12-14-2018

# Development of a Nanofluid Simulation Platform

Jabrane Nachit

# Development of a nanofluid simulation platform

By

Jabrane Nachit

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Aerospace Engineering

Mississippi State, Mississippi

December 2018

Development of a nanofluid simulation platform

By

Jabrane Nachit

Approved:

J. Mark Janus
(Major Professor)

David S. Thompson
(Graduate Coordinator/Committee Member)

Yu Lv
(Committee Member)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Jabrane Nachit

Date of Degree: December 14, 2018

Institution: Mississippi State University

Major Field: Aerospace engineering

Director of Thesis: Dr. J. Mark Janus

Title of Study: Development of a nanofluid simulation platform

Pages in Study 85

Candidate for Degree of Master of Science

Nano-fluids are colloidal solutions made up of particles of the nanometric scale suspended in a fluid. This type of solution has widespread great interest since the discovery of their particular properties. The Poisson-Nernst Planck system of equations (PNP) is one of the known models for the description of ion transport. This thesis aims to develop a method to solve the PNP equations in space and time for these nano-fluids. Additionally, a simulation platform (C++) is developed using an iterative scheme to solve the nonlinear equations resulting from the discretization of the system. After an overview of the literature on the subject, a discussion on the validity of the results obtained through the simulation platform through its comparison with literature and a commercial software package, COMSOL.

# ACKNOWLEDGMENT

This thesis would not have been possible without the precious support of some peoples and it is not easy to say the right words, the role they played. However, I would ask them to find here all my feelings of gratitude, coming from the heart, and accept my thanks.

My first thanks go to my thesis supervisor, Professor J. Mark Janus, who accompanied me throughout my training. His availability and his generous advice during some of my difficult moments have been of a very high quality, and of an immense utility; thank you very much Dr. Mark Janus. I also take this opportunity to thank all the members of Bagley College of Engineering.

I would like also to take this opportunity to thank all members of the Mississippi State University administration who made our stay enjoyable and I would especially like to mention Ms. Tamra Swann for her valuable assistance and availability.

I would also like to thank all the staff of the aeronautics pole of the International University of Rabat, in particular Mr. ElHachmi Essadiqi for his support and valuable advice, as well as all the professors who provided us with knowledge and guidance. My thoughts also go to the late Mrs. Jhane Khater, assistant in the UIR aerospace school, who was for our promotion more than a friend.

I would also like to thank Professor Mohammed Cherkaoui who, without him, this project could not have materialized. Special thanks also to Mr. Mfeddal Hilali, Director of Institute of Preparatory Schools of the International University of Rabat, who believed in me and who has always enlightened me with his counsel.

TABLE OF CONTENTS

# LIST OF FIGURES

# CHAPTER 1

## INTRODUCTION

In 1798, **Alessandro Volta** invented the first source of reliable and stable electricity. This was possible by successively stacking those elements following the order: copper, brine, zinc, brine, zinc… Stacking of zinc and copper plates separated by a tissue soaked with brine (saline solution) permitted the constitution of this source of electricity. The electrons provided by the electrochemical reaction between the zinc and the water supplied electrical systems. This source of electricity permitted many scientists to repeatedly study the electrical phenomena in a liquid or gaseous medium.

In 1808 **Ferdinand Friedrich Reuss**, a German researcher, became interested in the behavior of fluids subjected to an electric field and for the first time observed two major-electro kinetic phenomena [1]. In fact, he succeeded in circulating water through a clay agglomerate as a result of the application of a potential difference across the channel and thus described, for the first time, electro-osmosis. Therefore, electro-osmotic and electrophoretic phenomena were observed in the 1810s, based on a major technological innovation, the battery. In the following years, many scientific works were devoted to the understanding and the theoretical study of these electro-kinetic effects.

In the 1950s, **Pohl** studied the forces induced on dielectric particles by a non-uniform electric field. He therefore introduced the term of 'di-electrophorese', derived from the Greek word “phorein” which reflects the fact that a particle is transported according to its dielectric properties. Early on, **Pohl**'s research focused on industrial applications such as the separation of carbon black micro-particles from samples of polyvinyl chloride [2]. In the 1960s and 1970s, he concentrated his theoretical and experimental efforts on biological applications [3] such as the separation of cells or bacteria. The main constraint related to the experiences of di-electrophorese was creating an electric field's gradient strong enough to manipulate the micro-particles without reaching the surrounding dielectric breakdown voltage. Indeed, the voltages used during these experiments were on the order of several tens of kilovolts, which also required substantial material means. In 1978, **Pohl** published his work devoted to the study of di-electrophoresis [4], which is still a reference.

Since 1995, the rise of many works devoted to electro-osmosis and di-electrophoresis is mainly due to the use and the democratization within laboratories of the techniques and integration means resulting from the microelectronic revolution. Indeed, the systems miniaturization enable locally intense non-uniform electric fields by applying a few volts of potential. Research on di-electrophoresis addresses the theoretical aspects and focuses on specific applications such as biosensors, cell research, medical diagnosis, microfluidics, nano-assembly or particle filtration.

The current work falls within this framework of nanofluids studies and focuses on three major parts. First, a review of principle of modeling, discretization and numerical simulation, then a documentary study on nanofluids and electro kinetic phenomena, and finally, an implementation of a platform of simulations of nanofluids developed in C++ with comparison to a commercial code.

# CHAPTER II

# MODELING, DISCRETIZATION AND NUMERICAL SIMULATION

## 1. Modeling

### 1.1. What is a model?

The principle of a model is to replace a complex system by a simple object or operator reproducing the main aspects or behaviors of the original system (example: a reduced model, scale model, mathematical or numerical model, thought or reasoning model).

### 1.2. The reasons for modeling

In nature, the most interesting systems and physical phenomena are also the most complex to study. In fact, a large number of non-linear parameters interacting with each other (meteorology, fluid turbulence...) often governs these systems.

### 1.3. The different models

Some solution consists of using a series of experiments to analyze the parameters and magnitudes of the system. However, tests can be very expensive (e.g. flight-tests, tests with rare materials, very expensive instrumentation, etc.) and they can be dangerous (e.g. nuclear tests, space environment, etc.). Furthermore, it may be difficult to measure all the parameters: very small scale of the problem (e.g. life sciences, limit layer in fluid,etc.) or very large scale (e.g. astrophysics, meteorology, geophysics, etc.).

Fortunately, we can also construct mathematical models that allow the representation of the physical phenomenon. These models often use systems of nonlinear partial differential equations with, in general, no analytical solutions. It is then a question of numerically solving the problem by transforming the continuous equations of the physics into a discrete problem on a certain domain of computation (the mesh). In some cases, it is the only alternative (e.g. nuclear, astrophysical, etc.). In other cases, numerical simulations are carried out in parallel with experiments.

## 1.4. Modeling and numerical simulation

The different steps to model a complex system are:

- Search for a mathematical model representing the physical phenomenon,
- Equation layout,
- Development of a mesh, i.e. discretization of the equations,
- Resolution of discrete equations (linear systems to solve),
- Computer transcription and programming of discrete relationships,
- Numerical simulation and exploitation of results.

The engineer may intervene on one or more of these steps.

## 1.5. Concept of stability

There are three types of stability:

- The stability of a physical problem,
- The stability of a mathematical problem,
- The numerical stability of a calculation method.

### 1.5.1. Stability of physical problem: chaotic system

A problem is said to be chaotic if a small variation of the initial data leads to a very unpredictable variation of the results. This notion of chaos, linked to the physics of a problem, is independent of the mathematical model used and even more when it is question of the numerical method used to solve this mathematical problem. Many problems are chaotic, for example fluid turbulence.

### 1.5.2. Stability of a mathematical problem: sensitivity

A problem is very sensitive or poorly conditioned if a small variation of the data or parameters results in a large variation of the results. This notion of conditioning, linked to the mathematical problem, is independent of the numerical method used to solve it. To model a physical problem that is not chaotic, we build the best-conditioned mathematical model.

### 1.5.3. Stability of a numerical method

A method is unstable if it is subject to significant propagation of numerical discretization and rounding errors. A problem may be well conditioned while the numerical method chosen to solve it can be unstable. In this case, it is imperative to change the numerical method. On the other hand, if the original problem is ill-conditioned, no numerical method can remedy it. It will then be necessary to try to find a different mathematical formulation of the same problem, if we know that the underlying physical problem is stable.

## 2. Discretization of systems of partial differential equations

### 2.1. The three main categories of methods

To pass from the continuous exact problems governed by a system of partial differential equations to the discrete problem, there are three main categories of methods:

- The finite differences

The method consists in replacing the partial derivatives by divided differences or combinations of point values of the function in a finite number of discrete points or nodes of the mesh. The advantages of the methods are a great simplicity of writing and a low cost of calculation. While its disadvantages are the limitation to simple geometries and difficulties in taking into account Neumann boundary conditions.

- The finite volumes

The method integrates, on elementary volumes of simple form, the equations written in the form of conservation law. It thus provides conservative, discrete approximations in a natural way and is particularly well adapted to the equations of fluid mechanics. Its implementation is simple with rectangular elementary volumes. Advantages: allows treating complex geometries with volumes of any shape and more natural determination of Neumann boundary conditions. Disadvantage: few theoretical results of convergence.

- The finite elements.

The method consists in approximating, in a finite-dimensional subspace, a problem written in variational form (as a minimization of the energy in general) in a space of infinite dimension. The approximate solution is, in this case, a function determined by a finite number of parameters such as, for example, its values at certain points or nodes of the mesh. Advantages: possible processing of complex geometries, numerous theoretical results on convergence. Disadvantage: complexity of implementation and high cost in computing time and memory.

## 2.2. The finite differences

### 2.2.1. Principle - Precision order

The finite difference method consists in approximating the derivatives of the equations of physics by means of the Taylor series expansion and is deduced directly from the definition of the derivative. It is due to the work of several mathematicians of the 18th century (e.g. Euler, Taylor, Leibniz, etc.).

Given a function of space and time, u(x, y, z, t), by definition of the derivative, we have:

$$\frac{\partial u}{\partial x} = lim_{\Delta x \to 0} \frac{u\left(x + \Delta x, y, z, t\right) - u\left(x, y, z, t\right)}{\Delta x} \tag{1}$$

If Δx is small, a Taylor expansion of u (x + Δx, y, z, t) in the neighborhood of x gives:

$$u\left(x + \Delta x, y, z, t\right) = u(x, y, z, t) + \Delta x \frac{\partial u}{\partial x}\left(x, y, z, t\right) + \frac{\Delta x^2}{2} \frac{\partial^2 u}{\partial x^2}\left(x, y, z, t\right) + \frac{\Delta x^3}{6} \frac{\partial^3 u}{\partial x^3}\left(x, y, z, t\right) + \ldots$$

Truncating the series to the first order at Δx, we obtain:

$$\frac{u(x + \Delta x, y, z, t) - u(x, y, z, t)}{\Delta x} = \frac{\partial u}{\partial x}(x, y, z, t) + \mathcal{O}(\Delta x) \tag{2}$$

The approximation of the derivative is then of order 1 indicating that the truncation error tends to zero as $\mathcal{O}(\Delta x)$ the first power of Δx. By definition: the power of Δx with which the truncation error tends to $\frac{\partial u}{\partial x}\left(x\right)$ zero is called the order of the method.

### 2.2.2. Indices based notation - case 1D

Consider a mono-dimensional case where one wants to determine a quantity u(x) on the interval [0, 1]. The search for a discrete solution of the magnitude of the problem leads to the development of a mesh of the interval definition. We consider a mesh (or calculation grid) composed of N + 1 points $x_i$ for i = 0, ..., N regularly spaced with a step Δx. The points $x_i = i\ \Delta x$ are called the nodes of the mesh. The original continuous problem for the determination of a magnitude on an infinite set dimension is thus reduced to the search for N discrete values of this magnitude at the different nodes of the mesh.

Notation: we denote $u_i$ the discrete value of u(x) at the point $x_i$, i.e. $u_i = u(x_i)$. Similarly we note the derivative of u(x) at the node $x_i$ : $\left(\frac{\partial u}{\partial x}\right)_{x=x_i} = \left(\frac{\partial u}{\partial x}\right)_i = u_i'.$

This notation is used in an equivalent way for all derivatives of successive order of u.

The finite difference scheme of order 1 presented above is written in index notation as follow:

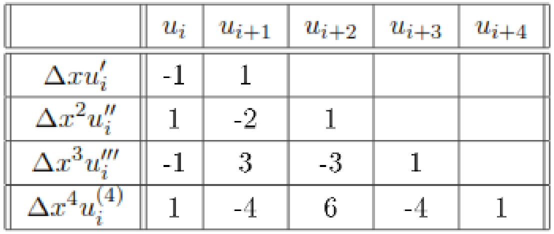
$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_{i+1} - u_i}{\Delta x} + \mathcal{O}(\Delta x) \tag{3}$$

This scheme is called "forward" or upwind. It is possible to construct another first-order scheme, called "backward":

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_i - u_{i-1}}{\Delta x} + \mathcal{O}(\Delta x) \tag{4}$$

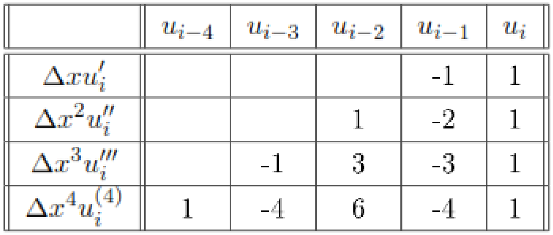
### 2.2.3. Some schemes in 1D

finite difference forward order 1



| | $u_i$ | $u_{i+1}$ | $u_{i+2}$ | $u_{i+3}$ | $u_{i+4}$ |
|---|---|---|---|---|---|
| $\Delta x u_i'$ | -1 | 1 | | | |
| $\Delta x^2 u_i''$ | 1 | -2 | 1 | | |
| $\Delta x^3 u_i'''$ | -1 | 3 | -3 | 1 | |
| $\Delta x^4 u_i^{(4)}$ | 1 | -4 | 6 | -4 | 1 |

finite difference backward order 1



| | $u_{i-4}$ | $u_{i-3}$ | $u_{i-2}$ | $u_{i-1}$ | $u_i$ |
|---|---|---|---|---|---|
| $\Delta x u_i'$ | | | | -1 | 1 |
| $\Delta x^2 u_i''$ | | | 1 | -2 | 1 |
| $\Delta x^3 u_i'''$ | | -1 | 3 | -3 | 1 |
| $\Delta x^4 u_i^{(4)}$ | 1 | -4 | 6 | -4 | 1 |

## 2.3. The finite volumes

### 2.3.1. Principle

The Finite Volumes method consists of integrating, on elementary volumes, the equations written in an integral form. This method is particularly well adapted to the spatial discretization of conservation laws and is thus widely used in fluid mechanics. Its implementation is simple if the elementary volumes or "control volumes" are rectangles in 2D or parallelepipeds in 3D. However, the Finite Volumes method allows using volumes of any shape and therefore to treat complex geometries, unlike Finite Differences. Numerous numerical simulation codes in fluid mechanics are based on this method: Fluent, StarCD, CFX, FineTurbo, elsA, etc.

### 2.3.2. Finite volumes for a conservation law

Consider a conservation law of a physical quantity w in a mesh of volume Ω, involving a flow F(w) and a source term S(w). Its expression in an integral form is:

$$\frac{\partial}{\partial t}\int_\Omega w\, d\Omega + \int_\Omega \mathrm{div}\, F(w)\, d\Omega = \int_\Omega S(w)\, d\Omega \tag{5}$$

Let Σ be the surface of the mesh, of external normal n. The Ostrogradski theorem leads to:

$$\frac{\partial}{\partial t}\int_{\Omega} w\, d\Omega + \oint_{\Sigma} F.n\, d\Sigma = \int_{\Omega} S\, d\Omega \qquad (6)$$

The integral represents the sum of flows through each $\oint_{\Sigma} F.n\, d\Sigma$ side of the mesh. The flow is assumed constant on each face, the integral is reduced to a discrete sum on each face of the mesh, therefore:

$$\oint_{\Sigma} F.n\, d\Sigma = \sum_{\text{Faces of the mesh}} F_{face}.n_{face}\,\Sigma_{face} \qquad (7)$$

The quantity is an approximation of the flux F on one face of $F_{face} = F(w_{face})$ the volume, it is the numerical flow on the considered face. The spatial discretization is equivalent to calculating the flux balance on an elementary volume "cell". This balance includes the sum of the contributions evaluated on each face of the mesh. The way in which one approaches numerical flow as a function of the discrete variable determines the numerical scheme. The numerical scheme can also use an auxiliary variable, for example the gradient of the variable per mesh.

In regard to the temporal derivative, a fundamental element of the discretization in Finite Volumes is to assume that the quantity w is constant in each mesh and equal to an approximate value of its mean on the cell or its value at the center of the cell. On the other hand, the time derivative is evaluated by means of a numerical method of integration of the differential equation (e.g. Runge-Kutta, Euler explicit or implicit, etc.) and involves an integration time step Δt, the latter being either a constant or a variable. To formulate the ideas, we write the formulation with an explicit Euler method. If Δw is the increment of the magnitude w between two successive time iterations. We can thus write:

$$\frac{\partial}{\partial t}\int_{\Omega} w\, d\Omega = \Omega\left(\frac{dw}{dt}\right)_{\text{mesh}} = \Omega\,\frac{\Delta w}{\Delta t} \tag{8}$$

Finally, the conservation law discretized with the Finite Volume method can be written:

$$\Omega\,\frac{\Delta w}{\Delta t} + \sum_{faces} F_{face}.n_{face}\,\Sigma_{face} = \Omega\, S \tag{9}$$

The Finite Volumes method therefore consists of:

- Decompose the geometry into elementary cells (develop a mesh),
- Initialize the quantity, w, on the computational domain,
- Start the process of temporal integration until convergence with:
    - Calculation of the flux balance by mesh by a numerical scheme,
    - Calculation of the source term,
    - Calculation of the temporal increment by a numerical integration method,
    - Application of boundary conditions.

### 2.3.3. Mono-dimensional case

Consider a 1D conservation law:

$$\frac{\partial}{\partial t}\int u\, dx + \int \frac{\partial f(u)}{\partial x}\, dx = 0 \tag{10}$$

where u is a physical quantity dependent on the variables of space x and time t; and f(u) is a “flux” function of u.

The computational domain is divided into N cells of center $x_i$. Each cell has a size

$h_i = x_{i+1/2} - x_{i-1/2}$. The half-integer indices denote the interfaces of the cell with the neighboring cells (see figure below).

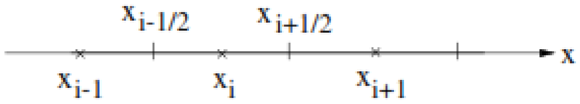


Figure 1.1 mesh 1D

Time is discretized in constant step intervals, Δt. The function u is assumed to be constant in each cell and equal to an approximate value of the mean. Let $u_i^n$ denote this average value in the i-th cell with center at $x_i$, and time t = nΔt, thus:

$$\forall x \in [x_{i-1/2}, x_{i+1/2}] \text{ and } t = n\Delta t, \quad u(x,t) = u_i^n$$

This mean value is often the value of the function u at the center, $x_{i,}$ of the cell, then we speak of Cell-Centered Finite Volumes, in this case $u_i^n = u(x_i, t)$ .

The spatial discretization by the Finite Volumes consists of integrating the conservation law cell by cell:

$$\frac{\partial}{\partial t}\int_{\text{mesh}} u\,dx + \int_{\text{mesh}} \frac{\partial f(u)}{\partial x}\,dx = 0 \tag{11}$$

Thus, for the i-th cell at time t = nΔt:

$$\frac{\partial}{\partial t}\int_{x_{i-1/2}}^{x_{i+1/2}} u\,dx + \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial f}{\partial x}\,dx = 0 \tag{12}$$

This is integrated as follows:

$$h_i \frac{\partial u_i^n}{\partial t} + \hat{f}_{i+1/2}^n - \hat{f}_{i-1/2}^n = 0 \tag{13}$$

The quantity $\hat{f}^n_{i+1/2}$ designates an approximation of the flux f(u) at the interface x $_{i+1/2}$ and the time nΔt. This is the numerical flow at the point x $_{i+1/2}$. This numerical flux is evaluated as a function of the average values of u in the neighboring cells, which determines the numerical scheme. An explicit Euler method is used to evaluate the time derivative (other schemes can be used). The discretized formulation in Finite Volumes of the conservation law is thus:

$$h_i \frac{u_i^{n+1} - u_i^n}{\Delta t} + \hat{f}^n_{i+1/2} - \hat{f}^n_{i-1/2} = 0 \tag{14}$$

## 2.4. The finite elements in 1D

### 2.4.1. Principle

The Finite Elements method consists in approximating, in a finite-dimensional subspace, a problem written in variational form in a space of infinite dimension. This variational form is, in general, equivalent to a form of minimization of energy (principle of virtual works). The approximate solution is, in this case, a function determined by a finite number of parameters, for example, its values at certain points (the nodes of the mesh). This method is particularly well adapted to problems of equilibrium. It allows one to treat complex geometries unlike Finite Differences, but it requires a great deal of computing time and memory. Many structural calculation codes are based on Finite Elements: ANSYS, CADDS, CATIA.

## 2.5. Chosen method and motivation

This thesis will present the basic ideas of the finite difference method, which is undoubtedly the most intuitive, simplest and most widely used method for numerically solving partial differential equations. Indeed, this method involves utilizing many concepts or numerical problems common to the various methods. Admittedly, unlike the finite elements and finite volumes, this technique is not suitable for non-Cartesian meshes but it is very intuitive. In addition, it is being used in numerous numerical analyses because the discretization is trivial and that the numerical scheme typically converges well. More than that, in 1D, the three methods can be shown to be equivalent.

It should be remembered that numerically solving partial differential equations means calculating a good approximation of the solution, if the problem has a unique solution, with a number of well-distributed points on the set where it is defined. The finite difference method approximates the operators by the Taylor formula. When evaluating the operators, we can choose to evaluate the spatial operators in two ways: (i) either at time t we then use an explicit scheme, (ii) or at time t + 1, an implicit scheme is then used.

# CHAPTER III

# ELECTROOSMOTIC FLOW AND ELECTRIC DOUBLE LAYER THEORY

Etymologically: electro-osmosis is derived from osmosis with the electro-prefix. It means “(Electricity) (Chemistry)” and “Movement of a fluid created by the force of Coulomb”. An electric field generates the Coulomb force that sets in motion the free charges in the diffuse layer. The movement of these charges, via the viscous bonds, transports the fluid. Thus, the electro-osmosis is defined as a phenomenon resulting from the movement of a fluid when a tangential electric field is applied to the diffuse layer.

This is the inverse phenomenon of the flow potential, where a transfer of liquid through a membrane causes a potential difference on both sides of the membrane. The electro-osmosis is a parasitic phenomenon in electro-dialysis, causing a water transfer that has the effect of diluting the solutions.

The electro-osmosis, which involves moving the constituents from the liquid phase of a porous medium by applying an electrical field as foresaid, is involved in many areas: geotechnical, soil remediation, biotechnology...

## 1. Microscopic modeling of electro-kinetic phenomena

### 1.1. Presentation of electro-kinetic phenomena

When an ionic solution and a solid are in contact, some physical phenomena, called "electro-kinetic phenomena ", occur. They are characterized either by the movement of liquid or solid particles generated by an electric field, or by the appearance of an electric current or a potential difference under the effect of a liquid or particle movement.

The main electro-kinetic phenomena are the electro-osmosis and the flow potential, the electrophoresis and the sedimentation potential (Figure 1.1.). They always manifest as follows:

- Electro-osmosis: consider a U-tube with a porous medium at the base and filled with an ionic liquid. With two electrodes placed on either side of the solid, the application of an electric field between the electrodes causes the movement of liquid through the porous medium: there is an increase of the water level in one of the branches of the tube and a decrease in the other, until reaching an equilibrium position.
- Flow potential: reciprocal phenomenon of electro-osmosis, the flow potential is a potential difference between the electrodes caused by the movement of all of the solution through the porous medium.
- Electrophoresis: applying an electric field between the two electrodes placed in the liquid causes the migration of the suspended solid particles in the liquid toward one or the other electrode.
- Potential of sedimentation: reciprocal to electrophoresis, displacement of particles suspended in an ionic liquid causes an electrical potential difference called potential of sedimentation.

The observation of electro-osmosis phenomenon dates from the early nineteenth century.

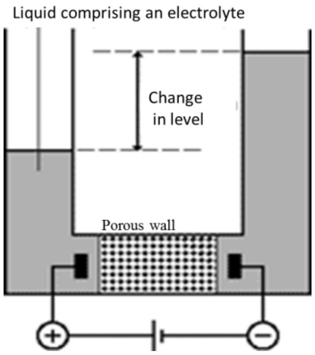


Figure 1.2 Electro-Osmosis - flow potential

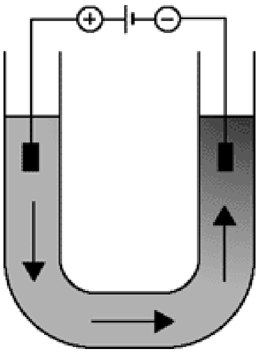


Figure 1.3 Electrophoresis - sedimentation potential

**Reuss F.** [5] was the first to study in detail electro-osmosis, by a classic experiment, the electrolytic decomposition of water through quartz powder. He was also the first to observe electrophoresis. Between 1852 and 1856, **Wiedemann G**. [6] [7] carried out the first quantitative measurements of the phenomenon and brought two important results: the difference in hydraulic pressure due to the overflow of liquid between the two sides of the porous medium is proportional to the electrical potential applied and is independent of the medium size. If the liquid is kept at the same level on both sides, the hydraulic flow is proportional to the applied electric potential and independent of the medium dimensions.

**Quincke G** [8 Quincke G. 1859] discovered the flowing potential and revealed experimentally that the direction of the electro-osmotic flow is not always the same as the electric current [9 **Quincke, G.: 1861**]. He was the first who suggested the existence of opposite charge layers to the solid-liquid interface, namely, the electric double layer: the layer charge of the liquid may move freely while that of the solid is immobile.

In 1878, **Dorn** [10] discovered and studied the potential of sedimentation, which today also carries the name of the “Dorn effect”.

Various theories have been developed to describe the electro-osmosis’ phenomenon and to quantify the induced hydraulic flow. The next section will present those that are most commonly known in the literature and those based on a microscopic scale description. They represent the phenomena at a pore scale. Among them, those based on the electrical double layer are of a particular interest here.

### 1.2. Spiegler's friction model

In this model [[11]***Spiegler, K. S****. : 1958*], the transport process caused by hydraulic, electric and osmotic forces are described in terms of ion concentrations, but also in terms of some coefficients of friction between the components. The friction between the moving components and the stationary porous medium is also included in the formulation. However, it is difficult to obtain accurate measurements of these parameters, and there is no comprehensive data for all types of environments. Therefore, this model is not predictive but heuristic. Its role is to provide a relatively simple picture of the complex transport mechanisms involved.

### 1.3. Ionic hydration model

This model assumes that the potential difference applied to both sides of a porous medium causes a migration of ions [[12]***Yeung****, A. T.: 1994*]. However, as these ions are hydrated they thus carry with them water molecules. Therefore, the amount of water that these ions transport by migrating is given by:

$$H = t_+N_+ - t_-N_-$$

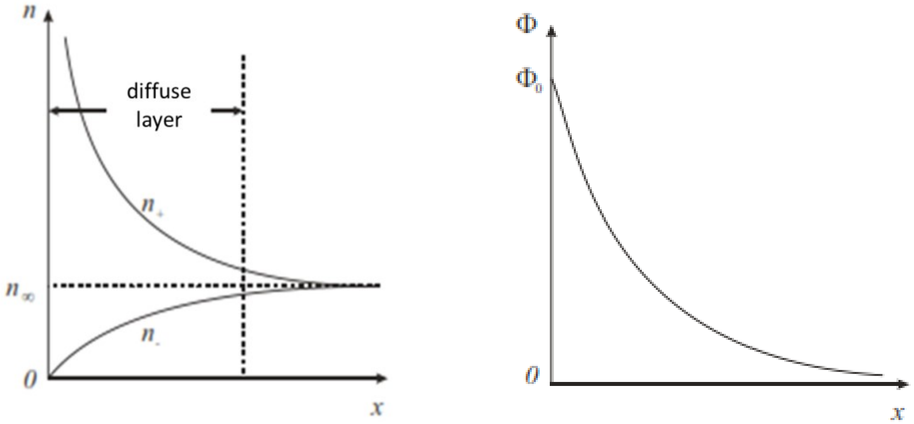


Figure 1.4 Electrical potential (Φ) and distribution of cations (n+) and anions (n-) as a function of the distance to the wall x

[[13]***Mitchell****, J. K. : 1993*].

Where H is the number of water molecules transported by electric charge passing through the medium; t + and t- are, respectively, the cation and anion transport coefficients; n+ and n- are, respectively, the number of water molecules bound by hydration transported with the cations and anions. The quantities of water transported predicted by this model, in ideal conditions, are far below those measured experimentally.

2. Theories based on the double layer

2.1. Definition

The solid wall carries an electric charge that we assume negative in this work. It attracts, by electrostatic forces, the cations (positively charged) that are in the solution. They therefore have a higher concentration near the wall, and try to diffuse since the thermal motion tends to harmonize the concentrations. They are restricted in this diffusion by the electric field created in the solid surface. Both actions will eventually counterbalance, to create an ions distribution at steady state (Fig. 1.3) [**Mitchell 1993**]. The opposite occurs for anions whose concentration is reduced near the wall. The distribution of cations is similar to that of air molecules in the atmosphere, where the escape tendency of the gas is balanced by the gravitational pull of the earth. The charged surface and the adjacent portion in which the charge is distributed are referred to as the "diffuse double layer". Several models attempt to describe it qualitatively and are themselves treated quantitatively by different theories.

The modeling of the double layer is based on a static description without an external electric field applied. However, theories that quantify the various transport coefficients are processed by fluid dynamic methods. The force induced by an external electric field has the following form:

$\vec{F} = q_e \vec{E}$ where: $\vec{F}$ is the induced force,

$q_e$ the electric charge of the medium,

and $\vec{E}$ the electric field.

This force acts on the diffused layer which is charged, causing a flow. There is no electrical effect on the neutral layer but it is driven by the diffuse layer by viscosity, resulting in a linear velocity profile in this area. The purpose of these theories is to know the velocity distribution inside the medium.

Thus, to enable the processing of the results, a linearization into two parts in a capillary is done (Figure 1.5.) using the following simplifications:

- The neutral layer speed is taken to be constant;
- The velocity at the wall is zero and increases linearly in the diffuse layer, up to the neutral layer.

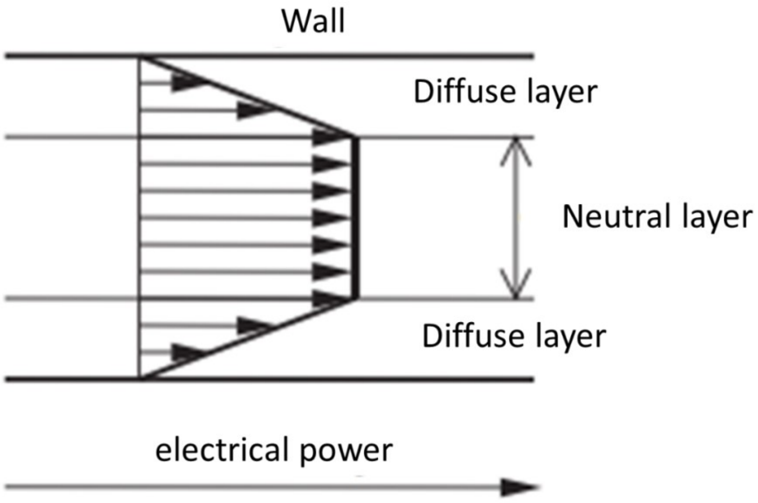


Figure 1.5 Linearized distribution of velocity in a capillary.

3. The double layer and electro-kinetic phenomena

Many studies are based on the double layer for interpreting the electro-kinetic phenomena. Consider the following electro-kinetic processes:

- By applying an electric field on a porous medium, the charges of the diffuse layer move in a direction, which depends on the sign, and orientation of the electric field; they drag the fluid because of the viscosity forces. This movement of the fluid defines electro-osmosis;
- Reciprocally, the flow of liquid through a porous medium causes movement of charges of the diffuse layer, which generates a potential difference of both sides of the porous medium, called flow potential;
- Under the effect of an electric field, the charge of moving particles suspended in an ionic liquid separates, setting in motion the particles, defining electrophoresis;
- Under the influence of gravitational forces, a downward movement animates the suspended particles in the liquid; this movement in the diffuse layer between the charges of the particles, causes a difference of potential, known as sedimentation.

## 4. The double-layer models

In general, the (solid) surfaces are charged and are in contact with solutions containing ions (electrolytes) (solid-fluid), with a consequent non-uniform redistribution of the ions. This redistribution of ions near the interface will be determined by both the electrostatic interactions and the diffusion associated with the thermal agitation.

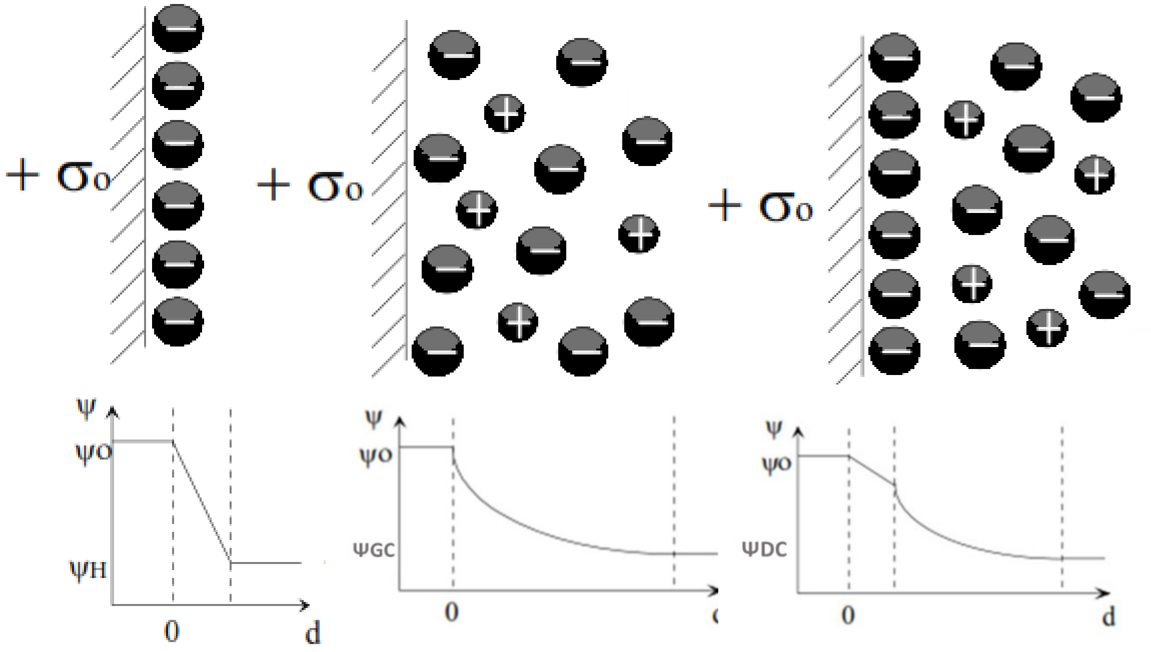
The main causes of the appearance of this charge in the interface are:

- Differences in the electronic affinities of the electrons of each phase,
- Differences in the electronic affinities of the ions of each phase,
- Retention or physical trapping of ions.

Several models have been developed to describe this interface.

### 4.1. Double layer and distribution of electrostatic potential near a charged plate

Consider an electrolyte where we would place a charged plate whose charge per unit area is given by $+\sigma_o$ (c / m$^2$). The simplest representation of the interface between this charged plate and the electrolyte is given by the Helmholtz model [14 **Russel, W. B.; Saville, D. A.; Schowalter, W. R**.]. This model, shown in Figure 1.6 (a), shows the distribution of ions in the interface and, in particular, in the region close to the surface of the plate. Thus, it is found that the ions having an opposite charge (counter ions) to that of the plate, will undergo an attraction and will position themselves near the surface of the plate, thus forming a compact layer called the Helmholtz layer. The electrostatic potential ψ then decreases linearly from ψ at the surface of the plate to reach a certain value ψ at the outer plane of the Helmholtz layer. This model (capacitor type) remains rather simplistic and its biggest disadvantage comes from the fact that it does not take into account the thermal agitation of the ions.



(a) (b) (c)

Figure 1.6 Different models representing the distribution of ions near a charged surface with the variations of electrostatic potential as a function of distance: (a) Helmholtz model, (b) Gouy-Chapman model, (c) Stern model or double layer.

The second model developed by Gouy and Chapman [$^{15}$**Hunter, R. J.**,1987.] is shown in Figure 1.6 (b). This model takes into account the disorder created by the thermal agitation but ignores the effect of the arrangement of the atomic structure of the plate on the ions located very close to the surface. The electrostatic potential then decreases exponentially along this layer of ions to reach a value $\psi_{G-C}$ at the outer plane of this diffuse layer.

The third model presented in the Figure 1.6 (c), originally developed by Stern and completed later by Grahame [16], is a combination of the two previous models. It consists of a compact first layer called the Stern layer, followed by a second diffuse layer where we find co-ions (ions with the same sign as the charged plate) and counter-ions. The co-ions are attracted by the presence of the Stern layer (made up of counter-ions) at the same time as they undergo a repulsion from the plate. The counter-ions are also subjected to attraction and repulsion from the plate and the Stern layer, respectively. To this must be added the effect of thermal agitation, causing this diffuse layer to have complex properties. This model is generally known as the double layer model and it is important to note that the concentration of counter-ions is considerably higher than that of the co - ions in this double layer. For the electrostatic potential there is a linear decrease in the Stern layer and then an exponential decrease in the diffuse layer until reaching a value $\psi_{G-C}$. However, the $\psi_{G-C}$ value of the electrostatic potential is not the final value, the electrostatic field will continue to decrease until a zero value is reached in the solution. Note that in literature there are other more complex models such as the triple layer model [17**Yates, D. E.; Healy, T. W.** 1975, 18 **Davis, J. A.; James, R. O.; Leckie, J. O.** 1978], or even four-layer models [19**Charmas, R.; Piasecki, W.; Rudzinski, W** 1995]. This thesis will focus on the double layer model. In physical terms the role of the double layer is to restore an equilibrium situation after the disturbance caused by the introduction of additional charges (i.e. the charged plate).

### 4.2. Models

So far, the model of the double layer has been described qualitatively, yet it is important to establish the equations that will characterize this double layer and in particular to characterize the spatial distribution of the electrostatic potential. Since this potential depends on the ion concentration, it would therefore be useful to characterize it as a function of this concentration.

The Helmholtz-Perrin model

**Von Helmholtz** [20Von Helmholtz, H. L. F. : 1879] introduced the concept of the double layer by repeating the idea of the existence Quincke charges layers opposed to the solid-liquid interface and established theoretically the equations of Wiedemann and Quincke.

**Perrin** resumed this calculation considering these layers as a virtual capacitor with flat faces [21Perrin,J. : 1904] [22Perrin, J. : 1905]. One of the plates, immovable, is the wall of the solid (negatively charged in this case). The other (positively charged), mobile and infinitely thin, is located in the liquid and passes through the center of gravity of the charges that are affixed to the wall by electrostatic forces. The distribution of ions and the corresponding potential are shown in Figure 1.5. Thus the zeta potential of the double layer corresponds to the potential separating the plates of the virtual capacitor:

$$\xi \quad = \frac{\sigma c e p}{\varepsilon} \tag{15}$$

where, $\xi$ is the potential difference between the two layers; $\sigma_c$ is the surface electric charge; $e_p$ is the thickness of the double layer; and $\varepsilon$ is the dielectric constant of the liquid. This scheme qualitatively explains electro-kinetic phenomena, but makes the capacitance of the capacitor constant, which contradicts experience.

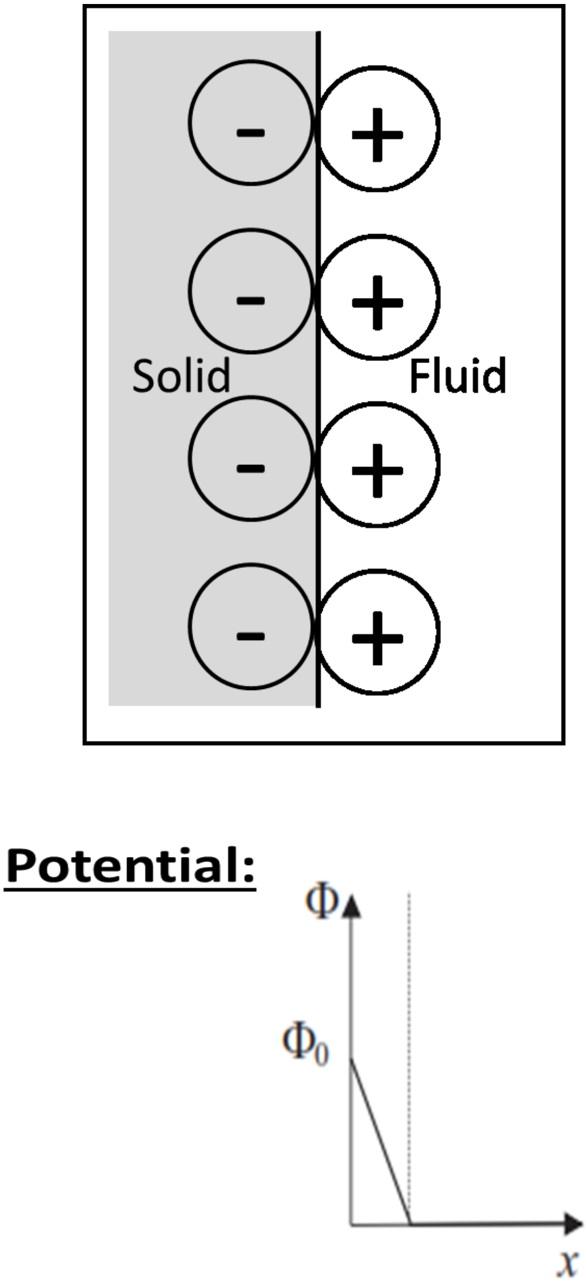


Figure 1.7 The model of double layer of Helmholtz-Perrin

## The model of Gouy-Chapman

Almost at the same time, two different studies, the first on the constitution of the electrical charge at the surface of an electrolyte [[23]**Gouy**, G. : **1909]** and the second on the theory of electro-capillarity [[24]**Chapman**, D. L. : **1913**], point out that the hypothesis of a strictly fixed arrangement of ions in the double layer is illusory. Indeed, to the electrostatic forces between the solid and the liquid, are added the thermal agitation forces. This leads to an ionic balance and a distribution presented in Figure 1.3. Their model, proposed separately, corrects this issue by considering the ions as point charges (Fig. 1.8) [**Gouy 1910 Chapman in 1913**]. This is called the **diffuse double layer**.

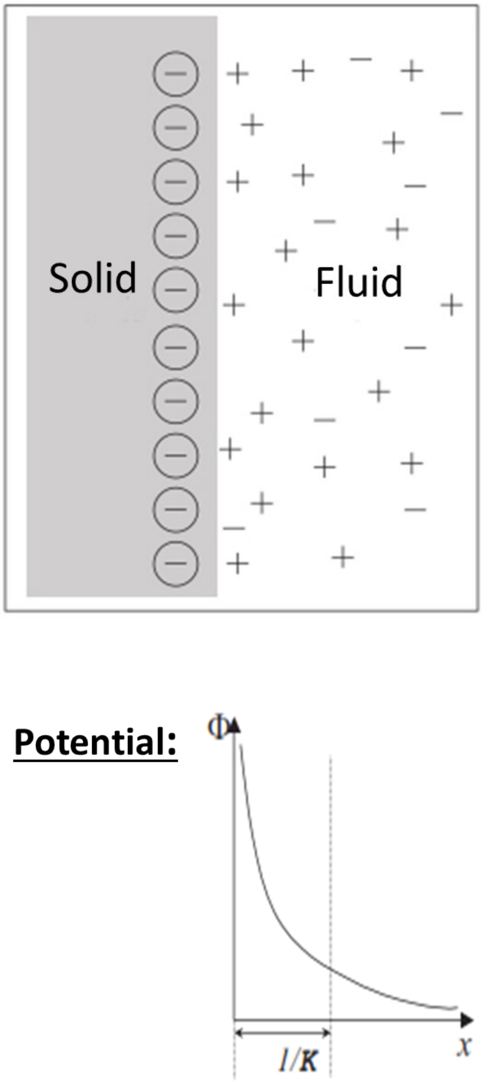


Figure 1.8 The model of double layer of Gouy-Chapman

The model of Stern:

**Stern** [[25]**Stern, O. : 1924**] corrected the model of Gouy-Chapman taking into account the size of ions and defining a maximum approach plane of ions to the solid wall, the Stern plane, located 4 or 5 Å or the equivalent of a hydrated ion radius. Furthermore, this model is based on a new concept, the specific adsorption of ions. Some ions are held in the vicinity of the solid-liquid interface to form a compact layer. This layer is very thin, cannot be affected by the flow, and remains stationary relatively to the solid. Beyond this compact layer, the ions are spread diffusely to form the diffuse layer (Fig. 1.9). Its thickness, which depends on the electrical resistivity of the liquid, is highly variable. Stern found that the maximum approach plane was the same for anions and cations.

The adsorption process reflects an ionic or electronic exchange at the interface. There are two different kinds of processes:

- Chemisorption or specific adsorption of ions: this is a chemical reaction that is at the origin of the ion exchange, hence the specificity of adsorption. This process could be at the origin of the creation of the double layer: ions adsorbed specifically by the solid are considered part of the solid. They create a surplus of opposite sign charges in the fluid that are distributed within the compact and diffuse layers. The compact layer (or double layer of Helmholtz or double inner layer) is between the solid wall and the Stern plane, which passes through the center of ions attached to the wall. Its thickness is a few atomic radius (4 or 5 Å). The ions composing it are specifically adsorbed and thus immobile. The solid surface potential, or Helmholtz potential is $\Phi 0$; that of Stern's plane $\Phi_d$. The potential distribution in this layer is linear because there is no free charge.

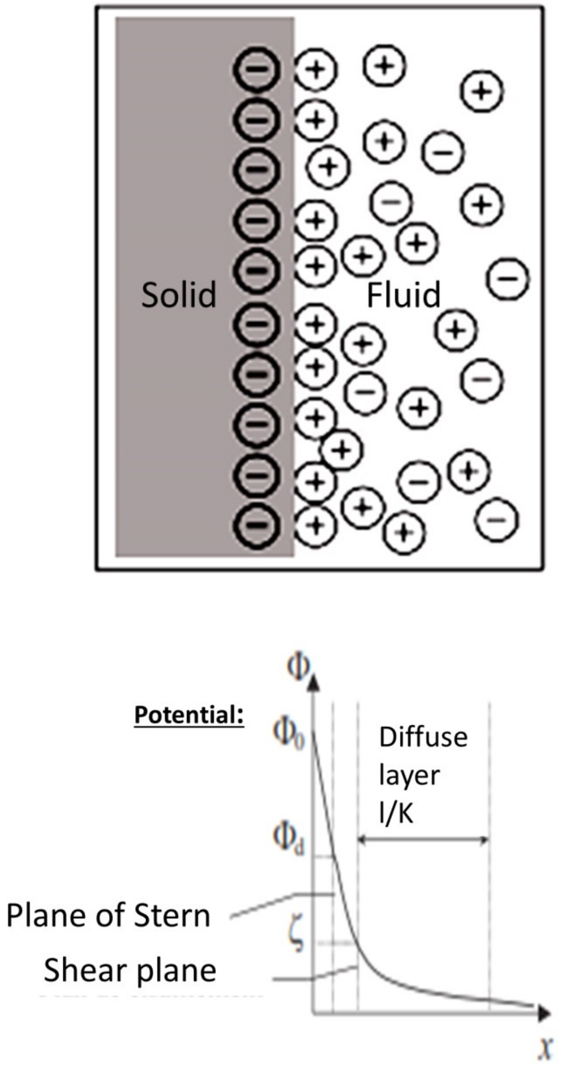


Figure 1.9 The model of double layer of Stern

- Physical adsorption or non-specific adsorption of ions: these are the van der Waals forces that attract fluid and ions and attach them to the solid to form the diffuse layer. Ions can change the adsorption site or move to the surface of the solid. Ion distribution in the diffuse layer is identical to that of the double layer of Gouy-Chapman, the potential distribution is governed by the theory of Gouy-Chapman.

The centers of the ions "attached" to the solid surface are on the Stern plane; ions whose centers are located after the Stern plane form the diffuse part of the double electric layer and are likely to move. Thus, the boundary between these ions is between one and two rays from the surface. This limit is called the shear plane. The potential of this plane is called the electro-kinetic potential, better known as the zeta potential (ζ). This potential that has the property of being experimentally measurable, is the subject of many studies to find if it can, and to what extent, represent an intrinsic material parameter.

The double layer can be seen as being composed of two capacitors in series. The differences between the experiment and the predictions of the theory of Gouy-Chapman are mainly due to the fact that it does not take into account the capacitor formed by the compact layer. However it is the capacitance of the capacitor that is causing problems because it is not accessible to measurement [26 **Delahay, P. : 1965**].

The model of Grahame

Based on the model of Stern, Grahame [1947] suggests that the capacitance of the double layer does not depend on the concentration of the electrolyte and divides the compact layer into two parts: internal and external (Figure 1.8.) [27 **Usui, S. : 1984**]. The specifically adsorbed ions are dehydrated or a little hydrated, they can approach the surface nearest the Stern plane, until the interior plane. The other layer consists of hydrated ions, attracted to the wall by electrostatic forces, and defining the external plane. The Gouy-Chapman theory can then be applied to the area situated after the external plane, i.e. the diffuse layer. This model, although more comprehensive and more generally in accord with experiment, is also more cumbersome to use.

## 4.3. The Poisson-Boltzmann theory

### 4.3.1. Electric double layer.

As mentioned previously, when a charged surface is in contact with a solution containing ions, the ions are distributed in a non-uniform manner, thus creating an electrostatic potential. This distribution of charges around the charged surface has been described in several models. The one proposed by Helmholtz in 1879 is the oldest and the simplest. As previously stated, it is based on a physical model that considers that ionic species form a single layer as in a plane capacitor. The counter-ions present in the solution are preferably placed in front of the charged surface to restore the electro-neutrality of the surface charges. This layer of counter-ions is supposed to be immobile on the surface of the particles and is called the Helmholtz layer.

Then the other model, introduced by Gouy (1910) and Chapman (1913) was constructed to calculate the electrostatic potential around the charged surface. In this theory, the charged surface is assumed a plane, of infinite extent and uniformly charged. This model, unlike the previous, takes into account the ions diffusion due to thermal agitation. As the thermal fluctuations tend to repulse counter-ions away from the charged surface, this leads to the formation of a double layer that combines the Stern layer and the more extended diffuse layer. The fictional boundary between these two layers is materialized by the Helmholtz plane. The quantitative studies relating to this double layer model are governed by two equations [[28]Masliyah, J. H. : 1994]: the Poisson’s equation and the Boltzmann equation.

### 4.3.2. Distribution of Boltzmann

In the case of a colloidal dispersion in contact with an ion reservoir, the main hypotheses in Boltzmann's theory assume that:

- Ions are considered as point charges but also decorrelated from each other.
- The dielectric properties of the solvent are uniform.
- The potential of the average force is equal on average to the local potential.

The distribution of ions in the electric double layer can be modeled using Boltzmann's theory. In the Gouy-Chapman model, the ions of the diffuse layer are in equilibrium, the Coulomb force is equal to the thermal force. The sum of the two forces is therefore zero, which implies that the electrochemical potential gradient is written in the following manner:

$$k_B T \nabla(\ln n_k) + e z_k \nabla \psi = 0 \tag{16}$$

with $k_B$ the Boltzmann constant, T temperature, $z_k$ the valence of the species $k$, $n_k$ the density number of species $k$, ψ the local electrostatic potential, and $e$ the charge of an electron. The integration of the previous equation gives the expression of the ion density:

$$k_B T \int_{n_k}^{n_k^{\infty}} \nabla(\ln n_k) + e z_k \int_{\psi}^{0} \nabla\psi = 0 \tag{17}$$

with $n_k^{\infty}$ being the density of the species k far from the charged surface, where the electrostatic potential is zero (ψ = 0). This shows that the ion density follows the Boltzmann distribution linking the local concentration of ions (density $n_k$) to the local electrostatic potential:

$$n_k = n_k^{\infty} \mathrm{e}^{-\frac{e z_k \psi}{k_B T}} \tag{18}$$

### 4.3.3. Poisson-Boltzmann equation

The Poisson equation links the electrostatic potential to the charge density. Indeed, the electrostatic potential ψ follows one of Maxwell's laws in the charged particle and in the solvent. It is Gauss's law for electricity which, in differential form, is written as follows:

$$\nabla \cdot (-\epsilon \nabla \psi) = \rho^f \tag{19}$$

with ϵ the relative electrical permittivity of the medium and ρ the volume density of mobile charges. This density is defined by:

$$\rho^f = \sum_{k=1}^{N} e z_k n_k \tag{20}$$

with N representing the number of ionic species present in the electrolyte of the charged system.

### 4.3.4. Poisson's equation:

By combining equations (18) and (19), the Poisson-Boltzmann equation can be written in the following way:

$$\nabla \cdot (-\epsilon \nabla \psi) = \sum_{k=1}^{N} e z_k n_k^{\infty} \mathrm{e}^{-\frac{e z_k \psi}{k_B T}} \tag{21}$$

This equation allows the calculation of the electrostatic potential created by the presence of the charged particle surrounded by an ionized solvent. It takes into account the electrostatic interactions between the charged particles and the surrounding ions, while also considering the thermal agitation of the ions.

### 4.3.5. Approximation of Debye-Hückel : Linearization of the Poisson - Boltzmann equation

The analytical solution of the Poisson-Boltzmann equation (21) is far from simple except for trivial geometries. The Debye- Hückel approximation therefore consists of looking for an approximate solution of this equation, considering that the electrostatic potential is low throughout the double layer or in other words that the electrical energy is lower than the thermal energy. Knowing that $e\psi << k_B T$, a development limited to the first order of the exponential can be done and gives:

$$\mathrm{e}^{-\frac{e z_k \psi}{k_B T}} = 1 - \frac{e z_k \psi}{k_B T} \tag{22}$$

This hypothesis allows the linearization of  equation (22) which becomes:

$$\Delta \psi = -\frac{1}{\epsilon} \sum_{k=1}^{N} e z_k n_k^{\infty} + \frac{1}{\epsilon} \sum_{k=1}^{N} \frac{e^2 z_k^2 n_k^{\infty} \psi}{k_B T} \tag{23}$$

However, the systems are electrically neutral and the term containing the expression $\sum_{k=1}^{N} ez_k n_k^{\infty}$ is annulled. The equation (23) is simplified to become:

$$\Delta\psi = \sum_{k=1}^{N} \frac{e^2 z_k^2 n_k^{\infty}}{\epsilon k_B T}\psi \tag{24}$$

The linearized Poisson-Boltzmann equation can be written as follows:

$$\Delta\psi = \kappa^2\psi \tag{25}$$

where:

$$\kappa^2 = \frac{2e^2}{\epsilon kT}\sum \frac{1}{2}\nu_i^2 n_{i\infty} \tag{26}$$

The parameter κ, and more precisely its inverse ($\kappa^{-1}$), represents the thickness of the double layer and is known as the Debye length. It is a function of the ionic force, thus:

$$\kappa^{-1} = \sqrt{\frac{\epsilon k_B T}{2e^2 n^{\infty}}} \tag{27}$$

This approximation, valid in the case of a low surface potential, can enable the analytical solution of the Poisson-Boltzmann equation. This estimate, which leads to the linearization of the Poisson-Boltzmann is known as the Debye-Hückel approximation.

The ions are seen as point charges in this model, they can approach without limit of the solid wall, and even stick. The electrical capacity of the double layer can then reach values much higher than those observed experimentally, showing the limits of this model, that Gouy had already highlighted [[29]**Durand-Vidal, S. and Simonin, J. P. : 2000**]

# CHAPTER IV

# THEORY OF THE POISSON-NERNST-PLANK EQUATION SYSTEM

Habitually, in the electrokinetic studying methods, the so-called Nernst-Planck equations are used for charged-species flow analysis. W. Nernst and P. Planck were the first to consider, at the end of the last century, that the ion transport was done under the influence of several driving forces. The Nernst-Planck equations reflect the fact that the total molar ion flux, $J_i$, of species i, represent the sum of several flows:

$\overrightarrow{J_i^{mig}}$ of migration $\overrightarrow{J_i^{conv}}$ of convection $\overrightarrow{J_i^{diff.}}$ of diffusion (28)

The diffusion flow $\overrightarrow{J_i^{diff.}}$ of a species is given by Fick's law:

$$\vec{J}_i = -D_i \overrightarrow{\mathrm{grad}}\, C_i \tag{29}$$

## 1. Fick and Ohm empirical law

In what follows, will be examined how general law explicates the empirical laws observed for the distribution, migration and movement, under pressure.

### Diffusion

The first empirical Fick's law states that the diffusion flux, for which the only driving force is a concentration gradient (grad ϕ = grad p = 0), remains proportional to that gradient. In the case of linear systems, the diffusion flow is thus given by:

$$J_i = -D_i \left( \frac{\partial c_i}{\partial x} \right)_{\phi,p} \tag{30}$$

$D_i$, as defined in this equation is a proportionality coefficient called the diffusion coefficient expressed by $m^2s^{-1}$, and particularly by $cm^2s^{-1}$.

It is important to note that the solution diffusion phenomena cannot be observed if the thermal agitation or even a mechanical one, homogenizes the solutes concentration. Diffusion phenomena in solution can, in fact, only be observed at the neighborhood of the solid walls. Indeed, at the solid/fluid interface, there is a "stagnant" solution layer called the diffusion layer with a thickness scale in the micrometer range. Thus, the diffusion phenomena are often measured in porous systems (e.g. sintered-glass membrane, polyacrylamide gel, dialysis membrane, etc.) where the convection is negligible. Fick's empirical law is easily verified by measuring, for example, the flow of dyes between two containers containing different concentrations, separated by a porous membrane. It is observed then, that the initial flow is proportional to the concentration difference, and inversely proportional to the membrane thickness. The containers color is constant because of thermal or mechanical agitation.

By comparison with what has been said previously, we have a general form (known as Einstein relation):

$$D = \mu k_B T \qquad (31)$$

where: D is the diffusion constant, μ is the mobility or the ratio of the particle's drift velocity to an applied force, kB is the Boltzmann constant, and T is the absolute temperature.

Two frequently used forms of the relation are:

- **The electrical mobility equation** (for diffusion of charged particles)

$$D = \frac{\mu_q k_B T}{q} \qquad (32)$$

- **The Stokes-Einstein equation** (for diffusion of spherical particles through a liquid with low Reynolds number)

$$D = \frac{k_B T}{6\Pi \eta r} \qquad (33)$$

where: q is the electrical charge of a particle, $\mu_q$ is the electrical mobility of the charged particle, η is the dynamic viscosity, r is the radius of the spherical particle.

The relationship between the diffusion and the electrochemical mobility coefficient is called Einstein relation, and the one linking the diffusion coefficient and the viscosity is called the Stokes-Einstein equation.

**The migration**

Electrical current is defined as a flow of positive charges between two equipotential surfaces, for example, potential $V_A$ and $V_B$. In the case of metallic conductors, this flow rate is equal to the electron flow between these two surfaces increased by the conductor cross-section area.

A proportionality empirical law between the current A to B and a metallic conductor terminal voltage, called Ohm's law, can be written:

$$R\, I_{A \to B} = V_A - V_B \qquad (34)$$

Where R is the resistance defined as proportionality factor having the units of ohm ($\Omega = V \cdot A^{-1}$). In ionic conductors, this law is also observed for each ionic species and $J_i$ current density which is the charge flow carried by species I, and is as follows:

$$\boldsymbol{j}_i = z_i F\, \boldsymbol{J}_i = -\sigma_i\, \mathbf{grad}\phi = \sigma_i\, \boldsymbol{E} \qquad (35)$$

Where σ is defined as a proportionality coefficient called the ionic conductivity whose units are $\Omega^{-1} \cdot m^{-1}$ , and also Siemens per meter($S \cdot m^{-1}$). Siemens is defined by $S = \Omega^{-1}$. The conductivity and conductance G(S) must not be confused, the latter is defined as the inverse of resistance.

A comparison with the previous phenomenological equation, where the driving force is the electric field, defined as the gradient of the electric potential (**grad** c = **grad** p = 0), shows us that:

$$\sigma_i = z_i^2 F^2 c_i \tilde{u}_i = z_i F c_i u_i = \lambda_i c_i = z_i^2 F^2 \frac{D_i c_i}{RT} \tag{36}$$

Where $u_i$ is called **electric mobility** or sometimes **electrophoretic mobility** ($m^2 \cdot V^{-1} \cdot s^{-1}$)

$$u_i = z_i F \tilde{u}_i \tag{37}$$

It is defined as the coefficient of proportionality between the speed and the electric field

$$\boldsymbol{v}_i = -z_i F \tilde{u}_i \,\mathbf{grad}\, \phi = z_i F \tilde{u}_i \boldsymbol{E} = u_i \boldsymbol{E} \tag{38}$$

By definition, the electric mobility is positive for cations and negative for anions.

$\lambda_i$ is called the molar ionic conductivity.

$$\lambda_i = z_i F u_i = \frac{\sigma_i}{c_i} \tag{39}$$

**2. Nernst – Planck Equation**

In the presence of diffusion and migration, the general equation of flow in electrochemical potential gradient mentioned above, is reduced to:

$$\boldsymbol{J}_i = -c_i \tilde{u}_i \mathbf{grad}\mu_i - z_i F c_i \tilde{u}_i \mathbf{grad}\phi = -D_i \left[ \mathbf{grad} c_i - \frac{z_i F c_i}{RT} \mathbf{grad}\phi \right] \tag{40}$$

This equation is known as the Nernst-Planck equation.

# CHAPTER V

# NUMERICAL SCHEME FOR THE NERNST-PLANCK EQUATION

As mentioned previously, at the nanometric scale the usual means for fluid manipulation and transport do not apply. Fluids confined within a nanostructure show a different behavior than when confined in larger structures. This can be attributed to electrostatic and Van Der Waals forces being as prevalent as inertial or viscous forces. Furthermore, characteristic properties intrinsic to the nanostructure's geometry (such as the Debye length) are at the same order as the dimensions of the nanostructure.

To manipulate particles on the nano-scale (say we want to separate red from white blood cells, for example), one common technique is to apply an electrostatic force on charged particles within the fluids, which would in turn induce their motion.

- **The electrical double layer (EDL)**

One important concept in the problem is that of the EDL, as mentioned previously. Most solid surfaces gain surface charges when in contact with an ionic aqueous solution. The electrostatic interaction between the charged surface and surrounding ions attract counter-ions. This causes a formation of a double layer predominantly occupied with counter-ions.

**Stern layer + diffuse layer = EDL**

The Stern layer does not allow movement due to strong electrostatic forces. The diffuse layer allows movement. The Debye length is defined as the distance of which the potential reaches 1/3 of the surface potential. It is referred to as the zeta potential. The numerical simulation needs to account for this difference between the EDL region (where gradients are high) and the bulk region (where gradients are low).

- **Electro-osmotic flow**

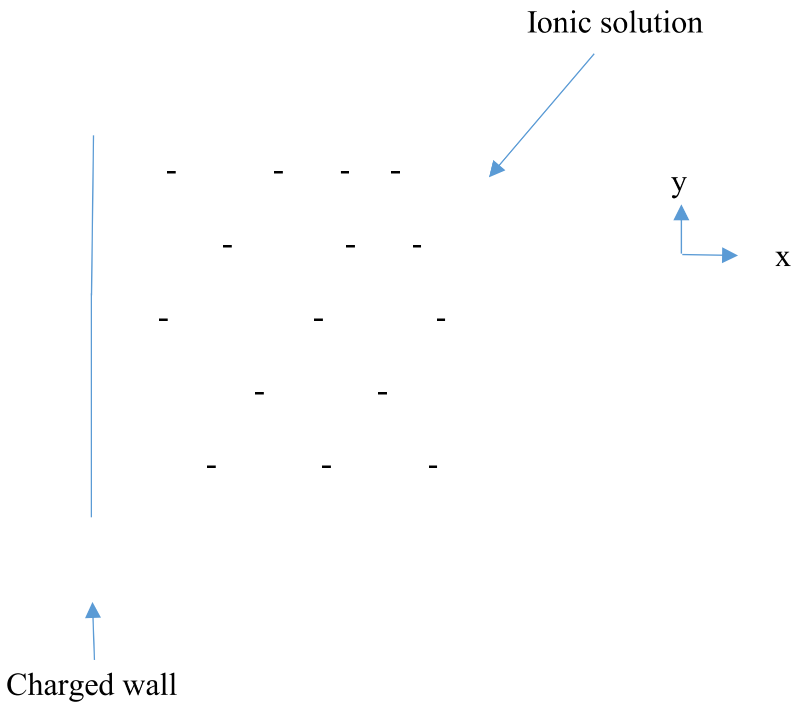
When an external electric field is applied to a stationary charged surface, the excessive counter-ions within the EDL of the charged surface migrate toward the oppositely charged electrode, dragging the fluid via viscous forces. This induced flow motion is referred to as electro-osmotic flow (EOF).

# IMPLEMENTATION IN C++ IN 1D

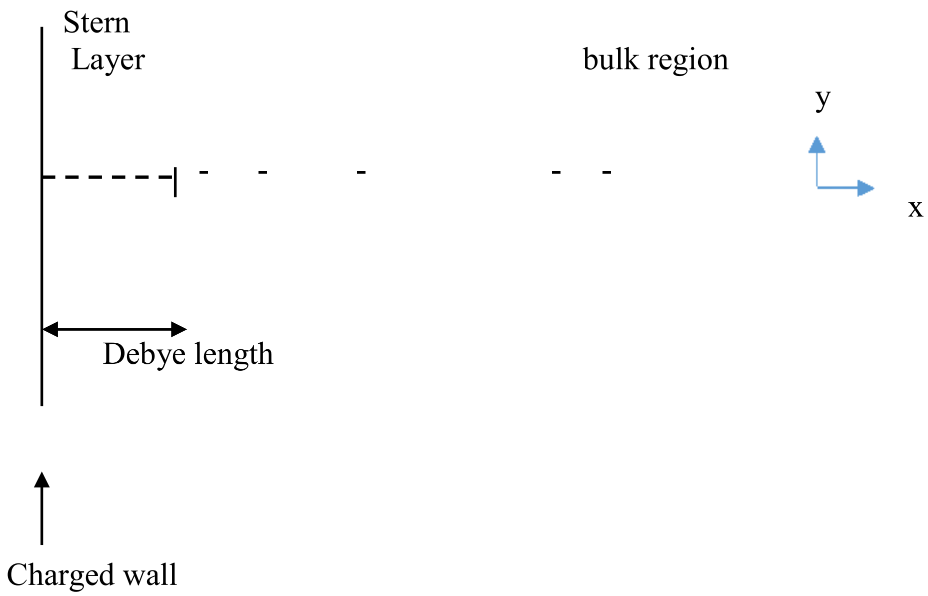
## 1. Simulation parameters and derived quantities

### 1.1. Problem definition:

The elementary problem in electric double layer theory is that of a charged wall in contact with an ionic solution.



The electric double layer also states that the domain can be roughly subdivided into two separate regions; the Stern layer in which little to no molecular movement is permitted due to the dominance of electrostatic force and the diffuse layer. As such, the situation can be schematized as follows:



### 1.2. Governing equations

If x ≤ Debye length

$$c_i = \frac{c_{bulk}}{|z_i|} e^{\frac{-z_i e\phi}{k_B T}}$$

(41) Boltzmann's ion distribution

$$\nabla(\varepsilon \nabla \phi) = -\left(\rho_o + \sum_{i=1}^{N} z_i e c_i\right)$$

Poisson equation

(42)

If x > Debye length:

$$\frac{\partial c_i}{\partial f} = \nabla \left\{ D_i \left[ \nabla c_i + \frac{z_i e}{k_B T} c_i \nabla \phi \right] \right\}$$

Nernst-Planck equation (43)

$$\nabla(\varepsilon \nabla \phi) = -\left(\rho_o + \sum_{i=1}^{N} z_i e c_i\right)$$

Poisson equation (44)

where:

$i = 1, 2, \ldots, N$ number of ionic species involved.

| | | |
|---|---|---|
| $c_i$ | : | ion concentration of species i |
| $D_i$ | : | diffusion coefficient |
| $z_i$ | : | valence number |
| $e$ | : | unit charge |
| $k_B$ | : | Boltzmann’s constant |
| $T$ | : | absolute temperature |
| $\varepsilon$ | : | permittivity |
| $\phi$ | : | potential |
| $\rho_o$ | : | charge density of the system |

### 1.3. Simplified equations and implementation

Assuming the charged wall is orders of magnitude longer that the length of the computational domain:

- A translational symmetry along the z-axis $\Rightarrow \frac{\partial}{\partial_z} = 0$
- A translational symmetry along the y-axis $\Rightarrow \frac{\partial}{\partial_y} = 0$

Therefore, the problem can be reduced to a single dimension along the x-axis.

The system is rewritten:

If x $\leq$ Debye length

$$c_i = \frac{c_{bulk}}{|z_i|} e^{\frac{-z_i e\phi}{k_B T}} \tag{45}$$

$$\frac{\partial}{\partial x}\frac{(\varepsilon \partial \phi)}{\partial x} = -\left(\rho_o + \sum_{i=1}^{2} z_i e c_i\right) \tag{46}$$

If x > Debye length

$$\left\{\begin{aligned} &\frac{\partial c_i}{\partial f} = \frac{\partial}{\partial x}\left\{D_i\left(\frac{\partial c_i}{\partial x} + \frac{z_i e}{k_B T} c_i \frac{\partial \phi}{\partial x}\right)\right\} && (47) \\ &\frac{\partial}{\partial x}\frac{(\varepsilon \partial \phi)}{\partial x} = -\left(\rho_o + \sum_{i=1}^{2} z_i e c_i\right) && (48) \end{aligned}\right.$$

where ε and $D_i$ are constants

### 1.4. Initial and boundary conditions

Initial condition: $c(t=0) = \frac{c_{bulk}}{|z|}$ ; $\phi$ (t=0)= 0 (49)

Boundary conditions: $\phi\ (x=0) = \phi_{wall}$ ; $c(x=L) = \frac{c_{bulk}}{|z|}$ (50)

$$\begin{aligned} &\text{At } x=0 && \left\{\begin{aligned} &\phi = \phi_{wall} \quad ; \quad D\left(\frac{\partial c}{\partial x} + \frac{ze}{k_o T} c \frac{\partial \phi}{\partial x}\right) = 0 && (51) \\ &\phi = 0 \quad ; \quad c = \frac{c_{bulk}}{|z|} && (52) \end{aligned}\right. \\ &\text{At } x = L && \end{aligned}$$

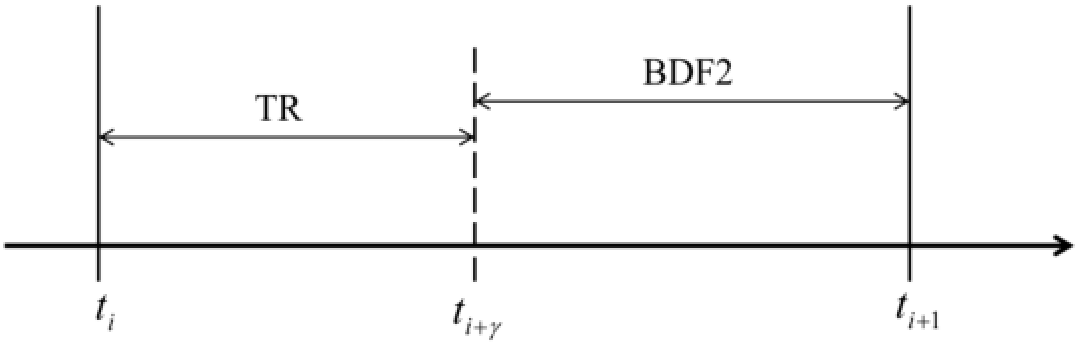
We assign to the wall a potential $\phi_{wall}$ , effectively yielding a $\phi_{wall}$ voltage across the length of the channel, due to the other end of the computational domain essentially acting as ground. We also express a no flux boundary at x = 0 and electroneutrality at x = L.

## 1.5. Discretization

To discretize the system, a composite trapezoidal and second-order backward differentiation method referred to as TR-BDF2 will be used. This algorithm is known to be stable and has sufficient damping to solve stiff problems, making it an interesting choice compared to methods such as Crank-Nicolson (which is also based on a trapezoidal method).

### 1.5.1. TR-BDF 2 method

This solution method is an algorithm known in the literature as TR-BDF2 [**Bank et al. 1985**]. It is an implicit method combining a trapezoidal (TR) method and a second order Backward Differential Formula (BDF2) method for the solution of the system at each integration step, hence its name, TR-BDF2. As illustrated in the following figure, this algorithm solves the system of $t_i$ to $t_{i+\gamma}$ with an implicit trapezoidal method, then from $t_{i+\gamma}$ to $t_{i+1}$ by a method of BDF2 implicit also, with $\gamma = 2 - \sqrt{2}$



The trapezoidal method is A-stable. When u' = au has Re a ≤ 0 the difference approximation has $|U_{n+1}| \leq |U_n|$ :

$$\frac{U_{n+1} - U_n}{\Delta t} = \frac{aU_{n+1} + aU_n}{2} \quad \Rightarrow \quad U_{n+1} = \frac{1 + a\,\Delta t/2}{1 - a\,\Delta t/2} U_n = GU_n \qquad (53)$$

The growth factor G has A-Stability :

$$|G| \leqslant 1 \text{ whenever } \operatorname{Re}(a\,\Delta t) \leqslant 0. \qquad (54)$$

The accuracy is second order: $U_n - u(n\Delta t)$ is bounded by $C(\Delta t)^2$ for $n\Delta t \leq T$. When $a$ is imaginary the stability can be very close to the edge, $|G| = 1$ , the trapezoidal method can then fail because non-linearity's can push it over the edge. To resolve this issue, alternating the trapezoidal method with backward differences BDF2 is used, since the BDF2 method is also second-order accurate.

$$\frac{U_{n+2} - U_{n+1}}{\Delta t} + \frac{U_{n+2} - 2U_{n+1} + U_n}{2\,\Delta t} = f(U_{n+2}). \qquad (55)$$

This stabilized option was proposed in [[30]**Bank et al., 1985]** for circuit simulation: the trapezoidal method determines $U_1$ from $U_0$, and then $U_2$ comes from BDF2.

The computing time in these implicit methods is often dominated by the solution of a nonlinear system for $U_{n+1}$ and then $U_{n+2}$. Some variant of Newton's method is a normal choice. Thus, an exact approximate Jacobian of the implicit part of the previous equations is needed, when a nonlinear vector f(U) replaces the scalar test case f = au. When writing f' for the matrix $\frac{\partial f_i}{\partial u_j}$, the Jacobians in the two cases are :

$$\text{(trapezoidal)} \quad I - \frac{\Delta t}{2} f', \qquad \text{(BDF2)} \quad \frac{3}{2} I - \Delta t f'. \tag{56}$$

It would have been ideal if those Jacobians were equal or proportional, but for the same Δt in the two methods, this is not the case. Therefore, the idea that **Bank et al** proposed was allowing different steps αΔt and (1 – α)Δt for the first and second method successively, with 0 < α < 1. So the trapezoidal method products $U_{n+\alpha}$ insted of $U_{n+1}$ :

$$U_{n+\alpha} - U_n = \frac{\alpha \Delta t}{2} (f(U_{n+\alpha}) + f(U_n)). \tag{57}$$

The BDF2 method determines $U_{n+1}$ from $U_n$ and the part-way value $U_{n+\alpha}$ and to maintain second-order accuracy, this requires coefficients A, B, and C that depend on α :(BDF2α) :

$$AU_{n+1} - BU_{n+\alpha} + CU_n = (1 - \alpha) \Delta t f(U_{n+1}). \tag{58}$$

here: $A = 2 - \alpha$ , $B = 1/\alpha$ and $C = (1 - \alpha)^2/\alpha$

The standard choice α = 1 / 2 gives A = 3/2 , B = 2 and C = 1 / 2. In accordance with the BDF2 previous equation, the step Δt moved to the right side is, in this case, Δt/2.

These values of A, B and C are chosen to give the exact solution U = t and U = $t^2$ when the right side are f = 1 and f = 2t, respectively.

The Newton method Jacobians in the previous equations, for $U_{n+\alpha}$ and the $U_{n+1}$ become:

$$\text{(trapezoidal)} \quad I - \frac{\alpha \Delta t}{2} f', \qquad \text{(BDF2}\alpha\text{)} \quad (2-\alpha)I - (1-\alpha)\Delta t f'. \tag{59}$$

When $\alpha = 2 - \sqrt{2}$ and f' is a constant matrix, these Jacobians are proportional :

$$\sqrt{2}\left[I - \frac{(2-\sqrt{2})\Delta t}{2} f'\right] = [\sqrt{2}I - (\sqrt{2}-1)\Delta t f']. \tag{60}$$

1.5.2. Discretization in time

As mentioned previously, the TR-BDF2 method will be used. It is a composite one step, two stage method, consisting of one phase of the trapezoidal method followed by another of the BDF2 method. The phases are so adjusted that both the trapezoidal and the BDF-2 stages use the same Jacobian matrix.

To advance the solution from t = $t_n$ to $t_{n+1} = t_n + \Delta t$, we first apply a trapezoidal rule (TR) to advance the solution to $t_{n+\gamma} = t_n + \gamma\Delta t$

$$\frac{\partial c}{\partial t} = \frac{\partial}{\partial x}\left[D\left(\frac{\partial c}{\partial x} + \frac{ze}{k_B T} c \frac{\partial \phi}{\partial x}\right)\right] \tag{61}$$

We choose $\gamma = 2 - \sqrt{2}$ , a value that minimizes the algorithm's local truncation error (see previous paragraph).

We define $f^n = \left\{ \frac{\partial}{\partial x} \left[ D \left( \frac{\partial c}{\partial x} + \frac{ze}{k_B T} c \frac{\partial \phi}{\partial x} \right) \right] \right\}^n$ The right-hand side of (61) at $t_n$ (62)

$$\Rightarrow \frac{c^{n+\gamma} - c^n}{\gamma \Delta t} = \frac{1}{2} \left( f^{n+\gamma} + f^n \right) \tag{63}$$

Thus, we write the TR step:

$$\Rightarrow \begin{cases} c^{n+\gamma} - \frac{\gamma \Delta t}{2} f^{n+\gamma} = c^n + \frac{\gamma \Delta t}{2} f^n & (64) \\ \frac{\partial}{\partial x} \left( \varepsilon \frac{\partial \phi^{n+\gamma}}{\partial x} \right) = - \left( \rho_o + \sum^{2} zec^{n+\gamma} \right) & (65) \end{cases}$$

A second-order backward difference (BDF2) is used to advance the solution from $t = t_{n+\gamma}$ to $t_{n+1} = t_{n+\gamma} + (1-\gamma)\Delta t$

$$\Rightarrow \frac{c^{n+1} - c^{n+\gamma}}{(1-\gamma) \Delta t} = \frac{1-\gamma}{2-\gamma} \frac{c^{n+\gamma} - c^n}{\gamma \Delta t} + \frac{1}{2-\gamma} f^{n+1} \tag{66}$$

Thus we write the BDF2 step:

$$\Rightarrow \begin{cases} c^{n+1} - \frac{1-\gamma}{2-\gamma} \Delta t f^{n+1} = \frac{1}{\gamma (2-\gamma)} c^{n+\gamma} - \frac{(1-\gamma)^2}{\gamma (2-\gamma)} c^n & (67) \\ \frac{\partial}{\partial x} \left( \varepsilon \frac{\partial \phi^{n+1}}{\partial x} \right) = - \left( \rho_o + \sum^{2} zec^{n+1} \right) & (68) \end{cases}$$

### 1.5.3. Discretization in space

The computational domain [0, L] is divided into $L_x$ subintervals defined by $x_i = i \Delta x$ for $i = 0, \ldots, L_x$

where $$\Delta x = \frac{L}{Lx}$$

Therefore, for $1 \leq i \leq L_{x-1}$

$$f\left(x_i\right) = \frac{\partial}{\partial x}\left\{D\left[\frac{\partial c}{\partial x} + \frac{ze}{k_B T} c \frac{\partial \phi}{\partial x}\right]\right\} \tag{69}$$

$\Rightarrow$

$$f\left(x_i\right) = \frac{D}{\Delta x^2}\left[\left(c_{i+1} - 2c_i + c_{i-1}\right) + \frac{ze}{4k_B T}\left(c_{i+1}\left(\phi_{i+2} - \phi_i\right) - c_{i-1}\left(\phi_i - \phi_{i-2}\right)\right)\right] + \Theta\left(\Delta x^2\right) \tag{70}$$

$$\frac{\partial}{\partial x}\left(\varepsilon \frac{\partial \phi}{\partial x}\right)\left(x_i\right) = \frac{\varepsilon}{\Delta x^2}\left(\phi_{i+1} - 2\phi_i + \phi_{i-1}\right) + \Theta\left(\Delta x^2\right) \tag{71}$$

## 1.6. Matrix form of the system

We define $c = (c_1, c_2, \ldots, c_{Lx})^T$ and $\Phi = (\Phi_1, \Phi_2, \ldots, \Phi_{Lx})^T$

From the previous equations, we can write the system in matrix form

TR-Step:

$$\left\{\begin{aligned} &\left(I - \frac{\gamma \Delta t}{2} F^{n+\gamma}\right) c^{n+\gamma} = \left(I + \frac{\gamma \Delta t}{2} F^n\right) c^n && (72) \\ &G\phi^{n+\gamma} = -\left(\rho_o + \sum^{2} zec^{n+\gamma}\right) && (73) \end{aligned}\right.$$

where:

$$F = \begin{cases} \dfrac{D}{\Delta x^2}\left(1 - \dfrac{ze}{4k_BT}\left(\phi_i - \phi_{i-2}\right)\right) & j = i - 1 \quad \text{lower diagonal} \quad (74) \\ -\dfrac{2D}{\Delta x^2} & j = i \quad \text{diagonal} \quad (75) \\ \dfrac{D}{\Delta x^2}\left(1 + \dfrac{ze}{4k_BT}\left(\phi_{i+2} - \phi\right.\right. & j = i + 1 \quad \text{upper diagonal} \quad (76) \end{cases}$$

$$G = \begin{cases} \dfrac{\varepsilon}{\Delta x^2} & j = i - 1 \quad \text{lower diagonal} \quad (77) \\ -\dfrac{2\varepsilon}{\Delta x^2} & j = i \quad \text{diagonal} \quad (78) \\ \dfrac{\varepsilon}{\Delta x^2} & j = i + 1 \quad \text{upper diagonal} \quad (79) \end{cases}$$

BDF2 step:

$$\begin{cases} \left(I - \dfrac{1-\gamma}{2-\gamma}\Delta t F^{n+1}\right)c^{n+1} = \dfrac{1}{\gamma\left(2-\gamma\right)}c^{n+\gamma} - \dfrac{\left(1-\gamma\right)^2}{\gamma\left(2-\gamma\right)}c^n & (80) \\ G\phi^{n+1} = -\left(\rho_o + \displaystyle\sum^{2} zec^{n+1}\right) & (81) \end{cases}$$

### 1.7. Discretization of boundary conditions

$$c_i^o = \frac{c_{bulk}}{|z|} \qquad \phi_i^o = 0 \qquad ; \qquad \text{for} \quad 1 \leq i \leq L_{x-1} \qquad (82)$$

$$\phi_o^n = \phi_{wall} \qquad D\left(\frac{c_1^n - c_o^n}{\Delta x} + \frac{ze}{k_BT}c_o^n\frac{\left(\phi_1^n - \phi_o^n\right)}{\Delta x}\right) = 0 \qquad (83)$$

$$\Rightarrow \qquad c_o^n = \frac{c_1}{1 - \frac{ze}{k_B T}(\phi_1 - \phi_o)} \tag{84}$$

$$\phi_{L_x}^n = 0 \qquad ; \qquad c_{L_x}^n = \frac{c_{bulk}}{|z|} \text{ for n} \geq 0 \tag{85}$$

**Numerical solver**

The TR-BDF2 solver along the boundary and initial conditions, yields the following system:

$$\text{TR-Step} \quad \left\{ \begin{array}{ll} \left(I - \frac{\gamma \Delta t}{2} F^{n+\gamma}\right) c^{n+\gamma} = \left(I + \frac{\gamma \Delta t}{2} F^n\right) c^n & (86) \\ G\phi^{n+\gamma} = -\left(\rho_o + \sum zec^{n+\gamma}\right) & (87) \end{array} \right.$$

The system is solved iteratively using a Newton-Raphson method where k denotes the Newton iteration:

$c^{n+\gamma,k+1} = c^{n+\gamma,k} + \delta c^{n+\gamma,k}$ With $c^{n+\gamma,o} = c^n$

$$\Rightarrow \qquad (I - \frac{\gamma \Delta t}{2} (\frac{\delta F}{\delta c})^{n+\gamma,k}) = -(c^{n+\gamma} - c^n) + \frac{\gamma \Delta t}{2} (F^{n+\gamma,k} + F^n) \tag{88}$$

where $R_{TR}$ is the residual and

$$F^{n+\gamma,k+1} = F^{n+\gamma,k} + \left(\frac{\partial F}{\partial c}\right)^{n+\gamma,k} \delta c^{n+\gamma,k} \tag{89}$$

The potential is then updated $G\phi^{n+\gamma,k} = -\left(\rho_o + \sum zec^{n+\gamma,k}\right) = -R_{TR}$ (90)

Repeat until convergence.

BDF2 – step

$$\begin{cases} \left(I - \frac{1-\gamma}{2-\gamma}\Delta t F^{n+1}\right) c^{n+1} = \frac{1}{\gamma(2-\gamma)} c^{n+\gamma} - \frac{(1-\gamma)^2}{\gamma(2-\gamma)} c^n & (91) \\ G\phi^{n+1} = -\left(\rho_o + \sum zec^{n+1}\right) & (92) \end{cases}$$

Same as the TR – step:

$c^{n+1,k+1} = c^{n+1,k} + \delta c^{n+1,k}$ with $c^{n+1,k+1} = c^{n+1,k} + \delta c^{n+1,k}$

$$\Rightarrow \quad \left(I - \frac{1-\gamma}{2-\gamma}\Delta t \left(\frac{\partial F}{\partial c}\right)^{n+1,k}\right) \delta c^{n+1,k} =$$

$$-\left(c^{n+1,k} - \frac{1}{\gamma(2-\gamma)} c^{n+\gamma} + \frac{(1-\gamma)^2}{\gamma(2-\gamma)} c^n\right) + \frac{1-\gamma}{2-\gamma}\Delta t F^{n+1} = -R_{BDF2} \qquad (93)$$

where $R_{BDF2}$ is the residual and $F^{n+1,k+1} = F^{n+1,k} + \left(\frac{\partial F}{\partial c}\right)^{n+1,k} \delta c^{n+1,k}$ (94)

The potential is then updated $G\phi^{n+1,k} = -\left(\rho_o + \sum zec^{n+1,k}\right)$ (95)

## 2. Results and Validation

The code developed here has been tested for several cases and compared with the COMSOL software for the same cases. The results of the comparisons were in agreement. In what follows some results are given as examples.

### 2.1. Example 1 : Sodium Chloride (NaCl)

- **parameters**:

- Physics parameters

| | | |
|---|---|---|
| Length of computational domain: | 0.5 | in nm |
| Temperature: | 298 | in K |
| Positive ion diffusivity: | 1.334e+009 | in nm^2/s |
| Negative ion diffusivity: | 2.032e+009 | in nm^2/s |
| Positive ion valence: | 1 | |
| Negative ion valence: | -1 | |

- Simulation parameters

| | | |
|---|---|---|
| dx: | 0.000607952 | in nm |
| dt: | 1.84803e-007 | in s |
| Number of grid points: | 822 | |

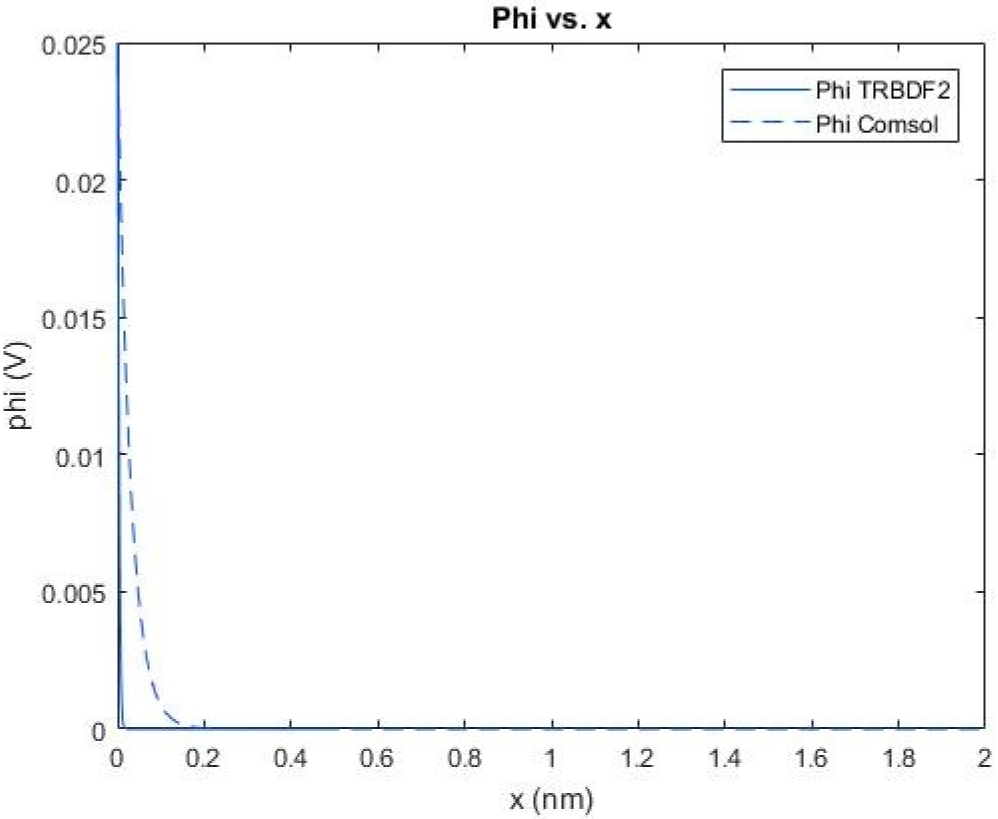
- Boundary & Initial conditions

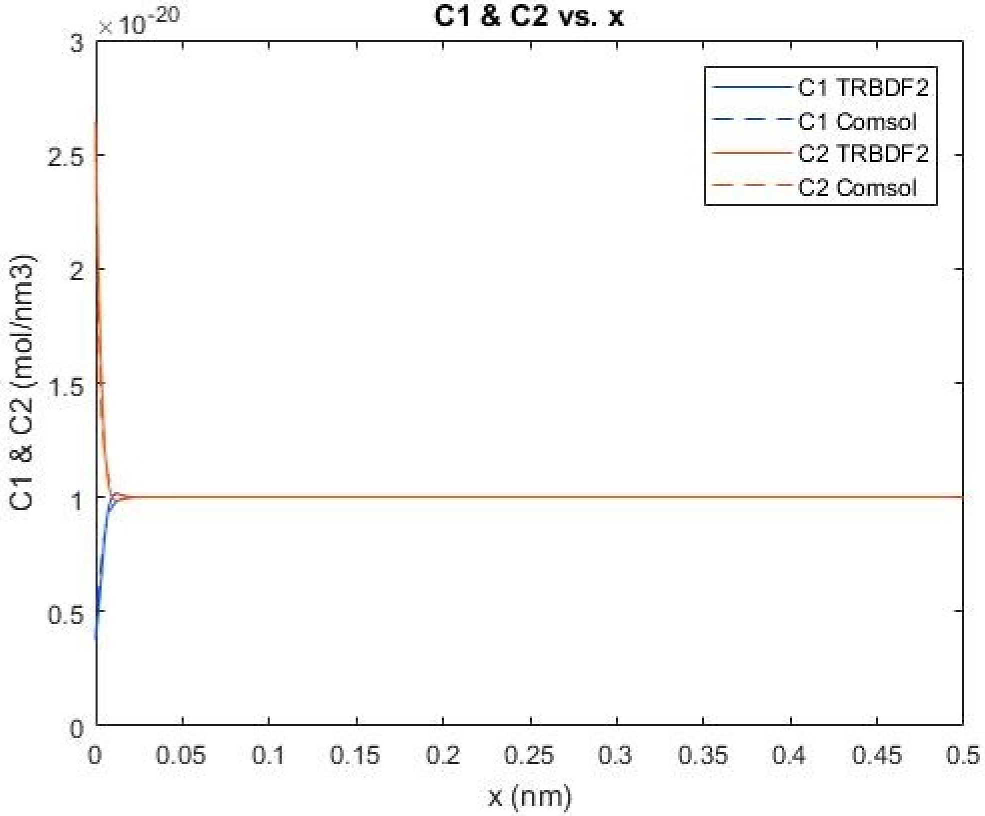
| | | |
|---|---|---|
| Potential at wall: | 0.025 | in V |
| Average concentration at t=0: | 1e-020 | in mol/(nm^3) |

- Derived parameters

| | | |
|---|---|---|
| Debye length: | 0.00303976 | in nm |
| Ionic bulk strength: | 1e+007 | in mol/m3 |

Thermal voltage: 0.0256794 in V





2.2. Example 2 : Sodium Sulfate (Na$_2$So$_4$)

- **Parameters**

- Physics parameters

| | | |
|---|---|---|
| Length of computational domain: | 0.5 | in nm |
| Temperature: | 298 | in K |
| Positive ion diffusivity: | 1.334e+009 | in nm^2/s |
| Negative ion diffusivity: | 1.065e+009 | in nm^2/s |
| Positive ion valence: | 1 | |
| Negative ion valence: | -2 | |

- Simulation parameters

| | | |
|---|---|---|
| dx: | 0.000303976 | in nm |

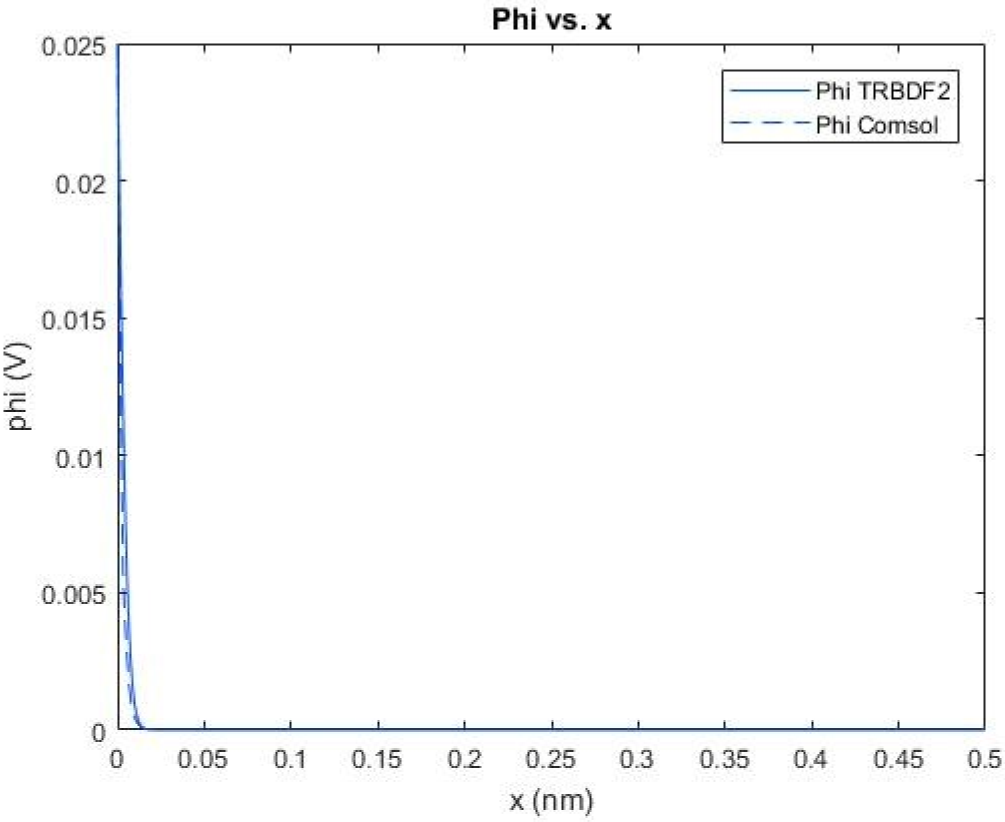
| | |
|---|---|
| dt: | 4.62007e-008 in s |
| Number of grid points: | 1644 |

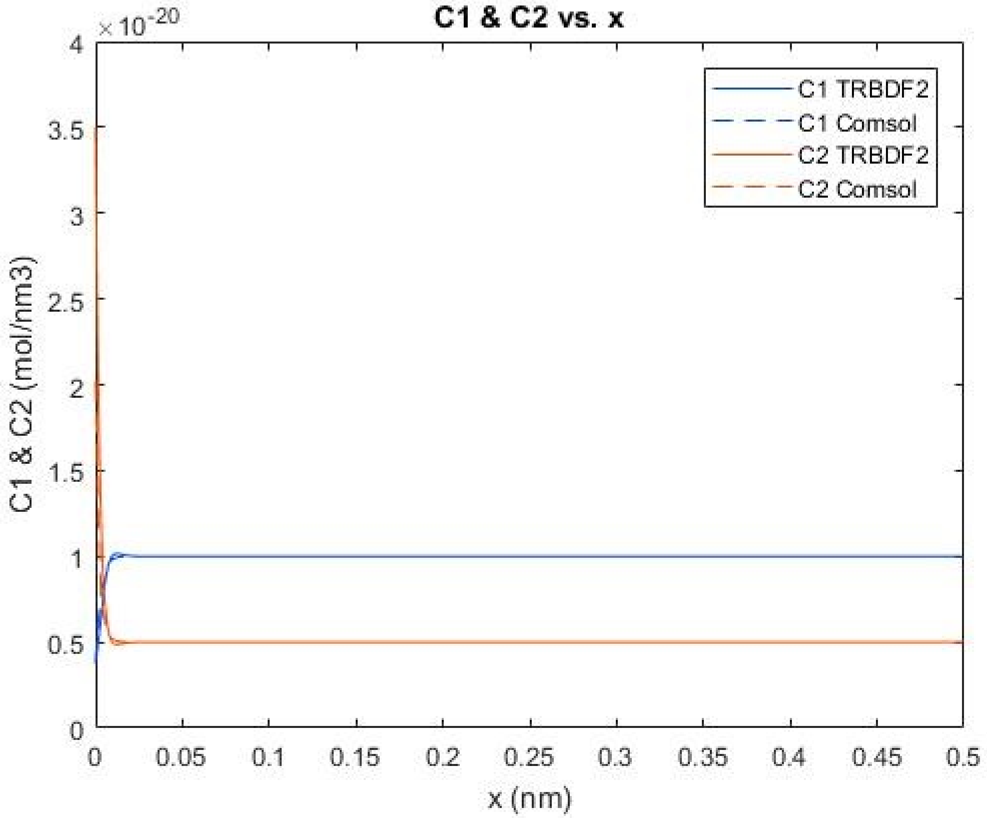
- Boundary & Initial conditions

| | |
|---|---|
| Potential at wall: | 0.025 in V |
| Average concentration at t=0: | 1e-020 in mol/(nm^3) |

- Derived parameters--

| | |
|---|---|
| Debye length: | 0.00151988 in nm |
| Ionic bulk strength: | 4e+007 in mol/m3 |
| Thermal voltage: | 0.0256794 in V |





2.3. Example 3 : Trisodium Phosphate ($Na_3PO_4$)

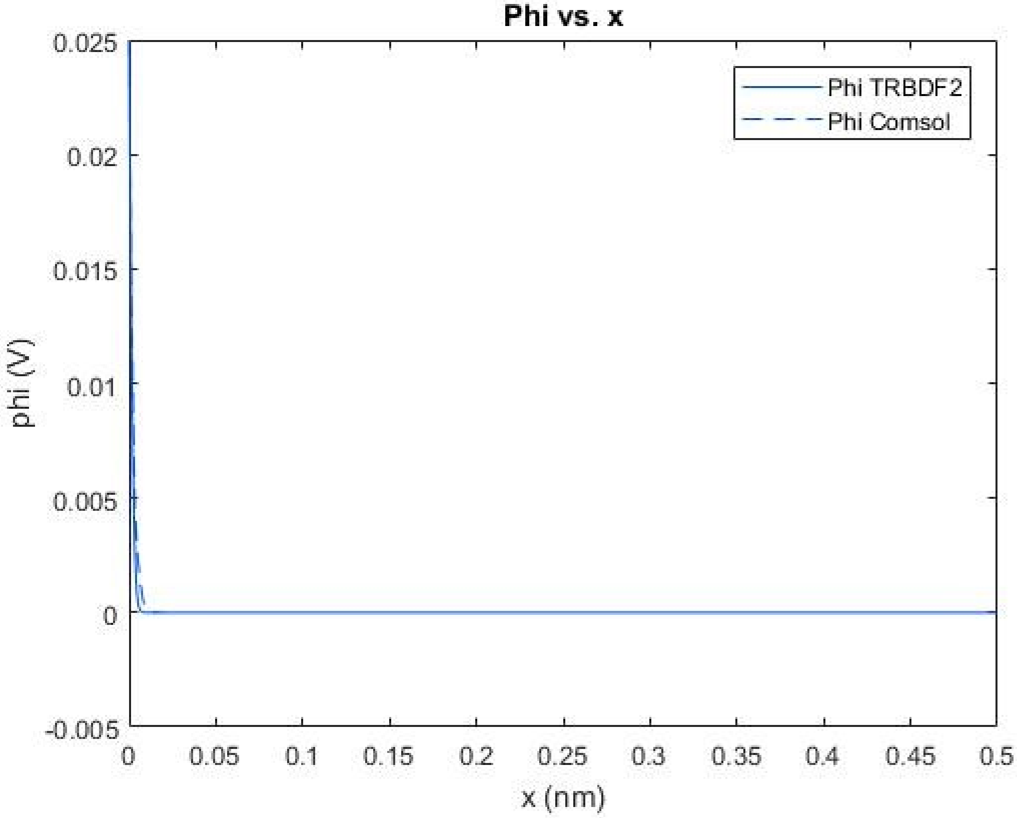
- **Parameters**
  - Physics parameters--

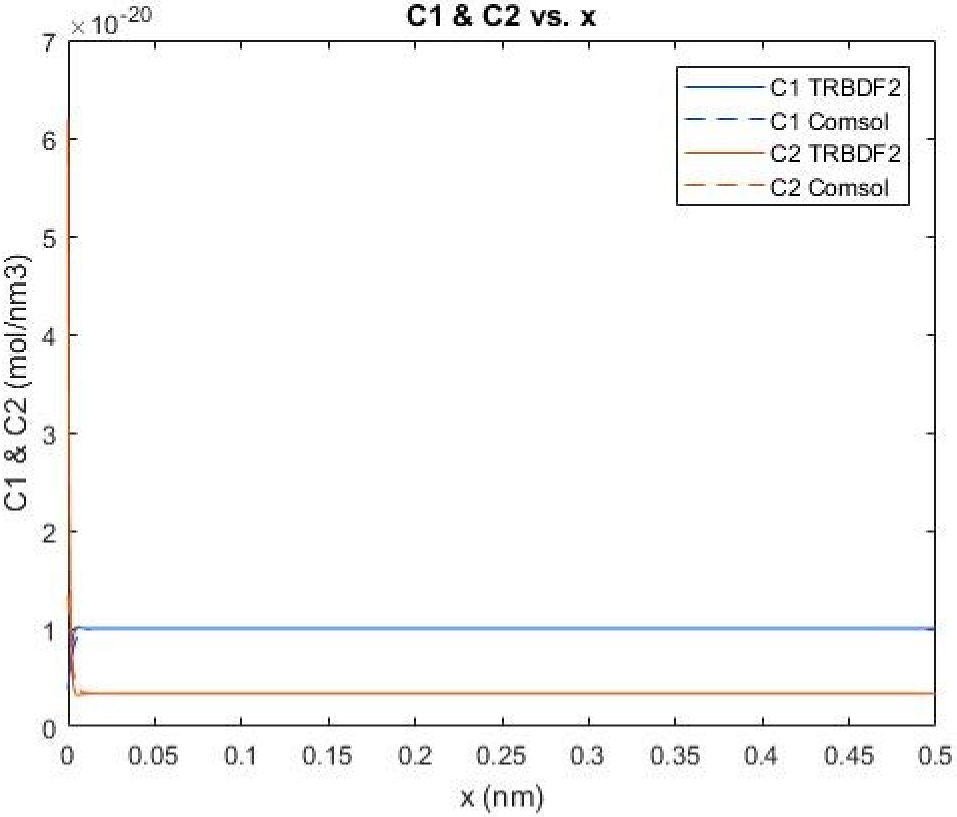
| | | |
|---|---|---|
| Length of computational domain: | 0.5 | in nm |
| Temperature: | 298 | in K |
| Positive ion diffusivity: | 1.334e+009 | in nm^2/s |
| Negative ion diffusivity: | 6.1e+008 | in nm^2/s |
| Positive ion valence: | 1 | |
| Negative ion valence: | -3 | |

  - Simulation parameters

| | | |
|---|---|---|
| dx: | 0.000271885 | in nm |

| | | | |
|---|---|---|---|
| | dt: | 3.69606e-008 | in s |
| | Number of grid points: | 1839 | |
| - | Boundary & Initial condition | | |
| | Potential at wall: | 0.025 | in V |
| | Average concentration at t=0: | 1e-020 | in mol/(nm^3) |
| - | Derived parameters-- | | |
| | Debye length: | 0.00135942 | in nm |
| | Ionic bulk strength: | 5e+007 | in mol/m3 |
| | Thermal voltage: | 0.0256794 | in V |




C1 & C2 vs. x
×10^-20
7
6
5
4
3
2
1
0
C1 & C2 (mol/nm3)
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
x (nm)
C1 TRBDF2
C1 Comsol
C2 TRBDF2
C2 Comsol

# CHAPTER VI

## GENERAL CONCLUSION AND PERSPECTIVES

The main objective of this thesis was to develop a nanofluid simulation platform. Challenged with the scarcity of computer codes dealing with this issue, apart from commercial software, and after several phases, this research work has led to the development of an innovative code.

In a first step, the principles of modeling, discretization and numerical simulations were reviewed. Then a study of nanofluids and electro-kinetic phenomena was carried out. To develop the simulation platform, the C ++ language was chosen, which also led to the research and the study of several computer development techniques. Finally, a code was developed and tested  and compared with the COMSOL industrial code.

The prospects are great and can relate either to the ergonomic side of the code or the fund. Indeed, the code can be improved ergonomically and include a graphical and contextual interface. The code can also easily be modified to include different methods such as the finite volume method or others.

In conclusion, this thesis has allowed us to realize that the goal of a research work is not necessarily to give concrete answers, but to try to contribute, even if in a modest way, to the current issues.

# APPENDIX A

# CODE

## Main C++ code

```
 1  #include <iomanip>
 2  #include <iostream>
 3  #include <cmath>
 4  #include <fstream>
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  #include "memory_alloc.h"
 9  #include "numerical_solver.h"
10  /// NOTE FOR USER #0: Notes will be scattered throughout the code, to find them use
CTRL+F and type in "NOTE FOR USER #X" with X being the one you are looking for.
11  #define EPSILON     0.01        // relative error
12  /// NOTE FOR USER #1: EPSILON can be changed to reflect desired accuracy, however due
to working with small numbers, it is not recommended to go below too low.
13  #define GAMMA       (2.-sqrt(2.))       // for TRBDF2
14  //#define GAMMA     1                   // Crank-Nicolson
15  /// NOTE FOR USER #2: Using GAMMA=1 makes this algorithm perform Crank-Nicolson
scheme, replace line 13 by line 14 to do so.
16
17  using namespace std;
18
19
20  // fixed constants
21
22  const double e=1.6022e-19; // in Coulomb
23  const double NA=6.0221e23; // in 1/mol
24  const double F=e*NA;
25  const double kb=1.3806e-23; // in J/K
26  const double Eps_r=78.5;
27  const double Eps_0=8.8454187817e-21; // in F/nm
28  const double Eps_0m=8.8454187817e-12; // in F/m
29  const double R=8.3144598; // J.K-1.mol-1
30
31  // dimensional parameters
32
33  /// x, t, C1 & C2, Phi
34  /// nm, s, mol/nm3, V
35
36  /// NOTE FOR USER #3: Below is the input data for the experiment to be simulated.
Some combinations may lead to obviously untrue results (negative concentrations, unstable
resolution).
37  ///                  I personally have yet to encounter such cases when reproducing
real-life experiments but know that it may happen.
38  const double Phi_wall=0.025; // in V
39  const double C_bulk=1e-20; // in mol/nm3
40  const double D1=1.334e9, D2=2.032e9; // in nm2/s
41  const double T=298.0; // in K
42  const double Eps=Eps_r*Eps_0; // in F/m
43  const double rho0=0;
44
45  const double z1=1, z2=-1;
46
47  // derived parameters
48  const double V_therm=R*T/F; // thermal voltage
49  const double Istr_bulk=0.5*(sq(z1)+sq(z2))*C_bulk*1e27; // bulk ionic strength
50
51  const double debye_length=sqrt((Eps_r*Eps_0m*V_therm)/(2*F*Istr_bulk))*1e9; // in nm
52  /// NOTE FOR USER #4: This is calculated and shown in the Info.txt output file. This
serves mostly to check if things don't go wildly wrong.
53  ///                  The drop in potential to 0 should occur in 10~30 debye lengths
according to literature.
54  ///                  Though this is not set in stone, it just serves to give an idea
```

```
about the correct scale of the problem.

const double L=0.5; // in nm

const int Lx=300;
const double dx=L/Lx;
const double dt=0.5*dx*dx;

const int max_tsteps=1000;
const int MAX_NEWTONS=100;
/// NOTE FOR USER #5: Length of the domain and computational parameters. It is possible to change any of these (except dx, unless you make sure it fits with L and Lx).
///                   It is possible to pick a larger or smaller dt. The given value is the one I would recommend as it fits stability requirements quite well.
///                   If you are blessed with patience, you may change max_tsteps and MAX_NEWTONS to be as large as possible. Personally, I just prefer cases that converge slowly to be stopped early.



/// Functions
// Memory allocation

POINTER Alloc(unsigned N_bytes)
{
    POINTER p;

    p = malloc(N_bytes);

    if (p == (POINTER) NULL) {
        perror("ERROR in Alloc(): malloc() returned NULL pointer");
        exit(ERROR);
    }
    return p;
}

POINTER alloc_vector(int N, unsigned element_size)
{
    return Alloc((unsigned) (N*element_size));
}

POINTER alloc_matrix(int N_rows, int N_columns, unsigned element_size)
{
    int i, space, N_pointers;
    POINTER pA, array_origin;

        // due to alignment requirement
    N_pointers = (N_rows%2) ? N_rows+1 : N_rows;

    space = N_pointers*sizeof(POINTER) + N_rows*N_columns*element_size;
    pA = Alloc((unsigned) space);

    array_origin = ((char *) (pA)) + N_pointers*sizeof(POINTER);

    for (i = 0; i < N_rows; i++)
        ((POINTER *) pA)[i] = ((char *) array_origin) +
            i*N_columns*element_size;

    return pA;
}

// simulation

void init_grid(GRID *grid, TIME *time)
{
    grid->xmin=0;
    grid->xmax=L;
    grid->dx=dx;
    grid->Lx=Lx;
    grid->first=1;
    grid->last=grid->Lx-1;
    grid->modes=1+grid->last-grid->first;

    time->t=0;
    time->dt=dt;
    time->n=0;
}

double compute_norm(GRID *grid, double x[])
{
    double norm=0.;
    for (int i=grid->first; i<=grid->last; i++)
    {
        norm=norm+fabs(x[i]);
    }
    norm=norm/grid->modes;
    return norm;
}

void thomas_algorithm_solver(GRID *grid, double X[], double *A[], double b[])
{
    double cprime[grid->last], dprime[grid->last];

    cprime[grid->first]=A[grid->first][grid->first+1]/A[grid->first][grid->first];
    for (int i=grid->first+1; i<=grid->last; i++)
    {
        cprime[i]=A[i][i+1]/(A[i][i]-A[i][i-1]-cprime[i-1]);
    }
    dprime[grid->first]=b[grid->first]/A[grid->first][grid->first];
    for (int i=grid->first+1; i<=grid->last; i++)
    {
        dprime[i]=(b[i]-A[i][i-1]*dprime[i-1])/(A[i][i]-A[i][i-1]*cprime[i-1]);
    }
    X[grid->last]=dprime[grid->last];
    for (int i=grid->last-1; i>=grid->first; i--)
    {
        X[i]=dprime[i]-cprime[i]*X[i+1];
    }
}

static double *C1_n;
static double *C2_n;
static double *Phi_n;

static double *C1_npG;
static double *C2_npG;
static double *Phi_npG;

static double *C1_np1;
static double *C2_np1;
static double *Phi_np1;

static double *f1_n;
static double *f1_npG;
static double *f1_np1;
static double *f2_n;
static double *f2_npG;
static double *f2_np1;

static double **F1;
static double **F2;
static double **J;
static double *b;
static double *X;

int main()
{
    GRID grid;
    TIME time;
    init_grid(&grid, &time);
    int Lx=grid.Lx;

    // Output of simulation data
    ofstream Info("./Info.txt", ofstream::out);
    Info << "---INFO SECTION---" <<  endl;

    Info << "--Physics parameters--" << endl;
    Info << "Length of computational domain:\t\t" << L << "\tin nm" << endl;
    Info << "Temperature:\t\t" << T << "\tin K" << endl;
    Info << "Positive ion diffusivity:\t\t" << D1 << "\tin nm^2/s" << endl;
    Info << "Negative ion diffusivity:\t\t" << D2 << "\tin nm^2/s" << endl;
    Info << "Positive ion valence:\t\t" << z1 << endl;
    Info << "Negative ion valence:\t\t" << z2 << endl;

    Info << "--Simulation parameters--" << endl;
    Info << "dx:\t\t\t" << dx << "\tin nm" << endl;
    Info << "dt:\t\t\t" << dt << "\tin s" << endl;
    Info << "Number of grid points:\t\t" << grid.Lx << endl;

    Info << "--BC & IC--" << endl;
    Info << "Potential at wall:\t\t" << Phi_wall << "\tin V" << endl;
    Info << "Average concentration at t=0:\t\t" << C_bulk << "\tin mol/(nm^3)" <<
endl;

    Info << "--Derived parameters--" << endl;

    Info << "Debye length:\t\t" << debye_length << "\tin nm" << endl;
    Info << "Ionic bulk strength:\t\t" << Istr_bulk << "\t in mol/m3" << endl;
    Info << "Thermal voltage:\t\t" << V_therm << "\t in V" << endl;

    Info << "---END---" << endl;
    Info.close();


    C1_n=(double*)alloc_vector(grid.Lx,DOUBLE);
    C1_npG=(double*)alloc_vector(grid.Lx,DOUBLE);
    C1_np1=(double*)alloc_vector(grid.Lx,DOUBLE);

    C2_n=(double*)alloc_vector(grid.Lx,DOUBLE);
    C2_npG=(double*)alloc_vector(grid.Lx,DOUBLE);
    C2_np1=(double*)alloc_vector(grid.Lx,DOUBLE);

    Phi_n=(double*)alloc_vector(grid.Lx,DOUBLE);
    Phi_npG=(double*)alloc_vector(grid.Lx,DOUBLE);
    Phi_np1=(double*)alloc_vector(grid.Lx,DOUBLE);

    f1_n=(double*)alloc_vector(grid.Lx,DOUBLE);
    f1_npG=(double*)alloc_vector(grid.Lx,DOUBLE);
    f1_np1=(double*)alloc_vector(grid.Lx,DOUBLE);
    f2_n=(double*)alloc_vector(grid.Lx,DOUBLE);
    f2_npG=(double*)alloc_vector(grid.Lx,DOUBLE);
    f2_np1=(double*)alloc_vector(grid.Lx,DOUBLE);

    F1=(double**)alloc_matrix(grid.Lx,grid.Lx,DOUBLE);
    F2=(double**)alloc_matrix(grid.Lx,grid.Lx,DOUBLE);
    J=(double**)alloc_matrix(grid.Lx,grid.Lx,DOUBLE);
    b=(double*)alloc_vector(grid.Lx,DOUBLE);
    X=(double*)alloc_vector(grid.Lx,DOUBLE);

    /// Initial conditions
    for (int i=grid.first-1; i<=grid.last+1; i++)
    {
        Phi_n[i]=0;
        C1_n[i]=C_bulk/fabs(z1);
        C2_n[i]=C_bulk/fabs(z2);
    }

```

```
256      double phantom_phi_left_n, phantom_phi_right_n;
257      double phantom_phi_left_npG, phantom_phi_right_npG;
258      double phantom_phi_left_np1, phantom_phi_right_np1;
259
260      ofstream numerical_solver("./numerical_solver.txt", ofstream::out);
261      numerical_solver << "Time: 0" << endl;
262      numerical_solver << "Phi:" << endl;
263      for (int i=grid.first-1; i<=grid.last+1; i++)
264      {
265          numerical_solver << Phi_n[i] << " ";
266      }
267      numerical_solver << endl << "C1:" << endl;
268      for (int i=grid.first-1; i<=grid.last+1; i++)
269      {
270          numerical_solver << C1_n[i] << " ";
271      }
272      numerical_solver << endl << "C2:" << endl;
273      for (int i=grid.first-1; i<=grid.last+1; i++)
274      {
275          numerical_solver << C2_n[i] << " ";
276      }
277      cout << "Time: 0" << endl << endl;
278
279      // time loop
280      double norm_change_C1, norm_residual_C1;
281      double norm_change_C2, norm_residual_C2;
282      double norm_change_Phi, norm_residual_Phi;
283      double difference_between_tsteps_C1=1., difference_between_tsteps_C2=1.,
difference_between_tsteps_Phi=1.;
284
285      for (int n=1; n<=max_tsteps; n++)
286      {
287          time.n=n;
288          time.t=time.n*dt;
289          /// TR-step
290          norm_change_C1 = 1., norm_residual_C1 = 1.;
291          norm_change_C2 = 1., norm_residual_C2 = 1.;
292          norm_change_Phi = 1., norm_residual_Phi = 1.;
293          /// Boundary conditions at n+GAMMA
294          phantom_phi_left_n=2*Phi_n[0]-Phi_n[1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_n[0]+z2*e*C2_n[0]);
295          phantom_phi_right_n=0;
296          //phantom_phi_right_n=2*Phi_n[Lx]-Phi_n[Lx-1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_n[Lx]+z2*e*C2_n[Lx]);
297          /// NOTE FOR USER #6: If you plan on making the problem a wall at the Lx end.
Replace line 295 with line 296.
298          for (int i=grid.first-1; i<=grid.last+1; i++)
299          {
300              C1_npG[i]=C1_n[i];
301              C2_npG[i]=C2_n[i];
302              Phi_npG[i]=Phi_n[i];
303          }
304          phantom_phi_left_npG=phantom_phi_left_n;
305          phantom_phi_right_npG=phantom_phi_right_n;
306
307          cout << "TR-step" << endl;
308          F1[1][0]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_n[1]-phantom_phi_left_n));
309          F1[1][1]=-2*D1/sq(dx);
310          F1[1][2]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(Phi_n[3]-Phi_n[1]));
311          for (int i=grid.first+1; i<=grid.last-1; i++)
312          {
313              F1[i][i-1]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_n[i]-Phi_n[i-2]));
314              F1[i][i]=-2*D1/sq(dx);
315              F1[i][i+1]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(Phi_n[i+2]-Phi_n[i]));
316          }
317          F1[Lx-1][Lx-2]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_n[Lx-1]-Phi_n[Lx-3]));
318          F1[Lx-1][Lx-1]=-2*D1/sq(dx);
319          F1[Lx-1][Lx]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(phantom_phi_right_n-
Phi_n[Lx-1]));
```

```
        for (int i=grid.first; i<=grid.last; i++) // f=F*c
        {
            f1_n[i]=F1[i][i-1]*C1_n[i-1]+F1[i][i]*C1_n[i]+F1[i][i+1]*C1_n[i+1];
        }
        F2[1][0]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_n[1]-phantom_phi_left_n));
        F2[1][1]=-2*D2/sq(dx);
        F2[1][2]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(Phi_n[3]-Phi_n[1]));
        for (int i=grid.first+1; i<=grid.last-1; i++)
        {
            F2[i][i-1]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_n[i]-Phi_n[i-2]));
            F2[i][i]=-2*D2/sq(dx);
            F2[i][i+1]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(Phi_n[i+2]-Phi_n[i]));
        }
        F2[Lx-1][Lx-2]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_n[Lx-1]-Phi_n[Lx-3]));
        F2[Lx-1][Lx-1]=-2*D2/sq(dx);
        F2[Lx-1][Lx]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(phantom_phi_right_n-Phi_n[Lx-1]));
        for (int i=grid.first; i<=grid.last; i++) // f=F*c
        {
            f2_n[i]=F2[i][i-1]*C2_n[i-1]+F2[i][i]*C2_n[i]+F2[i][i+1]*C2_n[i+1];
        }
        // Newton iteration loop
        int k;
        for (k=1; k<=MAX_NEWTONS; k++)
        {
            cout << "Newton iteration: " << k << endl;
            /// C1 at n+GAMMA
            // evaluate jacobian
            F1[1][0]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_npG[1]-phantom_phi_left_npG));
            F1[1][1]=-2*D1/sq(dx);
            F1[1][2]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(Phi_npG[3]-Phi_npG[1]));
            for (int i=grid.first+1; i<=grid.last-1; i++)
            {
                F1[i][i-1]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_npG[i]-Phi_npG[i-2]));
                F1[i][i]=-2*D1/sq(dx);
                F1[i][i+1]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(Phi_npG[i+2]-Phi_npG[i]));
            }
            F1[Lx-1][Lx-2]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_npG[Lx-1]-Phi_npG[Lx-3]));
            F1[Lx-1][Lx-1]=-2*D1/sq(dx);
            F1[Lx-1][Lx]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(phantom_phi_right_npG-Phi_npG[Lx-1]));
            for (int i=grid.first; i<=grid.last; i++)
            {
                f1_npG[i]=F1[i][i-1]*C1_npG[i-1]+F1[i][i]*C1_npG[i]+F1[i][i+1]*C1_npG[i+1];
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                J[i][i-1]=-0.5*GAMMA*dt*F1[i][i-1];
                J[i][i]=1.-0.5*GAMMA*dt*F1[i][i];
                J[i][i+1]=-0.5*GAMMA*dt*F1[i][i+1];
            }
            // evaluate residual
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=-(C1_npG[i]-C1_n[i])+0.5*GAMMA*dt*(f1_npG[i]+f1_n[i]);
            }
            norm_residual_C1=compute_norm(&grid, b);
            cout << "norm_residual_C1 in newton=" << norm_residual_C1 << endl;
            thomas_algorithm_solver(&grid, X, J, b);
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=X[i];
            }
            // check for convergence
            norm_change_C1=compute_norm(&grid, b);
            cout << "norm_change_C1 in newton=" << norm_change_C1 << endl;
            for (int i=grid.first; i<=grid.last; i++)
            {
                C1_npG[i]=C1_npG[i]+b[i];
            }
            /// C2 at n+GAMMA
            // evaluate jacobian
            F2[1][0]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_npG[1]-phantom_phi_left_npG));
            F2[1][1]=-2*D2/sq(dx);
            F2[1][2]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(Phi_npG[3]-Phi_npG[1]));
            for (int i=grid.first+1; i<=grid.last-1; i++)
            {
                F2[i][i-1]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_npG[i]-Phi_npG[i-2]));
                F2[i][i]=-2*D2/sq(dx);
                F2[i][i+1]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(Phi_npG[i+2]-Phi_npG[i]));
            }
            F2[Lx-1][Lx-2]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_npG[Lx-1]-Phi_npG[Lx-3]));
            F2[Lx-1][Lx-1]=-2*D2/sq(dx);
            F2[Lx-1][Lx]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(phantom_phi_right_npG-Phi_npG[Lx-1]));
            for (int i=grid.first; i<=grid.last; i++)
            {
                f2_npG[i]=F2[i][i-1]*C2_npG[i-1]+F2[i][i]*C2_npG[i]+F2[i][i+1]*C2_npG[i+1];
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                J[i][i-1]=-0.5*GAMMA*dt*F2[i][i-1];
                J[i][i]=1.-0.5*GAMMA*dt*F2[i][i];
                J[i][i+1]=-0.5*GAMMA*dt*F2[i][i+1];
            }
            // evaluate residual
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=-(C2_npG[i]-C2_n[i])+0.5*GAMMA*dt*(f2_npG[i]+f2_n[i]);
            }
            norm_residual_C2=compute_norm(&grid, b);
            cout << "norm_residual_C2 in newton=" << norm_residual_C2 << endl;
            thomas_algorithm_solver(&grid, X, J, b);
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=X[i];
            }
            // check for convergence
            norm_change_C2=compute_norm(&grid, b);
            cout << "norm_change_C2 in newton=" << norm_change_C2 << endl;
            for (int i=grid.first; i<=grid.last; i++)
            {
                C2_npG[i]=C2_npG[i]+b[i];
            }
            /// Phi at n+GAMMA
            // evaluate Jacobian
            for (int i=grid.first; i<=grid.last; i++)
            {
                J[i][i-1]=Eps/sq(dx);
                J[i][i]=-2*Eps/sq(dx);
                J[i][i+1]=Eps/sq(dx);
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=-(J[i][i-1]*Phi_npG[i-1]+J[i][i]*Phi_npG[i]+J[i][i+1]*Phi_npG[i+1])-(rho0+z1*e*C1_npG[i]+z2*e*C2_npG[i]);
            }
            norm_residual_Phi=compute_norm(&grid, b);
```

```
444              cout << "norm_residual_Phi in newton=" << norm_residual_Phi << endl;
445              thomas_algorithm_solver(&grid, X, J, b);
446              for (int i=grid.first; i<=grid.last; i++)
447              {
448                  b[i]=X[i];
449              }
450              // check for convergence
451              norm_change_Phi=compute_norm(&grid, b);
452              cout << "norm_change_Phi in newton=" << norm_change_Phi << endl;
453              for (int i=grid.first; i<=grid.last; i++)
454              {
455                  Phi_npG[i]=Phi_npG[i]+b[i];
456              }
457              /// BC n+GAMMA
458              Phi_npG[0]=Phi_wall;
459              C1_npG[0]=C1_npG[1]/(1.-(z1*e/(kb*T))*(Phi_npG[1]-Phi_npG[0]));
460              C2_npG[0]=C2_npG[1]/(1.-(z2*e/(kb*T))*(Phi_npG[1]-Phi_npG[0]));
461
462              Phi_npG[Lx]=0;
463              C1_npG[Lx]=C_bulk/fabs(z1);
464              C2_npG[Lx]=C_bulk/fabs(z2);
465
466              /*Phi_npG[Lx]=-Phi_wall;
467              C1_npG[Lx]=C1_npG[Lx-1]/(1.+(z1*e/(kb*T))*(Phi_npG[Lx]-Phi_npG[Lx-1]));
468              C2_npG[Lx]=C2_npG[Lx-1]/(1.+(z2*e/(kb*T))*(Phi_npG[Lx]-Phi_npG[Lx-1]));*/
469              /// NOTE FOR USER #7: Refer to #6. Replace lines 462 to 464 with lines
466 to 468.
470
471              phantom_phi_left_npG=2*Phi_npG[0]-Phi_npG[1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_npG[0]+z2*e*C2_npG[0]);
472              phantom_phi_right_npG=0;
473              //phantom_phi_right_npG=2*Phi_npG[Lx]-Phi_npG[Lx-1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_npG[Lx]+z2*e*C2_npG[Lx]);
474              /// NOTE FOR USER #8: Refer to #6. Replace line 472 with line 473.
475              norm_change_Phi=norm_change_Phi/compute_norm(&grid,Phi_npG);
476              norm_change_C1=norm_change_C1/compute_norm(&grid,C1_npG);
477              norm_change_C2=norm_change_C2/compute_norm(&grid,C2_npG);
478              if ((norm_change_Phi<EPSILON) && (norm_change_C1<EPSILON) &&
(norm_change_C2<EPSILON))
479              {
480                  break;
481              }
482          }
483          if (k>MAX_NEWTONS)
484          {
485              cout << "ERROR: Newton method failed to converge after " << MAX_NEWTONS
<< " iterations" << endl;
486              exit(ERROR);
487          }
488
489          /// BDF2-step
490          norm_change_C1 = 1., norm_residual_C1 = 1.;
491          norm_change_C2 = 1., norm_residual_C2 = 1.;
492          norm_change_Phi = 1., norm_residual_Phi = 1.;
493          /// Boundary conditions at n+1
494          phantom_phi_left_npG=2*Phi_npG[0]-Phi_npG[1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_npG[0]+z2*e*C2_npG[0]);
495          phantom_phi_right_npG=0;
496          //phantom_phi_right_npG=2*Phi_npG[Lx]-Phi_npG[Lx-1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_npG[Lx]+z2*e*C2_npG[Lx]);
497          /// NOTE FOR USER #9: Refer to #6. Replace line 495 with line 496.
498          for (int i=grid.first-1; i<=grid.last+1; i++)
499          {
500              C1_np1[i]=C1_npG[i];
501              C2_np1[i]=C2_npG[i];
502              Phi_np1[i]=Phi_npG[i];
503          }
504          phantom_phi_left_np1=phantom_phi_left_npG;
505          phantom_phi_right_np1=phantom_phi_right_npG;
```

```

        cout << "BDF2-step" << endl;
        // Newton iteration loop
        for (k=1; k<=MAX_NEWTONS; k++)
        {
            cout << "Newton iteration: " << k << endl;
            /// C1 at n+1
            // evaluate jacobian
            F1[1][0]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_np1[1]-phantom_phi_left_np1));
            F1[1][1]=-2*D1/sq(dx);
            F1[1][2]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(Phi_np1[3]-Phi_np1[1]));
            for (int i=grid.first+1; i<=grid.last-1; i++)
            {
                F1[i][i-1]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_np1[i]-Phi_np1[i-2]));
                F1[i][i]=-2*D1/sq(dx);
                F1[i][i+1]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(Phi_np1[i+2]-Phi_np1[i]));
            }
            F1[Lx-1][Lx-2]=(D1/sq(dx))*(1.-0.25*(z1*e/(kb*T))*(Phi_np1[Lx-1]-Phi_np1[Lx-3]));
            F1[Lx-1][Lx-1]=-2*D1/sq(dx);
            F1[Lx-1][Lx]=(D1/sq(dx))*(1.+0.25*(z1*e/(kb*T))*(phantom_phi_right_np1-Phi_np1[Lx-1]));
            for (int i=grid.first; i<=grid.last; i++)
            {
                f1_np1[i]=F1[i][i-1]*C1_np1[i-1]+F1[i][i]*C1_np1[i]+F1[i][i+1]*C1_np1[i+1];
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                J[i][i-1]=-((1-GAMMA)/(2-GAMMA))*dt*F1[i][i-1];
                J[i][i]=1-((1-GAMMA)/(2-GAMMA))*dt*F1[i][i];
                J[i][i+1]=-((1-GAMMA)/(2-GAMMA))*dt*F1[i][i+1];
            }
            // evaluate residual
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=-(C1_np1[i]-(1/(GAMMA*(2-GAMMA)))*C1_npG[i]+((sq(1-GAMMA))/(GAMMA*(2-GAMMA)))*C1_n[i])+((1-GAMMA)/(2-GAMMA))*dt*f1_np1[i];
            }
            norm_residual_C1=compute_norm(&grid, b);
            cout << "norm_residual_C1 in newton=" << norm_residual_C1 << endl;
            thomas_algorithm_solver(&grid, X, J, b);
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=X[i];
            }
            // check for convergence
            norm_change_C1=compute_norm(&grid, b);
            cout << "norm_change_C1 in newton=" << norm_change_C1 << endl;
            for (int i=grid.first; i<=grid.last; i++)
            {
                C1_np1[i]=C1_np1[i]+b[i];
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                C1_np1[i]=C1_np1[i];
            }
            /// C2 at n+1
            // evaluate jacobian
            F2[1][0]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_np1[1]-phantom_phi_left_np1));
            F2[1][1]=-2*D2/sq(dx);
            F2[1][2]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(Phi_np1[3]-Phi_np1[1]));
            for (int i=grid.first+1; i<=grid.last-1; i++)
            {
                F2[i][i-1]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_np1[i]-Phi_np1[i-
2]));
                F2[i][i]=-2*D2/sq(dx);
                F2[i][i+1]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(Phi_np1[i+2]-Phi_np1[i]));
            }
            F2[Lx-1][Lx-2]=(D2/sq(dx))*(1.-0.25*(z2*e/(kb*T))*(Phi_np1[Lx-1]-Phi_np1[Lx-3]));
            F2[Lx-1][Lx-1]=-2*D2/sq(dx);
            F2[Lx-1][Lx]=(D2/sq(dx))*(1.+0.25*(z2*e/(kb*T))*(phantom_phi_right_np1-Phi_np1[Lx-1]));
            for (int i=grid.first; i<=grid.last; i++)
            {
                f2_np1[i]=F2[i][i-1]*C2_np1[i-1]+F2[i][i]*C2_np1[i]+F2[i][i+1]*C2_np1[i+1];
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                J[i][i-1]=-((1-GAMMA)/(2-GAMMA))*dt*F2[i][i-1];
                J[i][i]=1-((1-GAMMA)/(2-GAMMA))*dt*F2[i][i];
                J[i][i+1]=-((1-GAMMA)/(2-GAMMA))*dt*F2[i][i+1];
            }
            // evaluate residual
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=-(C2_np1[i]-(1/(GAMMA*(2-GAMMA)))*C2_npG[i]+((sq(1-GAMMA))/(GAMMA*(2-GAMMA)))*C2_n[i])+((1-GAMMA)/(2-GAMMA))*dt*f2_np1[i];
            }
            norm_residual_C2=compute_norm(&grid, b);
            cout << "norm_residual_C2 in newton=" << norm_residual_C2 << endl;
            thomas_algorithm_solver(&grid, X, J, b);
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=X[i];
            }
            // check for convergence
            norm_change_C2=compute_norm(&grid, b);
            cout << "norm_change_C2 in newton=" << norm_change_C2 << endl;
            for (int i=grid.first; i<=grid.last; i++)
            {
                C2_np1[i]=C2_np1[i]+b[i];
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                C2_np1[i]=C2_np1[i];
            }
            /// Phi at n+1
            // evaluate Jacobian
            for (int i=grid.first; i<=grid.last; i++)
            {
                J[i][i-1]=Eps/sq(dx);
                J[i][i]=-2*Eps/sq(dx);
                J[i][i+1]=Eps/sq(dx);
            }
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=-(J[i][i-1]*Phi_np1[i-1]+J[i][i]*Phi_np1[i]+J[i][i+1]*Phi_np1[i+1])-(rho0+z1*e*C1_np1[i]+z2*e*C2_np1[i]);
            }
            norm_residual_Phi=compute_norm(&grid, b);
            cout << "norm_residual_Phi in newton=" << norm_residual_Phi << endl;
            thomas_algorithm_solver(&grid, X, J, b);
            for (int i=grid.first; i<=grid.last; i++)
            {
                b[i]=X[i];
            }
            // check for convergence
            norm_change_Phi=compute_norm(&grid, b);
            cout << "norm_change_Phi in newton=" << norm_change_Phi << endl;
            for (int i=grid.first; i<=grid.last; i++)
```

```
629       {
630         Phi_np1[i]=Phi_np1[i]+b[i];
631       }
632       /// BC n+1
633       Phi_np1[0]=Phi_wall;
634       C1_np1[0]=C1_np1[1]/(1.-(z1*e/(kb*T))*(Phi_np1[1]-Phi_np1[0]));
635       C2_np1[0]=C2_np1[1]/(1.-(z2*e/(kb*T))*(Phi_np1[1]-Phi_np1[0]));
636
637       Phi_np1[Lx]=0;
638       C1_np1[Lx]=C_bulk/fabs(z1);
639       C2_np1[Lx]=C_bulk/fabs(z2);
640
641       /*Phi_np1[Lx]=-Phi_wall;
642       C1_np1[Lx]=C1_np1[Lx-1]/(1.+(z1*e/(kb*T))*(Phi_np1[Lx]-Phi_np1[Lx-1]));
643       C2_np1[Lx]=C2_np1[Lx-1]/(1.+(z2*e/(kb*T))*(Phi_np1[Lx]-Phi_np1[Lx-1]));*/
644       /// NOTE FOR USER #10: Refer to #6. Replace lines 637 to 639 with lines
641 to 643.
645
646       phantom_phi_left_np1=2*Phi_np1[0]-Phi_np1[1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_np1[0]+z2*e*C2_np1[0]);
647       phantom_phi_right_np1=0;
648       //phantom_phi_right_np1=2*Phi_np1[Lx]-Phi_np1[Lx-1]-
(sq(dx)/Eps)*(rho0+z1*e*C1_np1[Lx]+z2*e*C2_np1[Lx]);
649       /// NOTE FOR USER #11: Refer to #6. Replace line 647 with line 648.
650
651       norm_change_Phi=norm_change_Phi/compute_norm(&grid,Phi_np1);
652       norm_change_C1=norm_change_C1/compute_norm(&grid,C1_np1);
653       norm_change_C2=norm_change_C2/compute_norm(&grid,C2_np1);
654       if ((norm_change_Phi<EPSILON) && (norm_change_C1<EPSILON) &&
(norm_change_C2<EPSILON))
655       {
656         break;
657       }
658     }
659     if (k>MAX_NEWTONS)
660     {
661       cout << "ERROR: Newton method failed to converge after " << MAX_NEWTONS
<< " iterations" << endl;
662       exit(ERROR);
663     }
664     // end of TRBDF2
665
666     /// check time convergence
667     difference_between_tsteps_Phi=fabs(compute_norm(&grid, Phi_np1)-
compute_norm(&grid, Phi_n));
668     difference_between_tsteps_C1=fabs(compute_norm(&grid, C1_np1)-
compute_norm(&grid, C1_n));
669     difference_between_tsteps_C2=fabs(compute_norm(&grid, C2_np1)-
compute_norm(&grid, C2_n));
670
671     /// Values of n+1 becomes values of n at next time step
672     for (int i=grid.first; i<=grid.last; i++)
673     {
674       C1_n[i]=C1_np1[i];
675       C2_n[i]=C2_np1[i];
676       Phi_n[i]=Phi_np1[i];
677     }
678     /// Boundary conditions at n of next time step
679     Phi_n[0]=Phi_np1[0];
680     C1_n[0]=C1_np1[0];
681     C2_n[0]=C2_np1[0];
682
683     Phi_n[Lx]=Phi_np1[Lx];
684     C1_n[Lx]=C1_np1[Lx];
685     C2_n[Lx]=C2_np1[Lx];
686
687
688     /// Data storage in files
689     time.t=(time.n+1)*dt;
```

```
        numerical_solver << endl;
        numerical_solver << "Time: " << std::setprecision(12) << time.t << endl;
        numerical_solver << "Phi:" << endl;
        for (int i=grid.first-1; i<=grid.last+1; i++)
        {
            numerical_solver << Phi_n[i] << " ";
        }
        numerical_solver << endl << "C1:" << endl;
        for (int i=grid.first-1; i<=grid.last+1; i++)
        {
            numerical_solver << C1_n[i] << " ";
        }
        numerical_solver << endl << "C2:" << endl;
        for (int i=grid.first-1; i<=grid.last+1; i++)
        {
            numerical_solver << C2_n[i] << " ";
        }


difference_between_tsteps_Phi=difference_between_tsteps_Phi/compute_norm(&grid,Phi_n);
difference_between_tsteps_C1=difference_between_tsteps_C1/compute_norm(&grid,C1_n);
difference_between_tsteps_C2=difference_between_tsteps_C2/compute_norm(&grid,C2_n);
        if(difference_between_tsteps_Phi<EPSILON && difference_between_tsteps_C1<EPSILON && difference_between_tsteps_C2<EPSILON)
        {
            break;
        }

    }
    ofstream convergence_time("./convergence_time.txt", ofstream::out);
    convergence_time << "Time: " << std::fixed << std::setprecision(12) << time.t << endl;
    convergence_time.close();
    ofstream results("./results.dat", ofstream::out);
    results << "i\tx\tPhi\tC1\tC2" << endl;
    for (int i=grid.first-1; i<=grid.last+1; i++)
    {
        results << i << " " << i*dx << " " << Phi_n[i] << " " << C1_n[i] << " " << C2_n[i] << endl;
    }
    results.close();
    // output in data for matlab
    ofstream x("./x.dat", ofstream::out);
    for (int i=grid.first-1; i<=grid.last+1; i++)
    {
        x << i*dx << endl;
    }
    x.close();
    ofstream phi("./phi.dat", ofstream::out);
    for (int i=grid.first-1; i<=grid.last+1; i++)
    {
        phi << Phi_n[i] << endl;
    }
    phi.close();
    ofstream C1("./C1.dat", ofstream::out);
    for (int i=grid.first-1; i<=grid.last+1; i++)
    {
        C1 << C1_n[i] << endl;
    }
    C1.close();
    ofstream C2("./C2.dat", ofstream::out);
    for (int i=grid.first-1; i<=grid.last+1; i++)
    {
        C2 << C2_n[i] << endl;
    }
    C2.close();
    numerical_solver.close();
    return 0;
}


```

## Numerical solver

```
#ifndef NUMERICAL_SOLVER_H_INCLUDED
#define NUMERICAL_SOLVER_H_INCLUDED

using namespace std;

#define max(a,b)     ( ((a) > (b)) ? (a) : (b) )
#define min(a,b)     ( ((a) < (b)) ? (a) : (b) )
#define sq(x)    ((x)*(x))
#define max_iterations 1000

typedef struct {
    int Lx; // number of grid points
    int modes;  // number of unknowns
    double dx, xmin, xmax;
    int first, last; // first and last unknown cells (excluding BC)
} GRID;

typedef struct {
    int n;
    double dt, t;
    bool REDO_STEP;
} TIME;

void init_grid(GRID *grid, TIME *time);
double compute_norm(GRID *grid, double x[]);
void gauss_seidel_solver(GRID *grid, double X[], double *A[], double b[]);
void thomas_algorithm_solver(GRID *grid, double X[], double *A[], double b[]);

#endif // NUMERICAL_SOLVER_H_INCLUDED
```

## Memory allocation

```
#ifndef MEMORY_ALLOC_H_INCLUDED
#define MEMORY_ALLOC_H_INCLUDED

using namespace std;

#define INT     ((unsigned) sizeof(int))
#define FLOAT       ((unsigned) sizeof(float))
#define DOUBLE      ((unsigned) sizeof(double))
#define CHAR        ((unsigned) sizeof(char))

#define ERROR -1

typedef void *POINTER; // pointer to an unknown data type
```

```

POINTER Alloc(unsigned N_bytes);
POINTER alloc_vector(int N, unsigned element_size);
POINTER alloc_matrix(int N_rows, int N_columns, unsigned element_size);

#endif // MEMORY_ALLOC_H_INCLUDED
```

# REFERENCES

1 **T. Trindade, P. O'Brien**, and **N. L. Pickett**, «Nanocrystalline Semiconductors: Synthesis, Properties, and Perspectives», Chemistry of Materials, vol. 13, no. 11, p. 3843-3858, 2001

2 **Y. Xia and al**., « One-dimensional nanostructures: synthesis, characterization, and applications », Advanced Materials, vol. 15, no. 5, p. 353 –389, 2003.

3 **B. D. Gates, Q. Xu, M. Stewart, D. Ryan, C. G. Willson, and G. M. Whitesides**, « New Approaches to Nanofabrication: Molding, Printing, and Other Techniques», Chemical Reviews , vol. 105, no.4, p. 1171-1196, 2005.

4 **B. Nikoobakht**, « Toward industrial-scale fabrication of nanowire-based devices», Chemistry of Materials, vol. 19, no. 22, p. 5279 – 5284, 2007.

5 **Reuss, F**. : 1809, Sur un nouvel effet de l'électricité galvanique, Mémoires de la Société Impériale des Naturalistes de Moscou 2, 327–337.

6 **Wiedemann, G**. : 1852, Ueber die bewegung von flüssigkeiten im kreise der geschlossenen galvanischen säule, Poggendorf's Annalen der Physik und Chemie 87, 321–352.

7 **Wiedemann, G**. : 1856, Ueber die bewegung der flüssigkeiten im kreise der geschlossenen gal-vanischen säule und ihre beziehungen zur elektrolyse, Poggendorf's Annalen der Physik und Chemie 99, 177–233

8 **Quincke, G**. : 1859, Ueber eine neue art elektrischer ströme, Poggendorf's Annalen der Physik und Chemie 107, 1–47.

9 **Quincke, G**. : 1861, Ueber die fortführung materieller theilchen durch strömende elektricität, Poggendorf's Annalen der Physik und Chemie 113, 513–598.

10 **Dorn, E**. : 1878, Ueber die galvanischen ströme, welche beim strömen von flüssigkeiten durch□röhren erzeugt werden, Wiedemann's Annalen der Physik und Chemie 5, 20–44.

11 **Spiegler, K. S**. : 1958, Transport processes in ionic membranes, Trans. Faraday Soc. 54(9), 1408–1428.

12 **Yeung, A. T.** : 1994, Electrokinetic flow processes in porous media and their applications, in M. Corapcioglu (ed.), Advances in Porous Media, Vol. 2, Elsevier, Amsterdam, The Netherlands, chapter 5, pp. 309–395.

13 **Mitchell, J. K**. : 1993, Fundamentals of Soil Behavior, 2nd edition, John Wiley and Sons, New York.

[14] **Russel, W. B.; Saville, D. A.; Schowalter, W. R**. Colloidal dispersions, Cambridge University press. 1989.

[15] **Hunter, R. J.** Foundations of Colloid Science Vol. 1. Oxford Science publications. 1987

[16] **Hunter, R. J.** Foundations of Colloid Science Vol. 1. Oxford Science publications. 1987.

[17] **Yates, D. E.; Healy, T. W**. Mechanism of Anion Adsorption at the Ferric and Chromic Oxide/Water Interfaces. Journal of Colloid and Interface Science. 1975, 52, 222 – 228.

[18] **Davis, J. A.; James, R. O.; Leckie, J. O**. Surface Ionization and Complexation at the Oxide/Water Interface: I. Computation of Electrical Double Layer properties in Simple Electrolytes. Journal of Colloid and Interface Science. 1978, 63, 480 – 499.

[19] **Charmas, R.; Piasecki, W.; Rudzinski, W**. Four layer complexation model for ion adsoption at electrolyte/oxide interface: theoretical foundations. Langmuir. 1995, 11, 3199 – 3210.

20 **Von Helmholtz, H. L. F**. : 1879, Studien über electrische grenzschichten, Wiedemann's Annalen der Physik und Chemie 7, 337–382.

21 **Perrin,J.** : 1904, Mécanisme de l'électrisation de contact et solutions colloïdales I, Journal de Chimie Physique 2, 601–651

22 **Perrin, J.** : 1905, Mécanisme de l'électrisation de contact et solutions colloïdales II, Journal de Chimie Physique 3, 50–110

23 **Gouy, G**. : 1909, Sur la constitution de la charge électrique à la surface d'un électrolyte, Comptes Rendus de l'Académie des Sciences 149, 654–657

24 **Chapman, D. L**. : 1913, A contribution to the theory of electrocapillarity, Philosophical Magazine and Journal of Science, Ser. 6 25 (148), 475–481.

[25] **Stern, O.** : 1924, Zur theorie der elektrolytischen doppelschicht, Z. Elektrochem. 30, 508–516.

[26] **Delahay, P**. : 1965, Double Layer and Electrode Kinetics, Interscience Publishers, New York

[27] **Usui, S.** : 1984, Electrical double layer, in A. Kitahara et A. Watanabe (eds), Electrical Phenomena at Interfaces, Fundamentals, Measurements and Applications, Marcel Dekker, INC, New York and Basel, pp. 15–46

[28] **Masliyah, J. H.** : 1994, Electrokinetic Transport Phenomena, 12, Aostra Technical Publications

[29] **Durand-Vidal**, **S. et Simonin, J. P**. : 2000, Electrolytes at Interfaces, Kluwer, London.

[30] **BANK , R. E., COUGHRAN JR, W. C., FICHTNER, W., GROSSE, E., ROSE, D.& SMITH, R. , 1985** Transient simulation of silicon devices and circuits.