FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

Numerical Solution of the Poisson-Nernst-Planck Equation System

Siegfried Schöfer

Bachelor Thesis

Numerical Solution of the Poisson-Nernst-Planck Equation System

Siegfried Schöfer

Bachelor Thesis

Aufgabensteller:	Prof. Dr. U. Rüde
Betreuer:	Dipl.-Ing.(FH) D. Bartuschat, M.Sc.(hons)
Bearbeitungszeitraum:	13.01.2013 – 13.06.2013

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 12. Juni 2013 .....

Contents

1 Introduction

1.1 Motivation

Lab-on-a-chip systems are following the path of miniaturization. Smaller chips require less sample substances, promise quicker results and point-of-care diagnostics. Active research projects considering microfluidics and lab-on-a-chip devices are undertaken not only at the chair of system simulation at FAU. Recently, an international research project, called "DNA on Disk", was initiated with the goal to diagnose cancer, neurological defects or other diseases like malaria or tuberculosis with a few drops of blood [1]. Lab-on-a-chip devices require new techniques for sample transportation and mixing as traditional methods like pressure flow lose their efficiency on small scales [2, p. 243f.]. Therefore, electroosmosis is a key element for microfluidic devices.

Electric double layer theory plays a huge role in understanding electroosmotic transport in microfluidic systems. Recently, Masilamani wrote his master thesis about electroosmotic flow [3]. In his research, he assumed that the electric double layer formed much faster than the fluid flow developed. Since the formation of the electric double layer is analyzed in this thesis, this assumption is not reasonable anymore. Instead of using the Poisson-Boltzmann equations, which already describe the electric double layer in its final shape, the nonlinear coupled Poisson-Nernst-Planck equation system has to be analyzed and solved. This is done numerically with a combination of the finite difference and the finite volume method. In addition to the computation of the potential distribution, the ion concentrations of the different species have to be calculated, too.

If everything works out, the steady-state solution of the simulation will match the results of the analytic formula referenced in Masilamani [3]. In that case, the way is paved for applying the Poisson-Nernst-Planck equation system to electrophoretic phenomena [2, p. 295-361].

1.2 Overview

In section 2, this work starts with an introduction to electroosmotic flow, which is best understood by electric double layer theory. With knowledge of the physics, section 3 approaches the mathematical form, and the Poisson-Nernst-Planck equation system is introduced. These equations are applied to a microchannel geometry which is explained in section 4. Additionally, boundary conditions are described and derived in this section. The numerical discretization and the solution approach based on a mixed finite difference and finite volume approach is presented in sections 5 and 6. After heaving dealt with numerics, the implementation in three dimensions with waLBerla is described partially in section 7. Due to better analysis possibilities, the implementation was also performed in one dimension with C++ (see section 8). Important aspects are analyzed in this two chapters. In addition, a conclusion can be found in section 9, which summarizes the results.

2 Electroosmotic flow and electric double layer theory

2.1 Electroosmotic flow

> Electroosmosis is associated with the movement of a bulk electrolyte solution or a liquid carrying a free charge, relative to a stationary charged surface, under the influence of an imposed electric field [2, p. 229].

The purpose of this section is to bring the definition made by [2] to life. A natural incarnation of a microchannel is a glass capillary. Two important observations can be made. First, the channel walls works as insulator. This means that current cannot penetrate the walls. Second, the wall carries a charge. In case of glass it is a negative charge, which builds up due to adsorption effects or the dissociation of surface silanol groups [2, p. 229-231]. A more complete overview of charge building mechanisms can be found in [2, p. 106f.].

If the channel is filled with a dilute ionic solution, the ions screen the field of the charged walls. A negatively charged wall will attract positively charged ions from the solution according to the laws of electrostatics. These attracted ions, together with the charged wall, constitute the electric double layer. In the bulk fluid, near the middle of the channel, the ion species are mostly balanced.

Thus, only the electric double layer will experience a body force, if an external electric field is applied. The rest of the fluid is put in motion by viscous effects.

A schematic representation of the setting can be found in figure 1.

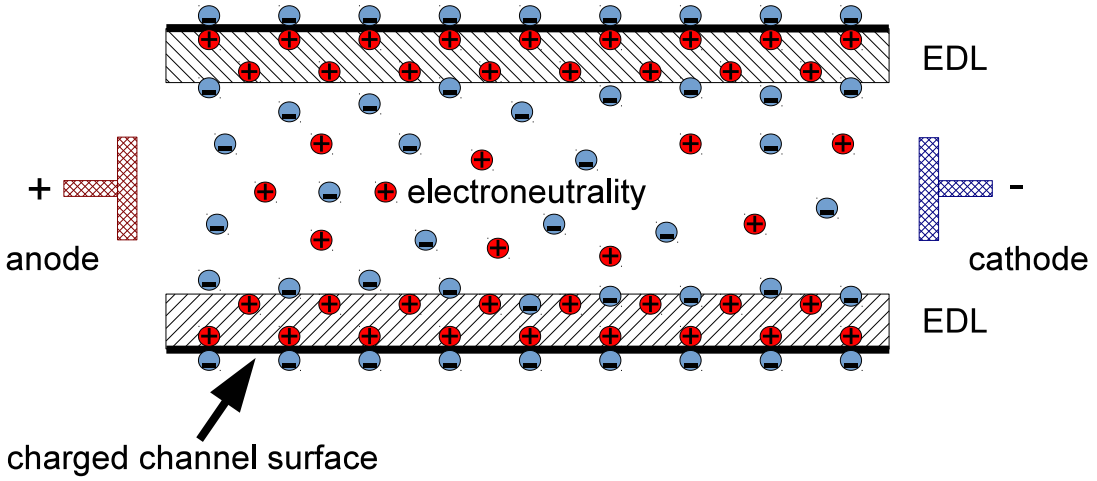


Figure 1: Based on figure 8.2 in [2]. This is a two-dimensional cross section of a microchannel. A binary symmetric electrolyte is flowing in the middle. Anode and cathode are not placed inside the fluid. They represent the external electric field. Notice that the walls of the microchannel are charged negatively. A positively charged electric double layer is therefore emerging. Electroneutrality dominates the bulk fluid. The positively charged part of the electric double layer is the component mainly effected by the external electric field. Therefore, this component transfers the motion, resulting from the electric force, to the rest of the fluid. Consequently, the fluid flows in the direction of the cathode.

2.2 Electric double layer

As already discussed in the previous section, an electric double layer builds up, when a plane with a charged surface is inserted into a dilute solution of a dissolved electrolyte. Note that a double layer can also build up, when a relative big particle is inserted into the dilute solution [2, Figure 9.2]. In classical theory [2, p. 108], enough ions exist around the plane, so that the potential far away from it is zero.

For most applications, this is not a restriction. Furthermore, there is no external fluid convection present in the emergence of the electric double layer. The only fluid motion present is attributed

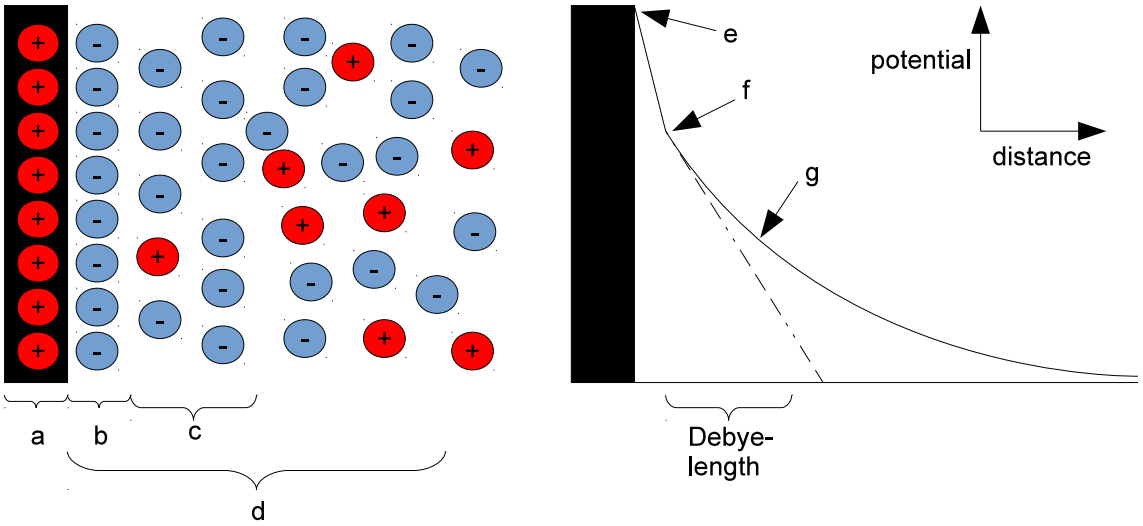


Figure 2: Based on figure 5.28 in [2, p. 170]. Legend: a: charged surface, b: Stern layer, c: shear plane, d: diffusive layer, e: surface potential ψ_S, f: Stern potential ψ_d, g: zeta potential ζ

to the Brownian motion and the electrostatic forces. Differently signed charges repel each other, whereas equally signed charges attract each other.

Without the charged plane, there is no preferred direction of motion. Brownian motion and electrostatic forces negotiate a homogeneous equilibrium. Within this equilibrium there exists on average no space charge and therefore no potential difference. The potential in this area, further referred to as bulk fluid, is set to zero.

The inserted plane distorts the charge homogeneity in the fluid around itself. There are now immobile charges at a fixed position which exert a preferred direction of motion on the remaining ions. Electrostatic attraction and repulsion apply as described earlier. Technically, the ions screen the charge singularity from the rest of the fluid. Far away from the plane, in the bulk fluid, a state of homogeneity is reached again. This non-uniform ionic distribution normal to the plane is called the electric double layer.

The Stern model [2, p. 170] separates the potential distribution into several layers/planes (see figure 2). The first layer is directly attached to the charged surface. Moreover, it is called the Stern layer and is considered to be immobile due to the strong electrostatic forces. Helmholtz noticed this layer very early and called the resulting double layer consisting of the surface charge and the attached charges a molecular condenser [2, p. 108]. The shear plane is attached next to the Stern layer. This is the mobile layer which can be moved by an external force. Finally, the area between the bulk fluid and the shear plane is classified as diffusive layer.

Following the Stern model, each layer is accompanied by a corresponding potential (see figure 2). The surface potential is followed by the Stern potential, which is located at the immobile layer. At the shear plane, the zeta potential is defined. Since all these layers are very close to each other, the zeta potential is often identified with the surface potential in literature which is mainly done because of simplicity [4, p. 515].

The main difference between the Stern model and the Gouy-Chapman model is that the latter assumes ions to be point charges. Nevertheless, it is a good approximation, if the surface potential is low (~ 0.025 V) [2, p. 109]. Besides that, the Poisson-Nernst-Planck equation (see section 3) is build upon the continuity assumption of the concentration of the different ion species. This means, that the size of the ions is also neglected here.

Considering the same situation as in figure 2, the potential ψ at distance x away from the surface, is related to the surface potential ψ_S according to Gouy-Chapman [2, p. 113] by the following equation:

$$\psi = \frac{2k_BT}{ze} \ln\left[\frac{1+\exp(-\kappa x)\tanh\left(\frac{ze\psi_S}{4k_BT}\right)}{1-\exp(-\kappa x)\tanh\left(\frac{ze\psi_S}{4k_BT}\right)}\right] \tag{1}$$

The equation also contains the absolute temperature T and the valence z of the symmetric electrolyte. Apart from the well known constants k_B and e, the Debye length κ^{-1} can be calculated [2, p. 114].

$$\kappa^{-1} = \left[\frac{\epsilon k_B T}{2e^2 z^2 n_\infty} \right]^{\frac{1}{2}} \tag{2}$$

The Debye length is the distance from the surface, where the potential ψ decayed to one third of the surface potential ψ_S [2, p. 114]. Consequently, it is often used to quantify the thickness of the electric double layer. The dielectric permittivity of the solvent ϵ and the bulk concentration n_∞ are necessary for the calculation of the Debye length.

3 Theory of the Poisson-Nernst-Planck equation system

The Poisson and the Nernst-Planck equations together form a nonlinear coupled equation system. Whereas the theory of the Poisson equation is mainly rooted in electrostatics, the Nernst-Planck equation is an incarnation of a conservation law, as will be shown later. The derivation of the Nernst-Planck equation is only valid for dilute solutions [2, p. 179-191].

3.1 Poisson equation

$$-\Delta\psi = \frac{\rho}{\epsilon} \tag{3}$$

$$-\Delta\psi = \frac{F}{\epsilon}\sum_i z_i c_i \tag{4}$$

The Poisson equation is directly taken from electrostatics [2, p. 232]. In short, the potential ψ is mainly determined by the charge density ρ, which is calculated with the molar concentrations c_i, the valence of the former, z_i, and the Faraday constant F. In case of a binary symmetric electrolyte, e.g. $NaCl$, which dissolves into Na^+ and Cl^-, the equation can be simplified further.

$$-\Delta\psi = \frac{F(c^+ - c^-)}{\epsilon} \tag{5}$$

3.2 Nernst-Planck equation

The microchannel is infinitely long and the walls are isolated (see section 4). Therefore, the individual ion species must be conserved.

$$\frac{\partial c_i}{\partial t} = -\nabla \cdot j_i + R_i \tag{6}$$

The flux function j_i represents the mass flux of the i-th ion species. Moreover, the source term R_i models chemical or nuclear reactions producing new species, which is assumed to be absent.

$$R_i = 0 \tag{7}$$

The ion mass flux is made up of three parts: a convective part due to the flow of the fluid, a diffusive part due to Fick's law and a migrative part due to an external force.

$$j_i = j_{i,c} + j_{i,d} + j_{i,m} = \tag{8}$$

$$= c_i u - D_i \nabla c_i + c_i \omega_i F_{i,e} \tag{9}$$

Besides the fluid convection with velocity u, the migrative part also adds convection to the equation. The migrative part consists of an external force, whose importance is regulated by the mobility ω_i. Apart from convection, the diffusion is only influenced by the diffusion constant D_i. Furthermore, the external force is caused by the external electric field which is proportional to the negative gradient of the potential ψ.

$$F_{i,e} = -z_i e \nabla\psi \tag{10}$$

The Nernst-Planck equation is constituted by (6) and (9).

$$\frac{\partial c_i}{\partial t} = -\nabla \cdot (c_i u - D_i \nabla c_i - c_i \omega_i z_i e \nabla\psi) \tag{11}$$

A more concise form of the Nernst-Planck equation can be obtained by summarizing multiple constants into the ionic mobility (12).

$$\mu_i = z_i e \omega_i \tag{12}$$

$$\frac{\partial c_i}{\partial t} = -\nabla \cdot (c_i u - D_i \nabla c_i - c_i \mu_i \nabla\psi) \tag{13}$$

4 Microchannel model

Before the numerical analysis can take place in sections 5 and 6, the microchannel geometry (see figure 3) and the boundary conditions for the model problem are described. Details about the Poisson-Nernst-Planck equation system have been treated in section 3.

4.1 Domain decomposition

As the geometry of the channel could be chosen freely, a cuboid was taken. This leads to a simple cartesian computational space (see figure 3c), where the unknowns reside in the center of cubic cells. Since the cuboid complies with the cartesian grid, there is no need for coordinate transformations or curvilinear grids. The spacing of a cartesian grid is equidistant:

$$\Delta x = \Delta y = \Delta z = h \tag{14}$$

4.2 Initial conditions

The potential distribution can be calculated by the Poisson equation (4) from the ion concentrations. Furthermore, the Nernst-Planck equation (13) only advances those concentrations. Consequently, only the distributions of the different ion species must be prescribed as initial conditions. Moreover, the concentrations of the different species must be proportional to their valence. As a consequence, symmetric undissolved electrolytes comply with electro-neutrality at the beginning of the simulation.

4.3 Boundary conditions

In this thesis, a capacitor (see figure 3a) and a microchannel geometry (see figure 3b) are used. The walls are marked with bold lines. The capacitor and microchannel geometries are considered infinitely long. This reduces the problem to a two-dimensional problem. Thus, a solution should not differ across the longitude of the channel.

4.3.1 Potential

The surface potential ψ_S is specified at the channel walls. It is a Dirichlet boundary condition. Furthermore, the channel is assumed to be infinitely long. Consequently, periodic boundary conditions are used at the longitudinal end of the computational domain.

4.3.2 Ion concentration

Since the channel is infinitely long, the same reasoning like above is applied. So, periodic boundary conditions are used in the cross section of the microchannel. Since the channel walls are insulating, charged particles cannot penetrate them. Technically, this is achieved by setting the ion mass flux in the direction of the boundary to zero. As a convention, the normal on the yz-plane is labeled n_x.

$$n_x \cdot j_i = 0 \tag{15}$$

$$j_{i,x} = 0 \tag{16}$$

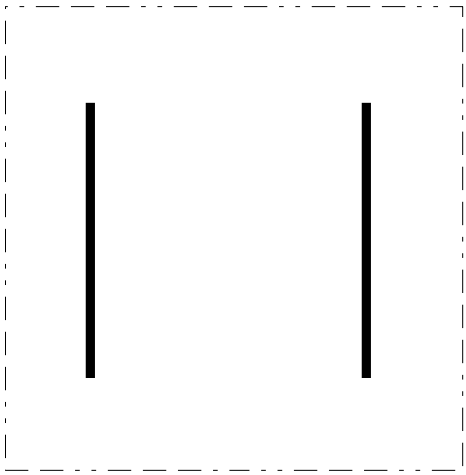
$$j_{i,x} = c_i \left(u_{i,x} - \mu_i \frac{\partial \psi}{\partial x} \right) - D_i \frac{\partial c_i}{\partial x} \tag{17}$$

$$D_i \frac{\partial c_i}{\partial x} = c_i \left(u_{i,x} - \mu_i \frac{\partial \psi}{\partial x} \right) \tag{18}$$

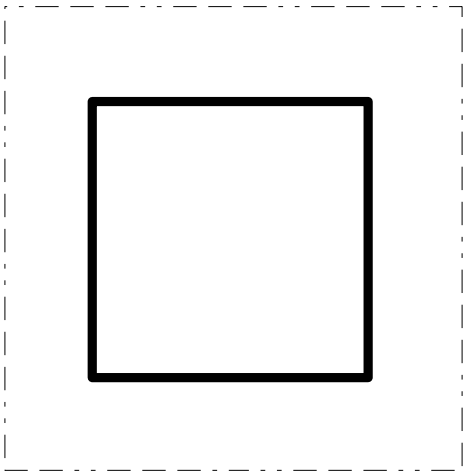
The velocity $u_{i,x}$ of the fluid in the direction of the wall at the wall is zero.

$$D_i \frac{\partial c_i}{\partial x} = -c_i \mu_i \frac{\partial \psi}{\partial x} \tag{19}$$

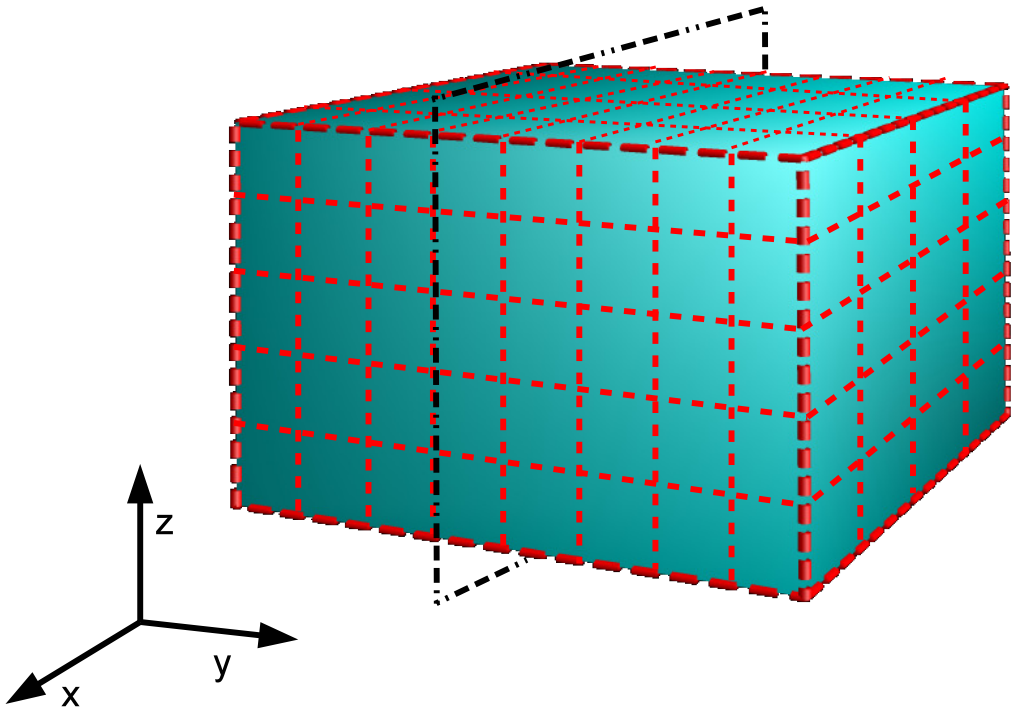
The derivation is also valid for the z-direction without loss of generality.



(a) capacitor profile



(b) microchannel profile



(c) computational domain

Figure 3: Map of the microchannel

$$D_i \frac{\partial c_i}{\partial z} = -c_i \mu_i \frac{\partial \psi}{\partial z} \tag{20}$$

This is a Robin boundary condition.

5 Numerical scheme for the Poisson equation

5.1 Finite difference discretization

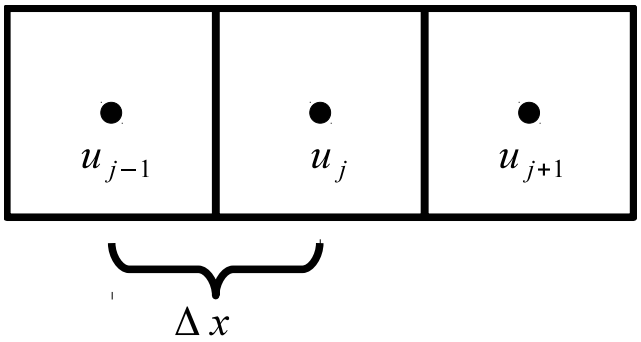


Figure 4: Example for the finite difference method

Finite differences are used to discretize the Poisson equation (4). It is a standard method for this type of partial differential equation [5]. A second-order approximation for the second derivative is given below. It can be obtained by using the Taylor approximation [6, p. 21ff.]. The situation is also illustrated in figure 4.

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_j = \frac{1}{\Delta x^2}(u_{j+1} - 2u_j + u_{j-1}) + O(\Delta x^2) \tag{21}$$

The Laplace operator consists of multiple second derivatives, one in each spatial dimension. Consequently, the Poisson equation (4) is discretized as shown below.

$$-\Delta\psi = -\frac{\partial\psi}{\partial x^2} + \frac{\partial\psi}{\partial y^2} + \frac{\partial\psi}{\partial z^2} = \frac{F}{\epsilon}\sum_i z_i c_i \tag{22}$$

$$\begin{aligned} -\Delta\psi(x,y,z) \approx & -\frac{1}{\Delta x^2}(\psi(x-1,y,z) - 2\psi(x,y,z) + \psi(x+1,y,z)) \\ & - \frac{1}{\Delta y^2}(\psi(x,y-1,z) - 2\psi(x,y,z) + \psi(x,y+1,z)) \\ & - \frac{1}{\Delta z^2}(\psi(x,y,z-1) - 2\psi(x,y,z) + \psi(x,y,z+1)) \\ & = \frac{F}{\epsilon}\sum_i z_i c_i(x,y,z) \end{aligned} \tag{23}$$

With the cartesian (14) spacing, it can be simplified further.

$$\begin{aligned} -\Delta\psi(x,y,z) \approx & -\frac{1}{h^2}[-6\psi(x,y,z) + \psi(x-1,y,z) + \psi(x+1,y,z) \\ & + \psi(x,y-1,z) + \psi(x,y+1,z) + \psi(x,y,z-1) + \psi(x,y,z+1)] \end{aligned} \tag{24}$$

Periodic and Dirichlet boundary conditions are used, which can just be set in the domain. For a map of the boundary conditions, look at section 4.

Besides the calculation of the Poisson equation, finite differences are also used for the first derivative of the potential in (19).

$$\left(\frac{\partial u}{\partial x}\right)_j = \frac{1}{2\Delta x}(u_{j+1} - u_{j-1}) + O(\Delta x^2) \tag{25}$$

5.2 Stencils

The stencil Ξ is a notation for a ξ-vector containing all coefficients of an equation working on a field. Usually, numerical solvers only need this field or computational array and the associated stencil. The stencil notation can include a scalar in front of the first squared bracket. This refers to multiplying the ξ-vector by this scalar.

$$\begin{aligned}\Xi = \Bigg[& \begin{pmatrix} \xi(x-1,y+1,z-1) & \xi(x,y+1,z-1) & \xi(x+1,y+1,z-1) \\ \xi(x-1,y,z-1) & \xi(x,y,z-1) & \xi(x+1,y,z-1) \\ \xi(x-1,y-1,z-1) & \xi(x,y-1,z-1) & \xi(x+1,y-1,z-1) \end{pmatrix} \\ & \begin{pmatrix} \xi(x-1,y+1,z) & \xi(x,y+1,z) & \xi(x+1,y+1,z) \\ \xi(x-1,y,z) & \xi(x,y,z) & \xi(x+1,y,z) \\ \xi(x-1,y-1,z) & \xi(x,y-1,z) & \xi(x+1,y-1,z) \end{pmatrix} \\ & \begin{pmatrix} \xi(x-1,y+1,z+1) & \xi(x,y+1,z+1) & \xi(x+1,y+1,z+1) \\ \xi(x-1,y,z+1) & \xi(x,y,z+1) & \xi(x+1,y,z+1) \\ \xi(x-1,y-1,z+1) & \xi(x,y-1,z+1) & \xi(x+1,y-1,z+1) \end{pmatrix} \Bigg] \end{aligned} \tag{26}$$

The coefficients of the potential field in the Poisson equation (24) can also be written as stencil.

$$\Xi_{Poisson} = \frac{1}{h^2} \left[\begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ -1 & 6 & -1 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right] \tag{27}$$

Equation (24) can now be rewritten stencil notation.

$$-\Delta\psi(x,y,z) \approx \Xi_{Poisson}(\psi(x,y,z)) \tag{28}$$

This leads to the discrete equation.

$$\Xi_{Poisson}(\psi) = \frac{F}{\epsilon} \sum_i z_i c_i \tag{29}$$

6 Numerical scheme for the Nernst-Planck equation

6.1 Finite volume discretization

The finite volume discretization is a standard technique and often found in literature [6, 7, 8, 9]. Finite volume methods come in naturally, when conservation laws have to be obeyed. Mathematically, a conservation law is written as follows [7, p. 609]:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \tag{30}$$

Here, u represents the quantity to be conserved and $f(u)$ is a, not necessarily linear, flux function. This is the scalar version of a conservation law. In contrast, the multi-dimensional version is written with the divergence operator. A short physical derivation of conservation laws can be found in [10]. As already shown in section 3, the Poisson-Nernst-Planck equation (13) can be written in conservation form, too. Henceforward, the subscript i, indicating the i-th ion species is omitted for brevity.

$$\frac{\partial c}{\partial t} + \nabla \cdot (cu - D\nabla c - c\mu\nabla\psi) = 0 \tag{31}$$

When one discretizes the conservation law, the continuous version is replaced by a discrete version [7, p. 609]. In principle, this discrete conservation law may also be fulfilled by a finite difference approximation, although this may not be the case by default. A finite difference approximation could also be non-conservative. In general, it is best, if a discretization is mimicking all important aspects of the corresponding continuous version. Instead of calculating an approximate value at a distinct point like finite differences do, finite volume discretizations approximate the *average* of the solution in a control volume. This control volume can be of arbitrary shape, which is another advantage over finite difference methods. In this thesis, cubic control volumes, as shown in figure 5, are used due to their simplicity.

In the Nernst-Planck equation (13), the unknown is the concentration of the particular ion species. In the following equation, the integration variables are marked with a dash and the average with a bar. The average values, the discretized unknowns, are assumed to reside in the cell center, which is correct up to second-order accuracy [6, p. 71].

$$\begin{aligned}
\bar{c}(x,y,z,t_n) &= \frac{1}{h^3} \iiint_V c(x',y',z',t_n)dV && (32)\\
&= \frac{1}{h^3} \int_{z-\frac{1}{2}}^{z+\frac{1}{2}} \int_{y-\frac{1}{2}}^{y+\frac{1}{2}} \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} c(x',y',z',t_n)dx'dy'dz' && (33)\\
&= c^n_{x,y,z} && (34)
\end{aligned}$$

To reduce the complexity of the notation, the dependencies of the variables on x, y, z and t are omitted. Furthermore, D and μ are constant. The Nernst-Planck equation (13) can be split in a convective and a diffusive part.

$$\frac{\partial c}{\partial t} + \underbrace{\nabla \cdot ((u - \mu\nabla\psi)c)}_{convection} + \underbrace{\nabla \cdot (-D\nabla c)}_{diffusion} = 0 \tag{35}$$

First, the semi-discrete form will be used and integrated in space. It will also be divided by the volume of the cube (h^3). This is necessary, since the average of c is used later.

$$\frac{1}{h^3} \iiint_V \frac{\partial c}{\partial t} \mathrm{d}V = \underbrace{\frac{1}{h^3} \iiint_V \left[-\nabla \cdot ((u - \mu\nabla\psi)c)\right] \mathrm{d}V}_{convection} + \underbrace{\frac{1}{h^3} \iiint_V \left[-\nabla \cdot (-D\nabla c)\right] \mathrm{d}V}_{diffusion} \tag{36}$$

Now the divergence theorem is applied.

$$\frac{1}{h^3} \iiint_V \frac{\partial c}{\partial t} dV = \underbrace{\frac{1}{h^3} \oiint_S c\left(\mu\nabla\psi - u\right) \mathrm{d}S}_{convection} + \underbrace{\frac{1}{h^3} \oiint_S D\nabla c \mathrm{d}S}_{diffusion} \tag{37}$$

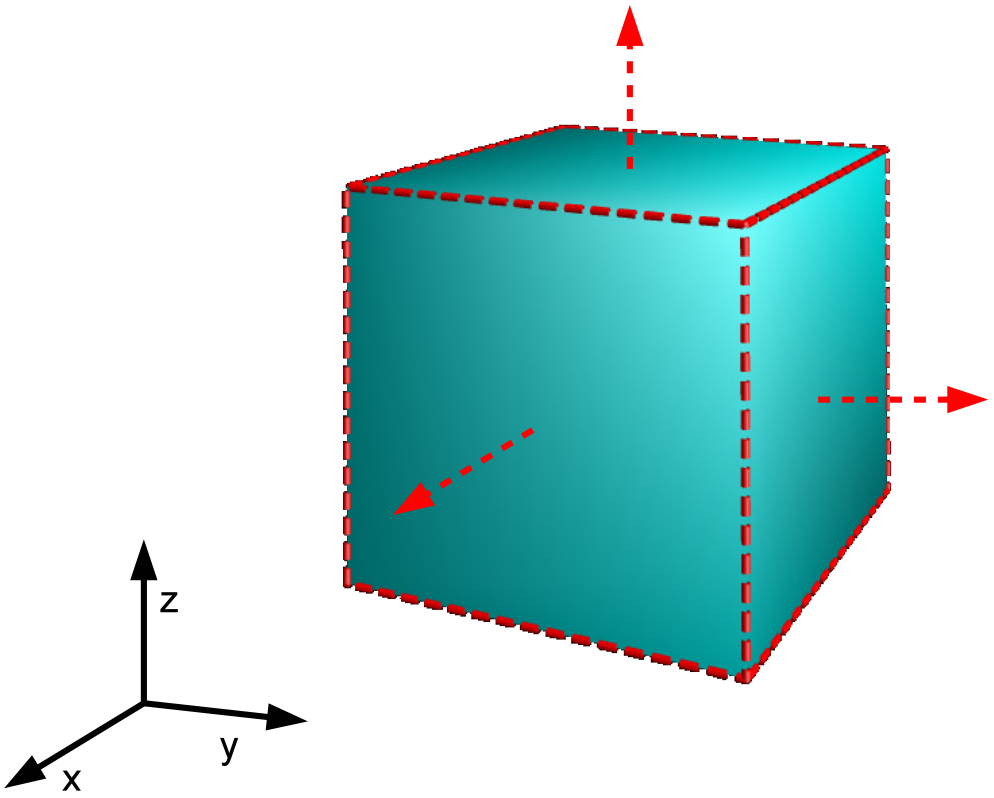


Figure 5: Control volume in the finite volume method. The red arrows indicate the normal surface vector.

Finally, the equation is integrated over time. Also, the integral and the derivative on the left hand side are exchanged. Here, c is assumed to be sufficiently smooth.

$$c_{x,y,z}^{n+1} - c_{x,y,z}^{n} = \underbrace{\frac{1}{h^3}\int_{t_n}^{t_{n+1}} \oiint_S c\left(\mu\nabla\psi - u\right) dSdt}_{convective\ flux} + \underbrace{\frac{1}{h^3}\int_{t_n}^{t_{n+1}} \oiint_S (D\nabla c) dSdt}_{diffusive\ flux} \tag{38}$$

In the next subsections, convective and diffusive fluxes will be formulated in a way, that computers actually may evaluate.

6.1.1 Convection

Convective equations must be discretized with care. To illustrate this, a simple conservation law (30) is used. In this case, the flux function $f(u) = c$ is inserted, where c is a constant value.

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x} \tag{39}$$

This equation may describe a wave traveling in one direction in a one-dimensional coordinate system. The solution of the equation can be written in terms of the initial condition [7, p. 549].

$$u(x,t) = u(x + ct, 0) \tag{40}$$

Assume now without loss of generality that $c < 0$. In that case, the wave travels from left $(-)$ to right $(+)$. In finite differences, this means one has to choose between forward, backward or centered space differences. Here, backward differences are the best choice. It is the only scheme that does not use the downwind value $u(x + h, t)$ which has no effect in the position x. The upwind value $u(x - h, t)$ plays a much greater role in determining the value $u(x, t + \Delta t)$ [7, p. 638]. Moreover, the discretization can be guaranteed to be stable when a CFL condition is enforced [7, p. 551]. Central

concept of upwinding

differences are the worst choice in that scenario since they are totally unstable [7, p. 550]. This example emphasizes the importance to choose the right discretization for the convection term.

The remaining part of this section will focus on transferring the concept of upwinding to finite volume methods. Firstly, the coefficient of the convective part in the Nernst-Planck equation (35) is not constant like in (39). It is even nonlinear. This makes the decision between backward or forward differences impracticable. Although it may be possible to switch between backward and forward differences locally, depending on the sign of the convection term, it was not chosen for this thesis.

In the theory of finite volume methods, there is a different trick that will do the job. Looking more closely, one can find *Riemann problems* at every cell interface. A Riemann problem [9, p. 19] [11, p. 783] is defined by a conservation law (30) and special initial values (41) at both sides of the interface.

Riemann problems

$$u^0 = \begin{cases} u_-, & x < 0 \\ u_+, & x \geq 0 \end{cases} \tag{41}$$

The scalar values u_- and u_+ are constant. This is exactly the case in this finite volume situation (see figure 3c, 5). Each cube has six interfaces and six Riemann problems have to be solved. Although this problem can be solved analytically [9, p. 37] [11, p. 788], it remains difficult, because an optimization problem has to be solved [9, p. 37]. To avoid that, the Lax-Friedrichs method is used. It can be shown that the Lax-Friedrichs method is an approximate Riemann solver [9, p. 41]. Moreover, a connection to the theory of finite differences can be drawn, where the Lax-Friedrichs method adds some artificial diffusion to the problem [11, p. 796]. In a nutshell, the Lax-Friedrichs method solves a *regularized* version of (30) with second order accuracy.

Lax-Friedrichs method

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = \frac{h^2}{2\Delta t} \frac{\partial^2 u}{\partial x^2} \tag{42}$$

Later, one can try to omit the artificial diffusion in the scheme, as the Nernst-Planck equation (35) already contains diffusion. Before showing the formula for the Lax-Friedrichs method, the flux form (46) for the finite volume method is presented for a scalar conservation law (30).

scalar flux form

$$\int_{x-\frac{1}{2}}^{x+\frac{1}{2}} \int_{t_n}^{t_{n+1}} \frac{\partial u}{\partial t} \mathrm{d}t\mathrm{d}x = -\int_{t_n}^{t_{n+1}} \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} \frac{\partial}{\partial x} f(u) \mathrm{d}x\mathrm{d}t \tag{43}$$

$$\int_{x-\frac{1}{2}}^{x+\frac{1}{2}} u^{n+1} - u^n \mathrm{d}x = -\int_{t_n}^{t_{n+1}} f_{x+\frac{1}{2}} - f_{x-\frac{1}{2}} \mathrm{d}t \tag{44}$$

$$\Delta x \left[u_x^{n+1} - u_x^n\right] = -\Delta t \left[\overline{f}_{x+\frac{1}{2}} - \overline{f}_{x-\frac{1}{2}}\right] \tag{45}$$

$$u_x^{n+1} = u_x^n - \frac{\Delta t}{\Delta x} \left[\overline{f}_{x+\frac{1}{2}} - \overline{f}_{x-\frac{1}{2}}\right] \tag{46}$$

This is called the flux form of the finite volume method. Here, the time average over f is marked with a bar. The discretization of the flux function $f(u)$ leads to a problem which is deeply connected to upwinding and the Riemann problem. The value u_x^n is different in each control volume. Therefore, the flux function is also discontinuous between the interface. When the Riemann problem is solved, the flux function is well-defined at the interface and can easily be integrated [9, p. 35ff.]. This is because the flux function is constant at the interface of a Riemann problem. Therefore, no time index will be attached to the flux function.

Multiple neighbouring Riemann problems may be solved independently, if they do not influence each other. This is enforced by a CFL condition [9, p. 36], eventually limiting the possible time step in relation to the grid size. The Lax-Friedrichs flux is defined as follows [7, p. 610] [9, p. 41].

Lax-Friedrichs flux

$$f^{LxF}(u_x^n, u_{x+1}^n) = \frac{1}{2}[f(u_x^n) + f(u_{x+1}^n)] - \frac{\Delta x}{2\Delta t}(u_{x+1}^n - u_x^n) \tag{47}$$

The flux can now be inserted into the flux form (46).

$$u_x^{n+1} = u_x^n - \frac{\Delta t}{\Delta x} \left[f^{LxF}(u_x^n, u_{x+1}^n) - f^{LxF}(u_{x-1}^n, u_x^n)\right] \tag{48}$$

Until now the flux form has only been derived for one dimension. Instead of dimensional splitting [12, p. 544-549], which essentially solves three one-dimensional problems in each dimension, an unsplit method is applied. In unsplit methods, all fluxes are accounted for in one step [12, p. 555-558]. For the explanation of the *unsplit method*, the flux function is separated into three parts. It should be mentioned that only cartesian domains are considered. The conservation law now contains the three flux functions: $f(u)$ for the flux in x-direction, $g(u)$ for the flux in y-direction and $h(u)$ for the flux in z-direction.

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) + \frac{\partial}{\partial y} g(u) + \frac{\partial}{\partial z} h(u) = 0 \tag{49}$$

multi-dimensional flux form

Again, the finite volume method can be written in flux form [12, p. 558].

$$\begin{aligned} u_{x,y,z}^{n+1} = u_{x,y,z}^{n} &+ \frac{\Delta t}{\Delta x}\left[f_{x-\frac{1}{2},y,z} - f_{x+\frac{1}{2},y,z}\right] \\ &+ \frac{\Delta t}{\Delta y}\left[g_{x,y-\frac{1}{2},z} - g_{x,y+\frac{1}{2},z}\right] + \frac{\Delta t}{\Delta z}\left[h_{x,y,z-\frac{1}{2}} - h_{x,y,z+\frac{1}{2}}\right] \end{aligned} \tag{50}$$

So, the fluxes in the preceding equation need to be defined.

$$f_{x+\frac{1}{2},y,z} = f(u_{x+\frac{1}{2},y,z}(0)) \tag{51}$$

Consequently, the function $u_{x+\frac{1}{2},y,z}(x,t)$ is the solution of the Riemann problem between two neighbouring control volumes:

$$\begin{aligned} &\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \\ &u^0(x) = \begin{cases} u_{x,y,z}^n & x < 0 \\ u_{x+1,y,z}^n & x > 0 \end{cases} \end{aligned} \tag{52}$$

The other fluxes can be derived analogously. Finally, everything is complete to derive the flux form for the convective part of the Poisson-Nernst-Planck equation (13, 38). The different parts of the flux function are noted with the coordinate direction raised.

$$f^x(c_i) = c_i\left(\mu_i \frac{\partial \psi}{\partial x} - u^x\right) \tag{53}$$

$$f^y(c_i) = c_i\left(\mu_i \frac{\partial \psi}{\partial y} - u^y\right) \tag{54}$$

$$f^z(c_i) = c_i\left(\mu_i \frac{\partial \psi}{\partial z} - u^z\right) \tag{55}$$

With (14), the flux form can be written like this:

$$c_{x,y,z}^{n+1} = c_{x,y,z}^{n} + \frac{\Delta t}{h}\left[f^x_{x-\frac{1}{2},y,z} - f^x_{x+\frac{1}{2},y,z} + f^y_{x,y-\frac{1}{2},z} - f^y_{x,y+\frac{1}{2},z} + f^z_{x,y,z-\frac{1}{2}} - f^z_{x,y,z+\frac{1}{2}}\right] \tag{56}$$

The calculation is done by applying Lax-Friedrichs fluxes (47). Additionally, the parts responsible for artificial diffusion are colored in red.

$$f^x_{x-\frac{1}{2},y,z} - f^x_{x+\frac{1}{2},y,z} = \frac{1}{2}\left(f^x(c^n_{x-1,y,z}) - f^x(c^n_{x+1,y,z})\right) + \frac{\Delta x}{2\Delta t}\left(c^n_{x-1,y,z} - 2c^n_{x,y,z} + c^n_{x+1,y,z}\right) \tag{57}$$

$$f^y_{x,y-\frac{1}{2},z} - f^y_{x,y+\frac{1}{2},z} = \frac{1}{2}\left(f^y(c^n_{x,y-1,z}) - f^y(c^n_{x,y+1,z})\right) + \frac{\Delta y}{2\Delta t}\left(c^n_{x,y-1,z} - 2c^n_{x,y,z} + c^n_{x,y+1,z}\right) \tag{58}$$

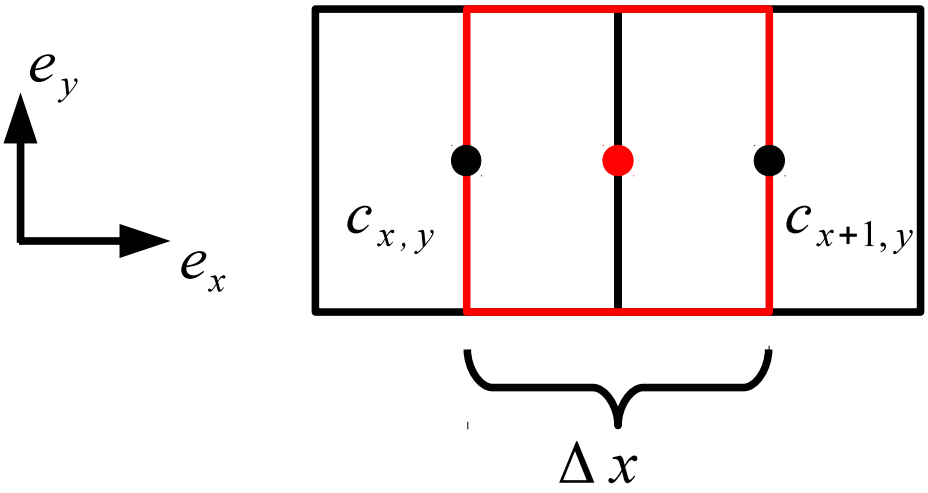


Figure 6: Gradient approximation with the divergence theorem.

$$f^z_{x,y,z-\frac{1}{2}} - f^z_{x,y,z+\frac{1}{2}} = \frac{1}{2}\left(f^z(c^n_{x,y,z-1}) - f^z(c^n_{x,y,z+1})\right) + \frac{\Delta z}{2\Delta t}\left(c^n_{x,y,z-1} - 2c^n_{x,y,z} + c^n_{x,y,z+1}\right) \tag{59}$$

$$\begin{aligned} c^{n+1}_{x,y,z} = c^n_{x,y,z} + \frac{\Delta t}{2h}\left[f^x(c^n_{x-1,y,z}) - f^x(c^n_{x+1,y,z}) + f^y(c^n_{x,y-1,z}) - f^y(c^n_{x,y+1,z})\right. \\ \left. + f^z(c^n_{x,y,z-1}) - f^z(c^n_{x,y,z+1})\right] + \frac{1}{2}\left[c^n_{x-1,y,z} + c^n_{x,y-1,z} + c^n_{x,y,z-1}\right. \\ \left. -6c^n_{x,y,z} + c^n_{x+1,y,z} + c^n_{x,y+1,z} + c^n_{x,y,z+1}\right] \end{aligned} \tag{60}$$

6.1.2 Diffusion

After the convective part of (38) has been treated, the diffusive term is next in line.

$$\begin{aligned} \frac{1}{h^3}\int_{t_n}^{t_{n+1}} \oiint_S D\nabla c dS dt = \frac{D}{h^3}\int_{t_n}^{t_{n+1}} \left[\iint_{S_{x+\frac{1}{2},y,z}} \frac{\partial c}{\partial x} dS - \iint_{S_{x-\frac{1}{2},y,z}} \frac{\partial c}{\partial x} dS\right. \\ \left. + \iint_{S_{x,y+\frac{1}{2},z}} \frac{\partial c}{\partial y} dS - \iint_{S_{x,y-\frac{1}{2},z}} \frac{\partial c}{\partial y} dS + \iint_{S_{x,y,z+\frac{1}{2}}} \frac{\partial c}{\partial z} dS - \iint_{S_{x,y,z-\frac{1}{2}}} \frac{\partial c}{\partial z} dS\right] dt \end{aligned} \tag{61}$$

When one wants to approximate the integral of the surface with the midpoint rule [13, p. 86], the value of the gradient at the mid-point between the centroids has to be known. The gradient can be calculated with first order central differences (25). Despite using the same approximation, it will be motivated differently.

The gradient will be derived with the divergence theorem, similar to the approach shown in [13, p. 273f.]. This approach is illustrated in two dimensions in figure 6. A control volume with the same size as the other control volumes is put around the point of interest. Then, the derivative in one direction is written as divergence.

$$\iiint_V \frac{\partial c}{\partial x} dV = \iiint_V \nabla \cdot (c e_x) dV \tag{62}$$

Furthermore, the divergence theorem is applied.

$$\iiint_V \frac{\partial c}{\partial x} dV = \oiint_S c e_x \cdot dS \tag{63}$$

The surface area of a cube in a cartesian grid (14) is numerically represented by h^2. This is included, when the surface integral is written as sum,

$$\iiint_V \frac{\partial c}{\partial x} dV = \sum_{k=1}^{6} c_k e_x \cdot h^2 e_k \tag{64}$$

where e_k represents the unit vector perpendicular to the k-th surface. The value of c in the middle of that surface is defined as c_k. Due to the orthogonality of e_x, e_y, e_z in a cartesian system, many terms vanish in the sum.

$$\iiint_V \frac{\partial c}{\partial x} dV = h^2 \left[c(x+1, y, z, t_n) - c(x, y, z, t_n) \right] \tag{65}$$

The first term of the equation can be approximated by taking the point in the middle as constant and multiplying it by the volume of the cube.

$$\iiint_V \frac{\partial c}{\partial x} dV \approx \Delta V \cdot \left(\frac{\partial c}{\partial x} \right)^n_{x+\frac{1}{2}, y, z} = h^3 \cdot \left(\frac{\partial c}{\partial x} \right)^n_{x+\frac{1}{2}, y, z} \tag{66}$$

Consequently, the last two equations can be combined.

$$\left(\frac{\partial c}{\partial x} \right)^n_{x+\frac{1}{2}, y, z} \approx \frac{c(x+1, y, z, t_n) - c(x, y, z, t_n)}{h} \tag{67}$$

Lastly, the discretized unknowns are inserted.

$$\left(\frac{\partial c}{\partial x} \right)^n_{x+\frac{1}{2}, y, z} = \frac{c^n_{x+1,y,z} - c^n_{x,y,z}}{h} \tag{68}$$

The derivatives are used together with the mid-point rule in the diffusive flux. From equations (61) and (68) follows:

$$\begin{aligned} \frac{1}{h^3} \int_{t_n}^{t_{n+1}} \oiint_S D \nabla c dS dt = \frac{D}{h^2} \int_{t_n}^{t_{n+1}} \big[& c^n_{x+1,y,z} - c^n_{x,y,z} - (c^n_{x,y,z} - c^n_{x-1,y,z}) \\ + c^n_{x,y+1,z} - c^n_{x,y,z} - (c^n_{x,y,z} - c^n_{x,y-1,z}) + c^n_{x,y,z+1} - c^n_{x,y,z} - (c^n_{x,y,z} - c^n_{x,y,z-1}) \big] dt = \\ \frac{D}{h^2} \int_{t_n}^{t_{n+1}} (c^n_{x+1,y,z} + c^n_{x,y+1,z} + c^n_{x,y,z+1} - 6c^n_{x,y,z} + c^n_{x-1,y,z} + c^n_{x,y-1,z} + c^n_{x,y,z-1}) dt \end{aligned} \tag{69}$$

This equation is very similar to the discretized version of the Poisson equation (see section 5.2). Indeed, the Poisson equation was also approximated with a second order approximation.

Finally, an Euler step is done to resolve the integration over time.

$$\frac{D \Delta t}{h^2} (c^n_{x+1,y,z} + c^n_{x,y+1,z} + c^n_{x,y,z+1} - 6c^n_{x,y,z} + c^n_{x-1,y,z} + c^n_{x,y-1,z} + c^n_{x,y,z-1}) \tag{70}$$

Alternatively to the derivation of the gradient via the divergence, multi-dimensional interpolation could have been used. Then, the gradient could be calculated analytically based upon this interpolation.

6.2 Stencils

The stencil notation has been introduced in section 5.2. It is also used for the discretization of the Nernst-Planck equation.

6.2.1 Convection

$$\Xi_{convection} = \left[\begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{\Delta t}{2h}\left(\mu_i \frac{\partial \psi}{\partial z}|_{x,y,z-1} - u^z_{x,y,z-1}\right) + \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{pmatrix} \right.$$
$$\begin{pmatrix} 0 & -\frac{\Delta t}{2h}\left(\mu_i \frac{\partial \psi}{\partial y}|_{x,y+1,z} - u^y_{x,y+1,z}\right) + \frac{1}{2} & 0 \\ \frac{\Delta t}{2h}\left(\mu_i \frac{\partial \psi}{\partial x}|_{x-1,y,z} - u^x_{x-1,y,z}\right) + \frac{1}{2} & 1-3 & -\frac{\Delta t}{2h}\left(\mu_i \frac{\partial \psi}{\partial x}|_{x+1,y,z} - u^x_{x+1,y,z}\right) + \frac{1}{2} \\ 0 & \frac{\Delta t}{2h}\left(\mu_i \frac{\partial \psi}{\partial y}|_{x,y-1,z} - u^y_{x,y-1,z}\right) + \frac{1}{2} & 0 \end{pmatrix}$$
$$\left. \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\frac{\Delta t}{2h}\left(\mu_i \frac{\partial \psi}{\partial z}|_{x,y,z+1} - u^z_{x,y,z+1}\right) + \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{pmatrix} \right] \tag{71}$$

6.2.2 Diffusion

$$\Xi_{diffusion} = \frac{D_i \Delta t}{h^2} \left[\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & -6 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right] \tag{72}$$

6.3 Boundary conditions

The boundary condition described in section 4.3 can be used to calculate the discretized unknown in the cell behind the boundary $\left(c^n_{0,y,z}\right)$. Again, (68) is used.

$$\left(D\frac{\partial c}{\partial x}\right)^n_{\frac{1}{2},y,z} = -\frac{1}{2} c^n_{\frac{1}{2},y,z} \mu \frac{\psi^n_{1,y,z} - \psi^n_{0,y,z}}{h} \tag{73}$$

$$D\left(\frac{c^n_{1,y,z} - c^n_{0,y,z}}{h}\right) = -\frac{1}{4}\left(c^n_{0,y,z} + c^n_{1,y,z}\right)\mu \frac{\psi^n_{1,y,z} - \psi^n_{0,y,z}}{h} \tag{74}$$

$$c^n_{0,y,z} = \frac{-4D - \mu\left(\psi^n_{1,y,z} - \psi^n_{0,y,z}\right)}{-4D + \mu\left(\psi^n_{1,y,z} - \psi^n_{0,y,z}\right)} c^n_{1,y,z} \tag{75}$$

7 Implementation in waLBerla in 3D

7.1 waLBerla

The numerical schemes are implemented in waLBerla, which is the abbreviation for *widely applicable lattice Boltzmann algorithm from Erlangen*. The software was developed at the chair of system simulation (LSS) at the Friedrich-Alexander-Universität Erlangen-Nürnberg. Furthermore, waLBerla was motivated by fluid simulation and the application of the lattice Boltzmann algorithm [14]. The package can also be used to simulate various problems on structured grids. A paper by C. Feichtinger et al. [15] gives insight into the general design of waLBerla. The most important concepts of waLBerla are blocks and sweeps [16]. Blocks are the result of the cartesian domain decomposition. Basically, the data is stored and distributed over the computational nodes. On the other hand, sweeps are functions that are executed in every time step. The process is taken care of by the waLBerla core. As a user, one has to write the sweep functions and tell the core when they have to be executed. Similarly, one has to create the fields. Due to its structured grid capabilities and wide use at the LSS, the waLBerla was chosen for this thesis. The waLBerla also comes with a working LSE (linear system of equations) solver module, which can be used to solve the Poisson equation [16]. Due to the fact that the three-dimensional algorithm did not converge to the right solution, the switch was made to a one-dimensional simulation for debugging purposes. Consequently, this section is rather short to only give a brief overview of the methodology applied.

7.2 Input file

The waLBerla can be configured by input files. This makes it easy to switch parameters. In this simulation, mostly SI units were used for the parameters. An input file module for electrokinetic simulations already existed, because my advisor, Dominik Bartuschat, was preparing a research work on these types of simulations. Consequently, the input file reader only had to be extended. It is further noted, that input files are not case sensitive. The implementation of the input file reader can be found in *walberla::electrokin_flow::EkfData::configure()*.

The following parameters are already implemented: zeta potential, temperature, k_B, e, N_A, dielectric constant (here equivalent to the solvent permittivity ϵ), ion concentration and electric field components.

For the solution of the Poisson-Nernst-Planck equation system (4, 13), the following parameters are needed: the Faraday constant F, the permittivity of the solvent ϵ, the diffusivity D_i, the zeta potential, the ion concentration and the ionic mobility μ_i.

For the last parameter, the former equation (12) is applied. Nonetheless, the mobility ω_i is unknown and must be calculated with the Stokes-Einstein relationship [2, p. 213].

$$\omega_i = \frac{D_i}{k_B T} \tag{76}$$

Thus, the ionic mobility can be reformulated.

$$\mu_i = \frac{z_i e D_i}{k_B T} \tag{77}$$

Consequently, only routines for reading D_i from the configuration file have to be written (see listing 1). Getter methods are not included in the listing.

Listing 1: Read additional parameters from input file (walberla::electrokin_flow::EkfData)

```
////////////////////////////////////
// Poisson-Nernst-Planck parameters //
////////////////////////////////////

if ( ekfParamsBlocks[0].isDefined( "posDiffusivity" ) ) {
    Convert(pc.CheckAndCompleteParam<real_t>("posDiffusivity",0,&PhysicalCheck::
        CheckNoCheck),posDiffusivity_);
} else {
    LOG_ERROR( "Missing diffusivity of positive ion species. posDiffusivity missing in
        ekfParams block" );
}
```

```
if ( ekfParamsBlocks[0].isDefined( "negDiffusivity" ) ) {
    Convert(pc.CheckAndCompleteParam<real_t>("negDiffusivity",0,&PhysicalCheck::
        CheckNoCheck),negDiffusivity_);
} else {
    LOG_ERROR( "Missing diffusivity of negative ion species. negDiffusivity missing in
         ekfParams block" );
}
```

A short part of the input file is printed here as example:

Listing 2: example configuration

```
SolvParams{

    // General solver parameters
    SOR;
    dump;

    MaxNumSolvIt 10000;
    ResidThresh 1e-10;

    SORParams{
        omegaSOR 1.7;
    }

}

// ... further configuration blocks
```

In the input file, also the boundary conditions are prescribed for each boundary of the cartesian grid. The LSE solver module supports Dirichlet, periodic and Neumann boundary conditions out of the box. A code was added to setup the necessary communication parts to get periodic boundary conditions for the concentration fields. For the Robin boundary conditions, a complete new handler had to be written. It can also be activated in the input file.

Listing 3: Part of the implementation of the Robin BC

```
template <typename Sten>
inline void PNPE_Robin_StrFwdBC<Sten>::treatBCDir(const uint_t x, const uint_t y,
    const uint_t z, const stencil::Direction d)
{
    using namespace stencil;
    posConc->GET_SCALAR(x+cx[d],y+cy[d],z+cz[d]) = (-4*posDiffusivity-posMu*(pot->
        GET_SCALAR(x,y,z) -
                zetaPot)) / (-4*posDiffusivity+posMu*(pot->GET_SCALAR(x,y,z) -
                    zetaPot)) * posConc->GET_SCALAR(x,y,z);

    negConc->GET_SCALAR(x+cx[d],y+cy[d],z+cz[d]) = (-4*negDiffusivity-negMu*(pot->
        GET_SCALAR(x,y,z) -
                zetaPot)) / (-4*negDiffusivity+negMu*(pot->GET_SCALAR(x,y,z) -
                    zetaPot)) * negConc->GET_SCALAR(x,y,z);
}
```

7.3 Algorithm

At the beginning of a waLBerla app, special data structures have to be initialized, depending on the functionality needed. Then, new block fields are allocated and initialized. Here, the ion concentrations are set to the bulk fluid concentration. Finally, the time loop is started. Inside the time loop the main work is done. Notice, that the boundary conditions for the concentrations have to be set manually whereas the boundary conditions for the Poisson problem are managed by waLBerla alone.

Algorithm 1 Poisson-Nernst-Planck solver in waLBerla

```
SetupWaLBerla();
InitPotentialBlockField();
InitPotentialRHS();
InitPotentialGradient();
InitPositiveIonConcentration();
InitNegativeIonConcentration();
InitPositiveIonConcentrationStencil();
InitNegativeIonConcentrationStencil();
time=0;
while time ≤ timeEnd do
  CalculatePotentialGradient();
  SetupConvectivePartOfConcentrationStencils();
  AddArtificialDiffusionToConcentrationStencils();
  AddDiffusionToConcentrationStencil();
  ApplyIonConcentrationBoundaryConditions();
  ApplyIonConcentrationStencils();
  CalculatePotentialRHS();
  SolvePoissonPotential();
  time++;
end while
```

7.4 Sweeps

After the initialization, all the sweeps are registered to the waLBerla core, which will execute them in order of registration. A sweep registration is shown in listing 4.

Listing 4: How to insert a sweep

```
USE_SweepSection(electrokin_flow::getPNPEPotRHSSweepUID()) {
    USE_Sweep() {
        swUseFunction("getPNPEPotRHSSweepUID", electrokin_flow::sweep::
            PNPE_FV_setPotentialRHS, FSUIDSet::all(),HSUIDSet::all(), BSUIDSet::all()
            );
    }
}
```

A sweep is executed in a time loop and runs over all blocks. Therefore it has a special method signature. The implementation of the sweep that sets up the right hand side in the Poisson equation for the potential is shown in listing 5.

Listing 5: Potential RHS Sweep

```
void PNPE_FV_setPotentialRHS(bd::BlockID block) {
    ///////////////
    // Logging  //
    ///////////////
    LOG_PROGRESS_DETAIL("Computation of charge density for Poisson equation (PNPE)");

    ////////////////////////
    // Fetch Data Fields //
    ////////////////////////
    shared_ptr<Field<real_t,1> > rhs =
    block->getBD<Field<real_t,1> >(lse_solver::getScalPotRHSUID());
    shared_ptr<Field<real_t,1> > pos_conc =
    block->getBD<Field<real_t,1> >(getPosIonConcSolUID());
    shared_ptr<Field<real_t,1> > neg_conc =
    block->getBD<Field<real_t,1> >(getNegIonConcSolUID());
    const uint_t zSize = rhs->zSize();
    const uint_t ySize = rhs->ySize();
    const uint_t xSize = rhs->xSize();

    ////////////////////////////////
```

```
    // Fetch Data From Input File //
    ////////////////////////////////
    const EkfDataID &ekfData = gd::getObject<EkfData>(getEkfDataUID());

    ////////////////////////
    // Iterate over Data //
    ////////////////////////
    real_t prefactor = ekfData->GetElectrCharge() * ekfData->GetAvogadroNr() / ekfData
        ->GetDielectricConst();
    for(uint_t z=1; z<zSize-1; ++z)
        for(uint_t y=1; y<ySize-1; ++y)
            for(uint_t x=1; x<xSize-1; ++x)
                rhs->GET_SCALAR(x,y,z) = prefactor * (pos_conc->GET_SCALAR(x,y,z) -
                    neg_conc->GET_SCALAR(x,y,z));
}
```

7.5 Results

After running the code, it can be observed that the simulation is stuck right in a steady state right from the beginning. The potential block field has the value of the zeta potential in all cells. To understand the steady state, on has to look on the Nernst-Planck equation (13) and the Robin boundary conditions for this equation (19).

$$u = 0 \tag{78}$$

$$\nabla c = 0 \tag{79}$$

$$\nabla \psi = 0 \tag{80}$$

With the conditions listed above, the Nernst-Planck equation and the resulting Robin boundary condition default to the following equation:

$$\frac{\partial c_i}{\partial t} = 0 \tag{81}$$

So the simulation is locked into a steady state right from the beginning. This can be seen as having lost a boundary condition during mirroring. If the setup is not symmetric like in figure 3a or 3b, then it is obvious to specify a Dirichlet boundary condition of zero for the potential in the bulk fluid. After mirroring, there is no boundary, where the zeta potential is not set. A quick fix might include setting the potential to zero in the middle of the channel. Although this explanation looks easy now, it was not apparent back then. Therefore, an implementation in one dimension was programmed to test the hypothesis that the numerical scheme was erroneous.

8 Implementation in C++ in 1D

8.1 Domain setup and boundary conditions

This implementation was carried out to test whether the discretization as derived in section 5 and 6 works in one dimension.

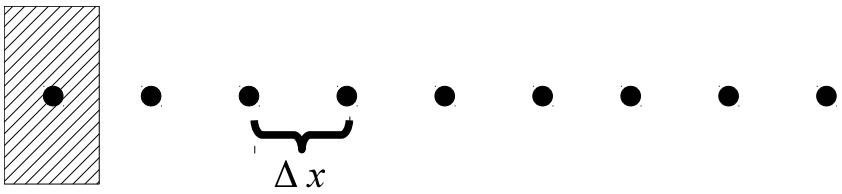


Figure 7: Half ion channel in 1D. Left: wall. Right: bulk fluid.

The mathematical structure of the boundary conditions stays mostly unchanged compared to 4.3. The following table lists all possible boundary conditions on the wall and bulk fluid side.

	wall	bulk fluid
ion concentration	Robin	Dirichlet
potential	Dirichlet	Dirichlet

In the bulk fluid Dirichlet boundary conditions are applied to the potential and the ion concentrations. The potential is supposed to vanish in the bulk whereas the ion concentrations are set to the bulk ion concentration.

8.2 Simulation parameters and derived quantities

Listing 6: Source file for grid class

```
---INFO SECTION---
Debye length:           9.2074e-07      in m
Electronic mobility:    4.25071e-08     in (m^2)/(Vs)
Zeta potential:         -0.025          in V
Bulk ion concentration: 0.0001          in mol/(m^3)
Solvent permittivity:   6.95393e-10     in C/(Vm)
Diffusivity:            1e-09           in (m^2)/s
Temperature:            273             in K
dt:                     1e-12           in s
dx:                     9.2074e-09      in m
Points:                 2000
---END---
```

8.3 Simplified equations and implementation

The Poisson equation is discretized (24) in one dimension as:

$$-\frac{1}{h^2}\left(\psi(x-1)-2\psi(x)+\psi(x+1)\right)=\frac{F}{\epsilon}\left(c^+(x)-c^-(x)\right) \tag{82}$$

The Nernst-Planck equation follows the derivation given in section 6.

$$c_x^{n+1}=c_x^n+\frac{\Delta t}{2h}\left(f^x(c_{x-1}^n)-f^x(c_{x+1}^n)\right)+\frac{1}{2}\left(c_{x-1}^n-2c_x^n+c_{x+1}^n\right)+\frac{D_i\Delta t}{h^2}\left(c_{x-1}^n-2c_x^n+c_{x+1}^n\right) \tag{83}$$

$$c_x^{n+1}=c_x^n+\frac{\Delta t}{2h}\left(f^x(c_{x-1}^n)-f^x(c_{x+1}^n)\right)+\left(\frac{1}{2}+\frac{D_i\Delta t}{h^2}\right)\left(c_{x-1}^n-2c_x^n+c_{x+1}^n\right) \tag{84}$$

$$f^x(c^n_{x-1}) = c^n_{x-1}\mu\frac{\partial\psi}{\partial x}\Big|^n_{x-1} \tag{85}$$

$$f^x(c^n_{x+1}) = c^n_{x+1}\mu\frac{\partial\psi}{\partial x}\Big|^n_{x+1} \tag{86}$$

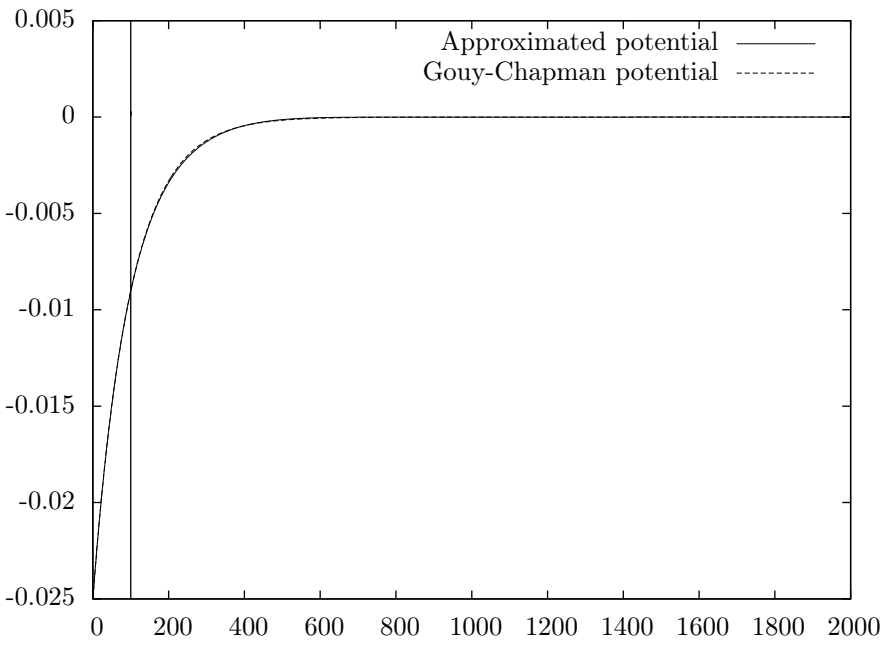
This can also be written in stencil form.

$$c^{n+1}_x = \left(\frac{D_i\Delta t}{h^2} + \frac{\Delta t}{2h}\mu\frac{\partial\psi}{\partial x}\Big|^n_{x-1} + \frac{1}{2}\right)c^n_{x-1} + \left(\frac{-2D_i\Delta t}{h^2}\right)c^n_x + \left(\frac{D_i\Delta t}{h^2} - \frac{\Delta t}{2h}\mu\frac{\partial\psi}{\partial x}\Big|^n_{x+1} + \frac{1}{2}\right)c^n_{x+1} \tag{87}$$

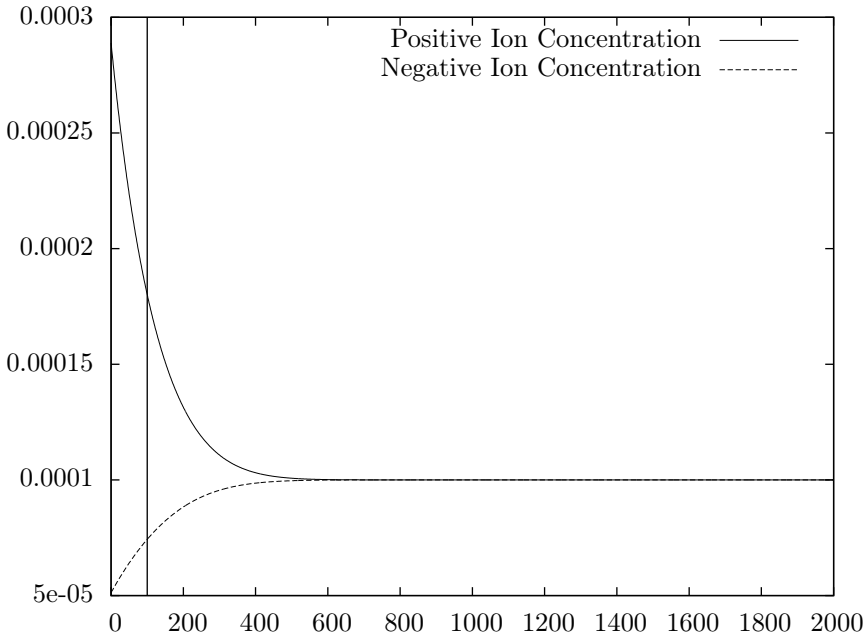
The implementation of the stencil and the whole algorithm in one dimension can be found in listing 10.

8.4 Results and Validation

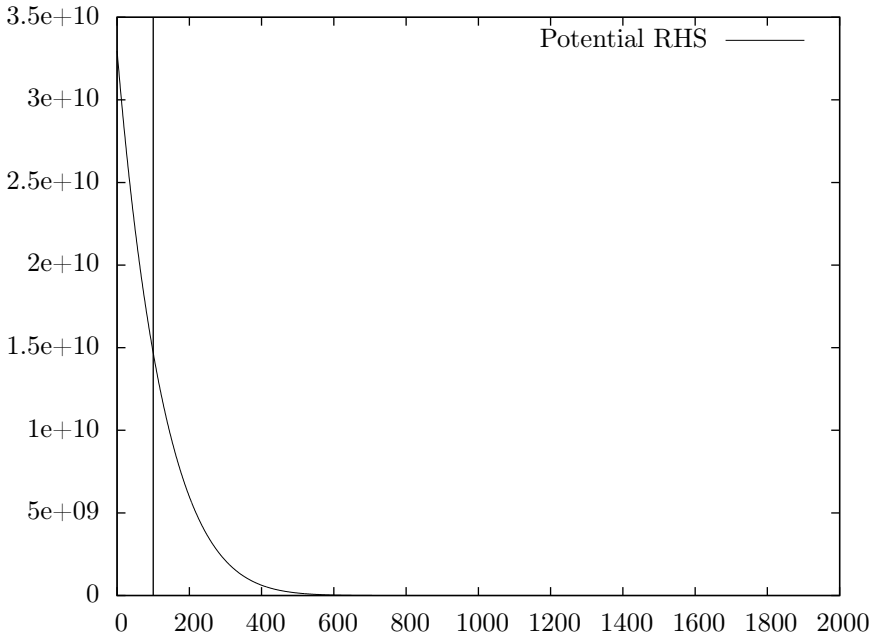
When the code stops the computation, when the calculated solution is near the solution given by the Gouy-Chapman approximation (1), which is very close to the actual solution for low surface potentials [2, p. 114], the results are reasonable. The distribution of the ion concentration (see figure 8b) looks qualitatively similar to results obtained by [2, p. 126] under similar conditions. This state is reached after 8.5 ns.



(a) Comparison between algorithmic potential and the potential obtained by the Gouy-Chapman approximation.



(b) Ion Concentrations obtained by the algorithm at Gouy-Chapman potential.



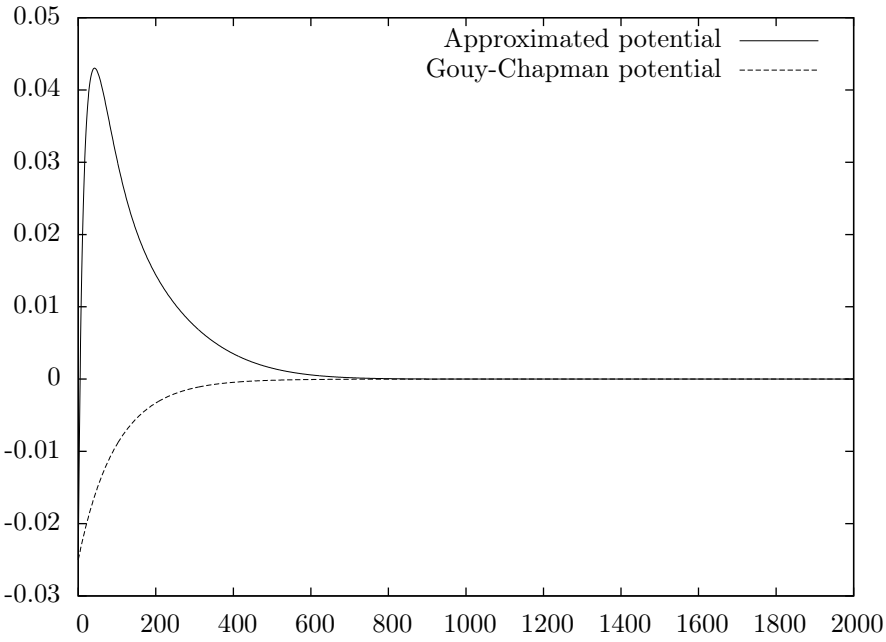
(c) Right hand side of the Poisson equation.

Figure 8: Solutions near Gouy-Chapman approximation. The Debye length is marked by a vertical line.

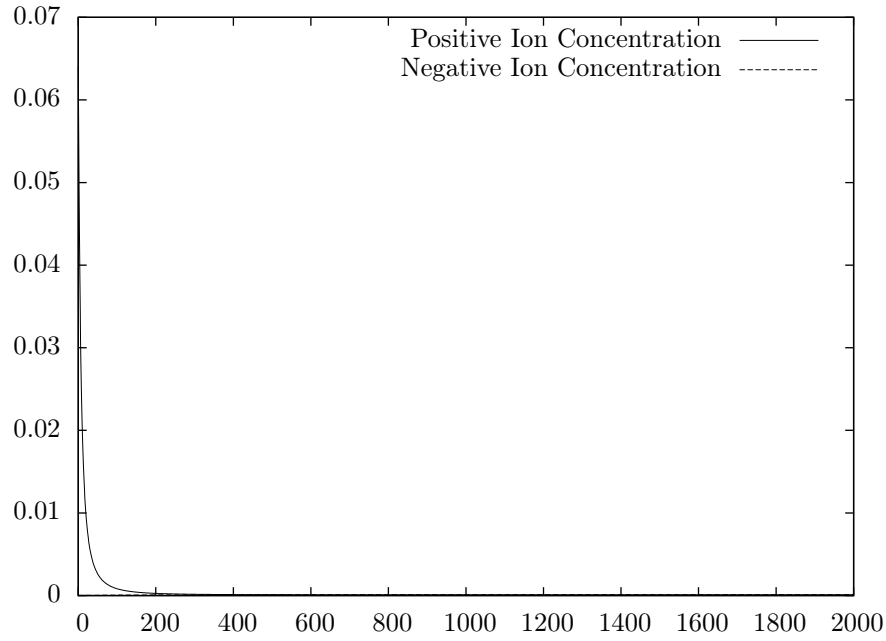
If the code does not stop near the Gouy-Chapman approximation, the calculated solution will diverge again. The calculated potential takes a different shape (see figure 9a) and the structure of the double layer is lost again. Presumably, the discretization derived in this work, does not reach a steady state that is physically correct. The calculation will slow down when one gets to such high potentials. Then it may happen that the CFL condition for the Lax-Friedrichs part of the discretization is no longer valid. Supposedly, the solver does not work, because it is not conserving the ion concentrations very well (see listing 7). This might explain why the solver does not reach a steady state.

Listing 7: Positive and negative ion concentration before and after one Nernst-Planck step

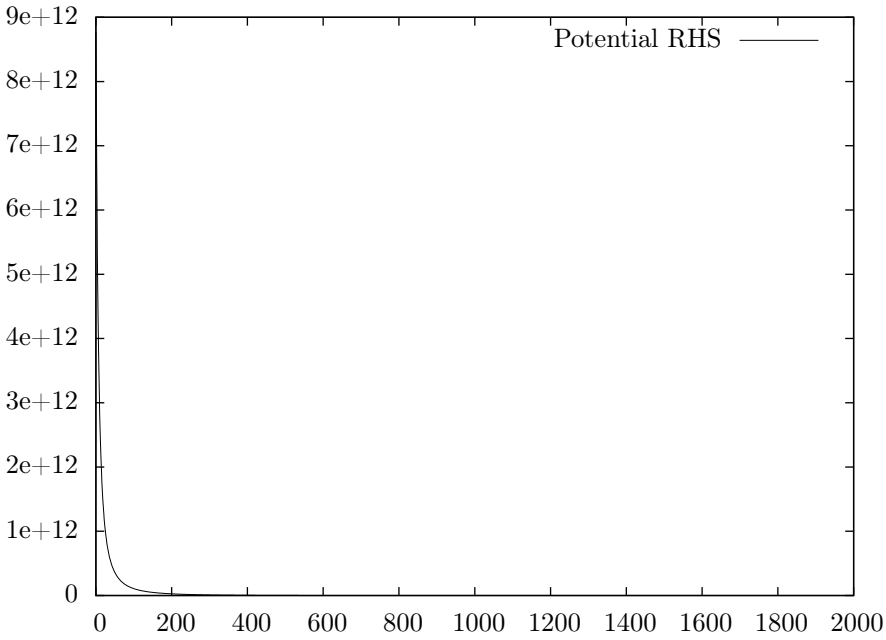
```
Time: 0 sumPosBefore: 0.199799999999994 sumPosAfter: 0.199800613019009 sumNegBefore: 0
    .199799999999994 sumNegAfter: 0.199799394405276 LInf(AnalyticSolution - Potential
    ) 0.0113902452878634

Time: 3.9326e-08 sumPosBefore: 0.220475506845978 sumPosAfter: 0.220476298188108
    sumNegBefore: 0.193255552946493 sumNegAfter: 0.193255411745256 LInf(
    AnalyticSolution - Potential) 9.99591249241684e-05
```



(a) Comparison between algorithmic potential and the potential obtained by the Gouy-Chapman approximation.



(b) Ion Concentrations obtained by the algorithm at Gouy-Chapman potential. The influence of the negative ions is negligible.



(c) Right hand side of the Poisson equation.

Figure 9: Solutions far away from the Gouy-Chapman approximation.

9 Conclusion

A new approach to the transient Poisson-Nernst-Planck equation system was proposed in this thesis. Consequently, a new discretization was developed. For the discretization of the Nernst-Planck equation, the finite volume method was chosen for its good conservation properties.

The challenging part of a convection-diffusion equation lies in the convective part, which is often responsible for instabilities. An approximate Riemann solver, namely the Lax-Friedrichs method, was chosen to oppose these difficulties.

Then, the method was extend for three dimensions with the help of waLBerla. Nevertheless, the discretization has only been tested in one dimension. The extension to three dimensions seems to be viable as the problem concerning steady state initial conditions can potentially be fixed.

The most intriguing part of this thesis is that there is a point in time, where the computed potential and the Gouy-Chapman approximation are nearly equivalent. Then the calculated ion distributions are also of physical importance. So, it might be a good idea to use the results of this solver as a good guess, if no other methods are available.

However, the solver does not reach a physically valid steady state. Thus, it cannot be used in multi physics simulations without care.

Future work could focus on improving the algorithm derived in this thesis. Of course, it would be interesting, why the algorithm does not converge to a physically valid steady state. It also would be interesting, to test the influence of the fluid convection on the algorithm.

10 Acknowledgment

I like to thank my advisor Dominik Bartuschat for introducing me to this interesting topic of electrofluidics. He is an adept in all questions concerning waLBerla. Without him, I would not have understood waLBerla so quickly. Furthermore, he was also a very competent discussion partner for questions that came up during the numerical solution. Thank you, Dominik, for your support.

11 Nomenclature

Δx	grid spacing in x-direction in m
Δy	grid spacing in y-direction in m
Δz	grid spacing in z-direction in m
ϵ	dielectric constant of the solvent in $\frac{C}{Vm}$
κ^{-1}	Debye length in m
μ	ionic mobility of an ion species in $\frac{C\frac{m}{s}}{N}$
μ_i	ionic mobility of the i-th particle in $\frac{C\frac{m}{s}}{N}$
ω_i	mobility of the i-th particle in $\frac{\frac{m}{s}}{N}$
ψ	potential in V
ψ_d	Stern potential in V
ψ_S	surface potential in V
ρ	charge density in $\frac{C}{m^3}$
ζ	zeta potential in V
c	concentration of an ion species $\frac{mol}{m^3}$
c^+	concentration of the positive ions in $\frac{mol}{m^3}$
c^-	concentration of the negative ions in $\frac{mol}{m^3}$
c_i	concentration of the i-th ion species in $\frac{mol}{m^3}$
D	diffusivity in $\frac{m^2}{s}$
D_i	diffusivity of the i-th ion species in $\frac{m^2}{s}$
e	elementary charge $e = 1.6022 \cdot 10^{-19} C$
F	Faraday constant $= N_A e$
$F_{i,e}$	external force in N
h	equidistant grid spacing in m
j_i	mass flux of the i-th ion species in $\frac{mol}{m^2 s}$
$j_{i,d}$	diffusive part of the ion mass flux of the i-th ion species $\frac{mol}{m^2 s}$
$j_{i,c}$	convective part of the ion mass flux of the i-th ion species $\frac{mol}{m^2 s}$
$j_{i,m}$	migrative part of the ion mass flux of the i-th ion species $\frac{mol}{m^2 s}$
k_B	Boltzmann constant $k_B = 1.3807 \cdot 10^{-23} \frac{J}{K} = 8.617386 \cdot 10^{-5} \frac{eV}{K}$
n_∞	bulk concentration in $\frac{1}{m^3}$
R_i	rate of production of species i due to chemical reactions per unit volume $\frac{mol}{m^3 s}$
T	absolute temperature in K
t	time in s
u	velocity of the fluid in $\frac{m}{s}$
z	valence of a symmetric electrolyte
z_i	valence of the i-th ion species

12 References

[1] Ben Schwan. *DNA auf einem Chip*. Feb. 2013. URL: http://heise.de/-1793710 (visited on 02/04/2013).

[2] *Electrokinetic and Colloid Transport Phenomena*. John Wiley & Sons, 2006.

[3] Kannan Masilamani. "WaLBerla: Investigation of Electrostatic Effects in Particulate and Electro-Osmotic Flows". MA thesis. FAU, 2010.

[4] Dzmitry Hlushkou, Drona Kandhai, and Ulrich Tallarek. "Coupled lattice-Boltzmann and finite-difference simulation of electroosmosis in microfluidic channels". In: *International journal for numerical methods in fluids* 46.5 (2004), pp. 507–532.

[5] Christoph Pflaum. *Simulation und wissenschaftliches Rechnen (SiwiR I) 2011/2012*. URL: http://www10.informatik.uni-erlangen.de/Teaching/Courses/WS2011/SiWiR/material/script.pdf (visited on 02/20/2013).

[6] Havard Lomax, Thomas H. Pulliam, and David W. Zingg. *Fundamentals of computational fluid dynamics*. Springer Berlin, 2001.

[7] Gilbert Strang. *Wissenschaftliches Rechnen*. Springer-Verlag Berlin Heidelberg, 2010.

[8] Olof Runborg. *Lecture 3 - Finite Volume Discretization of the Heat Equation*. Feb. 2012. URL: http://www.csc.kth.se/utbildning/kth/kurser/DN2255/ndiff12/Lecture3.pdf (visited on 02/01/2012).

[9] Siddhartha Mishra. *Numerical methods for conservation laws*. 2013. URL: http://www.math.ethz.ch/education/bachelor/lectures/fs2013/math/nhdgl/numcl_notes_HOMEPAGE.pdf (visited on 06/09/2013).

[10] Olof Runborg. *Lecture 1 - Introduction to PDEs*. Jan. 2012. URL: http://www.csc.kth.se/utbildning/kth/kurser/DN2255/ndiff12/Lecture1updated.pdf (visited on 01/18/2012).

[11] M. Hanke-Bourgeois. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Vol. 3. Teubner, 2009.

[12] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer London, Limited, 2009. ISBN: 9783540498346. URL: http://books.google.de/books?id=SqEjX0um8o0C.

[13] J.H. Ferziger and M. Perić. *Numerische Strömungsmechanik*. Springer-Verlag Heidelberg Berlin, 2008.

[14] Kristina Pickl. *waLBerla*. Jan. 2012. URL: http://www10.informatik.uni-erlangen.de/Research/Projects/walberla/index.shtml (visited on 01/21/2012).

[15] C. Feichtinger et al. "WaLBerla: HPC software design for computational engineering simulations". In: *Journal of Computational Science* 2.2 (2011), pp. 105–112.

[16] D. Bartuschat, D. Ritter, and U. Rude. "Parallel multigrid for electrokinetic simulation in particle-fluid flows". In: *High Performance Computing and Simulation (HPCS), 2012 International Conference on*. 2012, pp. 374–380. DOI: 10.1109/HPCSim.2012.6266940.

A Grid class

Listing 8: Header file for grid class

```
#ifndef ____Grid__
#define ____Grid__

#include <iostream>
#include <iomanip>
#include <algorithm>

class Grid {
    double *domain;
    int X_;
public:
    Grid(unsigned int X);
    Grid& operator=(const Grid& g);
    Grid(const Grid &g);
    ~Grid();
    double& operator()(unsigned int x);
    const double& operator()(unsigned int x) const;
    double LInf();
    Grid& multiply(double factor);
    inline unsigned int getX() const {
        return X_;
    }
    const unsigned int getLx();
};

Grid operator-(const Grid& g1, const Grid& g2);
Grid operator+(const Grid& g1, const Grid& g2);
std::ostream& operator  <<(std::ostream& out, const Grid& g);
#endif /* defined(____Grid__) */
```

Listing 9: Source file for grid class

```
#include "Grid.h"

Grid::Grid(unsigned int X) : X_(X) {
    domain = new double[X];
    for (unsigned int i = 0; i < X; i++) {
        domain[i] = 0.0;
    }
}

Grid::~Grid() {
    delete[] domain;
}

Grid::Grid(const Grid &g) : X_(g.X_), domain(new double[g.X_]) {
    std::copy(g.domain, g.domain + g.X_, domain);
}

Grid& Grid::operator=(const Grid& g) {
    if (this != &g) {
        double *newdomain = new double[g.X_];
        std::copy(g.domain, g.domain + g.X_, newdomain);
        delete[] domain;
        domain = newdomain;
        X_ = g.X_;
    }
    return *this;
}

double& Grid::operator()(unsigned int x) {
    return domain[x];
}

const double& Grid::operator()(unsigned int x) const {
```

```
    return domain[x];
}

const unsigned int Grid::getLx() {
    return X_;
}


std::ostream& operator<<(std::ostream& out, const Grid& g){
    for (unsigned int x = 0; x < g.getX(); x++) {
        out << std::fixed << g(x) << " ";
    }
    out << std::endl;
    return out;
}

double Grid::LInf() {
    double result = 0.0;
    for (unsigned int x = 0; x < X_; x++) {
        double a = (*this)(x);
        a = a < 0 ? -a : a;
        result = std::max(result, a);
    }
    return result;
}

Grid& Grid::multiply(double factor) {
    for (unsigned int x = 0; x < X_; x++) {
        (*this)(x) *= factor;
    }
    return (*this);
}

Grid operator+(const Grid& g1, const Grid& g2) {
    unsigned int X1 = g1.getX();
    unsigned int X2 = g2.getX();

    if (X1 == X2) {
        Grid g(X1);
        for (unsigned int x = 0; x < g.getX(); x++) {
            g(x) = g1(x) + g2(x);
        }
        return g;
    }
    return g1;
}

Grid operator-(const Grid& g1, const Grid& g2) {
    unsigned int X1 = g1.getX();
    unsigned int X2 = g2.getX();

    if (X1 == X2) {
        Grid g(X1);
        for (unsigned int x = 0; x < g.getX(); x++) {
            g(x) = g1(x) - g2(x);
        }
        return g;
    }
    return g1;
}
```

B Poisson-Nernst-Planck solver in 1D

Listing 10: 1D simulation of the Poisson-Nernst-Planck equation

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <fstream>
#include "Grid.h"
using namespace std;

// fixed mathematical constants
const double pi = 3.14159265;

// fixed physical constants
const double elementaryCharge = 1.6022e-19; // in Coulomb
const double Avogadro = 6.0221e23; // in 1 / mol;
const double Faraday = elementaryCharge * Avogadro;
const double Boltzmann_with_e = 8.617386e-5; // ev / K

const unsigned int lx = 2000;

// variable simulation parameters
const double zetaPotential = -0.025; // in V
const double ionConcentration = 1.0e-4; // in mol
const double solventPermittivity = 78.54 * 8.854e-12; // in C^2 / Nm^2
const double diffusivity = 1.0e-9; //in m^2//
const double temperature = 273.0; // in K
const double dt = 1e-12; // in s

// derived parameters
const double e_mobility = diffusivity/(Boltzmann_with_e * temperature);
const double debye_length = sqrt( (solventPermittivity * Boltzmann_with_e *
    temperature) / (2 * elementaryCharge * ionConcentration * Avogadro) ); // in m
const double h = debye_length / 100;

// precision of the solution of the potential
const double epsilon_jacobi = 1.0e-5;

// symmetric channel profile (false is default)
bool channel = false;
// stop when calculated Potential is near analytic solution
// should be set false when channel == true
bool stopAnalytic = true;

double sumOverGrid(Grid &toSumOver) {
    double sum = 0.0;
    for (unsigned int i = 1; i < toSumOver.getLx()-1; i++) {
        sum += toSumOver(i);
    }
    return sum;
}

int main(int argc, char* argv[]) {

    ofstream simInfo("./simInfo.txt", ofstream::out);
    simInfo << "---INFO SECTION---" << endl;
    simInfo << "Debye length:\t\t" << debye_length << "\tin m" << endl;
    simInfo << "Electronic mobility:\t" << e_mobility << "\tin (m^2)/(Vs)" << endl;
    simInfo << "Zeta potential:\t\t" << zetaPotential << "\t\tin V" << endl;
    simInfo << "Bulk ion concentration:\t" << ionConcentration << "\t\tin mol/(m^3)"
        << endl;
    simInfo << "Solvent permittivity:\t" << solventPermittivity << "\tin C/(Vm)" <<
        endl;
    simInfo << "Diffusivity:\t\t" << diffusivity << "\t\tin (m^2)/s" << endl;
    simInfo << "Temperature:\t\t" << temperature << "\t\tin K" << endl;
    simInfo << "dt:\t\t\t" << dt << "\t\tin s" << endl;
    simInfo << "dx:\t\t\t" << h << "\tin m" << endl;
    simInfo << "Points:\t\t\t" << lx << endl;
    simInfo << "---END---" << endl;
```

```
simInfo.close();

ofstream fluxConservation("./fluxConservation.txt", ofstream::out);

// initialization of grids
Grid Potential(lx);
Grid Potential_RHS(lx);
Grid PosIonConcentration(lx);
Grid NegIonConcentration(lx);
Grid Potential_Sol(lx);
// set analytical solution
for (unsigned int i = 0; i < lx; i++) {
    double term1 = 1 + tanh(zetaPotential / (4.0 * Boltzmann_with_e * temperature)
        ) * exp(-1.0/debye_length * i * h);
    double term2 = 1 - tanh(zetaPotential / (4.0 * Boltzmann_with_e * temperature)
        ) * exp(-1.0/debye_length * i * h);
    Potential_Sol(i) = 2 * Boltzmann_with_e * temperature * log(term1/term2);
}
// set Dirichlet BC for Potential
Potential(0) = zetaPotential;
if (channel) {
    Potential(lx-1) = zetaPotential;
}
// set IC for concentrations
for (unsigned int i = 0; i < lx; i++) {
    PosIonConcentration(i) = ionConcentration;
    NegIonConcentration(i) = ionConcentration;
}

// constants
double posMu = e_mobility;
double negMu = -e_mobility;

// start simulation loop
for (unsigned int iterations = 0; true; iterations++) {

    // calculate Potential RHS
    Potential_RHS = (PosIonConcentration - NegIonConcentration).multiply(Faraday/
        solventPermittivity);

    // calculate Potential by jacobi relaxation
    Grid jacobi_temp(Potential);
    for (unsigned int jacobi_iterations = 0; true; jacobi_iterations++) {
        if (channel) {
            Potential(lx/2) = 0.0;
        }
        for (int x = 1; x < lx-1; x++) {
            jacobi_temp(x) = 1.0/2.0 * (h * h * Potential_RHS(x) + Potential(x + 1
                ) + Potential(x - 1));
        }
        Grid difference_between_iterations = jacobi_temp-Potential;
        swap(jacobi_temp,Potential);
        if (difference_between_iterations.LInf() < epsilon_jacobi) break;
    }

    // calculate GRAD Potential
    Grid gradPot(lx);
    for (int x = 1; x < lx-1; x++) {
        gradPot(x) = 1.0/(2*h) * (Potential(x+1) - Potential(x-1));
    }

    // Robin BC on left side
    PosIonConcentration(0) = (-4 * diffusivity - posMu * (Potential(1) -
        zetaPotential)) / (-4 * diffusivity + posMu * (Potential(1) -
        zetaPotential)) * PosIonConcentration(1);
    NegIonConcentration(0) = (-4 * diffusivity - negMu * (Potential(1) -
        zetaPotential)) / (-4 * diffusivity + negMu * (Potential(1) -
        zetaPotential)) * NegIonConcentration(1);
    // Robin BC on right side if channel, else Dirichlet BC
    if (channel) {
```

```
            PosIonConcentration(lx-1) = (-4 * diffusivity - posMu * (Potential(lx-2) -
                 zetaPotential)) / (-4 * diffusivity + posMu * (Potential(lx-2) -
                zetaPotential)) * PosIonConcentration(lx-2);
            NegIonConcentration(lx-1) = (-4 * diffusivity - negMu * (Potential(lx-2) -
                 zetaPotential)) / (-4 * diffusivity + negMu * (Potential(lx-2) -
                zetaPotential)) * NegIonConcentration(lx-2);
        } else {
            PosIonConcentration(lx-1) = ionConcentration;
            NegIonConcentration(lx-1) = ionConcentration;
        }

        double sumPosBefore = sumOverGrid(PosIonConcentration);
        double sumNegBefore = sumOverGrid(NegIonConcentration);

        // iterate over negative concentration
        Grid tempNeg(NegIonConcentration);
        for (int x = 1; x < lx-1; x++) {
            double W, M, E;
            W = diffusivity * dt/(h*h) + dt/(2*h) * -negMu * gradPot(x+1) + 0.5;
            M = -2 * diffusivity * dt/(h*h);//1 - 1;
            E = diffusivity * dt/(h*h) + dt/(2*h) * +negMu * gradPot(x-1) + 0.5;
            tempNeg(x) = E * NegIonConcentration(x-1) + M * NegIonConcentration(x) + W
                 * NegIonConcentration(x+1);
        }
        swap(tempNeg,NegIonConcentration);

        // iterate over positive concentration
        Grid tempPos(PosIonConcentration);
        for (int x = 1; x < lx-1; x++) {
            double W, M, E;
            W = diffusivity * dt/(h*h) + dt/(2*h) * -posMu * gradPot(x+1) + 0.5;
            M = - 2 * diffusivity * dt/(h*h);//1 - 1;
            E = diffusivity * dt/(h*h) + dt/(2*h) * +posMu * gradPot(x-1) + 0.5;
            tempPos(x) = E * PosIonConcentration(x-1) + M * PosIonConcentration(x) + W
                 * PosIonConcentration(x+1);
        }
        swap(tempPos,PosIonConcentration);

        double sumPosAfter = sumOverGrid(PosIonConcentration);
        double sumNegAfter = sumOverGrid(NegIonConcentration);
        double linf = (Potential_Sol - Potential).LInf();

        fluxConservation << "Time: " << std::setprecision(15) << dt * iterations << "
            sumPosBefore: " <<
        sumPosBefore << " sumPosAfter: " << sumPosAfter << " sumNegBefore: " <<
            sumNegBefore <<
        " sumNegAfter: " << sumNegAfter << " LInf(AnalyticSolution - Potential) " <<
            linf << endl;

        cout << "\rTime: " << std::fixed << std::setprecision(12) << dt * iterations
            <<
        "\tLInf(AnalyticSolution - Potential): " << linf;

        bool stopSimulation = linf < 1e-4 && stopAnalytic;
        if (iterations % 100 == 0 || stopSimulation) {
            ofstream results("./results.dat", ofstream::out);
            for (unsigned int i = 0; i < lx; i++) {
                results << i << " " << Potential(i) << " " << PosIonConcentration(i)
                    << " " << NegIonConcentration(i) << " " << Potential_RHS(i) << "
                    " << Potential_Sol(i) << endl;
            }
            results.close();
            ofstream time("./time.txt", ofstream::out);
            time << "Time: " << std::fixed << std::setprecision(12) << dt * iterations
                 << endl;
            time.close();
        }

        if (stopSimulation) break;
    }
    fluxConservation.close();
```

```
    cout << endl;
    return 0;
}
```
