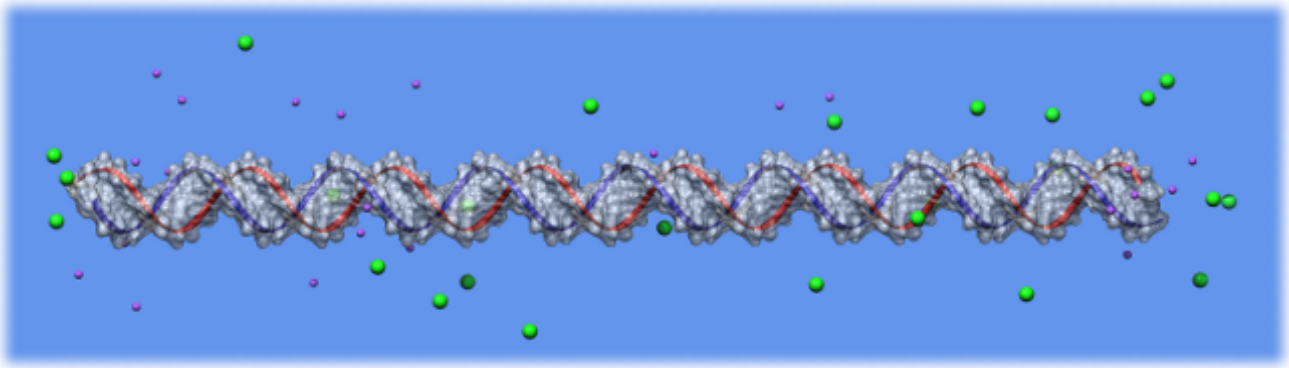
# ION software version 2.0 Users' Manual

Marcin Szypowski
Paweł Zieliński
Paweł Grochowski
Joanna Trylska




Work on the ION package was supported by the Ministry of Science and Higher Education (grant no N N301 245236).

# Contents

# 1 Introduction

Many biological processes are driven by electrostatic interactions between molecules. Therefore, the knowledge of the electrostatic potential that is generated by biomolecules is crucial for determining their physicochemical properties and possible interactions with other biomolecules. The distribution of the electrostatic potential around a molecule depends on its chemical composition and three-dimensional structure, as well as on its environment. An important component of the environment are the ions. Especially for highly charged molecules, such as nucleic acids, the distribution of counter-ions close to the surface is crucial for their functioning.

We present the ION software that determines the distribution of ions around biomolecules. The molecules and ions are modeled explicitly and represented by atomic partial charges and radii. The solvent is modeled implicitly as a continuum medium characterized by a dielectric constant (assumed 80 for water). The volume occupied by the molecule and ions is also characterized by a dielectric constant which is usually in the range 1–12.

The distribution of the electrostatic potential $\Psi$ is a solution of the Dirichlet problem for the Poisson equation:

$$\begin{aligned} -\nabla(\epsilon\nabla\Psi) &= 4\pi\rho, \quad \text{in } \Omega \\ \Psi &= F, \quad \text{on } \partial\Omega \end{aligned}$$

where $\epsilon$ is dielectric constant, $\rho$ the distribution of the electric charge in the system, $F$ the boundary condition and $\Omega$ is a three-dimensional finite box defining the area in which we calculate the solution. For molecules of irregular shapes the analytical solution of the Poisson equation is not known and numerical methods need to be applied.

In the Poisson-Boltzmann model the finite size of ions is ignored, and their electric charge is represented in the water environment with the Boltzmann distribution. For many applications such approximation is acceptable and found in many electrostatic software packages. However, for highly charged systems, such as nucleic acids, a more detailed description of ionic environment accounting for ion sizes is often needed. In order to account for the ions of finite size one can solve the Poisson equation with explicit treatment of ions. The solution has to be repeated for various ion configurations determined for example by applying the Monte-Carlo (MC) dynamics.

With the ION software one can perform the MC dynamics accounting for the finite size of ions. ION uses the PETSc package (http://www.mcs.anl.gov/petsc/petsc-as/) for calculating the solution of the Dirichlet problem for the Poisson equation in every iteration of the MC. Also, ION is fully parallelized with the MPI technology.

# 2 Software

In this section we describe how to install and use the ION software.

## 2.1 Requirements

ION is designated for Linux platforms and is distributed as a source code written in the C language. Therefore, a C compiler is required for installation. The compilation was tested with the GNU Compiler Collection (gcc) ver. 4.6.3. ION also uses the PETSc package (the recommended version is 3.3-p6) with Message Passing Interface (MPI) (see: http://www.mcs.anl.gov/petsc/petsc-as/). To install ION and PETSc the root privileges are not required.

## 2.2 Installation of PETSc

One should download the PETSc sources from http://www.mcs.anl.gov/petsc/petsc-as/ (in this example we use PETSc 3.3-p6). The file named `petsc-3.3-p6.tar.gz` should be unpacked in the user's directory:

```
tar -xzf petsc-3.3-p6.tar.gz
```

The above command should create the `petsc-3.3-p4` directory. For configuration one can use the scripts in the `petsc-3.3-p6/config` directory. For AMD 64-bit processor machines select the script `arch-gcc64-real.py` and in the `petsc-3.3-p6` directory type:

```
./config/examples/basic/arch-gcc64-real.py
```

For 32-bit processor machines select the script `arch-gcc-real.py` and in the `petsc-3.3-p6` directory type:

```
./config/examples/basic/arch-gcc-real.py
```

More configuration scripts can be found in `./config/examples/basic/` or in `./config/examples/`. One can obtain information on machine architecture by the `arch` command, x86 indicates a 32-bit machine and x86_64 indicates the AMD 64-bit processor. If the configuration ends successfully then one should set the environmental variables `PETSC_DIR` and `PETSC_ARCH`. The configuration script will provide this path for the selected architecture.
We recommend to add the following lines:

```
setenv PETSC_DIR full_path_to_petsc_instalation_directory
setenv PETSC_ARCH name_of_petsc_instalation
```

if you use csh or tcsh

or

```
export PETSC_DIR=full_path_to_petsc_instalation_directory
export PETSC_ARCH=name_of_petsc_instalation
```

if you use bash, sh or rc in the file .bashrc, .profile or .cshrc (depending on the type of the shell).

In order to compile the sources type:

```
make all test
```

## 2.3 ION Download

The source code and the manual are available from https://bionano.cent.uw.edu.pl/Software. After downloading unpack the sources:

```
tar -xvf ION.tar.gz
```

The above command should create the ION directory.

## 2.4 Code organization

The code is located in the top-level directory ION/ with the following subdirectories:

**src** - contains the source codes and the header file

**bin** - a directory to place binary files

**manual** - contains the manual

**example** - contains exemplary input files

## 2.5 Installation

To compile the sources and create the executable file you must use the makefile from the ION/ directory. Just type in the terminal:

```
make ION
```

If everything was successful, an executable file ION in the bin/ subdirectory should be created.

## 2.6 Usage

The easiest way to run the ION program is by using the makefile in the ION/ directory (the same file that was used in order to compile the sources). In this file you can find the following lines:

```
runION:
        -@${MPIEXEC} -np 2 ./bin/ION example/in.txt;
```

The second line starts with a variable that comes from the PETSc package and runs MPI processes (if you are not familiar with PETSc and with the makefile syntax do not change this). The `-np` option indicates the number of processors destined to run a job (in the above case it is 2). The `./bin/ION` executes the file ION from the bin/ directory. The argument `example/in.txt` names the control file which contains user-defined parameters for the simulation (see 2.7).
To run the simulation one should use:

```
make runION
```

For sequential processing one should use:

```
./bin/ION example/in.txt;
```

## 2.7 Control file and keywords

ION needs as an input parameter the name of an input control file. It is an ascii file that contains all parameters needed to define the simulation. Each line in the control file, except the comments, is constructed as follows:

`keyword value`

and is terminated with the `EOL` sign. If the first word in the line does not correspond to available keywords this line is treated as misspelled and the program stops. Otherwise, the second word is read as the parameter's value. Any word following the keyword and the value in the same line is treated as misspelled and the program stops with an error. Every position starting with '#' or '%' is treated as a comment and ignored.

All variables are implicitly set to zero, a default modifier indicates that a variable has another begin value. Description of the box:
**lx** *value* - the length of the box in the first direction (in Å)
**ly** *value* - the length of the box in the second direction (in Å)
**lz** *value* - the length of the box in the third direction (in Å)
**directionx** *number* - the number of discretization steps in the first direction (integer)
**directiony** *number* - the number of discretization steps in the second direction (integer)
**directionz** *number* - the number of discretization steps in the third direction (integer)

Free ions added by ION in the solvent (next to the ions read from the PQR file – see 3.1):
**ionk** *name* - the name of the **k**-th type of the ion (we recommend to use the element's chemical capitalized names such as NA, MG, CL etc. consistently with the names of ions that may appear in the PQR file, compare with the section 3.1
**ionk**n *number* - the number of **k**-th type ions (integer)
**ionk**r *value* - the radius of the **k**-th type ion (in Å)
**ionk**dr *value* - the $\Delta R_k$ of the **k**-th type ion (in Å)
**ionk**w *value* - the $w_k$ of the **k**-th type ion (in Å)
**ionk**ch *value* - the charge of the **k**-th type ion (in $e$)
Note: For **k** should be used consecutive integers starting with 1.

MC parameters:
**temperature** *value* - temperature of the system in K (default 300 K)
**niterations** *number* - the number of iterations of the Monte Carlo algorithm (integer)
**ionsDelta** *value* - ion's displacement in Å with random distance between 0 and *value* (default 1.0Å)
**ionsOneByOne** *yes/no* - if this parameter is set to *yes* the ions are moved in a fixed sequence during the MC algorithm. In the case of *no* – in random order by the shuffled cyclic method (default is *no*)
**nIonsToMove** *number/all* - the number of ions moved in one iteration of the Monte Carlo algorithm in a fixed or random sequence depend on **ionsOneByOne** parameter. If this parameter is set to *all* then all ions are displaced (default 1)
**enableSelfEnergy** *yes/no* - if the value of this parameter is *yes* the self energy of charges is eliminated from the total energy in the MC algorithm. Otherwise, (*no*) the self energy is not calculated (default *yes*)

Dielectric constant:
**solventEps** *value* - the dielectric constant of the solvent (default 80.0)

**moleculeEps** *value* - the dielectric constant inside the molecule and ions (default 4.0)
**wA** *value* - the width (see $w$ in Fig1) of the intermediate region between the areas with different dielectric constants for atoms of the molecule (default 1 Å)
**wI** *value* - the width (see $w$ in Fig1) of the intermediate region between the areas with different dielectric constants used for ions with no ionkw specified (default 1 Å)
**w_deltaR_ion** *value* - the value of the $\Delta R$ parameter (Fig. 1) used for ions with no ionkdr specified (default 1.0 Å)
**w_deltaR_atom** *value* - the value of the $\Delta R$ parameter (Fig. 1) for atoms of the molecule (default 1.0 Å)

Files:
**inputMolecule** *name* - the name of the input file of the molecule (in the PQR format)
**outputStatistics** *name* - the name of the output file with statistics from the MC algorithm
**outputStatesXYZ** *name* - the name of the output file in the XYZ format with the positions of ions in the solvent in selected MC steps as defined by the **outputStates_freq** parameter
**outputStatesPQR** *name* - the name of the output file in the PQR format with the positions of ions in the solvent in selected MC steps as defined by the **outputStates_freq** parameter
**outputStates_freq** *number* - how frequently to write the positions of ions in the solvent to the PQR file and XYZ file (default 0 – not outputted)
**iondensity** *name* - the root name of the files with average positions of different ions during the MC algorithm (in the DX format)
**potential** *name* - the name of the output file in the DX format with average potential calculated during the MC algorithm
**restartCoords** *name* - the name of the restart file (in the PQR format) with the positions of ions in the solvent. When restarted the program ignores the ions from the PQR file and defined by the **ionk** parameter

Others:
We discretize the position of the mobile particle (ion) with a fine resolution, e.g (**Mnx**, **Mny**, **Mnz**).
**Mnx** *value* - subdivision of the unit cell for the self-energy computation in the first direction. Default: 10.
**Mny** *value* - subdivision of the unit cell for the self-energy computation in the second direction. Default: 10.
**Mnz** *value* - subdivision of the unit cell for the self-energy computation in the third direction. Default: 10.
**qrSource** *mixed/pqr/input* - the source of the radius and charge for ions. Default: mixed.
**seed** *number* - the seed for the random number generator. If omitted the seed is based on the present time.
**timing** *yes/no* - estimates the time needed to finish the simulation. If this option is turned on, the program returns the estimated time to the end of the calculation after the first step of the MC algorithm.
**rtol** *value* - tolerance for solving the Poisson equation (see section 3.4, default 0.00001)
**extraVIons** *value* - the minimal separation between ions in the solvent (in Å). This separation is measured as the distance between the centers of ions minus the sum of their radii (default 1.76 Å)
**extraVAtom** *value* - the same as above but between the atoms in the molecule and ions in the solvent

(default 1.76 Å)
**distMol** *value* - if the distance (in Å) between an ion from the PQR file and the nearest atom in the molecule is smaller than this value, this ion is treated as a part of the molecule (default 3.0 Å)
**distSolv** *value* - if the distance (in Å) between an ion from the PQR file and the nearest atom in the molecule is greater than this value, this ion is treated as a "free-ion" (default 3.0 Å)
**centerMolecule** *yes/no* - if this parameter is set to *yes* the molecule is centered in the box $\Omega$ (see 3.1, default *yes*)
**minDistIonFromBorder** *value* - the minimal distance of ions from the border of the box. If the specified value is smaller than $w_k + \Delta R_k$ then the default of $w_k + \Delta R_k$ is enforced (in Å)

## 2.8 Example control file

```
######################################
# perform calculation for a molecule #
# in the trna.pqr file               #
# PDB code: 1GIX                     #
######################################
inputMolecule example/trna.pqr

######################################
# box 90A x 100A x 110A              #
# Note: The molecule is centered     #
# in the box                         #
######################################
lx 90.0
ly 100.0
lz 110.0

######################################
#     use a 1A discretization step   #
######################################
directionx 90
directiony 100
directionz 110

######################################
#   add 50 magnesium ions with       #
#   radii of 1A and charges of +2e   #
######################################
ion1 MG
ion1n 50
ion1r 0.72
ion1ch 2.0
```

```
#######################################
#    add 50 sodium ions with          #
#    radii of 1A and charges of +1e   #
#######################################
ion2 NA
ion2n 50
ion2r 0.98
ion2ch 1.0

#######################################
#  and add 76 chlorine ions with      #
#  radii of 2A and charges of -1e     #
#######################################
ion3 CL
ion3n 76
ion3r 1.81
ion3ch -1.0

#######################################
#   perform 5 iterations of           #
#   the Monte Carlo (MC)              #
#   algorithm in temperature 300 K    #
#######################################
temperature 300.0
niterations 5

#######################################
# during MC move one ion by 1A        #
#######################################
ionsDelta 1.0

#######################################
# Set the distance between atoms      #
# of the molecule and the closest ion #
# larger than 1.76A and the minimal   #
# distance between ions as 1.76A      #
#######################################
extraVAtom 1.76
extraVIons 1.76

#######################################
# write statistics from MC to         #
# a file states.txt                   #
#######################################
outputStatistics states.txt
```

```
#######################################
# write density of ions after MC     #
# to files density_NA.dx             #
# and density_MG.dx density_CL.dx    #
#######################################
iondensity density_

########################################
# write accepted positions of          #
# ions to states.pqr file in the PQR   #
# format in all iterations of MC       #
########################################
outputStatesPQR states.pqr
outputStates_freq 1

#######################################
# Set the dielectric constant of     #
# the solvent to 80 and of the       #
# molecule and ions to 4             #
# Set the width of the intermediate  #
# regions to 0.5A                    #
#######################################
solventEps 80.0
moleculeEps 4.0
wA 2.0
wI 1.2

#######################################
# To generate a pseudorandom         #
# number by the user                 #
#######################################
seed 123

#######################################
# How long do I have                 #
# to wait for the results?           #
#######################################
timing yes
```

### 2.9 File formats

Here we list the formats used in the ION software to represent the data. The first format is PQR which is described at http://apbs.wustl.edu/MediaWiki/index.php/PQR_format. The file in this format is used to describe the input molecule as well as output coordinates of the free ions from the MC simulation. The PQR file contains information about atoms, residues, coordinates, atomic radii and charges.
The second format is the XYZ format which contains the output coordinates of free ions from the MC simulation.
Average ion distributions can be written in the DX Data file format (DX) (http://www.cs.wpi.edu/research/DataExplorer/tutorial/data.html#dx). The name of the file including the statistics of the Monte-Carlo algorithm is specified as the value of the `outputStatistics` keyword. Every row of this file contains the configuration number, the number of the MC steps when this configuration has not changed and the energy of this configuration.

## 3 Theory and Methods

In this section we briefly describe the methods which are implemented in the ION software.

### 3.1 Preliminary steps

First, the center of a molecule is located in the center of the box $\Omega$. For this purpose the center of the molecule is defined as $(\frac{x_{min}+x_{max}}{2}, \frac{y_{min}+y_{max}}{2}, \frac{z_{min}+z_{max}}{2})$ where

$$\begin{aligned} x_{min} &= \min_i\{x : x = x_i - R_i\} \qquad x_{max} = \max_i\{x : x = x_i + R_i\} \\ y_{min} &= \min_i\{y : y = y_i - R_i\} \qquad y_{max} = \max_i\{y : y = y_i + R_i\} \\ z_{min} &= \min_i\{z : z = z_i - R_i\} \qquad z_{maz} = \max_i\{z : z = z_i + R_i\}, \end{aligned}$$

$i$ numbers the atoms in the input PQR file (including the atoms of the molecule) and ions (if structural ions are specified in this file), $R_i$ are atomic radii and $(x_i, y_i, z_i)$ are atomic coordinates read from the PQR file. Note: ionic coordinates which are saved in the PQR and XYZ outputs, as well as data in the DX files are translated back to the original position of the molecule specified in the PQR input file.

Second, the program checks if the molecule has enough space in the box i.e., if each atom is away from the box boundaries by more than $w_A + R_i$ (where $w_A$ is a `wA` parameter from the input file and $R_i$ are atomic radii from the PQR file):

$$\min\{|\mathbf{r}_i - \mathbf{r}| : \mathbf{r} \in \partial\Omega_h\} \geq R_i + w + \Delta R_{atom} \tag{1}$$

where $i$ numbers the atoms in the molecule, $\mathbf{r}_i$ are atomic positions, $\partial\Omega_h = \partial\Omega \cap \bar{\Omega}_h$ (see 3.2 and 5) and $\Delta R_{atom}$ is the value of the `w_deltaR_atom` parameter. If any atom is too close to the box boundary, the program terminates.

Third, ION searches and classifies the ions from the input PQR file with the following procedure:

- Find all ions in the PQR file whose type is defined in the input file (including the types with **ionk**n 0). Other ions which are not listed in the input file by **ionk** *name* are treated as atoms belonging to the molecule.
- Next, for every ion, check the minimal distance between the center of the ion and the center of any heavy atom in the molecule.
  - If this distance is smaller or equal than the value of the `distMol` parameter then treat this ion as an atom that belongs to the molecule.
  - If this distance is larger than `distSolv` then the ion is either treated as a free ion or it is ignored if `restartCoords` file is provided.
  - In any other case the program terminates after printing information about wrong line.

If `restartCoords` file is not provided then additional free ions defined with the `ionk` parameter are randomly distributed in the box $\Omega$. We discretize position of the free ions with a fine resolution, e.g (**Mnx**, **Mny**, **Mnz**). While distributing the ions in the solvent the ION program ensures that the ions cannot be closer to each other than specified by `extraVIons` (eq. 2) and closer to the nearest atom in the molecule than specified by `extraVAtom` (eq. 3). Also, the ions must be away from the border of the box by at least max($w_{I_k} + R_{I_k}$, `minDistIonFromBorder`). The above conditions can be expressed as

$$|\mathbf{r}_i - \mathbf{r}_j| \quad \geq \quad R_i + R_j + \delta_{ion} \tag{2}$$

where $i$, $j$ number all ions, $\mathbf{r}$ are ion positions, $R$ ion radii, $\delta_{ion}$ is the value of the `extraVIons` parameter,

$$|\mathbf{r}_i - \mathbf{r}_j| \quad \geq \quad R_i + R_j + \delta_{atom} \tag{3}$$

where $i$ numbers all ions, $j$ numbers all atoms, $\mathbf{r}$ are atomic or ionic positions, $R$ are atomic or ionic radii, $\delta_{atom}$ is the value of the `extraVAtom` parameter. The third condition is

$$\min\{|\mathbf{r}_i - \mathbf{r}| : \mathbf{r} \in \partial\Omega_h\} \geq \max\{R_i + w_{I_k} + \Delta R_{I_k}, R_{min}\} \tag{4}$$

where $i$ numbers all ions, $\mathbf{r}$ are ionic positions, $R$ ionic radii, $w_{I_k}$ is the value of the **ionk**w parameter or `wI` if **ionk**w not provided, $\Delta R_{I_k}$ is the `w_deltaR_ion` parameter and $R_{min}$ is `minDistIonFromBorder`. If `restartCoords` file is provided the free ions are read from the restart PQR file.

The ION program also calculates the volume of the solvent ($V_{solv}$). First, ION calculates the volume of the molecule ($V_{mol}$). It is done by summing up the grid points which are inside the spheres centered on atomic coordinates with their radii defined in the input PQR file. For this calculation ION uses a finer grid with the resolution decreased by a factor of 2 in comparison with the original one. Then ION calculates the volume of the box ($V_{box} = l_1 l_2 l_3$) and the volume of the solvent:

$$V_{solv} = V_{box} - V_{mol}$$

The volume of the solvent is used to calculate the concentration of free ions in the solvent. For the $i$-th ion this concentration is defined as

$$C_k = \frac{n_k}{V_{solv} N_A} 10^{27} \ \frac{\text{mol}}{\text{dm}^3}$$

where $n_k$ is the number of ions of $k$-th type, and $N_A$ is the Avogadro constant ($N_A = 6.02214179 \times 10^{23}\mathrm{mol}^{-1}$).

To visualize the system ION creates two files: epsilonmol.dx and box.dx. The first file contains the values of the dielectric constant in every point of the box corresponding to the molecule and the second file contains the borders of the $\Omega$.

## 3.2 Poisson equation

We consider the Poisson problem for the Dirichlet boundary condition:

$$\begin{aligned} -\nabla(\epsilon(r)\nabla\Psi(r)) &= 4\pi\rho(r), \quad \text{for } r \in \Omega \\ \Psi(r) &= F(r), \quad \text{for } r \in \partial\Omega \end{aligned}$$

where:
$\Omega = [0, l_1] \times [0, l_2] \times [0, l_3]$ defines a region where we search for the solution;
$\Psi(r)$ - is the electrostatic potential;
$\epsilon(r)$ - is the function of the dielectric constant;
$\rho(r)$ - is the charge density in $\Omega$ considering the distribution of atoms in the molecule and ions in the solvent:

$$\rho(r) = \sum_i q_i \delta(\mathbf{r} - \mathbf{r}_i)$$

where $i$ number both the atoms and ions in the system, $\mathbf{r}_i$ is the position of the $i$-th atom or ion, $q_i$ is the charge and $\delta(\mathbf{r})$ is the Dirac delta function.
$F(r)$ defines a boundary condition defined as

$$F(\mathbf{r}) = \frac{1}{4\pi\epsilon_s} \sum_i \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|}$$

where $i$ number both the atoms and ions in the system, $\mathbf{r}_i$ is the position of the $i$-th atom or ion, $q_i$ is the charge and $\epsilon_s$ defines the dielectric constant in the solvent.
We define the function of the dielectric constant $\epsilon(r)$ as

$$\epsilon(r) = \epsilon_m + (\epsilon_s - \epsilon_m) \prod_i H_i(|\mathbf{r} - \mathbf{r}_i|, R_i + \Delta R_i, w_i)$$

where

$$H_i(r, R_i + \Delta R_i, w_i) = \begin{cases} 0 & r \le R_i + \Delta R \\ 3(\frac{r - R_i - \Delta R_i}{w_i})^2 - 2(\frac{r - R_i - \Delta R_i}{w_i})^3 & R_i + \Delta R_i < r < R_i + \Delta R + w_i \\ 1 & r \ge R_i + \Delta R_i + w_i \end{cases}$$

and $\epsilon_m$ and $\epsilon_s$ are the values of the dielectric constant inside the molecule (the `moleculeEps` parameter) and in the solvent (the `solventEps` parameter), respectively, $\mathbf{r}_i$ is the position of the center of the $i$-th particle, $R_i$ is the radius of the particle, $w$ is the width of the interface between the areas with different dielectric constants (the `w` parameter), $\Delta R$ is the displacement of the interface (equal to the value of the `w_deltaR_ion` parameter for free ions and `w_deltaR_atom` for atoms in the molecule), and $i$ numbers all atoms and ions. Fig. 1 shows a chart of the dielectric constant function in one dimension for one particle.

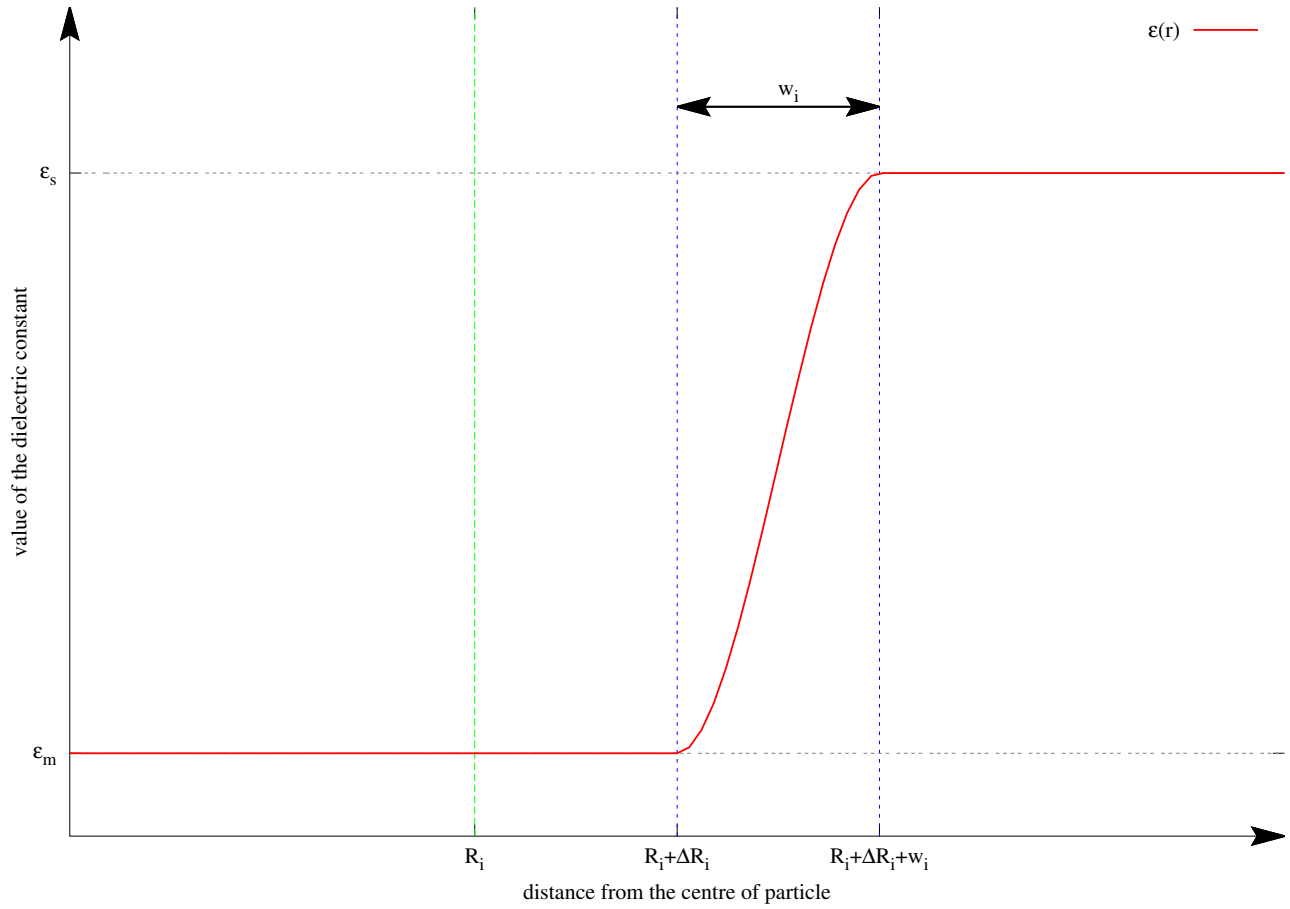



Figure 1: Plot of the dielectric constant function in one dimension for the $i$-th particle.

## 3.3 Discretization – the Finite Difference Method

In order to solve the Poisson equation 3.2 we use the finite difference method. The discretization of the grid is

$$\bar{\Omega}_h = \{(i_1 h_1, i_2 h_2, i_3 h_3), i_p = 0, 1, \ldots, N_p + 1,\ (N_p + 1)h_p = l_p,\ p = 1, 2, 3\}. \tag{5}$$

where we define $h = max\{h_1, h_2, h_3\}$ as the discretization step. $l_1, l_2, l_3$ corespond to the parameters `lx`, `ly` and `lz` and $N_1 + 1, N_2 + 1, N_3 + 1$ to `directionx`, `directiony` and `directionz`
If $u_{i,j,k} = u(ih_1, jh_2, kh_3)$ is the value of the $u$ function at the $(ih_1, jh_2, kh_3)$ point we can define the forward differences

$$\partial_1 u_{i,j,k} = (u_{i+1,j,k} - u_{i,j,k})/h_1$$
$$\partial_2 u_{i,j,k} = (u_{i,j+1,k} - u_{i,j,k})/h_2$$
$$\partial_3 u_{i,j,k} = (u_{i,j,k+1} - u_{i,j,k})/h_3$$

and backward differences

$$\begin{aligned}\bar{\partial}_1 u_{i,j,k} &= (u_{i,j,k} - u_{i-1,j,k})/h_1 \\ \bar{\partial}_2 u_{i,j,k} &= (u_{i,j,k} - u_{i,j-1,k})/h_2 \\ \bar{\partial}_3 u_{i,j,k} &= (u_{i,j,k} - u_{i,j,k-1})/h_3\end{aligned}$$

The positions of particle centers do not necessarily correspond to the positions of the grid points so we distribute each particle charge to at most eight nearest grid points with the following weight formula: $(1 - \frac{\tilde{x_1}}{h_1})(1 - \frac{\tilde{x_2}}{h_2})(1 - \frac{\tilde{x_3}}{h_3})$, where $\tilde{x}_i$ is the distance form the grid point to the center of the particle along the $i$-th coordinate.
We define the charge density at the $(ih_1, jh_2, kh_3)$ position as

$$\rho_{i,j,k} = \frac{Q_{i,j,k}}{h_1 h_2 h_3} \tag{6}$$

where $Q_{i,j,k}$ is the value of the charge on the grid point.
Then we can represent the Poisson equation at $(ih_1, jh_2, kh_3)$ as

$$-\Big(\partial_1(\epsilon_{i-\frac{1}{2},j,k}\bar{\partial}_1\Psi_{i,j,k}) + \partial_2(\epsilon_{i,j-\frac{1}{2},k}\bar{\partial}_2\Psi_{i,j,k}) + \partial_3(\epsilon_{i,j,k-\frac{1}{2}}\bar{\partial}_3\Psi_{i,j,k})\Big) = 4\pi\rho_{i,j,k} \tag{7}$$

When we substitute the forward and backward differences, and define

$$E_1^+ = \frac{1}{h_1^2}\epsilon_{i+\frac{1}{2},j,k},\ E_2^+ = \frac{1}{h_2^2}\epsilon_{i,j+\frac{1}{2},k},\ E_3^+ = \frac{1}{h_3^2}\epsilon_{i,j,k+\frac{1}{2}} \tag{8}$$

$$E_1^- = \frac{1}{h_1^2}\epsilon_{i-\frac{1}{2},j,k},\ E_2^- = \frac{1}{h_2^2}\epsilon_{i,j-\frac{1}{2},k},\ E_3^- = \frac{1}{h_3^2}\epsilon_{i,j,k-\frac{1}{2}} \tag{9}$$

$$E = E_1^+ + E_1^- + E_2^+ + E_2^- + E_3^+ + E_3^- \tag{10}$$

we obtain a linear system of equations

$$\begin{gathered}-E_1^+\Psi_{i+1,j,k} - E_2^+\Psi_{i,j+1,k} - E_3^+\Psi_{i,j,k+1} + E\Psi_{i,j,k} + \\ -E_1^-\Psi_{i-1,j,k} - E_2^-\Psi_{i,j-1,k} - E_3^-\Psi_{i,j,k-1} = 4\pi\rho_{i,j,k} \\ \text{for } (ih_1, jh_2, kh_3) \in \Omega_h \text{ and}\end{gathered} \tag{11}$$

$$\Psi_{i,j,k} = F_{i,j,k} \text{ for } (ih_1, jh_2, kh_3) \in \Gamma_h \tag{12}$$

This system of equations can be written in the matrix form.

$$A\mathbf{u}_h = \mathbf{f}_h \tag{13}$$

where $A$ is the matrix of the coefficients defined in eq. (8 - 10), $\mathbf{u}_h$ is the solution vector at points inside $\Omega$

$$\mathbf{u}_h = [\Psi_{1,1,1}, \Psi_{2,1,1}, \ldots, \Psi_{N_1,1,1}, \Psi_{1,2,1}, \Psi_{2,2,1}, \ldots, \Psi_{N_1,N_2,N_3}]^T$$

and $\mathbf{f}_h$ is the vector of the right hand side of the Poisson equation considering the border values

$$\mathbf{f}_h = \begin{bmatrix} 4\pi\rho_{1,1,1} - E_1^- F_{0,1,1} - E_2^- F_{1,0,1} - E_3^- F_{1,1,0} \\ 4\pi\rho_{2,1,1} - E_2^- F_{2,0,1} - E_3^- F_{2,1,0} \\ 4\pi\rho_{3,1,1} - E_2^- F_{3,0,1} - E_3^- F_{3,1,0} \\ \vdots \\ 4\pi\rho_{N_1-1,1,1} - E_2^- F_{N_1-1,0,1} - E_3^- F_{N_1-1,1,0} \\ 4\pi\rho_{N_1,1,1} - E_1^+ F_{N_1+1,1,1} - E_2^- F_{N_1,0,1} - E_3^- F_{N_1,1,0} \\ 4\pi\rho_{1,2,1} - E_1^- F_{0,2,1} - E_3^- F_{1,2,0} \\ 4\pi\rho_{2,2,1} - E_3^- F_{2,2,0} \\ 4\pi\rho_{3,2,1} - E_3^- F_{3,2,0} \\ \vdots \\ 4\pi\rho_{N_1-1,2,1} - E_3^- F_{N_1-1,1,0} \\ 4\pi\rho_{N_1,2,1} - E_1^+ F_{N_x,2,1} - E_3^- F_{N_1,2,0} \\ \vdots \\ 4\pi\rho_{N_1,N_2,1} - E_1^+ F_{N_1+1,N_2,1} - E_2^+ F_{N_1,N_2+1,1} - E_3^- F_{N_1,N_2,0} \\ 4\pi\rho_{1,1,2} - E_1^- F_{0,1,2} - E_2^- F_{1,0,2} \\ \vdots \\ 4\pi\rho_{N_1,N_2,N_3} - E_1^+ F_{N_1+1,N_2,N_3} - E_2^+ F_{N_1,N_2+1,N_3} - E_3^+ F_{N_1,N_2,N_{3+1}} \end{bmatrix}$$

## 3.4 Solving the discretized Poisson equation

Due to the properties of the discretization scheme, the $A$ matrix in eq. 13 is symmetric and positive defined. The size of this matrix can be very large but the matrix is sparse. This is the reason why the preconditioned conjugate gradient (PCG) method is used to solve the eq. 13. As a preconditioner the Block Jacobi Method is used with incomplete Cholesky factorization at every block. The number of blocks is equal to the number of processors used by the program. The iterative PCG method is terminated when

$$||M^{-1}Au_h - M^{-1}f_h||_2 \quad < \quad \eta||M^{-1}f_h||_2 \tag{14}$$

where $M$ is the preconditioner matrix and $\eta$ is the parameter that can be tuned in the control file with a keyword `rtol`, and $||\cdot||_2$ denotes the Euclidean norm.

## 3.5 Monte Carlo Method

The solution of the Poisson equation is used to calculate the electrostatic energy of the system

$$E = \frac{k_e}{2}\sum_{i,j,k}\Psi_{i,j,k}Q_{i,j,k} - E^M_{self} - \sum_I E^I_{self},$$

where $k_e = 1/(4\pi\epsilon_0) \approx 1389.35$ Å$e^{-2}$kJ/mol. The first term on the right side of the above expression is the electrostatic energy of the entire system, $E^M_{self}$ represents the self-energy of macromolecule charges due to the discretization of the electrostatic potential on the grid, and $E^I_{self}$ is the electrostatic energy of a single $I$-th ion in pure water. Next, the electrostatic energy of the system is used in the

Monte Carlo algorithm in the following procedure:

- We choose a random ion of the solvent and move it in a random direction for a random distance (maximal `ionsDelta` interval). After that its position is bound to subdivision.
- If the chosen ion is too close to the molecule (eq. 3), another ion (eq. 2) or the border of the box, the ion is moved back to its previous position and we go back to the previous step.
- Otherwise, we calculate the electrostatic energy ($E_{new}$) for the new state.
  - if $E_{new}$ is lower than the energy of the previous accepted state ($E_{old}$), then we accept this state
  - otherwise we draw a number $rnd$ from the interval $[0, 1]$ and check the condition

    $$e^{-(E_{new}-E_{old})/k_BT} > rnd$$

    where $T$ is the temperature (the `temperature` parameter) and $k_B$ is the Boltzman constant. If the condition is satisfied, we accept this state.
  - If the movement of the ion is not accepted, the ion is moved back to the previous position and the old state is accounted for in the statistical ensemble for the second time.

The number of iterations of the MC algorithm depends on the `niterations` parameter.