Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Materials Science and Engineering

Kseniia Koseniuk

Computational Modelling of Mass Transfer in Molten Salt Electrolyte for Na-Zn Liquid Metal Batteries

Master's thesis in Materials Science and Engineering
Supervisor: Kristian Etienne Einarsrud
Co-supervisor: Omar Emmanuel Godinez Brizuela
June 2023

NTNU
Norwegian University of
Science and Technology

Kseniia Koseniuk

Computational Modelling of Mass Transfer in Molten Salt Electrolyte for Na-Zn Liquid Metal Batteries

Master's thesis in Materials Science and Engineering
Supervisor: Kristian Etienne Einarsrud
Co-supervisor: Omar Emmanuel Godinez Brizuela
June 2023

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Materials Science and Engineering

Preface

Background for the report

This thesis is submitted as a partial requirement for obtaining the Master of Science in Engineering degree at the Norwegian University of Science and Technology. The thesis is awarded 30 credits, corresponding to approximately 800 hours of work carried out throughout 20 weeks of the 10th and last semester (spring 2023) of the degree at the Department of Material Science and Engineering. The report contains the theoretical framework and numerical implementation for creating transient simulations of electrokinetic transport in the molten salt electrolyte layer of the Na-Zn liquid metal battery using OpenFOAM computations fluid dynamics software. The work was carried out under the supervision of Professor Kristian Etienne Einarsrud and Postdoctoral fellow Omar Emmanuel Godinez Brizuela.

Acknowledgements

Firstly, I would like to thank my supervisor Kristian Etienne Einarsrud for giving me the opportunity to work on such an interesting subject that has sparked so much love for CFD in my heart. I am grateful for all the feedback and help you have given me always responding unreasonably quickly.

I want to also thank my co-supervisor Omar Emmanuel Godinez Brizuela for sharing the sacred knowledge of OpenFOAM with me and always making the time to help me and give me valuable feedback whenever I endured yet another crisis.

I thank my friends and classmates for all the fun times and struggles we have experienced together. It would not have been the same without you.

Finally, I want to thank my aunt and uncle for giving me the opportunity to get my higher education in Norway. I will always be grateful for the life this chance they took on me enables me to have.

Abstract

Liquid metal batteries (or LMBs) are an emerging battery technology gaining relevance for large-scale on-grid energy storage. This storage could contribute to decreasing the fluctuations in power supply resulting from the introduction of intermittent renewable energy sources, bridging the gap between energy supply and demand. The Na-Zn LMB has shown some potential for providing cheap energy storage employing highly abundant non-toxic materials. However, this battery exhibits high rates of self-discharge and other significant limitations. The electric current through the molten salt electrolyte layer is realised by the transport of ions, which thus has a significant impact on the battery's functioning and properties, such as coulombic efficiency, electrolyte resistance, potential drop, maximum charge and discharge currents and more. This report presents the modelling of the mass transport in said electrolyte layer including diffusion and migration mechanisms to provide a deeper understanding of the ways this transport can be simulated, how its behaviour is affected by the presence of a porous diaphragm, and its impact on the resistance present. This insight can help expand the existing knowledge on LMBs and ensure optimal performance for grid energy storage. In the present work, OpenFOAM software was used to create a solver that was used to model the described transport in the molten salt electrolyte layer of Na-Zn LMB with a diaphragm in place. It can be further expanded and applied in different ways than done in this project in future research. A modified version of the solver was used to test the approach against existing results in aqueous solution systems. Results under various conditions, representing the electrolyte layer during charging and discharging of the battery with various applied electric current densities and potential differences, have been produced and discussed. Different diaphragm properties were also tested and their impact on the mass transport was evaluated. The results provide insight into the ways the implementation of the model can be realized in OpenFOAM and into the behaviour of the transport under various conditions to allow for more efficient control of the process in the future for better battery operation. In future research, the model should be expanded to account for various mechanisms of convection, test different boundary condition implementations, and include all chemical compounds present in the electrolyte.

All used OpenFOAM solver codes and some example case setups can be accessed on GitHub: `https://github.com/XeniaKos/master`

Sammendrag

Flytende metall-batterier (eller FMB-er) er en fremvoksende batteriteknologi som blir stadig mer relevant for storskala energilagring på strømnettet. Denne lagringen kan bidra til å redusere variasjonene i strømforsyningen forårsaket økt bruk av fornybar energi, og dermed øke sammenfallet mellom energitilbud og etterspørsel. Na-Zn FMB har potensial for å tilby kosteffektiv energilagring ved å bruke ikke-giftige materialer som har stor tilgjengelighet. Imidlertid har denne type batteri en høy grad av selvutladning og andre betydelige begrensninger. Den elektriske strømmen gjennom det smeltede saltet i elektrolyttlaget drives av ionetransport, noe som dermed har betydelig innvirkning på batteriets drift og egenskaper, som koulombisk effektivitet, elektrolyttmotstand, elektrisk potentialfall, maksimale ladnings- og utladningsstrømmer og mer. Denne rapporten presenterer modelleringen av masseoverføringen i nevnte elektrolyttlaget, inkludert diffusjon og migrasjon, for å gi en dypere forståelse av hvordan denne transporten kan simuleres, hvordan den påvirkes av en porøs diafragm tilstedte, og dens innvirkning på motstanden som oppstår. Denne innsikten kan bidra til å utvide den eksisterende kunnskapen om FMB-er og sikre optimal ytelse for energilagring på strømnettet. I arbeidet utført ble OpenFOAM programvaren brukt til å opprette en numerisk løser som ble brukt til å modellere den beskrevne transporten i elektrolytten til Na-Zn FMB med en diafragm. Denne løseren kan videre utvides og anvendes på ulike måter enn det som er gjort i dette prosjektet i fremtidig forskning. En modifisert versjon av løseren ble brukt til å teste tilnærmingen mot eksisterende resultater i vannbaserte elektrolytter. Resultater under ulike forhold, som representerer elektrolyttlageret under ladding og utladning av batteriet med ulike påførte elektriske strømtettheter og potensialforskjeller, er blitt produsert og diskutert. Forskjellige diafragmsegenskaper ble også testet, og deres innvirkning på masseoverføringen ble evaluert. Resultatene gir innsikt i hvordan modellen kan gjennomføres i OpenFOAM og i transportens atferd under ulike forhold for å tillate mer effektiv styring av prosessen i fremtiden for bedre batteridrift. I fremtidig forskning bør modellen utvides for å ta hensyn til ulike konveksjonsmekanismer, teste ulike implementeringer av randbetingelser og inkludere alle kjemiske forbindelser som er tilstede i elektrolytten.

Alle utviklede OpenFOAM løsere samt med scalarTransportFoam løser som var brukt som grunnlag til de utviklede løsere og enkelte eksempeler er tilgjengelig på GitHub:
`https://github.com/XeniaKos/master`

Contents

Preface i

Abstract ii

Sammendrag iii

Figure list vii

1 Introduction 1
- 1.1 Background and motivation . . . 1
- 1.2 Na-Zn Liquid Metal Battery . . . 2
- 1.3 Aim and scope of the present work . . . 4

2 Literature review and state of the art 5
- 2.1 Liquid Metal Batteries . . . 5
- 2.2 Modelling of electrokinetic transport . . . 6

3 Theory 11
- 3.1 Dissolved species transport . . . 11
- 3.2 Summary . . . 16

4 Numerical approach 17
- 4.1 Finite volume method . . . 17
- 4.2 Time advancements . . . 21
  - 4.2.1 Forward Euler Scheme . . . 21
  - 4.2.2 Backward Euler Scheme . . . 22
- 4.3 Meshing . . . 22
- 4.4 Convergence . . . 22
- 4.5 Error and uncertainty . . . 23
- 4.6 Summary . . . 23

5 Implementation of the models in OpenFOAM 24
- 5.1 OpenFOAM environment . . . 24
  - 5.1.1 Meshing in OpenFOAM . . . 25
- 5.2 ScalarTransportFoam - the basis for the developed solvers . . . 25
- 5.3 SlayFoam solver for implementation testing . . . 28
- 5.4 SaltyFoam for electrokinetic transport in molten salt . . . 31
  - 5.4.1 Implementation of concentration boundary conditions in OpenFOAM . . 33

6 Case setups 37
- 6.1 Aqueous simulations . . . 37
  - 6.1.1 Mesh . . . 37
  - 6.1.2 Physical properties . . . 38
  - 6.1.3 Boundary and initial conditions . . . 40
  - 6.1.4 Solution and time controls . . . 40

6.2 Darcy-scale molten salt simulations 41
6.2.1 Mesh 43
6.2.2 Physical properties 44
6.2.3 Boundary and initial conditions 45
6.2.4 Solution and time controls 48

7 Results **51**
7.1 Verification of the results 51
7.2 Molten salt electrolyte simulation results 54
7.2.1 Convergence testing 54
7.2.2 Fixed current simulations 56
7.2.3 Fixed potential difference simulations 62
7.2.4 Diaphragms' parameter study 66

8 Discussion **69**
8.1 Aqueous model and simulations 69
8.1.1 Errors and uncertainties 69
8.2 Molten salt simulations 70
8.2.1 Errors and uncertainties 71

9 Concluding remarks **74**

10 Future work **75**

Bibliography **76**

11 Appendix **81**
11.1 Code: unedited scalarTransportFoam solver 81
11.1.1 scalarTransportFoam.C 81
11.1.2 createFields.H 83
11.1.3 Make/files 84
11.1.4 Make/options 84
11.2 Code: slayFoam solver 84
11.2.1 slayFoam.C 84
11.2.2 createFields.H 86
11.2.3 Make/files 88
11.2.4 Make/options 88
11.3 Code: saltyFoam solver 89
11.3.1 saltyFoam.C 89
11.3.2 createFields.H 91
11.3.3 Make/files 94
11.3.4 Make/options 95
11.4 Case setup: Permanganate tracer simulation compatible with the slayFoam solver 95
11.4.1 0 directory 95
11.4.2 constant directory 97
11.4.3 system directory 98
11.5 Case setup: FCD100 simulation compatible with the saltyFoam solver 103
11.5.1 0 directory 103
11.5.2 constant directory 109
11.5.3 system directory 109
11.6 Case setup: FPD0.018 simulation compatible with the saltyFoam solver 114
11.6.1 0 directory 114
11.6.2 constant directory 121

11.6.3 system directory 122

Figure list

1.1 Na-Zn liquid metal battery diagram [18]. 2

2.1 Schematic visualizations of the Darcy scale (left) and pore scale (right), where on the Darcy scale porous medium is described by porosity instead of an actual structure. 8
2.2 Schematic diagram showing the experimental setup for the work conducted by Rolle et al. [49], inspired by the diagram presented by Rolle et al. [49]. 10

4.1 An illustration of a 2D meshed solution domain. Computational nodes in the cell centres are shown in pink and are denoted with capital letters. Cell faces centres of the CVs/cells are denoted by lowercase letters. Vector $\vec{\mathbf{n}}_e$ is an example of a vector normal to face hosting the point e (also referred to as face e). 18
4.2 A close-up illustration of the 2D meshed solution domain presented in Figure 4.1 illustrating the used notation. 20

5.1 Diagram showing a general structure of an arbitrary simulation directory in OpenFOAM. 24
5.2 A flowchart illustrating the algorithm of scalarTransportFoam and the developed solvers. 27
5.3 Diagram showing the structure of an OpenFOAM simulation directory typical for the simulations created in the present work. 33

6.1 An illustration on the computational domain used in aqueous simulations: the type of the pink top and bottom boundaries is set to wall, the type of the blue left and right boundaries is set to patch, while the clear front and back boundaries are set as empty. 38
6.2 Mesh used for the aqueous simulations created using blockMesh with 280 divisions in the x-direction and 150 - in the y-direction (blue lines). 38
6.3 An illustration on the computational domain used in molten salt simulations: the pink top and bottom boundaries are defined as walls, the blue left and right boundary types are set to patch, while the clear front and back boundaries are set as empty. 44
6.4 Initial concentration conditions for Na^+ and Zn^{2+} ions for discharge process simulations (all FCD and FPD simulations). 46

7.1 Results of the permanganate tracer simulation from [49] (a) compared to the results of the permanganate simulation produced using slayFoam solver (b). . . . 52
7.2 Results of the Allura Red tracer simulation from [49] (a) compared to the results of the Allura Red simulation produced using slayFoam solver (b). 53
7.3 Results of the New Coccine tracer simulation from [49] (a) compared to the results of the New Coccine simulation produced using slayFoam solver (b). 54

7.4 Relative error in the resistance measurement compared to the next tested grid refinement after 15000 s. The pink line corresponds with simulations (G1, G3, G6, G7, G9, G11) where all tested meshes had cells aligning with the diaphragms' boundaries, while for the tests represented with the purple line (G2, G4, G5, G7, G8, G9, G10, G11), not all did. 55
7.5 Relative error in the resistance measurement compared to the next tested time step size after 30000 s using a charging current density of 100 mA/cm^2 and a mesh with 80000 cells. 56
7.6 Relative error in the resistance measurement compared to the next tested time step size after 110 s using a charging current density of 2000 mA/cm^2 and a mesh with 80000 cells. 56
7.7 Molar concentration distributions for fixed current charging battery simulations FCC100, FCC1000, and FCC2000 at the time the Na$^+$ concentration has passed zero (the dashed lines indicate the diaphragm's location). 57
7.8 Molar concentration distributions for fixed current discharging battery simulations FCD100, FCD1000, and FCD2000 at the time the Zn^{2+} concentration has passed zero (the dashed lines indicate the diaphragm's location). 58
7.9 Electric potential at the non-zero boundary normalized against the first recorded value as a function of time normalized by the end time for simulations C100, C1000, and C2000. 59
7.10 Electric potential at the non-zero boundary normalized against the first recorded value as a function of time normalized by the end time for simulations D100, D1000, and D2000. 59
7.11 Electrolyte resistance values for fixed current charging and discharging simulations plotted against time normalized by the end time for each simulation. 60
7.12 Electrolyte conductivity values for fixed current charging and discharging simulations plotted across the electrolyte layer thickness at the end time for each simulation (the dashed lines indicate the diaphragm's location). 61
7.13 Electrolyte resistivity, r, values for fixed current charging simulations plotted across the electrolyte layer thickness at the end time for each simulation (the dashed lines indicate the diaphragm's location). 61
7.14 Molar concentration distributions for fixed electric potential difference charging battery simulations FPC0.025, FFC0.333, and FPC0.679 at the selected end times (the dashed lines indicate the diaphragm's location). 62
7.15 Molar concentration distributions for fixed electric potential difference discharging battery simulations FPD0.018, FPD0.236, and FPD0.48 at the selected end times (the dashed lines indicate the diaphragm's location). 64
7.16 Molar concentration distributions for 0.99 V fixed electric potential difference charging and discharging battery simulations (FPC0.99 and FPD0.99) at the selected end times (the dashed lines indicate the diaphragm's location). 64
7.17 Electrolyte resistance values for fixed potential difference charging and discharging simulations plotted against time normalized by the end time for each simulation. 65
7.18 Electrolyte conductivity values for fixed potential difference charging and discharging simulations plotted across the electrolyte layer thickness at the end time for each simulation (the dashed lines indicate the diaphragm's location). 66
7.19 Electrolyte resistance plotted against time normalized by the end time for each of simulations PT0.6, PT0.85, and PT0.95. 67
7.20 Absolute values of mol% change in the amount of Cl$^-$ ions in the electrolyte against the simulation end times for fixed electric current (FC), fixed potential difference (FP, excluding cases FPC0.025 and FPD0.018), and diaphragm parameter testing (PT) simulations with trend lines for FC and FC cases. 68

8.1 Molar concentration distributions for discharging 0.99 V fixed electric potential difference simulation (FPD0.99) at the end time (a dip in Na^+ concentration is highlighted by the red circle and arrow). 72

List of Tables

5.1 Information about the objects defined in the createFields.H file of the scalarTransportFoam solver. . . . 28
5.2 The notation of variables used in solver codes and the corresponding notation used in the equations in the report together with the physical meaning behind them. . . . 29
5.3 Summary of changes made to variables and constants in scalarTransportFoam when creating slayFoam. . . . 31

6.1 Transport properties used for the aqueous simulations. . . . 39
6.2 A summary of constants used in all aqueous simulations. . . . 39
6.3 Porous matrix tortuosity, τ^2, and background electrolyte conductivity, kap, values used in simulations with permanganate, Allura Red, and New Coccine tracer plumes. . . . 39
6.4 Initial concentrations of tracer plumes. . . . 40
6.5 Fixed value left and right boundary conditions for the electric potential for all aqueous cases. . . . 40
6.6 Parameters used for matrix solution algorithms for aqueous simulations. . . . 41
6.7 Numerical schemes used in aqueous solution simulations done using the slayFoam solver. . . . 41
6.8 Summary of the notation used for convergence testing simulations run under fixed current conditions for a charging battery, where GCT stands for Grid Convergence Testing and TSCT stands for Time-Step Convergence Testing. . . . 42
6.9 Summary of the notation used for fixed current density and fixed potential difference charging and discharging simulations. . . . 43
6.10 Summary of the notation and parameters used for diaphragm properties' testing fixed current density charging simulations. . . . 43
6.11 Diffusion coefficients used in all molten salt electrolyte simulations for the three modelled ionic species. . . . 44
6.12 A summary of universal constants used in all molten salt simulations. . . . 45
6.13 The overview of the concentration boundary condition types used for Na^+ and Zn^{2+} ions. . . . 46
6.14 Values used for the fixed gradient concentration BCs for the fixed current density simulations. . . . 47
6.15 Electric potential fixed gradient boundary condition values for the left boundary for all fixed current density molten salt electrolyte simulations. . . . 47
6.16 Values used for fixed value potential BCs and in initial distribution conditions for the fixed electric potential difference simulations. . . . 48
6.17 Time controls used for the convergence testing simulations. . . . 49
6.18 Time controls used for the fixed current density, fixed potential difference, and diaphragm testing simulations. . . . 50

7.1 Values of the centre of mass displacements reported by Rolle et al. [49] (S) and achieved in the conducted simulations (S'). . . . 51

7.2 The relative change in the number of mols of Cl^- in the electrolyte between the beginning and the end time of the fixed current simulations. 62
7.3 The relative change in the number of mols of Cl^- in the electrolyte between the beginning and the end time of the fixed electric potential difference simulations. . 66
7.4 The relative change in the number of mols of Cl^- in the electrolyte between the beginning and the end time of the diaphragm parameters testing simulations. . . 67

1 Introduction

1.1 Background and motivation

The demand for a reliable supply of electricity has been increasing for many decades. Simultaneously, the desire to reduce the strain of energy production on the environment has pushed a transition from fossil fuel to renewable sources [1]. Such sources, like wind or solar, are highly intermittent and less reliable. The previously existing fluctuations in energy demand are now accompanied by fluctuations in supply, making the balance between the two even more challenging to achieve [2]. Large-scale on-grid energy storage is important for achieving this balance and would improve the grid's resilience [3]. However, for the most part, electrical grids have limited energy storage capacity often with low energy efficiency and form flexibility [4].

The necessary grid flexibility today is for the most part achieved by employing either Pumped Hydro Energy Storage (PHES) or traditional power plants. PHES allows one to store energy in form of the potential energy of water by pumping it to a higher altitude. Although PHES exhibits high efficiency (about 80% [5]) and is cheap, especially if used for a long time, it is space- and water-demanding, making it impossible to realize in certain cases. On the other hand, when traditional thermal power plants are used to provide grid flexibility they are forced to operate in part-load mode, making them less efficient, polluting the environment with emissions even more than they normally would [5].

These issues are a part of the reason for the increased interest in battery grid energy storage in recent years which promise features of flexibility, high energy efficiency and long cycle life. However, not all battery technologies are optimal for this kind of storage for various reasons. For example, the popular Li-ion battery dominating the market for portable electronics has more than good enough electrical properties for grid storage but due to the high cost of such storage [6], limited resources of Li [7], and safety concerns associated with the use of organic solvers it is not the best choice for on-grid storage [2].

Liquid Metal Batteries (LMBs) have emerged as a technology potentially suitable for large-scale on-grid storage [8, 9]. As the name suggests, the three electro-active components of these batteries, the two electrodes and the electrolyte, are liquid during operation. These components naturally separate into three layers due to immiscibility and density differences between them at the working temperatures [10]. The liquid state of the battery comes with some physical advantages. The charge transfer is very fast due to the electrode-electrolyte interface being liquid, vastly reducing the charge transfer resistance and the associated charge transfer overpotential [11]. In addition, the molten salt electrolyte has high conductivity and allows for rapid mass transfer. These properties in turn result in low ohmic and mass transfer potential losses. This allows LMBs to achieve relatively high voltage efficiencies even at high current densities [12]. These batteries also have longer service life than their solid-state counterparts. The end of life in solid-state batteries is often caused by microstructural degradation of the electrode which of course cannot take place in the liquid state [13]. A longer lifetime contributes to a lower total cost and emissions per use [2]. Another advantage of LMBs is their potential for very low cost compared to other battery technologies due to their use of inexpensive active materials [14, 15].

Some of the main disadvantages of LMBs are high rates of self-discharge (discharge over time with no connection to an external circuit), high operating temperatures, unwanted solid formation, and corrosion of the components [15, 10, 16]. These issues must be addressed before the potential application of such cells, by carefully investigating the causes and gaining an understanding of the ways they can be managed.

The present work is a part of the SOLISTICE project [17] that has 12 partners from 7 European countries, including NTNU in Trondheim. The objective of the project is to solve the discrepancy between energy supply and demand by developing Na-Zn LMB for stationary energy storage. Therefore, Na-Zn LMB is in focus in this thesis. The other partners of SOLISTICE are working on other aspects of Na-Zn LMB, like container materials, electrolyte composition, battery cell experiments, the socio-economic implications of Na-Zn LMB use, and more.

1.2 Na-Zn Liquid Metal Battery

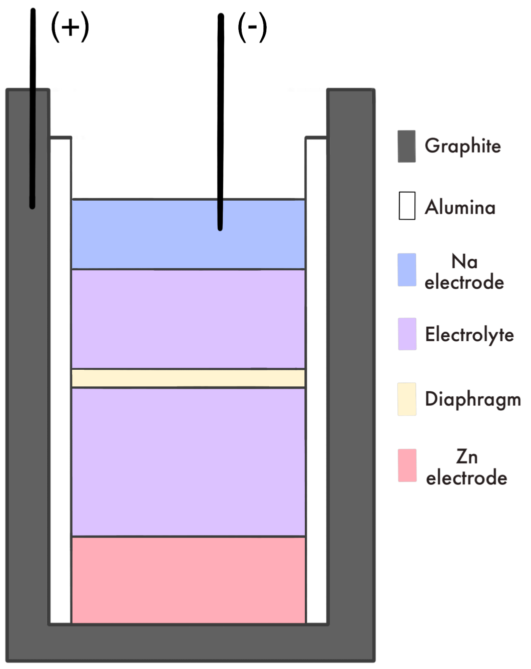


Figure 1.1: Na-Zn liquid metal battery diagram [18].

Na-Zn LMB consists of three layers of electroactive components placed in a container, as illustrated by Figure 1.1. The positive liquid Zn electrode with the highest specific mass rests at the bottom of the cell in contact with the graphite crucible which also serves as the positive current collector. On top of the Zn electrode rests a NaCl-CaCl$_2$-ZnCl$_2$ molten salt electrolyte inside of which lays a diaphragm composed of a porous ceramic material. Finally, at the top floats the low-density liquid Na electrode. A molybdenum wire has been used as the negative current collector inserted around 2cm into the electrolyte in the discharged state. Alloying of Mo with Zn resulting in solid formation has been reported [10], indicating a need for a better choice of current collector material. It is important to note that the sides of the container are insulated with alumina to prevent electrical contact between the two electrodes that would result in short-circuit [19]. The approach of using alumina for insulation can prove too expensive and fragile for grid-scale applications, with other LMB designs using metal foam to avoid contact between the negative electrode and the vessel walls [13]. In addition, sodium reactions with the alumina lining have been reported, making the selection of a more optimal insulating material more so relevant [10].

The cell reactions of the Na-Zn LMB are as follows [10]:

Negative electrode:

$$2\,\mathrm{Na} \underset{\text{charge}}{\overset{\text{discharge}}{\rightleftharpoons}} 2\,\mathrm{Na}^{+} + 2\,\mathrm{e}^{-} \tag{1.1}$$

Positive electrode:

$$\mathrm{Zn}^{2+} + 2\,\mathrm{e}^{-} \underset{\text{charge}}{\overset{\text{discharge}}{\rightleftharpoons}} \mathrm{Zn} \tag{1.2}$$

Total cell reaction:

$$\mathrm{ZnCl_2} + 2\,\mathrm{Na} \underset{\text{charge}}{\overset{\text{discharge}}{\rightleftharpoons}} \mathrm{Zn} + 2\,\mathrm{NaCl} \tag{1.3}$$

As illustrated by the reaction equations (1.1)-(1.3), during the charging of the battery Na^+ ions in the electrolyte gain electrons and get reduced to Na metal forming the negative electrode. Simultaneously, on the positive electrode Zn metal gets oxidized to Zn^2+ in the electrolyte [19]. During discharge, the given reactions proceed in the opposite direction. The current flow through the electrolyte layer between the electrodes happens by means of ion transport, making it critical for the functioning of the battery and its' optimization.

The high rate of self-discharge is a significant drawback of Na-Zn LMBs. The self-discharge rate is generally dictated by the battery's storage temperature, electrode and electrolyte compositions, and the current collector. The main source of self-discharge in Na-Zn LMB is considered to be the loss of Na. The main mechanisms for this loss are the evaporation of Na metal, the dissolution of Na metal into the electrolyte, and a reaction of Na with $ZnCl_2$ in the electrolyte. It is hypothesized that the last-mentioned mechanism can be suppressed by controlling the transport of $ZnCl_2$ into the upper compartment of the cell by optimizing the diaphragm properties [10]. The high cell temperature increases the rate of loss of Na by allowing for a prolonged reaction of Na with $ZnCl_2$, accelerating the Na dissolution, and increasing the vapour pressure of Na. Thus, developing an electrolyte with a lower melting point to decrease the operating temperature is important moving forward [19].

Because the current flow through the electrolyte layer is carried by ions, their transport is important to understand to better the battery operation. It bears great significance for the crucial battery properties including the ohmic drop in the electrolyte, achievable current densities, and coulombic efficiency (which can be defined as the ratio of the total charge extracted from the battery to the total charge previously put into the battery over a full cycle [20]). The effects of the diaphragm on this transport must be better understood for optimal battery performance. Modelling of ionic transport can provide insight into its performance under various conditions and help guide experimental work in the right direction. Most of the simulation work on mass transport in LMBs is focused on advection and does not consider the presence of a porous diaphragm [11, 13, 21, 22]. However, convection is not the only transport phenomenon that is present and needs to be accounted for. Migration is a significant contributor to the dissolved species transport inside the electrolyte layer of LMB, making it an important transport mechanism to consider [11]. Migration is present both in the open electrolyte volume and in the electrolyte taking up the pore space of the porous diaphragm. This project is focused mainly on modelling migration and diffusion in the molten salt electrolyte layer in the Na-Zn LMB, accounting for the presence of porous media to expand the existing knowledge on the subject and help improve the battery's properties.

1.3 Aim and scope of the present work

The overall objective of this thesis is to extend the existing body of knowledge related to mass transport in Liquid Metal Batteries. This will be achieved by:

1. Providing an overview of electrokinetic mass transfer in molten salt systems with a porous material obstruction
2. Creating a transient Darcy-scale model for electrokinetic mass transport in molten salt electrolyte with a porous obstruction in place, enforcing electroneutrality in the bulk fluid
3. Testing the model under a relaxed electroneutrality condition against existing results for a similar model of electrically driven transport through a porous medium
4. Applying the model to the Na-Zn LMB electrolyte layer under charging and discharging conditions taking a porous diaphragm into consideration testing both fixed electric current density and fixed electric potential difference conditions
5. Investigating the effects of porous diaphragms' properties on ion transport and the resistance in the electrolyte layer.

The simulations in the present work are realized using OpenFOAM [23] version 2212, an open-source computational fluid dynamics software. The software has an extended array of features and solvers that can be used to create models for a wide range of fluid dynamics and transport problems. In addition, it lets users create new solvers, that employ the pre-existing functionalities to better suit the specific problems.

2 Literature review and state of the art

2.1 Liquid Metal Batteries

In the sixties, LMBs were a topic of active research, with interest fading in the seventies, due to LMBs' low specific energy and unsuitability for mobile applications. The interest has spiked again recently due to LMBs' potential in low-cost grid-scale applications [21]. Many potential LMB chemistries have been and are being investigated [15, 24, 25]. Some of the most important properties potential active materials should have are low melting temperature and low price, but other aspects like corrosivity and toxicity should be considered. There must also be a substantial difference in the electronegativity of the electrode materials in order to drive the electrochemical process and allow for a favourable open circuit voltage (OCV) which is commonly used to evaluate the electrical potential capability of batteries. In addition, the densities of the electrode and electrolyte materials must be suitable to achieve the separation of the layers. The most commonly considered LMB chemistries are Li, Na, K, Mg, or Ca-based [24].

Lithium is a material of interest due to it having the lowest oxidation potential of all elements, providing a possibility for producing high OCV when combined with various different cathode materials. It has a low melting temperature at 180 °C [24]. However, Li is a low-abundance [7] relatively high-price [26] material in high demand due to alternative competing battery technologies, such as Li-ion. Li-Bi, Li-Te, Li-Sb, and Li-Sn-Pb are the most commonly considered Li-based LMB cell options. For these systems, open circuit voltage varies from around 0.75 V for Li-Pb to 1.7 V for Li-Te cells. Te has a high cost and low abundance reducing the prospects of its use. Bi and Sb are attractive candidates due to their relatively low cost and high OCV. Although the Li-Bi cells show good rate capability and long cyclic performance, the price of Bi is too high for large-scale storage [24]. Sb has a relatively high melting temperature (631°C) resulting in higher operational costs, corrosion, and design issues [24]. This temperature, however, can be reduced to 450°C by alloying Sb with Pb without compromising the battery's OCV and Coulombic efficiency [24].

Sodium is an attractive choice for electrode material because of its high abundance [7], low cost, low melting temperature, relatively low oxidation potential, and low environmental impact, promising great scalability. Commonly researched material combinations for Na-based LMBs are Na-Bi, Na-Sn, and Na-S (only a partially liquid cell) [24]. The Na-Sn cell showed relatively low OCVs (0.33 - 0.43 V) and required a quite high operational temperature of 700 °C. Na-Bi cells showed superior properties in comparison but exhibited high rates of self-discharge caused by high Na solubility in the electrolyte. This is an important issue when using Na as the anode, making suppression of Na solubility necessary. This can be done by optimizing the electrolyte composition or reducing the operational temperature (although it is hard to go below 500 °C). This issue can also be combated by introducing a solid ion selective membrane, eliminating the need for electrolyte [24]. This has been done in Na-S cells (also known as ZEBRA batteries) that have shown good cell performance with nearly no self-discharge [27]. However, these cells come with some disadvantages as well, like additional costs of high-quality membranes and weak cell robustness with the possibility of membrane cracking [28].

Calcium is a highly abundant element [7], however, the Ca-based LMB cells like Ca-Bi, Ca-Sb, and Ca-Mg are held back by calcium's high melting temperature. The high solubility of Ca in molten salt electrolytes is also a challenge when it comes to the coulombic efficiency of these batteries [24].

The reactions taking place in LMBs, allowing them to be charged and discharged, are controlled by the mass transport of reactants and products at the electrode-electrolyte interfaces. It has been reported that the majority of the voltage drop in LMBs can be caused by poor mass transport, especially at high current densities [29]. Specie and potential distribution models in Li-Bi LMB done by Duczeka et al. [30] have shown that mass transport and concentration gradients significantly affect the cell overpotentials and thus the cell voltage and performance. The maximum cell current is also known to be mass transport-limited [30]. In addition, understanding and controlling mass transfer in LMBs is crucial for avoiding dendrite formation, short-circuiting, phase segregation, and other potential issues and for ensuring highly efficient functioning of the batteries. However, the mass transfer phenomenon in LMBs is highly complex with the coupling of magnetohydrodynamic, heat transfer and electrokinetic effects present, making its' optimization far from trivial [31, 32].

With ions being charged particles, electrokinetically driven transport plays an important role in the total mass transfer in the electrolyte layer of LMBs. Electrokinetic transport includes phenomena such as electromigration, electrophoresis, electroosmosis, and other electrically driven flows.

2.2 Modelling of electrokinetic transport

A significant amount of research on the modelling of mass transport in LMBs, including electrokinetic transport in the molten salt electrolyte, exists. However, most of the research on mass transport in the electrolyte layer is focused on different types of convection, since it is an important contribution and has a significant impact on the functioning of LMBs [11, 32, 33]. In addition, the considerations of a porous media present in the electrolyte layer are normally missing [13, 21, 22, 34]. This can be explained by the fact that most LMB technologies do not employ such porous diaphragms as Na-Zn LMBs do [24, 25], making it especially interesting to investigate.

On the other hand, modelling electrokinetic transport (also in porous media) outside of LMB applications is an active area of research. It has gained a lot of attention in the micro- and nanofluidics applications, especially in recent years proving relevant for water purification and desalination, soil remediation, nanoelectronics and energy conversion, ion-selective transport systems and many others [35, 36, 37]. A typical model used when studying electrokinetic transport is the Poisson-Nernst-Planck model. This model couples the Poisson equation for the electric potential distribution with the Nernst-Planck equation for the concentration distribution to model the transport behaviour of charged species [38].

A considerable amount of research on electrokinetic transport in porous media and tight channels has been focusing on dilute aqueous solutions due to a wide range of applications [35, 37]. A phenomenon called electroosmosis is of great importance in this field because it makes these applications possible, requiring a deeper understanding of the subject. Electroosmotic flow is of relevance for polar solvent systems and has been extensively studied and modelled throughout the years [35, 36]. The modelling of mass transport in ionic mixtures, in general, is better described for dilute systems since the assumption of a dilute solution simplifies the modelling approach and equations used significantly. Since the electrolyte in the LMBs is composed of molten salts, electroosmotic flow is not expected (due to the solvent not being polar and ion concentrations being very high). In addition, molten salt mixtures do not constitute dilute solutions and behave differently from commonly modelled dilute aqueous solutions.

This project focuses mainly on developing a model of migration- and diffusion-dominated mass transport in a molten salt electrolyte in Na-Zn LMBs considering the presence of porous media to bridge the gap in the present knowledge.

Modelling of charged species transport can be conducted in numerous ways. As done by Yeh et al. [35], ion transport modelling is often categorised into three approaches based on the scale they consider: continuum, mesoscopic and molecular modelling. The continuum models provide information on macroscopic transport phenomena. This approach normally can be used when the length scale is greater than 1nm. Fluid can be treated as a continuum on this scale, which allows for employing the Navier-Stokes equation to describe the fluid flow. The results are often obtained relatively quickly, due to the coarseness of the scale used during computations. Molecular dynamics modelling is used when molecular behaviour and interactions are of interest, normally on a much smaller scale. This approach provides detailed information about transport taking all microscopic effects into consideration but is much more time and resource intensive. This makes this approach not favourable for describing "the big picture". Mesoscopic modelling can be described as the link between the two previously mentioned approaches, where the continuity and momentum equations are solved by tracking the movements of the molecule groups [35]. Considering that the length scales for the system investigated in this project are much larger than 1 nm, continuum modelling is the preferred approach.

In continuum modelling of the mass transport through a porous medium, it is convenient to differentiate between two scales illustrated in Figure 2.1: pore scale and macro scale (or Darcy scale). On the pore scale, each point in space is occupied by a specific phase, fluid or solid, applying the continuum assumption to each phase separately. Thus the relevant spatial structure of the porous material must be resolved in order to model on this scale. This allows one to capture local interactions between the phases and the transport mechanisms. On the contrary, the Darcy scale treats porous medium as a continuum where at each point in space both liquid and solid states exist simultaneously. The effects of the porous obstruction on the transport are then captured by using materials' bulk properties, such as porosity and tortuosity. This provides information about the averaged transport behaviour, blind to the local effects and interactions. As indicated in Figure 2.1, the Darcy scale is normally used on systems with a high ratio of total system size to the average pore size. This ensures a volume of porous material big enough to be representative, making the transport less dependent on local variations [39].

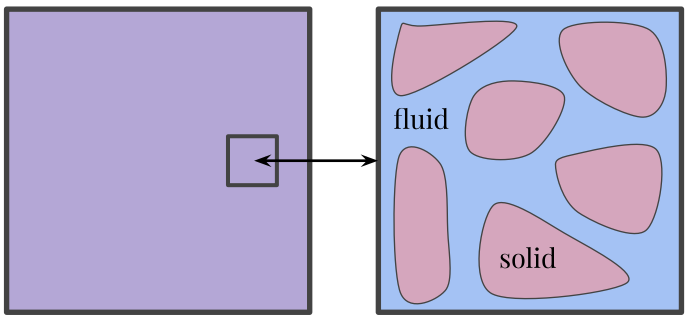


Figure 2.1: Schematic visualizations of the Darcy scale (left) and pore scale (right), where on the Darcy scale porous medium is described by porosity instead of an actual structure.

The Nernst-Planck equation is commonly used to describe the transport of charged species in various scenarios with promising results [35, 40, 41, 42]. The main advantage of the Nernst-Planck model is that it is accurate even with some deviations from the necessary assumptions, while not requiring a lot of unknown variable properties [43].

With process-based modelling of electrokinetic transport in heterogeneous porous medium Sprocati and Rolle [44] have proven the interplay between advection, electroosmosis and electromigration to be highly complex even for dilute aqueous solutions. This helped them conclude that multidimensional numerical simulations are necessary to better understand the transport mechanisms and interpret observed behaviour [44].

Huang et al. [38] have shown that the Nernst-Planck model is also valid for reactive transport cases with reaction-driven convection. This was done by conducting numerical simulations and comparing the results to those acquired from physical experiments and previously verified models [38].

Novotny and Gas [40] have created a mathematical model capable of calculating concentration profiles, electric potential and volume charge density of all electrolytes in the diffuse part of the electrical double layer that allows for deviation from electroneutrality. This model is based on the Poisson-Nernst-Planck equations. They have produced results by employing this model in COMSOL Multiphysics [45] CFD software.

Pimenta and Alves have developed implementations of electrically-driven flow models in OpenFOAM [46]. The charged specie transport model was based on the Poisson-Nernst-Planck equations as well and was coupled with the Navier-Stokes equations for the fluid flow modelling. Their work provides valuable insight into boundary condition implementation and discretization for such models. Zero flux boundary conditions in the present work were realized in a way similar to the one used by Pimenta and Alves [46].

In their works, Zadin et al. [47] and Danilov et al. [48] have modelled ionic transport in batteries mathematically using the Nernst-Plank equation. In the model, they apply fixed concentration gradient boundary conditions at the electrode-electrolyte boundaries, where the gradients' value is calculated using a given electric current, assuming diffusion-dominated transport through the boundary. This approach is also implemented in the present work.

Successful simulations of ion transport in LMBs exist. Duczeka et al. [30] have simulated the potential and species distribution in Li-Bi LMB in 1D using OpenFOAM finite volume method-based solver. The specie transport model was also based on the Poisson-Nernst-Planck equations with enforced electroneutrality, similar to the molten salt model presented in this thesis. The results were verified in a comparative study against COMSOL Multiphysics [45] finite element method-based solver results, showing a great correspondence between the methods. Experimentally obtained values were also used to assess the quality of the results, proving a fairly good fit.

Zhou et al. [32] have created a 2D multi-field coupled model for the Li-Bi LMB that included thermal convection, electro-vortex flow, solutal convection, diffusion, and electromigration (governed by the Nernst-Planck equation). Simulated discharge curves were verified against the experimental ones with small error, which is thought to mostly be caused by insufficient accuracy of material properties. They have concluded that thermal convection was the dominant flow mechanism in the molten salt electrolyte of a Li-Bi LMB cell and thus had the greatest contribution to the transport of ions. This notion is widely accepted in the literature [11, 13, 30]. This implies that, in general, convection can not be simply neglected unless one aims to isolate the effects of another mass transport mechanism for various purposes.

Rolle et al. [49] have investigated the influence of species-specific properties like valence and diffusivity on macroscopic electromigration. This was done by conducting both physical experiments and simulations (with equivalent setups) observing the displacement of tracer plumes under the influence of an electric field. Although the model presented in the article somewhat differs from the main molten salt model showcased in the present work, the results of the article were used to test the created model and validate that the modelled transport behaves in an expected way, by creating a similar solver with more relaxed electroneutrality requirement and comparing the results. This approach was chosen due to the fact that more suitable results of models of electrokinetic transport in molten salt systems and their comparison to physical experiments with as much data presented were not found. Therefore this article shall be discussed in more detail.

An illustration of the experimental setup used in the article inspired by the diagram created by Rolle et al. [49] is presented in Figure 2.2. The setup used consisted of a 30 cm long compartment filled with a porous glass medium placed in a glass tank filled with an aqueous electrolyte. The volumetric porosity, or ratio of void space and total available space in the compartment, was equal to 0.4. Two electrodes were placed one on each side of the compartment 40 cm apart with a potential difference of 200 V. The potential difference was measured across 28 cm. The different tracer compounds investigated and their valences were permanganate (-1), Allura Red (-2), and New Coccine (-3). Four different electrolyte composition scenarios were considered.

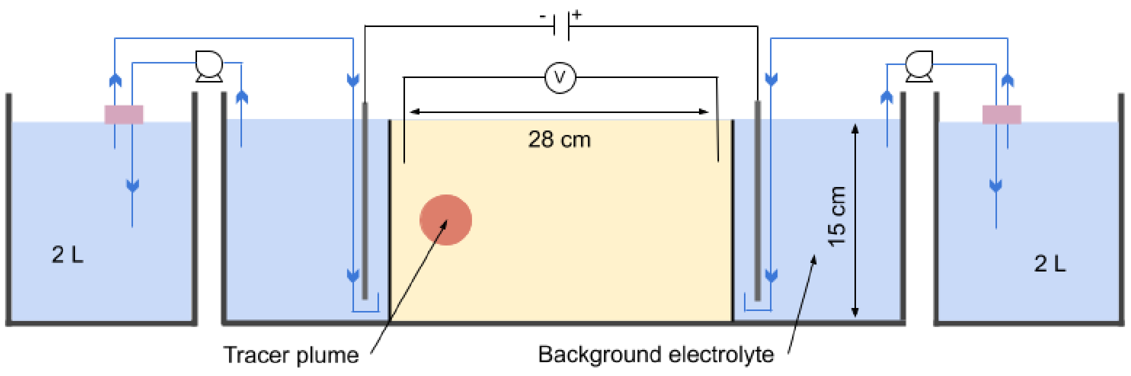


Figure 2.2: Schematic diagram showing the experimental setup for the work conducted by Rolle et al. [49], inspired by the diagram presented by Rolle et al. [49].

The model used for simulations was based on the Poisson-Nernst-Planck equations (3.9) (3.15) (3.16) (notice that the absolute permittivity of the system should reflect the presence of a porous material). The simulations were performed with NP-Phreeqc-EK, which is a sequential coupling of COMSOL Multiphysics [45] and PhreeqcRM [50]. The first is used to solve the Poisson-Nernst-Planck equations, while the second one can be used for describing many different equilibrium and kinetics-controlled reactions. The simulation domain was 2D spanning over 28x15 cm discretized into 13400 triangles. The initial radiuses of the tracers drop were 2.15 cm. The tortuosity of the porous material was used as a fitting parameter, with the best correspondence between simulation and experimental results achieved with tortuosity values of 1.3-1.43.

The results of this study have confirmed the validity of the Poisson-Nernst-Planck model for multicomponent electrokinetic transport of ions in dilute aqueous solutions in porous media. They have also provided evidence of the effects of diffusivity and ion valence in electromigration. These properties impact the mass distribution, mass spreading in both longitudinal and lateral directions, plume displacement velocities, shapes of the plume fronts and tails, and dilution of the injected solution. These differences depend on the charge of the ionic species, their mobility, as well as their coulombic interactions, and the fulfilment of the electroneutrality condition on the pore scale [49].

3 Theory

The transport of ions in a solution is a complex phenomenon that depends heavily on the properties of the system and the ions themselves. No analytical model capturing this process is therefore plausible, making approximate numerical solutions of governing equations the closest alternative. Thus, the theoretical background is of vital importance to correctly apply the model and interpret the results.

In the following section, background on the physical phenomena and an overview of the governing equations are presented.

3.1 Dissolved species transport

The dissolved species transport can be modelled by describing the concentration as a function of time and space employing a general mass balance equation [51]:

$$\frac{\partial c_i}{\partial t} + \nabla \cdot \mathbf{J}_i^{tot} = r_i, \tag{3.1}$$

where c_i is the molar concentration of dissolved component i, t is time, $\mathbf{J}_i^{tot}$ is the total mass flux of i and r_i is the source/sink term, associated with chemical reactions.

Mass transport mechanisms

The total mass flux, $\mathbf{J}_i^{tot}$, of an interacting specie i in a solution under an applied electric field, can be described by the extended Nernst-Planck equation [52]:

$$\mathbf{J}_i^{tot} = -u_i c_i \nabla \mu_i + \mathbf{v} c_i = -D_i \nabla c_i - D_i c_i \nabla \ln \gamma_i - z_i u_i F c_i \nabla \phi + \mathbf{v} c_i, \tag{3.2}$$

where u_i is the mobility of component i, $\nabla \mu_i$ is the gradient of its electrochemical potential, D_i is denoting the diffusion coefficient of the component i in the liquid phase, γ_i is its activity coefficient, z_i is the charge of the component, F is the Faraday constant, R is the universal gas constant, T is the absolute temperature, ϕ denotes the electric potential and v is the velocity of the bulk fluid. Mobility u_i relates to how fast an ion moves in response to an electric field. It can be related to the diffusivity in dilute solutions with the Nernst-Einstein equation [43]:

$$u_i = \frac{D_i}{RT}. \tag{3.3}$$

The expression (3.2) is valid for all mobile species in ideal systems (no ion-ion interactions etc [52]). For dilute solutions with low ionic strength, $I = \frac{1}{2}\sum_i c_i z_i^2$, in isothermal conditions, the gradients of ionic strength can be assumed small. This means that the gradients of the logarithms of the activity coefficients and their contributions to the total flux are small and can be neglected, resulting in the following expression:

$$\mathbf{J}_i^{tot} = \underbrace{-D_i \nabla c_i}_{\mathbf{J}_i^{dif}} \underbrace{-z_i u_i F c_i \nabla \phi}_{\mathbf{J}_i^{mig}} \underbrace{+\mathbf{v} c_i}_{\mathbf{J}_i^{adv}}, \tag{3.4}$$

where $\mathbf{J}_i^{dif}$ is the diffusive flux, $\mathbf{J}_i^{adv}$ is the advective flux and $\mathbf{J}_i^{mig}$ represents the migration flux.

Diffusion of a fluid property (such as temperature, solute concentration and others) is its' net flux realized by thermal vibration in said fluid. The driving force for chemical compound diffusion is the chemical activity gradient, with specie travelling from regions with higher concentration to ones with lower concentration, reducing the Gibb's free energy for the solution [53]. Advection describes the mechanical transport of a fluid property along the bulk fluid flow. The driving force behind the fluid flow itself is the pressure gradient, which in turn can have a plethora of different causes, such as temperature differences, concentration differences and others [53]. Finally, migration is defined as the movement of ions in response to an electric field [43]. For this transport mechanism, the electric potential gradient is the driving force. In electrochemistry, this gradient is often caused by surfaces being polarized, inducing an electric field.

The velocity required for solving this equation for systems that include convection can be found by solving the Navier-Stokes equations [43]. For the purposes of describing migration, the velocity is set to zero for all cases in this work.

The electric current, **i** realized by the movement of charged particles in the solutions can be calculated in the following way:

$$\mathbf{i} = F \sum_i z_i \mathbf{J}_i^{tot} = -F \sum_i z_i D_i \nabla c_i - F^2 \nabla \phi \sum_i z_i^2 u_i c_i + F \mathbf{v} \sum_i z_i c_i. \tag{3.5}$$

The ionic conductivity of the solution, κ, is often introduced to this equation:

$$\kappa = F^2 \sum_i z_i^2 u_i c_i. \tag{3.6}$$

The conductivity of a material can also be derived from Ohm's law:

$$\kappa = \frac{\mathbf{i}}{-\nabla \phi}. \tag{3.7}$$

Electric potential in dilute solutions

Since ions carry charge they affect the electric field and, in multicomponent systems, experience coulombic interactions with each other. These effects are necessary to consider in an accurate model of transport. This means that equation (3.4) must be solved for all chemical components simultaneously and the effect of charge distribution on the electric field must be accounted for. The relation between the electric field and charge (or ionic specie concentration) distribution in homogeneous systems can be described with Poisson's law [43, 54]:

$$\nabla \cdot (\epsilon \nabla \phi) = -F \sum_{i=1}^{N} z_i c_i = -\rho_e, \tag{3.8}$$

where ϵ is the absolute permittivity of the system, N is the total number of charged species in the fluid and ρ_e is commonly referred to as charge density. For systems with uniform permittivity, this equation can be rewritten as follows:

$$\nabla^2 \phi = -\frac{\rho_e}{\epsilon}. \tag{3.9}$$

For dilute systems, local deviations from electroneutrality are accepted due to low concentrations. This can, for example, be inside the double layer near electrodes or other boundaries. This means that equation (3.9) should be used for modelling such systems on the micrometre scale, where deviations from electroneutrality are important and must be resolved. This provides a two-way coupling between the electric field and ion concentration distribution and ensures that their effect on each other is captured. Thus, to apply this equation when modelling, one has to spatially resolve the areas with deviations from electroneutrality, such as the electrical double layer. Still, the electroneutrality condition must be met at the scale of the total system. This condition can mathematically be expressed in the following way [43]:

$$\sum_{i=1}^{N} z_i c_i = 0. \tag{3.10}$$

This equation is also valid for concentrated electroneutral solutions.

Molten salt considerations

The Nernst-Planck equation (3.4) is, in principle, valid only for dilute solutions. There are several reasons for this. Firstly, migration and diffusion fluxes must be defined with respect to some average velocity of the fluid. In dilute systems, only the solvent velocity contributes to this average velocity, but that is not the case in concentrated solutions. In addition, as previously stated, the simplification of equation (3.2) leading to a more commonly known and used equation (3.4) is based on the assumption of a dilute solution. The driving force for diffusion is the gradient of chemical potential which is related to the activity gradient. The activity gradient is equal to the concentration gradient only in extremely dilute solutions. Finally, the transport properties used only consider interactions and friction between a solute component and the solvent, while inter-component interactions are substantial in concentrated solutions [43].

The equations for material balance (3.1), current flow (3.5), and electroneutrality (3.10) remain valid for concentrated solutions, but the described issues demand changes to the flux equation. The mentioned difficulties can be avoided by replacing the Nernst-Planck equation (3.4) for the total flux with the multicomponent diffusion equation [43]:

$$c_i \nabla \mu_i = \sum_j K_{ij}(\mathbf{v}_j - \mathbf{v}_i) = RT \sum_j \frac{c_i c_j}{c_T \mathscr{D}_{ij}}(\mathbf{v}_j - \mathbf{v}_i), \tag{3.11}$$

where K_{ij} are friction or interaction coefficients, $\mathbf{v}_i$ is the velocity of component i (not for each molecule but average for the species), $c_T = \sum_i c_i$ is the total concentration, $\mathscr{D}_{ij}$ is a diffusion coefficient that describes the interaction of the components i and j. This equation is more general than the Nernst-Planck equation since it relates the driving force (the left-hand side of the equation) to a linear combination of resistances for the movement of i (the right-hand side of the equation) instead of just one resistance coming from the solvent. In order to use this equation for each of the components together with the material balance equation (3.1), they must be inverted to express the components' flux densities in terms of the driving forces. These inversions and the resulting expressions get increasingly more convoluted with every new component introduced to the system. More components also imply more unknown properties like $\mathscr{D}_{ij}$, which can be difficult or impossible to measure.

Still, equation (3.4) is generally used for most cases with migration transport and is very prevalent in the literature. This is due to its' simplicity and relative accuracy even with broken assumptions [43]. It is also more convenient to use because the physical properties it requires are more often known and are easier to measure than the ones needed when using equation (3.11).

Molten salts can be considered to be highly concentrated solutions. Thus, the condition of electroneutrality for the mixture must apply both locally and at the continuum scale, as the net charge of the molten salt electrolyte should be zero and high concentrations of different ions are available to balance each other out locally [30]. These solutions are not necessarily homogeneous (especially when it comes to electrical properties), since concentration gradients can affect the properties like, for example, conductivity, significantly. When ionic transport is present, these variations in properties can also change with time. This makes Poisson's equation (3.8) inapplicable for these solutions. Therefore, a different approach must be used to determine the electric potential distribution.

As ions travel they transport charge. This charge must be conserved. One can rewrite the mass conservation equation (3.1) as a charge conservation equation by multiplying it with $z_i F$ (summing up the contribution of all chemical components for convenience):

$$\frac{\partial}{\partial t} F \sum_i z_i c_i = -\nabla \cdot F \sum_i z_i \mathbf{J}_i^{tot}. \tag{3.12}$$

Under the assumption of electroneutrality, the left-hand side of the equation is equal to zero. Inserting equation (3.5) into equation (3.12) transforms the expression to:

$$\nabla \cdot (\kappa \nabla \phi) + \underbrace{F \sum_i z_i \nabla \cdot (D_i \nabla c_i)}_{\text{Diffusion current}} + F \sum_i z_i \nabla \cdot (\mathbf{v} c_i) = 0, \tag{3.13}$$

which can be used for calculating the potential distribution in an electrolyte. The magnitude of the electric field is not its' only important parameter for migration, its direction and variation thereof must also be considered [55].

Assuming both bulk and local electroneutrality, equation (3.13) can also be used to describe dilute aqueous solutions with inhomogeneous conductivity caused by the concentration variations throughout the volume. With no fluid flow present (third term of equation (3.13)) and a negligible sum of diffusion currents (second term of the equation), this equation can be reduced to:

$$\nabla \cdot (\kappa \nabla \phi) = 0. \tag{3.14}$$

The sum of diffusion currents is often negligible when the contributions of the different compounds cancel each other out. This can be assumed to be the case when the diffusion coefficients of the compounds are approximately equal.

Porosity considerations

The porous diaphragm located in the molten salt electrolyte layer of the Na-Zn LMB affects the transport of ions. Instead of introducing an actual porous geometry to the simulations, one can look at the transport on the Darcy scale, where the porous material is treated as a volume, which affects the transport properties. Then, instead of resolving the transport at the pore scale, one can capture the effects of the porous material on the transport by accounting for them in the governing equations.

When describing the macroscopic mass transfer through a porous medium instead of free space on the Darcy scale, equation (3.1) becomes:

$$\frac{\partial \omega c_i}{\partial t} + \nabla \cdot \mathbf{J}_i^{tot} = r_i, \tag{3.15}$$

and equation (3.4) transforms into:

$$\mathbf{J}_i^{tot} = \underbrace{-\omega D_i' \nabla c_i}_{\mathbf{J}_i^{dif}} \underbrace{-\omega z_i u_i' F c_i \nabla \phi}_{\mathbf{J}_i^{mig}} \underbrace{+\mathbf{q}^* c_i}_{\mathbf{J}_i^{adv}}, \tag{3.16}$$

where ω is the porosity accessible to the fluid, D_i' is the effective diffusion coefficient, u_i' is the effective ion mobility and q^* is the specific solute flux. Note that here it is implied that the volumetric accessible porosity (volume fraction of accessible empty space in the material) is equal to the areal accessible porosity (ratio between the accessible pore area and total area of a cross-section of the material) everywhere throughout the material. Otherwise, the accumulation term in equation (3.15) would be affected by the volumetric accessible porosity, while the macroscopic fluxes in equation (3.16) would depend on the areal accessible porosities of the media. In these equations describing the total macroscopic transport through the material one value of the porosity is enough to consider. This is due to the areal porosities of the areas normal to the transport direction averaged over the thickness of the material being equal to the volumetric porosity of the material.

The macroscopic diffusion coefficient D_i' can be calculated using the following relation: $D_i' = D_i/\tau^2$, where τ^2 is tortuosity, which has many definitions, the one used here being:

$$\tau^2 = \left(\frac{L_s}{L}\right)^2, \tag{3.17}$$

where L_s is the length of the path taken by species through the porous medium and L is the distance between the paths' start and end [49, 56].

The mobility u_i' can be determined by inserting the macroscopic diffusion coefficient D_i' into the Nernst-Einstein relation (3.3).

Specific solvent flux can also be understood as the average velocity of the fluid across the whole area normal to the direction of the flow, including the area unavailable to the flow, which equals to the average velocity of the fluid multiplied by the areal porosity of the material, which can often be assumed to be equal to volumetric porosity.

By affecting the transport of ions, the presence of a porous matrix also alters the current density:

$$\mathbf{i} = F \sum_i z_i \mathbf{J}_i^{tot} = -F \sum_i z_i \omega D_i' \nabla c_i - F^2 \nabla \phi \sum_i z_i^2 \omega u_i' c_i + F \mathbf{q}^* \sum_i z_i c_i. \tag{3.18}$$

The ionic conductivity through a saturated insulating porous medium with no surface conductance, κ', can be calculated as follows [57]:

$$\kappa' = F^2 \omega \sum_i z_i^2 u_i' c_i. \tag{3.19}$$

It is important to note, that on the pore scale even in a porous medium, the regular governing equations (3.1), (3.4), and (3.13) are valid because no effects are averaged over the system and are rather resolved locally throughout the entire domain.

3.2 Summary

In this section, the necessary theoretical background for the simulations conducted in the present work is described. This includes all the equations needed for the created models.

Altogether, the concentration distributions of ionic species in dilute aqueous solutions can be described using equations (3.1) and (3.4). The electric potential distribution in those cases can be described using many different equations depending on the assumptions made. This includes equations (3.9) (allowing for local deviations from electroneutrality), (3.13), and (3.14) (enforcing electroneutrality everywhere throughout the solution). Although equation (3.9) can provide a more accurate picture of potential distribution when modelling ionic transport in dilute aqueous solutions, equation (3.14) was used. This was done to avoid the need to resolve the electrical double layer, which is necessary when using equation (3.9), because it is highly computationally demanding due to the very low scale of the double layer relative to the size of the modelled systems.

The appropriate scaling of the equations and the parameters should be done when dealing with transport through a porous medium, replacing equations (3.1) and (3.4) with equations (3.15) and (3.16). The presence of a porous matrix must also be reflected in calculations of ionic conductivity, replacing equation (3.6) with equation (3.19).

The concentration distributions of ionic species in molten salts can be described using equation (3.11). However, due to the complexity of this approach and its' demand for parameters commonly unknown, its use was omitted. Instead, equations (3.1) and (3.4) are commonly used, due to their simplicity and their relative accuracy even under broken assumptions [43]. Potential distribution equation (3.13) is well-suited for molten salt modelling, enforcing electroneutrality on both the local and the Darcy scales. Again, porosity must be accounted for when necessary, scaling the equations and the parameters used.

4 Numerical approach

Flow and transport phenomena can often be described using systems of non-linear partial differential equations (PDEs), which seldom have analytical solutions. It is, however, possible to approximate the solutions by employing numerical methods with the help of computers.

These approximations rely on a process called discretization, during which a continuous PDE solution is split into a finite number of values existing at discrete locations on the time and space domains. Many discretization methods exist, such as Finite Element Method (FEM), Finite Difference Method (FDM), and Finite Volume Method (FVM). The FVM is one of the most popular methods for CFD for multiple reasons. Its' main advantage is that the produced discretization is automatically conservative due to the mathematical formulation of the method, which mirrors the principles of the conservation laws. The following section will present a brief overview of relevant CFD principles and describe the applied algorithms.

4.1 Finite volume method

The Finite Volume Method is a discretization method used to approximate PDE solutions by transforming them into systems of algebraic equations, which can then be solved by a computer. This is done over the entire computational space and/or time domain by dividing it into discrete volumes(cells)/intervals and solving the conservation equations for each of them [58].

As the first step of the FVM process, the solution domain is divided into small contiguous non-overlapping control volumes (CVs), or cells, illustrated in Figure 4.1.

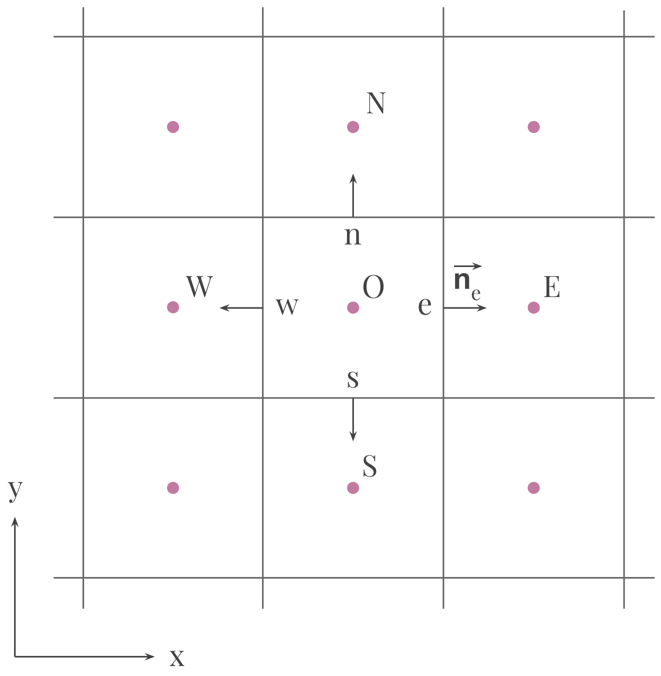


Figure 4.1: An illustration of a 2D meshed solution domain. Computational nodes in the cell centres are shown in pink and are denoted with capital letters. Cell faces centres of the CVs/cells are denoted by lowercase letters. Vector $\vec{\mathbf{n}}_e$ is an example of a vector normal to face hosting the point e (also referred to as face e).

Thereafter, the equations are discretized. A feature specific to the FVM is the fact that it considers the integral form of the conservation equations. For a generic quantity ψ, the integral form of its' conservation equation can be written as:

$$\underbrace{\int_V \frac{\partial(\rho\psi)}{\partial t} dV}_{\text{transient tern}} = \underbrace{\int_S \rho\psi\mathbf{v}\cdot\mathbf{n}dS}_{\text{convective tern}} - \underbrace{\int_S \Gamma\nabla\psi\cdot\mathbf{n}dS}_{\text{diffusive term}} + \underbrace{\int_V r_\psi dV}_{\text{source term}}, \tag{4.1}$$

where ρ is density, v is velocity, $\mathbf{n}$ is a vector normal to surface S, Γ is the diffusivity for the quantity ψ, r_ψ is a source or sink of ψ and V is volume.

The conservation equation (4.1) applies to the entire domain and to all CVs individually. The global conservation equation can be obtained by summing the integrals for all CVs since they are contiguous and negative surface flux out of one CV cancels out with that same flux entering the neighbouring CV that shares a face with it.

The total surface integral for each of the CVs can thus be expressed as a sum of integrals over all faces of that CV:

$$\int_S \rho\psi\mathbf{u}\cdot\mathbf{n}\,dS - \int_S \Gamma\nabla\psi\cdot\mathbf{n}\,dS = \sum_j \int_{S_j} \rho\psi\mathbf{u}\cdot\mathbf{n}_j\,dS_j - \sum_j \int_{S_j} \Gamma\nabla\psi\cdot\mathbf{n}_j\,dS_j \tag{4.2}$$

where j is the number of faces of the relevant cell. The surface and volume integrals for each cell can then be approximated using different numerical methods like midpoint or trapezoid rules transforming them into a system of algebraic equations. This system of equations can be written in matrix form and solved using given boundary conditions. Since these matrices can normally consist of a high number of values direct inversion methods like Gaussian elimination are not optimal to use. Therefore, different more effective iterative solvers have been developed, such as GAMG, PCG, PBiCG and more. GAMG solver is used for simulations conducted in this project.

Integral approximation

Since the exact values of the variable ψ across all boundaries of a given CV are not known, the integrals in the equation (4.2) have to be approximated. The only known ψ values are in the cell nodes, calculated from the previous cell (or from the boundary conditions). In order to approximate the integrals the unknown values of the fluxes, and thus the conserved variables, at cell faces are required (for example, at the cell face centres). An example of such surface integral approximation can be the midpoint rule, illustrated here for a flux component f (which can be convective or diffusive by nature), normal to a surface e with an area S_e, one of the j total surfaces:

$$\int_{S_e} f dS = \overline{f_e} S_e \approx f_e S_e, \tag{4.3}$$

where subscript e indicates the centre location on a specific face of the CV (see Figure 4.1) and $\overline{f_e}$ is the average value of flux f over the entire face e. The unknown cell face centre f_e flux value needed to approximate the integral can be obtained from the known cell node value using interpolation schemes [59].

Volume integrals (like the one in equation (4.2)) must also be approximated since again, the values of the variables throughout the entire cell volume are unknown, with only the nodal values available. This can easily be done without interpolation, by approximating the average integrand value over the cell volume to be equal to the known cell centre value [59]:

$$\int_{V_i} q dV = \overline{q} V_i \approx q_O V_i, \tag{4.4}$$

where i indicates the i-th cell with volume V_i, q is the integrand (which can be the source/sink term as in equation (4.2)), and q_O represents the cell centre point value of q (see Figure 4.1). If q is constant or varies linearly throughout the CV this approximation actually provides the exact solution for the integral. This is a second-order approximation, thus, otherwise, a second-order error will be present. Approximation methods of higher orders exist. They produce results with higher accuracy but require q values at more locations than just the cell centres. This process again requires interpolation in order to acquire those values [59].

Interpolation schemes

Numerous interpolation schemes of varying accuracy exist and in FVM are used for approximating the unknown values needed for integral calculations. The discussed fluxes f normally contain several variables or variable gradients, like density, velocity, and conserved quantity ψ for convective flux. Often variables like density or diffusivity are assumed to be constant for the entire volume, making their values on the cell faces known. Sometimes the velocity values are also known. However, the value of the generic quantity ψ, the conservation of which is described by the governing equation (4.2) is what we normally are trying to determine. Thus the cell face values of ψ and its' gradient normally have to be expressed in terms of cell centre values using interpolation.

The schemes used for interpolating the cell face values of the gradients of the relevant quantity are often (and further in this thesis) referred to as gradient schemes. Equivalently, Laplacian values are estimated using Laplacian schemes, divergence - with divergence schemes. The schemes employed for estimating the cell face value of the quantity ψ itself, are referred to as simply interpolation schemes.

Central-difference scheme (CDS) or linear interpolation is the simplest second-order interpolation scheme that is widely used in CFD [60]. It is also the interpolation scheme used predominantly in the simulations conducted for this project. As an example of the way these schemes operate mathematically, this scheme will be described closer. Applying this method, the desired value of ψ_e located on a cell face can be expressed in terms of the known cell node values in the following way:

$$\psi_e = \psi_E \Lambda_e + \psi_O (1 - \Lambda_e), \tag{4.5}$$

where Λ_e is defined as

$$\Lambda_e = \frac{x_e - x_O}{x_E - x_O}, \tag{4.6}$$

where x indicates the x-axis position for a point indicated by the subscript (see Figure 4.2) [59]. This, of course, can be done in other directions equivalently. Notice that this approach approximates the variable profile between the centroids with a straight line, while higher-order schemes employ higher-order polynomials. This method, as all other second-order schemes, may produce oscillatory or unstable solutions. The stability of such methods oftentimes comes at the expense of accuracy [61]. For that reason, other higher-order schemes are often used.

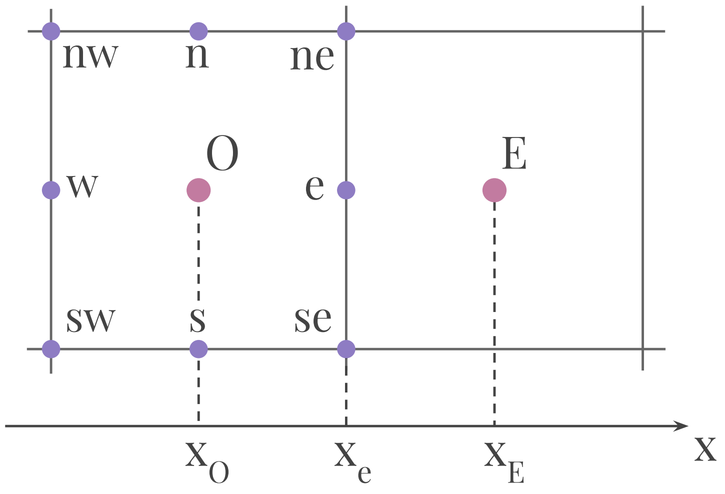


Figure 4.2: A close-up illustration of the 2D meshed solution domain presented in Figure 4.1 illustrating the used notation.

4.2 Time advancements

For transient simulations, time discretization of the governing equations is necessary in addition to the previously discussed space discretization. In CFD time is treated as an additional coordinate along which the integral of the transient term is evaluated.

A governing equation describing transient behaviour for a general case takes on the following form:

$$\underbrace{\frac{\partial(\rho\psi)}{\partial t}}_{\text{transient term}} + \mathcal{L}(\psi) = 0, \tag{4.7}$$

where the function $\mathcal{L}(\psi)$ is a spatial operator that includes non-transient terms (advection, diffusion, sources and others, presented in equation (4.1)) [58]. After spatial discretization around the centroid, O, of a CV cell with volume V_O, this expression becomes

$$\frac{\partial(\rho_O\psi_O)}{\partial t}V_O + L(\psi_O^t) = 0, \tag{4.8}$$

where $L(\psi_O^t)$ is the spatial discretization operator at some reference time t. It is common to treat the transient term using the finite difference method because the grid in the transient space is structured. Further, the space operator $L(\psi)$ is discretized as described before at some time t, while the transient term is estimated using one of the various existing transient schemes [58]. The most relevant of these schemes for the present work are described next.

4.2.1 Forward Euler Scheme

When using the Forward Euler scheme, the expansion of the transient term around time t is done in a forward manner. The first-order approximation of the time derivative of some function K using a forward time step Taylor expansion of K can be expressed as follows [58]:

$$\frac{\partial K(t)}{\partial t} = \frac{K(t+\Delta t) - K(t)}{\Delta t} + \mathcal{O}(\Delta t), \tag{4.9}$$

where Δt is the time step and $\mathcal{O}(\Delta t)$ is the error term resulting from the higher order terms being neglected. Using this approach on the transient term of the relevant equation (4.8), provides us with the discretized equation:

$$\frac{(\rho_O\psi_O)^{t+\Delta t} - (\rho_O\psi_O)^t}{\Delta t}V_O + L(\psi_O^t) = 0. \tag{4.10}$$

Here, all the spatial terms are evaluated at the old time t, making it possible to compute the value of ψ_O at time $t+\Delta t$ explicitly, based on those known old values, without the need to solve a system of equations. Therefore, this method belongs to the class of explicit transient schemes and is also referred to as the Explicit Euler Scheme [58]. Such methods have high computational efficiency. However, few commercial CFD software use this approach, due to its' stability issues. These methods are only conditionally stable, with the conditions limiting the size of the time step severely, especially for cases with high spatial resolution. When dealing with more complex problems with various transport mechanisms present, the choice of the time step becomes far from straightforward. Simultaneously, an inadequate choice of Δt will cause oscillations in the results. Thus, this method is not recommended for general transient problems [62].

4.2.2 Backward Euler Scheme

The Backward Euler scheme is based on time discretization in the opposite direction from the Forward Euler scheme. Here, the approximation of the time derivative uses Taylor expansion of the function at time $t - \Delta t$ as the starting point, taking on the following form [58]:

$$\frac{\partial K(t)}{\partial t} = \frac{K(t) - K(t - \Delta t)}{\Delta t} + \mathcal{O}(\Delta t). \tag{4.11}$$

Applying this to the governing equation (4.8), the fully discretized equation becomes

$$\frac{(\rho_O \psi_O)^t - (\rho_O \psi_O)^{t-\Delta t}}{\Delta t} V_O + L(\psi_O^t) = 0. \tag{4.12}$$

Thus, evaluating the ψ field at a new time step requires solving a system of equations. This type of scheme is called implicit. This method is always stable, regardless of the size of the time step, enabling the solution to proceed rapidly in time. However, this scheme is still low-order, producing low accuracy results, unless a small time step is used [58].

This scheme was used in all simulations conducted in the present work due to its' unconditional stability and adequate accuracy with the use of appropriate time steps.

4.3 Meshing

The process of dividing the computational domain into discrete non-overlapping consecutive control volumes is called meshing. This process creates a grid, or mesh, discretizing the space. This enables one to apply the finite volume method for solving equations over the domain and conducting CFD simulations. This step is very important since the size of the error in FVM computations, as well as computational time depends on the size and shape of the grid. In general, using a coarser mesh with large cells will yield less accurate approximation results, while applying a finer mesh will result in more accurate results, but will require more computations.

4.4 Convergence

A numerical model is said to be convergent if and only if the solution of the discretized equations asymptotically approaches some fixed value as the spacial grid cell size and time step used for discretization approach zero. The solution is called consistent when that value is equal to the exact solution of the differential equation [59, 63].

A common way to check if a model is convergent is to acquire multiple solutions using different values of spatial and temporal steps and then compare the produced results. The model has converged if further refinement of time and distance steps does not result in a significant change in the results. This ensures that there is no dependence of the solution on the used discretization [59, 63].

4.5 Error and uncertainty

Simulations can be a useful tool to provide insight into various transport phenomena, however, computational modelling is not free of errors and uncertainties. Understanding the source of these deficiencies and their magnitude is therefore important.

As done by Versteeg and Malalasekera [62], **error** in a CFD model can be defined as a recognizable deficiency that is not a result of a lack of knowledge. These deficiencies include *numerical errors*, such as discretization and roundoff errors, as well as convergence errors of iterative methods, *coding errors* resulting from mistakes in the software, and *user errors* due to incorrect use of the software.

Uncertainty on the other hand, are potential deficiencies in a model that are stemming from a lack of knowledge [62]. Typical sources of uncertainty can be identified as *input uncertaity* caused by a lack of available information about or approximation of boundary conditions, domain geometry, compound properties and other aspects of the model and *physical model uncertainty* that are a result of misrepresentation of the transport processes or their simplification in the model.

4.6 Summary

In this chapter, an introduction to numerical methods for solving systems of partial differential equations, including the finite volume method are presented. These tools allow one to approximate the solution to the governing equations and create simulations. The FVM solves the integral form of the governing equations, providing a solution where the relevant quantities are conserved for each control volume cell across the computational domain. The coupled set of algebraic equations is then written in a matrix form, which is in this project solved using the geometric agglomerated algebraic multigrid (GAMG) solver [64]. A review of the algebraic multigrid approach is given by Stüben [65], while Behrens [64] provides a closer overview of GAMG.

The central-difference scheme [60] was the spatial discretization tool of choice for all terms except for migration, which was discretized using the Van Leer scheme [66]. Van Leer method was used for migration terms since they otherwise suffered from oscillations. Van Leer is a second-order scheme, that does not have the same disadvantages as CDS. A review of the Van Leer scheme is provided by van Leer [66].

Temporal discretization is carried out by means of the Backward Euler scheme, ensuring the stability of the solutions unconditioned by the size of the time step.

5 Implementation of the models in OpenFOAM

In order to create simulations based on the discussed governing equations, new OpenFOAM solvers were developed. A solver, in the OpenFOAM sense, is an application for numerically solving a set of governing equations using user-defined properties as well as boundary and initial conditions. In this section, the general structure of OpenFOAM cases and solvers are described. This includes a preexisting OpenFOAM solver for diffusive-advective transport of a scalar (named scalarTransportFoam) since it was used as the basis for the created solvers. This is done to provide some insight on how these solvers are structured and function in general. The developed solvers for electrokinetic transport in aqueous solutions (slayFoam) and molten salts (saltyFoam) are also described.

5.1 OpenFOAM environment

Many software for solving systems of partial differential equations numerically exist. The one chosen for this thesis is OpenFOAM. OpenFOAM is an open-source CFD software. It is commonly used by scientists and engineers for solving various PDE-based problems, which include, but are not limited to fluid dynamics [23].

Firstly, some background on the general structure of OpenFOAM-based simulations shall be explained to provide context. The purpose of all used dictionaries and files is explained as well.

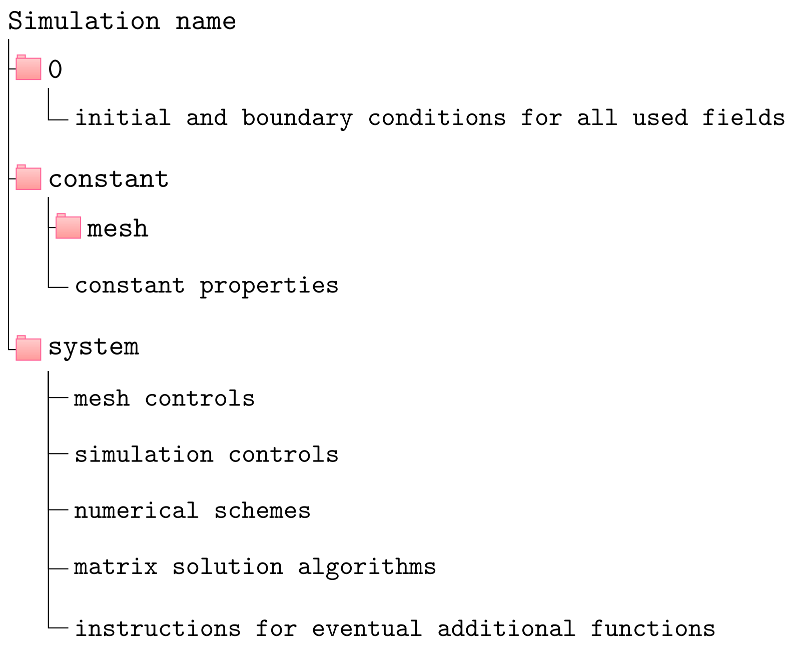

Simulation name
0
initial and boundary conditions for all used fields
constant
mesh
constant properties
system
mesh controls
simulation controls
numerical schemes
matrix solution algorithms
instructions for eventual additional functions

Figure 5.1: Diagram showing a general structure of an arbitrary simulation directory in OpenFOAM.

Figure 5.1 showcases the type of files that can be found in a generic OpenFOAM simulation directory. The three main directories that must be included are "0", "constant", and "system". Directory "0" serves as the initial time step of the simulation, storing initial and boundary conditions for the variables used in the simulation. The "constant" directory stores properties that remain constant during the simulation, such as the eventually generated mesh and physical constants. Finally, the "system" directory stores the necessary instruction files. This includes the controls for the meshing process and the time aspects of the simulation, discretization schemes controls, allowing the user to choose the schemes used for every part of the equations, matrix inversion method controls, and other dictionaries containing instructions for any additional functionalities used. These files can easily be changed by the user and allow for a high level of control over a specific case.

5.1.1 Meshing in OpenFOAM

Many methods for meshing exist. The goal is to choose the meshing method capable of producing accurate enough results, without using unreasonable computational resources. Due to the simplicity of the geometries investigated in the present work, OpenFOAMs' blockMesh is adequate.

The blockMesh utility allows one to generate a cartesian mesh consisting of three-dimensional hexahedral blocks in accordance with the user-defined specifications. It writes out the mesh data for points, faces, cells, and boundaries in a directory called polyMesh that are later used for computations. The polyMesh directory contains the mesh as a set of hierarchical lists that define the computational grid. This way, the mesh is defined by a list of volumes, where each volume is defined by a list of planar surfaces, and subsequently each surface is defined by a list of points. The size, shape and number of divisions of the mesh as well as boundary types are chosen by the user.

5.2 ScalarTransportFoam - the basis for the developed solvers

Many OpenFOAM solvers, including scalarTranportFoam, consist of three main parts collected in a main directory: a file containing the code solving the equations in .C format (scalarTransportFoam.C), a file defining all the necessary fields and constants in .H format (createFields.H) and a Make directory, containing files needed for compiling the code, transforming it into a program one can run.

ScalarTransportFoam is based on the advection-diffusion equation:

$$\frac{\partial T}{\partial t} = \nabla \cdot (D_t \nabla T) - \nabla \cdot (\mathbf{v} T), \tag{5.1}$$

where T is originally temperature but can be replaced with any scalar property of the fluid transported by advective and diffusive fluxes (concentration, for example), D_t is the thermal diffusivity of the fluid, which again can be replaced with diffusivity corresponding with the transported scalar property, and $\mathbf{v}$ is the velocity of the fluid.

The software solves this equation with the FVM by approximating it as a system of algebraic equations written in a matrix form. This form of the equation (5.1) is located in the scalarTransportFoam.C file of the solver, written in a way illustrated in a code listing 5.1, with "//" indicating a comment explaining the code. The full scalarTransportFoam.C file, together with all other solver files and three example case setups relevant to this report, can be found in the Appendix.

```
    while (simple.loop())      //time loop
    {
        Info<< "Time=" << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())      //correction loop
        {
            fvScalarMatrix TEqn
            (
                fvm::ddt(T)                        //accumulation
                + fvm::div(phi, T)                 //advection
                - fvm::laplacian(DT, T)            //diffusion
             ==
                fvOptions(T)
            );

            TEqn.relax();
            fvOptions.constrain(TEqn);
            TEqn.solve();
            fvOptions.correct(T);
        }

        runTime.write();
    }
```

Listing 5.1: A part of the scalarTransportFoam.C file that solves equation (5.1) for all mesh cells and time steps one by one (hence the while (simple.loop()) for time steps).

It is important to discuss the OpenFOAM notation for the matrix form of the equations to draw parallels between the code and the advection-diffusion equation. fvm is a namespace that contains the code for calculating the matrix coefficients for the different discretized operators (here ddt - time derivative, div - divergence, and laplacian is self-explanatory). A namespace fvc (not used here, but equally as important) contains the code for calculating the actual field values for various operators using current values.

The equation to be solved is located inside an inner correction loop located inside the outermost time loop. Thus, this solution is approximated for the whole computational domain for each time step. Within a time step the equation may be solved several times, corrected in each loop to achieve a more accurate result. The number of corrector loops applied is defined by the user. The overall algorithm for scalarTransportFoam, and the developed solvers based on it, is visualized in Figure 5.2.

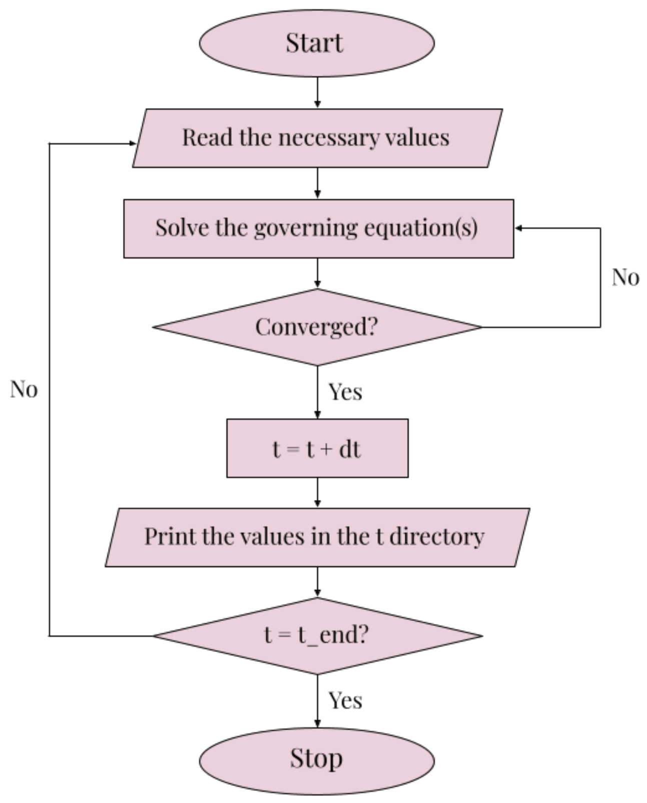

Start
Read the necessary values
Solve the governing equation(s)
No
Converged?
No
Yes
t = t + dt
Print the values in the t directory
t = t_end?
Yes
Stop

Figure 5.2: A flowchart illustrating the algorithm of scalarTransportFoam and the developed solvers.

As illustrated by Figure 5.2, the first step of the solution process is to access all the needed variables and constants, after which the equations are solved for all time steps. The instructions for this process are contained in a header file createFields.H. The type of each variable or constant has to be given. The types used in the present work are volScalarField, surfaceScalarField, volVectorField, dimensionedScalar, and IOdictionary. The names provide some information about the variables. "vol" indicates that the values of the field are given for the cell centres, as opposed to "surface" fields, which contain values projected on the cell faces. "Scalar" and "Vector" indicate whether the values are scalar or vector. Constants are normally defined as dimensionedScalars, including dimensionless constants for which all units are to zero. IOdirectory type is used as a placeholder for files that normally contain values of different parameters. This is done, so that the parameters can be easily changed in the simulation directory, without having to change and compile the solver files.

Additional information about the variables besides their type is required in the header file. This is illustrated using a concentration field as an example, presented in the code listing 5.2. This looks slightly different for dimensionedScalars because they are constant. There, only the object name, dimensions, and the location file have to be listed besides the constant's type and name. In scalarTransportFoam the IOdictionary is defined in a way similar to the fields, except it is located in the constant directory instead of the timeName directory, doesn't have to be written for each time step, and must only be read if modified.

Once a variable is defined, upon running the simulation, OpenFOAM will search for its' value in the indicated directory.

```
    volScalarField C1                 //variable type and name
    (
        IOobject
        (
            "C1",                     //file name containing values
            runTime.timeName(),       //directory to search for C1 in
            mesh,                     //indicating mesh object
            IOobject::MUST_READ,      //frequency of value reading
            IOobject::AUTO_WRITE      //frequency of value output
        ),
        mesh
    );
```

Listing 5.2: Example of a field definition located in createFields.H file.

The objects defined for scalarTransportFoam solver, their types, and locations are presented in Table 5.1.

Table 5.1: Information about the objects defined in the createFields.H file of the scalarTransportFoam solver.

Object	Symbol	Type	Location
Temperature	T	volScalarField	time directory
Velocity	U	volVectorField	time directory
Transport properties	transportProperties	IOdictionary	constant directory
Conductivity	DT	dimensionedScalar	transportProperties

In order for the scalarTransportFoam.C file to access the information about the variables stored in the createFields file, it must be called in the code. This is done prior to the introduction of the equations by adding a line: "#include "createFields.H" ". In the same way, other instructions, or code, need to be included, ensuring the functioning of the time and correction loops, different fvm and fvc operators, and more.

5.3 SlayFoam solver for implementation testing

To test the created modelling approach, a slayFoam solver was developed and the simulations from Scenario 1 presented by Rolle et al. [49] were replicated. SlayFoam solver is essentially based on the same model as the saltyFoam solver, except the electroneutrality condition is less strict (making it more suitable for modelling aqueous solutions) and diffusion current is neglected.

The article [49] investigates ion transport in a dilute aqueous solution through porous media. Thus, equations (3.9),(3.15) and (3.16) can be employed to model this process resulting in a two-way coupling between the ion concentrations and the electric field. This is done by the articles' authors. In order for a simulation based on these equations to produce a stable solution, the electrical double layer must be resolved. This is computationally costly when working on systems 0.28 m by 0.15 m in size, since the electrical double layer is very small for such systems, and would require a very fine mesh to be resolved. Therefore, the equation (3.9) was replaced with equation (3.13), avoiding this issue.

There are two additional differences between the slayFoam model used during the testing and the one presented in the article [49]. Firstly, the articles' authors modelled all 16 of the ions present in the solution. Doing so is not strictly necessary to get qualitatively similar results and capture the main effects of the electrokinetic transport. Simply neglecting the presence of background ions would not result in a physically attainable model, due to a high inhomogeneity of the systems' properties. Thus, the background ions present in the electrolyte were taken into account mathematically by adding a quantitative contribution they had to the total conductivity of the solution κ or kappa in code. This background electrolyte conductivity was given the name kap.

The final difference between the models lies in the fact that the original model from the paper [49] takes reactions and interactions between the ionic compounds into consideration, and the developed model does not. These changes were introduced to reduce the computational demand but will result in less accurate results.

Table 5.2: The notation of variables used in solver codes and the corresponding notation used in the equations in the report together with the physical meaning behind them.

Notation in the report	Notation in code	Physical meaning
c_i	C1 and C2	Molar concentration
ϕ	Fi	Electric potential
D_i	DC1 and DC2	Diffusion coefficient
z_i	z1 and z2	Ionic charge
ω	p	Volumetric porosity
κ	kappa	Ionic conductivity
τ^2	tau*tau	Tortuosity
$\frac{p}{\tau^2}$	Gamma	Coefficient accounting for porous material effects
$\nabla\phi$	gradFi	Gradient of electric potential (projected on cell faces)
$\nabla\phi$	vgradFi	Gradient of electric potential (cell center values)
-	kap	Conductivity of the background electrolyte

SlayFoam solver is structured in a way similar to the scalarTransportFoam solver, containing the slayFoam.C and createFields.H files, as well as the Make directory. This solver is based on the Nernst-Planck and Laplace equations. Thus, two coupled equations must be solved: one for the concentration distribution (equation (3.16) inserted into equation (3.15)) and one for the electric potential distribution (equation (3.14)). The equations are again written in matrix form, as illustrated by the code listing 5.3. The notation of variables used in the code and their correspondence to the notation used when describing the equations in the present report are explained in Table 5.2.

Several changes are introduced in this solver when compared to the scalarTransportFoam. The differences in the mass balance equation codes correspond to the differences between equations (3.1 and 5.1) and (3.15 and 3.16). The advective term is neglected and a migration term is added to the specie balance equation on line 13 of the code listing 5.3. The temperature, T, is replaced with concentration and a second species balance equation is added on lines 23-35 to allow the user to model the transport of up to two different ionic compounds. Moreover, the Darcy-scale effects of a porous medium are accounted for by scaling the relevant equations and constants with constants p and Gamma. In addition, an equation for calculating the electric potential field is added in line 41 representing the matrix form of the equation (3.14). Finally, all the necessary new parameters and variables are added as well, such as gradFi (gradient of electric potential on the cell faces) on line 7 and kappa on line 37, calculated using equation (3.19). A field named vgradFi was added on line 45 which was not necessary for the computations but was helpful during post-processing providing insight into nodal values of the electric potential gradient.

```
 1      while (simple.loop())
 2      {
 3          Info << "Time=" << runTime.timeName() << nl << endl;
 4
 5          while (simple.correctNonOrthogonal())
 6          {
 7              surfaceScalarField gradFi=fvc::snGrad(Fi)*mesh.magSf();
 8
 9              fvScalarMatrix C1Eqn
10              (
11                  p*fvm::ddt(C1)
12                  - fvm::laplacian(Gamma*DC1, C1)
13                  - z1*F*fvm::div(Gamma*mu1*gradFi,C1)    //migration
14                  ==
15                  fvOptions(C1)
16              );
17
18              C1Eqn.relax();
19              fvOptions.constrain(C1Eqn);
20              C1Eqn.solve();
21              fvOptions.correct(C1);
22
23              fvScalarMatrix C2Eqn
24              (
25                  p*fvm::ddt(C2)
26                  - fvm::laplacian(Gamma*DC2, C2)
27                  - z2*F*fvm::div(Gamma*mu1*gradFi,C2)
28                  ==
29                  fvOptions(C2)
30              );
31
32              C2Eqn.relax();
33              fvOptions.constrain(C2Eqn);
34              C2Eqn.solve();
35              fvOptions.correct(C2);
36
37              kappa=kap+F*F*(z1*z1*Gamma*mu1*C1+Gamma*z2*z2*mu2*C2);
38
39              fvScalarMatrix FiEqn
40              (
41                  fvm::laplacian(fvc::interpolate(kappa),Fi)
42              );
43              FiEqn.solve();
44
45              vgradFi=fvc::grad(Fi);      //cell center values of gradient of
        electric potential added to help monitor the values
46          }
47          runTime.write();
48      }
```

Listing 5.3: Equations of the slayFoam solver written in matrix form located in the slayFoam.C file.

Corresponding changes were made to the createFields.H file, summarized in Table 5.3. The new constants were stored in the "properties" IOdictionary for convenience as well. The variables and constants were assigned types that were appropriate for them and can be found in the

Appendix 11.2.2.

Table 5.3: Summary of changes made to variables and constants in scalarTransportFoam when creating slayFoam.

Variable/constant in scalarTransportFoam	Replaced with variable/ constant in slayFoam	New variables	New constants
T	C1, C2	Fi, gradFi,	z1, z2 mu1, mu2,
DT	DC1, DC2	kappa	p, Gamma F, T, R, kap

5.4 SaltyFoam for electrokinetic transport in molten salt

To be able to describe a molten salt electrolyte-based system, adjustments to the solver had to be made. Parts of these adjustments correspond with the differences between modelling molten salt solutions and aqueous ones described in the theory section, while parts are made to better suit the Na-Zn LMB electrolyte modelling.

The matrix forms of the equations used were written in a way shown in the code listing 5.4.

```
    while (simple.loop())
    {
        Info << "Time=" << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
            Gamma=p/(tau*tau);
            surfaceScalarField gradFi=fvc::snGrad(Fi)*mesh.magSf();
            fvScalarMatrix C1Eqn
            (
                p*fvm::ddt(C1)
                - fvm::laplacian(Gamma*DC1, C1)
                - z1*F*fvm::div(fvc::interpolate(Gamma)*mu1*gradFi,C1)
                ==
                fvOptions(C1)
            );

            C1Eqn.relax();
            fvOptions.constrain(C1Eqn);
            C1.solve();
            fvOptions.correct(C1);

            fvScalarMatrix C2Eqn
            (
                p*fvm::ddt(C2)
                - fvm::laplacian(Gamma*DC2, C2)
                - z2*F*fvm::div(fvc::interpolate(Gamma)*mu2*gradFi,C2)
                ==
                fvOptions(C2)
            );

            C2Eqn.relax();
            fvOptions.constrain(C2Eqn);
            C2Eqn.solve();
            fvOptions.correct(C2);

            C3 = (-z1*C1-z2*C2)/z3
```

```
38
39              kappa=F*F*(z1*z1*mu1*Gamma*C1+z2*z2*mu2*Gamma*C2+z3*z3*mu3*Gamma*C3);
40              //k is introduced to check the electroneutrality of the solution
41              k=z1*C1+z2*C2+z3*C3;
42
43              fvScalarMatrix FiEqn
44              (
45                  fvm::laplacian(fvc::interpolate(kappa),Fi)
46                  + F*fvc::laplacian(z1*Gamma*DC1,C1)
47                  + F*fvc::laplacian(z2*Gamma*DC2,C2)
48                  + F*fvc::laplacian(z3*Gamma*DC3,C3)
49              );
50              FiEqn.solve();
51
52              vgradFi=fvc::grad(Fi);
53
54              idif = -F*(z1*DC1*Gamma*fvc::grad(C1)+z2*DC2*Gamma*fvc::grad(C2)+
55              z3*DC3*Gamma*fvc::grad(C3));
56
57              itot = idif -F*F*fvc::grad(Fi)*(z1*z1*mu1*Gamma*C1+z2*z2*mu2*Gamma*C2
58              +z3*z3*mu3*Gamma*C3);
59          }
60
61          runTime.write();
62      }
```

Listing 5.4: Equations of the saltyFoam solver written in matrix form located in the saltyFoam.C file.

Firstly, this solver is made with the capability of modelling the 3 ions. This was done in order to capture the transport of the ions essential for the functioning of a Na-Zn LMB: Na^+, Zn^{2+}, and Cl^-. So compared to the slayFoam solver, it includes an additional equation for calculating the concentration of a third ion (Cl^-) based on the electroneutrality condition (equation (3.10)), shown on line 37 in code listing 5.4 below. C1 was used for Na^+ concentration, C2 - for Zn^{2+}, and C3 - for Cl^-.

Secondly, a property k was added on line 41 measuring the charge present in the solution used to check if the electroneutrality condition is properly enforced.

Additionally, since this solver is made for molten salt modelling, equation (3.13) was used to calculate the electric potential field and enforce electroneutrality replacing equation (3.14) used in slayFoam on lines 45-48 of code listing 5.4.

The same equation (3.19) is used to calculate the electrolyte conductivity, kappa, however, no background electrolyte is present, and thus, its' conductivity, kap, is not present. In this equation, the conductivity of the porous material comprising the diaphragm was assumed negligible in comparison to the conductivity of the molten salt, because alumina has previously been used for this purpose [10] and it has a low conductivity [67].

Finally, new functionalities calculating the total, itot, and the diffusion, idif, currents are added on lines 54-58, based on equation (3.18) with no advection.

Porosity and tortuosity are changed from constants to scalar fields (volScalarField) when compared to the slayFoam solver to enable them to have different values in different locations in space. This allows the user to introduce a portion of space that mathematically behaves as a porous medium while keeping the rest of the space unchanged.

For a molten salt case simulated in this project, the simulation directory could initially look in the way presented in Figure 5.3. This structure in particular was used for the fixed potential charging and discharge simulations done with the saltyFoam solver. Here, C1, C2, Fi, p, and tau are variable fields used in the simulation. In the constant directory, PolyMesh stores the generated mesh files, while the properties file stores the physical constants. BlockMeshDict contains the meshing parameters. ControlDict includes time and results-generation controls. FvSchemes and fvSolution files store interpolation schemes for all equation terms and matrix inversion methods respectively. SetFieldsDict stores controls for an additional functionality allowing the user to assign fields different values in specified areas of the mesh. Finally, setExprFieldsDict also contains the controls for a functionality that enables one to initiate variable fields where the resulting values are a function of space and/or time. If a simulation employs coded boundary conditions, upon starting a simulation, a new directory called dynamicCode, storing instructions for coded boundary conditions, will be created in addition to the existing main three. Thereafter, new directories with calculated fields will be produced for different timesteps, with the frequency specified by the user.

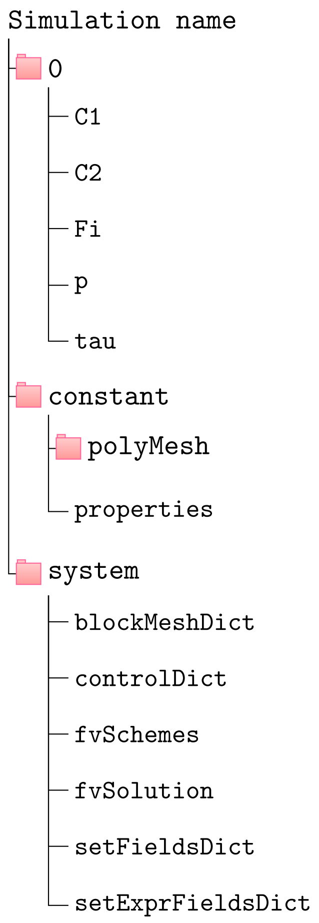


Figure 5.3: Diagram showing the structure of an OpenFOAM simulation directory typical for the simulations created in the present work.

5.4.1 Implementation of concentration boundary conditions in OpenFOAM

Concentration boundary conditions for the molten salt simulations done with the saltyFoam solver can be categorized into four types: zero gradient, fixed gradient, calculated gradient, and zero flux. The two lastly-mentioned BC types require more than simple value input, and the implementation of all these types will be described further.

Zero gradient BC is a pre-existing OpenFOAM BC type, that ensures no gradient of a property over a distance from a cell next to a boundary to the boundary itself. Thus, it simply extrapolates the value to the patch from the nearest cell. It is well-suited for boundaries that do not affect the transport.

Fixed gradient BC for concentration is based on the value of the applied fixed current density, as expressed by the following relation used for calculating its' value [47, 48]:

$$\nabla c_{i,f} \cdot \mathbf{n}_f = \frac{\mathbf{i}_f}{zFD_i}, \tag{5.2}$$

where $\mathbf{n}_f$ is a vector normal to the boundary and the subscript f, in general, indicates a value being evaluated at the boundary f. The resulting numerical values used for the simulations are summarized in Table 6.14. This condition is based on the assumption that close to the surface of an electrode a diffusion-dominated boundary layer exists, where the migrations contribution to the total flux is negligible, caused by it being close to a metallic electrode that is a lot more conductive. The current density through the boundary remains fixed, while the potential difference varies accordingly. This approach can be commonly found in the literature on ionic transport in the electrolyte layer of batteries [47, 48].

Calculated concentration **gradient** BC expands the fixed gradient approach, to fit the fixed potential condition. Thus, here the current density is allowed to vary as a function of time and space, while the potential difference across the electrolyte is kept constant. This condition can be expressed mathematically with the given equation:

$$\nabla c_{i,f} \cdot \mathbf{n}_f = \frac{\kappa_f \nabla \phi_f}{zFD_i}, \tag{5.3}$$

which is equation (5.2) combined with Ohm's law (3.7).

Since the values of the conductivity and the potential gradient at the boundary are not explicitly known at the start of the simulations, this condition can not be set by a simple value as with fixed gradient BCs. Rather, it must be coded, using the discretized version of equation (5.3), with the necessary values at the boundary being calculated and updated each time step. In OpenFOAM, this can be done using codedMixed BC type. The implementation is shown in code listing 5.5, using Na$^+$ concentration as an example.

```

    inlet
    {
        type                codedMixed;

        refValue            uniform  0.0;
        refGradient         uniform  0.0;
        valueFraction       uniform  0.0;

        name      NaFlux;

        code
        #{
            //transport properties
            const scalar z = 1;
            const scalar D1 = 8.01e-09;
            const scalar F = 96485;

            //values of variables at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField,
            scalar>("Fi");

            //values of variables at the cell centre next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField
    , scalar>("Fi").patchInternalField();
            const tmp<scalarField>& kappa_O = patch().lookupPatchField<
    volScalarField, scalar>("kappa").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = (kappa_O.ref()[faceID]*(Fi_O.ref()[faceID] -
                Fi[faceID])*this->patch().deltaCoeffs())/(z*F*D1);     //the
    concentration gradient expression
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }
```
Listing 5.5: Calculated gradient boundary condition for Na$^+$ concentration at the left boundary (next to Na electrode) located in the C1 file in the 0 directory.

Zero flux BC ensures no flux of concentration across a boundary. For cases with no advection, the migration and the diffusion fluxes must cancel each other out to result in zero total. This can be expressed mathematically in the following way:

$$\mathbf{J}_i^{tot} = [D_i \nabla c_i + c_i u_i \nabla \phi]_f \cdot \mathbf{n}_f = 0, \tag{5.4}$$

Formally, this is a Robin-type (or mixed-type) boundary condition for concentration because it is a function of both the concentration and its gradient. However, equation (5.4) can be rewritten into a Neumann-type BCs for easier implementation [46]:

$$\nabla c_{i,f} \cdot \mathbf{n}_f = -c_{i,f} \frac{u_i}{D_i} \nabla \phi_f \cdot \mathbf{n}_f. \tag{5.5}$$

Here, the subscript f represents the boundary cell face location.

The implementation of this approach for defining the boundaries was again conducted using codedMixed BC type. The code listing 5.6 provides an example of what this implementation looks like for the Na$^+$ ion at the boundary next to the Na electrode in the OpenFOAM case setup.

```
    outlet
    {
        type            codedMixed;

        refValue        uniform 0.0;
        refGradient     uniform 0.0;
        valueFraction   uniform 0.0;

        name    zeroNaFlux;

        code
        #{
            //transport properties
            const scalar Mu1 = 1.16e-12;
            const scalar D1 = 8.01e-09;

            //values of variables at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField,
    scalar>("Fi");
            const scalarField& C1 = patch().lookupPatchField<volScalarField,
    scalar>("C1");

            //values of variables at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField
    , scalar>("Fi").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = -(C1[faceID]*Mu1/D1)*(Fi[faceID] - Fi_O.ref()[
    faceID])*this->patch().deltaCoeffs();
//the concentration gradient expression
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }
```
Listing 5.6: Zero flux boundary condition for Cl^- ions located in the C3 file in the 0 directory.

6 Case setups

6.1 Aqueous simulations

Models corresponding to the Scenario 1 simulations conducted by Rolle et al. [49] were created. One simulation was conducted for each of the three experiments of Scenario 1 (permanganate, allura red, and new coccine tracer experiments) resulting in a total of 3 simulations. The simulations implement the approach used in the paper when it comes to treating tortuosity as a fitting parameter to achieve the centre of mass displacements equal to the ones calculated from the experiments.

6.1.1 Mesh

The simulations were conducted over a rectangular 2D domain. BlockMesh was chosen as the meshing method for the conducted simulations due to the simplicity of the shape of the computational domain. It is important to note, that OpenFOAM always uses 3D meshes, however, 2D simulations can be conducted by setting the boundaries normal to the third dimension as "empty" (front and back boundaries for the relevant simulations). No mesh refinement is necessary for that third direction, since no gradients will be present there and it will not be included in the computations. This approach also allows properties like current density to be meaningful. The top and bottom boundary types were set as walls, while the left (inlet) and right (outlet) boundary types were defined as patches. The boundaries and their types are illustrated by Figure 6.1.

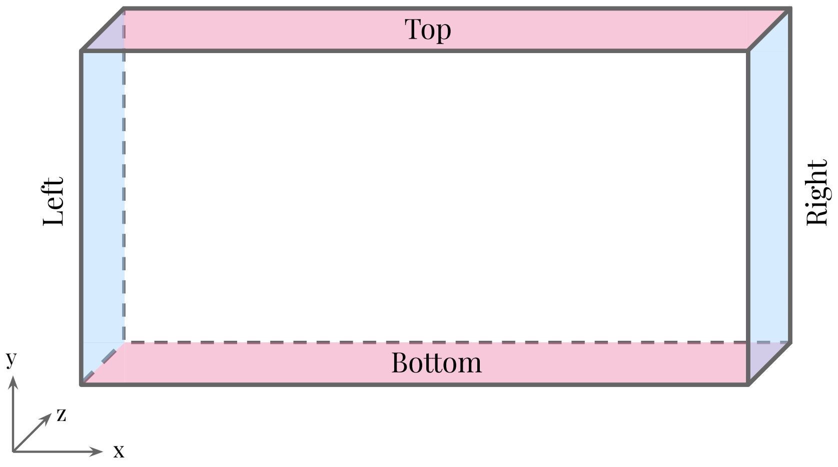


Figure 6.1: An illustration on the computational domain used in aqueous simulations: the type of the pink top and bottom boundaries is set to wall, the type of the blue left and right boundaries is set to patch, while the clear front and back boundaries are set as empty.

The same mesh was used for the migration simulations of all three tracers. The computational domain size was set to 28x15 cm (in correspondence with the experiments of Rolle et al. [49]) with 280x150 mesh divisions. This mesh is presented in Figure 6.2. These parameters resulted in a mesh finer than the one used in the paper by Rolle et al. [49] and should therefore be fine enough to produce the desired results. The mesh resolution was equal in both directions to ensure no dependence of the numerical error on the directions in space.

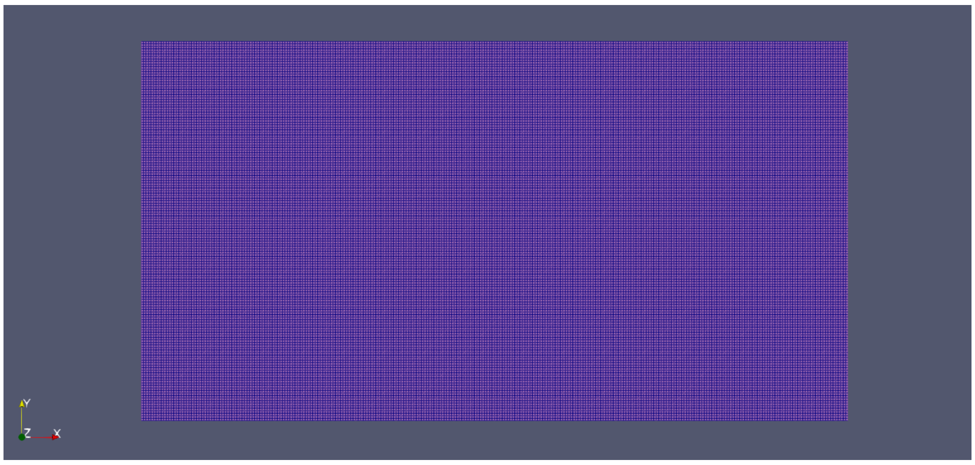


Figure 6.2: Mesh used for the aqueous simulations created using blockMesh with 280 divisions in the x-direction and 150 - in the y-direction (blue lines).

6.1.2 Physical properties

The values of the simulation-specific ionic property constants used in the three simulations done when verifying the model against the results from the Rolle et al. paper [49] are presented in Table 6.1. These values correspond with the ones used in the article. Number 1 in the names of the properties indicates the main tracer chemical, while number 2 is reserved for the properties of a supporting traces compound where relevant.

Table 6.1: Transport properties used for the aqueous simulations.

Simulation/ tracer chemical	z1 [-]	z2 [-]	DC1 [m^2/s]	DC2 [m^2/s]	mu1 [s·mol/kg]	mu2 [s·mol/kg]
Permanganate	-1	0	$1.5 \cdot 10^{-9}$	0	$6.12 \cdot 10^{-13}$	0
Allura Red	-2	-2	$5.46 \cdot 10^{-10}$	$9.83 \cdot 10^{-10}$	$2.23 \cdot 10^{-13}$	$4.01 \cdot 10^{-13}$
New Coccine	-3	-2	$4.06 \cdot 10^{-10}$	$9.83 \cdot 10^{-10}$	$1.66 \cdot 10^{-13}$	$4.01 \cdot 10^{-13}$

The constants equal for the three cases are presented in Table 6.2.

Table 6.2: A summary of constants used in all aqueous simulations.

Constant	Value	Significance
R	8.314 kg·m^2/s^2·K·mol	the universal gas constant
T	295 K	temperatue
F	96485 A·s/mol	Faraday constant
ω	0.4	porosity

The values of fitted tortuosity, τ^2, and estimated background electrolyte conductivity, kap, for all simulations, are presented in Table 6.3.

Table 6.3: Porous matrix tortuosity, τ^2, and background electrolyte conductivity, kap, values used in simulations with permanganate, Allura Red, and New Coccine tracer plumes.

Simulation	τ^2 [-]	kap [S/m]
Permanganate	1.44	$2.91 \cdot 10^{-2}$
Allura Red	1.18	$2.92 \cdot 10^{-2}$
New Coccine	1.29	$3.07 \cdot 10^{-2}$

The background electrolyte conductivity, kap, was estimated using the average values of the reported experimental current and electric potential measurements [49] together with the physical dimensions of the system and applying the extended version of equation (3.7):

$$kap = \frac{Il}{\Delta\phi A}, \tag{6.1}$$

where l is the distance across which the electric potential is measured and A is the estimated area perpendicular to the current flow taken up by the electrolyte. The following relation was used as an estimate for A:

$$A = Ht\omega, \tag{6.2}$$

where H is the hight of the system, t is its' thickness and ω is the accessible area porosity of the matrix material. Since accessible area porosity is unknown for the system, it was assumed to be equal to the known accessible volumetric porosity.

The conductivity of the porous matrix itself is neglected since the conductivity of glass is normally much lower than the contribution of the dissolved salts [68]. In addition, this way of estimating the conductivity of the system accounts for the contribution of the surface conductivity, which is an important term, that would not be accounted for if the conductivity was estimated from the ionic concentrations using equation (3.19) [69].

6.1.3 Boundary and initial conditions

Concentrations

Table 6.4 presents the initial concentrations of the tracer plumes. The concentration of these ions in the rest of the solution outside of the plume was set to 0. C1 here is the concentration of the main tracer compounds like permanganate, Allura Red or New Coccine, while C2 is the concentration of SO_4^{2-} that was present in the plumes during the 2 out of 3 experiments. ZeroGradient concentration boundary condition was applied in the top, bottom, left, and right boundaries.

Table 6.4: Initial concentrations of tracer plumes.

Simulation	C1 [mM]	C2 [mM]
Permanganate	3	0
Allura Red	2.4	0.6
New Coccine	2.25	1.125

setExprFields functionality was used in order to achieve the desired tracer concentrations in the plumes while keeping them 0 elsewhere.

Electric potential

Dirichlet-type boundary conditions (BCs) for electric potential were used on the left and right boundaries. The values on the left side were set to 0 as a reference point for all simulations, while on the right side, they were set to a value corresponding to the average measured potential reported for each of the experiments by Rolle et al. [49]. These values are presented in Table 6.5. The top and bottom boundaries were set to zeroGradient.

Table 6.5: Fixed value left and right boundary conditions for the electric potential for all aqueous cases.

Simulation	Left Fi BC [V]	Right Fi BC [V]
Permanganate	0	143.2
Allura Red	0	158.9
New Coccine	0	146.1

6.1.4 Solution and time controls

The GAMG solver was used for solving the matrix form of the equations for C1, C2, and Fi. The parameters used with GAMG are listed in Table 6.6 in a way they are listed in the fvSolution file.

Table 6.6: Parameters used for matrix solution algorithms for aqueous simulations.

C1 and C2	
solver	GAMG
smoother	DILU
tolerance	10^{-6}
reltol	0
Fi	
solver	GAMG
smoother	GaussSeidel
tolerance	10^{-8}
reltol	0

The numerical schemes used for approximating all needed values are summarized in Table 6.7. The schemes recommended for scalarTransportFoam (given in a tutorial case for that solver) were used for all terms of the equations except the new migration term added in the slayFoam solver. The divergence schemes necessary for solving the migration term were set to the vanLeer scheme since it was found to produce stable results.

Table 6.7: Numerical schemes used in aqueous solution simulations done using the slayFoam solver.

Time schemes	
default	Euler
Gradient schemes	
default	Gauss linear
Divergence schemes	
default	Gauss vanLeer
Laplacian schemes	
default	Gauss linear corrected
Interpolation schemes	
default	linear
Surface gradient schemes	
default	corrected

All simulations were run for a total transport time of 7200 s each (corresponding with the work presented by Rolle et al. [49]) with a time advancement step size of 10 s.

6.2 Darcy-scale molten salt simulations

A total of 42 simulations of ion transport in molten salt electrolyte were conducted. Firstly, 11 simulations to check grid convergence were created, distinguished by a letter G in their names (for example G1, G2 and so on). Additionally, 14 simulations were run to check for time step convergence (8 with a current density of 100 mA/cm^2 and 6 with a current density of 2000 mA/cm^2), indicated by the letter T in their names and a number reflecting the current density used (for example T100.1, T100.2 and so on and T2000.1, T2000.2 and so forth). This notation for the convergence testing simulations is summarized in Table 6.8.

Table 6.8: Summary of the notation used for convergence testing simulations run under fixed current conditions for a charging battery, where GCT stands for Grid Convergence Testing and TSCT stands for Time-Step Convergence Testing.

Simulation name	Number of grid divisions	Time step size [s]	i [mA/cm^2]	Purpose
G1	48x24	5	100	GCT
G2	50x25	5	100	GCT
G3	96x48	5	100	GCT
G4	100x50	5	100	GCT
G5	200x100	5	100	GCT
G6	240x120	5	100	GCT
G7	400x200	5	100	GCT
G8	600x300	5	100	GCT
G9	800x400	5	100	GCT
G10	1000x500	5	100	GCT
G11	1200x600	5	100	GCT
T100.1	400x200	1000	100	TSCT
T100.2	400x200	500	100	TSCT
T100.3	400x200	100	100	TSCT
T100.4	400x200	50	100	TSCT
T100.5	400x200	10	100	TSCT
T100.6	400x200	5	100	TSCT
T100.7	400x200	1	100	TSCT
T100.8	400x200	0.5	100	TSCT
T2000.1	400x200	5	2000	TSCT
T2000.2	400x200	1	2000	TSCT
T2000.3	400x200	0.5	2000	TSCT
T2000.4	400x200	0.1	2000	TSCT
T2000.5	400x200	0.05	2000	TSCT
T2000.6	400x200	0.01	2000	TSCT

Three different fixed current densities were investigated under both charging and discharging battery conditions. These simulations are named FCC100, FCC1000, FCC2000, FCD100, FCD1000, and FCD2000, where the letters FC in the names stand for "fixed current", the third letters C and D indicate the charging and discharging states respectively and the numbers correspond with the current density used (i.e. 100 mA/cm^2, 1000 mA/cm^2, and 2000 mA/cm^2).

Four charge and four discharge simulations under fixed potential differences were conducted, named FPC0.025, FPC0.333, FPC0.679, FPC0.99 and FPD0.018, FPD0.236, FPD0.48, and FPD0.99, named using the same principle as for fixed current simulations, where the letters FP now stand for "fixed potential", while the numbers reflect the magnitude of the applied fixed electric potential difference between the left and right boundaries. The notation used for simulations run under fixed current density and fixed potential difference conditions is summarized in Table 6.9. The three lowest potential differences used in charging and discharging fixed potential difference cases correspond with the average potential differences observed during the fixed current density simulations. More specifically, for simulation FPC0.025 the used voltage of 0.025 V corresponds with the average potential difference observed for simulation FCC100 over time. Equivalently, FPC0.333 corresponds with FCC1000, FPC0.679 - with FCC2000, FPD0.018 - with FCD100, FPD0.236 - with FCD1000, and finally FPD0.48- with FCD2000. The remaining two fixed potential difference simulations, FPC0.99 and FPD0.99, apply a potential difference that is based on the voltages recorded in lab-scale physical experiments done on the Na-Zn LMB cell by Xu et al. [10].

Table 6.9: Summary of the notation used for fixed current density and fixed potential difference charging and discharging simulations.

Simulation name	Fixed parameter	Process	Fixed i [mA/cm^2]	Fixed $\Delta\phi$ [V]
FCC100	Current	Charging	100	-
FCC1000	Current	Charging	1000	-
FCC2000	Current	Charging	2000	-
FCD100	Current	Discharging	100	-
FCD1000	Current	Discharging	1000	-
FCD2000	Current	Discharging	2000	-
FPC0.025	Potential	Charging	-	0.025
FPC0.333	Potential	Charging	-	0.333
FPC0.679	Potential	Charging	-	0.679
FPC0.99	Potential	Charging	-	0.99
FPD0.018	Potential	Discharging	-	0.018
FPD0.236	Potential	Discharging	-	0.236
FPD0.48	Potential	Discharging	-	0.48
FPD0.99	Potential	Discharging	-	0.99

Finally, different diaphragm parameters were investigated by conducting simulations PT0.6, PT0.85, and PT0.95, where the number in the name indicates the diaphragm porosity used. Aside from varying porosity and tortuosity values, these cases had the same setup as simulation FCC100, with case PT0.85 being the same as FCC100 except for the chosen end time. The parameters used for these simulations are summarized in Table 6.10. The tortuosity values used were based on a tortuosity-porosity relation (valid for $\omega \in[0.5,1]$) described by Matyka et al. [70]:

$$\tau = a(1-\omega)+1, \tag{6.3}$$

where a is a fitting parameter, equal to 0.38 for the conducted simulations, calculated using the original values of $\omega = 0.85$ and $\tau^2 = 1.12$, chosen based on the results from previous work conducted on the solute transport through a porous diaphragm in Na-Zn LMB [18].

Table 6.10: Summary of the notation and parameters used for diaphragm properties' testing fixed current density charging simulations.

Simulation name	Porosity, ω	Tortuosity, τ^2	Fixed i [mA/cm^2]
PT0.6	0.6	1.32	100
PT0.85	0.85	1.12	100
PT0.95	0.95	1.04	100

6.2.1 Mesh

The computational domain size was set to 8 cm in the x-direction and 4 cm in the y-direction. The domain was divided into 80000 cells, 400 in the x-direction and 200 in the y-direction for all simulations except the simulations done for grid convergence investigation. As illustrated in Figure 6.3 the left and right patches were set to represent the electrolyte next to the Na and Zn electrodes respectively. Top and bottom boundaries were defined as walls, while front and back boundaries were kept empty to keep the simulations 2D.

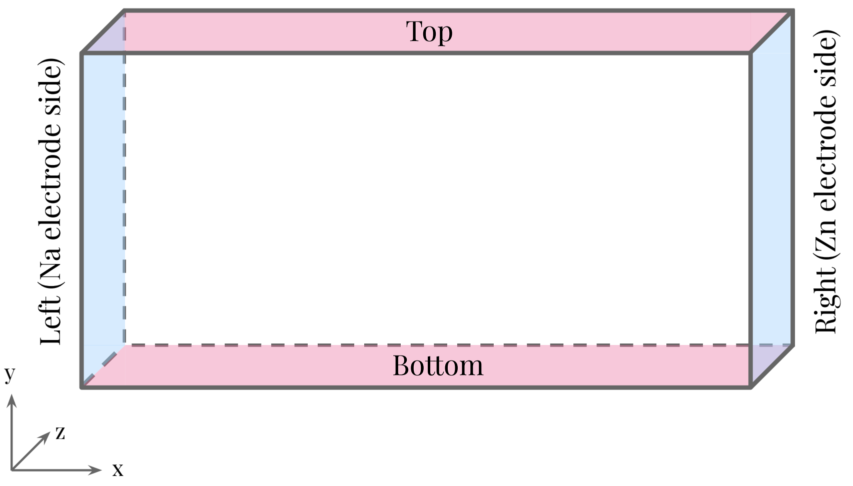


Figure 6.3: An illustration on the computational domain used in molten salt simulations: the pink top and bottom boundaries are defined as walls, the blue left and right boundary types are set to patch, while the clear front and back boundaries are set as empty.

6.2.2 Physical properties

The temperature was set to 833 K for all simulations corresponding to the temperature used during physical experiments on the Na-Zn LMBs [10].

All the compound-specific variables and constants were distinguished by numbers in the names. Number 1 corresponds with Na^+ ion (meaning properties like C1, z1, mu1 and so on belong to the Na^+ ion), 2 - with Zn^{2+}, and 3 - with Cl^-.

Transport properties of the modelled species are comprised in Table 6.11. These values are based on the available data at 1100K in NaCl [71]. The value used for Zn is the diffusion coefficient for cadmium due to more appropriate values being unavailable.

Table 6.11: Diffusion coefficients used in all molten salt electrolyte simulations for the three modelled ionic species.

Ion	Used diffusion coefficient [m^2/s]
Na^+	$8.01 \cdot 10^{-9}$
Zn^{2+}	$6.35 \cdot 10^{-9}$
Cl^-	$7.45 \cdot 10^{-9}$

The diaphragm properties were kept constant, with $\omega = 0.85$ and $\tau^2 = 1.12$ for all simulations, except simulations PT0.6, PT0.85, and PT0.95, where those were varied in ways described earlier. The chosen porosity of 0.85 is based on the porosity of the diaphragm used in physical experiments done on the Na-Zn LMB cell, while the tortuosity of 1.12 corresponds with the value estimated for such diaphragm in earlier work [18].

The universal constants used in all molten salt simulations are given in Table 6.12.

Table 6.12: A summary of universal constants used in all molten salt simulations.

Constant	Value	Significance
R	8.314 kg·m^2/s^2·K·mol	the universal gas constant
F	96485 A·s/mol	Faraday constant

6.2.3 Boundary and initial conditions

Concentration

The initial concentration distributions differ for charge and discharge simulations. For the charging simulations, the initial concentration of Zn^{2+} ions in the electrolyte was set to 0. NaCl was assumed to be the only salt present in the electrolyte to simplify the system since $CaCl_2$ is not involved in the cell reaction. The initial concentration Na^+ ions was calculated from the available data on the density of molten NaCl and its' molar mass according to equation (6.4):

$$c_{ion} = x_{ion}\frac{\rho_{salt}}{M_{salt}}, \tag{6.4}$$

where x_{ion} is the stochiometric coefficient of the considered ion in the relevant salt (equal to 1 for both Na^+ and Cl^- in NaCl), ρ_{salt} is the density of the salt, and M_{salt} is its' molar mass. The value of molten NaCl density at 1100 K was used, $\rho_{NaCl} = 1542$ kg/m^3 instead of the working temperature of 833 K. This was the lowest temperature at which the density data was available, mostly due to the pure NaCl being solid at temperatures below 1074 K under atmospheric pressure [72].

For discharge simulations, the initial concentration on the left side of the diaphragm was set to 100 mol% NaCl, while inside of the diaphragm and on its right size, it was set to around 43 wt% NaCl and 57 wt% $ZnCl_2$. Using NaCl-$ZnCl_2$ salt mixture density data at 843 K and 53.3 mol% $ZnCl_2$ reported by Janz et al. [73] $\rho_{mixture} = 2331$ kg/m^3, this corresponds with 17111.6 mol NaCl/m^3 salt and 9766.2 mol $ZnCl_2$/m^3 in terms of molar concentration, calculated using the following relation:

$$c_{salt} = w_{salt}\frac{\rho_{mixture}}{M_{salt}}, \tag{6.5}$$

where w_{salt} is the weight fraction of a specific salt in the salt mixture. These initial concentration distributions are illustrated in Figure 6.4. No initial conditions for Cl^- ion concentration were needed, because it was calculated from the electroneutrality condition.

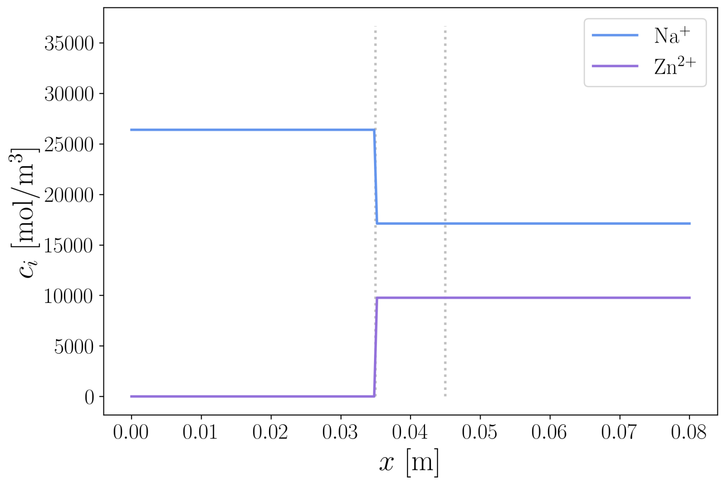


Figure 6.4: Initial concentration conditions for Na$^+$ and Zn^{2+} ions for discharge process simulations (all FCD and FPD simulations).

The boundary condition types for Na$^+$ and Zn^{2+} concentrations used for all simulations are summarized in Table 6.13. No boundary conditions were necessary for the calculation of the Cl$^-$ concentration field, since it was based on the electroneutrality condition and not a mass-balance equation.

Table 6.13: The overview of the concentration boundary condition types used for Na$^+$ and Zn^{2+} ions.

Simulations	c_{Na^+} (C1) BC type		$c_{Zn^{2+}}$ (C2) BC type	
	Left boundary	Right boundary	Left boundary	Right boundary
G1-G11 T100.1-T100.8 T2000.1-T2000.6 All FCC All FCD All PT	Fixed gradient, calculated using equation (5.2) (based on fixed current)	Zero flux, based on equation (5.5), implementation shown in code listing 5.6	Zero flux	Fixed gradient
All FPC All FPD	Calculated gradient, based on equation (5.3), implementation shown in code listing 5.5 (based on fixed potential)	Zero flux	Zero flux	Calculated gradient

For simulations with fixed applied current, the fixed concentration gradient BCs based on the applied current were used for both ionic specie at the boundaries across which they are transported. Their values, calculated using equation (5.2), are presented in Table 6.14.

Table 6.14: Values used for the fixed gradient concentration BCs for the fixed current density simulations.

Simulation	∇c_{Na^+}, [mol/m^4] at the left boundary	$\nabla c_{Zn^{2+}}$, [mol/m^4] at the right boundary
FCC100	$-1.29 \cdot 10^6$	$6.96 \cdot 10^5$
FCC1000	$-1.29 \cdot 10^7$	$6.96 \cdot 10^6$
FCC2000	$-2.59 \cdot 10^7$	$1.39 \cdot 10^7$
FCD100	$1.29 \cdot 10^6$	$-6.96 \cdot 10^5$
FCD1000	$1.29 \cdot 10^7$	$-6.96 \cdot 10^6$
FCD2000	$2.59 \cdot 10^7$	$-1.39 \cdot 10^7$

As presented in Table 6.13, both Na^+ and Zn^{2+} were set to experience zero flux at the electrode opposite of their own. ZeroGradient BC was applied at the top and bottom boundaries.

Electric potential

For convergence testing and the **fixed current simulations**, Neumann-type BCs were used for the electric potential on the left patch, meaning the electric potential gradient was given at that boundary. This was implemented using fixedGradient boundary condition type in OpenFOAM. The value of the gradient at the boundary on the electrolyte side was assumed to be equal to the gradient at that boundary on the electrode side. This assumption is equivalent to assuming no charge-transfer overpotential, which is believed to be very low in LMB due to liquid-liquid interfaces. This approach avoids the use of the variable conductivity of salt. It was therefore calculated for a given current density with the conductivity of Na metal using Ohm's law:

$$\nabla \phi = -\frac{i}{\kappa}. \tag{6.6}$$

The conductivity of Na at 873 K was used, $\kappa_{Na}^{873K} = 3.1 \cdot 10^6$S/m. The resulting electric potential gradient values for the different current densities used are presented in Table 6.15. The potential BC at the right boundary was given as a fixed value of 0 V, while the top and bottom boundaries were set to zeroGradient.

The initial electric potential for the fixed current simulations was set to 0 everywhere besides the boundaries.

Table 6.15: Electric potential fixed gradient boundary condition values for the left boundary for all fixed current density molten salt electrolyte simulations.

Simulations	Current density, i [mA/cm^2]	$\nabla \phi$, [V/m] at the left boundary
FCC100 G1-G11 T100.1-T100.8 PT0.6, PT0.85, PT0.95	100	$3.23 \cdot 10^{-4}$
D100	100	$-3.23 \cdot 10^{-4}$
C1000	1000	$3.23 \cdot 10^{-3}$
D1000	1000	$-3.23 \cdot 10^{-3}$
C2000 T2000.1-T2000.6	2000	$6.45 \cdot 10^{-3}$
D2000	2000	$-6.45 \cdot 10^{-3}$

For the **fixed electric potential difference** simulations, the potential values at the left and right boundaries were set using fixedValue BC type in OpenFOAM. The right boundary was kept at 0 V for all cases, while the value at the left boundary was adjusted to fit the desired potential difference. An initial linear potential distribution was introduced between the boundaries to avoid a large potential jump at the beginning of the simulation, destabilizing the solution to the model. This was done using the setExprFields functionality in OpenFOAM. After the first time step, the potential distribution is calculated and updated to match the actual solution to the model. The potential boundary and initial conditions for these simulations are summarized in Table 6.16.

Table 6.16: Values used for fixed value potential BCs and in initial distribution conditions for the fixed electric potential difference simulations.

Simulation name	ϕ at the left boundary [V]	ϕ at the right boundary [V]	Initial ϕ distribution [V]
FPC0.025	0.025	0	0.025-0.3125x
FPC0.333	0.333	0	0.333-4.1625x
FPC0.679	0.679	0	0.679-8.4875x
FPC0.99	0.99	0	0.99-12.375x
FPD0.018	0.018	0	0.018-0.225x
FPD0.236	0.236	0	0.236-2.95x
FPD0.48	0.48	0	0.48-6x
FPD0.99	0.99	0	0.99-12.375x

6.2.4 Solution and time controls

The GAMG solver was used for solving the matrix form of the equations for C1, C2, C3, and Fi. The parameters used with the GAMG solver were unchanged from the ones used for the aqueous solution simulations listed in Table 6.6, except for the fact that C3 field was added following the same setup as C1 and C2.

The numerical schemes used for these simulations were kept the same as for the aqueous simulations as well and can be found in Table 6.7.

The chosen time controls for convergence testing simulations are presented in Table 6.17. The output frequency was adjusted to fit the end time well, resolving the variations in time, without producing unnecessary high amounts of data. The values are not mentioned here, because they do not affect the computational process itself. The rest of the time controls were left unchanged from the recommended values from the scalarTransportFoam tutorial case.

Table 6.17: Time controls used for the convergence testing simulations.

Simulation name	Time step size [s]	End time [s]
G1	5	15000
G2	5	15000
G3	5	15000
G4	5	15000
G5	5	15000
G6	5	15000
G7	5	15000
G8	5	15000
G9	5	15000
G10	5	15000
G11	5	15000
T100.1	1000	30000
T100.2	500	30000
T100.3	100	30000
T100.4	50	30000
T100.5	10	30000
T100.6	5	30000
T100.7	1	30000
T100.8	0.5	30000
T2000.1	5	140
T2000.2	1	140
T2000.3	0.5	140
T2000.4	0.1	140
T2000.5	0.05	140
T2000.6	0.01	140

At a certain point in time during the charging of the battery, the concentration of Na^+ ions at the left boundary reaches zero, as a consequence of Na^+ being consumed at that boundary. The last reported time step before that point for each simulation was used as the ending point or end time. Equivalently, for the discharging battery simulations the last reported time step before the time at which the Zn^{2+} concentration has reached zero was used as the end time. An exception was made for simulations FPC0.025 and FPD0.018 where an end time of 46000 s was chosen since the driving force was low and the process was slow. The chosen end times and time step sizes for the rest of the molten salt simulations are listed in Table 6.18.

Table 6.18: Time controls used for the fixed current density, fixed potential difference, and diaphragm testing simulations.

Simulation name	Time step size [s]	End time [s]
FCC100	5	46000
FCC1000	0.2	460
FCC2000	0.1	110
FCD100	5	22000
FCD1000	0.2	255
FCD2000	0.1	72
FPC0.025	10	46000
FPC0.333	1	6800
FPC0.679	0.5	1100
FPC0.99	0.1	420
FPD0.018	10	46000
FPD0.236	2	12200
FPD0.48	1	2800
FPD0.99	0.1	610
PT0.6	5	11200
PT0.85	5	10200
PT0.9	5	10000

7 Results

7.1 Verification of the results

The expected displacements of the plum's centres of mass, S, (as reported by Rolle et al. [49]) and the displacements achieved in the conducted simulations, S', are presented in Table 7.1. With the chosen tortuosity values, τ^2, the achieved centre of mass displacement values are equal to the reported ones (to the extent of the accuracy of the reported values).

Table 7.1: Values of the centre of mass displacements reported by Rolle et al. [49] (S) and achieved in the conducted simulations (S').

Tracer chemical	Reported S [cm]	Simulated S' [cm]
Permanganate	14	14
Allura Red	12.5	12.5
New Coccine	12	12

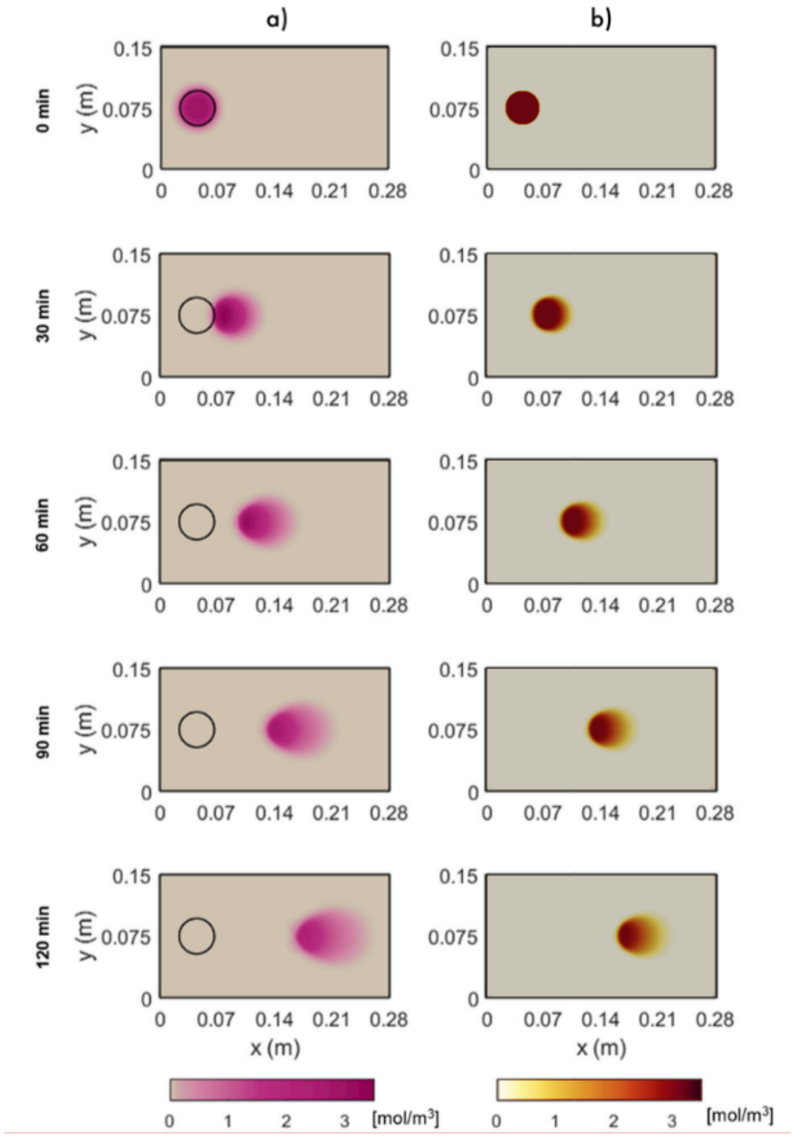


Figure 7.1: Results of the permanganate tracer simulation from [49] (a) compared to the results of the permanganate simulation produced using slayFoam solver (b).

From the results presented in Figures 7.1, 7.2, and 7.3 it is apparent that the model created in the present work exhibits similar transport behaviour as the model created by Rolle et al. [49]. The plume's centres of mass are transported a distance equal to the reported values and in the same direction as them. The tortuosity values used in the conducted simulations ($\tau^2 \in [1.29, 1.44]$) are close to the ones used in the article [49] ($\tau^2 \in [1.30, 1.43]$) as well. However, the resulting shapes and sizes of the plumes differ noticeably from the ones achieved by Rolle et al. [49]. For permanganate tracer simulation the observed plume shapes themselves and the concentration distribution within them are similar to the reported ones, in contrast with their size (of the degree of tracer spreading out), which is smaller in the present work. The shapes observed in allura red and new coccine simulation's results are less round or symmetrical and have a more uneven concentration distribution than that of the results reported by Rolle et al. [49]. Thus, the model implemented in the slayFoam solver underpredicts the degree of tracer spreading for all tested tracer compounds. This could be the result of the simplifications and assumptions introduced to the model. The concentrations of all compounds present in the background electrolyte were assumed constant and homogeneous and the interspecies interactions with the background electrolyte species were neglected. However, the electrostatic interactions between charged components are known to affect the transport of multicomponent electrolytes in saturated porous media [41].

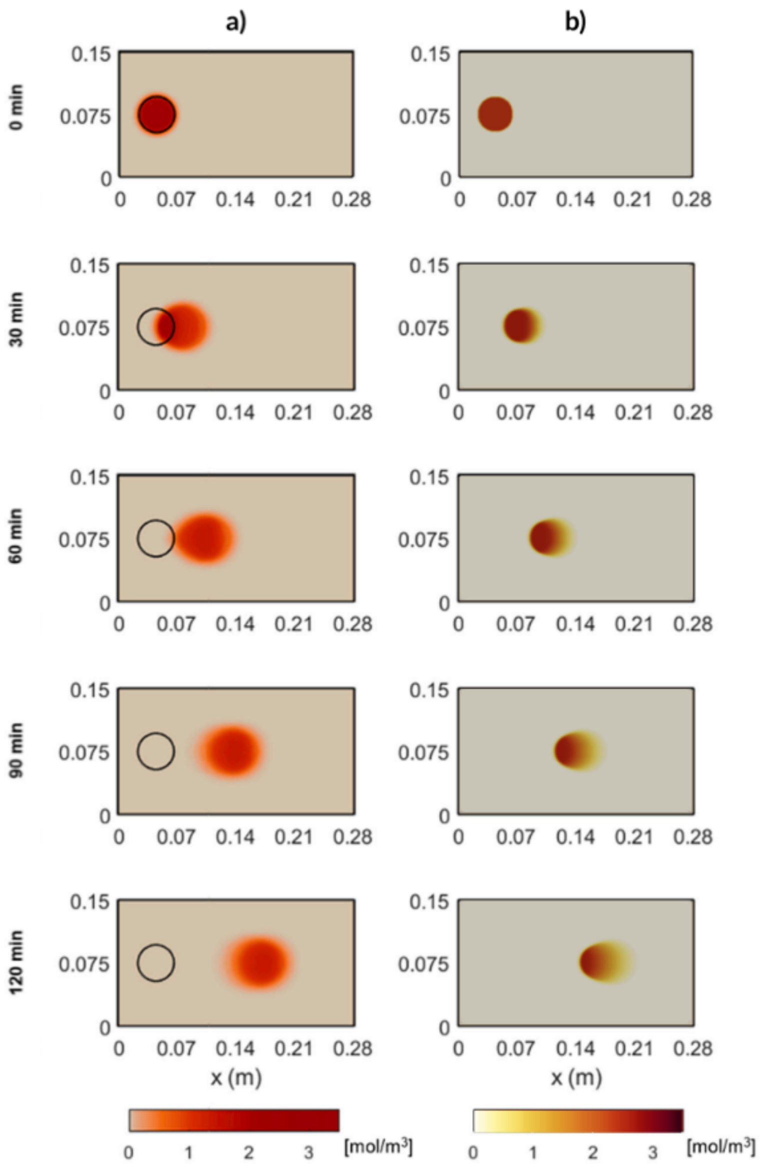


Figure 7.2: Results of the Allura Red tracer simulation from [49] (a) compared to the results of the Allura Red simulation produced using slayFoam solver (b).

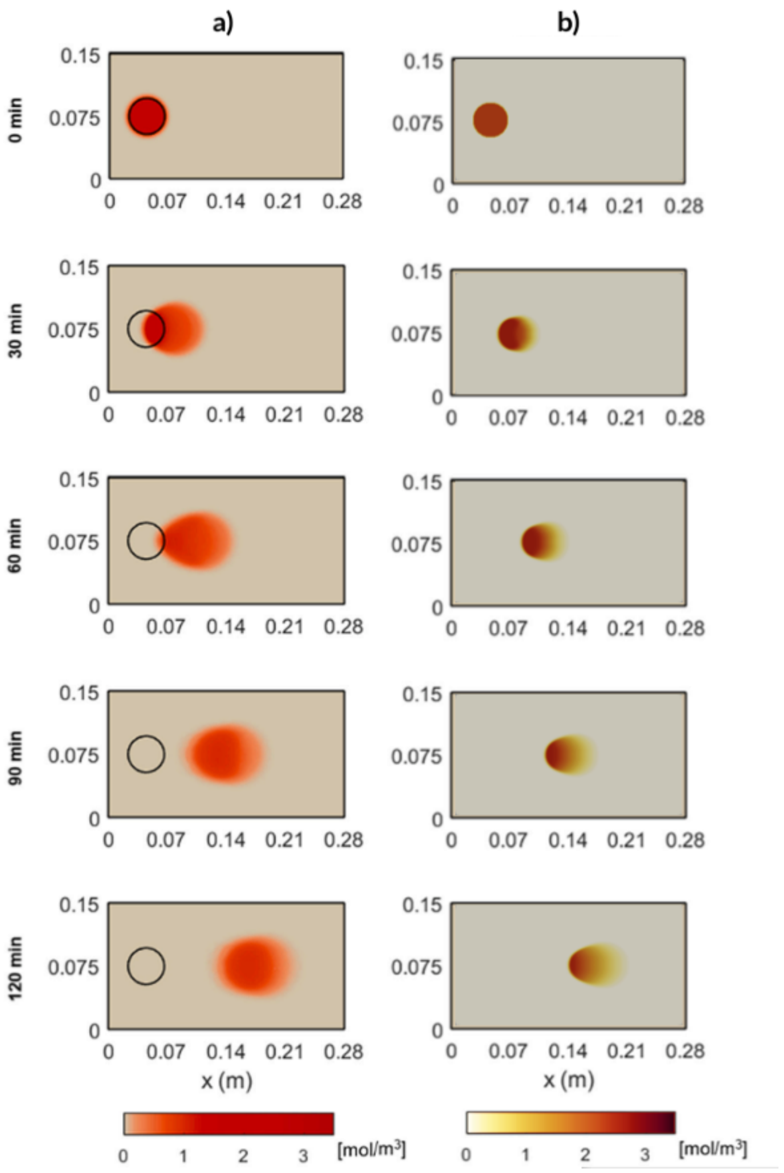


Figure 7.3: Results of the New Coccine tracer simulation from [49] (a) compared to the results of the New Coccine simulation produced using slayFoam solver (b).

7.2 Molten salt electrolyte simulation results

7.2.1 Convergence testing

For convergence testing, the value of the total resistance in the system was calculated for simulations done using different mesh resolutions and time step sizes. The resistance was chosen as the test parameter because it is affected by all concentration distributions and is therefore expected to converge to a single value as the concentration results converge.

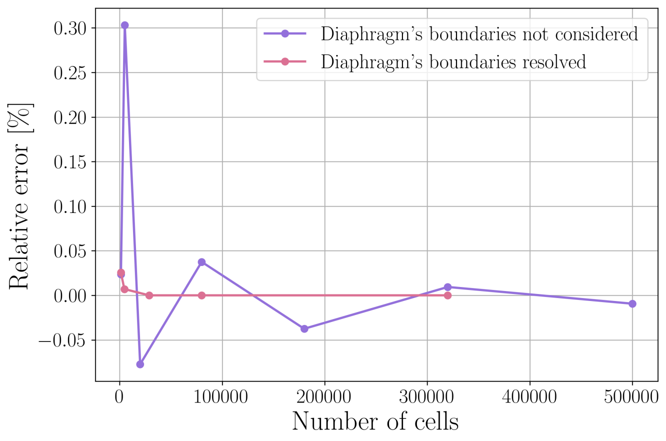


Figure 7.4: Relative error in the resistance measurement compared to the next tested grid refinement after 15000 s. The pink line corresponds with simulations (G1, G3, G6, G7, G9, G11) where all tested meshes had cells aligning with the diaphragms' boundaries, while for the tests represented with the purple line (G2, G4, G5, G7, G8, G9, G10, G11), not all did.

The results for the grid convergence testing are presented in Figure 7.4. They reveal that the alignment of the grid with the diaphragms' boundaries is essential when seeking convergence. The convergence of the results is much more noticeable and quick, and the relative change in the resistance measurement is in general lower when only using meshes that resolve the diaphragms' sides. Based on these results, a grid with 400 divisions in the x-direction and 200 divisions in the y-direction (G7) was chosen for all other simulations since it resolves the boundaries of the diaphragm and produces a converged solution, while not being unnecessarily refined.

The results for the time step convergence testing with current densities of 100 mA/cm^2 and 2000 mA/cm^2 under charging conditions are presented in Figures 7.5 and 7.6 respectively. The results appear to have converged when using a 5 s time step with i = 100 mA/cm^2 and 0.1 s with i = 2000 mA/cm^2. The time step sizes for all other simulations were chosen based on these results. More specifically, 5 s for simulations with i = 100 mA/cm^2, 0.1 s for the ones done with i = 2000 mA/cm^2, and the time step sizes for other simulations were chosen relative to these values based on how much faster or slower the process progressed in comparison.

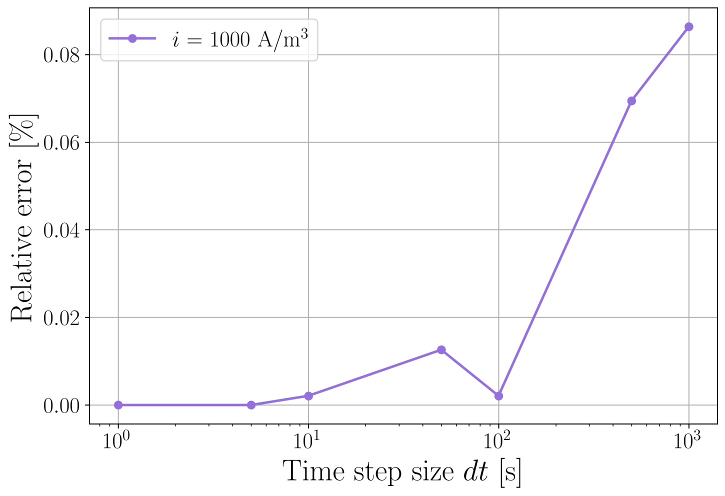


Figure 7.5: Relative error in the resistance measurement compared to the next tested time step size after 30000 s using a charging current density of 100 mA/cm^2 and a mesh with 80000 cells.

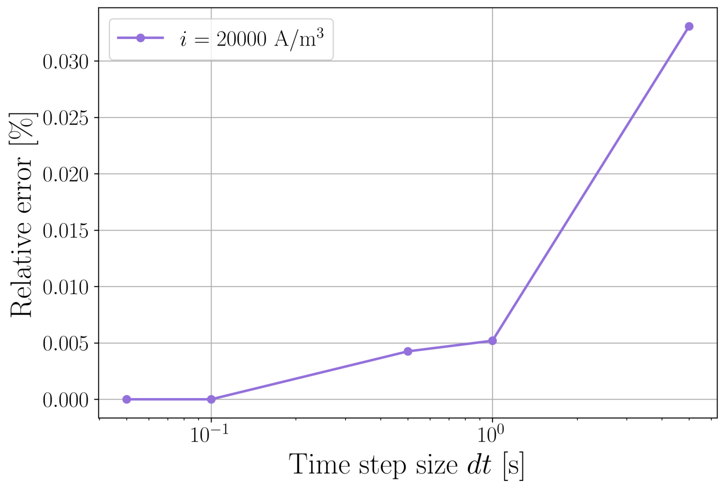


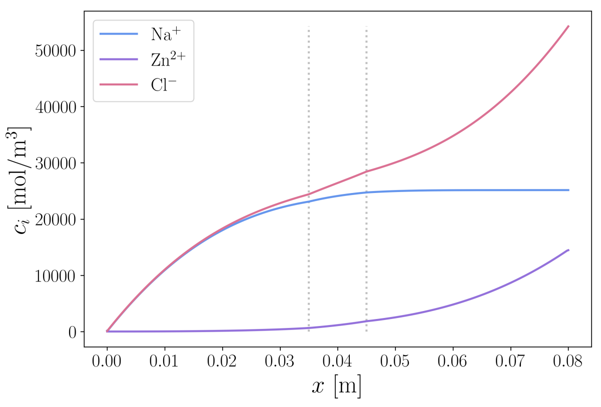
Figure 7.6: Relative error in the resistance measurement compared to the next tested time step size after 110 s using a charging current density of 2000 mA/cm^2 and a mesh with 80000 cells.

7.2.2 Fixed current simulations

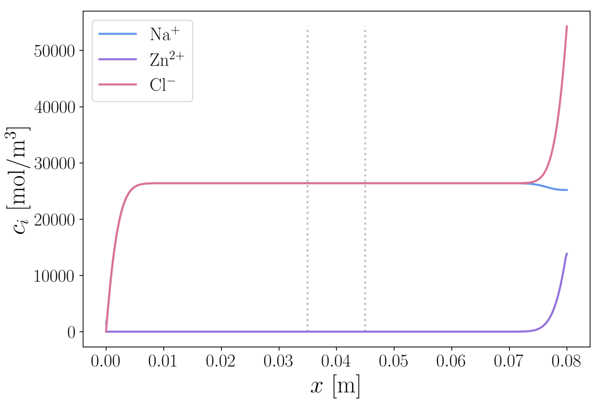
The plots of molar concentration distributions recorded at the end time for simulations FCC100, FCC1000, and FCC2000 are presented in Figure 7.7. The equivalent is done for simulations FCD100, FCD1000, and FCD2000 in Figure 7.8.

What can be clearly seen from these plots is that the simulations done with lower current densities exhibit smaller gradients in concentration near the left and right boundaries, meaning the transport is not as strongly dominated by migration as for the simulations run with higher current densities, and more diffusion has time to take place.

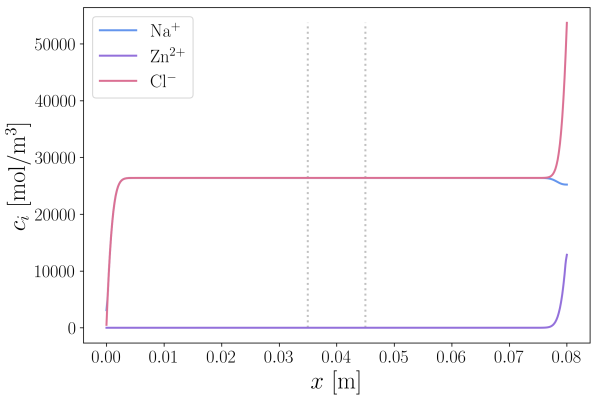
Under closer inspection, it is visible that the plots in Figures 7.7 and 7.8 indicate that the solution is electroneutral. This was further investigated, and for all molten salt simulations, the electrolyte solution was indeed electroneutral over the entire area at all recorded times.



(a) FCC100 after 46000 s



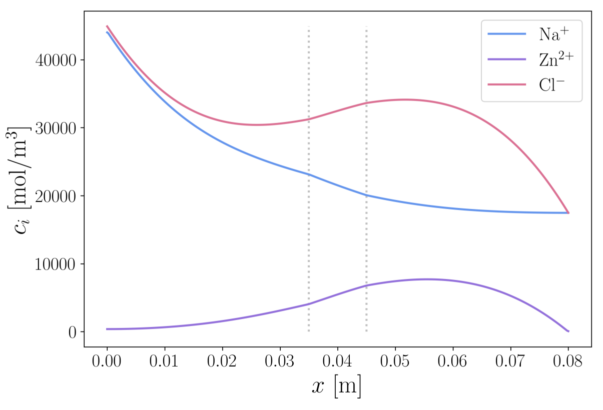
(b) FCC1000 after 460 s



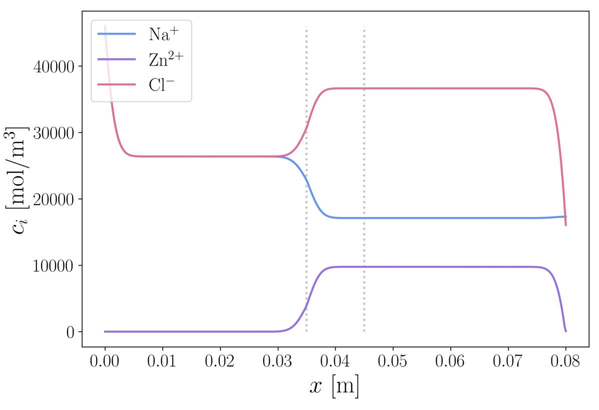
(c) FCC2000 after 110 s

Figure 7.7: Molar concentration distributions for fixed current charging battery simulations FCC100, FCC1000, and FCC2000 at the time the Na$^+$ concentration has passed zero (the dashed lines indicate the diaphragm's location).

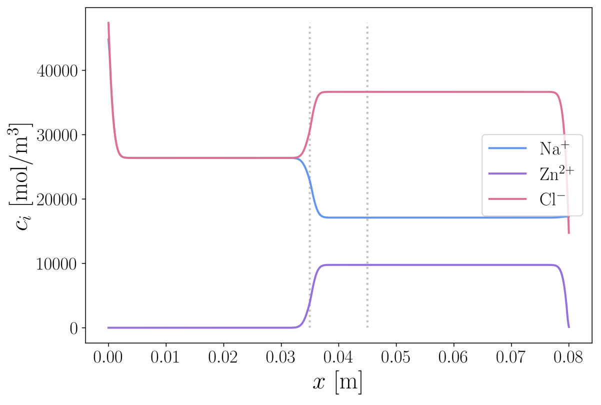
For the high fixed current density charging simulations (Figure 7.7), a dip in Na$^+$ concentration in the vicinity of the right boundary is observed. In contrast, a relatively smaller increase in Na$^+$ concentration near the right boundary is present for high fixed current discharge simulations (Figure 7.8).



(a) FCD100 after 22000 s



(b) FCD1000 after 255 s



(c) FCD2000 after 72 s

Figure 7.8: Molar concentration distributions for fixed current discharging battery simulations FCD100, FCD1000, and FCD2000 at the time the Zn^{2+} concentration has passed zero (the dashed lines indicate the diaphragm's location).

The electric potential values at the non-zero boundary were recorded over time for all fixed current simulations. Their values normalized against the starting electric potentials plotted against dimensionless time (normalized against the total time for each simulation) are presented in Figures 7.9 and 7.10 for charging and discharging simulations respectively. The relative decrease in the electric potential difference observed appears to scale linearly with inversed current density for both the charging and discharging processes simulated.

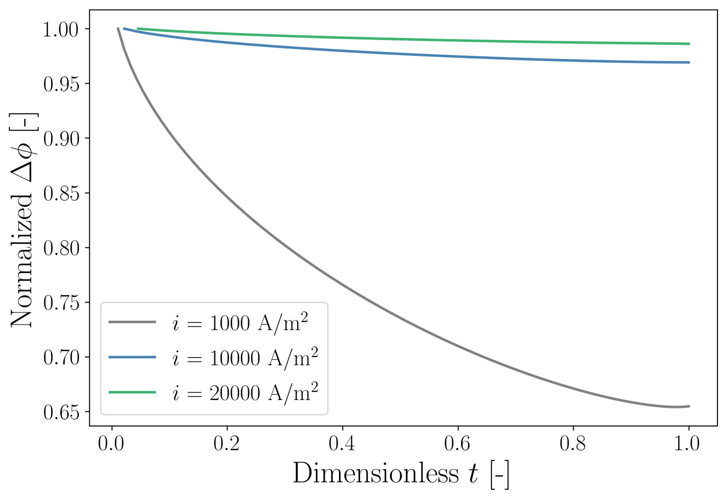


Figure 7.9: Electric potential at the non-zero boundary normalized against the first recorded value as a function of time normalized by the end time for simulations C100, C1000, and C2000.

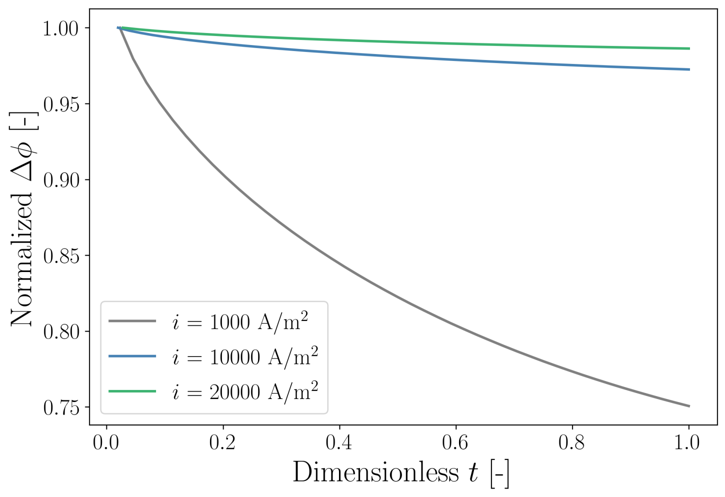
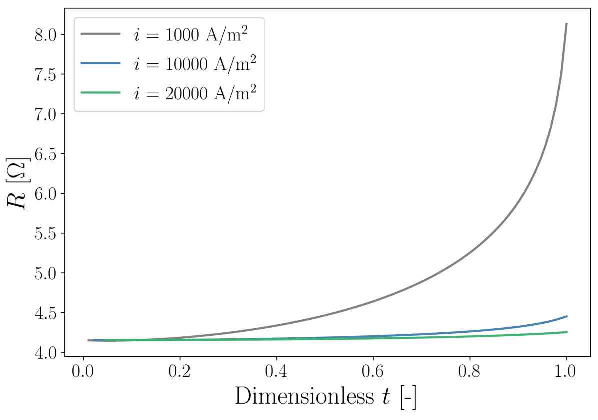
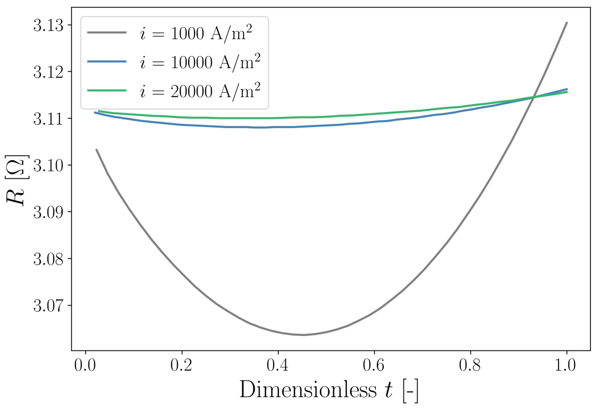


Figure 7.10: Electric potential at the non-zero boundary normalized against the first recorded value as a function of time normalized by the end time for simulations D100, D1000, and D2000.

The resistance in the molten salt electrolyte layer was recorded over time and plotted against time normalized against the total time for each simulation. These results are presented in Figure 7.11. These relative changes in the resistance values appear to correspond with the relative changes in the voltage discussed earlier.



(a) Charging simulations

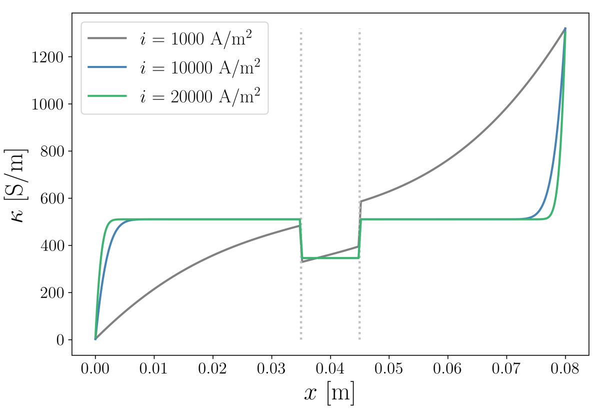


(b) Discharging simulations

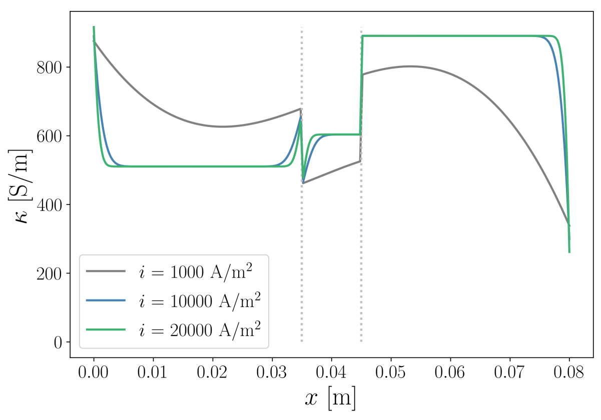
Figure 7.11: Electrolyte resistance values for fixed current charging and discharging simulations plotted against time normalized by the end time for each simulation.

The conductivity values across the length of the electrolyte layer (x-axis) at the last time step for all fixed current simulations are presented in Figure 7.12. The jumps in the conductivity values in the middle of the electrolyte layer are caused by there being less electrolyte volume in this region due to the presence of the porous diaphragm. Figure 7.13 shows the plot of resistivity values at the end time for fixed current charging simulations. These results are included to provide further insight into the conductivity and the resistance measurements. By looking at the conductivity plots alone, the reason why simulation FCC100 experiences the highest resistance might not be obvious since it appears to have a slightly higher average conductivity than simulations FCC1000 and FCC2000. This is because the resistance is not a function of the average conductivity, rather it is equal to the integral of resistivity averaged over the cross area across the length of the electrolyte layer.

One can think of a solution as multiple resistors connected in series in the direction of the current flow, each adding their contribution to the total resistance. By comparing the conductivity and resistivity plots presented in Figures 7.12a and 7.13 one can see that intervals with low concentration, and thus low conductivity, have a much larger contribution to the resistivity, and thus the total resistance, than the rest of the solution. Simultaneously, the intervals with high conductivity, do not cancel out those effects, they do not increase the total resistance. Thus solutions that end up with larger intervals with low concentration and low conductivity end up with the highest resistance.



(a) Charging simulations



(b) Discharging simulations

Figure 7.12: Electrolyte conductivity values for fixed current charging and discharging simulations plotted across the electrolyte layer thickness at the end time for each simulation (the dashed lines indicate the diaphragm's location).

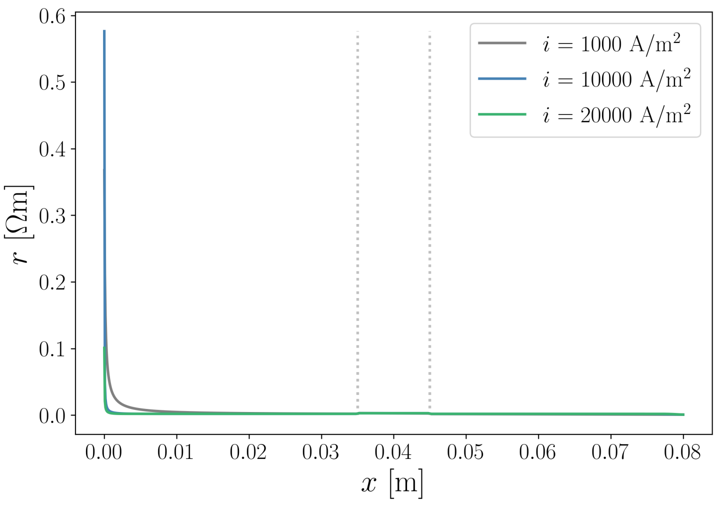


Figure 7.13: Electrolyte resistivity, r, values for fixed current charging simulations plotted across the electrolyte layer thickness at the end time for each simulation (the dashed lines indicate the diaphragm's location).

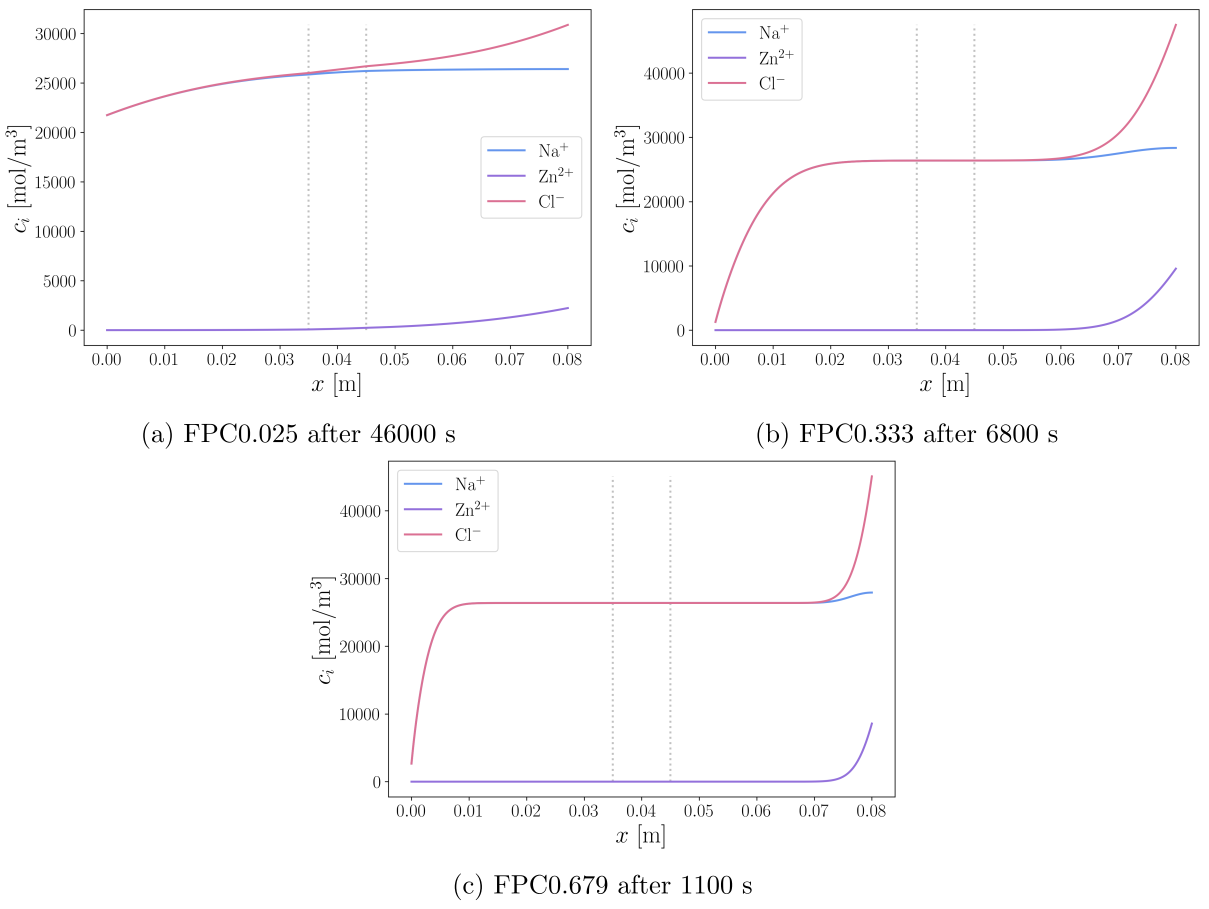
Cl^- ions are neither consumed nor produced by the cell reactions during charge and discharge. Therefore the amount of Cl^- in the solution should stay constant. This was checked for all simulations comparing the amount of Cl^- at the beginning of the simulations and at the end time. The relative change in the amount for fixed current simulations is presented in Table 7.2. The observed values are low corresponding well with the expected results. Out of all molten salt simulations conducted in the present work, the only simulations that exhibited a reduction in the amount of Cl^- in the electrolyte were fixed current charging simulations with a current of 100 mA/cm^2 (FCC100, PT0.6, PT0.85, and PT0.95) and fixed current discharging simulation with a current of 2000 mA/cm^2 (FCD2000). The values in general, do not appear random, rather they seem to be affected by the conditions under which the simulations are run. The relative change of the amount of Cl^- for the fixed current density simulations is the highest for the cases with the lowest applied current density (cases where diffusion is more dominant).

Table 7.2: The relative change in the number of mols of Cl^- in the electrolyte between the beginning and the end time of the fixed current simulations.

Simulation	Δc_{Cl^-} [mol%]
FCC100	-0.818
FCC1000	0.010
FCC2000	0.061
FCD100	0.548
FCD1000	0.039
FCD2000	-0.052

7.2.3 Fixed potential difference simulations

The produced molar concentration distributions for simulations created under the fixed potential difference conditions with values corresponding with the potentials differences observed during the fixed current experiments are presented in Figures 7.14 and 7.15 for charge and discharge processes respectively. The concentration distribution results for simulations done with the potential difference value corresponding to the reported values from the physical cell experiments [10] (0.99 V) are presented in Figure 7.16 both for charging and discharging.



(a) FPC0.025 after 46000 s

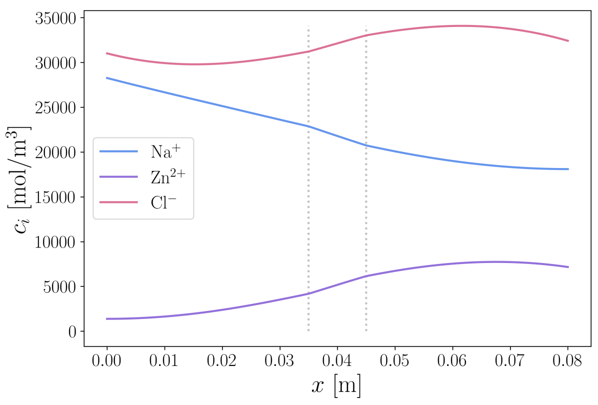
(b) FPC0.333 after 6800 s

(c) FPC0.679 after 1100 s

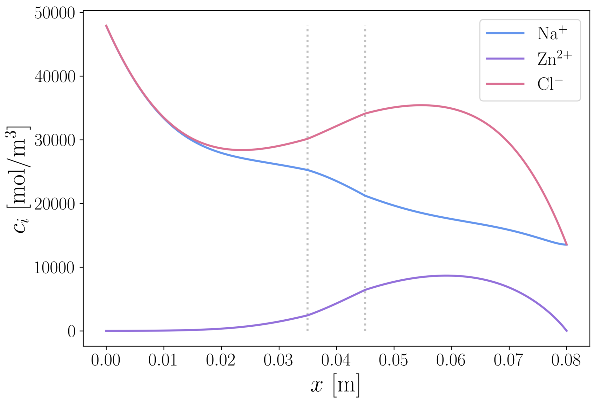
Figure 7.14: Molar concentration distributions for fixed electric potential difference charging battery simulations FPC0.025, FFC0.333, and FPC0.679 at the selected end times (the dashed lines indicate the diaphragm's location).

In Figure 7.14 (fixed potential difference charging simulations) an increase in Na^+ concentration near the right boundary is seen for the two simulations with higher potential differences (FPC0.333 and FPC0.679). A decrease in Na^+ concentration is seen in Figure 7.15 in the same location is seen for fixed potential difference discharge simulations with the two higher potential difference cases (FPD0.236 and FPD0.48). The same trends are observed for the charge and discharge simulations done with the highest potential difference (FPC0.99 and FPD0.99) showcased in Figure 7.16. It should be noticed that these observations are opposite to the ones done for the fixed current simulations, where a decrease of Na^+ concentration was seen for charging simulations and an increase for discharging.

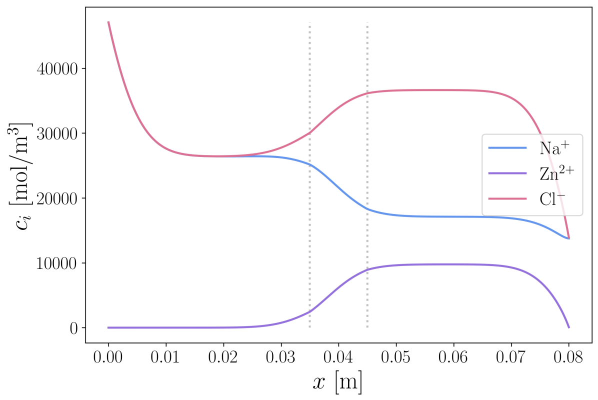
The fixed electric potential difference simulations done with the potential differences corresponding to the ones observed for the fixed electric current density cases (FPC0.025, FPC0.333, FPC0.679 and FPD0.018, FPD0.236, FPD0.48) exhibit higher end times than the corresponding fixed current density simulations. This is an expected trend caused by the differences in the boundary conditions. For fixed current simulations the concentration gradients at the boundaries are set, so the rate of diffusion transporting the ions through the boundary has a constant (relatively high) rate. This is the result of the diffusion-dominated boundary layer assumption. For fixed potential difference cases, the concentration gradient is created by migration and diffusion is merely a response to that gradient, thus much less diffusive transport through the boundary is present, and it is migration-dominated. This notion was confirmed with the total and diffusion current density measurements at the boundaries. Thus, only a (smaller) part of the potential difference goes to the migration current, resulting in a smaller concentration gradient and less active transport through the boundary.



(a) FPD0.018 after 46000 s

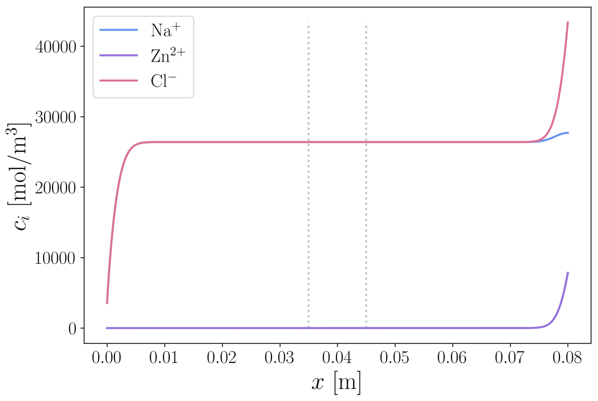


(b) FPD0.236 after 12200 s

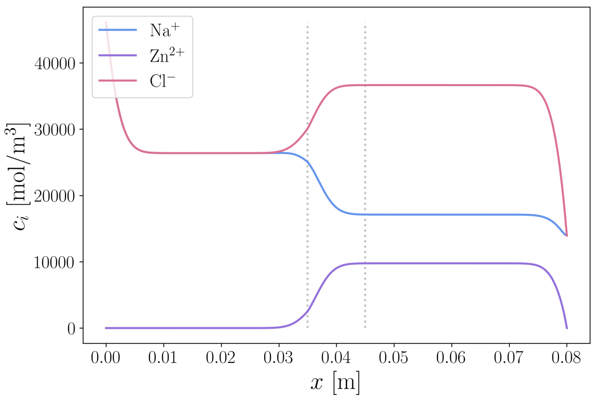


(c) FPD0.48 after 2800 s

Figure 7.15: Molar concentration distributions for fixed electric potential difference discharging battery simulations FPD0.018, FPD0.236, and FPD0.48 at the selected end times (the dashed lines indicate the diaphragm's location).



(a) FPC0.99 after 420 s



(b) FPD0.99 after 610 s

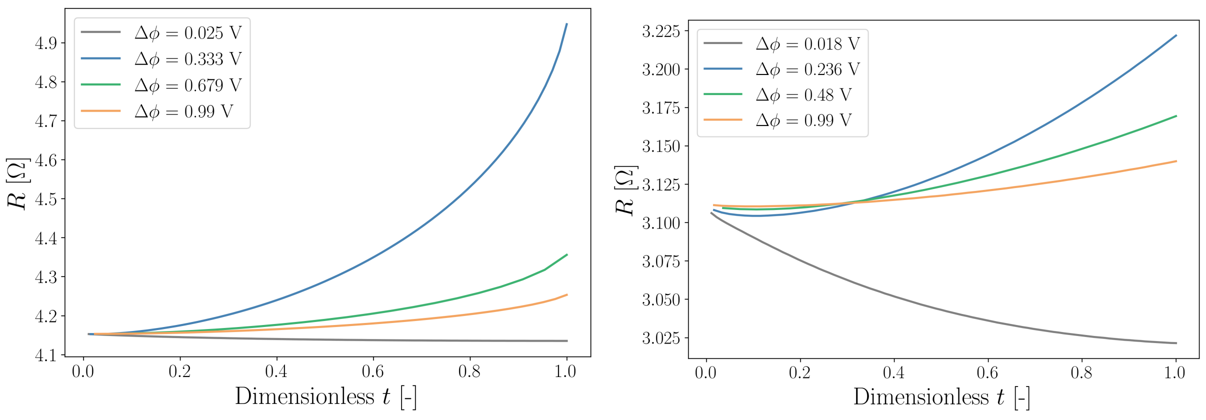
Figure 7.16: Molar concentration distributions for 0.99 V fixed electric potential difference charging and discharging battery simulations (FPC0.99 and FPD0.99) at the selected end times (the dashed lines indicate the diaphragm's location).

The resistance of the electrolyte layer measurements for all fixed electric potential difference cases across the time normalized by the end time values for each simulation are presented in Figure 7.17. Figure 7.18 shows the conductivity measurements for these simulations recorded at the last time steps and plotted across the length of the domain. In Figure 7.17 one can see that the charge and discharge simulations run with the lowest potential differences (FPC0.025 and FPD0.018) exhibit a time-dependent behaviour different to all other simulations. For these cases, the resistance decreases with time. These results go hand in hand with the concentration and conductivity distributions for these simulations presented in Figures 7.14a, 7.15a and 7.18. Due to the low potential difference values in these cases, the driving force for migration is low, resulting in small concentration gradients at the boundaries that the diffusion is capable of smoothing over even further. Thus, no regions with a low abundance of charged species are present for the low-potential cases, preventing the resistance from increasing.

For the rest of the fixed potential difference simulations, the resistance in general increases with decreasing potential difference used. This, again is explained by the conductivity results shown in Figure 7.18. For the high potential cases, the intervals with low conductivity are much smaller, thus they have a smaller contribution to the total resistance. The size of these intervals increases with decreasing potential differences used since diffusion has enough time to broaden them.

As illustrated by Figure 7.17b, for the discharging battery fixed potential difference cases FPD0.236, FPD0.48, and FPD0.99, the resistance values cross at a certain time. In the beginning, the resistance is the highest for the case with the highest potential difference and decreases with decreasing potential. This is likely due to the fact that in the beginning, the diffusion has not had enough time to make the intervals with low conductivity large enough for cases with lower potential differences to have a greater impact on the total resistance than the intervals with low conductivity of the high potential difference cases.

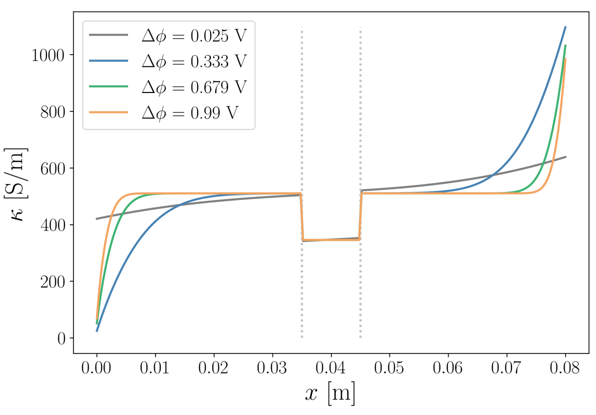
From concentration distributions plots shown in Figures 7.15 and 7.16b, one can see that by the end of the discharge simulations, the number of ions present for conduction on the right side from the diaphragm is much lower for the low potential difference cases, but in the beginning that is not the case. More ions have time to escape by means of diffusion before the concentration at the right boundary reaches zero, decreasing the conductivity. While in the beginning of the simulations, more ions get enough time to exit the area by means of migration for the high potential difference cases, due to a larger driving force.



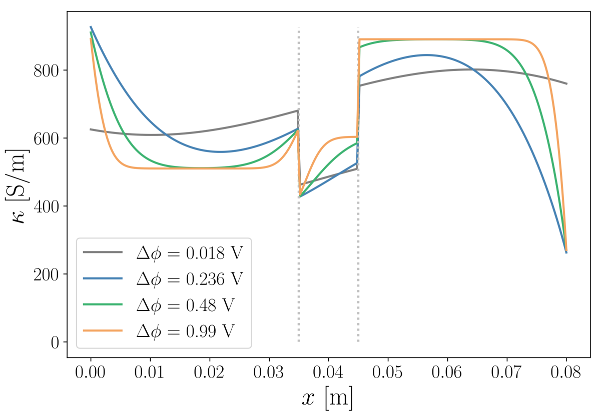
(a) Charging simulations

(b) Discharging simulations

Figure 7.17: Electrolyte resistance values for fixed potential difference charging and discharging simulations plotted against time normalized by the end time for each simulation.



(a) Charging simulations



(b) Discharging simulations

Figure 7.18: Electrolyte conductivity values for fixed potential difference charging and discharging simulations plotted across the electrolyte layer thickness at the end time for each simulation (the dashed lines indicate the diaphragm's location).

The relative changes in the number of moles of Cl^- ions in the electrolyte layer throughout the simulation time for fixed electric potential difference simulations are presented in Table 7.3. The calculated changes are low, indicating that the amount of Cl^- is relatively well conserved. For the fixed current density simulations, the cases with the lowest applied current densities exhibited the highest relative change of the amount of Cl^-, which would correspond with fixed potential difference simulations FPC0.025 and FPD0.018. However, the two fixed potential difference simulations exhibiting the highest degree of deviation from full Cl^- conservation were cases FPC0.333 and FPD0.236. Their values are the highest out of all molten salt simulations conducted in the present work, indicating that the initial and boundary conditions used for Na^+ and Zn^{2+} concentrations as well as for the electric potential affect these results.

Table 7.3: The relative change in the number of mols of Cl^- in the electrolyte between the beginning and the end time of the fixed electric potential difference simulations.

Simulation	Δc_{Cl^-} [mol%]
FPC0.025	0.230
FPC0.333	1.101
FPC0.679	0.622
FPC0.99	0.445
FPD0.018	0.635
FPD0.236	1.263
FPD0.48	0.657
FPD0.99	0.313

7.2.4 Diaphragms' parameter study

For the diaphragm parameter investigation simulations, the recorded resistance measurements are plotted against the dimensionless time normalized by the total time for each simulation and presented in Figure 7.19. The decrease in porosity and increase in tortuosity values of the diaphragm result in an increase in the total resistance of the electrolyte layer. This trend is expected since this increases the degree of obstruction for ion transport and the resistance contribution of the diaphragm itself increases, due to less volume being available for conducting fluid.

The time until the breakthrough of Zn^{2+} concentration through the left side of the diaphragm, which can be used as a measure of how well the diaphragm prevents the mixing of solutes in the two compartments of the electrolyte, increases with increasing tortuosity, which is expected because ions are forced to travel a longer distance, which takes more time. The relative increase in time, however, is not equal to the relative increase in tortuosity. The likely explanation for this discrepancy lies in the porosity values. Tortuosity is higher for the diaphragms with lower porosity, where the Zn^{2+} concentration has less volume to fill up until breakthrough, which takes less time. The relative increase in breakthrough time (0.12) is close to the value relative increase in tortuosity (0.27) multiplied by the relative decrease in porosity (0.37) between the most extreme cases tested ($\omega = 0.6$ and $\omega = 0.95$).

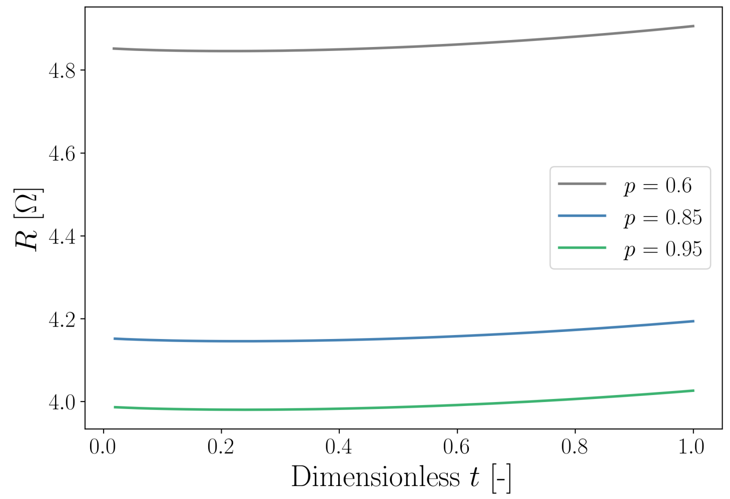


Figure 7.19: Electrolyte resistance plotted against time normalized by the end time for each of simulations PT0.6, PT0.85, and PT0.95.

The relative increase in the total resistance is higher than the relative increase in the time it takes Zn^{2+} to reach the left compartment of the electrolyte layer which is unfortunate for the battery's properties. Ideally, the diaphragm would prevent the movement of Zn^{2+} ions to prevent Zn co-deposition on the Na electrode and reduce self-discharge without increasing the resistance too much. To achieve that a faradaically selective membrane would be ideal.

The relative changes in the number of moles of Cl^- ions in the electrolyte layer across the simulation time for the diaphragm properties testing simulations are presented in Table 7.4. The values are negative indicating a reduction in the amount of Cl^- present. However, these values are still relatively low indicating a high degree of Cl^- mass conservation.

Table 7.4: The relative change in the number of mols of Cl^- in the electrolyte between the beginning and the end time of the diaphragm parameters testing simulations.

Simulation	Δc_{Cl^-} [mol%]
PT0.6	-0.317
PT0.85	-0.298
PT0.95	-0.294

These Cl^- conservation results for the molten salt simulations are summarized in Figure 7.20, excluding simulations FPC0.025 and FPD0.018 since they were not stopped at the time the reactant was depleted at the boundary like the rest of the simulations because the transport was slow for these simulations. For a given simulation type (either fixed current density or fixed potential difference) the absolute value of the relative change in the amount of Cl^- appears to increase with increasing end time. In addition, the fixed potential difference simulations exhibited higher degrees of deviation from total Cl^- conservation and a stronger dependence of it on the end time. This might be the result of the boundary condition implementation.

The fixed potential difference cases employ coded boundary conditions on both the left and the right boundaries simultaneously, while the fixed current density cases only use coded BC on one of these boundaries at a time. These coded BCs use discretized forms of equations (5.3) and (5.5) which results in numerical error. Even though no boundary conditions were applied for the Cl^- concentration equation, its solution is still affected by them, because it is calculated from the Na^+ and Zn^{2+} concentration solutions that are based on these BCs. The charge in the amount of Cl^- reflects the discrepancy between the change in the amount of Na^+ and the change in the amount of Zn^{2+}, which should correspond with half of the change in Na^+ to conserve the charge but in reality, some deviations are present. The comparison of the degree of Cl^- conservation in the fixed current density and fixed potential difference cases indicates that the Calculated concentration gradient BC introduced in the fixed potential difference simulations might have a greater influence of the deviations from full Cl^- conservation than the Zero flux BC used in both simulation types.

The end-time dependency of these deviations might be explained by numerical errors adding up and growing over time, doing so more quickly for the fixed potential difference cases due to those cases employing two coded BCs simultaneously instead of one as done in fixed current density cases.

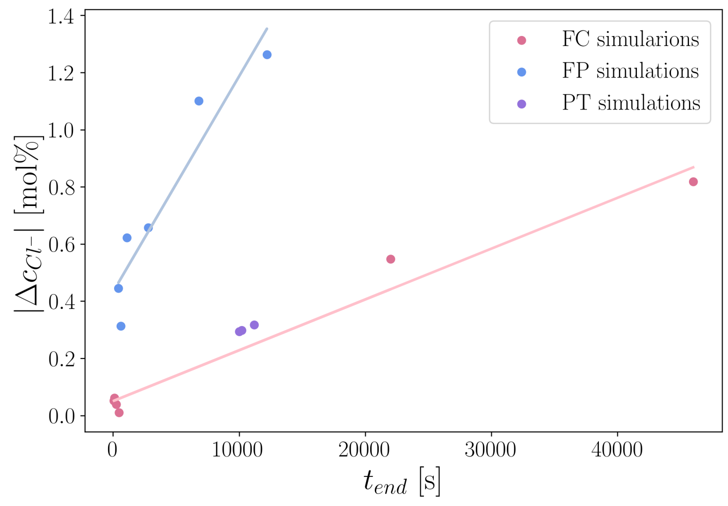


Figure 7.20: Absolute values of mol% change in the amount of Cl^- ions in the electrolyte against the simulation end times for fixed electric current (FC), fixed potential difference (FP, excluding cases FPC0.025 and FPD0.018), and diaphragm parameter testing (PT) simulations with trend lines for FC and FC cases.

8 Discussion

8.1 Aqueous model and simulations

Overall, a good agreement is observed between the results generated by the model implemented in the slayFoam solver and the results reported by Rolle et al. [49]. The tracer plumes exhibit similar behaviour to the reported ones, with equal centre of mass displacements in the correct direction produced using fitted tortuosity values (1.29-1.44) very close to the ones used in the paper (1.3-1.43). However, the shapes and sizes of the plumes differ from the reported ones and are in general smaller and for the divalent and trivalent tracer compounds less symmetrical as well. Permanganate tracer exhibited a shape most similar to the reported ones, which is likely due to it being a monovalent ion. The charges of the ions affect the concentration distributions because of interspecies interactions [49]. These interactions were neglected in the model, but that effect was likely the weakest for the monovalent ion, resulting in a more similar plume shape.

8.1.1 Errors and uncertainties

Numerical error

Iterative error from the matrix solutions for these simulations was kept low at 10^{-6} for concentration equations and 10^{-8} for the electric potential. Thus, it is likely not the greatest contribution to the total error of the solutions.

The values of the discretization errors are not known for the aqueous solution simulations, since convergence testing was not conducted due to limitations within the time budget. However, the orders of accuracy of the discretization methods used are known and can provide some insight into the error present. The temporal discretization is first-order accurate ($\mathcal{O}(\Delta t)$), while the interpolation, gradient, divergence and Laplacian schemes were all second-order accurate ($\mathcal{O}(\Delta x^2)$). The error from the temporal discretization is estimated from the time step size to be in the order of 10 (notice that this error size is relevant for the time in the simulations, not the concentrations even though it contributes to it) whereas the error due to spatial discretization is estimated to be in the order of 10^{-6}.

Uncertanties

Several assumptions and simplifications were made in the model. Firstly, the ions present in the background electrolyte were not modelled. Thus, their concentration was assumed to be constant and homogeneous, and their presence was only accounted for in terms of their contribution to the electric conductivity of the system. As a result, the interspecies interactions were neglected and the background conductivity was assumed constant and homogeneous. This assumption and its implications are likely the largest contributors to the uncertainty in the system and are the probable explanation for the discrepancy in the shapes and sizes of the plumes.

Secondly, the contribution of the sum of diffusion currents in the electric potential equation (3.14) was neglected assuming that the contributions of the different compounds cancel each other out. However, the diffusion coefficients of ions present in the solution were not equal with some being around three times higher than others (for Na^+ and New Coccine^{3+}), meaning this assumption contributed to the uncertainty in the simulations. In addition, this form of the electric potential equation was used to avoid resolving the electrical double layer and reduce computational time. Meanwhile, Rolle et al. [49] employed a different form of the electric potential equation (3.9). This can also be a possible explanation for the observed discrepancies in the shapes and sizes of the plumes.

The same compound (such as diffusivities and mobilities) and system properties (such as domain size, 2D simplification, and system temperature) were used in the present work as in the work of Rolle et al. [49]. The fitted tortuosity values were also very similar to the reported ones. Thus, these simplifications likely did not introduce significant uncertainty to the simulations when compared to the simulations presented in the article. However, they do result in deviations from the results of the conducted physical experiments.

The concentration boundary conditions likely did not have a large impact on the uncertainty in the system as well, since the properties investigated (plume concentrations) did not cross or interact with any boundaries. The potential boundary conditions were simplified and average values of the recorded experimental electric potential values were used. This could have an impact on the migration and is a source of uncertainty.

8.2 Molten salt simulations

The created model is capable of simulating Na^+, Zn^{2+}, and Cl^- concentration distributions in the electrolyte layer of the Na-Zn LMB under various conditions, such as charging and discharging battery with either fixed electric current density or fixed electric potential difference applied. However, it can be applied to other systems with three ionic species being transported by migration and diffusion.

The aqueous simulations provided insight that even with uncertainties present and simplifications introduced an ionic transport model based on the Nernst-Planck equation coupled with an equation for electric potential can be implemented in OpenFOAM software producing relatively accurate results. However, the executed implementation has certain limitations.

One of the limitations of the developed model is its' lack of consideration of density gradients and volume changes. As a result, the simulations must be stopped when the concentrations start to reach zero at either the left of the right boundary, making it impossible to model the full processes of charge and discharge. This limits the highest electric current density or voltage that can be applied in the model because the concentrations reach zero at the boundaries rapidly. This effect is not physical, because advection would prevent such density gradients from taking place.

In the results, a trend is observed that in cases with a lower applied current density or a lower fixed electric potential difference across the electrolyte, the mass transport is more strongly influenced by diffusion. Migration becomes more pronounced with increasing electric current and electric potential differences. These effects correspond well with the way migration and diffusion are understood in the literature [40, 43]. In addition, for all cases, the concentrations of the reactive species sink where they are consumed and increase where they are produced. These results help confirm that in general, the solutions to the model behave in ways they are expected to.

The resistance and potential difference measurements illustrate that the concentration distributions (and thus the mass transport) have a large influence on the properties of the electrolyte important to the functioning of the battery, such as the total resistance, the ohmic drop, and thus the coulumbic efficiency of the battery. Avoiding the formation of regions with low concentrations of highly-conductive species is expected to help keep the total resistance of the electrolyte layer minimal. No regions with very low total concentration would form in real life as they do in the simulations because convection would prevent this from happening. However, regions with low concentrations of the reacting species would still increase the resistivity (or concentration overpotential) and are thus unwanted. This illustrates the fact that even though a separation of the electolyte layer into two compartments is wanted for decresing self-discharge rate and the amount of Zn co-deposition on the Na electrode, a certain degree of electrolyte mixing is desired to minimize the resistance.

What can clearly be seen from the results of the diaphragm parameter testing simulations is that the total resistance of the electrolyte layer with a diaphragm in place increases with decreasing diaphragm porosity (and the following increase in tortuosity). This corresponds with the behavior expected from the literature [43]. The resulting increase in the resistance was higher than the resulting increase in the time passed until Zn^{2+} has exited the left side of the diaphragm. This indicates that a porous diaphragm might not be the best choice of a separator in the battery. However, in physical cells with convection in place, the diaphragm might reduce the advective transport of Zn^{2+} ions to the electrolyte close to the Na electrode more effectively than it does migration and diffusion, proving it to be a very effective choice. This makes models that include migration, diffusion, and advection more so interesting to investigate.

8.2.1 Errors and uncertainties

Numerical error

The iterative error for the molten salt model simulations was set to 10^{-6} for the mass balance equations and 10^{-8} for the electric potential. These values are low and likely do not affect the results significantly.

The discretization error was estimated and managed by means of grid and time step convergence testing. By using the grid resolution and time steps at which the solution appeared to have converged discretization error was kept minimal.

Uncertanties

The model was built under the assumption that the Nernst-Planck equation is valid for the modelled system. Nevertheless, the used form of the Nernst-Planck equation is only valid for dilute ideal solutions, which does not reflect the modelled system. However, the Nernst-Planck equation is considered to exhibit good accuracy even with deviations from the necessary assumptions [43] and in the literature on modelling of migration and diffusion in LMB electrolyte layer based on the Poisson-Nernst-Planck model using OpenFOAM the simulation results were validated against experimentally obtained data with a fairly good agreement [30]. Still, the degree of uncertainty in the present work resulting from these assumptions is difficult to access quantitatively due to the lack of experimentally obtained results.

The advective flux was neglected in the constructed model, which not only results in limitations for the model but also contributes to its uncertainty. Advection is expected in a fully liquid cell [13] and would contribute to the solute transport and thus resistance as well.

In addition, the created models are realized on a quasi-2D domain, while the batteries constructed in real life are, of course, three-dimensional. The uncertainty caused by neglecting the

3D effects is difficult to quantify. However, the boundary and initial conditions for the models would be constant in the third neglected direction and are not expected to contribute with significant new effects.

In the constructed Na-Zn LMB cells $CaCl_2$ is present in the electrolyte layer [10, 19]. In the model, however, NaCl and $ZnCl_2$ were assumed to be the only salts composing the electrolyte. This simplification affects the accuracy of the results because all interspecies interactions as well as conductivity contributions of $CaCl_2$ are being neglected.

The way the initial concentrations on the right side of the diaphragm for the discharging battery simulations were chosen, used values that were not well corresponding with each other. Density data with different concentrations was used due to human error, and in hindsight, it would be better to just use 50 mol% $ZnCl_2$. However, these concentrations are in a way arbitrary, since different concentrations can be observed in the electrolyte depending on how charged or discharged it is and where it is in the process. The density might not be realistic, but convection was not modelled, so these effects were not as important.

The diffusion coefficients are assumed to be constant scalar properties for all conducted simulations, which is not the case in reality. Most of the material properties used were taken at different temperatures that do not correspond with the temperature of the modelled cell, due to the unavailability of the desired data, caused by neglecting the presence of $CaCl_2$. In addition, due to the lack of data for the diffusivity of Zn^{2+} ions in NaCl molten salt, the diffusivity of Cd^{2+} was used, contributing to the uncertainty of the simulations. Including $CaCl_2$ in the simulations would allow one to use more accurate data for material properties to decrease the uncertainty.

The reported dips and increases in Na^+ concentration near the right boundary observed for higher current density and potential difference cases (FCC1000, FCC2000, FCD1000, FCD2000, FPC0.333, FPC0.679, FPC0.99, FPD0.236, FPD0.48, and FPD0.99) did not have any obvious physical reason and are likely a result of the Na^+ concentration boundary condition at the left boundary. An example of such Na^+ concentration dip is presented in Figure 8.1 for the FPD0.99 case. The zero flux BC is applied for Na^+ at the right boundary near this region. This condition likely forced such concentration gradients to prevent the migration and diffusion of the Na^+ ions through the right boundary and they are likely an artefact of the zero flux boundary condition implementation.

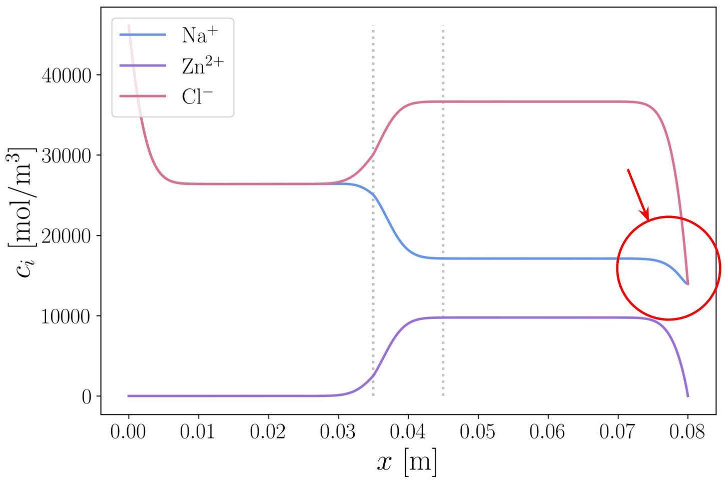


Figure 8.1: Molar concentration distributions for discharging 0.99 V fixed electric potential difference simulation (FPD0.99) at the end time (a dip in Na^+ concentration is highlighted by the red circle and arrow).

The degree of deviation from the total conservation of the non-reacting Cl^- ions (and thus the deviation from the conservation of charge) for the molten salt simulations was likely caused by the uncertainty introduced by the boundary condition implementation. This would explain the fixed potential difference simulations having a lower degree of conservation, since they, in addition to the coded zero flux BC, employ the coded calculated concentration gradient BC. In it, the value of conductivity in the cell centre next to the boundary patch is used for computation, since using its value at the patch would result in the concentration gradient at the patch being calculated from the conductivity value that is in turn calculated using the concentration values at the patch that are not known. Using the value close to the patch thus introduces uncertainty that comes in addition to the uncertainty caused by the use of the discretized value of the electric potential gradient that is present in both the calculated gradient and zero flux BCs. Nevertheless, the total deviations of the amount of Cl^- are generally low, but could still be improved by finding a different way of approaching the BC implementation.

9 Concluding remarks

In the present work, the electrokinetic transport of ionic species in the molten salt electrolyte of the Na-Zn liquid metal battery with a porous diaphragm in place was investigated by means of numerical modelling.

Firstly, an overview of and some general background on the electrokinetic mass transfer in aqueous solutions and molten salt systems with a porous material obstruction were given. Based on this knowledge, two Darcy-scale models were implemented in an open-source computational fluid dynamics software OpenFOAM. One solver for simulating transient electrokinetic transport in aqueous solutions in the presence of a porous obstruction and one for doing so with molten salt solutions. The aqueous solution model can be thought of as a special case of the main Poisson-Nernst-Planck molten salt model with relaxed electroneutrality condition and was created to test and verify the transport behaviour predicted by the model against the results reported in the literature. These tests have revealed that the model predicted the transport of ions relatively well, with them moving the expected distance in the correct direction. However, this model systematically underpredicted the amount of spreading of the tracer compounds and thus did not produce expected plume shapes and sizes. This was likely a result of the simplifications introduced in the created model and caused by not modelling the ions present in the background electrolyte and using a different electric potential equation than the one used in the article [49].

The molten salt model was applied to the simplified Na-Zn LMB electrolyte layer with a porous diaphragm under various conditions: fixed current density at the boundaries for charging and discharging processes, fixed electric potential difference across the system for charging and discharging processes. Different diaphragm porosity and tortuosity values were tested as well under charging fixed current density conditions. This provided insight into ways these processes can be modelled and the impact various boundary and initial conditions as well as diaphragm properties have on the transport.

Additionally, some interesting observations were made. The concentration distributions appear to have a significant influence on the resistance in the electrolyte layer, making the transport of ions important to understand and manage. The diaphragms' properties affect the mass transport and electrolyte layer properties as well. A decrease in porosity and an increase in tortuosity resulted in an increase in resistivity and in the time it takes Zn^{2+} to cross the diaphragm. However, the observed relative increase in the total resistance was higher than the relative increase in the time it took Zn^{2+} to break through the diaphragm and reach the left compartment electrolyte layer. Since one of the main purposes of the diaphragm is to control the transport of Zn^{2+} ions and a minimal resistance in the electrolyte layer is desired, such porous diaphragm is thus not the optimal choice for a separator in a Na-Zn LMB cell where the mass transport is dominated by diffusion and migration. However, it may perform much better in its role as a separator in a cell with convection, making such scenarios important to investigate before drawing final conclusions concerning its use in physical Na-Zn LMBs.

10 Future work

Because advection is believed to be the dominating mass transport mechanism in the molten salt electrolyte layer of LMBs, it is important to include it in future modelling work and create models with both electrokinetic and advective transport mechanisms while taking the presence of a porous diaphragm into consideration. The ways a porous diaphragm would affect the convection in the cell are of great interest since they would impact the important properties of the cell, such as concentration distributions, resistance, coulombic efficiency and more. Accounting for advection would also decrease the limitations of the model and allow one to run simulations under higher current density or electric potential difference conditions by evening out the observed unrealistic density gradients.

Future models including $CaCl_2$ as a part of the molten salt electrolyte of the Na-Zn liquid metal battery should be investigated. This will help reduce the uncertainty and produce more accurate results. Expanding the model implementation shown in the present work can be a possible way of doing so.

To combat the uncertainties introduced by the boundary condition implementation in the molten salt simulations likely causing the deviations from Cl^- and charge conservation as well as unexpected behaviour of Na^+ concentration near the right boundary, new ways of implementation should be investigated.

Verifying the model against experimental results for a molten salt system would help provide more insight into the discrepancy in results introduced by uncertainties of the model. This, however, can prove practically challenging, due to high temperatures and would likely only be possible for a model that includes advection, eliminating the possibility of investing diffusion and migration alone.

In general, recreating the previous work on the modelling of different mechanisms of mass transport in LMBs under various conditions and now including a porous diaphragm would be interesting to further understand its impact on mass transport and battery properties, enabling one to optimize them.

Bibliography

[1] S. Bilgen. Structure and environmental impact of global energy consumption. *Renewable and Sustainable Energy Reviews*, 38:890–902, oct 2014.

[2] M. M. Rahman, A. O. Oni, E. Gemechu, and A. Kumar. Assessment of energy storage technologies: A review. *Energy Conversion and Management*, 223:113–295, nov 2020.

[3] M. S. Whittingham. History, evolution, and future status of energy storage. *Proceedings of the IEEE*, 100(Special Centennial Issue):1518–1534, May 2012.

[4] H. S. Hirsh, Y. Li, D. H. S. Tan, M. Zhang, E. Zhao, and Y. S. Meng. Sodium-ion batteries paving the way for grid energy storage. *Advanced Energy Materials*, 10(32):2001274, 2020.

[5] A. Blakers, M. Stocks, B. Lu, and C. Cheng. A review of pumped hydro energy storage. *Progress in Energy*, 3(2):022003, March 2021. Publisher: IOP Publishing.

[6] X. Fan, B. Liu, J. Liu, J. Ding, X. Han, Y. Deng, X. Lv, Y. Xie, B. Chen, W. Hu, and C. Zhong. Battery Technologies for Grid-Level Large-Scale Electrical Energy Storage. *Transactions of Tianjin University*, 26(2):92–103, April 2020.

[7] A. A. Yaroshevsky. Abundances of chemical elements in the Earth's crust. *Geochemistry International*, 44(1):48–55, January 2006.

[8] K. Wang, K. Jiang, B. Chung, T. Ouchi, P. J. Burke, D. A. Boysen, D. J. Bradwell, H. Kim, U. Muecke, and D. R. Sadoway. Lithium–antimony–lead liquid metal battery for grid-level energy storage. *Nature*, 514(7522):348–350, oct 2014.

[9] J. G. Simpson, G. Hanrahan, E. Loth, G. M. Koenig, and D. R. Sadoway. Liquid metal battery storage in an offshore wind turbine: Concept and economic analysis. *Renewable and Sustainable Energy Reviews*, 149:111–387, oct 2021.

[10] J. Xu, O. S. Kjos, K. S. Osen, A. M. Martinez, O. E. Kongstein, and G. M. Haarberg. Na-Zn liquid metal battery. *Journal of Power Sources*, 332:274–280, nov 2016.

[11] P. Personnettaz, S. Landgraf, M. Nimtz, N. Weber, and T. Weier. Mass transport induced asymmetry in charge/discharge behavior of liquid metal batteries. *Electrochemistry Communications*, 105:106496, August 2019.

[12] H. Kim, D. Boysen, J. Newhouse, B. Spatocco, B. Chung, P. Burke, D. Bradwell, K. Jiang, A. Tomaszowska, K. Wang, W. Wei, L. Ortiz, S. Barriga, S. Poizeau, and D. Sadoway. Liquid Metal Batteries: Past, Present, and Future. *Chemical reviews*, 113, nov 2012.

[13] D. H. Kelley and T. Weier. Fluid mechanics of liquid metal batteries. *Applied Mechanics Reviews*, 70(2), January 2018.

[14] R. D. Deshpande, J. Li, Y.-T. Cheng, and M. W. Verbrugge. Liquid metal alloys as self-healing negative electrodes for lithium ion batteries. *Journal of The Electrochemical Society*, 158(8):A845, May 2011. Publisher: IOP Publishing.

[15] S. Zhang, Y. Liu, Q. Fan, C. Zhang, T. Zhou, K. Kalantar-Zadeh, and Z. Guo. Liquid metal batteries for future energy storage. *Energy & Environmental Science*, 14(8):4177–4202, August 2021. Publisher: The Royal Society of Chemistry.

[16] H. Zhou, H. Li, Q. Gong, S. Yan, X. Zhou, S. Liang, W. Ding, Y. He, K. Jiang, and K. Wang. A sodium liquid metal battery based on the multi-cationic electrolyte for grid energy storage. *Energy Storage Materials*, 50:572–579, September 2022.

[17] Solstice project. `https://www.solstice-battery.eu`.

[18] K. Koseniuk. Modelling of mass transport through porous diaphragms in na-zn liquid metal batteries. *Project thesis*, 2022.

[19] J. Xu, A. M. Martinez, K. Sende Osen, O. S. Kjos, O. E. Kongstein, and G. M. Haarberg. Electrode Behaviors of Na-Zn Liquid Metal Battery. *Journal of The Electrochemical Society*, 164(12):A2335, August 2017. Publisher: IOP Publishing.

[20] A. Tornheim and D. C. O'Hanlon. What do Coulombic Efficiency and Capacity Retention Truly Measure? A Deep Dive into Cyclable Lithium Inventory, Limitation Type, and Redox Side Reactions. *Journal of The Electrochemical Society*, 167(11):110520, July 2020. Publisher: IOP Publishing.

[21] T. Weier, A. Bund, W. El-Mofid, G. M. Horstmann, C.-C. Lalau, S. Landgraf, M. Nimtz, M. Starace, F. Stefani, and N. Weber. Liquid metal batteries - materials selection and fluid dynamics. *IOP Conference Series: Materials Science and Engineering*, 228(1):012013, July 2017. Publisher: IOP Publishing.

[22] W. Herreman, S. Bénard, C. Nore, P. Personnettaz, L. Cappanera, and J.-L. Guermond. Solutal buoyancy and electrovortex flow in liquid metal batteries. *Phys. Rev. Fluids*, 5:074501, Jul 2020.

[23] OpenFOAM. `https://www.openfoam.com`, December 2022.

[24] H. Li, H. Yin, K. Wang, S. Cheng, K. Jiang, and D. R. Sadoway. Liquid Metal Electrodes for Energy Storage Batteries. *Advanced Energy Materials*, 6(14):1600483, 2016.

[25] H. Kim, D. A. Boysen, J. M. Newhouse, B. L. Spatocco, B. Chung, P. J. Burke, D. J. Bradwell, K. Jiang, A. A. Tomaszowska, K. Wang, W. Wei, L. A. Ortiz, S. A. Barriga, S. M. Poizeau, and D. R. Sadoway. Liquid Metal Batteries: Past, Present, and Future. *Chemical Reviews*, 113(3):2075–2099, March 2013. Publisher: American Chemical Society.

[26] M. Azevedo, N. Campagnol, T. Hagenbruch, K. Hoffman, A. Lala, and O. Ramsbottom. Lithium and cobalt. *A Tale of Two Commodities*, 2018.

[27] T. Oshima, M. Kajita, and A. Okuno. Development of Sodium-Sulfur Batteries. *International Journal of Applied Ceramic Technology*, 1(3):269–276, 2004. https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1744-7402.2004.tb00179.x.

[28] K. B. Hueso, M. Armand, and T. Rojo. High temperature sodium batteries: status, challenges and future trends. *Energy & Environmental Science*, 6(3):734–749, 2013.

[29] H. Kim, D. A. Boysen, T. Ouchi, and D. R. Sadoway. Calcium–bismuth electrodes for large-scale energy storage (liquid metal batteries). *Journal of Power Sources*, 241:239–248, November 2013.

[30] C. Duczek, N. Weber, O. E. Godinez-Brizuela, and T. Weier. Simulation of potential and species distribution in a Li|| Bi liquid metal battery using coupled meshes. *Electrochimica*

Acta, 437:141413, 2023.

[31] D. Agarwal, R. Potnuru, C. Kaushik, V. R. Darla, K. Kulkarni, A. Garg, R. K. Gupta, N. Tiwari, and K. S. Nalwa. Recent advances in the modeling of fundamental processes in liquid metal batteries. *Renewable and Sustainable Energy Reviews*, 158:112–167, April 2022.

[32] X. Zhou, C. Gao, Y. Shen, H. Li, S. Yan, H. Zhou, K. Wang, and K. Jiang. Multi-field coupled model for liquid metal battery: Comparative analysis of various flow mechanisms and their effects on mass transfer and electrochemical performance. *Energy Reports*, 8:5510–5521, November 2022.

[33] R. F. Ashour and D. H. Kelley. Convection-Diffusion Model of Lithium-Bismuth Liquid Metal Batteries. In G. Lambotte, J. Lee, A. Allanore, and S. Wagstaff, editors, *Materials Processing Fundamentals 2018*, The Minerals, Metals & Materials Series, pages 41–52, Cham, 2018. Springer International Publishing.

[34] N. Weber, V. Galindo, F. Stefani, and T. Weier. The Tayler instability at low magnetic Prandtl numbers: between chiral symmetry breaking and helicity oscillations. *New Journal of Physics*, 17(11):113013, October 2015. Publisher: IOP Publishing.

[35] H.-C. Yeh, M. Wang, C.-C. Chang, and R.-J. Yang. Fundamentals and Modeling of Electrokinetic Transport in Nanochannels. *Israel Journal of Chemistry*, 54(11-12):1533–1555, 2014.

[36] A. Alizadeh, W.-L. Hsu, M. Wang, and H. Daiguji. Electroosmotic flow: From microfluidics to nanofluidics. *ELECTROPHORESIS*, 42(7-8):834–868, 2021.

[37] J. Isidro, R. López-Vizcaíno, A. Yustres, C. Sáez, V. Navarro, and M. A. Rodrigo. Recent progress in physical and mathematical modelling of electrochemically assisted soil remediation processes. *Current Opinion in Electrochemistry*, 36:101115, December 2022.

[38] P.-W. Huang, B. Flemisch, C.-Z. Qin, M. O. Saar, and A. Ebigbo. Validating the Nernst-Planck transport model under reaction-driven flow conditions using RetroPy v1.0. *EGUsphere*, pages 1–39, November 2022. Publisher: Copernicus GmbH.

[39] S. Molins, C. Soulaine, Ni. I. Prasianakis, A. Abbasi, P. Poncet, Anthony J. C. Ladd, V. Starchenko, S. Roman, D. Trebotich, H. A. Tchelepi, and C. I. Steefel. Simulation of mineral dissolution at the pore scale with evolving fluid-solid interfaces: review of approaches and benchmark problem set. *Computational Geosciences*, 25(4):1285–1318, August 2021.

[40] T. Novotný and B. Gaš. Mathematical model of electromigration allowing the deviation from electroneutrality. *ELECTROPHORESIS*, 42(7-8):881–889, 2021.

[41] M. Rolle, R. Sprocati, M. Masi, B. Jin, and M. Muniruzzaman. Nernst-Planck Based Description of Transport, Coulombic Interactions and Geochemical Reactions in Porous Media: Modeling Approach and Benchmark Experiments. *Water Resources Research*, 54, April 2018.

[42] M. Muniruzzaman, C. Haberer, P. Grathwohl, and M. Rolle. Multicomponent ionic dispersion during transport of electrolytes in heterogeneous porous media: Experiments and model-based interpretation. *Geochimica et Cosmochimica Acta*, 141:656–669, September 2014.

[43] J. Newman and K.E. Tomas-Alyea. *Electrochemical systems*, chapter 11, 12. John Wiley

and Sons, New Jersey, 3 edition, 2004.

[44] R. Sprocati and M. Rolle. On the interplay between electromigration and electroosmosis during electrokinetic transport in heterogeneous porous media. *Water Research*, 213:118161, April 2022.

[45] COMSOL Multiphysics. `https://www.comsol.com`.

[46] F. Pimenta and M.A. Alves. Numerical simulation of electrically-driven flows using openfoam, 2018.

[47] V. Zadin, D. Brandell, H. Kasemägi, A. Aabloo, and J. O. Thomas. Finite element modelling of ion transport in the electrolyte of a 3D-microbattery. *Solid State Ionics*, 192(1):279–283, June 2011.

[48] D. Danilov and P. H. L. Notten. Mathematical modelling of ionic transport in the electrolyte of Li-ion batteries. *Electrochimica Acta*, 53(17):5569–5578, July 2008.

[49] M. Rolle, M. Albrecht, and R. Sprocati. Impact of solute charge and diffusion coefficient on electromigration and mixing in porous media. *Journal of Contaminant Hydrology*, 244:103933, January 2022.

[50] D. L. Parkhurst and L. Wissmeier. PhreeqcRM: A reaction module for transport simulators based on the geochemical model PHREEQC. *Advances in Water Resources*, 83:176–189, September 2015.

[51] C. Liu, J. Shang, and J. M. Zachara. Multispecies diffusion models: A study of uranyl species diffusion. *Water Resources Research*, 47(12), 2011.

[52] P. Rasouli, C. I. Steefel, K. U. Mayer, and M. Rolle. Benchmarks for multicomponent diffusion and electrochemical migration. *Computational Geosciences*, 19(3):523–533, June 2015.

[53] J. Bear. *Modeling Phenomena of Flow and Transport in Porous Media.* Springer, January 2018.

[54] B. J. Kirby. *Micro- and Nanoscale Fluid Mechanics: Transport in Microfluidic Devices.* Cambridge University Press, Cambridge, 2010.

[55] C. Peng, J. O. Almeira, and A. Abou-Shady. Enhancement of ion migration in porous media by the use of varying electric fields. *Separation and Purification Technology*, 118:591–597, October 2013.

[56] L. Shen and Z. Chen. Critical review of the impact of tortuosity on diffusion. *Chemical Engineering Science*, 62(14):3748–3755, July 2007.

[57] J. Cai, W. Wei, X. Hu, and David A. Wood. Electrical conductivity models in saturated porous media: A review. *Earth-Science Reviews*, 171:419–433, August 2017.

[58] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics.* Springer Cham, 1 edition, 2015.

[59] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics.* Springer, Berlin, Heidelberg, 2002.

[60] F. Moukalled, L. Mangani, and M. Darwish. Gradient Computation. In F. Moukalled, L. Mangani, and M. Darwish, editors, *The Finite Volume Method in Computational Fluid*

Dynamics: An Advanced Introduction with OpenFOAM® and Matlab, Fluid Mechanics and Its Applications, pages 273–302. Springer International Publishing, Cham, 2016.

[61] P. Chévrier and H. Galley. A Van Leer finite volume scheme for the Euler equations on unstructured meshes. *M2AN - Modélisation mathématique et analyse numérique*, 27(2):183–201, 1993.

[62] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method.* Pearson Education, 2007.

[63] D. B. Thompson. Numerical methods 101—convergence of numerical models. 1992.

[64] T. Behrens. Openfoam's basic solvers for linear systems of equations. *Chalmers, Department of Applied Mechanics*, 18(02), 2009.

[65] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1):281–309, March 2001.

[66] Bram van Leer. Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme. *Journal of Computational Physics*, 14(4):361–370, March 1974.

[67] M. Michálek, J. Sedláček, M. Parchoviansky, M. Michálková, and D. Galusek. Mechanical properties and electrical conductivity of alumina/MWCNT and alumina/zirconia/MWCNT composites. *Ceramics International*, 40(1, Part B):1289–1295, January 2014.

[68] J. M. Stevels. The Electrical Properties of Glass. In O. Madelung, A. B. Lidiard, J. M. Stevels, and E. Darmois, editors, *Electrical Conductivity II / Elektrische Leitungsphänomene II*, Handbuch der Physik / Encyclopedia of Physics, pages 350–391. Springer, Berlin, Heidelberg, 1957.

[69] D. L. Johnson and P. N. Sen. Dependence of the conductivity of a porous medium on electrolyte conductivity. *Physical Review B*, 37(7):3502–3510, March 1988. Publisher: American Physical Society.

[70] M. Matyka, A. Khalili, and Z. Koza. Tortuosity-porosity relation in porous media flow. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 78:026306, September 2008.

[71] G. J. Janz and N. P. Bansal. Molten Salts Data: Diffusion Coefficients in Single and Multi-Component Salt Systems. *Journal of Physical and Chemical Reference Data*, 11(3):505–693, July 1982.

[72] W.M. Haynes (ed.). *CRC Handbook of Chemistry and Physics.* CRC Press LLC, Boca Raton, Florida, 94th edition, 2013-2014.

[73] G. J. Janz, R. P. T. Tomkins, C. B. Allen, J. R. Downey, Jr., G. L. Garner, U. Krebs, and S. K. Singer. Molten salts: Volume 4, part 2, chlorides and mixtures—electrical conductance, density, viscosity, and surface tension data. *Journal of Physical and Chemical Reference Data*, 4:871–1178, October 1975.

11 Appendix

11.1 Code: unedited scalarTransportFoam solver

11.1.1 scalarTransportFoam.C

```
    /*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    scalarTransportFoam

Group
    grpBasicSolvers

Description
    Passive scalar transport equation solver.

    \heading Solver details
    The equation is given by:

    \f[
        \ddt{T} + \div \left(\vec{U} T\right) - \div \left(D_T \grad T \right)
        = S_{T}
    \f]

    Where:
    \vartable
        T       | Passive scalar
        D_T     | Diffusion coefficient
        S_T     | Source
    \endvartable

    \heading Required fields
    \plaintable
        T       | Passive scalar
        U       | Velocity [m/s]
    \endplaintable

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Passive scalar transport equation solver."
    );

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"

    simpleControl simple(mesh);

    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nCalculating scalar transport\n" << endl;

    #include "CourantNo.H"

    while (simple.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
            fvScalarMatrix TEqn
            (
                fvm::ddt(T)
              + fvm::div(phi, T)
              - fvm::laplacian(DT, T)
             ==
                fvOptions(T)
            );

            TEqn.relax();
            fvOptions.constrain(TEqn);
            TEqn.solve();
            fvOptions.correct(T);
        }

        runTime.write();
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************************************* //
```

11.1.2 createFields.H

```

Info<< "Reading field T\n" << endl;

volScalarField T
(
    IOobject
    (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field U\n" << endl;

volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading transportProperties\n" << endl;

IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);


Info<< "Reading diffusivity DT\n" << endl;

dimensionedScalar DT("DT", dimViscosity, transportProperties);

#include "createPhi.H"
```

```

#include "createFvOptions.H"
```

11.1.3 Make/files

```
scalarTransportFoam.C

EXE = $(FOAM_APPBIN)/scalarTransportFoam
```

11.1.4 Make/options

```

EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling
```

11.2 Code: slayFoam solver

11.2.1 slayFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    scalarTransportFoam

Group
```

```
    grpBasicSolvers

Description
    Passive scalar transport equation solver.

    \heading Solver details
    The equation is given by:

    \f[
        \ddt{T} + \div \left(\vec{U} T\right) - \div \left(D_T \grad T \right)
        = S_{T}
    \f]

    Where:
    \vartable
        T       | Passive scalar
        D_T     | Diffusion coefficient
        S_T     | Source
    \endvartable

    \heading Required fields
    \plaintable
        T       | Passive scalar
        U       | Velocity [m/s]
    \endplaintable

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Passive scalar transport equation solver."
    );

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"

    simpleControl simple(mesh);

    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info << "\nCalculating scalar transport\n" << endl;

    while (simple.loop())
    {
        Info << "Time = " << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
            surfaceScalarField gradFi=fvc::snGrad(Fi)*mesh.magSf();
            fvScalarMatrix C1Eqn
            (
                    p*fvm::ddt(C1)
                  - fvm::laplacian(Gamma*DC1, C1)
                  - z1*F*fvm::div(Gamma*mu1*gradFi,C1)
                 ==
                    fvOptions(C1)
                );

                C1Eqn.relax();
                fvOptions.constrain(C1Eqn);
                C1Eqn.solve();
                fvOptions.correct(C1);


                fvScalarMatrix C2Eqn
                (
                    p*fvm::ddt(C2)
                  - fvm::laplacian(Gamma*DC2, C2)
                  - z2*F*fvm::div(Gamma*mu2*gradFi,C2)
                 ==
                    fvOptions(C2)
                );

                C2Eqn.relax();
                fvOptions.constrain(C2Eqn);
                C2Eqn.solve();
                fvOptions.correct(C2);

                kappa=kap+F*F*(z1*z1*Gamma*mu1*C1+z2*z2*Gamma*mu2*C2);

                fvScalarMatrix FiEqn
                (
                    fvm::laplacian(fvc::interpolate(kappa),Fi)
                );

                FiEqn.solve();
                vgradFi=fvc::grad(Fi);
            }

            runTime.write();
        }

        Info<< "End\n" << endl;

        return 0;
}
```

11.2.2 createFields.H

```

Info<< "Reading field C1\n" << endl;

volScalarField C1
(
    IOobject
    (
        "C1",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field C2\n" << endl;
volScalarField C2
(
    IOobject
    (
        "C2",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field Fi\n" << endl;

volScalarField Fi
(
    IOobject
    (
        "Fi",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

volVectorField vgradFi
(
    IOobject
    (
        "vgradFi",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    fvc::grad(Fi)
);

Info<< "Reading properties\n" << endl;

IOdictionary properties
(
    IOobject
    (
        "properties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

dimensionedScalar DC1("DC1", dimensionSet(0, 2, -1, 0, 0, 0, 0), properties);

dimensionedScalar DC2("DC2", dimensionSet(0, 2, -1, 0, 0, 0, 0), properties);

dimensionedScalar z1(”z1”, dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

dimensionedScalar z2(”z2”, dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

dimensionedScalar F(”F”, dimensionSet(0, 0, 1, 0, −1, 1, 0), properties);

dimensionedScalar T(”T”, dimensionSet(0, 0, 0, 1, 0, 0, 0), properties);

dimensionedScalar R(”R”, dimensionSet(1, 2, −2, −1, −1, 0, 0), properties);

dimensionedScalar mu1(”mu1”, dimensionSet(−1, 0, 1, 0, 1, 0, 0), properties);

dimensionedScalar mu2(”mu2”, dimensionSet(−1, 0, 1, 0, 1, 0, 0), properties);

dimensionedScalar kap(”kap”, dimensionSet(−1, −3, 3, 0, 0, 2, 0), properties);

dimensionedScalar Gamma(”Gamma”, dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

dimensionedScalar p(”p”, dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

volScalarField kappa
(
    IOobject
    (
        ”kappa”,
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    kap+F*F*(z1*z1*Gamma*mu1*C1+z2*z2*Gamma*mu2*C2)
);

#include ”createFvOptions.H”
```

11.2.3 Make/files

```
slayFoam.C

EXE = $(FOAM_USER_APPBIN)/slayFoam
```

11.2.4 Make/options

```
EXE_INC = \
    −I$(LIB_SRC)/finiteVolume/lnInclude \
    −I$(LIB_SRC)/meshTools/lnInclude \
    −I$(LIB_SRC)/sampling/lnInclude

EXE_LIBS = \
    −lfiniteVolume \
    −lfvOptions \
    −lmeshTools \
    −lsampling
```

11.3 Code: saltyFoam solver

11.3.1 saltyFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    scalarTransportFoam

Group
    grpBasicSolvers

Description
    Passive scalar transport equation solver.

    \heading Solver details
    The equation is given by:

    \f[
        \ddt{T} + \div \left(\vec{U} T\right) - \div \left(D_T \grad T \right)
        = S_{T}
    \f]

    Where:
    \vartable
        T       | Passive scalar
        D_T     | Diffusion coefficient
        S_T     | Source
    \endvartable

    \heading Required fields
    \plaintable
        T       | Passive scalar
        U       | Velocity [m/s]
    \endplaintable

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
```

```
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Passive scalar transport equation solver."
    );

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"

    simpleControl simple(mesh);

    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info << "\nCalculating scalar transport\n" << endl;

    //#include "CourantNo.H"

    while (simple.loop())
    {
        Info << "Time = " << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
            Gamma=p/(tau*tau);
            surfaceScalarField gradFi=fvc::snGrad(Fi)*mesh.magSf();
            fvScalarMatrix C1Eqn
            (
                p*fvm::ddt(C1)
              - fvm::laplacian(Gamma*DC1, C1)
              - z1*(F/R)*(1/T)*fvm::div(fvc::interpolate(Gamma)*DC1*gradFi,C1)
             ==
                fvOptions(C1)
            );

            C1Eqn.relax();
            fvOptions.constrain(C1Eqn);
            C1Eqn.solve();
            fvOptions.correct(C1);

            fvScalarMatrix C2Eqn
            (
                p*fvm::ddt(C2)
              - fvm::laplacian(Gamma*DC2, C2)
              - z2*(F/R)*(1/T)*fvm::div(fvc::interpolate(Gamma)*DC2*gradFi,C2)
             ==
                fvOptions(C2)
            );

            C2Eqn.relax();
            fvOptions.constrain(C2Eqn);
            C2Eqn.solve();
            fvOptions.correct(C2);

            C3 = (-z1*C1-z2*C2)/z3;

            kappa=F*F*(z1*z1*mu1*Gamma*C1+z2*z2*mu2*Gamma*C2+z3*z3*mu3*Gamma*C3);

            k=z1*C1+z2*C2+z3*C3;

            fvScalarMatrix FiEqn
            (
                fvm::laplacian(fvc::interpolate(kappa),Fi)
                + F*fvc::laplacian(z1*Gamma*DC1,C1)
                + F*fvc::laplacian(z2*Gamma*DC2,C2)
                + F*fvc::laplacian(z3*Gamma*DC3,C3)
            );

            FiEqn.solve();
            vgradFi=fvc::grad(Fi);

            idif=-F*(z1*DC1*Gamma*fvc::grad(C1)+z2*DC2*Gamma*fvc::grad(C2)+z3*DC3*
    Gamma*fvc::grad(C3));

            itot = idif-F*F*fvc::grad(Fi)*(z1*z1*mu1*Gamma*C1+z2*z2*mu2*Gamma*C2+
    z3*z3*mu3*Gamma*C3);

        }

        runTime.write();
    }

    Info<< "End\n" << endl;

    return 0;
}
```

11.3.2 createFields.H

```

Info<< "Reading field C1\n" << endl;

volScalarField C1
(
    IOobject
    (
        "C1",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field C2\n" << endl;

volScalarField C2
(
    IOobject
    (
        "C2",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field Fi\n" << endl;

volScalarField Fi
(
    IOobject
    (
        "Fi",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

volVectorField vgradFi
(
    IOobject
    (
        "vgradFi",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    fvc::grad(Fi)
);

Info<< "Reading properties\n" << endl;

IOdictionary properties
(
    IOobject
    (
        "properties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

dimensionedScalar DC1("DC1", dimensionSet(0, 2, -1, 0, 0, 0, 0), properties);

dimensionedScalar DC2("DC2", dimensionSet(0, 2, -1, 0, 0, 0, 0), properties);

dimensionedScalar DC3("DC3", dimensionSet(0, 2, -1, 0, 0, 0, 0), properties);

dimensionedScalar F("F", dimensionSet(0, 0, 1, 0, -1, 1, 0), properties);

dimensionedScalar T("T", dimensionSet(0, 0, 0, 1, 0, 0, 0), properties);

dimensionedScalar R("R", dimensionSet(1, 2, -2, -1, -1, 0, 0), properties);

dimensionedScalar mu1("mu1", dimensionSet(-1, 0, 1, 0, 1, 0, 0), properties);

dimensionedScalar mu2("mu2", dimensionSet(-1, 0, 1, 0, 1, 0, 0), properties);

dimensionedScalar mu3("mu3", dimensionSet(-1, 0, 1, 0, 1, 0, 0), properties);

dimensionedScalar z1("z1", dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

dimensionedScalar z2("z2", dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

dimensionedScalar z3("z3", dimensionSet(0, 0, 0, 0, 0, 0, 0), properties);

Info<< "Reading field C3\n" << endl;

volScalarField C3
(
    IOobject
    (
        "C3",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    (-z1*C1-z2*C2)/z3
);

#include "createFvOptions.H"

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    mesh
);

volScalarField tau
(
    IOobject
    (
        "tau",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    mesh
);

volScalarField Gamma
(
    IOobject
    (
        "Gamma",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    p/(tau*tau)
);
```

```

volScalarField kappa
(
    IOobject
    (
        "kappa",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    F*F*(z1*z1*mu1*Gamma*C1+z2*z2*Gamma*mu2*C2+z3*z3*mu3*Gamma*C3)
);

volScalarField k
(
    IOobject
    (
        "k",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    z1*C1+z2*C2+z3*C3
);

volVectorField idif
(
    IOobject
    (
        "idif",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    -F*(z1*DC1*Gamma*fvc::grad(C1)+z2*DC2*Gamma*fvc::grad(C2)+z3*DC3*Gamma*fvc::
    grad(C3))
);

volVectorField itot
(
    IOobject
    (
        "itot",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    idif-F*F*fvc::grad(Fi)*(z1*z1*mu1*Gamma*C1+z2*z2*mu2*Gamma*C2+z3*z3*mu3*Gamma*
    C3)
);
```

11.3.3 Make/files

```

saltyFoam.C

EXE = $(FOAM_USER_APPBIN)/saltyFoam
```

11.3.4 Make/options

```

EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling
```

11.4 Case setup: Permanganate tracer simulation compatible with the slayFoam solver

This case setup can be used to create all aqueous tracer plume simulations with some adjustments to the boundary and initial conditions corresponding, as well as constant properties described in the section Aqueous simulations 6.1.

11.4.1 0 directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2212                                  |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    arch        "LSB;label=32;scalar=64";
    class       volScalarField;
    location    "0";
    object      C1;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 -3 0 0 1 0 0];

internalField   uniform 0;

boundaryField
{
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}

// ************************************************************************* //
```

Listing 11.1: C1 file for the Permanganate tracer simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2212                                  |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    arch        "LSB;label=32;scalar=64";
    class       volScalarField;
    location    "0";
    object      C2;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 -3 0 0 1 0 0];

internalField   uniform 0;

boundaryField
{
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}

```

```
// ************************************************************************* //
```

Listing 11.2: C2 file for the Permanganate tracer simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      Fi;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 2 -3 0 0 -1 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type    fixedValue;
        value    uniform 0;
    }

    outlet
    {
        type    fixedValue;
        value   uniform 143.2;
    }

    upperWall
    {
        type    zeroGradient;
    }

    lowerWall
    {
        type    zeroGradient;
    }

    frontAndBack
    {
        type    empty;
    }
}

// ************************************************************************* //
```

Listing 11.3: Fi file for the Permanganate tracer simulation.

11.4.2 constant directory

```

/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      properties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

DC1                 1.5e-09;

DC2                 0;

z1                  -1;

z2                  1;

F                   96485;

R                   8.31446;

T                   295;

mu1                 6.12e-13;

mu2                 0;

kap                 2.91e-2;

Gamma               0.2777777778;

p                   0.4;

// ************************************************************************* //
```

Listing 11.4: properties file for the Permanganate tracer simulation.

11.4.3 system directory

```

/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

```

```
scale   0.01;

vertices
(
    (0 0 0)
    (28 0 0)
    (28 15 0)
    (0 15 0)
    (0 0 0.1)
    (28 0 0.1)
    (28 15 0.1)
    (0 15 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (280 150 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    upperWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    lowerWall
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (2 6 5 1)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

// ************************************************************************* //
```
Listing 11.5: blockMeshDict file for the Permanganate tracer simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     scalarTransportFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         7200;

deltaT          10;

writeControl    runTime;

writeInterval   100;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

// ************************************************************************* //
```
Listing 11.6: controlDict file for the Permanganate tracer simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         Gauss vanLeer;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

// ************************************************************************* //
```

Listing 11.7: fvSchemes file for the Permanganate tracer simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    C1
    {
        solver          GAMG;
        smoother   DILU;
        tolerance       1e-6;
        relTol          0;
    }

    C2
    {
        solver          GAMG;
        smoother   DILU;
        tolerance       1e-6;
        relTol          0;

    }

    Fi
    {

        solver          GAMG;
        smoother         GaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }

}

SIMPLE
{
    nNonOrthogonalCorrectors  0;
}

// ************************************************************************* //
```

Listing 11.8: fvSolution file for the Permanganate tracer simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      setExprFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

expressions
(
    C1
    {
        field       C1;
        dimensions  [0 -3 0 0 1 0 0];

        constants
        {
            centre  (0.043 0.075 0);
        }

        variables
        (
            "radius = 0.0215"
        );

        condition
        #{
            (mag(pos() - $[(vector)constants.centre]) < radius)

        #};

        expression
        #{
            3
        #};

    }

);

// ************************************************************************* //
```

Listing 11.9: setExprFieldsDict file for the Permanganate tracer simulation.

11.5 Case setup: FCD100 simulation compatible with the saltyFoam solver

This case setup can be used to create all fixed current simulations with some adjustments to the boundary and initial conditions corresponding to the values described in section Boundary and initial conditions 6.2.3 as well as time controls described in the Darcy-scale molten salt simulations 6.2 section.

11.5.1 0 directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      C1;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 -3 0 0 1 0 0];

internalField   uniform 26386;

boundaryField
{
    inlet
    {
        type          fixedGradient;
        gradient      uniform 1.29e06; //i = 1000A/m2
    }

    outlet
    {
        type             codedMixed;

        refValue         uniform 0.0;
        refGradient      uniform 0.0;
        valueFraction    uniform 0.0;

        name     zeroNaFlux;

        code
        #{
            //transport properties
            const scalar Mu1 = 7.5e-13;
            const scalar D1 = 0.00000000482;

            //variable values at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField, scalar>("Fi");
            const scalarField& C1 = patch().lookupPatchField<volScalarField, scalar>("C1");

            //variable values at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField, scalar>("Fi").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = -(C1[faceID]*Mu1/D1)*(Fi[faceID] - Fi_O.ref()[faceID])*this->patch().deltaCoeffs();
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }

    upperWall
    {
        type     zeroGradient;
    }

    lowerWall
    {
        type     zeroGradient;
    }

    frontAndBack
    {
        type      empty;
    }
}

// ************************************************************************* //
```

Listing 11.10: C1 file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      C2;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 -3 0 0 1 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            codedMixed;

        refValue        uniform 0.0;
        refGradient     uniform 0.0;
        valueFraction   uniform 0.0;

        name    zeroZnFlux;

        code
        #{
            //transport properties
            const scalar Mu2 = 3.07e-13;
            const scalar D2 = 0.00000000197;

            //variable values at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField,
    scalar>("Fi");
            const scalarField& C2 = patch().lookupPatchField<volScalarField,
    scalar>("C2");

            //variable values at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField
    , scalar>("Fi").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = -(C2[faceID]*Mu2/D2)*(Fi[faceID] - Fi_O.ref()[
    faceID])*this->patch().deltaCoeffs();
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }

    outlet
    {
        type        fixedGradient;
        gradient    uniform -6.96e05; //i = 1000A/m2
    }

    upperWall
    {
        type        zeroGradient;
    }

    lowerWall
    {
        type        zeroGradient;
    }

    frontAndBack
    {
        type        empty;
    }
}

// ************************************************************************* //
```

Listing 11.11: C2 file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      Fi;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 2 -3 0 0 -1 0];

internalField   uniform 0;

boundaryField
```

```
{
    inlet
    {
        type      fixedGradient;
        gradient      uniform  -3.226e-04;
    }

    outlet
    {
        type      fixedValue;
        value     uniform  0;
    }

    upperWall
    {
        type      zeroGradient;
    }

    lowerWall
    {
        type      zeroGradient;
    }

    frontAndBack
    {
        type      empty;
    }
}

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }

    outlet
    {
        type      zeroGradient;
    }

    upperWall
    {
        type      zeroGradient;
    }

    lowerWall
    {
        type      zeroGradient;
    }

    frontAndBack
    {
        type      empty;
    }
}

// ************************************************************************* //
```

Listing 11.12: p file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      tau;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }

    outlet
    {
        type      zeroGradient;
    }

    upperWall
    {
        type      zeroGradient;
    }

    lowerWall
    {
        type      zeroGradient;
    }

    frontAndBack
    {
        type      empty;
    }
}

// ************************************************************************* //
```

Listing 11.13: tau file for the FCD100 simulation.

11.5.2 constant directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      properties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

F                  96485;

R                  8.31446;

T                  833;

DC1                8.01e-09;

DC2                7.45e-09;

DC3                6.35e-09;

mu1                1.16e-12;

mu2                1.08e-12;

mu3                9.17e-13;

z1                 1;

z2                 2;

z3                 -1;

// ************************************************************************* //
```

Listing 11.14: properties file for the FCD100 simulation.

11.5.3 system directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                                  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale   0.01;  //inlet and outlet should not be walls, patches?

vertices
(
    (0 0 0)
    (8 0 0)
    (8 4 0)
    (0 4 0)
    (0 0 0.1)
    (8 0 0.1)
    (8 4 0.1)
    (0 4 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (400 200 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    upperWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    lowerWall
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (2 6 5 1)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

// ************************************************************************* //
```

Listing 11.15: blockMeshDict file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     slaysaltFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         22000;

deltaT          5;

writeControl    runTime;

writeInterval   500;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

```

```
runTimeModifiable true;


// ************************************************************************* //
```

Listing 11.16: controlDict file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         Gauss vanLeer;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

// ************************************************************************* //
```

Listing 11.17: fvSchemes file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
```

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    C1
    {
        solver          GAMG;
        smoother   DILU;
        tolerance       1e-6;
        relTol          0;
    }

    C2
    {
        solver          GAMG;
        smoother   DILU;
        tolerance       1e-6;
        relTol          0;

    }

    Fi
    {

        solver          GAMG;
        smoother        GaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }

}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
}

// ************************************************************************* //
```

Listing 11.18: fvSolution file for the FCD100 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

```

```
defaultFieldValues
(
    volScalarFieldValue p 1
    volScalarFieldValue tau 1
    volScalarFieldValue C1 26386
    volScalarFieldValue C2 0
);

regions
(
    boxToCell
    {
        box (0.035 0 0) (0.045 0.04 0.001);
        fieldValues
        (
            volScalarFieldValue p 0.85
            volScalarFieldValue tau 1.12
        );
    }

    boxToCell
    {
        box (0.035 0 0) (0.08 0.04 0.001);
        fieldValues
        (
            volScalarFieldValue C1 17111.6
            volScalarFieldValue C2 9766.2
        );
    }
);

// ************************************************************************* //
```

Listing 11.19: setFieldsDict file for the FCD100 simulation.

11.6 Case setup: FPD0.018 simulation compatible with the saltyFoam solver

This case setup can be used to recreate all fixed electric potential difference simulations with some adjustments to the boundary and initial conditions corresponding to the values described in section Boundary and initial conditions 6.2.3 as well as time controls described in the Darcy-scale molten salt simulations 6.2 section.

11.6.1 0 directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      C1;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 -3 0 0 1 0 0];

internalField   uniform 26386;

boundaryField
{
    inlet
    {
        type            codedMixed;

        refValue        uniform 0.0;
        refGradient     uniform 0.0;
        valueFraction   uniform 0.0;

        name    NaFlux;

        code
        #{
            //transport properties
            const scalar z = 1;
            const scalar D1 = 8.01e-09;
            const scalar F = 96485;

            //variable values at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField,
    scalar>("Fi");

            //variable values at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField
    , scalar>("Fi").patchInternalField();
            const tmp<scalarField>& kappa_O = patch().lookupPatchField<
    volScalarField, scalar>("kappa").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = -(kappa_O.ref()[faceID]*(Fi_O.ref()[faceID] - Fi
    [faceID])*this->patch().deltaCoeffs())/(z*F*D1);
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }

    outlet
    {
        type            codedMixed;

        refValue         uniform  0.0;
        refGradient      uniform  0.0;
        valueFraction    uniform  0.0;

        name     zeroNaFlux;

        code
        #{
            //transport properties
            const scalar Mu1 = 1.16e-12;
            const scalar D1 = 8.01e-09;

            //varible values at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField,
    scalar>("Fi");
            const scalarField& C1 = patch().lookupPatchField<volScalarField,
    scalar>("C1");

            //varible values at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField
    , scalar>("Fi").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = -(C1[faceID]*Mu1/D1)*(Fi[faceID] - Fi_O.ref()[
    faceID])*this->patch().deltaCoeffs();
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }

    upperWall
    {
        type     zeroGradient;
    }

    lowerWall
    {
        type     zeroGradient;
    }

    frontAndBack
    {
        type     empty;
    }
}

// ************************************************************************* //
```
Listing 11.20: C1 file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      C2;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 -3 0 0 1 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            codedMixed;

        refValue        uniform 0.0;
        refGradient     uniform 0.0;
        valueFraction   uniform 0.0;

        name    zeroZnFlux;

        code
        #{
            //transport properties
            const scalar Mu2 = 1.08e-12;
            const scalar D2 = 7.45e-09;

            //varible values at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField, scalar>("Fi");
            const scalarField& C2 = patch().lookupPatchField<volScalarField, scalar>("C2");

            //varible values at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField, scalar>("Fi").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = -(C2[faceID]*Mu2/D2)*(Fi[faceID] - Fi_O.ref()[faceID])*this->patch().deltaCoeffs();
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }

    outlet
    {
        type            codedMixed;

        refValue        uniform 0.0;
        refGradient     uniform 0.0;
        valueFraction   uniform 0.0;

        name    ZnFlux;

        code
        #{
            //transport properties
            const scalar D2 = 7.45e-09;
            const scalar z = 2;
            const scalar F = 96485;

            //varible values at the patch
            const scalarField& Fi = patch().lookupPatchField<volScalarField,
    scalar>("Fi");

            //varible values at the cell center next to boundary patch
            const tmp<scalarField>& Fi_O = patch().lookupPatchField<volScalarField
    , scalar>("Fi").patchInternalField();
            const tmp<scalarField>& kappa_O = patch().lookupPatchField<
    volScalarField, scalar>("kappa").patchInternalField();


            forAll(patch().Cf(), faceID)
            {
                this->refValue() = 0.0;
                this->valueFraction() = 0.0;
                this->refGrad() = (kappa_O.ref()[faceID]*(Fi[faceID] - Fi_O.ref()[
    faceID])*this->patch().deltaCoeffs())/(z*F*D2);
            }
        #};

        codeInclude
        #{
            #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolime/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude \
        #};
    }

    upperWall
    {
        type    zeroGradient;
    }

    lowerWall
    {
        type    zeroGradient;
    }

    frontAndBack
    {
        type    empty;
    }
}

// ************************************************************************* //
```

Listing 11.21: C2 file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      Fi;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 2 -3 0 0 -1 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type    fixedValue;
        value   uniform 0.018;
    }

    outlet
    {
        type     fixedValue;
        value    uniform 0;
    }

    upperWall
    {
        type    zeroGradient;
    }

    lowerWall
    {
        type    zeroGradient;
    }

    frontAndBack
    {
        type    empty;
    }
}

```

```
// ************************************************************************* //
```

Listing 11.22: Fi file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type    zeroGradient;
    }

    outlet
    {
        type    zeroGradient;
    }

    upperWall
    {
        type    zeroGradient;
    }

    lowerWall
    {
        type    zeroGradient;
    }

    frontAndBack
    {
        type    empty;
    }
}

// ************************************************************************* //
```

Listing 11.23: p file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      tau;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type    zeroGradient;
    }

    outlet
    {
        type    zeroGradient;
    }

    upperWall
    {
        type    zeroGradient;
    }

    lowerWall
    {
        type    zeroGradient;
    }

    frontAndBack
    {
        type    empty;
    }
}

// ************************************************************************* //
```

Listing 11.24: tau file for the FPD0.018 simulation.

11.6.2 constant directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      properties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

```

```
F                   96485;

R                   8.31446;

T                   833;

DC1                 8.01e-09;

DC2                 7.45e-09;

DC3                 6.35e-09;

mu1                 1.16e-12;

mu2                 1.08e-12;

mu3                 9.17e-13;

z1                  1;

z2                  2;

z3                  -1;

// ************************************************************************* //
```

Listing 11.25: properties file for the FPD0.018 simulation.

11.6.3 system directory

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale   0.01;  //inlet and outlet should not be walls, patches?

vertices
(
    (0 0 0)
    (8 0 0)
    (8 4 0)
    (0 4 0)
    (0 0 0.1)
    (8 0 0.1)
    (8 4 0.1)
    (0 4 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (400 200 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    upperWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    lowerWall
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (2 6 5 1)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

// ************************************************************************* //
```

Listing 11.26: blockMeshDict file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     slaysaltFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         46000;

deltaT          10;

writeControl    runTime;

writeInterval   500;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

// ************************************************************************* //
```

Listing 11.27: controlDict file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
```

```
{
    default          Gauss linear;
}

divSchemes
{
    default          Gauss vanLeer;
}

laplacianSchemes
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default          linear;
}

snGradSchemes
{
    default          corrected;
}

// ************************************************************************* //
```

Listing 11.28: fvSchemes file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    C1
    {
        solver          GAMG;
        smoother  DILU;
        tolerance       1e-6;
        relTol          0;
    }

    C2
    {
        solver          GAMG;
        smoother  DILU;
        tolerance       1e-6;
        relTol          0;

    }

    Fi
    {

        solver          GAMG;
        smoother        GaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }

}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
}

// ************************************************************************* //
```

Listing 11.29: fvSolution file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      setExprFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

expressions
(
    Fi
    {
        field       Fi;
        dimensions  [1 2 -3 0 0 -1 0];

        constants
        {
            side  (0.08 0.04 0.001);
        }

        variables
        (

        );

        condition
        #{
            (mag(pos() - $[(vector)constants.side]) > 0)

        #};

        expression
        #{
            0.018 - 0.225*pos().x()
        #};

    }

);

// ************************************************************************* //
```

Listing 11.30: setExprFieldsDict file for the FPD0.018 simulation.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2212                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue p 1
    volScalarFieldValue tau 1
    volScalarFieldValue C1 26386
    volScalarFieldValue C2 0
);

regions
(
    boxToCell
    {
        box (0.035 0 0) (0.045 0.04 0.001);
        fieldValues
        (
            volScalarFieldValue p 0.85
            volScalarFieldValue tau 1.12
        );
    }

    boxToCell
    {
        box (0.035 0 0) (0.08 0.04 0.001);
        fieldValues
        (
            volScalarFieldValue C1 17111.6
            volScalarFieldValue C2 9766.2
        );
    }
);

// ************************************************************************* //
```

Listing 11.31: setFieldsDict file for the FPD0.018 simulation.