float(cell.value))
for cell in list(sheet.columns)[2]:
    y2.append(float(cell.value))
for cell in list(sheet.columns)[3]:
    y3.append(float(cell.value))
plt.figure()
plt.grid()
plt.plot(x,y1,label="$\epsilon^{2}$=0.01",color="red",linewidth=2)
plt.plot(x,y2,label="$\epsilon^{2}$=0.1",color="yellow",linewidth=2)
plt.plot(x,y3,label="$\epsilon^{2}$=0.3",color="blue",linewidth=2)
plt.xlabel("x")
plt.ylabel("$\phi$(x)")
plt.title("Ion Concentration Distribution")
plt.xlim(0,300)
plt.ylim(0.8,2.0)
plt.legend()
plt.show()
```

D. Poisson-Boltzman解作图部分

Here is the program to draw the figure.

Input Python source:

```
from numpy import *
import numpy as np
import matplotlib.pyplot as plt
import openpyxl
from openpyxl import Workbook
wb=openpyxl.load_workbook('Poisson_Boltzman.xlsx')
sheet=wb.get_sheet_by_name('Sheet')
x=[];y1=[];y2=[];y3=[]
for cell in list(sheet.columns)[0]:
    x.append(float(cell.value))
for cell in list(sheet.columns)[1]:
    y1.append(float(cell.value))
for cell in list(sheet.columns)[2]:
    y2.append(float(cell.value))
for cell in list(sheet.columns)[3]:
    y3.append(float(cell.value))
plt.figure()
plt.grid()
plt.plot(x,y1,label="$\epsilon^{2}$=0.01",color="red",linewidth=2)
plt.plot(x,y2,label="$\epsilon^{2}$=0.1",color="yellow",linewidth=2)
plt.plot(x,y3,label="$\epsilon^{2}$=0.3",color="blue",linewidth=2)
plt.xlabel("x")
plt.ylabel("$\phi$(x)")
plt.title("Solution to Poisson Boltzman Equation")
plt.xlim(0,300)
plt.ylim(-11600,0.0)
plt.legend()
plt.show()
```

参考文献

[1] Sherman, Jack; Morrison, Winifred J. (1949). "Adjustment of an Inverse Matrix Corresponding to Changes in the Elements of a Given Column or a Given Row of the Original Matrix (abstract)". Annals of Mathematical Statistics. 20: 621. doi:10.1214/aoms/1177729959.

[2] Max A. Woodbury, The Stability of Out-Input Matrices. Chicago, Ill., 1949. 5 pp. MR32564.

[3] G.L. Gouy, j. de phys 9, 457 (1910).

[4] D.L. Chapman, Philos. Mag. 25, 475 (1913).

[5] Adaptive Poisson–Boltzmann Solver - A free, open-source Poisson-Boltzmann electrostatics and biomolecular solvation software package.

[6] Zap - A Poisson–Boltzmann electrostatics solver.

[7] MIBPB Matched Interface & Boundary based Poisson–Boltzmann solver.

[8] AFMPB Adaptive Fast Multipole Poisson–Boltzmann Solver, free and open-source.

[9] Donald Bashford and David A. Case GENERALIZED BORN MODELS OF MACROMOLECULAR SOLVATION EFFECTS Annu. Rev. Phys. Chem. 2000, 51, 129-152 doi:10.1146/annurev.physchem.51.1.129.

[10] Y. Levin Electrostatic correlations: from plasma to biology. Rep. Prog. Phys. 65 1577 doi:10.1088/0034-4885/65/11/201.

[11] A. Yu. Grosberg, T. T. Nguyen, and B. I. Shklovskii Colloquium: The physics of charge inversion in chemical and biological systems Rev. Mod. Phys. 74, 329 doi: 10.1103/RevModPhys.74.329.