Univ.Prof. Dr. Ansgar Jüngel

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

# Modelling, analysis and numerical discretisation of Maxwell-Stefan Cross-Diffusion systems

ausgeführt am Institut für
Analysis und Scientific Computing
der Technischen Universität Wien

unter der Anleitung von
Univ.Prof. Dr.rer.nat. Ansgar Jüngel

durch

**Georg Josef Simbrunner, MSc BSc BA**
Matrikelnummer: 0900487
Messerschmidtgasse 2/2/6 1180 Wien

Wien, 14. Mai 2018

Georg Simbrunner

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit, einschließlich Tabellen und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Georg Josef Simbrunner, am 14. Mai 2018

**Abstract**

Whenever describing purely diffusive behavior in multicomponent mixtures, the Maxwell-Stefan equations pose an important framework for various engineering applications, which include polymers (cf. [19, 54]), plasmas (cf. [26, 42]), ultrafiltration, electrolysis (cf. [69]) and even diffusion processes in the human lung (cf. [15]).

In this thesis, the link between molecular diffusion and the continuum-mechanical description in the Maxwell-Stefan equation is made plausible by following the recent exposition by Boudin in [13], which illustrates the physical assumptions under which the Maxwell-Stefan equations pose a valid model for describing multi-component diffusion on a macroscopic scale. In addition, several current efforts of generalizing some of the simplifications made in the model are stated.

Furthermore, an important result for the existence of weak solutions to the Maxwell-Stefan equations (as presented in [40, 65]) is discussed in detail, by hinting links to the more general framework of the "Boundedness-By-Entropy" method developed by Jüngel in [37].

Finally, a new conforming lowest-order Finite Element (FEM) discretization in space and a semi-implicit Euler discretization in time is discussed, which employs an "entropy-variable" formulation of the Maxwell-Stefan equations using techniques from Jüngel. For this purpose, Python code employing this method has been written to solve 1D and 2D problems even on complex polygonal geometries. The performance of the code is then investigated by a benchmark, which employs the Method of Manifactured Solutions (MMS), yielding a novel model problem with a given analytic solution.

**Zusammenfassung**

Die Maxwell-Stefan Gleichungen sind ein wichtiges Modell mit zahlreichen Anwendungen in Industrie und Technik, welche rein diffusive Prozesse zwischen zwei und mehr Gasen beschreiben können. Als einige Anwendungen der Maxwell-Stefan-Gleichungen (je nach Anwendungen mit leichten Modifikationen) seien unter anderem Polymere (siehe [19, 54]), Plasmen (siehe [26, 42]), Ultrafiltration, Elektrolyse (siehe [69]) und sogar die Beschreibung diffusiver Phänomene in der menschlichen Lunge (siehe [15]) zu nennen.

In dieser Arbeit wird zunächst zum besseren Verständnisses des Modells eine Herleitung der Maxwell-Stefan Gleichungen aus den Boltzmann-Gleichungen anhand eines jüngeren Resultats von Boudin in [13] unternommen, um den physikalischen Anwendungsbereich und die Verbindung zwischen mikroskopischer und makroskopischer Beschreibung deutlich zu machen. Weiters wird ein kurzer Überblick auf aktuelle Forschungsfragen gegeben, wonach möglicherweise einige Simplifizierungen des Modells fallen gelassen werden können.

Weiters wird ein wichtiges Resultat zur Existenz schwacher Lösungen der Maxwell-Stefan Gleichung (siehe [40, 65]) ausgearbeitet, wobei die Einordnung des Beweises in den verallgemeinerten Rahmen der "Boundedness-By-Entropy"-Methode für Kreuz-Diffusionssysteme von Jüngel [37] besprochen wird.

Im praktischen Teil der Arbeit wird anhand der in dem Existenzbeweis verwendeten Methode, welche sogenannte Entropievariablen verwendet, eine neuartige numerische Diskretisierung vorgestellt, welche die Maxwell-Stefan-Gleichungen löst. Die Ortsdiskretisierung wird hierbei mit einer Finite Elemente-Diskretisierung niedrigster Ordnung vorgenommen, während für die Zeitschritte ein semi-implizites Euler-Verfahren verwendet wird. Zur Untersuchung der Methode wurde Code in Python geschrieben, welcher in 1D und 2D auch auf komplexen polygonalen Gebieten die Maxwell-Stefan Gleichungen lösen kann. Mittels der "Method of Manifactured Solutions", welche ein Problem mit einer bekannten analytischen Lösung erzeugt und im Fall der Maxwell-Stefan Gleichungen erstmals angewendet wurde, wird hierbei die Performance des Codes genauer untersucht und diskutiert.

**Acknowledgements**

First and foremost, I want to express my deep gratitude to my parents and family, for both their moral and financial support throughout all my studies, especially for enabling me to study at Imperial College London. I am deeply indebted to them for all their unconditional and loving support of my educational path.

Secondly, I'd like like to thank my employer IMS Nanofabrication for granting me a sabbatical ("Bildungskarenz") in order to complete this thesis, foremost my group leader Dr. Thomas König for his essential support regarding my continuing education. I'd also like to thank my current and former colleagues in the High-Performance Computing (HPC) group at IMS for some rubber-ducking sessions over lunch and also for allowing me to perform some computations over the weekend on our HPC development cluster at IMS.

Thirdly, I'd like to thank my colleagues at TU Wien, which have always been incredibly supportive and open, most of which have also become very dear friends over the years.

I'd also like to thank the guys at Float (`www.float.wien`), for sharing their office with me for 4 months during my sabbatical and for allowing me to use their Gitlab server infrastructure for hosting my thesis and my code (special thanks to Jürgen Fritz for manually installing all my LaTeX-dependencies on their server on a Friday night).

I am also indebted to the Numpy community for the quick and useful help when a nasty memory leak in my version of the Numpy library has been responsible for sleepless nights on my side.[1]

Last, and by all means not least, I'd like to thank Professor Jüngel for his patient and engaging guidance and support, which I'd also like to extend to Univ.Ass. DI Oliver Leingang for patiently answering my questions regarding nonlinear PDEs.

[1] Cf. `https://github.com/numpy/numpy/issues/10788`

*Das Studium der Maxwell'schen Abhandlung ist nicht leicht.*

— J. Stefan (1871) in [64] on Maxwell's work (1866) [47]

# Contents

# 1 Introduction

The Maxwell-Stefan equations are used to describe diffusive transport in multicomponent mixtures, which were originally stated by the physicists J.C. Maxwell [47] and J. Stefan [64] in the 19th century.
In the 1960s, Duncan and Toor have experimentally demonstrated in [24] the limitations of the linear Fickian diffusion model used for describing a wide range of diffusional processes in engineering, which failed to describe the phenomenon of uphill diffusion, where the flux of the individual components is not in the direction of the concentration gradient, cf. [43] or [67, ch. 5]. However, it should be noted that there exists a more general Fickian framework where these phenomena can be described as well, cf. [67, ch. 3.2] or [66], but this yields a matrix formulation with positive and negative coefficients, which is also not symmetric. Therefore the Maxwell-Stefan framework is preferred in this case, especially if the appropriate Maxwell-Stefan diffusion coefficients are given by experiments and/or auxiliary models, cf. [67, ch. 4].
Apart from modeling diffusional processes in dilute gases, Maxwell-Stefan type equations can be also employed to study multicomponent mass transfer in polymers (cf. [19, 54]), plasmas (cf. [42, 26]) and can be used to model ultrafiltration, electrolysis and porous catalysts (cf. [69]).
Another interesting application of the Maxwell-Stefan includes the modeling of diffusional processes in the lower respiratory airways as presented in [15], which can be used to find oxygen peaks of patients with chronic obstructive bronchopneumopathies, which are administered a mixture of helium and oxygen.
The model for these various processes relevant in various applications throughout science and engineering will be shortly introduced in section 1.1 and will then be made more rigorous in section 2, where a derivation of the model from statistical physics is presented.

## 1.1 Preliminaries

In order to study the Maxwell-Stefan equation on a spacial domain $\Omega \subset \mathbb{R}^d$ for $d \in \mathbb{N}^+$, one considers a gaseous mixture of $N+1$ molar concentrations $c_i$ of gases under isothermal and isobaric conditions, where $c_i : \mathbb{R} \times \Omega \to [0,1] : (t, \underline{x}) \mapsto c_i(\underline{x}, t)$ and $1 \leq i \leq N+1$ and $N \in \mathbb{N}^+$. The suitable non-linear system to describe the diffusion of multi-component mixtures, which will be derived in section 2, is given by

$$\frac{\partial c_i}{\partial t}(\underline{x}) + \operatorname{div}(J_i) = r_i(c(\underline{x}, t)) \quad \text{in } \Omega, t > 0, 1 \leq i \leq N+1. \tag{1.1a}$$

On the boundary of the domain $\Omega$ we consider homogeneous Neumann boundary conditions for all concentrations $c_i$ (i.e. no flux across the boundary $\partial\Omega$), which are given by

$$\nabla c_i \cdot \nu = 0 \quad \text{on } \partial\Omega, t > 0, 1 \leq i \leq N+1, \tag{1.1b}$$

where $\nu$ denotes the outwards pointing unit normal vector on $\partial\Omega$.
The initial concentrations are given by measurable functions $c_i^0$, such that there holds

$$c_i(\cdot, 0) = c_i^0 \quad \text{in } \Omega, 1 \leq i \leq N+1. \tag{1.1c}$$

The molar fluxes are related to the concentration gradients, i.e. by

$$\nabla c_i = -\sum_{\substack{j=1 \\ i \neq j}}^{N+1} \frac{c_j J_i - c_i J_j}{Đ_{ij}}. \tag{1.2}$$

In (1.2) $Đ_{ij}$ denote the Maxwell-Stefan diffusion coefficients[2] and $\underline{J_i} \in \mathbb{R}^d$ denotes the molar flux, which itself is defined as

$$\underline{J_i}(c_i, \underline{u_i}) = c_i(\underline{x}, t)\underline{u_i}(\underline{x}, t), \tag{1.3}$$

where $\underline{u_i}(\underline{x}, t)$ for $1 \leq i \leq N+1$ denotes the velocity of the components of the mixture.
By the definition of matrix-vector multiplication, one can reformulate (1.2) in matrix form as

$$\underbrace{\nabla \underline{c}'}_{\in \mathbb{R}^{(N+1)\times d}} = \underbrace{A(\underline{c}(\underline{x}, t))}_{\in \mathbb{R}^{(N+1)\times(N+1)}} \underbrace{J(\underline{c}(\underline{x}, t))}_{\in \mathbb{R}^{(N+1)\times d}}, \tag{1.4}$$

where the matrix $A$ is given via

$$A_{ij}(\underline{x}, t) = \begin{cases} d_{ij} c_i(\underline{x}, t) & i \neq j \\ -\sum_{\substack{j=1 \\ j\neq i}}^{N+1} d_{ij} c_j(\underline{x}, t) & i = j \end{cases}, \tag{1.5}$$

Note that in (1.5) the short-hand $d_{ij} = \frac{1}{Đ_{ij}}$ is employed, while $\underline{c}' = (\underline{c}, c_{N+1}) \in \mathbb{R}^{N+1}$, where $\underline{c} = (c_1, \ldots, c_N) \in \mathbb{R}^N$.
Due to the assumption of isothermal and isobaric conditions, the sum of the molar concentrations is constant and can be set to one, which serves as the following closure relation[3]:

$$\sum_{j=1}^{N+1} c_i(t, \underline{x}) = 1 \quad \text{on } \mathbb{R}^+ \times \Omega. \tag{1.6}$$

However, the matrix $A$ can not be inverted, as only $N$ equations of (1.2) are linearly independent.[4] Hence the system needs an additional closure relation to be well-posed, therefore it is normally assumed that there holds a transient equimolar diffusion in the mixture before reaching the stationary state (cf. [10], [14, sec. 2]), which can be expressed via

$$\sum_{k=1}^{N+1} \underline{J_k}(t, \underline{x}) = \underline{0} \quad \text{on } \mathbb{R}^+ \times \Omega. \tag{1.7}$$

Via the closure relation (1.7), a straightforward dimensional reduction can be performed in order to obtain a lower-dimensional system where the resulting matrix $A_0 \in \mathbb{R}^{N\times N}$ can in fact be inverted.[5]
By (1.7) there holds equivalently

$$\underline{J_{N+1}} = -\sum_{j=1}^{N} \underline{J_j} = -\underline{J_i} - \sum_{\substack{j=1 \\ j\neq i}}^{N} \underline{J_j} \quad 1 \leq i \leq N.$$

[2]For further information how these coefficients can be obtained in engineering, cf. [67, ch. 4].
[3]This can be further motivated, c.f. [28, ch. 2] or [10].
[4]It can be easily proven that the $N+1$ fluxes are not linearly independent by summing (1.2) over all $N+1$ components, which yields zero. Hence the $N+1$ fluxes can't be linearly independent as this holds for arbitrary concentrations, which can be interpreted as coefficients to the flux-vectors.
[5]See Lemma 3.3 for further details.

From (1.2), (1.6) and (1.7) there holds

$$\begin{aligned}
\nabla c_i &= -\underline{J}_i \sum_{\substack{k=1\\k\neq i}}^{N+1} \frac{c_i}{\mathfrak{D}_{ik}} + c_i \sum_{\substack{j=1\\j\neq i}}^{N} \frac{\underline{J}_j}{\mathfrak{D}_{ij}} + c_i \frac{\underline{J}_{N+1}}{\mathfrak{D}_{i,N+1}} \\
&= -\underline{J}_i \sum_{\substack{k=1\\k\neq i}}^{N+1} \frac{c_i}{\mathfrak{D}_{ik}} + c_i \sum_{\substack{j=1\\j\neq i}}^{N} \frac{\underline{J}_j}{\mathfrak{D}_{ij}} - \frac{c_i}{\mathfrak{D}_{i,N+1}} \left( \underline{J}_i + \sum_{\substack{j=1\\j\neq i}}^{N} \underline{J}_j \right) \\
&= -J_i \left( \frac{c_i}{\mathfrak{D}_{i,N+1}} + \sum_{\substack{j=1\\j\neq i}}^{N+1} \frac{c_j}{\mathfrak{D}_{ij}} \right) + c_i \sum_{\substack{j=1\\j\neq i}}^{N} \left( \frac{1}{\mathfrak{D}_{ij}} - \frac{1}{\mathfrak{D}_{i,N+1}} \right) \underline{J}_j \\
&= -J_i \underbrace{\left( \frac{1}{\mathfrak{D}_{i,N+1}} + \sum_{\substack{j=1\\j\neq i}}^{N+1} \left( \frac{1}{\mathfrak{D}_{ij}} - \frac{1}{\mathfrak{D}_{i,N+1}} \right) c_j \right)}_{-a^0_{ii}} + \sum_{\substack{j=1\\j\neq i}}^{N} \underbrace{c_i \left( \frac{1}{\mathfrak{D}_{ij}} - \frac{1}{\mathfrak{D}_{i,N+1}} \right)}_{-a^0_{ij}} \underline{J}_j
\end{aligned}$$

By above computation, one can define the $N$-dimensional diffusion matrix $A_0 \in \mathbb{R}^{N\times N}$ by[6]

$$(A_0)_{ij}(\underline{c}) = \begin{cases} -\left(d_{ij} - d_{i,N+1}\right) c_i & i \neq j \\ d_{i,N+1} + \sum_{\substack{j=1\\j\neq i}}^{N} \left(d_{ij} - d_{i,N+1}\right) c_j & i = j \end{cases} \quad 1 \leq i,j \leq N \tag{1.8}$$

Assuming for the moment that $A_0(\underline{c})$ can be inverted, we can reformulate the coupled system (1.1a) and (1.2) as an $N$-dimensional system via

$$\begin{aligned}
\frac{\partial}{\partial t} \begin{pmatrix} c_1(\underline{x},t) \\ \vdots \\ c_N(\underline{x},t) \end{pmatrix} - \operatorname{div} \left( A_0(\underline{c}(\underline{x},t))^{-1} \begin{pmatrix} \nabla_{\underline{x}} c_1(\underline{x},t) \\ \vdots \\ \nabla_{\underline{x}} c_N(\underline{x},t) \end{pmatrix} \right) &= \begin{pmatrix} r_1(\underline{c}(\underline{x},t)) \\ \vdots \\ r_N(\underline{c}(\underline{x},t)) \end{pmatrix} \\
\frac{\partial}{\partial t} \underline{c}(\underline{x},t) - \operatorname{div}(A_0(\underline{c}(\underline{x},t))^{-1} \nabla_{\underline{x}} \underline{c}(\underline{x},t)) &= \underline{r}(\underline{c}) \quad \text{in } \Omega, t > 0.
\end{aligned} \tag{1.9}$$

In Lemma 3.3, it will be proven that $A_0$ can in fact be inverted. However, as stated in [40], $A_0^{-1}(\underline{c})$ may not be positive definite, which we will overcome by exploiting the entropy structure of the Maxwell-Stefan equation, which will be discussed in detail in Section 3.

## 1.2 Thesis outline

The thesis is organized as follows:
In section 2, a derivation of the Maxwell-Stefan equations via the Boltzmann equations is given, which was derived by Boudin in [13]. Futhermore, an overview on current research topics in understanding diffusion from first principles is given.
In section 3, some of the results presented in section 1 will be made more rigorous and necessary conditions for the existence of a solution to the Maxwell-Stefan system (1.1) will be presented, which was first proven by Jüngel and Stelzer in [40, 65].

[6] We actually use the convention that the coefficients in the derivation of $A_0$ are the coefficients of the matrix $-A_0$ to be consistent with [40]. Note that the contribution of the $N+1$-st summand in the term defining $a^0_{ii}$ is trivial.

In section 4, the setup of a new discretization of the Maxwell-Stefan system is presented, which was used to produce code that numerically solves the system in so-called "entropy variables" using ideas from section 3. Furthermore, several issues related to writing efficient Python code for solving the Maxwell-Stefan equations are addressed.
In section 5, the numerical performance of the code solving the Maxwell-Stefan equations is discussed on several model problems, most prominently on a problem obtained by the Method of Manifactured solutions (MMS), yielding a benchmark where an analytical solution is given, against which the output of the solver can be compared. The relation between timestep size and spatial discretization to obtain low-error solutions is further discussed.
Finally, in section 6, conclusions from the numerical experiments and a further outlook on open research topics is presented.

# 2 Derivation of the Maxwell-Stefan system via a diffusion limit

In this section, a recent result from Boudin as given in [13] is presented, which recovers the Maxwell-Stefan system (1.1) from a scaling argument derived from the weak formulation of the Boltzmann-equation.
This approach was chosen as the derivation is more rigorous than in many other approaches, e.g. in [10, 67, 69], which are more focused on applications in engineering rather than mathematical rigor. Furthermore, this presentation yields some results which are important to the later exposition in the existence proof, as e.g. the symmetry of the Maxwell-Stefan diffusion coefficients $Đ_{ij}$, which can be derived quite naturally in that framework. Hence it is suitable to gain a better understanding of the modeling behind the Maxwell-Stefan equations and also presenting the various assumptions necessary to derive the equations.
The presentation of the Boltzmann equations and the collision operators as stated in sections 2.1 and 2.3.1 is based on the expositions found in [18, 36, 53, 58].

## 2.1 A few notes on the Boltzmann equation

We consider a mixture of $N+1$ components of species $\mathcal{A}_i$, which are distributed according to the probability density function $f_i(t, \underline{x}, \underline{v})$ in three-dimensional space, i.e. $\underline{x}, \underline{v} \in \mathbb{R}^3$. Hereby $\underline{x}$ denotes the position in three-dimensional space, $\underline{v}_i$ the velocity of a given particle and $t$ the moment in time currently considered.
For any time $t$, the differential $f_i(t, \underline{x}, \underline{v}) d\underline{x} d\underline{v}$ is proportional to the number of molecules for the species $\mathcal{A}_i$ in an element of the 6D-phase-space centered at the point $(\underline{x}, \underline{v}) \in \mathbb{R}^6$. Thus we can express the molar concentrations of the species $\mathcal{A}_i$ by

$$c_i(t, \underline{x}) = \int_{\mathbb{R}^3} f_i(t, \underline{x}, \underline{v}) d\underline{v} \quad t \geq 0, \underline{x} \in \Omega. \tag{2.1}$$

For this derivation, we assume that the mixture is non-reactive (thus $r_i = 0$ in (1.1a) for $1 \leq i \leq N+1$), hence only mechanical collisions are considered.
For the derivation of the Boltzmann equation we start with the collision-free version of the Boltzmann equation, which is given for the distribution function $f_i(\underline{x}, \underline{v}_i, t)$ and an external force field $\underline{\mathcal{F}}(\underline{x}, t)$ (e.g. a gravitational field) by

$$\frac{df_i}{dt} = \frac{\partial f_i}{\partial t} + \underline{v}_i \cdot \nabla_{\underline{x}} f_i + \frac{\underline{\mathcal{F}}}{m_i} \cdot \nabla_{\underline{v}_i} f_i = 0 \tag{2.2}$$

for arbitrary $\underline{x} \in \Omega$, $\underline{v}_i \in \mathbb{R}^3$ and $t \in \mathbb{R}^+$. In (2.2), $m_i$ denotes the mass of the considered particle of species $\mathcal{A}_i$ as usual.
Above equation is a direct result of a version of Liouville's theorem formulated in $\mu$-space[7] and can be found in various textbooks, e.g [36, Ch. 3.1] or [53, Ch. 3.3.1].
In the case of the Maxwell-Stefan equations considered here, we will neglect external force fields such as electric or gravitational fields, therefore the following assumption is made:

> **Assumption 2.1 (Neglection of external forces).** *We neglect the last term of* (2.2) *as we neglect external forces, thus* $\underline{\mathcal{F}}(t, \underline{x}) = \underline{0}$ *for* $t \geq 0$, $\underline{x} \in \Omega$.

[7] The 6D phase space in which the Boltzmann equations are formulated is sometimes called $\mu$-space (compare e.g. [53, p. 22]), I will use this convention as a short hand as well.

## 2.2 Elastic two particle collisions

In order to further study a collision model in the Boltzmann equations, elastic two particle collisions have to be considered. For the Boltzmann equation with collision term, which is going to be introduced shortly, we will therefore make the following assumptions:

> **Assumption 2.2 (Elastic collision).** *The collisions between the particles are assumed to be elastic, i.e. both momentum and kinetic energy should be conserved during the collision.*

Let $\underline{v}_i$ be the velocity of the particle of species $\mathcal{A}_i$ before the collision, then the momentum is defined by

$$\underline{p}_i = m_i \underline{v}_i. \tag{2.3}$$

Let the primed quantities denote the respective quantities after the collision, then there should hold for a closed system of $n$ particles due to the conservation of momentum

$$\sum_{i=1}^{n} \underline{p}_i = \sum_{i=1}^{n} \underline{p}'_i, \tag{2.4a}$$

and due to the conservation of kinetic energy (elastic collisions)

$$\sum_{i=1}^{n} \frac{m_i (\underline{v}_i)^2}{2} = \sum_{i=1}^{n} \frac{m_i (\underline{v}'_i)^2}{2}. \tag{2.4b}$$

Let us now consider the collision of two particles of different species and study the momentum exchange involved. In that case, (2.4a) and (2.4b) simplify to

$$m_i(\underline{v}_i - \underline{v}'_i) + m_j(\underline{v}_j - \underline{v}'_j) = \underline{0} \tag{2.5a}$$

$$m_i(\underline{v}_i + \underline{v}'_i) \cdot (\underline{v}_i - \underline{v}'_i) + m_j(\underline{v}_j + \underline{v}'_j) \cdot (\underline{v}_j - \underline{v}'_j) = 0, \tag{2.5b}$$

by using the straightforward to prove identity for arbitrary $\underline{x}, \underline{y} \in \mathbb{R}^d$ and $d \geq 1$

$$|\underline{x}|^2 - \left|\underline{y}\right|^2 = (\underline{x} + \underline{y}) \cdot (\underline{x} - \underline{y}). \tag{2.6}$$

From (2.5a) there holds $m_i(\underline{v}_i - \underline{v}'_i) := -m_j(\underline{v}_j - \underline{v}'_j)$, thus substituting into (2.5b) yields

$$m_j(\underline{v}_j - \underline{v}'_j) \cdot \left[-(\underline{v}_i + \underline{v}'_i) + (\underline{v}_j + \underline{v}'_j)\right] = 0.$$

As $m_j \neq 0$, there are two possible solutions. Either there holds $\underline{v}_j = \underline{v}'_j$, which automatically implies $\underline{v}_i = \underline{v}'_i$ by (2.5a). This corresponds to the pre- and post-collisional velocities not changing during the collision. The other, more interesting solution is for $\underline{v}_i \neq \underline{v}'_i$ (and hence $\underline{v}_j \neq \underline{v}'_j$), from which we can infer that the arithmetic mean of the pre- and post-collisional velocities for both species is conserved, i.e.

$$\underline{v}_i + \underline{v}'_i = \underline{v}_j + \underline{v}'_j. \tag{2.7}$$

By above equation, we can express $\underline{v}_i$ and substitute it into (2.5a) and solve for the pre-collisional velocity for particle of species $\mathcal{A}_j$, i.e. $\underline{v}_j$. The same procedure can be applied to compute $\underline{v}_i$.

As a result of this computation, the pre-collisional velocities w.r.t. the post-collisional velocities are given by

$$\underline{v}_i = \frac{(m_i - m_j)\underline{v}_i' + 2m_j\underline{v}_j'}{m_i + m_j} = \frac{1}{m_i + m_j}\left[m_i\underline{v}_i' + m_j\underline{v}_j' + m_j(\underline{v}_j' - \underline{v}_i')\right] \tag{2.8a}$$

$$\underline{v}_j = \frac{(m_j - m_i)\underline{v}_j' + 2m_i\underline{v}_i'}{m_i + m_j} = \frac{1}{m_i + m_j}\left[m_i\underline{v}_i' + m_j\underline{v}_j' - m_i(\underline{v}_j' - \underline{v}_i')\right]. \tag{2.8b}$$

We will choose a more convenient representation of (2.8) by introducing an arbitrary vector $\underline{\sigma}$ on the unit sphere $\mathbb{S}^2$, which can e.g. be parametrized in spherical coordinates by

$$\underline{\sigma} : [0, 2\pi) \times [0, \pi] \to \mathbb{R}^3 : \underline{\sigma}(\varphi, \theta) \mapsto \begin{pmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ \cos(\theta) \end{pmatrix}. \tag{2.9}$$

Thus we arrive at the following formulation[8] by reformulating (2.8) by choosing $\underline{\sigma} \in \mathbb{S}^2$

$$\underline{\sigma} = \frac{\underline{v}_j - \underline{v}_i}{\left|\underline{v}_j - \underline{v}_i\right|}, \tag{2.10}$$

thus yielding

$$\underline{v}_i = \frac{1}{m_i + m_j}\left(m_i\underline{v}_i' + m_j\underline{v}_j' + m_j\left|\underline{v}_j' - \underline{v}_i'\right|\underline{\sigma}\right) \tag{2.11a}$$

$$\underline{v}_j = \frac{1}{m_i + m_j}\left(m_i\underline{v}_i' + m_j\underline{v}_j' - m_i\left|\underline{v}_j' - \underline{v}_i'\right|\underline{\sigma}\right). \tag{2.11b}$$

## 2.3 Mono-species collision operators

### 2.3.1 Heuristic derivation of the collision operators

In order to understand the physics behind the collision operator better, a few heuristic assumptions from physical modeling need to be considered. In order to derive the collision term, we start with stating several assumptions made on microscopic level for the particles, which are taken from [53, p. 47]:

**Assumption 2.3 (Neglection of multi-particle collisions).** *Only collisions between two molecules at a time are considered. This is justified as collisions of more than two particles are by far less likely (although such events have a finite positive probability) as 2-collisions. Note that this assumption is only justified for dilute gas mixtures.*

**Assumption 2.4 (Molecular chaos).** *The velocities of the particles in the mixture are statistically independent, which is also called "molecular chaos"-hypothesis (cf. [53, p. 47]).*

[8] Note that the conservation of momentum (2.5a) and kinetic energy (2.5b) yields 4 equations for 6 unknowns. $\underline{\sigma}$ parametrizes the 2 degrees of freedom that are left.

Due to assumption 2.1, the term containing the force $\underline{\mathcal{F}}$ in 2.2 is neglected, thus yielding

$$\frac{\partial f_i}{\partial t} + \underline{v}_i \cdot \nabla_{\underline{x}} f_i = 0, \tag{2.12}$$

which is also called free transport equation in [18] for each of the species probability densities $f_i$ for $1 \leq i \leq N+1$.
As e.g. stated in [36], the result of a collision can be interpreted as individual particles jumping to another trajectory in $\mu$-space due to the collision. In the collisionless setting as in (2.2), particles remain on their trajectory in the 6D $\mu$-space.
If the result of a collision increases the number of particles with have velocity $\underline{v}_i$, then (2.12) modifies to

$$\frac{\partial f_i}{\partial t} + \underline{v}_i \cdot \nabla_{\underline{x}} f_i = G_i(t, \underline{x}, \underline{v}_i),$$

where we have introduced the gain term $G(t, \underline{x}, \underline{v}_i) \geq 0$. This term can be interpreted as a probability density which states the probability of a particle with velocity $\underline{v}_i$ being gained at the spacial location $\underline{x}$ at time $t$.[9]
Similarly, one can define a loss-term $L_i(t, \underline{x}, \underline{v}_i) \geq 0$, which defines the probability density of a particle with velocity $\underline{v}_i$ being lost at $(\underline{x}, t)$, hence resulting in

$$\frac{\partial f_i}{\partial t} + \underline{v}_i \cdot \nabla_{\underline{x}} f_i = -L_i(t, \underline{x}, \underline{v}_i).$$

Taking both of above considerations into account, one arrives at the balance equation

$$\frac{\partial f_i}{\partial t}(t, \underline{x}, \underline{v}_i) + \underline{v}_i \cdot \nabla_{\underline{x}} f_i(t, \underline{x}, \underline{v}_i) = G_i(t, \underline{x}, \underline{v}_i) - L_i(t, \underline{x}, \underline{v}_i). \tag{2.13}$$

By assumption 2.4, we can describe the probabilities of two particles of species $\mathcal{A}_i$ and $\mathcal{A}_j$ with velocities $\underline{v}_i$ and $\underline{v}_j$ respectively at the same location $\underline{x}$ at time $t$ as the product of the probability density functions, i.e. by

$$f_i(t, \underline{x}, \underline{v}_i) f_j(t, \underline{x}, \underline{v}_j).$$

Let us note in passing that $f_i$ and $f_j$ have to be understood as the probability density function associated with the species of the $i$-th and $j$-th particle being present. If both particles belong to the same species, say $\mathcal{A}_i$, then the distribution functions are equal (although they have independent arguments).
By assumption 2.3 it is sufficient to only consider the collision of two individual particles at a time.
If two particles would always have the same probability to collide independently from their pre-collisional velocities and scattering angle, then the product of the probability densities would be equal to the probability of obtaining a particle of post-collisional velocity $\underline{v}' \in \mathbb{R}^3$, i.e. all particle configurations satisfying (2.5) would then have the same probability to interact. However, this is not a sufficient description of the general case. Therefore an interaction strength is introduced, which models the probability of the collisions of two particles with given velocities and angle to each other. This is expressed by a probability density function $B(\underline{v}_i, \underline{v}_j, \sigma)$ called a *collision-kernel*[10], which is a function of the scattering angle $\underline{\sigma} \in \mathbb{S}^2$ and the velocities $\underline{v}_i, \underline{v}_j \in \mathbb{R}^3$. The

[9] Note that the total number of particles of species $\mathcal{A}_i$ does not change under the conditions we are considering (non-reactive mixtures), but the number of particles moving with a velocity in a certain velocity-band does.

[10] The collision kernel is related to the interpretation as a physical *cross-section*. Therefore we will switch between those two interpretations of $B$ as a cross-section (physical interpretation) and a kernel (mathematical interpretation).

specific form of $B$ depends on the underlying model to represent the particles and the forces acting during the collisions.
The term for gaining particles of post-collisional velocities $\underline{v}_i'$ from arbitrary pre-collisional velocities is given by the integral over all scattering angles $\underline{\sigma} \in \mathbb{S}^2$ and all possible post-collisional velocities $\underline{v}_j' \in \mathbb{R}^3$, thus by keeping in mind $\underline{v}_k := \underline{v}_k(\underline{v}_i', \underline{v}_j, \sigma)$ for $k \in \{i, j\}$, which relation can be derived via (2.11), to be given by

$$G_i(t, \underline{x}, \underline{v}_i') = \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i, \underline{v}_j, \underline{\sigma}) f_i(t, \underline{x}, \underline{v}_i(\underline{v}_i', \underline{v}_j')) f_j(t, \underline{x}, \underline{v}_j(\underline{v}_i', \underline{v}_j')) d\underline{\sigma} d\underline{v}_j'. \tag{2.14a}$$

Similarly, one can define a loss-term, which accounts for losing particles with pre-collisional velocity $\underline{v}_i$ (as a particle having post-collisional velocity $\underline{v}_i'$ can't have pre-collisional velocity $\underline{v}_i$, as the case $\underline{v}_i = \underline{v}_i'$ is neglected) by

$$L_i(t, \underline{x}, \underline{v}_i) = \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) f_i(t, \underline{x}, \underline{v}_i'(\underline{v}_i, \underline{v}_j')) f_j(t, \underline{x}, \underline{v}_j') d\underline{\sigma} d\underline{v}_j'. \tag{2.14b}$$

As stated in section 2.2, there is a possible solution $\underline{v}_i = \underline{v}_i'$ and $\underline{v}_j = \underline{v}_j'$ which satisfies (2.5), which don't correspond to the gain- and loss-terms. However, as this solution is a set of measure zero in the set of all solutions satisfying (2.5), this case can be neglected, as stated in [18].
The cross-sections $B_i$ are assumed to satisfy the following micro-reversibility assumption:

**Assumption 2.5 (Microreversibility for cross-section).** *For any cross-section $B$ there should hold the micro-reversibility assumptions*

$$B(\underline{v}_i, \underline{v}_j, \underline{\sigma}) = B(\underline{v}_j, \underline{v}_i, \underline{\sigma}) \tag{2.15a}$$
$$B(\underline{v}_i, \underline{v}_j, \underline{\sigma}) = B(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \tag{2.15b}$$

By assumption 2.5 we can put (2.14a) and (2.14b) in the RHS of (2.13) together and thus obtain for $1 \leq i \leq N+1$ that there holds

$$Q_i^m(f, f)(\underline{v}_i') = G - L = \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \sigma) \left( f(\underline{v}_i) f(\underline{v}_j) - f(\underline{v}_i') f(\underline{v}_j') \right) d\sigma d\underline{v}_j' \tag{2.16}$$

where $\underline{v}_i$, $\underline{v}_j$ are defined by (2.11).[11] By $Q_i^m$ we denote the mono-species collision operator for the species $\mathcal{A}_i$, where the subscript for $f$ has been dropped as there is only one species to be considered.

### 2.3.2 Weak formulation of the monospecies collision operator

By making use of assumption 2.5, one can derive a weak formulation of (2.16).
For that purpose, we multiply (2.16) by an arbitrary test function $\psi(\underline{v}_i') : \mathbb{R}^3 \to \mathbb{R}$ and integrate over all possible velocities. Hence there holds by using the notation $f_k := f(\underline{v}_k)$ and $f_k' := f(\underline{v}_k')$ for $k \in \{i, j\}$ for all $\psi(\underline{v}_i')$, that

$$\begin{aligned} Q_i^{w,m}[\psi] :&= \int_{\mathbb{R}^3} Q_i^m(f, f)(\underline{v}_i') \psi(\underline{v}_i') d\underline{v}_i' \\ &= \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \left( f_i f_j - f_i' f_j' \right) \psi(\underline{v}_i') d\underline{\sigma} d\underline{v}_j' d\underline{v}_i'. \end{aligned} \tag{2.17}$$

[11] Note that due to (2.11) the functions $v_i$ and $v_j$ are actually functions of the form $\underline{v}_k(\underline{v}_i', \underline{v}_j')$ for $k \in \{i, j\}$.

As performed in various mathematical publications (e.g. in [22, 68]), a somewhat "special" weak formulation for the collision term in the Boltzmann equations is chosen, which has several benefits as will be demonstrated shortly. As performed there, this computation does not define function spaces very rigorously, thus ignoring problems with integrability at infinity.[12] The idea of the weak formulation considered here is to apply several changes of variables on (2.17) to obtain equivalent weak formulations, which are then combined to a very desirable form, which make it easy to obtain several useful corollaries needed later on.
First, we perform the change of variables $\eta : \mathbb{R}^6 \to \mathbb{R}^6 : (\underline{v}_i', \underline{v}_j') \mapsto (\underline{v}_j', \underline{v}_i')$. The jacobian of this change of variables is trivial to compute and given by

$$\eta(\underline{v}_i', \underline{v}_j') = \left(\begin{array}{c|c} \mathbf{0}_3 & I_3 \\ \hline I_3 & \mathbf{0}_3 \end{array}\right) \begin{pmatrix} \underline{v}_i' \\ \underline{v}_j' \end{pmatrix}, \tag{2.18}$$

where $I_n$ denotes the $n$-dimensional identity matrix and $\mathbf{0}_n \in \mathbb{R}^{n\times n}$ the zero-matrix. As $\det(d\eta) = -1$ there holds by performing the change of variables on the RHS of (2.17), (2.15) and Fubini's theorem that

$$\begin{aligned} Q_i^{w,m}[\psi] &= \int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_i(\underline{v}_j', \underline{v}_i', \underline{\sigma}) \left(f_i f_j - f_i' f_j'\right) \psi(\underline{v}_j') \, |-1| \, d\underline{\sigma} d\underline{v}_i' d\underline{v}_j' \\ &= \int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \left(f_i f_j - f_j' f_i'\right) \psi(\underline{v}_j') d\underline{\sigma} d\underline{v}_j' d\underline{v}_i'. \end{aligned} \tag{2.19}$$

Let us now consider the change of variables $\eta : \mathbb{R}^6 \to \mathbb{R}^6 : (v_i', v_j') \mapsto (v_i, v_j)$. The relation between the primed post-collisional variables and pre-collisional variables is defined by (2.8).[13] Let $v_{\alpha,(\beta)}$ and $v'_{\alpha,(\beta)}$ for $\alpha \in \{i, j\}$ and $\beta = 1, 2, 3$ denote the $\beta$-th component of the vectors $\underline{v}_\alpha$ and $\underline{v}_\alpha'$ respectively. By using this notation, we can now compute the jacobian $d\eta \in \mathbb{R}^{6\times 6}$ by computing the differentials

$$\frac{\partial v_{i,(k)}}{\partial v'_{i,(l)}} = \frac{1}{m_i + m_j}(m_i\delta_{kl} - m_j\delta_{kl}) \tag{2.20a}$$

$$\frac{\partial v_{i,(k)}}{\partial v'_{j,(l)}} = \frac{1}{m_i + m_j}(m_j\delta_{kl} + m_j\delta_{kl}) \tag{2.20b}$$

$$\frac{\partial v_{j,(k)}}{\partial v'_{i,(l)}} = \frac{1}{m_i + m_j}(m_i\delta_{kl} + m_i\delta_{kl}) \tag{2.20c}$$

$$\frac{\partial v_{j,(k)}}{\partial v'_{j,(l)}} = \frac{1}{m_i + m_j}(m_j\delta_{kl} - m_i\delta_{kl}), \tag{2.20d}$$

where $\delta_{\alpha\beta}$ denotes the Kronecker-delta. Equivalently in matrix form, there holds

$$d\eta = \frac{1}{m_i + m_j}\left(\begin{array}{c|c} (m_i - m_j)I_3 & 2m_jI_3 \\ \hline 2m_iI_3 & (m_j - m_i)I_3 \end{array}\right) \in \mathbb{R}^{6\times 6}. \tag{2.21}$$

A straightforward computation shows that there holds $\det(d\eta) = -1$. Furthermore it is important to note that $\eta$ is an involutory map, i.e $\eta \circ \eta = \mathrm{id}$ or equivalently $\eta^{-1} = \eta$. Let $\underline{v}_i = \eta_1(\underline{v}_i', \underline{v}_j') = \eta \circ \pi_1$, $\underline{v}_j = \eta_2(\underline{v}_i', \underline{v}_j') = \eta \circ \pi_2$ as well as $\underline{v}_i' = \eta_1^{-1}(\underline{v}_i, \underline{v}_j) = \eta_1(\underline{v}_i, \underline{v}_j)$ and $\underline{v}_j' = \eta_2^{-1}(\underline{v}_i, \underline{v}_j) = \eta_2(\underline{v}_i, \underline{v}_j)$.[14]

[12] One can e.g. assume smooth functions $\psi \in C^\infty(\mathbb{R}^3, \mathbb{R})$ and $f \in \mathcal{D}(\mathbb{R}^3, \mathbb{R})$ as in [22].

[13] For the simplicity of the computation, we use (2.8) rather than the equivalent formulation (2.11) used throughout this section.

[14] Hereby $\pi_i$ for $i \in \{1, 2\}$ denotes the projection on the first or last three elements of a 6D vector.

By this notation, we can study the action of $\eta$ on the distribution functions $f(\underline{v}_k) = f(\eta_k^{-1}(\underline{v}_i', \underline{v}_j'))$ and $f(\underline{v}_k')$ for $k \in \{i, j\}$. There holds for the change of variables $(\underline{v}_i', \underline{v}_j') \to (\underline{v}_i, \underline{v}_j)$

$$\begin{aligned} f(\underline{v}_k(\underline{v}_i, \underline{v}_j)) = f(\eta_1^{-1}(\underline{v}_i, \underline{v}_j)) &\mapsto f(\eta \circ \eta^{-1} \circ \pi_k(\underline{v}_i, \underline{v}_j)) = f(\pi_k(\underline{v}_i, \underline{v}_j)) = f(\underline{v}_k) \\ f(\underline{v}_k') &\mapsto f(\eta \circ \pi_k(\underline{v}_i, \underline{v}_j)) = f(\eta_k(\underline{v}_i, \underline{v}_j)) = f(\underline{v}_k'(\underline{v}_i, \underline{v}_j)) \end{aligned}$$

By the means of the jacobian of the change of coordinates and using the considerations above, then (2.17) can be written via Theorem A.5 as

$$\begin{aligned} Q_i^{w,m}[\psi] &= \int_{\mathbb{R}^6 \times \mathbb{S}^2} B_i(\underline{v}_i, \underline{v}_j, \underline{\sigma}) \left[ f(\eta_1(\underline{v}_i, \underline{v}_j)) f(\eta_2(\underline{v}_i, \underline{v}_j)) - f(\underline{v}_i) f(\underline{v}_j) \right] \psi(\eta_1(\underline{v}_i, \underline{v}_j)) d(\underline{\sigma}, \underline{v}_i, \underline{v}_j) \\ &= \int_{\mathbb{R}^6 \times \mathbb{S}^2} B_i(\dots) \left[ f(\eta_1^{-1}(\underline{v}_i, \underline{v}_j)) f(\eta_2^{-1}(\underline{v}_i, \underline{v}_j)) - f(\underline{v}_i) f(\underline{v}_j) \right] \psi(\eta_1^{-1}(\underline{v}_i, \underline{v}_j)) d(\underline{\sigma}, \underline{v}_i, \underline{v}_j) \\ &= - \int_{\mathbb{R}^6 \times \mathbb{S}^2} B_i(\dots) \left[ f(\underline{v}_i) f(\underline{v}_j) - f(\eta_1^{-1}(\underline{v}_i, \underline{v}_j)) f(\eta_2^{-1}(\underline{v}_i, \underline{v}_j)) \right] \psi(\eta_1^{-1}(\underline{v}_i, \underline{v}_j)) d(\underline{\sigma}, \underline{v}_i, \underline{v}_j). \end{aligned}$$

Note that in the first equality the microreversibility assumption (2.15) was used and in the second equality that $\eta$ is involutory. As the areas of integration before and after the change of variables coincide, we can relabel the integration variables $\underline{v}_k \to \underline{v}_k'$ for $k \in \{i, j\}$ to obtain

$$\begin{aligned} Q_i^{w,m}[\psi] &= - \int_{\mathbb{R}^6 \times \mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \left[ f(\underline{v}_i') f(\underline{v}_j') - f(\underline{v}_i(\underline{v}_i', \underline{v}_j')) f(\underline{v}_j(\underline{v}_i', \underline{v}_j')) \right] \psi(\underline{v}_i(\underline{v}_i', \underline{v}_j')) d(\underline{\sigma}, \underline{v}_i', \underline{v}_j') \\ &= - \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \left( f_j' f_i' - f_i f_j \right) \psi(\underline{v}_i) d\underline{\sigma} d\underline{v}_j' d\underline{v}_i'. \end{aligned} \tag{2.22}$$

By performing the change of coordinates $\eta : (\underline{v}_i, \underline{v}_j) \mapsto (\underline{v}_j, \underline{v}_i)$ on (2.22) (which has the same transformation matrix as (2.18)), one obtains

$$Q_i^{w,m}[\psi] = - \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_j, \underline{v}_i, \underline{\sigma}) \left( f_i f_j - f_i' f_j' \right) \psi(\underline{v}_j) d\underline{\sigma} d\underline{v}_j d\underline{v}_i.$$

Again one makes use of Fubini's theorem, uses the microreversibility assumptions (2.15) and as the areas of integration are invariant by relabeling variables, to arrive at

$$Q_i^{w,m}[\psi] = - \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \left( f_i' f_j' - f_i f_j \right) \psi(\underline{v}_j) d\underline{\sigma} d\underline{v}_j' d\underline{v}_i'. \tag{2.23}$$

By adding the four expressions (2.17), (2.19), (2.22) and (2.23) and rearranging, while defining $\psi_k := \psi(\underline{v}_k)$ and $\psi_k' := \psi(\underline{v}_k')$ for $k \in \{i, j\}$, there holds the weak formulation

$$Q_i^{w,m}[\psi] = -\frac{1}{4} \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} \int_{\mathbb{S}^2} B_i(\underline{v}_i', \underline{v}_j', \underline{\sigma}) \left( f_j' f_i' - f_i f_j \right) \left( \psi_i + \psi_j - \psi_i' - \psi_j' \right) d\underline{\sigma} d\underline{v}_j' d\underline{v}_i'. \tag{2.24}$$

This weak form proves to be very useful, as (2.24) has to hold for an arbitrary test function $\psi(\underline{v})$. One can therefore plug in the choices

- $\psi(\underline{v}) = 1$
- $\psi(\underline{v}) = \underline{v}_{(k)}$ for $k = 1, 2, 3$, i.e. the $k$-th component of $\underline{v}$
- $\psi(\underline{v}) = \frac{|\underline{v}|^2}{2}$.

As in the case of the monospecies collision operator there holds $m_i = m_j$ in (2.8), thus one can obtain by a straightforward computation that there holds

$$\int_{\mathbb{R}^3} Q_i^m(f,f)(\underline{v}_i')1d\underline{v}_i' = 0 \tag{2.25a}$$

$$\int_{\mathbb{R}^3} Q_i^m(f,f)(\underline{v}_i')\underline{v}_{i,(k)}d\underline{v}_i' = 0 \quad k = 1,2,3 \tag{2.25b}$$

$$\int_{\mathbb{R}^3} Q_i^m(f,f)(\underline{v}_i')\frac{|\underline{v}_i|^2}{2}d\underline{v}_i' = 0 \tag{2.25c}$$

This result will be very useful, as we can show that the number of molecules, the momentum and the kinetic energy is conserved by the mono-species collision operator.

## 2.4 Bi-species collision operator

Let us now consider the collisional interaction between two different species $\mathcal{A}_i$ and $\mathcal{A}_j$ for $1 \leq i,j \leq N+1$. Furthermore let $f := f(\underline{v}_i')$ and $g := g(\underline{v}_j')$ be non-negative functions, which correspond to probabilities of particles of the respective species with the respective post-collisional velocities being present.
The bi-species collision operators are defined by

$$Q_{ij}^b(f,g)(\underline{v}_i') = \int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}(\underline{v}_i',\underline{v}_j',\underline{\sigma})\left[f(\underline{v}_i)g(\underline{v}_j) - f(\underline{v}_i')g(\underline{v}_j')\right]d\underline{\sigma}d\underline{v}_j' \tag{2.26}$$

As in section 2.3 we assume that there holds the microreversibility assumption 2.5 for the cross section $B_{ij}$ in (2.26).
We also need to define an operator describing the collisions of molecules of the species $\mathcal{A}_j$ with the species $\mathcal{A}_i$, as due to the different masses involved the pre-collisional velocities compute slightly different than in (2.11) and are given as

$$\hat{\underline{v}}_i = \frac{1}{m_i+m_j}\left(m_j\underline{v}_i' + m_i\underline{v}_j' + m_i\left|\underline{v}_j' - \underline{v}_i'\right|\underline{\sigma}\right) \tag{2.27a}$$

$$\hat{\underline{v}}_j = \frac{1}{m_i+m_j}\left(m_j\underline{v}_i' + m_i\underline{v}_j' - m_j\left|\underline{v}_j' - \underline{v}_i'\right|\underline{\sigma}\right). \tag{2.27b}$$

By the definitions in (2.27), one can define

$$Q_{ji}^b(g,f)(\underline{v}_i') = \int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ji}(\underline{v}_i',\underline{v}_j',\underline{\sigma})\left[g(\hat{\underline{v}}_i)g(\hat{\underline{v}}_j) - f(\underline{v}_i')g(\underline{v}_j')\right]d\underline{\sigma}d\underline{v}_j'. \tag{2.28}$$

### 2.4.1 Weak formulation of the bi-species collision operator

We shall now formulate a weak formulation for (2.26). Again, there are several weak formulations possible, however we will choose a convenient version by performing very similar arguments involving the change of variables as in section 2.3.2.
By multiplying (2.26) with an arbitrary[15] test function $\psi$ and integrating over all possible

[15] Arbitrary is as always restricted to being to a certain functional analytical setting involving the correct function spaces. For the time being this setting is not defined rigorously and the restrictions on the functions are such that the Lebesgue integrals are well-defined.

post-collisional velocities $\underline{v}_i'$, and using $g_\alpha = g(\underline{v}_\alpha)$ and $g_\alpha' = g(\underline{v}_\alpha')$ for $\alpha \in \{i,j\}$, there holds

$$\begin{aligned} Q_{ij}^{w,b}[\psi] :&= \int_{\mathbb{R}^3} Q_{ij}^b(f,g)(\underline{v}_i')\psi(\underline{v}_i')dv_i' \\ &= \int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}(\underline{v}_i',\underline{v}_j',\underline{\sigma})\left[f_ig_j - f_i'g_j'\right]\psi(\underline{v}_i')d\underline{\sigma}d\underline{v}_j'd\underline{v}_i'. \end{aligned} \tag{2.29a}$$

As seen in section 2.3.2 we will now perform a change of coordinates $\eta : \mathbb{R}^6 \to \mathbb{R}^6 : (\underline{v}_i', \underline{v}_j') \mapsto (\underline{v}_i, \underline{v}_j)$, which jacobian has been computed in (2.21).
Thus there holds with similar arguments as in section 2.3.2

$$Q_{ij}^{w,b}[\psi] = -\int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}(\underline{v}_i',\underline{v}_j',\underline{\sigma})\left[f_ig_j - f_i'g_j'\right]\psi(\underline{v}_i)d\underline{\sigma}d\underline{v}_j'd\underline{v}_i'. \tag{2.29b}$$

By adding (2.29a) and (2.29b), one arrives at

$$Q_{ij}^{w,b}[\psi] = \frac{1}{2}\int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}(\underline{v}_i',\underline{v}_j',\underline{\sigma})\left[f_ig_j - f_i'g_j'\right]\left[\psi_i' - \psi_i\right]d\underline{\sigma}d\underline{v}_j'd\underline{v}_i'. \tag{2.30}$$

By a similar argument using a change of variables from primed to unprimed variables as demonstrated in section 2.3, (2.30) can be further simplified. There holds by suppressing the arguments of $B_{ij}$ (as we make use of (2.15) anyway)

$$\begin{aligned} Q_{ij}^{w,b}[\psi] &:= -\frac{1}{2}\int_{\mathbb{R}^6\times\mathbb{S}^2} B_{ij}f_i'g_j'\left[\psi_i' - \psi_i\right]d(\underline{\sigma},\underline{v}_j',\underline{v}_i') + \frac{1}{2}\int_{\mathbb{R}^6\times\mathbb{S}^2} B_{ij}f_ig_j\left[\psi_i' - \psi_i\right]d(\underline{\sigma},\underline{v}_j',\underline{v}_i') \\ &= -\frac{1}{2}\int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}f_i'g_j'\left[\psi_i' - \psi_i\right]d(\underline{\sigma},\underline{v}_j',\underline{v}_i') + \frac{1}{2}\int_{\mathbb{R}^6\times\mathbb{S}^2} B_{ij}f_i'g_j'\left[\psi_i - \psi_i'\right]d(\underline{\sigma},\underline{v}_j',\underline{v}_i') \end{aligned}$$

By simplifying the above, one arrives at the representation

$$Q_{ij}^{w,b}[\psi] = \int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}(\underline{v}_i',\underline{v}_j',\underline{\sigma})f_i'g_j'\left[\psi_i - \psi_i'\right]d\underline{\sigma}d\underline{v}_j'd\underline{v}_i'. \tag{2.31}$$

For the collision operator for particles of species $\mathcal{A}_j$ colliding with particles of species $\mathcal{A}_i$, one can formulate a weak formulation as well by multiplying (2.28) by a test function $\phi : \mathbb{R}^3 \to \mathbb{R}$ and integrating as before, thus defining

$$Q_{ji}^{w,b}(\phi) := \int_{\mathbb{R}^3} Q_{ji}^b(f,g)(\underline{v}_i')\phi(\underline{v}_i')dv_i'.$$

For notational convenience, we will drop the hats of the pre-collisional velocity variables for the $\mathcal{A}_j \to \mathcal{A}_i$-interaction which were used in section 2.4.
As already demonstrated in this section one can now perform the all changes of coordinates as seen before in this section. The change of coordinates $\eta : \mathbb{R}^6 \to \mathbb{R}^6 : (\underline{v}_i', \underline{v}_j') \mapsto (\underline{\hat{v}}_i, \underline{\hat{v}}_j)$ can be computed by (2.27) to be

$$d\eta = \frac{1}{m_i + m_j}\left(\begin{array}{c|c} (m_j - m_i)I_3 & 2m_iI_3 \\ \hline 2m_jI_3 & (m_i - m_j)I_3 \end{array}\right), \tag{2.32}$$

for which holds $\det(d\eta) = -1$ as before. With the fact that this transformation conserves the measure during the transformation as well, the same analysis using several changes of variables, which was demonstrated throughout this section, can be applied to $Q_{ji}^{w,b}(\phi)$.

Hence for the sum of the collision operators, the following weak formulation can be worked out, which is given by

$$Q_{ij}^{w,b}[\psi] + Q_{ji}^{w,b}[\phi] = -\frac{1}{2}\int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} B_{ij}(\underline{v}_i', \underline{v}_j', \underline{\sigma})\left[f_i g_j - f_i' g_j'\right]\left[\psi_i + \phi_j - \psi_i' - \phi_j'\right] d\underline{\sigma} d\underline{v}_j' d\underline{v}_i'. \tag{2.33}$$

Again the choice for the weak formulations made in (2.31) and (2.33) is quite advantageous when it comes to proving fundamental physical properties. As the equations hold for arbitrary test functions, one can make the following choices:

- By choosing $\psi = 1$ in (2.31), there holds

$$Q_{ij}^{w,b}(1) = 0, \tag{2.34a}$$

  which we will later on use to prove the conservation of the total number of particles of species $\mathcal{A}_i$.

- By choosing $\psi_k(\underline{v}_i') = m_i v_{i,(k)}'$ and $\phi_k(\underline{v}_i') = m_j v_{j,(k)}'$ for $k = 1, 2, 3$ there holds from (2.33)

$$Q_{ij}^{w,b}[\psi_k] + Q_{ji}^{w,b}[\phi_k] = 0 \quad 1 \leq i, j \leq N+1, i \neq j. \tag{2.34b}$$

  by working out via (2.11) that there holds $\underline{\psi}(\underline{v}_i(\underline{v}_i', \underline{v}_j')) + \underline{\psi}(\underline{v}_j(\underline{v}_i', \underline{v}_j')) - \underline{\psi}(\underline{v}_i') - \underline{\phi}(\underline{v}_j') = 0$, where $\underline{\phi} = (\phi_1, \phi_2, \phi_3)^T$ and $\underline{\psi} = (\psi_1, \psi_2, \psi_3)^T$.

- By choosing $\psi = \frac{m_i}{2}\left|\underline{v}_i\right|^2$ and $\phi = \frac{m_j}{2}\left|\underline{v}_j\right|^2$ in (2.33), there holds

$$Q_{ij}^{w,b}[\psi] + Q_{ji}^{w,b}[\phi] = 0 \quad 1 \leq i, j \leq N+1, i \neq j. \tag{2.34c}$$

By the results of sections 2.3 and 2.4, the coupled Boltzmann equations with no external force field for a set of unknown distribution functions $f_i$ for $1 \leq i \leq N+1$ is hence given by

$$\frac{\partial f}{\partial t} + \underline{v} \cdot \nabla_x f_i = Q_i^m(f_i, f_i) + \sum_{\substack{j=1 \\ j \neq i}}^{N+1} Q_{ij}^b(f_i, f_j). \tag{2.35}$$

## 2.5 The Maxwell-Stefan diffusion limit of the Boltzmann equation

We will now follow the main argument in [13]. The Maxwell-Stefan equations are used to describe purely diffusional behavior and thus can't account for any advection phenomena or similar transport processes.
The assumptions made in [13, sec. 4] are the following:

**Assumption 2.6 (Uniform temperature).** *There exists a uniform (in space) and constant (in time) temperature.*

**Assumption 2.7 (Bulk velocity behavior).** *At any time, the bulk velocity of the mixture is small and goes to zero when performing a limit on the scaled Boltzmann equation for vanishing Mach- and Knudsen-numbers.*

**Assumption 2.8 (Maxwellian Molecules).** *Each cross section $B_{ij}$ only depends on $\underline{v}_i', \underline{v}_j'$ and $\underline{\sigma}$ through the deviation angle $\theta \in [0, \pi]$ between $\underline{v}_i' - \underline{v}_j'$ and $\sigma$. For each pair $(i, j)$ with $1 \leq i, j \leq N+1$ and $i \neq j$ there exists a function $b_{ij} : [-1, 1] \to \mathbb{R}^+$ that*

$$B_{ij}(\underline{v}_i', \underline{v}_j', \sigma) = b_{ij}\left(\frac{\underline{v}_i' - \underline{v}_j'}{|\underline{v}_i' - \underline{v}_j'|} \cdot \underline{\sigma}\right) = b_{ij}(\cos\theta) \tag{2.36}$$

**Assumption 2.9 (Properties of $b_{ij}$).** *The function $b_{ij}$ is assumed to be an even function, i.e. $b_{ij}(x) = b_{ij}(-x)$. Furthermore we assume $b_{ij} \in L^1([-1, 1])$ which can be deduced by Grad's cutoff assumption for collision kernels.*

By using these assumptions, one can prove the following straightforward lemma:

**Lemma 2.1.** *There holds $b_{ij} = b_{ji}$.*

*Proof.* Thanks to the microreversibility assumption 2.5 for the cross section $B_{ij}$ and assumptions 2.8 and 2.9, there holds

$$\begin{aligned} b_{ij}(\cos\theta) &= b_{ij}\left(\frac{\underline{v}_i' - \underline{v}_j'}{\left|\underline{v}_i' - \underline{v}_j'\right|} \cdot \underline{\sigma}\right) = B_{ij}(\underline{v}_i', \underline{v}_j', \sigma) = B_{ji}(\underline{v}_j', \underline{v}_i', \sigma) \\ &= b_{ji}\left(\frac{\underline{v}_j' - \underline{v}_i'}{\left|\underline{v}_i' - \underline{v}_j'\right|} \cdot \underline{\sigma}\right) = b_{ji}\left(\frac{\underline{v}_i' - \underline{v}_j'}{\left|\underline{v}_i' - \underline{v}_j'\right|} \cdot \underline{\sigma}\right) = b_{ji}(\cos\theta) \end{aligned}$$

□

We will now consider a scaled version of the Boltzmann equations. In [58, eq. (2.18)] a dimensionless version of the Boltzmann equation is derived and given by

$$\mathrm{St}\frac{\partial f}{\partial t} + \underline{v} \cdot \nabla_{\underline{x}} f = \frac{1}{\mathrm{Kn}} Q(f, f) \tag{2.37}$$

where the Knudsen number Kn is given by

$$\mathrm{Kn} = \frac{\lambda}{l_0}, \tag{2.38a}$$

whereas the Strouhal number Sn is defined as

$$\mathrm{Sn} = \frac{l_0}{c t_0}. \tag{2.38b}$$

In (2.38), $t_0$ is a characteristic time scale, $l_0$ is a characteristic length scale reflecting how far gas is transported in the time $t_0$, and $\lambda = c\tau$ is the mean free path of molecules ($c$ is the speed of sound and $\tau$ the mean free time of molecules).

In the analysis carried out in [58, Ch. 2.2.1/2.2.2] it is also stated that the ratio $\frac{l_0}{t_0}$ corresponds to the bulk velocity denoted by $u_0$. This property can be related to the Mach-number Ma

$$\mathrm{Ma} = \frac{u_0}{c}.$$

It is further stated that by only considering small fluctuations about a reference flow, for the bulk velocity there holds $u_0 \ll \frac{l_0}{t_0}$, corresponds to a situation where

$$\mathrm{Ma} \ll \mathrm{St}.$$

Applying the dimensionless form (2.37) to our structure of the collision term in (2.35), while considering a diffusive limit, where the mean free path $\lambda \to 0$ and thus $\mathrm{Kn} \to 0$, while considering small $\mathrm{Ma} \sim \mathrm{Sn} \to 0$, we can write the diffusive limit by assumption 2.7 as follows:

**Definition 2.1 (Diffusional limit of the Boltzmann equation).** *Let $\varepsilon > 0$, and the solutions of the scaled version of the multi-species Boltzmann equation be denoted by $f_i^\varepsilon$ for $1 \leq i \leq N+1$, then each distribution function $f_i^\varepsilon$ shall solve the scaled version of the Boltzmann equation* (2.35)*, which is given by*

$$\varepsilon \frac{\partial f}{\partial t} + \underline{v} \cdot \nabla_{\underline{x}} f_i^\varepsilon = \frac{1}{\varepsilon} Q_i^m(f_i, f_i) + \frac{1}{\varepsilon} \sum_{\substack{j=1 \\ j \neq i}}^{N+1} Q_{ij}^b(f_i, f_j) \quad \text{on } \mathbb{R}^+ \times \Omega \times \mathbb{R}^3. \tag{2.39}$$

This scaling argument is quite common and has e.g. also been derived in [36, Ch. 2.1] as so-called "diffusion scaling".
By means of the distribution function $f_i^\varepsilon$ one can define the concentration function similarly to (2.1) by

$$c_i^\varepsilon(t, x) = \int_{\mathbb{R}^3} f_i^\varepsilon(t, \underline{x}, \underline{v}) d\underline{v}. \tag{2.40}$$

### 2.5.1 Ansatz function for the diffusion limit of the Boltzmann equations

The initial conditions of (2.39) are assumed to be Maxwellian functions with small macroscopic velocity (as only diffusional processes are to be considered here). This model leads to the following assumption:

**Assumption 2.10.** *The initial condition of* (2.39) *is given by*

$$f_i^{\varepsilon,0}(\underline{x}, \underline{v}) = c_i^0(\underline{x}) \left( \frac{m_i}{2\pi k T} \right)^{\frac{3}{2}} \exp\left[ -\frac{m_i}{2kT} \left| \underline{v} - \varepsilon \underline{u}_i^0(\underline{x}) \right|^2 \right] \quad \underline{x} \in \Omega, \underline{v} \in \mathbb{R}^3, \tag{2.41}$$

*where the temperature $T > 0$ is a fixed constant (cf. assumption 2.6) and the initial concentrations $c_i^0 : \Omega \to \mathbb{R}^+$ and $\underline{u}_i^0 : \Omega \to \mathbb{R}^3$ do not depend on $\varepsilon$. Furthermore there shall hold*

$$\sum_{k=1}^{N+1} c_k^0 = 1 \quad \text{on } \Omega.$$

From the definition of $f_i^{\varepsilon,0}$ as slightly perturbed Maxwellian functions, one can compute the following relation:

**Lemma 2.2.** *For $f_i^{\varepsilon,0}$ as defined in (2.41), there holds*

$$\underline{I}(\underline{x}) := \frac{1}{\varepsilon}\int_{\mathbb{R}^3} \underline{v} f_i^{\varepsilon,0}(\underline{x},\underline{v})d\underline{v} = c_i^0(\underline{x})\underline{u}_i^0(\underline{x}) \quad x \in \Omega. \tag{2.42}$$

*Proof.* We will prove this vector valued equation for the $k$-th component $v_{(k)}$ and $k = 1,2,3$. By reordering the order of integration, one can iterate the the integral by a permutation $\{1,2,3\} \to \{k,l,m\}$ such that

$$I_{(k)}(\underline{x}) = K_1(\underline{x})\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K_3(v_{(l)},v_{(m)})\int_{-\infty}^{\infty} v_{(k)}\exp\left[-K_2\left(v_{(k)}-\varepsilon u_{i,(k)}^0\right)^2\right]dv_{(k)}dv_{(l)}dv_{(m)},$$

where for convenience we set

$$\begin{aligned} K_1(\underline{x}) &:= \frac{1}{\varepsilon}c_i^0(x)\left(\frac{m_i}{2\pi kT}\right)^{\frac{3}{2}} \\ K_2 &:= \frac{m_i}{2kT} \\ K_3(v_{(l)},v_{(m)}) &:= e^{-K_2\sum_{j\neq k}(v_{(j)}-\varepsilon u_{i,(j)}^0)^2}. \end{aligned}$$

By making use of the solution of the Gaussian integral, i.e. there holds for $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$ that

$$\int_{-\infty}^{\infty}\exp\left[-\alpha(x-\beta)^2\right]dx = \sqrt{\frac{\pi}{\alpha}}, \tag{2.43}$$

one can obtain via integration by parts that

$$K_4(\underline{x}) := \int_{-\infty}^{\infty} v_{(k)}\exp\left[-K_2\left(v_{(k)}-\varepsilon u_{i,(k)}^0\right)^2\right]dv_{(k)} = \frac{\sqrt{\pi}\varepsilon u_i^0(\underline{x})}{\sqrt{K_2}}.$$

It remains to compute

$$I_{(k)} = K_1(\underline{x})K_4(\underline{x})\int_{-\infty}^{\infty} e^{-K_2(v_{(m)}-\varepsilon u_{i,(m)}^0)^2}\left[\int_{-\infty}^{\infty} e^{-K_2(v_{(l)}-\varepsilon u_{i,(l)}^0)^2}dv_{(l)}\right]dv_{(m)}.$$

As both of these integrals are Gaussian integrals of the form (2.43), there holds

$$I_{(k)} = K_1(\underline{x})K_4(\underline{x})\sqrt{\frac{\pi}{K_2}}\sqrt{\frac{\pi}{K_2}},$$

thus by re-substituting the constants and simplifying, there holds the claim (2.42). □

**Assumption 2.11 (Maxwellian time evolution).** *It is assumed that the evolution of system (2.39) is given by a local Maxwellian state, i.e. the distribution function $f_i^\varepsilon$ is given by*

$$f_i^\varepsilon(t,\underline{x},\underline{v}) = c_i(\underline{x},t)\left(\frac{m_i}{2\pi kT}\right)^{\frac{3}{2}}\exp\left[-\frac{m_i}{2kT}\left|\underline{v}-\varepsilon\underline{u}_i(t,\underline{x})\right|^2\right] \quad t>0, \underline{x}\in\Omega, \underline{v}\in\mathbb{R}^3, \tag{2.44}$$

*where we suppose that there exist $\underline{u}_i^\varepsilon : \mathbb{R}^+\times\Omega \to \mathbb{R}^3$ and $c_i^\varepsilon : \mathbb{R}^+\times\mathbb{R}^3 \to \mathbb{R}$ for $1 \le i \le N+1$.*

**Lemma 2.3.** *For the Maxwellian* (2.44) *the moments of order 0 and 1 w.r.t. to* $\underline{v}$ *are given by*

$$I_0(t,\underline{x}) = \int_{\mathbb{R}^3} f_i^\varepsilon(t,\underline{x},\underline{v})d\underline{v} = c_i^\varepsilon(t,\underline{x}) \tag{2.45a}$$

$$\underline{I}_1^\varepsilon(t,\underline{x}) = \int_{\mathbb{R}^3} \underline{v} f_i^\varepsilon(t,\underline{x},\underline{v})d\underline{v} = \varepsilon c_i^\varepsilon(t,\underline{x})u_i^\varepsilon(t,\underline{x}) \tag{2.45b}$$

*Proof.* In order to prove (2.45a) one has to simply compute three Gaussian integrals via (2.43). To prove (2.45b) one can use the proof from Lemma 2.2 by making use of the identity (2.42) and substituting $c_i^0(x) \to c_i^\varepsilon(t,\underline{x})$ and $\underline{u}_i^0(\underline{x}) \to \underline{u}_i^\varepsilon(t,\underline{x})$. □

**Lemma 2.4 (Mass conservation).** *Under the assumption that the distribution function is given by* (2.44) *(cf. assumption 2.11), there holds that mass is conserved by* (2.39).

*Proof.* By (2.25) and (2.34), there holds that the zeroth and first moment in $\underline{v}$ of the collision operators for any $f_i^\varepsilon$ is trivial, thus (2.39) simplifies to

$$\varepsilon\frac{\partial}{\partial t}\int_{\mathbb{R}^3} f_i^\varepsilon(t,\underline{x},\underline{v})d\underline{v} + \nabla_{\underline{x}}\cdot\int_{\mathbb{R}^3} f_i^\varepsilon(t,\underline{x},\underline{v})\underline{v}d\underline{v} = 0$$

By (2.45) there holds for $1 \le i \le N+1$

$$\frac{\partial c_i^\varepsilon}{\partial t} + \nabla_{\underline{x}}\cdot(c_i^\varepsilon(\underline{x})\underline{u}_i^\varepsilon(\underline{x})) = 0 \tag{2.46}$$

□

### 2.5.2 Balance of Momentum

In order to derive the balance law for the momentum of species $\mathcal{A}_i$, one multiplies (2.39) by $\underline{v}$ and thus obtains by observing, that the contribution for the mono-species collision operator is trivial due to (2.25b) for the $k$-th component of $\underline{v}$, that there holds

$$\varepsilon\frac{\partial}{\partial t}\int_{\mathbb{R}^3} v_{(k)} f_i^\varepsilon(t,\underline{x},\underline{v})d\underline{v} + \nabla_{\underline{x}}\cdot\int_{\mathbb{R}^3} v_{(k)} f_i^\varepsilon(t,\underline{x},\underline{v})\underline{v}d\underline{v} = \frac{1}{\varepsilon}\int_{\mathbb{R}^3} v_{(k)} Q_{ij}^b(f_i^\varepsilon, f_i^\varepsilon)(\underline{v})d\underline{v}. \tag{2.47}$$

First, we consider the RHS of (2.47), by setting

$$\Theta_{(k)}^\varepsilon(t,\underline{x}) := \frac{1}{\varepsilon}\sum_{\substack{j=1\\ j\neq i}}^{N+1}\int_{\mathbb{R}^3} v_{(k)} Q_{ij}^b(f_i^\varepsilon, f_i^\varepsilon)(\underline{v})d\underline{v}.$$

By using the weak formulation (2.31) of the bi-species collision operator with the specific test functions $\psi_k(\underline{v}) = v_{(k)}$, (2.36) from assumption 2.8 and the expression for the pre-collisional velocities (2.11), there holds

$$
\begin{aligned}
&\Theta^{\varepsilon}_{(k)} \\
&= \frac{1}{\varepsilon}\sum_{\substack{j=1\\ j\neq i}}^{N+1}\int_{\mathbb{R}^3}\int_{\mathbb{R}^3}\int_{\mathbb{S}^2} b_{ij}\left(\frac{\underline{v}_i'-\underline{v}_j'}{|\underline{v}_i'-\underline{v}_j'|}\cdot\underline{\sigma}\right) f_i^{\varepsilon}(\underline{v}_i')f_j^{\varepsilon}(\underline{v}_j')(v_{i,(k)}-v_{i,(k)}')d\underline{\sigma}d\underline{v}_j'd\underline{v}_i' \\
&= \frac{1}{\varepsilon}\sum_{\substack{j=1\\ j\neq i}}^{N+1}\int_{\mathbb{R}^6\times\mathbb{S}^2} b_{ij}\left(\frac{\underline{v}_i'-\underline{v}_j'}{|\underline{v}_i'-\underline{v}_j'|}\cdot\underline{\sigma}\right) f_i^{\varepsilon}(\underline{v}_i')f_j^{\varepsilon}(\underline{v}_j')(v_{i,(k)}-v_{i,(k)}')d(\underline{v}_i',\underline{v}_j',\underline{\sigma}) \\
&= \frac{1}{\varepsilon}\sum_{\substack{j=1\\ j\neq i}}^{N+1}\frac{m_j}{m_i+m_j}\int_{\mathbb{R}^6\times\mathbb{S}^2} b_{ij}\left(\frac{\underline{v}_i'-\underline{v}_j'}{|\underline{v}_i'-\underline{v}_j'|}\cdot\underline{\sigma}\right) f_i^{\varepsilon}(\underline{v}_i')f_j^{\varepsilon}(\underline{v}_j')(v_{j,(k)}'-v_{i,(k)}'-|\underline{v}_j'-\underline{v}_i'|\sigma_{(k)})d(\underline{v}_i',\underline{v}_j',\underline{\sigma}).
\end{aligned}
$$

The term containing $\sigma_{(k)}$ has trivial contribution for all $k$, which can be shown as follows: Due to the parametrization of $\underline{\sigma}$ in polar coordinates as seen in (2.9), one can deduce that for $k=1,2$, as from the integration over $\mathbb{S}^2$ the integration over the polar angle $\varphi$ yields

$$
\int_0^{2\pi}\cos\varphi d\varphi = \int_0^{2\pi}\sin\varphi d\varphi = 0.
$$

For $k=3$, there holds by the substitution $s=\cos\theta$

$$
\int_{\mathbb{S}^2} b_{ij}\left(\frac{\underline{v}_i'-\underline{v}_j'}{|\underline{v}_i'-\underline{v}_j'|}\cdot\underline{\sigma}\right)\sigma_{(3)}d\underline{\sigma} = 2\pi\int_0^{\pi}\sin\theta\cos\theta b_{ij}(\cos\theta)d\theta = \int_{-1}^{1} s b_{ij}(s)ds = 0,
$$

as due to assumption 2.9 $b_{ij}$ is an even function. Thus $\Theta^{\varepsilon}_{(k)}$ reduces to

$$
\Theta^{\varepsilon}_{(k)} = \frac{1}{\varepsilon}\sum_{\substack{j=1\\ j\neq i}}^{N+1}\frac{m_j}{m_i+m_j}\int_{\mathbb{R}^3} f_i^{\varepsilon}(\underline{v}_i')\int_{\mathbb{R}^3} f_j^{\varepsilon}(\underline{v}_j')\left(v_{j,(k)}'-v_{i,(k)}'\right)\int_{\mathbb{S}^2} b_{ij}\left(\frac{\underline{v}_i'-\underline{v}_j'}{|\underline{v}_i'-\underline{v}_j'|}\cdot\underline{\sigma}\right)d\underline{\sigma}d\underline{v}_j'd\underline{v}_i'.
$$

By making use of (2.45) and (2.36) from assumption 2.8 there holds

$$
\underline{\Theta}^{\varepsilon} = \sum_{\substack{j=1\\ j\neq i}}^{N+1}\frac{2\pi m_j\|b_{ij}\|_{L^1([0,\pi])}}{m_i+m_j}(c_i^{\varepsilon}c_j^{\varepsilon}\underline{u}_j^{\varepsilon}-c_j^{\varepsilon}c_i^{\varepsilon}\underline{u}_i^{\varepsilon}). \tag{2.48}
$$

Let us now consider the remaining terms in (2.47). For the term containing the time derivative, there holds due to (2.45b)

$$
\varepsilon\frac{\partial}{\partial t}\int_{\mathbb{R}^3}\underline{v}f_i^{\varepsilon}(t,\underline{x},\underline{v})d\underline{v} = \varepsilon^2\frac{\partial}{\partial t}\left[c_i^{\varepsilon}(t,\underline{x})\underline{u}_i^{\varepsilon}(t,\underline{x})\right]. \tag{2.49}
$$

The divergence term in (2.47) can be further simplified by performing the substitution $\eta_{(k)} = v_k - \varepsilon u^{\varepsilon}_{i,(k)}(t,\underline{x})$

$$
\begin{aligned}
\Xi_{(l)} &= \nabla_{\underline{x}}\cdot\int_{\mathbb{R}^3} v_{(k)}f_i^{\varepsilon}(\underline{v})\underline{v}d\underline{v} \\
&= K_1\sum_{k=1}^{3}\frac{\partial}{\partial x_{(k)}}\left[\int_{\mathbb{R}^3} c_i^{\varepsilon}(t,\underline{x})(\eta_{(k)}+\varepsilon u^{\varepsilon}_{i,(k)})(\eta_{(l)}+\varepsilon u^{\varepsilon}_{i,(l)})e^{-K_2|\underline{\eta}|^2}d\underline{\eta}\right] \\
&= K_1\sum_{k=1}^{3}\frac{\partial}{\partial x_{(k)}}\left[\int_{\mathbb{R}^3} c_i^{\varepsilon}(t,\underline{x})\left(\eta_{(k)}\eta_{(l)}+\varepsilon\left(\eta_{(k)}u^{\varepsilon}_{i,(l)}+\eta_{(l)}u^{\varepsilon}_{i,(k)}\right)+\varepsilon^2 u^{\varepsilon}_{i,(k)}u^{\varepsilon}_{i,(l)}\right)e^{-K_2|\underline{\eta}|^2}d\underline{\eta}\right],
\end{aligned}
$$

by using the abbreviations

$$K_1 = \Big(\frac{m_i}{2\pi kT}\Big)^{\frac{3}{2}} \tag{2.50a}$$

$$K_2 = \frac{m_i}{2kT}. \tag{2.50b}$$

As there holds for arbitrary $\gamma \in \mathbb{R}^+$ and $j \in \mathbb{N}$

$$\int_{-\infty}^{\infty} \tau^{2j+1} e^{-\gamma\tau^2} d\tau = 0 \tag{2.51a}$$

$$\int_{-\infty}^{\infty} \tau^2 e^{-\gamma\tau^2} d\tau = \frac{\sqrt{\pi}}{2\gamma^{\frac{3}{2}}}, \tag{2.51b}$$

all terms containing a single $\eta_{(\alpha)}$ or the product $\eta_{(\alpha)}\eta_{(\beta)}$ for $1 \leq \alpha, \beta \leq 3$ and $\alpha \neq \beta$ have trivial contributions. Hence the terms of $\mathcal{O}(\varepsilon)$ have no contributions and the expression for $\Xi$ reduces via (2.51b) and (2.43) to

$$\Xi_{(l)} = \varepsilon^2 \sum_{k=1}^{3} \frac{\partial}{\partial x_{(k)}} \Big[ c_i^\varepsilon u_{i,(k)}^\varepsilon u_{i,(l)}^\varepsilon \Big] + \frac{kT}{m_i} \frac{\partial c_i^\varepsilon}{\partial x_{(l)}}. \tag{2.52}$$

By collecting the results from (2.49), (2.52) and (2.48) and plugging them in (2.47), there holds

$$\varepsilon^2 \left[ \frac{\partial}{\partial t}(c_i^\varepsilon u_i^\varepsilon) + \nabla_{\underline{x}} \cdot (c_i^\varepsilon u_i^\varepsilon \otimes u_i^\varepsilon) \right] + \frac{kT}{m_i} \nabla_{\underline{x}} c_i^\varepsilon = \sum_{\substack{j=1 \\ j \neq i}}^{N+1} \frac{2\pi m_j \|b_{ij}\|_{L^1([0,\pi])}}{m_i + m_j} (c_i^\varepsilon c_j^\varepsilon \underline{u}_j^\varepsilon - c_j^\varepsilon c_i^\varepsilon \underline{u}_i^\varepsilon). \tag{2.53}$$

## 2.6 Macroscopic equations and formal asymptotics

By using the conservation of mass (2.46) and the evolution of momentum for the scaled Boltzmann equation (2.53), one can infer that the Maxwellian distribution functions (2.44) are a solution[16] for the initial-boundary value problem of the scaled Boltzmann equations (2.39), if the unknowns $(c_i^\varepsilon, \underline{u}_i^\varepsilon)$ for $1 \leq i \leq N+1$ solve

$$\frac{\partial c_i^\varepsilon}{\partial t} + \nabla_{\underline{x}} \cdot (c_i^\varepsilon \underline{u}_i^\varepsilon) = 0 \tag{2.54a}$$

$$\varepsilon^2 \frac{m_i}{kT} \left[ \frac{\partial}{\partial t}(c_i^\varepsilon u_i^\varepsilon) + \nabla_{\underline{x}} \cdot (c_i^\varepsilon u_i^\varepsilon \otimes u_i^\varepsilon) \right] + \nabla_{\underline{x}} c_i^\varepsilon = \sum_{\substack{j=1 \\ j \neq i}}^{N+1} \frac{c_i^\varepsilon c_j^\varepsilon \underline{u}_j^\varepsilon - c_j^\varepsilon c_i^\varepsilon \underline{u}_i^\varepsilon}{\Delta_{ij}}. \tag{2.54b}$$

Hereby the constants $\Delta_{ij}$ are given via (2.53) by

$$\Delta_{ij} = \frac{(m_i + m_j)kT}{2\pi m_i m_j \|b_{ij}\|_{L^1([0,\pi])}}. \tag{2.55}$$

Let us note in passing that due to the symmetry of $b_{ij}$ (cf. Lemma 2.1) there holds $\Delta_{ij} = \Delta_{ji}$ for the coefficients in (2.55).
In order to deduce the Maxwell-Stefan equations, we first need to define

$$\underline{J}_i^\varepsilon(t, \underline{x}) := \frac{1}{\varepsilon} \int_{\mathbb{R}^3} \underline{v} f_i^\varepsilon(t, \underline{x}, \underline{v}) d\underline{v} \overset{(2.45b)}{=} c_i^\varepsilon(t, \underline{x}) \underline{u}_i^\varepsilon(t, \underline{x}) \quad t \geq 0, \underline{x} \in \Omega, \tag{2.56}$$

[16]Note that the Maxwellian distribution functions $f_i^\varepsilon$ are related to the molar concentrations $c_i^\varepsilon$ via (2.40).

as well as the formal diffusive limits of $c_i^\varepsilon$ and $\underline{J}_i^\varepsilon$ by

$$c_i(t,\underline{x}) = \lim_{\varepsilon\to 0^+} c_i^\varepsilon$$
$$\underline{J}_i^\varepsilon(t,\underline{x}) = \lim_{\varepsilon\to 0^+} \underline{J}_i^\varepsilon.$$

By (2.56) we can rewrite (2.54) by letting $\varepsilon \to 0^+$ and thus obtain a system in the unknowns $(c_i, \underline{J}_i)$ for $1 \le i \le N+1$ by

$$\frac{\partial c_i}{\partial t} + \nabla_{\underline{x}} \cdot \underline{J}_i = 0 \tag{2.58a}$$
$$\nabla_{\underline{x}} c_i = \sum_{\substack{j=1\\ j\neq i}}^{N+1} \frac{c_i \underline{J}_j - c_j \underline{J}_i}{\Delta_{ij}}. \tag{2.58b}$$

However, we have to prove that the sum over all $c_i$ is constant in the limit, which was postulated in (1.6). In order to achieve this, we have to prove the kinetic energy conservation before applying the diffusive limit. For that purpose, we pre-compute two auxiliary integrals. The first one, denoted $\Gamma$, is given via

$$\Gamma = \int_{\mathbb{R}^3} |\underline{v}|^2 f_i^\varepsilon(\underline{v}) d\underline{v},$$

which can be solved via the substitution $\eta_{(k)} = v_{(k)} - \varepsilon u_i^\varepsilon(t,\underline{x})$ and by using the notation in (2.50), as there holds

$$\begin{aligned}\Gamma &= K_1 \int_{\mathbb{R}^3} \sum_{k=1}^3 (\eta_{(k)} + \varepsilon u_{i,(k)}^\varepsilon)^2 c_i^\varepsilon e^{-K_2|\underline{\eta}|^2} d\underline{\eta} \\ &= K_1 \int_{\mathbb{R}^3} c_i^\varepsilon(t,\underline{x}) \sum_{k=1}^3 \left(\eta_{(k)}^2 + 2\varepsilon\eta_{(k)} u_{i,(k)}^\varepsilon + \varepsilon^2 [u_{i,(k)}^\varepsilon]^2\right) e^{-K_2|\underline{\eta}|^2} d\underline{\eta}.\end{aligned}$$

Collecting the terms $\mathcal{O}(1)$ of the expansion $\Gamma = \sum_k \Gamma_k \varepsilon^k$, one has to compute the integral for $\Gamma_0$ by changing integral and summation and employing (2.43) and (2.51b) for a suitable permutation $\{1,2,3\} \to \{k,l,m\}$ for each $k$ to obtain

$$\begin{aligned}\Gamma_0 &= K_1 c_i^\varepsilon(t,\underline{x}) \sum_{k=1}^3 \int_{-\infty}^{\infty} e^{-K_2\eta_{(m)}^2} \left[\int_{-\infty}^{\infty} e^{-K_2\eta_{(l)}^2} \left[\int_{-\infty}^{\infty} \eta_{(k)}^2 e^{-K_2\eta_{(k)}^2} d\eta_{(k)}\right] d\eta_{(l)}\right] d\eta_{(m)} \\ &= c_i^\varepsilon \sum_{k=1}^3 K_1 \frac{\pi}{K_2} \frac{\sqrt{\pi}}{2K_2^{\frac{3}{2}}} = 3c_i^\varepsilon \frac{kT}{m_i}\end{aligned}$$

Note that $\Gamma_1 = 0$ due to (2.51a) and $\Gamma_2 = \sum_{k=1}^3 u_{i,(k)}$ with similar computations as for the $\mathcal{O}(\varepsilon^2)$ term in (2.52), although we are going to neglect this higher order term.
The second integral to compute is denoted $\underline{\Pi}$ and is given by

$$\Pi_{(\alpha)} := \int_{\mathbb{R}^3} |\underline{v}|^2 v_{(\alpha)} f_i^\varepsilon(\underline{v}) d\underline{v} = K_1 c_i^\varepsilon \int_{\mathbb{R}^3} \sum_{k=1}^3 (\eta_{(k)} + \varepsilon u_{i,\alpha}^\varepsilon)^2 (\eta_{(\alpha)} + \varepsilon u_{i,(\alpha)}^\varepsilon) e^{-K_2|\eta|^2} d\underline{\eta}.$$

By taking into account all terms that contain uneven powers of $\eta_{(\alpha)}$ yield trivial contributions to the integral, one obtains the representation

$$\Pi_{(\alpha)} := K_1 c_i^\varepsilon \sum_{k=1}^3 \int_{\mathbb{R}^3} \left(\varepsilon u_{i,(\alpha)}^\varepsilon (2\eta_{(k)}^2 \delta_{\alpha k} + \eta_{(k)}^2) + \varepsilon^3 [u_{i,(k)}^\varepsilon]^3\right) e^{-K_2|\eta|^2} d\underline{\eta}$$

Performing the expansion $\Pi_{(\alpha)} = \sum_k \Pi_{k,(\alpha)} \varepsilon^k$, one obtains by (2.43) and (2.51b) as before that

$$\Pi_{1,(\alpha)} = 5\varepsilon \frac{kT}{m_i} c_i^\varepsilon u_{i,(\alpha)}^\varepsilon$$

and $\Pi_{3,(\alpha)} = \sum_k [u_{i,(k)}^\varepsilon]^3 c_i^\varepsilon$, which we are going to neglect as higher order term.

We can now multiply (2.39) by $\frac{|\underline{v}|^2}{2}$, integrate it w.r.t. $\underline{v}$ and sum over all $1 \leq i \leq N+1$. As given by (2.25c), (2.34c) and the results of the expansion of the time derivative and divergence term $\Gamma$ and $\Pi$ in this section, one can equate the lowest order terms in $\mathcal{O}(\varepsilon)$

$$3\frac{\partial}{\partial t}\left[\sum_{k=1}^{N+1} c_i^\varepsilon\right] + 5\nabla_{\underline{x}} \cdot [c_i^\varepsilon \underline{u}_i^\varepsilon] = 0 \tag{2.59}$$

By performing the limit $\varepsilon \to 0$ in (2.59), there holds by using (2.46)

$$\sum_{i=1}^{N+1} \frac{\partial c_i}{\partial t} = 0 \tag{2.60a}$$

$$\nabla_{\underline{x}} \cdot \left[\sum_{k=1}^{N+1} \underline{J}_i\right] = 0. \tag{2.60b}$$

The equality (2.60b) is consistent with the closure relation (1.7). The equality (2.60a) ensures that the total molar mass $c = \sum_{k=1}^{N+1} c_i^0$ is conserved over time. Thus the derivation of the Maxwell-Stefan system (2.58) is consistent with the boundary conditions.

## 2.7 Remarks on Boudin's Diffusional Limit and outlook

The coefficients $\Delta_{ij}$ as derived in (2.55) can be identified with the binary Maxwell-Stefan diffusion coefficients $Đ_{ij}$, which have the physical dimension $m^2 s^{-1}$. As we have shown that under the given assumptions there holds $\Delta_{ij} = \Delta_{ji}$, one can also deduce the symmetry of the binary diffusion coefficients, which are normally derived by the Onsager relations, cf. [51] or [67, p. 31f].

Trying to better understand connections between a mesoscopic and microscopic aspects of diffusional processes is an active field of mathematical research, which is trying to reduce some of the assumptions necessary to the derive the results in section 2. For example, there have been various attempts to get rid of several assumptions made for the cross-section $B$, which were made in this chapter. This holds for example for Grad's angular cutoff assumption, which was needed for several ground-breaking proofs (e.g. [22, 30]) as this theorem has far-reaching consequences (e.g. [58, Th. 2.3.1, 2.3.4, 2.3.6, Prop. 3.2.1, 3.2.2]). There is currently quite an active field of research on trying to get rid of this assumption while still proving existence results for a (weak) solution to the Boltzmann equations. This is e.g. motivated as the cutoff assumption is known to propagate singularities in the solution, see e.g. [2, 6, 11] or to obtain better smoothness properties of the solution, see [1, 45]. Some of these efforts to overcome this shortcoming can e.g. be studied in [31].

There has also been a quite recent (2017) result to study more general cross-sections than considered here (as presenting the results in [14]) by Boudin himself, see [12].

# 3 Global-in-time existence proof

In this section, a step-by-step derivation of the key result in [40] is presented, which is based on the PhD thesis by Stelzer [65].
The proof presented here provides us with the existence of a (weak) solution of the Maxwell-Stefan equations under fairly general conditions. However, as far as the uniqueness of the solution is concerned, this is still an open problem as stated in [40].
Previously to Jüngel's paper there have been results by Giovangigli in his monograph [28] for the existence of a unique global solution of the Maxwell-Stefan system close to the thermodynamical equilibrium and Bothe has shown in [10] the existence of a *unique* classical local-in-time solution. Boudin studied a ternary system in [13], where he assumed that the diffusivities are equal, thus leading to a further simplification of the system, such that he could prove the existence of a global unique solution.
As stated in section 1, the Maxwell-Stefan system is given by (1.1). In order to prove the existence of a solution to this system, the system is reformulated by introducing a formulation exploiting the entropy structure of the problem. This method was developed and generalized by Jüngel to be applicable to many relevant systems in physics and biology, e.g. in [37] or [35, ch. 4].

## 3.1 Entropy formulation

In order to prove the existence of a solution to the Maxwell-Stefan equations, we make use of a technique which can be applied to cross-diffusion systems, as which the Maxwell-Stefan equations can be interpreted.
As Jüngel states in [37], the proof relies on the fact, that a reaction-diffusion system of the form

$$\frac{\partial u}{\partial t} - \operatorname{div}(A(u)\nabla u) = f(u), \tag{3.1a}$$

with initial values $u(\cdot, 0) = u^0$ and homogeneous Neumann boundary conditions possesses a gradient-flow structure, i.e. it can be written in the form

$$\frac{\partial u}{\partial t} - \operatorname{div}\left(B\nabla\frac{\delta\mathcal{H}}{\delta u}\right) = f(u). \tag{3.1b}$$

This leads us to the following definition explaining above terms (taken from [35, Def. 4.1]):

**Definition 3.1 (Entropy formulation).** *The equality* (3.1b) *is called the entropy formulation of* (3.1a)*, if there exists a convex function $h : \mathcal{D} \to [0, \infty)$ (which is called* ***entropy density****), which defines the variational (Fréchet-)derivative $\frac{\delta\mathcal{H}}{\delta u}$ of the* ***entropy functional***

$$\mathcal{H}[u] = \int_\Omega h(u(t, \underline{x}))d\underline{x} \tag{3.2}$$

*and the matrix $B = h''(u)A(u)$ is positive definite.*

By identification of the Fréchet-derivative of $\mathcal{H}$ with its Riesz representative $Dh(u)$ one can introduce the entropy variable

$$w = Dh(u). \tag{3.3}$$

By this definiton, the system (3.1b) can be written in entropy variables as

$$\frac{\partial u(w)}{\partial t} - \operatorname{div}\left(B(w)\nabla w\right) = f(u(w)). \tag{3.4}$$

One can show that if there holds $f(u)\cdot w \leq 0$ and $B$ is positive semidefinite, then $\mathcal{H}$ is a Lyapunov functional (i.e. $\frac{d}{dt}\mathcal{H}(u(t)) \leq 0$ for all $t > 0$). Hence it is justified that $\mathcal{H}$ can be interpreted as an entropy.
In order to exploit the entropy structure of (1.9), one introduces the entropy density associated to the Maxwell-Stefan system by

$$h(\underline{c}) = \sum_{i=1}^{N} c_i(\ln(c_i)-1) + c_{N+1}(\ln(c_{N+1})-1) \quad c_1,\dots,c_N \geq 0, \sum_{i=1}^{N} c_i \leq 1, \tag{3.5}$$

where $c_{N+1}(\underline{c}(\underline{x},t))$ can be determined by $\underline{c} = (c_1,\dots,c_N)$ via (1.6). By definition of the entropy density (3.5) and (3.3), one can define the entropy variables for the Maxwell-Stefan equation by

$$w_i = \frac{\partial h}{\partial c_i} = \ln\left(\frac{c_i}{c_{N+1}}\right). \tag{3.6}$$

Similarly, one can compute the Hessian $H(\underline{c}) = \nabla^2 h(\underline{c})$, which is given by

$$H(\underline{c}) = \nabla^2 h(\underline{c}) = \frac{\partial^2 h}{\partial c_i \partial c_j} = \frac{1}{c_{N+1}} + \frac{\delta_{ij}}{c_i}. \tag{3.7}$$

It is important that (3.6) can be inverted.[17] Indeed one can derive from (3.6) that there holds equivalently

$$c_i(1+e^{w_i}) + e^{w_i}\sum_{\substack{j=1\\ j\neq i}}^{N} c_j = e^{w_i} \quad 1 \leq i \leq N,$$

which can be written in matrix form via

$$\underbrace{\left[I_N + \underline{\kappa}\otimes\underline{\mathbb{1}}\right]}_{\Xi}\cdot\underline{c} = \underline{\kappa}, \tag{3.8}$$

where $I_N \in \mathbb{R}^{N\times N}$ is the identity matrix, $\underline{\mathbb{1}} = (1,\dots,1)^T \in \mathbb{R}^N$ and $\underline{\kappa} = (e^{w_1},\dots,e^{w_N})^T \in \mathbb{R}^N$ for $1 \leq i \leq N$. The matrix $\Xi$ can be inverted by using the Sherman-Morrison formula (cf. Lemma A.1), where by substituting $A = I_N$ in (A.4) there holds

$$(I_N + \underline{\kappa}\otimes\underline{\mathbb{1}})^{-1} = I_N - \frac{\underline{\kappa}\otimes\underline{\mathbb{1}}}{1+\underline{\kappa}\cdot\underline{\mathbb{1}}}. \tag{3.9}$$

By using (3.9) to invert (3.8) there holds

$$c_i = [\Xi^{-1}\cdot\underline{\kappa}]_{(i)} = \sum_{j=1}^{N}\left(\delta_{ij} - \frac{e^{w_i}}{1+\sum_{k=1}^{N} e^{w_k}}\right)e^{w_j} = e^{w_i}\left(1 - \frac{\sum_{j=1}^{N} e^{w_j}}{1+\sum_{k=1}^{N} e^{w_k}}\right).$$

[17] The existence of the inverse is important as we want to be able to recover the original solution $c_i$ from the entropy variables $\underline{w}$, cf. hypothesis **H1** in section 3.2.

By simplifying the above, one obtains

$$c_i = \frac{e^{w_i}}{1 + \sum_{j=1}^{N} e^{w_j}}. \tag{3.10}$$

Hence one can interchange between the variables $c_i$ and the entropy variables $w_i$ for $1 \leq i \leq N$ via (3.6) and (3.10).
Having defined the entropy variables, (1.9) can be reformulated in entropy variable formulation as

$$\frac{\partial}{\partial t}\underline{c} - \operatorname{div}(B(\underline{w})\nabla\underline{w}) = \underline{r}(\underline{c}), \tag{3.11}$$

where $B \in \mathbb{R}^{N\times N}$ is given by

$$B(w) = A_0(c)^{-1}H(c)^{-1}. \tag{3.12}$$

In (3.12) the reduced matrix $A_0 \in \mathbb{R}^{N\times N}$ is given by (1.8).

## 3.2 Outline of the proof

While the original existence result for the Maxwell-Stefan equations has been formulated in [40], the technicalities of the proof have been further generalized in [37], which provides a high level view on which conditions have to be met in order to be able to prove the existence of a solution. Jüngel has formulated four hypotheses in [35, 37], which need to be proven for a given system in order for proving existence of a weak solution via the boundedness-by-entropy approach:

**H1:** There exists a convex nonnegative function $h \in C^2(\mathcal{D}, [0,\infty))$ with $\mathcal{D} \subset \mathbb{R}^n$ and $n \geq 1$, such that its derivative $Dh : \mathcal{D} \to \mathbb{R}^n$ is invertible on $\mathbb{R}^n$.

**H2':** Let $\mathcal{D} \subset (0,1)^n$ and for all $z = (z_1, \ldots, z_n) \in \mathbb{R}^n$ and $u = (u_1, \ldots, u_n) \in \mathcal{D}$ there shall hold that

$$z^T D^2h(u)A(u)z \geq \sum_{i=1}^{n} u_i^{2a} z_i^2 \quad a \geq -\frac{1}{2}. \tag{3.13}$$

**H2":** There exists an $\alpha^* > 0$ such that for all $u \in \mathcal{D}$ and $1 \leq i, j \leq n$ there holds $|A_{ij}| \leq a^* |u_j|^a$.

**H3:** There holds $A \in C^0(\mathcal{D}, \mathbb{R}^{n\times n})$, $f \in C^0(\mathcal{D}, \mathbb{R}^n)$, and there exists a positive constant $C_f \in \mathbb{R}^+$ such that

$$f(u) \cdot h'(u) \leq C_f(1 + h(u)) \quad \forall u \in \mathcal{D}. \tag{3.14}$$

In section 3.4, we will prove the necessary properties for the Maxwell-Stefan system to show that the hypotheses H1-H3 do in fact hold in combination with the entropy density defined in (3.5). By that method one can prove that there exists a weak solution to the Maxwell-Stefan system (1.1).

## 3.3 Assumptions for the Maxwell-Stefan system

In order to rigorously prove the existence of (weak) solutions to the Maxwell-Stefan system, we need to state several assumptions for the system, which partly follow naturally from the physical interpretations of the quantities involved. The existence of a solution can be derived under the following assumptions:

**Assumption 3.1 (Domain).** *The domain $\Omega \subset \mathbb{R}^d$ for $1 \leq d \leq 3$ is a bounded domain with a Lipschitz boundary, i.e. there holds $\partial\Omega \in C^{0,1}$.*

The restriction on the smoothness on the domain will be needed later on in the proof to use various embedding arguments, cf. section A.2.2.
For the initial data there holds that they are nonnegative measurable functions, indeed as $c_i$ are assumed to be molar concentrations, there holds that $\underline{c}^0 \in [0,1]^{N+1}$ naturally by the governing physical interpretation. Hence we assume the following:

**Assumption 3.2 (Initial data).** *The initial molar concentrations $\underline{c}^0 = (c_1^0, \ldots, c_N^0)^T$ for $N \geq 2$ are non-negative measurable functions, and there holds $c_{N+1}^0 = 1 - \sum_{j=1}^N c_j^0$ and*

$$\sum_{i=1}^{N} c_i^0 \leq 1 \tag{3.15}$$

In section 2 we have derived the representation (2.55) for the Maxwell-Stefan diffusion coefficients $Đ_{ij}$, which justifies that the coefficients are symmetrical and positive. Thus it is quite natural to make the following assumption:

**Assumption 3.3 (Diffusion matrix).** *The diffusion matrix $Đ_{ij} \in \mathbb{R}^{(N+1)\times(N+1)}$ is a symmetrical matrix with $Đ_{ij} > 0$ for $i \neq j$.*

In hypothesis **H3** in section 3.2 we have seen that is important to have appropriate bounds for the source term, which will be made more clear in the proof in section 3.5. In order to satisfy **H3** with the entropy density (3.5) for the Maxwell-Stefan system, there is the following restriction on the source term to be imposed:

**Assumption 3.4 (Production rates).** *The production rate functions $r_i \in C^0([0,1]^{N+1}, \mathbb{R})$ for $1 \leq i \leq N+1$ satisfy the relations*

$$\sum_{i=1}^{N+1} r_i(\underline{c}) = 0 \tag{3.16a}$$

$$\sum_{i=1}^{N+1} r_i(\underline{c}) \ln(c_i) \leq C_r \quad \forall \underline{c} \in [0,1]^{N+1} \tag{3.16b}$$

*where $C_r \in \mathbb{R}^+$ is a positive constant.*

### 3.4 Important bounds for the Maxwell-Stefan system

In order to prove the existence of a solution to the Maxwell-Stefan system (1.1), we need to further investigate appropriate bounds for the diffusion matrix $A(\underline{c})$ as given in (1.5).
For that purpose, the following assumption is made:

**Assumption 3.5 (Positivity of molar concentrations).** *In this section, we assume that $\underline{c}$ is strictly positive, i.e. there holds $c_i \geq \eta$ for $1 \leq i \leq N$ ($c_{N+1}$ is given by (1.6)) and $\eta \in \mathbb{R}^+$.*

By taking into account these properties for the Maxwell-Stefan cross diffusion matrix $A(\underline{c})$, we can prove the following result, which was originally proven in [10] and is also given in [40]:

**Lemma 3.1 (Properties of $A$).** *Let $\delta = \min_{i,j=1,\dots,N+1,i\neq j} d_{ij} > 0$ and $\Delta = 2\sum_{i,j=1,i\neq j}^{N+1} d_{ij}$. Then the spectrum $\sigma(-A)$ of $-A$ (where $A$ is given by (1.5)) satisfies*

$$\sigma(-A) \subset \{0\} \cup [\delta, \Delta).$$

*Proof.* By assumption 3.3, the matrix $A$ as defined in (1.5) is quasi-positive (cf. definition A.5) and irreducible (cf. definition A.3) by inspection, as the off-diagonal elements are all strictly positive. By Theorem A.1 there holds that the spectral bound $s(A)$ (cf. definition A.2) is a simple eigenvalue of $A$. Furthermore Theorem A.1 states that the eigenvector $\underline{v}$ associated with $\rho(A)$ is strictly positive, i.e. $v_{(i)} > 0$ for $1 \leq i \leq N$. By definition A.2, as $s(A)$ is the eigenvalue with the largest real part, one can represent the spectrum $\sigma(A)$ of the matrix $A$ via

$$\sigma(A) \subset \{s(A)\} \cup \{z \in \mathbb{C} : \Re(z) < s(A)\}.$$

The next step is to show that $\underline{c}' \in \mathbb{R}^{N+1}$ is in fact a strictly positive eigenvector to the eigenvalue 0, thus by definition there holds $(A - 0 \cdot I_{N+1})\underline{c}' = \underline{0}$. Indeed, by the definition (1.5) of $A$ there holds

$$[A\underline{c}']_{(i)} = A_{ii}c_i + \sum_{\substack{j=1\\ j\neq i}}^{N+1} A_{ij}c_j = -\sum_{\substack{j=1\\ j\neq i}}^{N+1} d_{ij}c_jc_i + \sum_{\substack{j=1\\ j\neq i}}^{N+1} d_{ij}c_ic_j = 0 \quad 1 \leq i \leq N+1.$$

As according to Theorem A.1 only the eigenvector associated with $s(A)$ is strictly positive, this implies $s(A) = 0$ and thus

$$\sigma(A) \subset \{0\} \cup \{z \in \mathbb{C} : \Re(z) < 0\}.$$

The analysis of the spectrum can be further refined by introducing the matrix $C = \operatorname{diag}\left(\sqrt{c_1}, \dots, \sqrt{c_{N+1}}\right)$ and performing the similarity transformation to obtain $A_S = CAC^{-1}$, where $C^{-1}$ is given by $C^{-1} = \operatorname{diag}\left(\frac{1}{\sqrt{c_1}}, \dots, \frac{1}{\sqrt{c_{N+1}}}\right)$. By inspection there holds for the elements $a_{ij}^S := (A_S)_{ij}$

$$a_{ij}^S = \begin{cases} a_{ii} & i = j \\ d_{ij}\sqrt{c_ic_j} & i \neq j \end{cases} \quad 1 \leq i, j \leq N+1, \tag{3.17}$$

thus $A_S$ is a real symmetric matrix, for which the spectral theorem [32, Satz 12.3.6] can be applied, yielding $A_S$ diagonalizable with an orthonormal basis of eigenvectors. Furthermore $A$ and $A_S$ are similar (via the basis change described by $C$), thus their spectra coincide. As all eigenvalues $\lambda_i \in \mathbb{R}$ of $A_S$ are known to be situated on the real axis of the complex plane (cf. [32, Satz 12.6.6]), the spectrum of $A$ reduces to

$$\sigma(A_S) = \sigma(A) \subset \{0\} \cup \{z \in \mathbb{R} : z < 0\} = (-\infty, 0].$$

The bounds can be further refined by the following argument: One introduces the matrix $A_s(\alpha) := A_S - D_\alpha$ where $\alpha \in (0, \delta)$,$\sqrt{\underline{c}'} := \left(\sqrt{c_1}, \ldots, \sqrt{c_{N+1}}\right)$ and $D_\alpha := \alpha\sqrt{\underline{c}'} \otimes \sqrt{\underline{c}'}$.[18] Furthermore, $-\alpha$ is an eigenvalue of $A_S(\alpha)$ with the strictly positive eigenvector $\sqrt{\underline{c}}$, which can be proven by computing

$$\begin{aligned}
\left[\left(A^S(\alpha) + \alpha I_{N+1}\right)\sqrt{\underline{c}'}\right]_{(i)} &= (a_{ii}^S(\alpha) + \alpha)\sqrt{c_i} + \sum_{\substack{j=1\\ j\neq i}}^{N+1} a_{ij}^S(\alpha)\sqrt{c_j} \\
&= \left( -\sum_{\substack{j=1\\ j\neq i}}^{N+1} d_{ij}c_j - \alpha c_i + \alpha \right)\sqrt{c_i} + \sum_{\substack{j=1\\ j\neq i}}^{N+1} (d_{ij}\sqrt{c_i c_j} - \alpha\sqrt{c_i c_j})\sqrt{c_j} \\
&= -\sum_{\substack{j=1\\ j\neq i}}^{N+1} d_{ij}c_j\sqrt{c_i} + \alpha(1 - c_i)\sqrt{c_i} + \sum_{\substack{j=1\\ j\neq i}}^{N+1} (d_{ij}c_j\sqrt{c_i} - \alpha c_j\sqrt{c_i}) \\
&= \sqrt{c_i}\left[\alpha(1 - c_i) - \alpha\sum_{\substack{j=1\\ j\neq i}}^{N+1} c_j\right] \overset{(1.6)}{=} \sqrt{c_i}\left[\alpha(1 - c_i) - \alpha(1 - c_i)\right] = 0.
\end{aligned}$$

By Theorem A.1 there holds for $\alpha \in (0, \delta)$

$$\sigma(A_S(\alpha)) \subset (-\infty, -\alpha].$$

As $A_S(\alpha)$ and $D_\alpha$ are both symmetric matrices, one can apply Theorem A.2 to get the bounds

$$\lambda_i(A_S) = \lambda_i(A_S(\alpha) + D_\alpha) \leq \lambda_i(D_\alpha) + \underbrace{\lambda_{N+1}(A_S(\alpha))}_{=-\alpha}$$

By Lemma A.3 there holds $\lambda_i(D_\alpha) = 0$ for $1 \leq i \leq N$ and $\lambda_{N+1}(D_\alpha) = \alpha|\underline{c}| = \alpha$. Thus we are able to restrict the bounds also for $\sigma(A_S)$ by $\lambda_i \leq -\alpha$ and $\lambda_{N+1} \leq 0$. As $A_S$ is similar to $A$, there holds for $\alpha \in (0, \delta)$

$$\sigma(A_S) = \sigma(A) \subset \{0\} \cup (-\infty, -\alpha].$$

This implies $\sigma(-A) \subset \{0\} \cup [\delta, \infty)$.
Now the upper bound of the spectrum of $-A$ needs to be computed. Let $\|\cdot\|_F$ denote the Frobenius norm and as the spectral norm $\|\cdot\|_2 = \rho(\cdot)$ can be bounded by the Frobenius norm

[18] Note that $(\sqrt{\underline{c}'} \otimes \sqrt{\underline{c}'})_{ij} = \sqrt{c_i c_j}$ and by the definition of $\delta$ for the entries $a_{ij}^S(\alpha)$ of the matrix $A_S(\alpha)$ there still holds $a_{ij}^S(\alpha) > 0$ for $i \neq j$, thus the matrix is still quasi-positive and irreducible.

(cf. [3, p. 39]), there holds

$$\rho(-A) \leq \|-A\|_F = \left( \sum_{i,j=1}^{N+1} a_{ij}^2 \right)^{\frac{1}{2}} = \left( \sum_{i=1}^{N+1} \left( \sum_{\substack{j=1 \\ j\neq i}}^{N+1} d_{ij} c_j \right)^2 + \sum_{\substack{i,j=1 \\ j\neq i}}^{N+1} (d_{ij} c_i)^2 \right)^{\frac{1}{2}}$$
$$\overset{\text{(A.27)}}{\leq} \left( 3 \sum_{\substack{i,j=1 \\ j\neq i}}^{N+1} (d_{ij} c_i)^2 \right)^{\frac{1}{2}} < 2 \sum_{\substack{i,j=1 \\ j\neq i}}^{N+1} d_{ij} = \Delta.$$

by making use of $c_i \in (0,1)$. □

The next step is to check the following restrictions of $A$ and $A_S$ on appropriate subspaces can be inverted:

**Lemma 3.2.** *Let $\widetilde{A} = A|_{im(A)}$ with $A$ defined in (1.5) and $\widetilde{A}_S = A_S|_{im(A_S)}$ with $A_S = CAC^{-1}$ and $C = \operatorname{diag}\left(\sqrt{c_1}, \ldots, \sqrt{c_{N+1}}\right)$. Then $A$ and $A_S$ are invertible on their images $A$ and $A_S$ and there holds*

$$\sigma(-\widetilde{A}) = \sigma(-\widetilde{A}_S) \subset [\delta, \Delta) \tag{3.18a}$$
$$\sigma((-A_S)^{-1}) \subset (1/\Delta, 1/\delta] \tag{3.18b}$$

*with $\delta, \Delta \in \mathbb{R}$ defined as in Lemma 3.1.*

*Proof.* As seen in the proof of Lemma 3.1, we have shown that zero is a simple eigenvalue to $A$ with eigenvector $\underline{c}$, hence $\ker(A) = \operatorname{span}(\underline{c})$. By the same argumentation and the proof for Lemma 3.1, we have proven for the matrix $A_S$ defined in (3.17) that $\ker(A_S) = \operatorname{span}(\sqrt{\underline{c}'})$. By a direct computation and a representation of the space $\{\mathbb{1}\}^\perp$ one can show that $\operatorname{im}(A) = \{\mathbb{1}\}^\perp$, where $\mathbb{1} = (1, \ldots, 1)^T \in \mathbb{R}^{N+1}$. As $A_S$ is symmetric, it follows by [32, Satz 9.8.5, 12.1.7] that

$$\mathbb{R}^{N+1} = \ker(A_S)^\perp \oplus \ker(A_S) = \operatorname{im}((A_S)^T) \oplus \ker(A_S) = \operatorname{im}(A_S) \oplus \ker(A_S). \tag{3.19}$$

Furthermore by [61, Th. 3.4], as 0 is a real and semisimple eigenvalue of $A$ by Lemma 3.1, there holds

$$\mathbb{R}^{N+1} = \operatorname{im}(A) \oplus \ker(A). \tag{3.20}$$

Furthermore both $\widetilde{A}$ and $\widetilde{A}_S$ are endomorphisms and by the definition of these maps there holds $\sigma(\widetilde{A}) \subset \sigma(A)$ and $\sigma(\widetilde{A}_S) \subset \sigma(A_S)$. We claim that $\underline{0}$ is not contained in $\sigma(\widetilde{A})$ or $\sigma(\widetilde{A}_S)$. This is proven by contraposition. Suppose $\underline{0} \in \sigma(\widetilde{A})$ and $\underline{0} \in \sigma(\widetilde{A}_S)$ then there exists a $\underline{x} \neq \underline{0}$, such that $\widetilde{A}\underline{x} = \underline{0}$ ($\widetilde{A}_S\underline{x} = \underline{0}$ respectively). However, this implies $\underline{x} \in \ker(\widetilde{A})$ (respectively $\underline{x} \in \ker(\widetilde{A}_S)$). However, as the nullspace of the map has no intersection with the image except $\underline{x} = \underline{0}$ due to (3.20) (respectively (3.19)), this is a contradiction to the assumption $\underline{x} \neq \underline{0}$. By contraposition, $\widetilde{A}$ and $\widetilde{A}_S$ are invertible on their respective domain, and in combination with the results from Lemma 3.1 there holds (3.18). □

Due to Lemma 3.2 one can invert (1.4) as due to $\sum_{i=1}^{N+1} \underline{J}_{i,(\iota)} = 0$ for $0 \leq \iota \leq d$ from (1.7) there holds that $\underline{J}'_{(\iota)} = (J_{1,(\iota)}, \dots, J_{N+1,(\iota)}) \in \{\mathbb{1}\}^{\perp} = \mathrm{im}(A)$. Thus by inverting (1.4) and obtaining $\nabla c' = \widetilde{A} J'$ as in (1.1a), the latter can be written as

$$\frac{\partial c}{\partial t} - \mathrm{div}_{\underline{x}}(-\widetilde{A}(\underline{c})\nabla \underline{c}) = r(c). \tag{3.21}$$

As already introduced in section 1, an $N$-dimensional system by using the closure relations (1.6) and (1.7) is considered. In order to perform the dimensional reduction, one introduces the matrices for a basis change via[19]

$$X = I_{N+1} - \Lambda, \quad X^{-1} = I_{N+1} + \Lambda, \quad \Lambda = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 0 \end{pmatrix}.$$

This coordinate transformation can e.g. be motivated from section 1 in the derivation of the reduced matrix $A_0$ in (1.8). By performing a basis change on (3.21) to the $X$-basis, there holds in the variables $\underline{c} = (c_1, \dots, c_N)^T \in \mathbb{R}^N$ and $J = (\underline{J}_1, \dots, \underline{J}_N)^T \in \mathbb{R}^{N \times d}$ by (1.6) and (1.7) that

$$\begin{pmatrix} \underline{c} \\ 1 \end{pmatrix} = X^{-1} \underline{c}$$
$$\begin{pmatrix} J \\ 0 \end{pmatrix} = X^{-1} J.$$

As seen in section 1 one can compute that in the coordinates in the $X$-Basis there holds

$$\begin{pmatrix} \nabla \underline{c} \\ 0 \end{pmatrix} = X^{-1} \nabla \underline{c} = (X^{-1} A X) X^{-1} J = \begin{pmatrix} -A_0 J \\ 0 \end{pmatrix},$$

where $A_0$ is given by (1.8). Applying $X^{-1}$ to (3.21) (performing the basis transformation to $X$-coordinates) one obtains an $N$-dimensional system

$$\frac{\partial c_i}{\partial t} - \mathrm{div}_{\underline{x}}(A_0^{-1} \nabla \underline{c}_i) = r_i(\underline{c}) \quad 1 \leq i \leq N, \tag{3.22}$$

where $c_{N+1}$ and $\underline{J}_{N+1}$ can by recovered by the closure relations (1.6) and (1.7).
Thus we have used the closure relations to reformulate the system as a lower dimensional system where the diffusion matrix $A$ (now $A_0$) can be inverted. In order to make this statement more precise, we need to prove the following relation for $A_0$:

**Lemma 3.3 (Properties of $A_0$).** *The matrix $A_0 \in \mathbb{R}^{N \times N}$ from (1.8) can be inverted and its spectrum can be bounded by*

$$\sigma(A_0) \subset [\delta, \Delta),$$

*where $\delta, \Delta \in \mathbb{R}$ are defined as in Lemma 3.1. The elements of $A_0^{-1}(\underline{c})$ are uniformly bounded for $\underline{c} \in [0,1]^N$.*

[19] The inverse $X^{-1}$ can e.g. be computed from $X$ via (3.9) and vice versa, as seen before.

*Proof.* As $A_0 = -X^{-1}AX$ is similar to $-A$, their spectra coincide. As

$$X^{-1}AX = \begin{pmatrix} -A_0 & b \\ \underline{0} & 0 \end{pmatrix}$$

is an upper triangular block matrix and 0 is a simple eigenvalue of $A$ (cf. Lemma 3.1), zero can't be an eigenvalue of $A_0$ and the latter is thus invertible. From Lemma 3.1 there even holds

$$\sigma(A_0) \cup \{0\} = \sigma(-A) \subset \{0\} \cup [\delta, \Delta),$$

thus $\sigma(A_0) \subset [\delta, \Delta)$. Now one needs to prove at the elements of $A_0^{-1}$ are bounded from above. The entries of $A_0$ as given by (1.8) can be bounded by

$$|a_{ij}^0| \leq |d_{i,N+1}| + \sum_{\substack{k=1 \\ k \neq i}}^{N} |d_{ik} - d_{i,N+1}| =: K_i(Đ) \leq K(Đ).$$

Above, we have defined $K := \max_{1 \leq i \leq N} K_i > 0$, for which holds $|a_{ij}^0| \leq K < \infty$. By Lemma A.4 one can represent $A_0^{-1}$ via (A.7). As the determinant of a matrix can be represented by the product of its eigenvalues, there holds $\det(A_0) \geq \delta^N$. As we've established that $A_0$ is positive definite, there holds for the elements $\widetilde{a}_{ij}^0$ of $\operatorname{adj}(A_0)$ defined in (A.1b) by a simple corollary of Hadamard's inequality (A.9), that $|\widetilde{a}_{ij}^0| \leq K^{N-1}(N-1)^{\frac{N-1}{2}}$. Thus there holds

$$|(A_0^{-1})_{ij}| = \frac{|\widetilde{a}_{ij}^0|}{\det(A_0)} \leq \delta^{-N} K^{N-1}(N-1)^{\frac{N-1}{2}} =: C(N, Đ) < \infty, \tag{3.23}$$

which concludes the proof. □

> **Remark 3.1.** *The uniform boundedness of the inverse $A_0^{-1}$ from Lemma 3.3 actually corresponds to hypothesis **H2"** from section 3.2.*

We will now study the Hessian matrix of the entropy density defined in (3.5):

**Lemma 3.4.** *The Hessian* (3.7) *of the Maxwell-Stefan entropy density* (3.5) *defines an SPD (symmetric, positive definite) matrix.*

*Proof.* By inspection of (3.7), there holds $h_{ij} = h_{ji}$, thus it is symmetric. Hence for the proof of the positive definiteness of $H$, one can use the principal minor criterion (cf. [32, Satz 9.10.13]), which yields positive definiteness if for all principal minors $H_k \in \mathbb{R}^{k \times k}$ (cp. definition A.8) and $1 \leq k \leq N$ there holds $\det(H_k) > 0$.
As $H_k$ for $H$ in (3.7) can be represented by a rank-one perturbation of a regular matrix via

$$H_k = \operatorname{diag}\left(\frac{1}{c_1}, \ldots, \frac{1}{c_k}\right) + \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \otimes \begin{pmatrix} c_{N+1} \\ \vdots \\ c_{N+1} \end{pmatrix}, \tag{3.24}$$

there holds by (A.5) in Lemma A.2 that

$$\begin{aligned}\det(H_k) &= \left(\prod_{i=1}^{k}\frac{1}{c_i}\right)\left(1+\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}^T \operatorname{diag}(c_1,\dots,c_k)\begin{pmatrix}c_{N+1}\\ \vdots\\ c_{N+1}\end{pmatrix}\right)\\ &= \left(\prod_{i=1}^{k}\frac{1}{c_i}\right)\left(1+c_{N+1}\sum_{j=1}^{k}c_j\right) > 0,\end{aligned}$$

as $c_\alpha \in (0,1)$ for $1 \le \alpha \le N+1$ due to assumption 3.5. Thus $H$ is SPD. □

With these results we can now prove the following properties for the entropy matrix $B$ in (3.4):

**Lemma 3.5 (Properties of $B$).** *The matrix $B = A_0^{-1}H^{-1}$ is an SPD matrix. Furthermore, the elements of $B$ are bounded uniformly in $\underline{c} \in [0,1]^{N+1}$.*

*Proof.* By the definition of $A_0$ in (1.8) and $H$ in (3.7), one can compute the elements $\beta_{ij}$ of $B^{-1} = HA_0$ as

$$\beta_{ij} = \begin{cases} d_{i,N+1}\left(1-\sum_{\substack{k=1\\k\neq i}}^{N}c_k\right)\left(\frac{1}{c_i}+\frac{1}{c_{N+1}}\right)+\sum_{\substack{k=1\\k\neq i}}^{N}\left(\frac{d_{k,N+1}}{c_{N+1}}+\frac{d_{ik}}{c_i}\right)c_k & i=j\\ \frac{d_{i,N+1}}{c_{N+1}}\left(1-\sum_{\substack{k=1\\k\neq i}}^{N}c_k\right)\frac{d_{j,N+1}}{c_{N+1}}\left(1-\sum_{\substack{k=1\\k\neq j}}^{N}c_k\right)+\sum_{\substack{k=1\\k\neq j,i}}^{N}d_{k,N+1}\frac{c_k}{c_{N+1}}-d_{ij} & i\neq j,\end{cases}$$

thus by inspection $B^{-1}$ is symmetric. Due to Lemma A.5 (as we've proven that $H$ is SPD in Lemma 3.4), $H^{-1}$ is SPD as well. Using Theorem A.4, the number of positive eigenvalues of $A_0 = H^{-1}B^{-1}$ equals the one for $B^{-1}$, thus $B^{-1}$ has only positive eigenvalues by Lemma 3.3. Hence $B^{-1}$ and $B$ (again by Lemma A.5) are SPD. Now the uniform boundedness of $B$ shall be proven. The inverse $H^{-1} = (\mathfrak{H}_{ij})_{1\le i,j\le N}$ can be computed by using the rank-one perturbation representation of $H$ from (3.24) (as $H_N = H$) and employing the Sherman-Morrison formula (A.4) to arrive at

$$\mathfrak{H}_{ij}(\underline{c}') = \begin{cases}(1-c_i)c_i & i=j\\ -c_ic_j & i\neq j\end{cases} \quad 1\le i,j\le N.$$

By naming $\alpha_{ij} := (A_0^{-1})_{ij}$, one can represent the elements of $B$ by

$$b_{ij}(\underline{c}, A_0^{-1}) = \begin{cases}\alpha_{ii}(1-c_i)c_i - \sum_{\substack{k=1\\k\neq i}}^{N}\alpha_{ik}c_ic_k & i=j\\ -\alpha_{ii}c_ic_j+\alpha_{ij}(1-c_j)c_j-\sum_{\substack{k=1\\k\neq i,j}}^{N}\alpha_{ik}c_ic_k & i\neq j\end{cases} \quad 1\le i,j\le N. \tag{3.25}$$

As in Lemma 3.3 uniform bounds for $\alpha_{ij}$ in $\underline{c}\in\mathbb{R}^N$ were presented in (3.23) and $\underline{c}\in[0,1]^N$, it is straightforward from above representation that

$$|b_{ij}(\underline{c}(\underline{w}))| \le N \max_{1\le i,j\le N}|\alpha_{ij}| \overset{(3.23)}{\le} NC(N,\text{Đ}) = \widetilde{C}(N,\text{Đ}), \tag{3.26}$$

thus all $b_{ij}$ are uniformly bounded in $\underline{c}$ for $1\le i,j\le N$. □

We now have all the properties which are needed to prove the existence of a solution via the boundedness-by-entropy method. The analysis necessary on $A$ is quite general for existence proofs by the boundedness-by-entropy method, which is summarized in the following remark:

**Remark 3.2.** *Jüngel has stated in [35, Rem. 4.3] that in order for $B(w)$ in (3.4) to be SPD, the eigenvalues of the original Cross-Diffusion matrix $A(u)$ from (3.1a) all need to be positive (which was made plausible in the proof of Lemma 3.5 when employing Theorem A.4). Thus this is a requirement for any Cross-Diffusion system if $B$ shall be positive definite.*

## 3.5 Existence proof by the boundedness-by-entropy method

In this section, the existence of a solution to the initial-boundary value problem of the reduced Maxwell-Stefan system in entropy variables is derived, which is given as

$$\frac{\partial}{\partial t}c_i(\underline{w}(\underline{x},t)) - \operatorname{div}_{\underline{x}}(B(\underline{w})\nabla_{\underline{x}}w(\underline{x},t)) = r_i(\underline{c}(\underline{x},t)) \tag{3.27a}$$

$$\nabla w_i \cdot \underline{\nu} = 0 \text{ on } \partial\Omega \quad w_i(\cdot,0) = w_i^0 \text{ on } \Omega \tag{3.27b}$$

for $1 \leq i \leq N$. In (3.27), $c_i(\underline{w})$ can be recovered by (3.10). We still assume that $\underline{c} \in (0,1)^{N+1}$ as in assumption 3.5 in this section.
The initial values $w_i^0$ can be computed from $c_i^0$ via (3.6), for which holds $w_i \in L^\infty(\Omega)$ for $1 \leq i \leq N$ if $\underline{c} \in (0,1)^{N+1}$. Then there holds the following theorem:

**Theorem 3.1 (Global existence of a solution).** *Let assumptions 3.1, 3.2, 3.3, 3.4 hold. Then there exists a weak solution to (1.1) which satisfies*

$$c_i \in L^2_{loc}(0,\infty;H^1(\Omega)), \quad \frac{\partial c_i}{\partial t} \in L^2_{loc}(0,\infty;\mathcal{V}')$$

*and $\underline{c}(\cdot,t) = (c_1,\dots,c_{N+1}) \in [0,1]^{N+1}$ in $\Omega$ and $t > 0$. $\mathcal{V}'$ denotes the topological dual space of*

$$\mathcal{V} = \{u \in H^2(\Omega) : \nabla u \cdot \nu = 0\}. \tag{3.28}$$

**Remark 3.3.** *Theorem 3.1 has been proven with the hypotheses **H1**, **H2'**, **H2"**, **H3** from section 3.2 in [35, Thm. 4.1] instead of the customly tailored assumptions in section 3.3, which generalizes the proof to many other systems relevant in physics and biology, see [35, ch. 4].*

The proof is performed in several steps, which will be split into several sections for better readability.

### 3.5.1 Step 1: Definition of an approximate system

The first step is to introduce a weak formulation for (3.27). By multiplying (3.27a) with an appropriate test function $\underline{v} \in \mathcal{V}^N$ ($\mathcal{V}^N$ is the cartesian product of the function space $\mathcal{V}$ defined

in (3.28)) and performing integration over the spacial domain $\Omega$, one obtains

$$\int_\Omega \sum_{i=1}^N \frac{\partial c_i}{\partial t}(\underline{w}(t,\underline{x}))v_i(\underline{x})d\underline{x} - \int_\Omega \sum_{i=1}^N \left[\sum_{k=1}^d \frac{\partial}{\partial x_k}\left(\sum_{j=1}^N b_{ij}(\underline{w})\frac{\partial w_j}{\partial x_k}\right) v_i\right] d\underline{x} = \int_\Omega \sum_{i=1}^N r_i(\underline{c}(\underline{w}))v_i d\underline{x}.$$

By employing integration by parts and using the homogeneous Neumann boundary condition (3.27b) on the boundary integral, one can infer

$$\begin{aligned}
-\int_\Omega \sum_{i=1}^N \left[\sum_{k=1}^d \frac{\partial}{\partial x_k}\left(\sum_{j=1}^N b_{ij}\frac{\partial w_j}{\partial x_k}\right) v_i\right] d\underline{x} &= \int_\Omega \sum_{i=1}^N \left[\sum_{k=1}^d \left(\sum_{j=1}^N b_{ij}\frac{\partial w_j}{\partial x_k}\right)\frac{\partial v_i}{\partial x_k}\right] d\underline{x} \\
&\quad - \int_{\partial\Omega} \sum_{i,j=1}^N v_i b_{ij} \underbrace{\nabla_{\underline{x}} w_j \cdot \underline{\nu}}_{\overset{(3.27b)}{=}0} d\underline{x} \\
&= \int_\Omega \sum_{i=1}^N \sum_{j=1}^N b_{ij}\left(\nabla_{\underline{x}} w_j \cdot \nabla_{\underline{x}} v_i\right) d\underline{x} \\
&= \int_\Omega \nabla \underline{v} : B \nabla \underline{w} d\underline{x},
\end{aligned}$$

where the last equality above the shorthand (A.3) was used.
For the time being, the weak formulation of (3.27) can be written as

$$\int_\Omega \frac{\partial \underline{c}}{\partial t}(\underline{w}) \cdot \underline{v} d\underline{x} + \int_\Omega \nabla \underline{v} : B \nabla \underline{w} d\underline{x} = \int_\Omega \underline{r} \cdot \underline{v} d\underline{x} \quad \forall \underline{v} \in \mathcal{V}^N, \tag{3.29}$$

where the specific structure of the term can be found explicitly above.
As stated in [35, p. 88f], we need to formulate an approximate system in order the weak formulation is slightly adapted in order to prove the existence of a weak solution to (3.29):

- One introduces a higher-order regularisation term $\varepsilon((-\Delta)^m + \mathrm{id})$ with $m > \frac{d}{2}$, $\varepsilon > 0$ and id denotes the identity operator. As Jüngel states, this is due to two reasons:
  - The operator $\underline{w} \mapsto \operatorname{div} B(\underline{w})\nabla \underline{w}$ is usually not uniformly elliptic but the regularized system arising by adding $\varepsilon((-\Delta)^m + \mathrm{id})$ can be proven to be uniformly elliptic w.r.t. an appropriate Sobolev norm.
  - The regularized equation can be solved in a more regular space $H^m(\Omega;\mathbb{R}^N)$, which has a continuous embedding into $L^\infty(\Omega;\mathbb{R}^N)$.[20] This is important as this ensures the boundedness of $w$ and thus also ensures the well-definedness of the inverse $\underline{c} = h^{-1}(\underline{w})$.
- The time derivative is discretized by an implicit Euler discretisation. This is used to avoid problems with regularity in the time variable and leads to an elliptic system to be solved in every timestep.

As we merely consider $1 \leq d \leq 3$, it is sufficient to introduce the regularisation term

$$\mathfrak{R}_\varepsilon(\underline{w},\underline{v}) = \varepsilon \int_\Omega \sum_{|\underline{\alpha}|=2} D^{\underline{\alpha}}\underline{w} \cdot D^{\underline{\alpha}}\underline{v} + \underline{w} \cdot v d\underline{x}.$$

[20] A definition of vector valued Lebesgue and Sobolev spaces can be found in definitions A.13 and A.14.

Thus the regularized version of (3.29) is given by

$$\int_\Omega \frac{\partial \underline{c}}{\partial t}(\underline{w}^\varepsilon)\cdot \underline{v} d\underline{x} + \int_\Omega \nabla \underline{v} : B\nabla \underline{w}^\varepsilon d\underline{x} + \mathfrak{R}_\varepsilon(\underline{w}^\varepsilon, \underline{v}) = \int_\Omega \underline{r}\cdot \underline{v} d\underline{x} \quad \forall \underline{v}\in \mathcal{V}^N.$$

By performing the discretisation of the time derivative by an implicit Euler discretisation (lowest order discretisation with discretisation error scaling with $\mathcal{O}(\tau)$), one obtains an approximate system, in which we search for $\underline{w}^{k+1}$ in an appropriate function space, such that for all test functions $\underline{v}\in\mathcal{V}^N$ there holds

$$\begin{aligned}\frac{1}{\tau}\int_\Omega \left(c(\underline{w}^{(\varepsilon,\tau),k+1}) - c(\underline{w}^{(\varepsilon,\tau),k})\right)\cdot \underline{v} d\underline{x} + \int_\Omega \nabla \underline{v} : B(w^{(\varepsilon,\tau),k+1})\nabla \underline{w}^{(\varepsilon,\tau),k+1} d\underline{x} \\ + \varepsilon \int_\Omega \sum_{|\underline{\alpha}|=2} D^\alpha \underline{w}^{(\varepsilon,\tau),k+1}\cdot D^\alpha \underline{v} + \underline{w}^{(\varepsilon,\tau),k+1}\cdot v dx = \int_\Omega \underline{r}(\underline{w}^{(\varepsilon,\tau),k+1})\cdot \underline{v} d\underline{x},\end{aligned} \tag{3.30}$$

where we use the shorthand notation $\underline{w}^{(\varepsilon,\tau),k} = \underline{w}^{(\varepsilon,\tau)}(\underline{x}, t^k)$ with $t^k = \tau k$ for $k\in\mathbb{N}$.

**Remark 3.4.** *To ease up notation, the $(\varepsilon,\tau)$-superscript in (3.30) will be dropped (thus denoting $\underline{w}^{(\varepsilon,\tau),k}$ by $\underline{w}^k$). However, one should keep in mind that the solution is dependent on the regularisation term which is dependent on $\varepsilon>0$ and the time discretisation step-size $\tau>0$. This convention will uphold until section 3.5.5 where the limit of $\underline{c}^{(\varepsilon,\tau)}\to\underline{c}$ (where $\underline{c}(\underline{w})$ is a weak solution to (3.29))will be discussed.*

The first step is to show the existence of a solution $\underline{w}^{k+1}$ of the approximate system (3.30) if at the previous timestep $t^k$ there exists a $\underline{w}^k\in L^\infty(\Omega,\mathbb{R}^N)$. This will be performed in section 3.5.2.

### 3.5.2 Step 2: Solution of a linearized version of the approximate system

**Lemma 3.6.** *Let the assumptions of Theorem 3.1 hold and let $\underline{w}^k\in L^\infty(\Omega;\mathbb{R}^N)$. Then there exists a weak solution $\underline{w}\in\mathcal{V}^N$ to (3.30).*

*Proof.* In this proof we wish to apply the Leray-Schauder fixed point theorem (cf. Theorem A.6) to prove the existence of a solution for the $k+1$-th timestep. Thus one has to construct a continuous and compact map $S : L^\infty(\Omega;\mathbb{R}^N)\times[0,1]\to L^\infty(\Omega;\mathbb{R}^N)$. For that purpose, we will define the bilinear form $a_{\underline{\hat{w}}}(\underline{w},\underline{v})$ and the linear form $F_{\underline{\hat{w}},\sigma}(\underline{v})$, such that for a fixed $\underline{\hat{w}}\in L^\infty(\Omega;\mathbb{R}^N)$ there exists $\underline{w}\in\mathcal{V}^N$ such that there holds

$$a_{\underline{\hat{w}}}(\underline{w},\underline{v}) = F_{\underline{\hat{w}},\sigma}(\underline{v}) \quad \forall \underline{v}\in\mathcal{V}^N, \tag{3.31}$$

where

$$a_{\underline{\hat{w}}}(\underline{w},\underline{v}) = \int_\Omega \nabla\underline{v} : B(\underline{\hat{w}})\nabla\underline{w} d\underline{x} + \varepsilon\int_\Omega \sum_{|\alpha|=2} D^\alpha\underline{w}\cdot D^\alpha\underline{v} + \underline{w}\cdot v dx \tag{3.32a}$$

$$F_{\underline{\hat{w}},\sigma}(\underline{v}) = -\frac{\sigma}{\tau}\int_\Omega \left(c(\underline{\hat{w}}) - c(\underline{w}^k)\right)\cdot\underline{v} d\underline{x} + \sigma\int_\Omega \underline{r}(\underline{c}(\underline{\hat{w}}))\cdot\underline{v} d\underline{x}. \tag{3.32b}$$

Note that be introducing $\underline{\hat{w}}$ we have linearized (3.30). Due to the structure of the variational problem (3.31), the standard technique is to use the Lax-Milgram lemma (cf. Lemma A.7) to prove the existence of an unique solution to the linearized problem. Due to Lemma 3.5 the elements of $B$ are bounded, thus $|b_{ij}(\underline{c}(\underline{\hat{w}}))| < C$ holds for $C > 0$ as given in (3.26) independently of $\underline{\hat{w}}$. Thus there holds by employing the Cauchy-Schwarz inequality (cf. Theorem A.12 with $p = q = 2$) multiple times in both the discrete and continuous version

$$\begin{aligned} a_{\underline{\hat{w}}}(\underline{w}, \underline{v}) &\leq C\|\nabla \underline{w} : \nabla \underline{v}\|_{L^1(\Omega)} \\ &\quad + \varepsilon \left( |\underline{w}|_{H^2(\Omega;\mathbb{R}^N)} |\underline{v}|_{H^2(\Omega;\mathbb{R}^N)} + \|\underline{w}\|_{L^2(\Omega;\mathbb{R}^N)} \|\underline{v}\|_{L^2(\Omega;\mathbb{R}^N)} \right) \\ &\leq C|\underline{w}|_{H^1(\Omega;\mathbb{R}^N)} |\underline{v}|_{H^1(\Omega;\mathbb{R}^N)} + \varepsilon \|\underline{w}\|_{H^2(\Omega;\mathbb{R}^N)} \|\underline{v}\|_{H^2(\Omega;\mathbb{R}^N)} \\ &\leq \max\{C, \varepsilon\} \|\underline{w}\|_{H^2(\Omega;\mathbb{R}^N)} \|\underline{v}\|_{H^2(\Omega;\mathbb{R}^N)} \end{aligned}$$

In order to prove the coercivity of $a_{\underline{\hat{w}}}(\cdot, \cdot)$, and we make use of the result from Lemma 3.5 that $B$ is positive definite there holds the estimate

$$a_{\underline{\hat{w}}}(\underline{w}, \underline{w}) = \int_\Omega \sum_{k=1}^{d} \begin{pmatrix} \frac{\partial w_1}{\partial x_k} \\ \vdots \\ \frac{\partial w_N}{\partial x_k} \end{pmatrix}^T B(\underline{\hat{w}}) \begin{pmatrix} \frac{\partial w_1}{\partial x_k} \\ \vdots \\ \frac{\partial w_N}{\partial x_k} \end{pmatrix} d\underline{x} + \varepsilon \int_\Omega \sum_{|\underline{\alpha}|=2} |D^{\underline{\alpha}} \underline{w}|^2 + |\underline{w}|^2 d\underline{x} > \varepsilon \int_\Omega \sum_{|\underline{\alpha}|=2} |D^{\underline{\alpha}} \underline{w}|^2 + |\underline{w}|^2 d\underline{x}.$$

Via the generalized Poincaré-inequality in Theorem A.8 (where we make use of assumption 3.1) one can show by choosing $p(u) = \|u\|_{L^2(\Omega)}$, that there holds[21]

$$\begin{aligned} \varepsilon \int_\Omega \sum_{k=1}^{N} \sum_{|\alpha|=2} (D^\alpha w_k)^2 + (w_k)^2 d\underline{x} &= \varepsilon \sum_{k=1}^{N} |w_k|^2_{H^2(\Omega)} + \|w_k\|^2_{L^2(\Omega)} \\ &\overset{(A.27)}{\geq} \frac{\varepsilon}{2} \sum_{k=1}^{N} (|w_k|_{H^2(\Omega)} + \|w_k\|_{L^2(\Omega)})^2 \\ &\overset{(A.25)}{\geq} \varepsilon \frac{1}{2C_p^2} \sum_{k=1}^{N} \|w_k\|^2_{H^2(\Omega)} \\ &= \varepsilon K_p \|\underline{w}\|^2_{H^2(\Omega;\mathbb{R}^N)}, \end{aligned}$$

thus $a_{\underline{\hat{w}}}(\cdot, \cdot)$ is coercive. For $F(\cdot)$, as $\underline{c} \in [0, 1]^N$ there holds by applying the Hölder inequality

$$\begin{aligned} F_{\underline{\hat{w}}, \sigma}(\underline{v}) &\leq \frac{\sigma}{\tau} \|\underline{c}(\underline{\hat{w}}) - \underline{c}(\underline{w}^k)\|_{L^2(\Omega;\mathbb{R}^N)} \|\underline{v}\|_{L^2(\Omega;\mathbb{R}^N)} + \sigma \|r\|_{L^2(\Omega;\mathbb{R}^N)} \|\underline{v}\|_{L^2(\Omega;\mathbb{R}^N)} \\ &\leq \frac{\sigma}{\tau} \operatorname{meas}(\Omega) N \|\underline{v}\|_{L^2(\Omega;\mathbb{R}^N)} + \sigma \|\underline{r}\|_{L^\infty(\Omega;\mathbb{R}^N)} \operatorname{meas}(\Omega) \|\underline{v}\|_{L^2(\Omega;\mathbb{R}^N)} \\ &\leq C(\Omega, \tau, N, \underline{r}) \|\underline{v}\|_{H^2(\Omega;\mathbb{R}^N)}, \end{aligned}$$

for some $0 < C(\Omega, \tau, N, \underline{r}) < \infty$, hence by Theorem A.7 there exists a unique solution $\underline{w}_{\sigma, \underline{\hat{w}}} \in \mathcal{V}^N$ to (3.31) for any given $\sigma \in [0, 1]$ and $\underline{\hat{w}} \in L^\infty(\Omega; \mathbb{R}^N)$. As the spacial dimension considered is $1 \leq d \leq 3$, there holds that the embedding $H^2(\Omega) \hookrightarrow L^\infty(\Omega)$ is compact (cf. definition A.17) by the Rellich-Kondrachov theorem (cf. Theorem A.9 part ii, as $m = 2$ and $p = 2$ and thus $mp > d$). This is indeed valid, as the composition $E = E_1 \circ E_2$ of the compact embedding $E_1 : H^2(\Omega) \hookrightarrow C^{0,\beta}(\overline{\Omega})$ for $0 < \beta < \frac{1}{2}$ as given by Theorem A.9 part ii. and the continuous

[21] It can be proven easily, that for a linear polynomial $p(\underline{x}) = k_0 + \sum_{i=1}^{d} k_i x_i$ for $\underline{k} \in \mathbb{R}^{d+1}$ there holds $\|u\|_{L^2(\Omega)} = 0 \Rightarrow u = 0$ $\lambda^d$-a.e., which is necessary for Theorem A.8 to hold.

embedding $E_2 : C^{0,\beta}(\overline{\Omega}) \hookrightarrow L^\infty(\Omega)$ yields a compact embedding, as the composition of a compact and a continuous map is compact. Hence by applying the theorem componentwise, there holds the compact embedding $H^2(\Omega;\mathbb{R}^N) \hookrightarrow L^\infty(\Omega;\mathbb{R}^N)$.[22]
We now wish to apply Theorem A.6 to $S$. For this application, we need make the choice of the sought operator $S$ more rigorous. By above result, we can define the operator $S_0 : L^\infty(\Omega,\mathbb{R}^N)\times[0,1] \to \mathcal{V}^N : S(\underline{\hat{w}},\sigma) \mapsto \underline{w}_{\sigma,\underline{\hat{w}}}$. By embedding $\mathcal{V}^N \hookrightarrow L^\infty(\Omega;\mathbb{R}^N)$ we can define the operator $S$ as stated at the beginning of the proof as composition $S = S_0 \circ I : L^\infty(\Omega;\mathbb{R}^N)\times[0,1] \to L^\infty(\Omega;\mathbb{R}^N)$ of the solution operator $S_0$ and the embedding operator $I := \mathrm{id}_{H^2(\Omega;\mathbb{R}^N)\to L^\infty(\Omega;\mathbb{R}^N)}$. The idea is that the composition of a compact (I) and continuous ($S_0$) map is compact, thus $S$ is a compact (and continuous) operator.[23]
As $\underline{w}(\underline{x},t) = \underline{0} \in \mathcal{V}^N$ is a solution for $\sigma = 0$ as $a(\underline{0},\underline{v}) = \underline{0}$ for all $\underline{v} \in \mathcal{V}^N$ and the Lax-Milgram lemma guarantees a unique solution, there holds $S(\underline{\hat{w}},0) = 0$. Thus the last and key component in order to apply Theorem A.6 is to show that for each element of the set $\{\underline{w} \in L^\infty(\Omega;\mathbb{R}^N) : S(\underline{w},\sigma) = \underline{w}\ \forall\sigma \in [0,1]\}$ there holds $\|\underline{w}\|_{L^\infty(\Omega;\mathbb{R}^N)} \leq C$, i.e. to find a $L^\infty$-bound for $\underline{w}$ independent of $\sigma$. Let $\underline{w} \in L^\infty(\Omega;\mathbb{R}^N)$ be a fixed point of $S$, then $\underline{w}$ solves (3.31) with $\underline{\hat{w}} = \underline{w}$. By choosing the specific test function $\underline{v} = \underline{w} \in \mathcal{V}^N$, there holds

$$\begin{aligned}\frac{\sigma}{\tau}\int_\Omega \left(c(\underline{w}) - c(\underline{w}^k)\right)\cdot \underline{w}d\underline{x} + \int_\Omega \left(\nabla\underline{w} : B(w)\nabla\underline{w} + \varepsilon\sum_{|\underline{\alpha}|=2}|D^{\underline{\alpha}}\underline{w}|^2 + |\underline{w}|^2\right)dx \\ = \sigma\int_\Omega \underline{r}(\underline{w})\cdot\underline{w}d\underline{x}\end{aligned} \tag{3.33}$$

The first term of the LHS can be estimated by using the definition of $\underline{w}$ via the convex[24] entropy density function $h$ in (3.6), hence by Taylor expansion there holds

$$h(\underline{c}) - h(\underline{\hat{c}}) \leq \nabla_{\underline{c}} h\cdot(\underline{c} - \underline{\hat{c}}) \quad \forall \underline{c},\underline{\hat{c}} \in (0,1)^N.$$

As $w = \nabla_{\underline{c}} h$ it can be inferred that

$$\frac{\sigma}{\tau}\int_\Omega \left(c(\underline{w}) - c(\underline{w}^k)\right)\cdot\underline{w}d\underline{x} \geq \frac{\sigma}{\tau}\int_\Omega \left(h(\underline{c}(\underline{w})) - h(\underline{c}(\underline{w}^k))\right)d\underline{x} \geq -\frac{1}{\tau}\,\mathrm{meas}(\Omega)(N+1). \tag{3.34}$$

By using the positive definiteness of $B$ due to Lemma 3.5, there holds

$$\int_\Omega \nabla\underline{w} : B(w)\nabla\underline{w}d\underline{x} = \int_\Omega \sum_{k=1}^{d}\begin{pmatrix}\frac{\partial w_1}{\partial x_k}\\ \vdots \\ \frac{\partial w_N}{\partial x_k}\end{pmatrix}^T B(w)\begin{pmatrix}\frac{\partial w_1}{\partial x_k}\\ \vdots \\ \frac{\partial w_N}{\partial x_k}\end{pmatrix}d\underline{x} \geq 0. \tag{3.35}$$

For the production rates $\underline{r} \in \mathbb{R}^N$ using $r_{N+1} = -\sum_{i=1}^N r_i$, there holds

$$\begin{aligned}\int_\Omega \underline{r}(\underline{c}(\underline{w}))\cdot\underline{w}d\underline{x} &= \int_\Omega \sum_{i=1}^{N} r_i(\underline{c}(\underline{w}))\left(\ln c_i(\underline{w}) - \ln(c_{N+1}(\underline{w}))\right)d\underline{x} = \int_\Omega \sum_{i=1}^{N+1} r_i(\underline{c}(\underline{w}))\ln c_i(\underline{w})d\underline{x} \\ &\leq \left\|\sum_{i=1}^{N+1} r_i(\underline{c}(\underline{w}))\ln c_i(\underline{w})d\underline{x}\right\|_{L^\infty(\Omega)}\mathrm{meas}(\Omega).\end{aligned}$$

[22] It should be noted that by applying Rellich-Kondrachov theorem we have also made use of assumption 3.1.
[23] For more details on the continuity of $S$, cf. [37].
[24] Note that we have proven that the Hessian of $h$ is positive definite in Lemma 3.4, thus $h$ is a convex function in $\underline{c}$.

By using assumption 3.4 the $L^\infty$-term can be bounded by a constant $C_r > 0$ and thus

$$\int_\Omega \underline{r}(\underline{c}(\underline{w})) \cdot \underline{w} d\underline{x} \leq C_r \operatorname{meas}(\Omega). \tag{3.36}$$

The generalized Poincaré-inequality (Theorem A.8) states that there exists a $K_p(\Omega) := \frac{1}{C_p^2} > 0$ such that

$$K_p \|\underline{w}\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq |\underline{w}|^2_{H^2(\Omega,\mathbb{R}^N)} + \|\underline{w}\|^2_{L^2(\Omega;\mathbb{R}^N)}. \tag{3.37}$$

By applying (3.34), (3.35), (3.36) and (3.37) to (3.33), there holds

$$\sigma \int_\Omega h(c(\underline{w}))d\underline{x} + K_p \varepsilon\tau \|\underline{w}\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq \sigma \int_\Omega h(\underline{c}(\underline{w}^k))d\underline{x} + \sigma\tau C_r \operatorname{meas}(\Omega).$$

By further using that $\sigma \in [0,1]$ and thus

$$\sigma K_p \varepsilon\tau \|\underline{w}\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq K_p \varepsilon\tau \|\underline{w}\|^2_{H^2(\Omega;\mathbb{R}^N)},$$

one can infer the estimate

$$\int_\Omega h(c(\underline{w}))d\underline{x} + K_p \varepsilon\tau \|\underline{w}\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq \int_\Omega h(\underline{c}(\underline{w}^k))d\underline{x} + \tau C_r \operatorname{meas}(\Omega). \tag{3.38}$$

As $\mathcal{H}[c]$ can be bounded from below by a negative constant by (3.34), this yields an uniform bound in $\sigma \in [0,1]$ such that $\|\underline{w}\|_{H^2(\Omega;\mathbb{R}^N)} \leq C(\varepsilon,\tau) < \infty$. By embedding $H^2(\Omega;\mathbb{R}^N) \hookrightarrow L^\infty(\Omega,\mathbb{R}^N)$ as seen before, there exists a $C > 0$ such that $\|\underline{w}\|_{L^\infty(\Omega;\mathbb{R}^N)} \leq C$. Thus by Theorem A.6, there exists a fixed point $\underline{w} \in L^\infty(\Omega;\mathbb{R}^N)$ of $S(\underline{w},1)$ and hence a solution for (3.30). As $\underline{w} \in H^2(\Omega;\mathbb{R}^N)$ by above estimate, there holds $\underline{w} \in \mathcal{V}^N$. □

### 3.5.3 Step 3: Entropy dissipation estimate

The next step in the proof of Theorem 3.1 is to show an estimate for the entropy dissipation term in (3.30). As the Matrix $B(\underline{w}^{k+1})$ is SPD, it can be diagonalized and has a positive smallest eigenvalue $\lambda_{\min} > 0$. By this argument, there holds

$$\int_\Omega \nabla \underline{w}^{k+1} : B(\underline{w}^{k+1}) \nabla \underline{w}^{k+1} d\underline{x} \geq \int_\Omega \lambda_{\min} \sum_{i=1}^N |\nabla \underline{w}_i^{k+1}|^2 d\underline{x}$$

However, as $B$ depends (even nonlinearly) on $\underline{w}^{k+1}$ (therefore also $\lambda_{\min} := \lambda_{\min}(B(\underline{w}^{k+1}))$), this does not yield a positive lower bound independent of $\underline{w}^{k+1}$, as this estimate is not uniform. However, Jüngel and Stelzer have shown that the following uniform estimate holds in the variables $\sqrt{c_i(\underline{w}^{k+1})}$:

**Lemma 3.7.** *Let $\underline{w}^{k+1} \in \mathcal{V}^N$ be a weak solution to* (3.30)*. Then there holds*

$$\int_\Omega \nabla \underline{w}^{k+1} : B(\underline{w}^{k+1}) \nabla \underline{w}^{k+1} d\underline{x} \geq \frac{4}{\Delta} \int_\Omega \sum_{i=1}^N |\nabla \sqrt{c_i(\underline{w}^{k+1})}|^2 d\underline{x}, \tag{3.39}$$

*where $\sqrt{\underline{c}} := (\sqrt{c_1}, \ldots, \sqrt{c_N})^T$ and $\Delta$ is defined as in Lemma 3.1.*

*Proof.* In this proof, we will employ the short-hand $\underline{c}^k = \underline{c}(\underline{w}(t_k, \underline{x})) \in \mathbb{R}^N$ and $\hat{\underline{c}}^k = (\underline{c}^k, c^k_{N+1}) \in \mathbb{R}^{N+1}$.[25] First, we need to prove that there holds

$$\int_\Omega \nabla \underline{w}^{k+1} : B(\underline{w}^{k+1}) \nabla \underline{w}^{k+1} d\underline{x} = \int_\Omega \nabla \ln(\hat{\underline{c}}^{k+1}) : (-\widetilde{A}^{-1}) \nabla \hat{\underline{c}}^{k+1} d\underline{x}, \tag{3.40}$$

with $\widetilde{A} \in \mathbb{R}^{(N+1)\times(N+1)}$ as defined in Lemma 3.2. For that purpose, one defines $z = (\underline{z}_1, \dots, \underline{z}_N)^T = B(w^{k+1}) \nabla w^{k+1} \in \mathbb{R}^{N\times d}$, $\underline{z}_{N+1} = -\sum_{i=1}^N \underline{z}_i$ and $\hat{z} = (z, \underline{z}_{N+1}) \in \mathbb{R}^{(N+1)\times d}$. By (3.6), there holds

$$\begin{aligned} \nabla w^{k+1} : B(w^{k+1}) \nabla w^{k+1} = \nabla w^{k+1} : \hat{z} &= \sum_{i=1}^N \left( \nabla \ln(c_i^{k+1}) - \nabla \ln(c_{N+1}^{k+1}) \right) \cdot \underline{z}_i \\ &= \sum_{i=1}^{N+1} \nabla \ln(c_i^{k+1}) \cdot \hat{\underline{z}}_i = \ln(\hat{\underline{c}}^{k+1}) : \hat{z}. \end{aligned} \tag{3.41}$$

By using $\nabla w^{k+1} = H \nabla_{\underline{x}} \underline{c}^{k+1}$ and the definition of $B = A_0^{-1} H^{-1}$ there holds $z = A_0^{-1} \nabla \underline{c}^k$. By inverting this relation one defines $\nabla c^k = A_0 z$. Thus $c_i^{k+1}$ can be expressed by using (1.8) as

$$\nabla c_i^k = (A_0 z)_{(i)} = \sum_{\substack{j=1 \\ j\neq i}}^N (d_{ij} - d_{i,N+1})(\underline{z}_i c_j^{k+1} - \underline{z}_j c_i^{k+1}) + d_{i,N+1} \underline{z}_i = -(Az)_{(i)} = -(\widetilde{A}\hat{z})_{(i)},$$

since each column $\underline{z}_i \in \mathrm{im}(A)$. As $\widetilde{A}\hat{z} \in \mathrm{im}(A)$, we have $(-\widetilde{A}z)_{N+1} = \sum_{i=1}^N (-\widetilde{A}z)_i = \nabla c_{N+1}^{k+1}$. Thus $\hat{\underline{c}}^{k+1} = -\widetilde{A}\hat{z}$ and hence $\hat{z} = (-\widetilde{A})^{-1} \nabla \hat{\underline{c}}^{k+1}$. By substituting the result for $\hat{z}$ into (3.41) there holds the claim (3.40). As seen in the proof for Lemma 3.2 one can infer that for the images of $\widetilde{A} = A|_{\mathrm{im}(A)}$ and $\widetilde{A}_S = A_S|_{\mathrm{im}(A_S)}$ there holds $\mathrm{im}(A) = \{\mathbb{1}\}^\perp$ and $\mathrm{im}(A_S) = \mathrm{span}\{\sqrt{\hat{\underline{c}}^{k+1}}\}^\perp = \{C^{\frac{1}{2}} \hat{\underline{x}} : \hat{\underline{x}} \in \mathrm{im}(A)\}$, where $C^{\pm\frac{1}{2}} := \mathrm{diag}\left( (c_1^{k+1})^{\pm\frac{1}{2}}, \dots, (c_{N+1}^{k+1})^{\pm\frac{1}{2}} \right) \in \mathbb{R}^{(N+1)\times(N+1)}$. As the definition $-A = C^{\frac{1}{2}}(-A_S)C^{-\frac{1}{2}}$ implies that $-\widetilde{A} = C^{\frac{1}{2}}(-\widetilde{A}_S)C^{-\frac{1}{2}}$, one can deduce $(-\widetilde{A}_S)^{-1} = C^{-\frac{1}{2}}(-\widetilde{A})^{-1}C^{\frac{1}{2}}$, and thus, as the smallest eigenvalue $\lambda_{\min}((-A_S)^{-1}) \geq \frac{1}{\Delta}$ by Lemma 3.2, there holds

$$\begin{aligned} \nabla \ln(\hat{\underline{c}}^{k+1}) : (-\widetilde{A})^{-1} \hat{\underline{c}}^{k+1} &= 4(\nabla \sqrt{\hat{\underline{c}}^{k+1}}) : C^{-\frac{1}{2}}(-\widetilde{A})^{-1} C^{\frac{1}{2}} \nabla \sqrt{\hat{\underline{c}}^{k+1}} \\ &= 4 \nabla \sqrt{\hat{\underline{c}}^{k+1}} : (-\widetilde{A}_S)^{-1} \nabla \sqrt{\hat{\underline{c}}^{k+1}} \geq \frac{4}{\Delta} \sum_{i=1}^N \left| \nabla \sqrt{c_i^{k+1}} \right|^2. \end{aligned}$$

We have thus found a positive uniform lower bound in $\underline{c}(\underline{w})$ of the entropy dissipation term. □

> **Remark 3.5.** *The result of Lemma 3.7 proves that assumption **H2'** from section 3.2 holds for $a = -\frac{1}{2}$.*

### 3.5.4 Step 4: Uniform estimates

In the next step, uniform estimates for the entropy are derived, which is stated in the following lemma:

[25] In order to avoid confusing typography in the superscripts, hatted variables are used instead of the usual convention in section 3, that primed variables denote the respective quantities in $\mathbb{R}^{N+1}$.

**Lemma 3.8 (Discrete Entropy inequality).** *Let $\underline{w}^{k+1} \in \mathcal{V}^N$ be a weak solution to (3.30). Then for $k \geq 0$ there holds*

$$\mathcal{H}[\underline{c}^k] + \frac{4\tau}{\Delta}\int_\Omega |\nabla\sqrt{\underline{c}^k}|^2 d\underline{x} + K\varepsilon\tau\|\underline{w}^{k+1}\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq \mathcal{H}[\underline{c}^{k-1}] + \widetilde{C}_r, \tag{3.42a}$$

*where the functional $\mathcal{H}[\cdot]$ is defined in (3.2) and $\widetilde{C}_r(\tau,\Omega) = C_r\tau\,\mathrm{meas}(\Omega)$, $K = \frac{1}{C_p^2}$ (see Theorem A.8) and $k \in \mathbb{N}^+$. By solving this estimate recursively there holds*

$$\mathcal{H}[\underline{c}^k] + \frac{4\tau}{\Delta}\sum_{j=1}^k\int_\Omega |\nabla\sqrt{\underline{c}^j}|^2 d\underline{x} + K\varepsilon\tau\sum_{j=1}^k\|\underline{w}^{k+1}\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq \mathcal{H}[\underline{c}^0] + k\widetilde{C}_r. \tag{3.42b}$$

*Proof.* Due to Lemma 3.6, there exists a solution $\underline{w}^{k+1} \in \mathcal{V}^N$ to (3.30). As (3.38) yields a uniform bound in $\sigma$ and (3.30) corresponds to the Leray-Schauder system we've constructed in the proof of Lemma 3.6 with $\sigma = 1$, (3.38) together with (3.39) implies (3.42a). By summing (3.42a) for $j = 1,\dots,k$ there holds

$$\sum_{j=1}^k\left(\mathcal{H}[\underline{w}^j] + \frac{4}{\Delta}\int_\Omega\sum_{i=1}^N|\nabla\sqrt{c^j_{(i)}}|^2 d\underline{x} + K\varepsilon\tau\|\underline{w}^j\|^2_{H^2(\Omega;\mathbb{R}^N)}\right) \leq \sum_{j=1}^k\mathcal{H}[\underline{w}^{j-1}] + k\tau C_r\,\mathrm{meas}(\Omega)$$

and thus by appropriately comparing the $\mathcal{H}[\underline{w}^j]$-terms on each side of the inequality there holds (3.42b). □

In this section, the original variables $\underline{c}$ are considered rather than the entropy variables as there are no uniform estimates in the $\underline{w}$-variables. The main argument is in section is now to define piecewise constant functions in time given as $\underline{w}^{(\tau)}(\underline{x},t) = \underline{w}^{k+1}(\underline{x})$ and $\underline{c}^{(\tau)}(\underline{x},t) = \underline{c}(\underline{w}^{k+1}(\underline{x}))$ for $x \in \Omega$, $t \in (k\tau,(k+1)\tau]$ and $k \in \mathbb{N}$. One introduces the discrete time derivative $D_\tau$ and the discrete time shift operator $\sigma_\tau$ via

$$D_\tau c^{(\tau)} = \frac{c^{(\tau)} - \sigma_\tau c^{(\tau)}}{\tau} = \frac{c^{(\tau)}(\underline{x},t) - c^{(\tau)}(\underline{x},t-\tau)}{\tau} \quad x \in \Omega, t \in (\tau,T].$$

By integrating (3.30) in the time variable and choosing piecewise constant-in-time test functions $\underline{\phi} : (0,T) \to \mathcal{V}^N$ there holds

$$\begin{aligned}&\frac{1}{\tau}\int_0^T\int_\Omega (c^{(\tau)} - \sigma_\tau c^{(\tau)})\cdot\underline{\phi}\,d\underline{x}dt + \int_0^T\int_\Omega \nabla\underline{\phi} : A_0^{-1}\nabla\underline{c}^{(\tau)}d\underline{x}dt\\ &+\varepsilon\int_0^T\int_\Omega\sum_{|\underline{\alpha}|=2} D^{\underline{\alpha}}\underline{w}^{(\tau)}\cdot D^{\underline{\alpha}}\underline{\phi} + \underline{w}^{(\tau)}\cdot\underline{\phi}\,d\underline{x}dt = \int_0^T\int_\Omega \underline{r}(\underline{c}^{(\tau)})\cdot\underline{\phi}\,d\underline{x}dt\end{aligned} \tag{3.43}$$

A density argument in [56, Prop. 1.36] shows that these piecewise constant-in-time functions are dense in $L^2(0,T;\mathcal{V}^N)$, thus (3.43) also holds for $\phi \in L^2(0,T;\mathcal{V}^N)$. On each of the discrete time intervals, one can show the following estimates:

**Lemma 3.9.** *There exists a constant $C > 0$ independent of $\varepsilon$, $\tau$ and $\eta$ (note that it is still assumed that $c_i \geq \eta > 0$ for $1 \leq i \leq N+1$ due to assumption 3.5) such that*

$$\|\sqrt{\underline{c}^{(\tau)}}\|_{L^2(0,T;H^1(\Omega;\mathbb{R}^N))} + \sqrt{\varepsilon}\|\underline{w}^{(\tau)}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} \leq C \tag{3.44a}$$

$$\|\underline{c}^{(\tau)}\|_{L^2(0,T;H^1(\Omega;\mathbb{R}^N))} + \|D_\tau \underline{c}^{(\tau)}\|_{L^2(0,T;(\mathcal{V}^N)')} \leq C \tag{3.44b}$$

*Proof.* Let us first note that for each function $f^{(\tau)}$ for $\tau \in \mathbb{R}^+$, which is piecewise constant on the tessellation $\mathcal{T}_\tau = \bigcup_{0\leq k\leq N_\tau}(k\tau,(k+1)\tau] \cup (N_\tau\tau, T]$ of $\Omega_T = [0,T]$ and $N_\tau := \lfloor \frac{T}{\tau} \rfloor \in \mathbb{N}$, we have an (exact) discrete quadrature rule

$$\int_0^T f^{(\tau)}(t)dt = \sum_{k=1}^{N_\tau} \tau f^\tau|_{(k\tau,(k+1)\tau]} + \tau^{-1}\lambda((N_\tau\tau, T])f_{N_\tau+1} =: \sum_{k=1}^{N_\tau+1} \sigma_k \tau f_k^\tau,$$

where $\sigma_k = 1$ for $0 \leq k \leq N_\tau$ and $\sigma_{N_\tau+1} = \tau^{-1}\lambda((N_\tau\tau, T])$. As $\sigma_k \in [0,1]$, we can bound it from above by 1, thus as we seek an upper bound we will wlog assume that $\tau N_\tau = T$ and the then trivial $N_\tau + 1$-st summand is omitted. Thus, by interpreting the sum over all $c^{(\tau)}$ for an appropriate $k = N_\tau$ as integration in the time domain, the discretization of (3.43) with the test function $\underline{\phi}|_{\mathcal{T}_j} = \underline{w}^j$ yields

$$\begin{aligned}
&\sum_{j=1}^{N_\tau} \int_\Omega (\underline{c}^{(\tau)}|_{\mathcal{T}_j} - \sigma_\tau \underline{c}^{(\tau)}) \cdot \underline{w}^j d\underline{x} + \tau \sum_{j=1}^{N_\tau} \int_\Omega \nabla \underline{w}^j : A_0^{-1}|_{\mathcal{T}_j} \nabla \underline{c}^{(\tau)}|_{\mathcal{T}_j} d\underline{x} \\
&+ \varepsilon\tau \sum_{j=1}^{N_\tau} \left( |\underline{w}^j|_{H^2(\Omega;\mathbb{R}^N)} + \|\underline{w}^j\|_{L^2(\Omega;\mathbb{R}^N)} \right) = \tau \sum_{j=1}^{N_\tau} \int_\Omega \underline{r}(\underline{c}^{(\tau)}|_{\mathcal{T}_j}) \cdot \underline{w}^j d\underline{x}.
\end{aligned}$$

Via the estimates (3.34), (3.35), (3.36) and (3.37) applied to the above, which leads to estimates as seen in (3.39) with $k = N_\tau$, we can infer

$$\begin{aligned}
&\mathcal{H}[c^{N_\tau}] + \frac{4\tau}{\Delta} \sum_{j=1}^{N_\tau} \sum_{i=1}^{N} \|\nabla \sqrt{c_{(i)}^j}\|^2_{L^2(\Omega;\mathbb{R}^d)} + \varepsilon\tau K_p \sum_{j=1}^{N_\tau} \|\underline{w}^j\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq \mathcal{H}[c^0] + \tau N_\tau \operatorname{meas}(\Omega) C_r \\
&\leq \mathcal{H}[c^0] + T \operatorname{meas}(\Omega) C_r =: C_1(T, \Omega, \underline{c}^0, C_r),
\end{aligned}$$

for a constant $C_1(T, \Omega, \underline{c}^0, C_r) > 0$ (and thus independent of $\varepsilon$ and $\tau$). Note that the estimate is bounded for any fixed $T$, but unbounded for $T \to \infty$. As we restrict our solution to be a member in the function space $L^2_{\text{loc}}(0,\infty;\cdot)$ in Theorem 3.1, this does not yield a problem however.
As $c_i \in (0,1)$ $\lambda^d$-a.e., there holds the estimate

$$\|(c_i^{(\tau)})^p\|_{L^\infty(0,T;L^\infty(\Omega))} \leq 1 \quad p \in \mathbb{R}^+. \tag{3.45}$$

By definition A.12, Hölder inequality and (3.45) there holds

$$\begin{aligned}
\|\sqrt{c_i^{(\tau)}}\|^2_{L^2(0,T;H^1(\Omega))} &= \|\sqrt{c_i^{(\tau)}}\|_{L^2(0,T;L^2(\Omega))} + \tau \sum_{k=1}^{N_\tau} \sum_{i=1}^{N} \|\nabla \sqrt{c_i^k}\|^2_{L^2(\Omega)} \\
&\leq T \operatorname{meas}(\Omega) + \frac{\Delta(d_{ij})}{4} C_1(T, \Omega, \underline{c}^0, C_r) =: C_2(T, \Omega, \underline{c}^0, C_r, d_{ij}),
\end{aligned}$$

as well as

$$\sqrt{\varepsilon}\|w_i^{(\tau)}\|^2_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} = \varepsilon\tau\sum_{j=1}^{N_\tau}\|\underline{w}^j\|^2_{H^2(\Omega;\mathbb{R}^N)} \leq \frac{1}{K_p(\Omega)}C_1(T,\Omega,\underline{c}^0,C_r) =: C_3(T,\Omega,\underline{c}^0,C_r),$$

for positive constants $C_2, C_3 \in \mathbb{R}^+$. Hence there exists a $C = \max\{C_2, C_3\} > 0$, such that (3.44a) holds.
For the uniform estimate of the time derivative (3.44b), let us first consider

$$\begin{aligned}\left\|\frac{\partial}{\partial x_l}c_i^k(\underline{x})\right\|^2_{L^2(\Omega)} &= 4\left\|\sqrt{c_i^k}\frac{\partial}{\partial x_l}\sqrt{c_i^k}\right\|^2_{L^2(\Omega)}\\ &\leq 4\left\|\frac{\partial}{\partial x_l}\sqrt{c_i^k}\right\|^2_{L^2(\Omega)}\left\|\sqrt{c_i^k}\right\|^2_{L^\infty(\Omega)} \leq 4\left\|\frac{\partial}{\partial x_l}\sqrt{c_i^k}\right\|^2_{L^2(\Omega)}\end{aligned}$$

By above estimate, one can compute the bound

$$\begin{aligned}\|c_i^{(\tau)}\|^2_{L^2(0,T;H^1(\Omega))} &= \|c_i^{(\tau)}\|^2_{L^2(0,T;L^2(\Omega))} + \tau\sum_{k=1}^{N_\tau}\sum_{i=1}^{N}\|\nabla c_i^k\|^2_{L^2(\Omega;\mathbb{R}^d)}\\ &\leq \|c_i^{(\tau)}\|^2_{L^2(0,T;L^2(\Omega))} + 4\tau\sum_{k=1}^{N_\tau}\sum_{i=1}^{N}\|\nabla\sqrt{c_i^k}\|^2_{L^2(\Omega;\mathbb{R}^d)}\\ &\leq T\,\mathrm{meas}(\Omega) + \Delta(d_{ij})C_1(T,\Omega,\underline{c}^0,C_r) =: \widetilde{C}_2(T,\Omega,\underline{c}^0,C_r,d_{ij}).\end{aligned}$$

Considering an arbitrary test function $\underline{\phi} \in L^2(0,T;H^2(\Omega;\mathbb{R}^N))$ and the embedding $L^2(\Omega) \hookrightarrow (H^1(\Omega))'$, starting from (3.43) and applying the Hölder inequality multiple times on the domain $Q_T = \Omega \times (0,T) \subset \mathbb{R}^{d+1}$ yields

$$\begin{aligned}&\frac{1}{\tau}\left|\int_0^T\int_\Omega(\underline{c}^{(\tau)} - \sigma_\tau\underline{c}^{(\tau)})\cdot\underline{\phi}\,d\underline{x}dt\right|\\ &\leq \|A_0^{-1}\|_{L^\infty(Q_T;\mathbb{R}^{N\times N})}\sum_{k=1}^{d}\|\partial_{x_k}\underline{c}\|_{L^2(Q_T;\mathbb{R}^N)}\|\partial_{x_k}\underline{\phi}\|_{L^2(Q_T;\mathbb{R}^N)}\\ &+ \varepsilon\|\underline{w}^{(\tau)}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))}\|\underline{\phi}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} + \|\underline{r}(\underline{c}^{(\tau)})\|_{L^2(Q_T;\mathbb{R}^N))}\|\underline{\phi}\|_{L^2(Q_T;\mathbb{R}^N)}\\ &\leq \left[C(N,d_{ij})\|\underline{c}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} + \sqrt{\varepsilon}\sqrt{\varepsilon}\|\underline{w}^{(\tau)}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} + r_\infty T\,\mathrm{meas}(\Omega)\right]\|\underline{\phi}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))}\\ &\leq \left[C(N,d_{ij}) + C_3(T,\Omega,\underline{c}^0,\underline{r}) + C_5(\underline{r},\Omega,T)\right]\|\underline{\phi}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))}\\ &= C_4(T,\Omega,\underline{c}^0,\underline{r},N,d_{ij})\|\underline{\phi}\|_{L^2(0,T;\mathcal{V}^N)},\end{aligned}$$

where $r_\infty$ denotes the maximum of $r$, which exists due to $r$ being a continuous function on a compact domain due to assumption 3.4. In above inequalities, (3.23) was used to bound the entries of $A_0^{-1}$. Furthermore, in order to use the bound of $\sqrt{\varepsilon}\|\underline{w}^{(\tau)}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} \leq C_3$ we've assumed $\varepsilon \leq 1$. As we want $\varepsilon \to 0$, this is not a too restrictive assumption. As of the above result, we have proven by definition of the dual norm, that $\|D_\tau c^{(\tau)}\|_{L^2(0,T,(\mathcal{V}^N)')} \leq C_4$, thus by choosing $C := \max\{\widetilde{C}_2, C_4\}$, there holds (3.44b). □

### 3.5.5 Step 5: Limits

**Remark 3.6.** *In this section, we perform both a limit in $(\tau, \varepsilon) \to 0$. For that purpose, we consider a subsequence of the arbitrary null sequence $\theta(n) = \frac{1}{n}$ and $\{\theta(n)\}_{n=1}^{\infty} \to 0$ such that $\tau_{n'} = \varepsilon_{n'} = \theta(n')$. To ease up notation, we drop superscripts not currently considered in the argument. For example, when considering the limit $\tau_{n'} \to 0$, $c^{(\tau_{n'})}$ denotes $c^{(\tau_{n'}, \varepsilon_{n'})}$.*

By the uniform estimates (3.44) from Lemma 3.9, one can now use a compactness result by Dreher and Jüngel [23] to the family of piecewise constant-in-time functions $c^{(\tau)}$. The Theorem is stated in Theorem A.10, in order to apply it, we must check its assumptions. We choose $B = L^p(\Omega; \mathbb{R}^N)$, $X = H^1(\Omega, \mathbb{R}^N)$ and $Y = (H^2(\Omega, \mathbb{R}^N))'$ (the admissible values for $p \in \mathbb{N}$ will be stated shortly).
Firstly, we want to apply Theorem A.9 part i., to prove the compact embedding $X \hookrightarrow B$, as $d = 3$, $m = 1$ and $p = 2$ in our case yields $mp \leq d$.[26] As Theorem A.9 part i. yields the restriction for $B = L^p(\Omega; \mathbb{R}^N)$ that $1 \leq p < \frac{dp}{d-mp}$, there holds $1 \leq p < 6$.
Secondly, in order to prove the continuous embedding $B \hookrightarrow Y$, the continuity of the embedding $L^p(\Omega; \mathbb{R}^N) \hookrightarrow (H^2(\Omega; \mathbb{R}^N))'$ can be argued as follows: As $H^2(\Omega; \mathbb{R}^N)$ is a Hilbert space, by Riesz representation theorem (cf. e.g. [25, p. 722]) there has to hold for each $\underline{\xi} \in (H^2(\Omega; \mathbb{R}^N))'$ that

$$\Xi := \left| \int_\Omega \underline{\xi} \cdot \underline{v} d\underline{x} \right| < \infty \quad \forall \underline{v} \in H^2(\Omega; \mathbb{R}^N).$$

Under the assumptions of Theorem A.12 there holds

$$\Xi \leq \|\underline{\xi}\|_{L^p(\Omega;\mathbb{R}^N)} \|\underline{v}\|_{L^q(\Omega;\mathbb{R}^N)}.$$

By [27, Cor. 7.11] as $d = 3$, there holds for each coordinate $v_k \in C_B^1(\Omega)$ where $C_B^m := \{u \in C^m(\Omega) : D^\alpha u \in L^\infty(\Omega) \wedge |\alpha| \leq m\}$ for $1 \leq k \leq N$ and $m \in \mathbb{N}$, thus $\|\underline{v}\|_{L^\infty(\Omega;\mathbb{R}^N)} < \infty$. By Lemma A.8 this implies that $\|\underline{v}\|_{L^q(\Omega;\mathbb{R}^N)} < \infty$ for $1 \leq q \leq \infty$, hence for $\underline{\xi} \in L^p(\Omega; \mathbb{R}^N)$ for $1 \leq p \leq \infty$, there holds $\Xi < \infty$, hence $L^p(\Omega; \mathbb{R}^N) \hookrightarrow (H^2(\Omega; \mathbb{R}^N))'$ is continuous.
Thirdly, by the Cauchy-Schwarz inequality and (3.44b) there holds

$$\|D_\tau \underline{c}^{(\tau)}\|_{L^1(0,T;(\mathcal{V}^N)')} = \int_0^T \|D_\tau \underline{c}^{(\tau)}\|_{(\mathcal{V}^N)'} 1 dt \leq \|1\|_{L^2([0,T])} \|D_\tau \underline{c}^{(\tau)}\|_{L^2(0,T;(\mathcal{V}^N)')} \leq TC.$$

We can thus apply Theorem A.10 part i., which yields as of (3.44b), that the sequence $\{c^{(\tau_j)}\}_{j=1}^{\infty}$ for $\lim_{j\to\infty} \tau_j = 0$ is relatively compact in $L^2(0, T; L^p(\Omega; \mathbb{R}^N))$ for admissible $p$. If one restricts to $p > 1$, then $L^2(0, T; L^p(\Omega; \mathbb{R}^N))$ is furthermore a reflexive Banach space. Hence one can now make use of the Eberlein-Šmuljan theorem (cf. Theorem A.11), which guarantees as $\|\underline{c}^{(\tau_j)}\|_{L^2(0,T;L^p(\Omega;\mathbb{R}^N))} \leq C$ by (3.44b) that there exists a weakly convergent subsequence $\underline{c}^{(\tau_{j'})} \rightharpoonup \underline{c}$ weakly in $L^2(0, T; L^p(\Omega; \mathbb{R}^N))$ for $1 < p < 6$. As relative compactness ensured by Theorem A.10 yields strong convergence (i.e. convergence in the norm) for a weakly convergent (sub)sequence, there holds that for a bounded sequence with $\lim_{j'\to\infty} \tau_{j'} \to 0$ that

$$c_i^{(\tau_{j'})} \to c_i \quad \text{strongly in } L^2(0, T; L^p(\Omega; \mathbb{R}^N)), 1 \leq i \leq N, 1 < p < 6. \tag{3.46}$$

As $c^{(\tau)}$ is bounded in $L^\infty(Q_T)$ (as argued in (3.45)), thanks to Lemma A.8 there also holds convergence $\lambda^{d+1}$-a.e. in $L^p(Q_T)$ for $1 < p \leq \infty$.

[26] Actually also for $d = 1, 2$ compact embedding could be proven by applying one of the other cases in Theorem A.9. Note that this yields different (more relaxed) restrictions on admissible $p$ as demonstrated in the proof here. However, as the most interest is surely for solutions in $\mathbb{R}^3$, we restrict the rest of the proof to $d = 3$.

By Lebesgue dominated convergence theorem (cf. Theorem A.13), as one can bound $A_0^{-1}$ as seen in (3.23) independently of $\underline{c}$ from above, there holds due to (3.46) that

$$A_0^{-1}(\underline{c}^{(\tau_{j'})}) \to A_0^{-1}(\underline{c}) \quad \text{strongly in } L^p(Q_T), p < \infty.$$

As $\underline{r}(\underline{c})$ is continuous due to assumption 3.4, one can also find bounds for Lebesgue dominated convergence theorem and thus

$$\underline{r}(\underline{c}^{(\tau_{j'})}) \to \underline{r}(\underline{c}) \quad \text{strongly in } L^p(Q_T), p < \infty.$$

By the boundedness shown in (3.44b) and Theorem A.11 there exist subsequences of $\{\tau_j\}_{j=1}^{\infty} \to 0$ such that for $1 \leq i \leq N$

$$\begin{aligned} D_\tau c_i^{(\tau_{j'})} &\rightharpoonup \frac{\partial}{\partial t} c_i \quad \text{weakly in } L^2(0,T,(\mathcal{V}^N)'), \\ \frac{\partial}{\partial x_l} c_i^{(\tau_{j'})} &\rightharpoonup \frac{\partial}{\partial x_l} c_i \quad \text{weakly in } L^2(0,T;L^2(\Omega)) \end{aligned}$$

By extension, there holds due to the composition of a weakly and a strongly convergent sequence

$$A_0^{-1}(\underline{c}^{(\tau_{j'})}) \frac{\partial}{\partial x_l} c_i^{(\tau_{j'})} \rightharpoonup A_0^{-1}(\underline{c}) \frac{\partial}{\partial x_l} c_i \quad \text{weakly in } L^2(0,T;L^2(\Omega)).$$

As $\varepsilon \to 0$ strongly and $\sqrt{\varepsilon}\|w^{(\tau)}\|_{L^2(0,T;H^2(\Omega;\mathbb{R}^N))} \leq C$ due to (3.44a), there holds

$$\varepsilon w^{(\tau_{j'})} \to 0 \quad \text{strongly in } L^2(0,T;H^2(\Omega;\mathbb{R}^N)).$$

This is sufficient to pass the limit $(\tau, \varepsilon) \to 0$ such that $\underline{c}^{(\tau,\varepsilon)}$ solving (3.43) converges to the solution $\underline{c}$ which solves the system

$$\int_0^T \langle \partial_t \underline{c}, \underline{v} \rangle \, dt + \int_0^T \int_\Omega \nabla \underline{v} : A_0^{-1}(\underline{c}) \nabla \underline{c} d\underline{x} dt = \int_0^T \int_\Omega \underline{r}(\underline{c}) \cdot \underline{v} d\underline{x} \quad \forall \underline{v} \in L^2(0,T;\mathcal{V}^N), \tag{3.47a}$$

which is equivalent to the strong formulation that there has to hold (understood distributionally) in $L^2(0,T;(\mathcal{V}^N)')$

$$\frac{\partial \underline{c}}{\partial t} - \operatorname{div}(A_0^{-1}(\underline{c}) \nabla \underline{c}) = \underline{r}(\underline{c}). \tag{3.47b}$$

There are a few subtleties left to show, which are e.g. discussed in [37] for more general cross-diffusion systems.
Firstly, by (3.44b) and $\varepsilon \to 0$ there holds $\partial_t \underline{c} \in L^2(0,T;(\mathcal{V}^N)')$ as the norm is bounded. By identifying $L^2(0,T;\cdot) = L^2_{\text{loc}}(0,\infty;\cdot)$, there holds the claim $\partial_t c_i \in L^2_{\text{loc}}(0,\infty;(\mathcal{V}^N)')$ as stated in Theorem 3.1. Via (3.44b) there also holds the claim from Theorem 3.1 that $c_i \in L^2_{\text{loc}}(0,\infty;H^1(\Omega))$. Secondly, as the homogeneous Neumann boundary conditions (1.1b) were built in the weak formulation as well as in the approximation space of all $\underline{w}_n$ obtained by the Leray-Schauder fixed point theorem, by the continuity of $c(\underline{w})$ they are satisfied by the limit.
Thirdly, due to (3.44b) and Theorem A.11 there exists a weakly convergent subsequence $c_i^{(\tau_{j'}),0} \rightharpoonup c_i^0$, such that the initial datum is satisfied in $\mathcal{V}'$. This is important, as theorem A.10 the estimate on the time derivative has only been required to be bounded on $t \in [\tau, T]$ and not $t \in [0,T]$. As we have actually shown the bounds up to $c^{(\tau_{j'}),0}$, we can choose the weakly convergent subsequence up that the previous timestep variable in the first timestep satisfies the initial datum in the $\mathcal{V}'$-sense.

The last point that needs discussion is the limit $\eta \to 0$ where $\eta$ was defined in assumption 3.5, to guarantee that $c_i \geq \eta > 0$ for $1 \leq i \leq N+1$.
As the entropy density (3.5) is defined for $c_i^0 = 0$ and finite, such that $\mathcal{H}[c_i^0]$ exists and is finite, all the bounds from Lemma 3.9 needed for the Eberlein-Šmuljan arguments in this section still hold in the limit $\eta \to 0$.[27] Furthermore by inspection of (1.8) the elements of $A_0$ are properly defined for for $c_i = 0$ or $c_i = 1$ for $1 \leq i \leq N+1$ and Lemma 3.3 guarantees that $A_0$ can be inverted.
By considering the slightly perturbed initial datum

$$c_i^{0,\alpha} = \frac{c_i^0 + \alpha}{1+\alpha} \geq \eta, \tag{3.48}$$

where $\alpha \in \mathbb{R}^+$ can be chosen to ensure assumption 3.5.
Then, by performing the limit $\eta \to 0$ (and hence $\alpha \to 0$), all non-vanishing limit objects are well-defined.
Hence we have shown all properties of Theorem 3.1, which thus yields the existence of a solution in any arbitrarily large but compact time interval.

[27] See the proof for Lemma 3.9 for details, where the bounds and their dependencies are derived.

# 4 Numerical Setup for solving the Maxwell-Stefan System

## 4.1 Introduction

For the purpose of numerically solving the Maxwell-Stefan equations, a new method to discretize the Maxwell-Stefan equations has been proposed by Jüngel, which makes use of several techniques employed by the boundedness-by-entropy method as presented in section 3. As stated before (cf. also [40]), the matrix $A^{-1}(\underline{c})$ in (1.9) may not be positive definite and also not necessarily symmetric.
By reformulating the system in entropy variables, as demonstrated in section 3.2, one obtains an equivalent system (3.11) with a matrix $B$ that is SPD, cf. Lemma 3.5.
Thus it is a reasonable endeavor to discretize the system in variables where the discretization has in general more favorable numerical features. The numerical discretization has been performed by using a (semi-)implicit Euler discretization, i.e. using a backwards Euler scheme, which uses the diffusion matrix $B$ from the previous timestep for simplicity. In the spatial domain, a lowest-order discretization with conforming P1-Lagrange elements via the Finite-Element method has been employed.
In this section, the mathematics behind the discretization is discussed further, which has been used to obtain Python code solving the Maxwell-Stefan equations in a novel entropy-variable discretization.

## 4.2 Solution strategy for the Maxwell-Stefan equations in entropy variables

In order to discretize the entropy formulation (3.11) with Finite Elements, we need to formulate the weak formulation of the system.
By multiplication with an arbitrary scalar test function $v \in V$ (where the appropriate function space $V$ has yet to be specified) and integration of (3.11) over the entire domain $\Omega$, one obtains

$$\int_\Omega \frac{\partial}{\partial t}\underline{c}(\underline{x},t)v(\underline{x})d\underline{x} - \int_\Omega \operatorname{div}\left[B(\underline{w}(\underline{x},t))\nabla \underline{w}(\underline{x},t)\right] v(\underline{x})d\underline{x} = \int_\Omega \underline{r}(\underline{c})v(x)d\underline{x} \quad \forall v \in V \tag{4.1}$$

For avoiding further complications with the non-linearities in $B(\underline{w})$, a semi-implicit Euler timestep discretisation has been chosen, i.e. we allow the matrix $B(\underline{w}(\underline{x},t))$ to depend on the previous timestep.
One obtains the discretized system in time at the $k$-th timestep via integration by parts (by exploiting the homogeneous Neumann boundary conditions from (1.1b)), which is given by

$$\begin{aligned}&\frac{1}{\Delta t}\int_\Omega [\underline{c}(\underline{w}(\underline{x},t^{k+1})) - \underline{c}(\underline{w}(\underline{x},t^k))]v(\underline{x})d\underline{x} + \int_\Omega B(\underline{w}(\underline{x},t^k))\nabla_{\underline{x}}\underline{w}(\underline{x},t^{k+1})\cdot\nabla_{\underline{x}}v(\underline{x})d\underline{x}\\ &= \int_\Omega \underline{r}(\underline{c})v(\underline{x})d\underline{x} \quad \forall v \in V.\end{aligned} \tag{4.2}$$

Due to (3.10), the system for the $j$-th component can be rewritten as

$$\begin{aligned}\int_\Omega \frac{e^{w_j(\underline{x},t^{k+1})}}{1+\sum_{i=1}^N e^{w_i(\underline{x},t^{k+1})}}v(\underline{x})d\underline{x} = &\int_\Omega \frac{e^{w_j(\underline{x},t^k)}}{1+\sum_{i=1}^N e^{w_i(\underline{x},t^k)}}v(\underline{x})d\underline{x}\\ &- \Delta t\int_\Omega B(\underline{w}(\underline{x},t^k))\nabla_{\underline{x}}w_j(\underline{x},t^{k+1})\cdot\nabla_{\underline{x}}v(\underline{x})d\underline{x}\\ &+ \Delta t\int_\Omega r_j(\underline{c}(\underline{w}(\underline{x},t^{k+1}))v(\underline{x})d\underline{x} \quad \forall v \in V.\end{aligned} \tag{4.3}$$

In order to solve this fully nonlinear coupled system in every timestep, we employ Newton's method, as e.g. proposed in [46, sec. 1.2.2].
The unknowns in (4.3) are the functions $\underline{w}(\underline{x}, t^{k+1})$ at the new timestep $t^{k+1}$. In our Finite Element setting, we represent the $\underline{w}(\underline{x}, t^{k+1})$ variables by a finite basis spanned by the test functions $\{\phi_1(\underline{x}), \ldots, \phi_{\text{NDof}}(\underline{x})\}$, where NDof denotes the number of degrees of freedoms (DOFs) on the tesselation of the domain $\Omega$. To perform the spatial discretization lowest order Lagrange elements have been chosen (although the code would accept arbitrary order Lagrange elements as well), i.e. conformal P1-elements, which linear local shape functions are determined by nodal evaluation on the vertex points (in 2D, in 1D simply at the start- and end-point of the tesselation intervals), see figure 1.

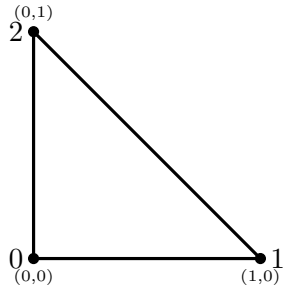



Figure 1: Degrees of freedom (DOFs) for the linear conforming $\mathbb{P}^1[\mathbb{R}^2]$ Lagrange element on the reference triangle $\hat{T}$. The linear functionals defining the local shape function are nodal evaluations on the vertex points of the triangle, uniquely determining the basis functions on the reference triangle.

The unknowns to solve for are the scalar basis function coefficients $\alpha_{j,1}^{k+1}, \ldots, \alpha_{j,\text{NDof}}^{k+1}$ associated with the vertex points of the tesselation and the gas components at the timestep $t^{k+1}$ where $k \geq 0$ and $1 \leq j \leq N$ (i.e. the components for the $N$ different gases). One obtains the representation of the entropy variables given by

$$w_j(\underline{x}, t^{k+1}) = \sum_{i=1}^{\text{NDof}} \alpha_{j,i}^{k+1} \phi_i(\underline{x}) \quad 1 \leq j \leq N \tag{4.4}$$

By substituting (4.4) into (4.3), we obtain for the $j$-th component of the system

$$\begin{aligned}
&\int_\Omega \frac{\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{k+1} \phi_l(\underline{x})\right]}{1 + \sum_{i=1}^{N} \exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{i,l}^{k+1} \phi_l(\underline{x})\right]} v(\underline{x}) d\underline{x} = \\
&\int_\Omega \frac{\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{k} \phi_l(\underline{x})\right]}{1 + \sum_{i=1}^{N} \exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{i,l}^{k} \phi_l(\underline{x})\right]} v(\underline{x}) d\underline{x} \\
&- \Delta t \int_\Omega B(\underline{w}(\underline{x}, t^k)) \left( \sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{k+1} \nabla_{\underline{x}} \phi_l(\underline{x}) \right) \cdot \nabla_{\underline{x}} v(\underline{x}) d\underline{x} \\
&+ \Delta t \int_\Omega r_j(\underline{c}(\underline{w}(\underline{x}, t^{k+1})) v(\underline{x}) d\underline{x} \quad \forall v \in V_h
\end{aligned} \tag{4.5}$$

It is important to note that above system (4.5) is in fact linear in terms of the test function $v(\underline{x}) = \sum_{j=1}^{\text{NDof}} \alpha_l \phi_l(\underline{x})$ for all arbitrary $\alpha_l \in \mathbb{R}$, therefore it is sufficient that (4.5) holds for all

basis elements $\phi_l$ and $1 \leq l \leq \text{NDof}$. Thus we obtain

$$\begin{aligned}
&\int_\Omega \frac{\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{k+1}\phi_l(\underline{x})\right]}{1+\sum_{i=1}^{N}\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{i,l}^{k+1}\phi_l(\underline{x})\right]}\phi_n(\underline{x})d\underline{x} = \\
&\int_\Omega \frac{\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{k}\phi_l(\underline{x})\right]}{1+\sum_{i=1}^{N}\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{i,l}^{k}\phi_l(\underline{x})\right]}\phi_n(\underline{x})d\underline{x} \\
&-\Delta t\int_\Omega B(\alpha_{1,1}^k,\dots,\alpha_{N,\text{NDof}}^k)\left(\sum_{l=1}^{\text{NDof}}\alpha_{j,l}^{k+1}\nabla_{\underline{x}}\phi_l(\underline{x})\right)\cdot\nabla_{\underline{x}}\phi_n(\underline{x})d\underline{x} \\
&+\Delta t\int_\Omega r_j(\underline{c}(\underline{w}(\underline{x},t^{k+1}))\phi_n(\underline{x})d\underline{x} \quad 1\leq n\leq \text{NDof}
\end{aligned} \tag{4.6}$$

At this level, we now employ Newton's method in order to solve this nonlinear system of equations for the new coefficients $\alpha_{j,l}^{k+1}$ and $1 \leq j < N$, $1 \leq l \leq \text{NDof}$. We want to find the roots of a set of (nonlinear) equations $R_i(x_1,\dots,x_n) = 0$ for $1 \leq i \leq N$ by

$$\sum_{j=1}^{m}\frac{\partial}{\partial x_j}R_i(x_1^{(\eta)},\dots,x_n^{(\eta)})\delta x_j = -R_i(x_1^{(\eta)},\dots,x_N^{(\eta)}) \quad i=1,\dots,N \tag{4.7a}$$

$$x_j^{(\eta+1)} = x_j^{(\eta)} + \kappa\delta x_j \quad j=1,\dots,N, \tag{4.7b}$$

where $\kappa \in (0,1]$ denotes the relaxation parameter, which can be set smaller than 1 in case if a damped Newton method is necessary in order to obtain convergence.
We therefore reformulate (4.6) by shifting all terms to the left-hand-side, thus obtaining the algebraic structure from (4.7a). As initial guess for the root, the coefficients from the previous time step can be used quite naturally, thus for the $k+1$-th timestep we set $\alpha_{j,l}^{(0)} := \alpha_{j,l}^k$.
One therefore obtains

$$\begin{aligned}
0 &\stackrel{!}{=} R_{\tau,j}(\alpha_{1,1}^{(\eta)},\dots,\alpha_{N,\text{NDof}}^{(\eta)}) \\
&= \int_\Omega \frac{\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{(\eta)}\phi_l(\underline{x})\right]}{1+\sum_{i=1}^{N}\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{i,l}^{(\eta)}\phi_l(\underline{x})\right]}\phi_\tau(\underline{x})d\underline{x} \\
&- \int_\Omega \frac{\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{j,l}^{(0)}\phi_l(\underline{x})\right]}{1+\sum_{i=1}^{N}\exp\left[\sum_{l=1}^{\text{NDof}} \alpha_{i,l}^{(0)}\phi_l(\underline{x})\right]}\phi_\tau(\underline{x})d\underline{x} \\
&+\Delta t\int_\Omega B(\alpha_{1,1}^{(0)},\dots,\alpha_{N,\text{NDof}}^{(0)})\left(\sum_{l=1}^{\text{NDof}}\alpha_{j,l}^{(\eta)}\nabla_{\underline{x}}\phi_l(\underline{x})\right)\cdot\nabla_{\underline{x}}\phi_\tau(\underline{x})d\underline{x} \\
&-\Delta t\int_\Omega r_j(\underline{c}(\underline{w}(\underline{x},t^{k+1}))\phi_\tau(\underline{x})d\underline{x}.
\end{aligned} \tag{4.8}$$

In a next step, we will make use of quadrature rules in order to evaluate the integrals over $\Omega$ in (4.8). We will employ Gauss quadrature[28], which will however not be exact due to the exponentials arising in (4.8). We can compute $R_{\tau,j}$ in the unknowns $\alpha_{j,l}^{(\eta)} \in \mathbb{R}$ at iteration step

[28] We consider Gauss quadrature with $N_q \in \mathbb{N}$ quadrature points $\hat{\underline{x}}_q$ $(1 \leq q \leq N_q)$ on the reference element $\hat{T}$ and quadrature weights $\omega_q$.

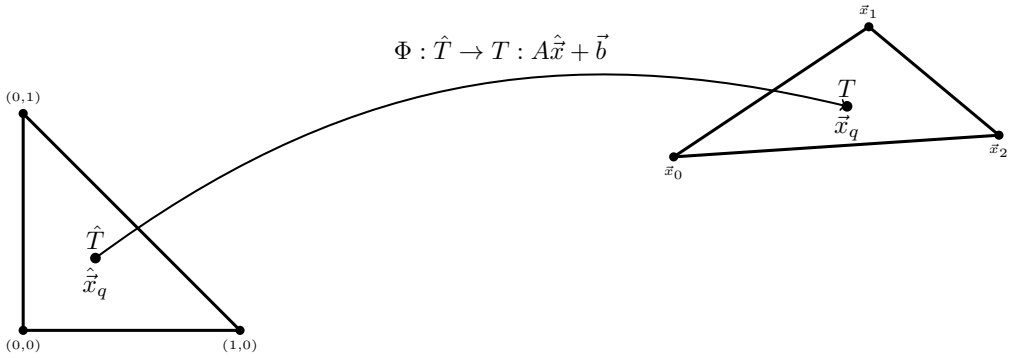



Figure 2: Pictographic Representation of the strategy for the finite element discretisation. One computes the values of the basis functions on the integration points $\underline{\hat{x}}_q$ on the reference element $\hat{T}$ and hence computes these values on an arbitrary element by using the affine diffeomorphism $\underline{x} = \Phi(\underline{\hat{x}}) = J_c\underline{\hat{x}} + \underline{b}$, which maps to an arbitrarily shaped triangle $T_c$ on the mesh. Here the centroid rule for numerical quadrature is depicted for simplicity.

$\eta \geq 0$, $\alpha_{j,l}^{(0)} := \alpha_{j,l}^{k}$ given from the previous timestep and decomposing the domain in cells $T_c$ via

$$
\begin{aligned}
&R_{\tau,j}(\alpha_{1,1}^{(\eta)}, \ldots, \alpha_{N,\mathrm{NDof}}^{(\eta)}) = \\
&\sum_{T_c \in \mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} \frac{\exp\left[\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}^{(\eta)} \phi_l(\underline{\hat{x}}_q)\right]}{1 + \sum_{i=1}^{N} \exp\left[\sum_{l=1}^{\mathrm{NDof}} \alpha_{i,l}^{(\eta)} \phi_l(\underline{\hat{x}}_q)\right]} \phi_\tau(\underline{\hat{x}}_q)\omega_q \\
&- \sum_{T_c \in \mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} \frac{\exp\left[\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}^{(0)} \phi_l(\underline{\hat{x}}_q)\right]}{1 + \sum_{i=1}^{N} \exp\left[\sum_{l=1}^{\mathrm{NDof}} \alpha_{i,l}^{(0)} \phi_l(\underline{\hat{x}}_q)\right]} \phi_\tau(\underline{\hat{x}}_q)\omega_q \\
&+ \Delta t \sum_{T_c \in \mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} B(\underline{\alpha}^{(0)}, \underline{\hat{x}}_q) \left( \sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}^{(\eta)} \nabla_{\underline{x}} \phi_l(\underline{\hat{x}}_q) \right) \cdot J_c^{-T} \nabla_{\underline{\hat{x}}} \phi_\tau(\underline{\hat{x}}_q)\omega_q \\
&- \Delta t \sum_{T_c \in \mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} r_j(\underline{c}(\underline{w}(\underline{\hat{x}}_q, t^{k+1}))\phi_\tau(\underline{\hat{x}}_q)\omega_q,
\end{aligned}
\tag{4.9}
$$

where $(J_c)_{ij} := \frac{\partial x_i}{\partial \hat{x}_j}$ denotes the Jacobian associated with the coordinate map from the reference element $\hat{T}$ to an arbitrary triangle $T_c$ on the mesh. For a triangle $T_c = \mathrm{conv}(\underline{x}_0, \underline{x}_1, \underline{x}_2)$, $J_c$ can be computed by $J_c = (\underline{x}_1 - \underline{x}_0, \underline{x}_2 - \underline{x}_0) \in \mathbb{R}^{2\times 2}$, cf. figure 2.
For the Jacobian of $R$ in terms of derivatives with respect to the unknowns $\alpha_{1,1}^{(\eta)}, \ldots, \alpha_{N,\mathrm{NDof}}^{(\eta)}$, one obtains

$$
\begin{aligned}
\frac{\partial R_{\tau,j}}{\partial \alpha_{m,n}} &= \int_\Omega \left(1 + \sum_{i=1}^{N} e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{i,l}\phi_l(\underline{x})}\right)^{-1} \left(\delta_{mj}\phi_n(\underline{x}) e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}\phi_l(\underline{x})}\right) \phi_\tau(\underline{x}) d\underline{x} \\
&- \int_\Omega \left(1 + \sum_{i=1}^{N} e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{i,l}\phi_l(\underline{x})}\right)^{-2} \left(\phi_n(\underline{x}) e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{m,l}\phi_l(\underline{x})}\right) \left(e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}\phi_l(\underline{x})}\right) \phi_\tau(\underline{x}) d\underline{x} \\
&+ \Delta t \int_\Omega \delta_{mj} \nabla_{\underline{x}} \phi_n(\underline{x}) \cdot \nabla_{\underline{x}} \phi_\tau(\underline{x}) d\underline{x}.
\end{aligned}
\tag{4.10}
$$

Again, we want to employ a more convenient formulation with quadrature and mapping back to a reference triangle $\hat{T}$ in order to compute $\phi(x)$ and $\nabla_{\underline{x}}\phi(x)$. By using arguments as before one obtains

$$\begin{aligned}\frac{\partial R_{\tau,j}}{\partial \alpha_{m,n}} &= \sum_{T_c\in\mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} \left(1+\sum_{j=1}^{N} e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}\phi_l(\hat{\underline{x}}_q)}\right)^{-1} \left(\delta_{mj}\phi_n(\hat{\underline{x}}_q) e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}\phi_l(\hat{\underline{x}}_q)}\right) \phi_\tau(\hat{\underline{x}}_q)\omega_q \\ &\quad - \sum_{T_c\in\mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} \left(1+\sum_{j=1}^{N} e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}\phi_l(\hat{\underline{x}}_q)}\right)^{-2} \left(\phi_n(\hat{\underline{x}}_q) e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{m,l}\phi_l(\hat{\underline{x}}_q)}\right) \cdot \\ &\quad \left(e^{\sum_{l=1}^{\mathrm{NDof}} \alpha_{j,l}\phi_l(\hat{\underline{x}}_q)}\right) \phi_\tau(\hat{\underline{x}}_q)\omega_q \\ &\quad + \Delta t \sum_{T_c\in\mathcal{T}} |\det(J_c)| \sum_{q=1}^{N_q} B(\underline{\alpha}^{(0)}, \hat{\underline{x}}_q)\delta_{mj} J_c^{-T}\nabla_{\hat{\underline{x}}}\phi_n(\hat{\underline{x}}_q)\cdot J_c^{-T}\nabla_{\hat{\underline{x}}}\phi_\tau(\hat{\underline{x}}_q)\omega_q.\end{aligned} \tag{4.11}$$

By means of above considerations, the strategy of solving the Maxwell-Stefan equations numerically in entropy variables can be summarized in high-level pseudo-code in algorithm 1.

**Data:** $c^k(\underline{x})$, Đ, $\Delta t$
**for** *j=1,...,*$n_t - 1$ **do**
  Compute $w^{k+j}(\underline{x})$ from $c^{k+j}(\underline{x})$ via (3.6).
  Compute $w^{k+j+1}(\underline{x})$ via Newton's method by passing Đ, $\Delta t, w^k$ (cf. algorithm 2).
  Compute $c^{k+j+1}(\underline{x})$ from $w^{k+j+1}(\underline{x})$ via (3.10).
**end**
**Result:** $c^{k+n_t}(\underline{x})$, $w^{k+n_t}(\underline{x})$

**Algorithm 1:** The solution strategy for solving the Maxwell-Stefan equations in pseudocode as used in `solve_maxwell_stefan` as seen in listing 3.

A more detailed overview of the implementation of Newton's method in the code and strategies

for damping the step-size in case of bad convergence properties can be studied in Algorithm 2.

**Data:** Ð, $\Delta t$, $w^k(\underline{x})$, $r$, tol=$1e-8$, $\beta = 0.8$
Set $w^{(\eta-1)} \leftarrow w^k$, i.e. use the old timestep as initial guess.
Assemble $\Xi(w^k, w^{(\eta-1)}, \Delta t, Ð)$ via (4.11).
Assemble $R(w^k, w^{(\eta-1)}, \Delta t, Ð, r)$ via (4.9).
Solve $\Xi \delta w^{(\eta)} = R$ for $\delta w^{(\eta)}$.
$w^{(\eta)} \leftarrow w^{(\eta-1)} + \delta w^{(\eta)}$.
$N_R \leftarrow \|R(w^k, w^{(\eta)}, \Delta t, Ð, r)\|_2$.
**while** $\max\left\{\left|\frac{\delta w^{(\eta)}}{w^{(\eta)}}\right|\right\} > tol \wedge N_R > N \cdot tol$ **do**
  Assemble $\Xi(w^k, w^{(\eta-1)}, \Delta t, Ð)$ via (4.11).
  Assemble $R(w^k, w^{(\eta-1)}, \Delta t, Ð, r)$ via (4.9).
  Solve $\Xi \delta w^{(\eta)} = R$ for $\delta w^{(\eta)}$.
  Update $w^{(\eta)} \leftarrow w^{(\eta-1)} + \delta w^{(\eta)}$.
  Compute $\widetilde{N}_R \leftarrow \|R(w^k, w^{(\eta)}, \Delta t, Ð, r)\|_2$.
  `/* Use damped Newton if residuum is worse than in the previous iteration. */`
  $n \leftarrow 0$
  **while** $\widetilde{N}_R > N_R$ **do**
    $n \leftarrow n + 1$
    Update $w^{(\eta)} \leftarrow w^{(\eta-1)} + \beta^n \delta w^{(\eta)}$.
    Update $\widetilde{N}_R \leftarrow \|R(w^k, w^{(\eta)}, \Delta t, Ð, r)\|_2$.
  **end**
  Update $N_R \leftarrow \widetilde{N}_R$.
**end**
Set $w^{k+1} \leftarrow w^{(\eta)}$.
**Result:** $w^{k+1}(\underline{x})$

**Algorithm 2:** Newton's method in pseudocode as used in `newton_solve` in listing 3.

It should be noted that when assembling the Jacobian $\Xi$ as described in Algorithm 2, the jacobian $\Xi$ as well as the residual vector $R$ need to be "flattened", i.e. the jacobian in (4.11) needs to be mapped to a matrix and the residual in (4.9) to a simple vector. This has been achieved such, that for $0 \leq i, k \leq N-1$ and $0 \leq j, l \leq \mathrm{NDof} - 1$ there holds

$$\Xi_{i\cdot\mathrm{NDof}+j,k\cdot\mathrm{NDof}+l} = \frac{\partial R_{i,j}}{\partial \alpha_{k,l}} \tag{4.12a}$$

$$R_{i\cdot\mathrm{NDof}+j} = R_{i,j}, \tag{4.12b}$$

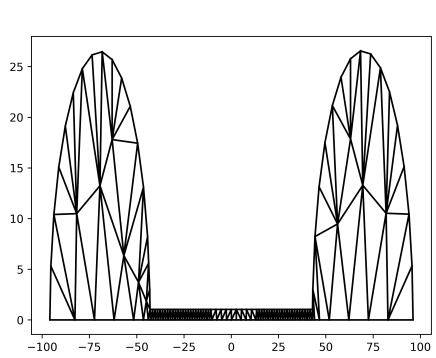
hence $\Xi \in \mathbb{R}^{(N\cdot\mathrm{NDof})\times(N\cdot\mathrm{NDof})}$ and $R \in \mathbb{R}^{N\cdot\mathrm{NDof}}$. In order to solve the linear system that arises in each step of the Newton's method until convergence, a direct linear solver[29] has been used, which can accept sparse matrices as input and is hence by far more memory efficient, as lots of entries equal to zero don't need to be stored.[30] As the geometries become more complex, the issue of preconditioning may become more and more prominent, which was however beyond the scope of the proof-of-concept implementation created for this thesis.

As the matrices may become large, one could also consider approximate solvers for the linear system, which may decrease the time needed for solving very high-dimensional systems. The Scipy sparse suite would e.g. support several well-established iterative solvers as GMRES or

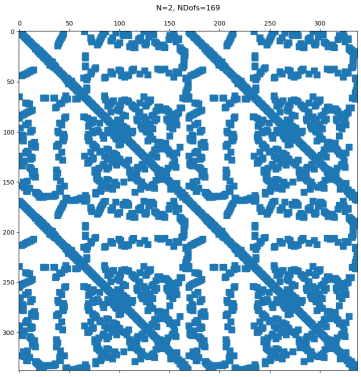
[29] Cf. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.spsolve.html` using UMFPACK by default.

[30] See figure 3, where the sparsity pattern of $\Xi$ is depicted for a model problem on two 2D-geometries.

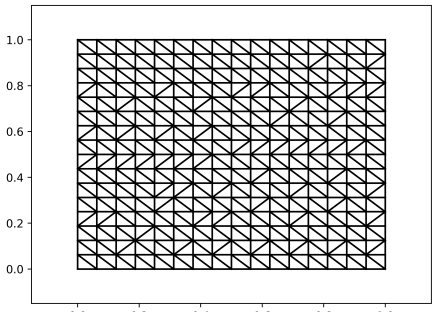
the CG-method, which could easily be patched into the code if needed.[31]
For extremely high dimensional problems, which may not even fit in the memory of one computational node alone, one should consider libraries like `petsc4py` provide very fast and scalable solvers that are able to parallelize the solution process even across several computational nodes using MPI, see e.g. [21].



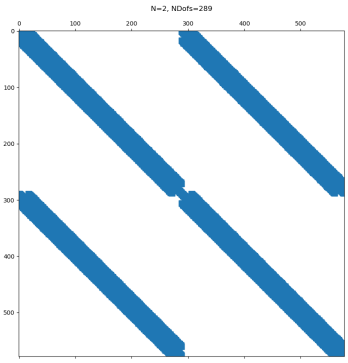

(a) 169 DOF complex triangulation




(b) 169 DOF: $\Xi$ sparsity pattern




(c) 289 DOF unit square triangulation




(d) 289 DOF: $\Xi$ sparsity pattern

Figure 3: The sparsity pattern (nonzero entries of the matrix are marked) of the Jacobian as assembled according to (4.11) and (4.12a) for a complex and a simple mesh pattern.

[31] For a complete list of possible sparse solvers, see e.g. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.html`.

## 4.3 More information on the finite element assembly

In order to produce performant Python code to solve the Maxwell-Stefan equations, various considerations had to be taken into account in order to undertake this task. However, it should be taken seriously that even a prolific computer scientist like Donald Knuth issues the following warning in [41]:

> The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

As a consequence, optimization should only be performed on the real bottlenecks of a given code, as every optimization normally impairs maintenability and portability across different platforms and hardware. A usual thumb's rule in software engineering is that according to the Pareto-principle, that in most applications 80% of the runtime is spent in 20% of the code (cf. [63, p.7]), which makes it worthwhile to further optimize the 20% as the most benefit can be gained from optimizing these sections.
In order to make an informed choice regarding the sections which are worthwhile optimizing, the usual approach is to gather data on the parts of the code, which make up most of the runtime. For this purpose, normally a profiler is used to prioritize the optimization efforts. For Python, there are various profilers available, cf. [29, ch. 2]. I have chosen to analyze the profiling data with the graphical tool `RunSnakeRun`[32], which provides a handy and quick overview of the performance hotspots from visualizing the raw data obtained by the widely used profiler `cProfile`.[33] In figure 4, en exemplary output of a run using the most naive implementation of assembling the Maxwell-Stefan jacobian and residual vector is depicted, which was sampled over computing 10 timesteps of the Duncan-Toor experiment as described in section 5.2. As can be deduced from figure 4, it is worthwhile to have a closer look at the assemblation of the jacobian, as the function makes up over 75% of the total runtime.
Hence the assemblation of the Jacobian $\Xi$ in Algorithm 2 has proven to be the performance hotspot of the code, as sparse matrix assemblation in Python is unfortunately not very fast in native Python, c.f. [20].
Some of the efforts in improving the performance of this code section to get decent performance are hence described in the following sections.

### 4.3.1 Make use of trivial concurrency when assembling Newton step

The first naive approach achieve a speed-up was to make use of trivial concurrency of the assemblation process, i.e. the independent assemblation of the jacobian $\Xi$ in (4.11) and the residual vector $R$ in (4.9). On modern multi-CPU systems (even most Laptops have 2 physical CPU cores by now), one can reduce the wall time needed for assembling the matrix and the vector by parallelizing these two as independent tasks with no data dependency whatsoever.[34] An implicit synchronization step is to wait until the result of both computations is available, as otherwise the linear system $\Xi\delta w^{(\eta)} = R$ can't be solved until both ingredients $\Xi \in \mathbb{R}^{(N\cdot\mathrm{NDof})\times(N\cdot\mathrm{NDof})}$ and $R \in \mathbb{R}^{N\cdot\mathrm{NDof}}$ are available. Even though the assemblation of $\Xi$ is a more complex task with a longer runtime $t_1[s]$ as the assemblation of $R$ taking $t_2[s]$, the overall wall time $t_W$ will approximately be $t_W = \max\{t_1, t_2\}$ instead of $t_W = t_1 + t_2$ in case of multiple cores being available on the system.

[32] Cf. `http://www.vrplumber.com/programming/runsnakerun`

[33] Cf. `https://docs.python.org/2/library/profile.html`

[34] Software developers experienced with parallel programming lingo would call this situation an "embarrassingly parallel problem".

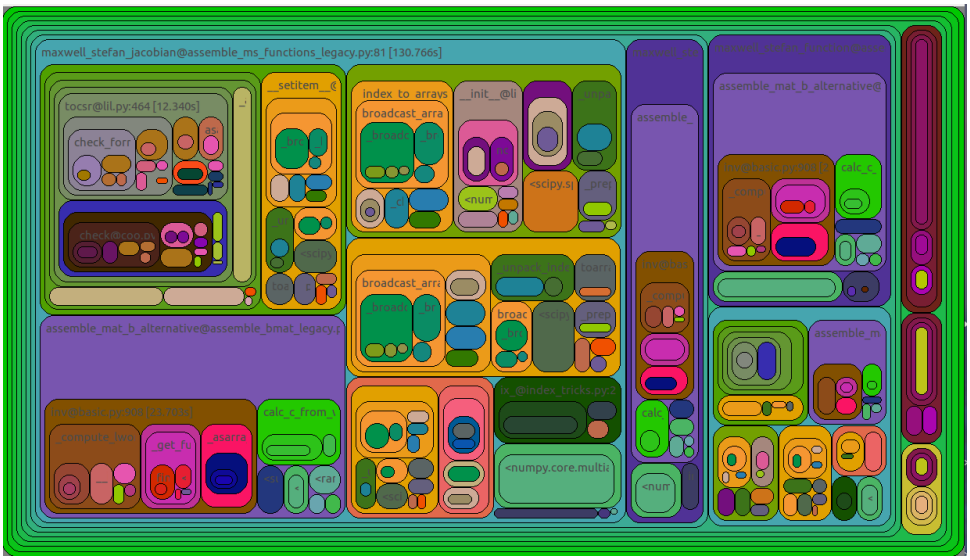



Figure 4: Graphical output of `RunSnakeRun` processing profiling data collected by `cProfile`. The block for the Maxwell-Stefan jacobian assembly is taking up over 75% of the runtime. This data was collected by performing 10 timesteps of the Duncan-Toor experiment, cf. section 5.2.

However, one has to be mindful that each parallelization does include some overhead for setting up processes/threads and communication, but as the job size is work-intensive enough as in this case, this still yields an overall performance gain by making better use of the multicore hardware at hand. In a typical example, assembling the jacobian took $2.1[s]$ while computing $R$ took $0.7[s]$, hence the overall performance gain was still reducing the runtime by 25% in each assemblation step.

When trying to perform parallel operations in Python, one has to be mindful of the concurrency limitations of the language due to the Global Interpreter Lock, in short GIL. In languages like C/C++, one normally uses the concept of threads, which are "light-weight" objects for the operating system (compared to processes), which allow to execute code in parallel inside a process. These threads can be forked off whenever needed and form "teams" of threads which can even itself fork off further teams of threads and join to the master thread after they have executed a parallelizable section of the code. When writing multi-threaded programs, one has to be very mindful that multiple threads accessing shared data do this such a "thread-safe" fashion, that the result of the program is deterministic and not dependent on which threads are accessing a shared resource at the same time. In Python[35], the GIL ensures that (C-)libraries which are not thread-safe can be run safely inside Python, by only allowing one thread at a time to be executed.

As a consequence, threading in Python is not really concurrent, as each Python thread (although being a full-weight POSIX thread in the OS) can only execute instructions when having acquired the GIL and release it after several "atomic" Python commands (which *loosely* map to Python interpreter instructions), allowing another thread to continue, cf. the presentation in [4]. There are some libraries like Numpy which allow to release the GIL during certain array operations relying on thread-safe C libraries[36], however most applications for native Python threads include I/O intensive programs (Python releases the GIL for certain I/O operations as well) like web-crawlers and are not performing computationally intensive tasks inside Python.

[35]This holds at least for the CPython interpreter, which is the most common Python interpreter. Other interpreters like Jython and IronPython do not have this limitation.

[36]Cf. `http://scipy-cookbook.readthedocs.io/items/ParallelProgramming.html`

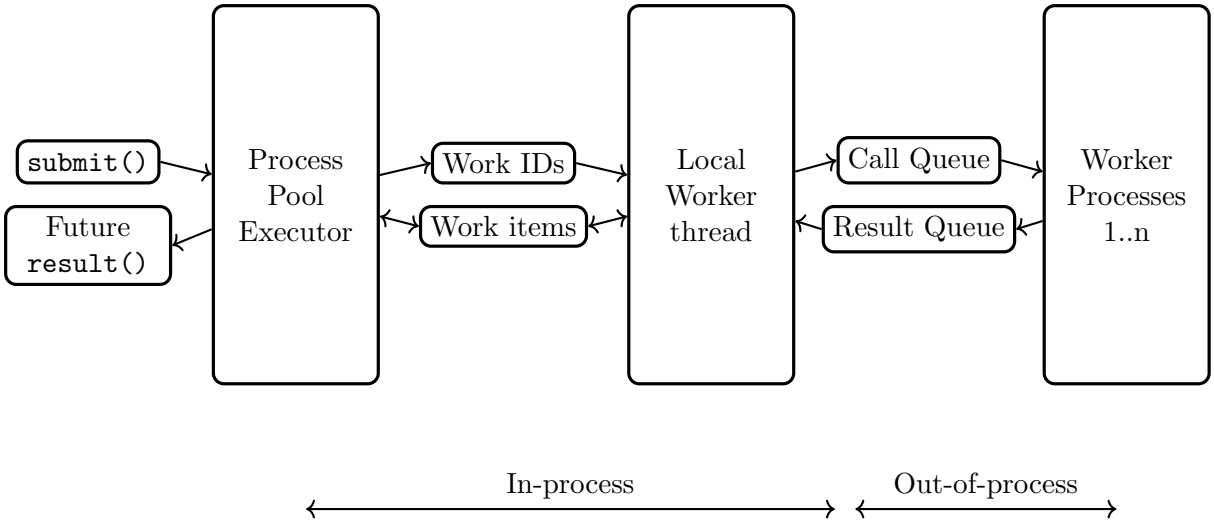



Figure 5: Pictographic representation of the internal structure of the `ProcessPoolExecutor` class, inspired by the module doc string in `https://github.com/agronholm/pythonfutures/blob/3.2.0/concurrent/futures/process.py`. Each job submitted to the executor gets a unique job ID which can then be requested by the `result()` method of `Future`-object returned upon submission.

The usual Python workaround for concurrency in scientific computing is to create several full-weight Python processes, where each of these processes has their own Python interpreter set up. These processes need to communicate their data via inter-process interfaces (e.g. via pipes), as different processes don't share the same address space in memory, which yields some overhead to native threading however. The OS can then attribute the available CPU resources to the different processes hence allow true multiprocessing, cf. [29].
The implementation of concurrently assembling the jacobian and the function has been performed with the `concurrent.futures` module, which provide the abstraction of a `ProcessPoolExecutor` and a `ThreadPoolExecutor`, which spawn a given number of Python processes/threads (in computer science lingo these are called "workers"), which can accept jobs, put them in a queue and submit them one after another to the available "workers", whenever they are ready to accept new jobs. The access to the `result()`-method of the futures returned by the executors upon job submission is a blocking call, which serves as a an implicit synchronization until the result of the computation is available. The implementation of this concurrent assemblation can be found in listing 3 in the functions `assemble_futures` (processes) and `assemble_futures_threading`.

### 4.3.2 Usage of COO-Matrix format in Scipy

A second approach to speed up the matrix computation was to use the COO-Matrix datatype for sparse matrices in Scipy[37], which was motivated by a discussion on Stackoverflow[38].
As stated before, sparse matrix datatypes have been employed as the jacobians for the Maxwell-Stefan system only have a limited amount of nontrivial elements. The native approach was to create a sparse matrix by specifying the dimensions and then looping over all intervals/triangles on the mesh and successively adding (possibly) non-trivial contributions to the jacobians.

[37] Cf. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html`
[38] Cf. `https://stackoverflow.com/questions/10522296/how-to-assemble-large-sparse-matrices-effectively-in-python-scipy`

However, it turned out that this was not the most efficient way to go in Python.
The COO-Matrix datatype chooses a different approach: This datatype accepts three arrays as input,

i. one representing the row position $i$,

ii. one the column position $j$,

iii. one representing the data to be placed on the position $(i, j)$, i.e. $\Xi_{ij}$.

While to COO-Matrix object is not suitable to be passed to sparse solvers in Scipy, it has an efficient `to_csr()` -method, which converts the COO-Matrix to a sparse matrix representation in the widely used CSR-format, which can indeed be used by the SciPy solvers for sparse linear systems. On each distinct index with a non-trivial value, there may be several contributions from different cells in the triangulation. Entries which are accessed multiple times hence need to be summed up later, which is performed by the `to_csr()`-method as demonstrated in the small interactive code snippet below.

```
>>> import scipy.sparse as sparse
>>> col_array = [2, 2, 0]
>>> row_array = [1, 1, 0]
>>> data = [1., 2., 3.]
>>> shape = (3, 3)
>>> coo = sparse.coo_matrix((data, (row_array, col_array)), shape=shape)
>>> print coo
  (1, 2)    1.0
  (1, 2)    2.0
  (0, 0)    3.0
>>> print coo.tocsr()
  (0, 0)    3.0
  (1, 2)    3.0
```

The `to_csr()`-method hence reduces multiple entries to the same index by adding them appropriately.
By the COO-matrix assembly approach, the runtime has been reduced to 33% of the original sparse matrix assembly, which is already a respectable speedup.
It should be noted that this could even be further improved, as due to the procedure of filling arrays with a distinct index and data for every cell and reducing multiple entries later on, one eliminates the data dependency for different cells contributing to certain entries of the jacobian. Hence one could parallelize the assembly by independently computing the contributions in different cells on multiple cores, while the final reduction of multiple entires to the same index is abstracted away in the `to_csr()`-method. At the time of the submission, unfortunately the parallel loop over all elements has not been running stable yet, as a major rewrite had been necessary to eliminate all Python object access in order to use native OpenMP multithreading inside C/Cython (see section 4.3.3), rendering the completion of this refactoring as a future task.

### 4.3.3 Usage of Cython to bridge high- and lowlevel language features

A third and final optimization I'd like to describe to more detail here involved the usage of the high-performance oriented extension Cython [5] of the Python language, which provides a

| | |
|---|---|
| Cython | 0.27.3 |
| numpy | 1.14.2 |
| scipy | 1.0 |
| sympy | 1.1.1 |
| triangle | 20170429 |

Table 1: The Python libraries for numerical computation used to produce the results as given in section 5. For a complete list of dependencies, consult the `requirements.txt` file in the repository.

compiler, which can create C code out of Python code, which can then be statically compiled to a library (resulting in a shared object `*.so` on Linux systems). Cython can also wrap C/C++ code to be accessible from Python, cf. [63]. It should be noted that Cython can also be used as a compiled Python-like programming language of its own, which is a superset of the Python language. By several extension keywords not available in plain Python, one can place optimization hints as for example data type annotations useful to the Cython compiler, which can then produce more efficient C code as output. Especially in scientific computing applications, one can easily use basic C data types like `int`, `double` etc., which can then be mapped from slow Python datatypes to its low-level equivalents in C in a very straightforward fashion.
I have hence placed several datatype annotations in the code using the `cdef`-keyword in order to optimize the existing Python code some more, resulting in very quick performance gains for yet reasonable refactoring efforts. The implementation finally obtained to create the results in section 5 could still be improved, as the main assemblation loop still requires frequent calls to the Python/C-API resulting in some call-overhead, as not all Numpy functions used in the code do have a native C interface yet. Nonetheless, using the Cython version of the assemblation process, the execution time for the assembly dropped down to 10% of the original implementation, which is a very satisfactory speedup. These efforts can be found in listing 4.

## 4.4 Setup of the code

For the purpose of solving the Maxwell-Stefan equations, a solver has been written in the programming language Python as part of this thesis, which relies heavily on backends written in the scientific libraries Scipy [34] and Numpy, both of which have several performant backends in C and Fortran.
The code has been tested to run under the Linux operating systems Ubuntu 16.4LTS (Debian derivate) and CentOS 7.4 (Red Hat derivate) with a CPython interpreter (Python 2.7.12) and under OpenSUSE 12.3 using an Anaconda 5.1 Python interpreter (Python 2.7.14).
The installation process is described to detail in the `README.md` manual file[39], hence I will only highlight one possible combination of steps to set up the code in Ubuntu shortly:

Listing 1: Installation script Ubuntu to be run in the main folder

```
1 #!/bin/sh
2 sudo apt install python-dev python-pip python-tk gcc
3 pip install --user -r requirements.txt
4 pip install ./ --user
```

The libraries necessary to reproduce all results given in section 5 are given in table 1.

[39]The code is hosted in a private repository on `https://bitbucket.org/gsmaster/mastertu_new`. Please send an E-mail to `g.simbrunner@gmail.com` for requesting access.

In order to ensure the correct installation on one's system, it is recommended to invoke `py.test` in the folder containing the `setup.py` installation script.
The results in sections 5.1.1 and 5.1.2 and 5.2 were computed on an Intel i7-5500U Broadwell CPU on the author's personal Lenovo X1 Laptop.

# 5 Numerical Results

In section 4, the procedure of discretizing the Maxwell-Stefan equations in entropy variables has been discussed, as well several aspects of implementing a performant solver in Python.
The scope is now to verify the success of this method to compute several model problems and to determine the accuracy of the method.
In order to undertake this task, the Method of Manufactured solutions (cf. section 5.1) has been carried out on the Maxwell-Stefan equation, to provide a new reliable benchmark for *any* Maxwell-Stefan equation solver implementation.
The results obtained by the code written for this purpose are discussed, as well as possible issues and caveats when discretizing the Maxwell-Stefan equations are presented.

## 5.1 Solver verification via manufactured solution

With the rising power of computer algebra systems (CAS), there have been quite interesting developments in solver verification procedures.
In this section I want to present the performance of the code written for this thesis by usage of the method of manufactured solution (MMS), which have become quite popular in engineering (cf. [52, 55, 57, 59]), as it presents a quite general framework to investigate the accuracy of a given solver even for complex nonlinear PDEs.
The general idea is quite simple: If one has obtained a solver which is able to integrate an arbitrary source term $f$, one can measure the quality of the solution produced by the code on an artificial manufactured problem, where one can construct an analytic solution and thus compare the numerical solution to the analytical one.
Suppose starting with a parabolic PDE with Neumann boundary conditions

$$u_t(\underline{x},t) + \mathcal{L}u(\underline{x},t) = f(\underline{x},t) \quad \text{in } \Omega, t>0 \tag{5.1a}$$
$$u(\cdot,0) = u_0(\cdot) \quad \text{in } \Omega, t=0 \tag{5.1b}$$
$$\nabla_{\underline{x}} u(\underline{x},t)\cdot\nu = g(\underline{x},t) \quad \text{on } \partial\Omega\times(0,\infty), \tag{5.1c}$$

where $\mathcal{L}$ denotes a (possibly nonlinear) elliptic operator acting on the spatial $\underline{x}$-variables. The idea is to use a sufficiently regular analytic ansatz function $\phi(\underline{x},t)$ satisfying the restraints

$$\phi(\underline{x},0) = u_0(\underline{x}) \quad \text{in } \Omega, t=0 \tag{5.2a}$$
$$\nabla_{\underline{x}}\phi(\underline{x},t)\cdot\nu = g(\underline{x},t) \quad \text{on } \partial\Omega\times(0,\infty) \tag{5.2b}$$

As $\phi(\underline{x},t)$ satisfies the boundary condition (5.2b) as well as the initial condition (5.2a), we can solve a modified problem of (5.1) which is exactly the same except for the source term $f(\underline{x},t)$, which is directly computed by the action of the differential operators $\mathcal{L}$ and $\frac{\partial}{\partial t}$ on the given "solution function" $\phi(\underline{x},t)$, thus yielding the right hand side $\widetilde{f}(\underline{x},t)$:

$$\widetilde{f}(\underline{x},t) := \frac{\partial\phi}{\partial t}(\underline{x},t) + \mathcal{L}\phi(\underline{x},t) \tag{5.3}$$

For the Maxwell-Stefan problem with homogeneous Neumann boundary conditions, one can choose the following ansatz function to be a manufactured analytical solution:

$$\phi(\underline{x},t) := \frac{1}{2(N+1)}\left(1 + e^{-kt}\prod_{i=1}^{d}\cos^2(\pi x_i)\right) \quad k>0 \tag{5.4}$$

One can easily compute that the solution function in (5.4) has been chosen such that it satisfies the homogeneous Neumann BCs as given in (1.1b) on the boundary $\partial\Omega$ of the $d$-dimensional cube domain $\Omega = (0,1)^d$.
The constant $\frac{1}{2(N+1)}$ in (5.4) has been chosen such that $\phi \in \left[\frac{1}{2(N+1)}, \frac{1}{N+1}\right]$, therefore we can ensure that all solution components are positive everywhere on the domain.
The initial condition for the numerical solver is simply given by $\phi(x,0)$, i.e.

$$\phi_0(\underline{x}) = \phi(\underline{x}, 0) = \frac{1}{2(N+1)} \left(1 + \prod_{i=1}^{d} \cos^2(\pi x_i)\right). \tag{5.5}$$

In [13, sec. 4], a simpler representation for the ternary Maxwell-Stefan system is given under the additional assumption $Đ_{12} = Đ_{13}$, where it is shown that the equations reduce to a heat equation for the first component and a drift-diffusion type equation in the second component. As demonstrated by Boudin there, there holds for

$$\beta := \frac{1}{Đ_{12}} - \frac{1}{Đ_{23}} \tag{5.6}$$

that the Maxwell-Stefan equations reduce to

$$r_1(\underline{x}, t) = \frac{\partial c_1}{\partial t} - Đ_{12}\Delta_{\underline{x}} c_1 \tag{5.7a}$$

$$r_2(\underline{x}, t) = \frac{\partial c_2}{\partial t} - \nabla_{\underline{x}} \cdot \left[\left(\frac{1}{Đ_{23}} + \beta c_1\right)^{-1} (\nabla_{\underline{x}} c_2 + \beta Đ_{12} c_2 \nabla_{\underline{x}} c_1)\right]. \tag{5.7b}$$

From this representation, one can compute for the ansatz functions $c_1(\underline{x}, t) = c_2(\underline{x}, t) = \phi(\underline{x}, t)$ the respective source terms obtained by the MMS ansatz, i.e. $\widetilde{r}_i(\underline{x}, t)$, by substituting (5.4) into (5.7). Hence one obtains a problem for which an analytic solution is known for $c_1$ and $c_2$ (and by using the identity $c_3 = 1 - c_1 - c_2$ also for $c_3$), which has similar numeric features as computing the problem with a different source term, where in general no analytical solution is known.
In order to have a diffusion matrix with nearly realistic Maxwell-Stefan diffusion coefficients present, the Maxwell-Stefan diffusion components from [13, Sec. 2] for $H_2 - N_2$ and $N_2 - CO_2$ interaction are used in the numerical experiments in sections 5.1.1 and 5.1.2. Note that the coefficient $Đ_{23}$, i.e. the diffusion coefficient for $H_2$-$CO_2$-interaction, is set to the value of $Đ_{13}$ (the diffusion coefficient for $H_2$-$N_2$-interaction) to make use of the simplification leading to (5.7).[40]

$$(Đ)_{ij} = \begin{pmatrix} 0 & 0.833 & 0.833 \\ 0.833 & 0 & 0.168 \\ 0.833 & 0.168 & 0 \end{pmatrix} \tag{5.8}$$

We also want to study the experimental convergence rate w.r.t. to the timestep and the spacial and temporal discretization. For that purpose one can define the error

$$e_h = \|\phi(t) - \phi_h(t)\|_{L^2(\Omega)} \tag{5.9}$$

[40] Note that the entries in (5.8) are actually scaled to $cm^2 s^{-1}$ to adapt to the smaller geometry compared to the setting considered in [13, Sec. 2], which in the MMS cases considered here is a unit interval/square.

at a given time $t$ and a spacial discretization constant $h \sim \mathcal{O}(\Delta x)$. If one can assume that the discretization error satisfies $e_h \leq Ch^p$ for some $p \in \mathbb{N}$ and $C > 0$, there should hold

$$\frac{e_{h_1}}{e_{h_2}} = \frac{Ch_1^p}{Ch_2^p}.$$

and therefore the experimental convergence rate $p$ is obtained by

$$p_{h_1 \to h_2} = \frac{\ln\left(\frac{e_{h_1}}{e_{h_2}}\right)}{\ln\left(\frac{h_1}{h_2}\right)}. \tag{5.10a}$$

If one assumes that $h \sim \Delta x \sim \mathcal{O}(N^{-1})$ with $N \in \mathbb{N}$ being the number of equidistant grid points in one spacial direction, (5.10a) reduces by substituting $N_i^{-1} = h_i$ for $i \in \{1, 2\}$ to

$$p_{N_1 \to N_2} = -\frac{\ln\left(\frac{e_{N_1}}{e_{N_2}}\right)}{\ln\left(\frac{N_1}{N_2}\right)}. \tag{5.10b}$$

### 5.1.1 1D case for manifactured solutions

In this section, the performance of the Maxwell-Stefan solver is evaluated by using the MMS method on the 1D-domain $\Omega = (0, 1)$. It is simple to verify that the homogeneous Neumann boundary conditions (1.1b) are satisfied by (5.4) and $d = 1$.
The RHS computed by the computer algebra system SymPy [49] (cf. the code in listing 2, which was cross-checked with output from Mathematica) yields the source terms

$$r_1(x,t) = \frac{1}{6}\left(2\pi^2 Đ_{12}\cos(2\pi x) - k\cos^2(\pi x)\right)e^{-kt} \tag{5.11a}$$

$$\begin{aligned}
r_2(x,t) = &-\frac{\frac{1}{6}e^{-kt}}{(Đ_{12}e^{kt} - \frac{1}{6}(Đ_{12} - Đ_{23})(e^{kt} + \cos^2(\pi x)))^2} \\
&\Bigg(\frac{\pi^2}{2}Đ_{12}(\frac{4}{6}(Đ_{12} - Đ_{23})(Đ_{23}e^{kt} \\
&- \frac{1}{6}(Đ_{12} - Đ_{23})(e^{kt} + \cos^2(\pi x)))\sin(\pi x)\sin(2\pi x)\cos(\pi x) \\
&+ (Đ_{12}e^{kt} - \frac{1}{6}(Đ_{12} - Đ_{23})(e^{kt} + \cos^2(\pi x)))(\frac{1}{6}(Đ_{12} - Đ_{23})(\cos(4\pi x) - 1) \\
&- 4(Đ_{23}e^{kt} - \frac{1}{6}(Đ_{12} - Đ_{23})(e^{kt} + \cos^2(\pi x)))\cos(2\pi x))) + k(Đ_{12}e^{kt} \\
&- \frac{1}{6}(Đ_{12} - Đ_{23})(e^{kt} + \cos^2(\pi x)))^2\cos^2(\pi x)\Bigg),
\end{aligned} \tag{5.11b}$$

such that (5.4) is an analytical solution to (1.9).
In figures 6 and 7 one can study quite closely the interplay of the timestep size $\Delta t$ and the uniform spacial mesh cell size $\Delta x$. If one considers the experimental convergence rates for increasing the mesh resolution at given timestep sizes in this section, we are presented with the at first glance counterintuitive fact that too fine mesh discretizations for a coarse timestep size can even worsen the convergence properties (manifesting in negative convergence rates for increasing the resolution, thus the error increases even though a finer mesh is computed).
In order to understand this behavior, the importance of the CFL number as a measure for the stability of the numerical scheme needs to be considered. Let the CFL number be denoted by

$C_{\Delta x,\Delta t}$, which is defined by

$$C_{\Delta x,\Delta t} = u\frac{\Delta t}{\Delta x}. \tag{5.12}$$

In (5.12) $u$ is a characteristic velocity determined by the features of the problem at hand. Keeping the dependency on $u$ aside, which is a problem dependent constant, there holds for the CFL number $C_{\Delta x,\Delta t} \gg 1$ if $\Delta x \ll \Delta t$. This is somewhat unexpected, since we use an implicit method[41], i.e. backwards Euler timestepping, which as an A-stable numerical scheme ensures stability also for $C_{\Delta x,\Delta_t} > 1$. As it seems, we are still bound to the fact that we make use of the diffusion matrix from the previous timestep, hence an explicit part is included in the timestepping scheme as well, which may be responsible for this behavior. Hence we need to be extra mindful when considering the analytical errors for different timesteps and spacial mesh resolutions, that a coupling of timestep size and spacial resolution is present and needs to be taken into account for low-error approximations.
In tables 2, 3 and 4, the experimental convergence rates for the first component $c_1$ can be found , where the spacial resolution is increased while the timestep is kept constant. From these tables it is quite obvious that for coarse timestep sizes, increasing the mesh resolution only decreases the error up to a point, above which finer mesh resolutions may actually *increase* the error. This behavior can be attributed to the CFL number growing due to $\Delta x$ being several orders of magnitude smaller than $\Delta t$ and hence a loss of stability in the timestepping scheme due to the semi-implicit discretization is present.
The same qualitative behavior also holds for the second component $c_2$, which shows an even larger dependency of an appropriately chosen timestep for finer mesh resolutions. The tables depicting the experimental convergence rates for different timestep sizes when increasing the mesh resolution can be found in tables 5, 6 and 7.

[41] Actually a semi-implicit method is employed, which uses the matrix $B$ from the old timestep, cf. section 4.

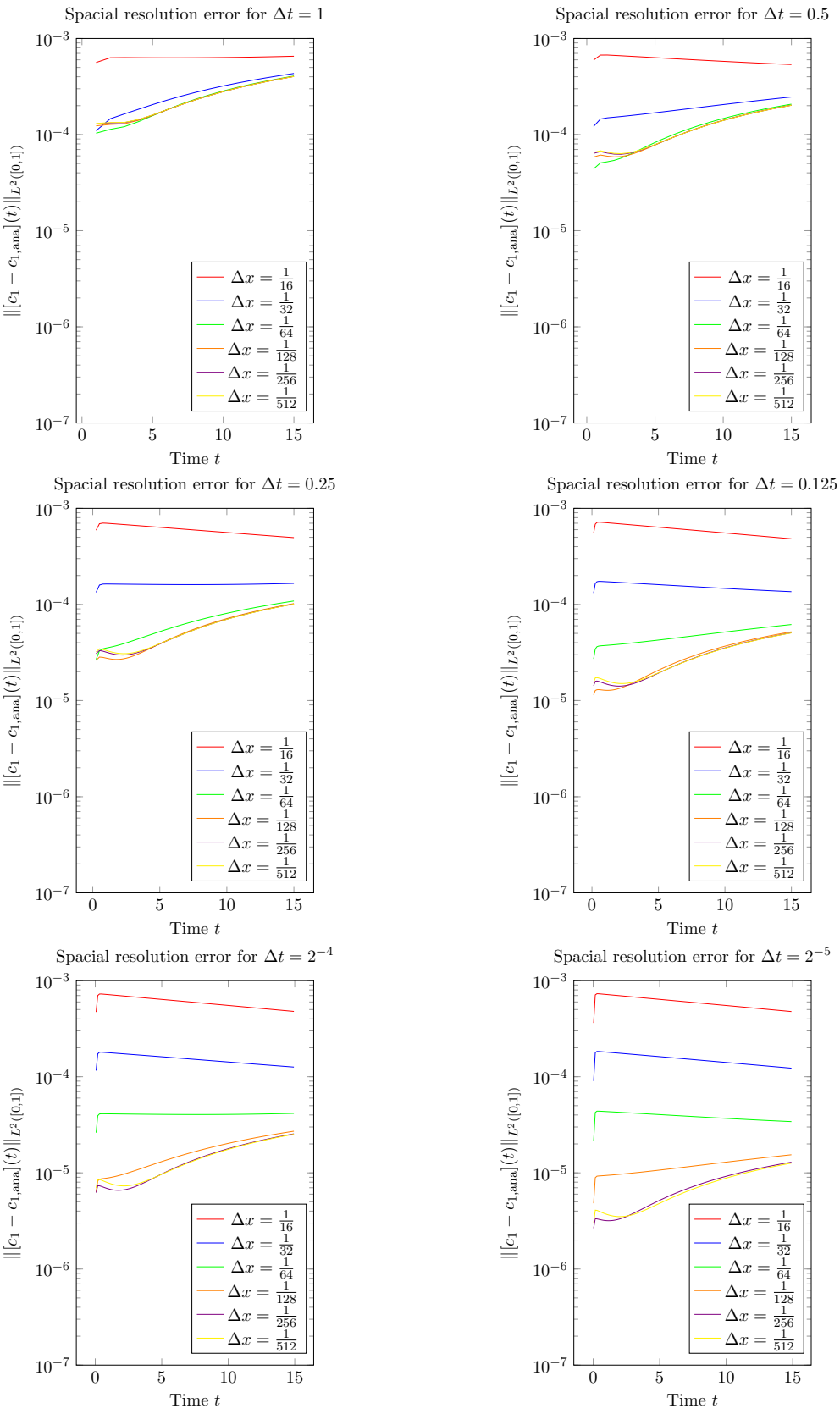



Figure 6: The plots of the analytical $L^2(\Omega)$-errors for $c_1$ for the 1D-MMS-approach as given in section 5.1.1 with $k = \frac{3}{100}$ chosen in (5.4) and (5.11).

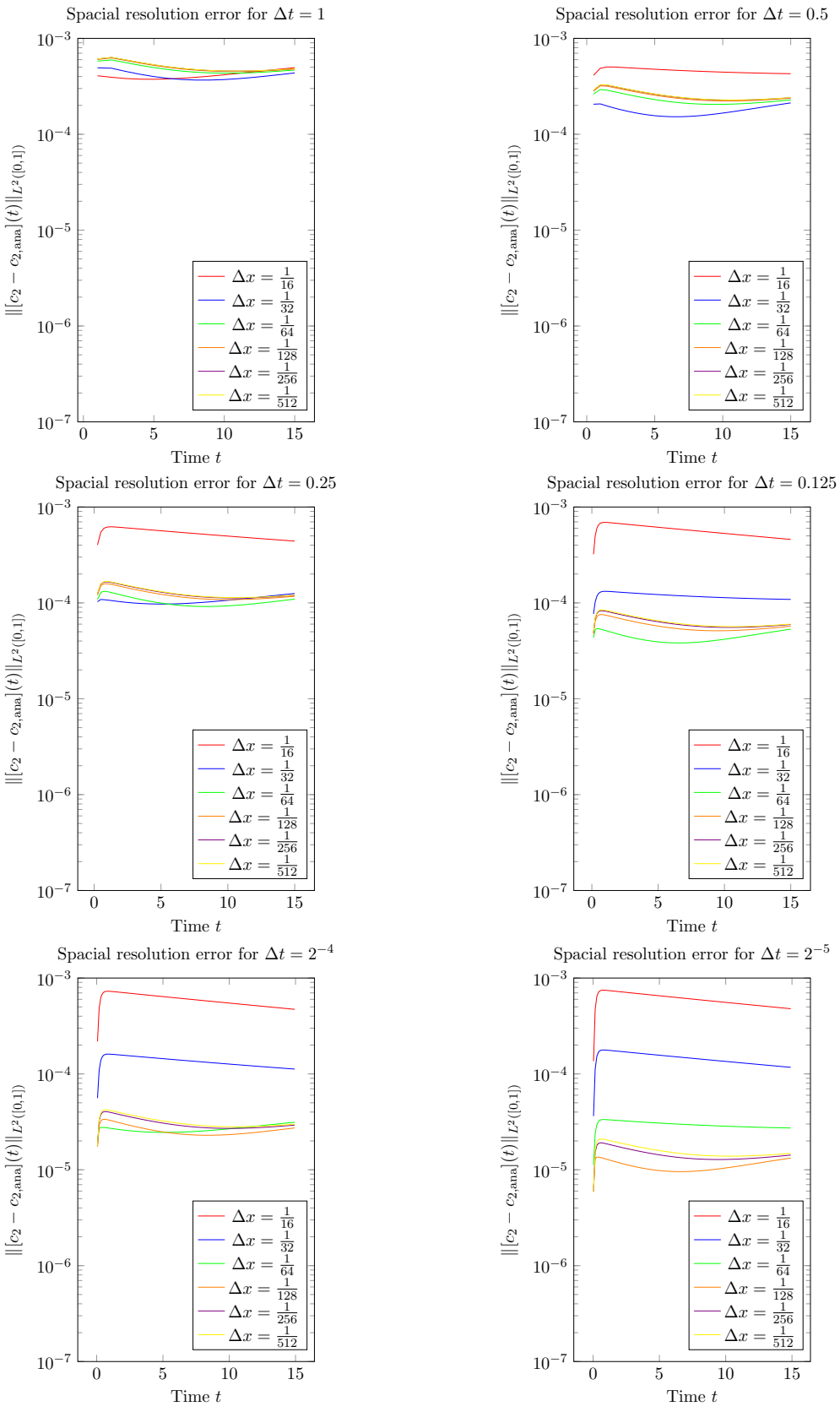



Figure 7: The plots of the analytical $L^2(\Omega)$-errors for $c_2$ for the 1D-MMS-approach as given in section 5.1.1 with $k = \frac{3}{100}$ chosen in (5.4) and (5.11).

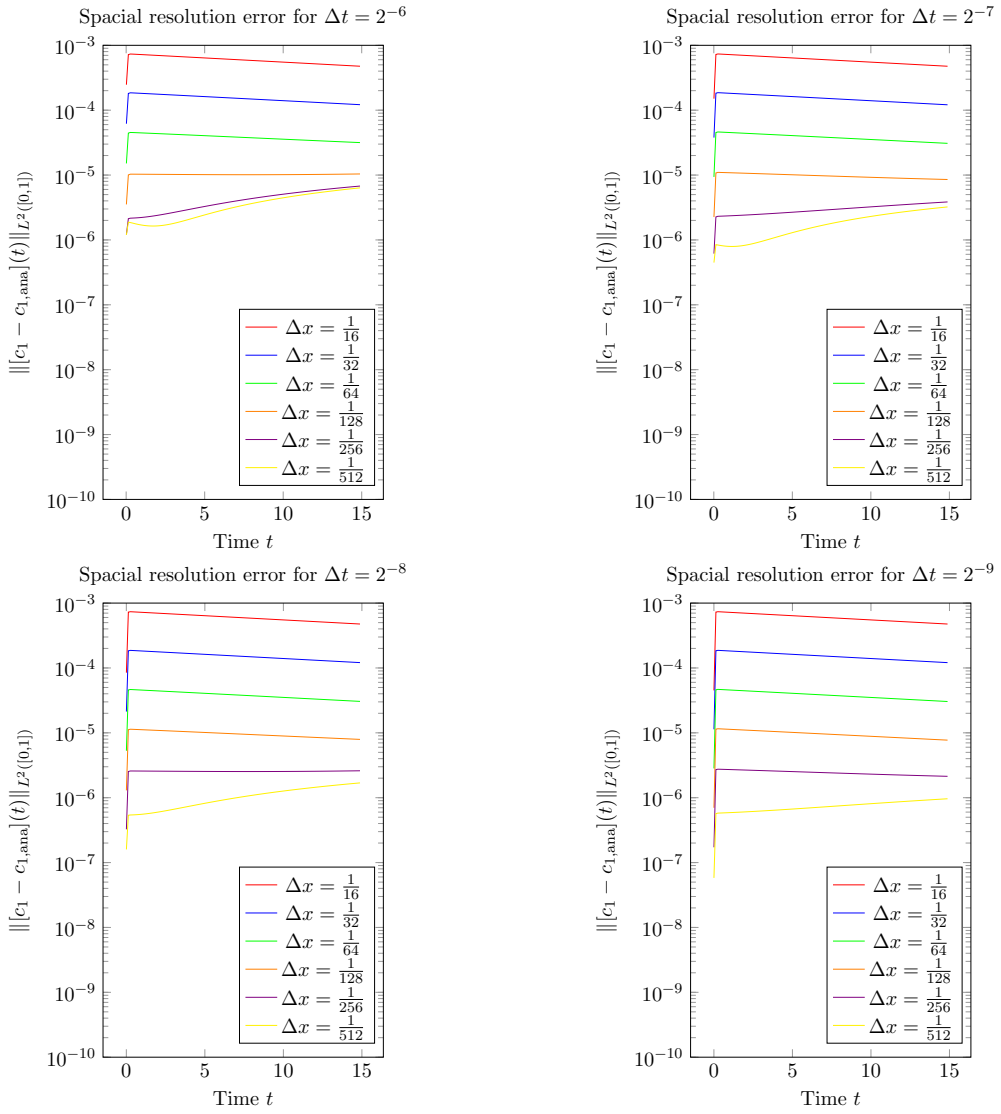



Figure 8: The plots of the analytical $L^2(\Omega)$-errors for $c_1$ for the 1D-MMS-approach as given in section 5.1.1 with $k = \frac{3}{100}$ chosen in (5.4) and (5.11).

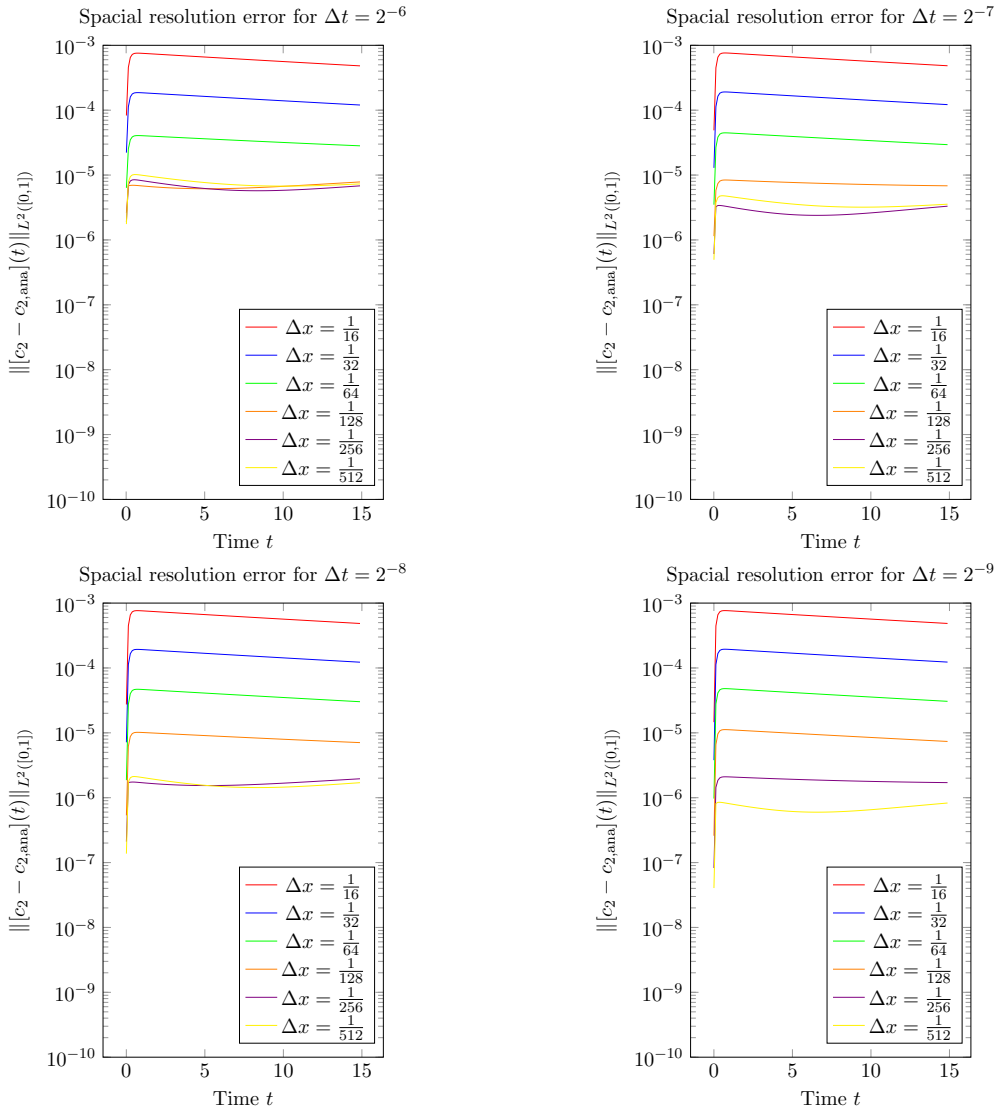



Figure 9: The plots of the analytical $L^2(\Omega)$-errors for $c_2$ for the 1D-MMS-approach as given in section 5.1.1 with $k = \frac{3}{100}$ chosen in (5.4) and (5.11).

| Time [s] | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ | $p_{128\to256}$ | $p_{256\to512}$ |
|---|---|---|---|---|---|
| 5 | 1.905670 | 1.029031 | 0.086651 | -0.011702 | -0.005031 |
| 10 | 1.494016 | 0.482375 | 0.059822 | 0.007799 | 0.001466 |
| 15 | 1.120167 | 0.248526 | 0.032477 | 0.005524 | 0.001210 |

Table 2: Experimental convergence rate table at selected times for the $c_1$-error presenting $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.5$.

| Time [s] | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ | $p_{128\to256}$ | $p_{256\to512}$ |
|---|---|---|---|---|---|
| 5 | 1.986032 | 1.914717 | 1.042173 | 0.100585 | -0.008305 |
| 10 | 1.915682 | 1.508433 | 0.487907 | 0.062106 | 0.008421 |
| 15 | 1.827963 | 1.135680 | 0.251254 | 0.033124 | 0.005691 |

Table 3: Experimental convergence rate table at selected times for the $c_1$-error presenting $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.125$.

| Time [s] | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ | $p_{128\to256}$ | $p_{256\to512}$ |
|---|---|---|---|---|---|
| 5 | 1.977869 | 1.995446 | 2.002272 | 2.007140 | 1.917687 |
| 10 | 1.977218 | 1.992753 | 1.990030 | 1.937252 | 1.513156 |
| 15 | 1.976728 | 1.990592 | 1.978133 | 1.850054 | 1.140778 |

Table 4: Experimental convergence rate table at selected times for the $c_1$-error presenting $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 2^{-9}$.

| Time [s] | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ | $p_{128\to256}$ | $p_{256\to512}$ |
|---|---|---|---|---|---|
| 5 | 1.608828 | -0.557846 | -0.150999 | -0.036744 | -0.009115 |
| 10 | 1.419528 | -0.303269 | -0.116012 | -0.029618 | -0.007423 |
| 15 | 1.011434 | -0.102862 | -0.059077 | -0.016187 | -0.004124 |

Table 5: Experimental convergence rate table at selected times for the $c_2$-error presenting $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.5$.

| Time [s] | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ | $p_{128\to256}$ | $p_{256\to512}$ |
|---|---|---|---|---|---|
| 5 | 2.332589 | 1.638292 | -0.546953 | -0.149866 | -0.036494 |
| 10 | 2.225202 | 1.442079 | -0.296889 | -0.114824 | -0.029334 |
| 15 | 2.078958 | 1.029255 | -0.100462 | -0.058343 | -0.015990 |

Table 6: Experimental convergence rate table at selected times for the $c_2$-error presenting $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.125$.

| Time [s] | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ | $p_{128\to256}$ | $p_{256\to512}$ |
|---|---|---|---|---|---|
| 5 | 1.978136 | 2.014589 | 2.085351 | 2.357688 | 1.647927 |
| 10 | 1.978627 | 2.011602 | 2.069999 | 2.250303 | 1.449483 |
| 15 | 1.978903 | 2.008920 | 2.054054 | 2.104186 | 1.035129 |

Table 7: Experimental convergence rate table at selected times for the $c_2$-error presenting $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 2^{-9}$.

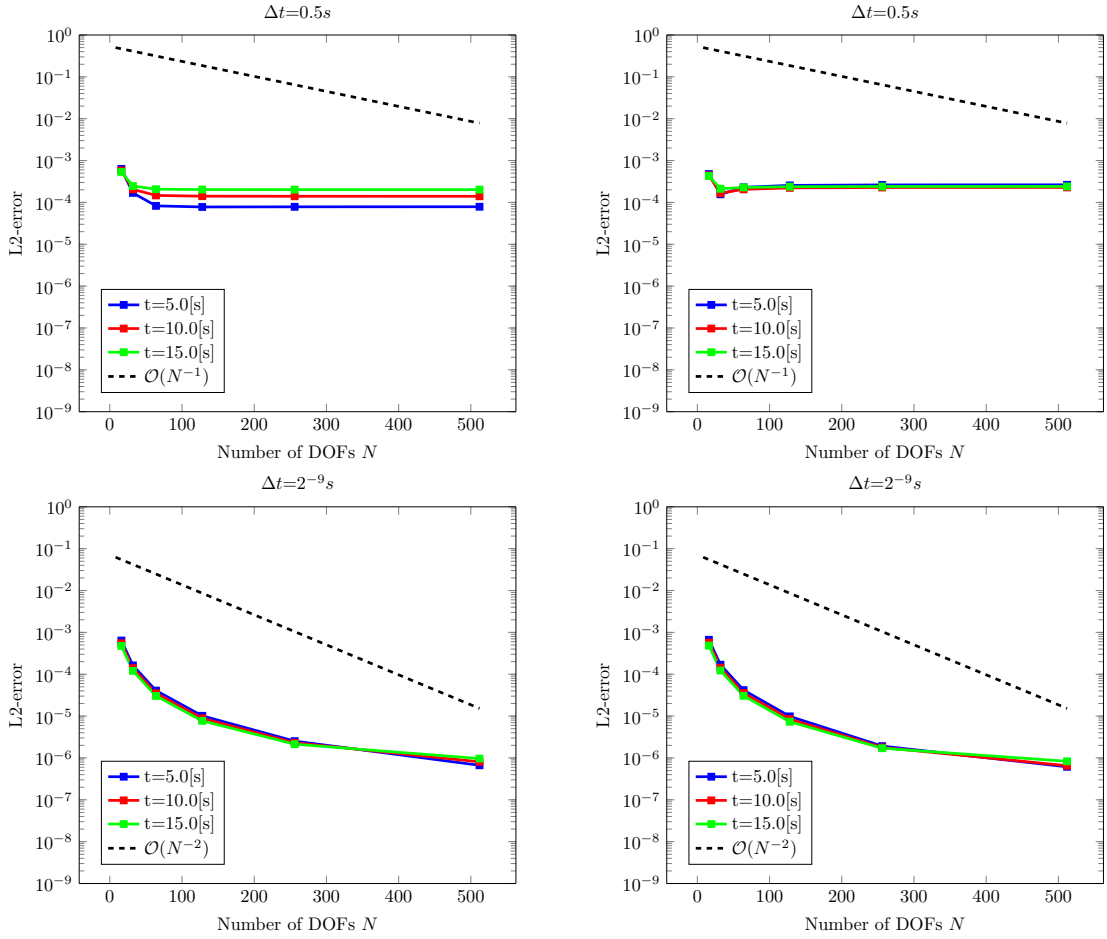



Figure 10: The L2-error dependency w.r.t. to the mesh resolution, as given by $\|[c_1 - c_{1,\text{ana}}](t)\|_{L^2([0,1])}$ on the left and $\|[c_2 - c_{2,\text{ana}}](t)\|_{L^2([0,1])}$ on the right at $t = 5, 10, 15[s]$. Only by choosing a very fine timestep size, increasing the mesh resolution leads to satisfactory convergence properties.

### 5.1.2 2D case for manufactured solutions

In this section, we now consider the performance of the Maxwell-Stefan solver on the unit square $\Omega = (0,1)^2$. In order to triangulate the domain, the Python package `triangle`[42] has been used, providing a Delaunay triangulation of the domain (see figure 3c).
It is straightforward to compute that the homogeneous Neumann boundary conditions are satisfied on each of the edges of the square by the ansatz function (5.4), such that there holds

$$\begin{pmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} \mp 1 \\ 0 \end{pmatrix} = 0 \quad \text{on } \{0\} \times [0,1] \cup \{1\} \times [0,1]$$
$$\begin{pmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \mp 1 \end{pmatrix} = 0 \quad \text{on } [0,1] \times \{0\} \cup [0,1] \times \{1\}$$

By using the computer-algebra algorithm as seen in listing 2, the following results for the MMS source terms obtained are given as follows:

$$\begin{aligned} r_1(x,y,t) = -\frac{1}{6}\Big( & 2\pi^2 D_{12}(-4\cos^2(\pi x)\cos^2(\pi y) + \cos^2(\pi x) + \cos^2(\pi y)) \\ & + k\cos^2(\pi x)\cos^2(\pi y)\Big) e^{-kt} \end{aligned} \tag{5.13a}$$

$$\begin{aligned} r_2(x,y,t) = & \frac{\frac{1}{6}e^{-kt}}{(D_{12}e^{kt} - \frac{1}{6}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y)))^2} \\ & (\pi^2 D_{12}(\frac{1}{12}(D_{12}-D_{23})(2D_{23}e^{kt} - \frac{1}{3}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y))) \\ & (\cos(\pi(2x-2y)) + \cos(\pi(2x+2y)) - 2)\cos^2(\pi x)\cos^2(\pi y) \\ & - (D_{12}e^{kt} - \frac{1}{6}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y))) \\ & (\frac{1}{12}(D_{12}-D_{23})(\cos(4\pi x) - 1)\cos^4(\pi y) \\ & + \frac{1}{12}(D_{12}-D_{23})(\cos(4\pi y) - 1)\cos^4(\pi x) \\ & + 4(-D_{23}e^{kt} + \frac{1}{6}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y)))\cos^2(\pi x)\cos^2(\pi y) \\ & + 2(D_{23}e^{kt} - \frac{1}{6}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y)))\sin^2(\pi x)\cos^2(\pi y) \\ & + 2(D_{23}e^{kt} - \frac{1}{6}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y)))\sin^2(\pi y)\cos^2(\pi x))) \\ & - k(D_{12}e^{kt} - \frac{1}{6}(D_{12}-D_{23})(e^{kt} + \cos^2(\pi x)\cos^2(\pi y)))^2\cos^2(\pi x)\cos^2(\pi y)) \end{aligned} \tag{5.13b}$$

In figures 11 and 12 the numerical performance of the code on a unit square mesh can be studied, as the analytical error is known due to the MMS method. Most of the analysis carried remarked in section 5.1.1 seem also to be valid here, although the restriction of the timestep size seems to be less prominent as in 1D. The experimental convergence rates, as seen in tables 8, 9 for $c_1$ and 10, 11 for $c_2$, are even almost doubled compared to the 1D case and are also dropping less rapidly to very bad convergence behavior for coarse timesteps compared to its 1D equivalents.

[42] Cf. `https://github.com/drufat/triangle`

| Time [s] | $p_{8\to16}$ | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ |
|---|---|---|---|---|
| 5 | 3.734122 | 3.648561 | 2.612119 | 2.104564 |
| 10 | 3.753508 | 3.889441 | 3.238723 | 2.208426 |
| 15 | 3.757053 | 3.946002 | 3.605493 | 2.356740 |

Table 8: Experimental convergence table presenting the experimental convergence rates $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.5$ for the first component.

| Time [s] | $p_{8\to16}$ | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ |
|---|---|---|---|---|
| 5 | 3.733123 | 3.645885 | 2.605533 | 2.087762 |
| 10 | 3.753256 | 3.889005 | 3.235295 | 2.194979 |
| 15 | 3.756929 | 3.946275 | 3.607364 | 2.347811 |

Table 9: Experimental convergence table presenting the experimental convergence rates $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.25$ for the first component.

| Time [s] | $p_{8\to16}$ | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ |
|---|---|---|---|---|
| 5 | 2.134330 | 2.120504 | 2.348947 | 2.278502 |
| 10 | 2.466942 | 2.167397 | 2.403406 | 2.358967 |
| 15 | 2.761745 | 2.210887 | 2.469008 | 2.440262 |

Table 10: Experimental convergence table presenting the experimental convergence rates $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.5$ for the second component.

| Time [s] | $p_{8\to16}$ | $p_{16\to32}$ | $p_{32\to64}$ | $p_{64\to128}$ |
|---|---|---|---|---|
| 5 | 2.097847 | 2.069538 | 2.183829 | 2.633448 |
| 10 | 2.452896 | 2.122454 | 2.243307 | 2.633159 |
| 15 | 2.755883 | 2.172019 | 2.314358 | 2.634613 |

Table 11: Experimental convergence table presenting the experimental convergence rates $p_{N_1\to N_2}$ for the increasing number of mesh points $N$ for timestep size $\Delta t = 0.25$ for the second component.

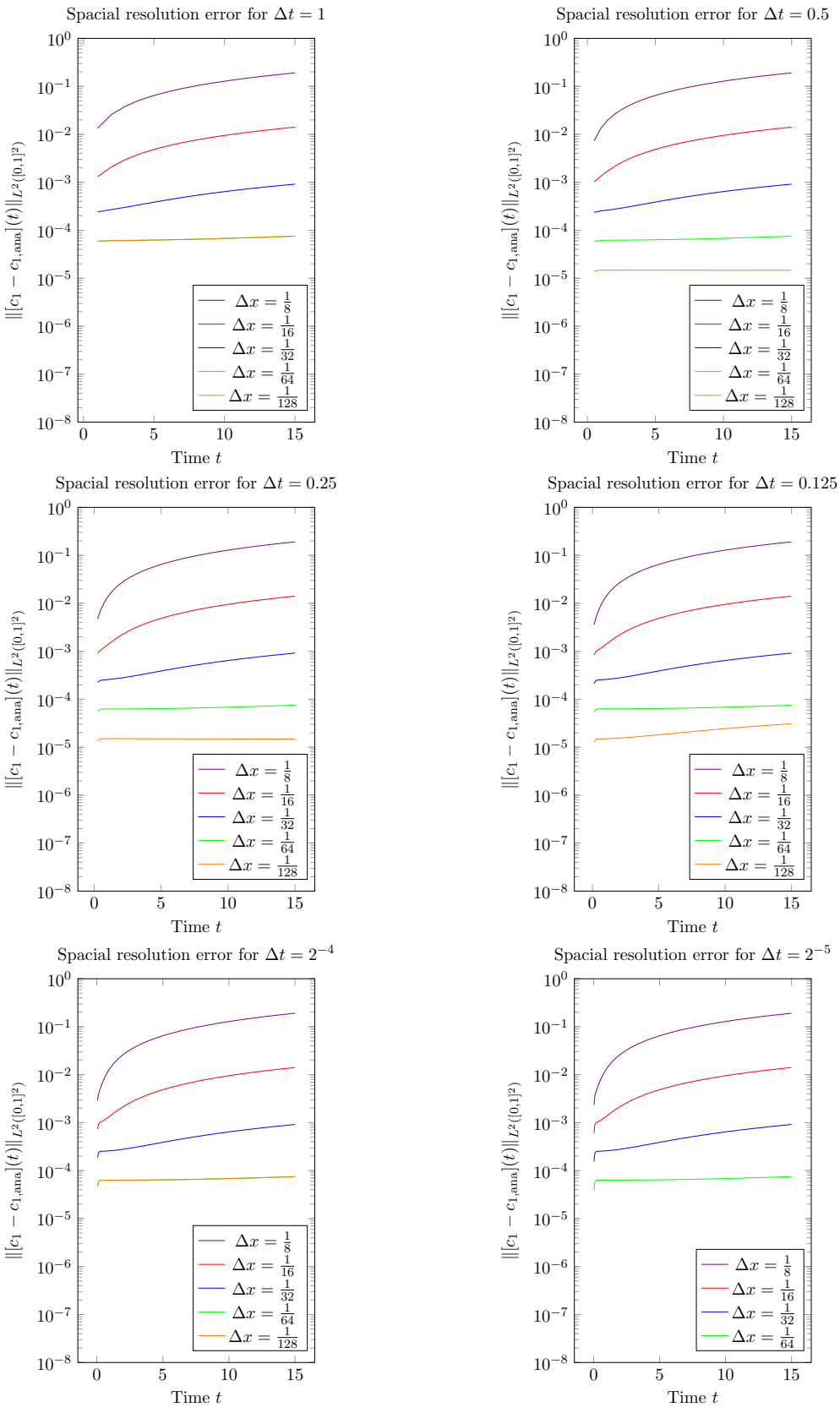



Figure 11: The plots of the analytical $L^2(\Omega)$-errors for $c_1$ for the 2D-MMS-approach as given in section 5.1.2 with $k = \frac{2}{100}$ chosen in (5.4) and (5.13).

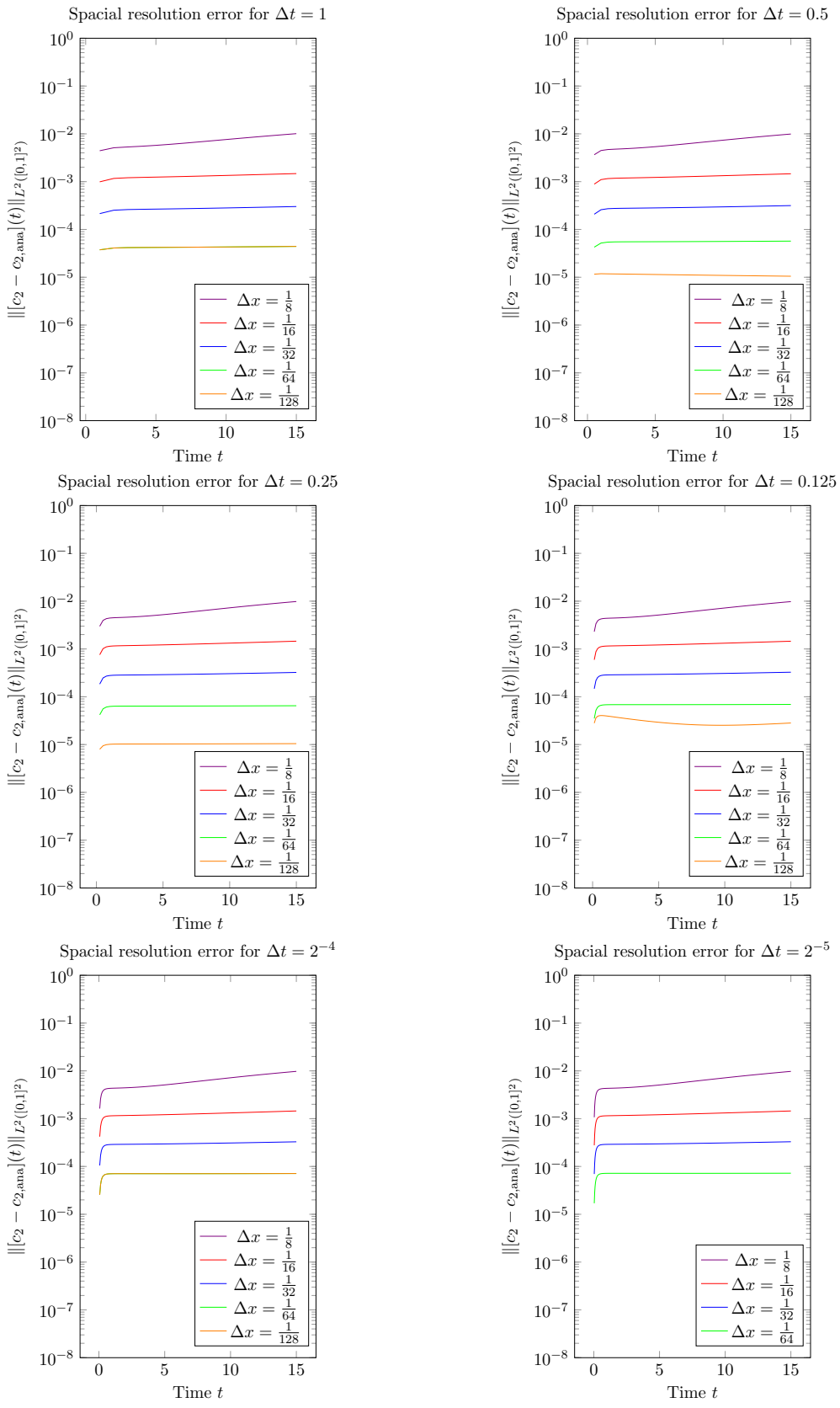



Figure 12: The plots of the analytical $L^2(\Omega)$-errors for $c_2$ for the 2D-MMS-approach as given in section 5.1.2 with $k = \frac{2}{100}$ chosen in (5.4) and (5.13).

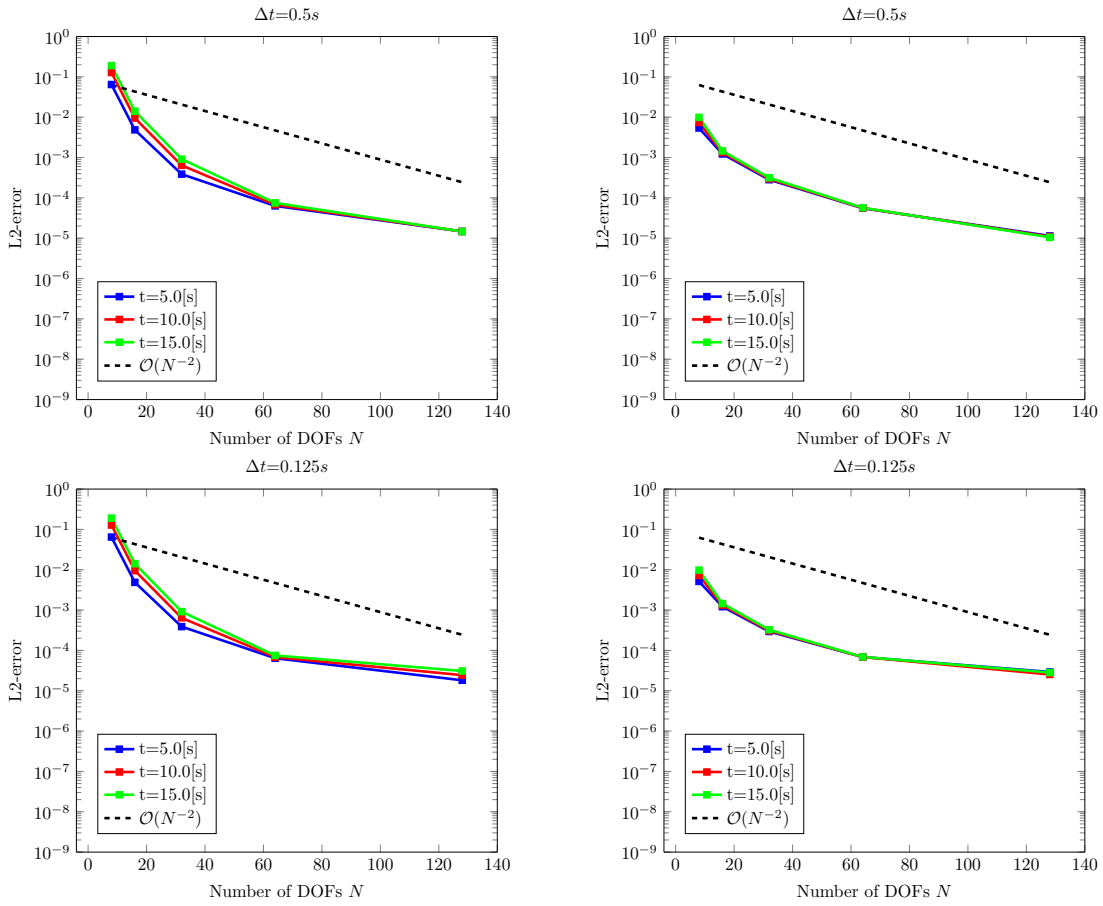



Figure 13: The L2-error dependency w.r.t. to the mesh resolution, as given by $\|[c_1 - c_{1,\text{ana}}](t)\|_{L^2([0,1]^2)}$ on the left and $\|[c_2 - c_{2,\text{ana}}](t)\|_{L^2([0,1]^2)}$ on the right at $t = 5, 10, 15[s]$.

## 5.2 Simulation for the Duncan-Toor experiment

A very well-studied example for situations where the Maxwell-Stefan diffusion laws are superior to classical Fickian theory is the Duncan-Toor experiment, which studies situations where uphill-diffusion phenomena can be observed, i.e. the flux of a gas is directed opposite to its concentration gradient, cf. [67, ch. 5.2].
This experiment by Duncan and Toor as presented in [24] serves as a reference point for most expositions of the Maxwell-Stefan theory as it is a simple yet explanatory experiment, e.g. in [15, sec. 2], [43], [67, sec. 5.4.2] and [69, sec. 2.3].
The Maxwell-Stefan diffusion coefficients for the diffusivities have been taken from [67, p. 130] and are listed in Table 12.

| Species | $H_2$ $(j=1)$ | $N_2$ $(j=2)$ | $CO_2$ $(j=3)$ |
|---|---|---|---|
| $H_2$ $(i=1)$ | - | 83.3 | 68.0 |
| $N_2$ $(i=2)$ | | - | 16.8 |
| $CO_2$ $(i=3)$ | | | - |

Table 12: The Duncan-Toor diffusivities taken from [67, p. 130] for 35.2°C and 1-atm pressure. All entries are given in $\left[\frac{mm^2}{s}\right]$.

| Variable | Value $[mm]$ |
|---|---|
| $R_1$ | 26.49 |
| $R_2$ | 26.58 |
| $l$ | 85.9 |
| $d$ | 2.08 |

Table 13: The key values for setting up the computational Duncan-Toor domain as given in the experiment by Duncan and Toor (taken from [15]), cf. figure 15.

| Species | $c_i^0$ (left bulb) | $c_i^0$ (right bulb) |
|---|---|---|
| $H_2$ $(i=1)$ | 0.000 | 0.501 |
| $N_2$ $(i=2)$ | 0.501 | 0.499 |
| $CO_2$ $(i=3)$ | 0.499 | 0.000 |

Table 14: The initial concentrations for the molar fractions for the Duncan-Toor experiment (taken from [15, Ch. 2]).

In [15, sec. 2] quite extensive numerics for the Duncan-Toor experiment has been carried out, thus this can be used as a further reference solution against which the solution of the code can be compared. As stated in [15], due to the axial symmetry of the experiment, one can restrict to simulating the radial part of the bulbs and the tubes, as the mole fractions and the fluxes do not depend on the angle and the angular component of the flux vector has trivial contributions. The results of the simulation can be observed in figures 17 and 18 for timestep sizes of $\Delta t = 1[s]$ $\Delta t = 0.5[s]$. In above example, 10 hours of concentration exchange have been simulated. The peaks and the shape of the concentrations are in accordance with the reference figures in [15, Fig. 4] or [43, Fig. 3], however the time scale in my figures seems to be off by a factor of $\mathcal{O}(10)$, which I can not explain satisfactorily so far and is hopefully due to a dimensional error in the discretization of some sort. It should however be noted that the uphill-diffusion for Nitrogen, which is the main feature of this experiment, where $N_2$ moves in opposite direction to

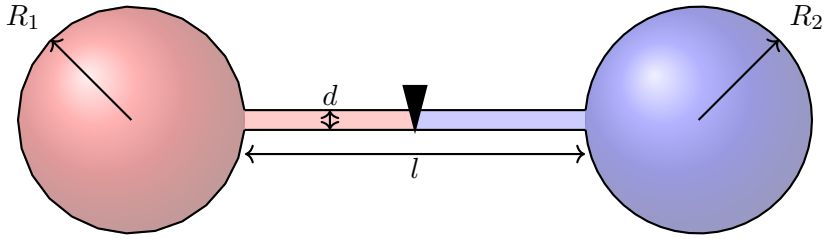



Figure 14: The geometry of the Duncan-Toor experiment, which consists of two circular bulbs connected by a "bridge" of small diameter $d$. The two bulbs contain different mixtures of nitrogen ($N$), carbon dioxide ($CO_2$) and oxygen ($O_2$). The valve in the middle is opened at $t = 0$, thus allowing for concentration exchange via diffusion.

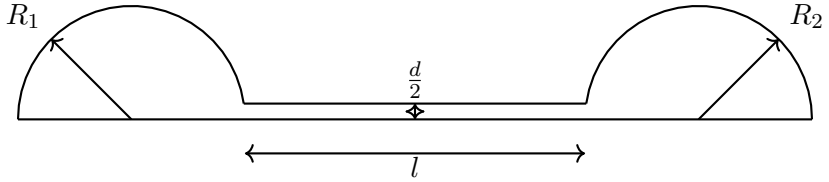
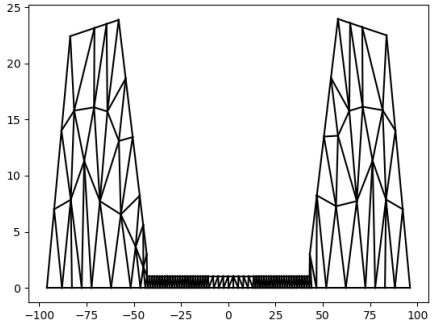
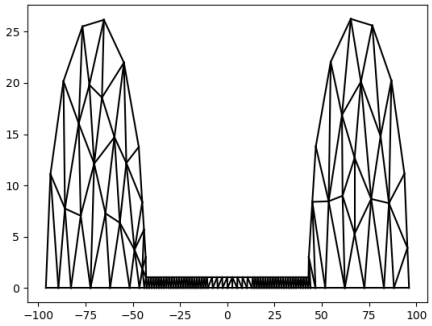



Figure 15: The 2D-computational domain for the Duncan-Toor experiment. The values chosen for $R_1$, $R_2$, $d$ and $l$ are specified in Table 13 as taken from the original experiment by Duncan and Toor [24].

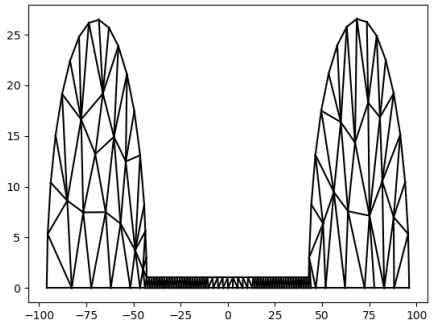
the concentration gradient of the species, is clearly visible in the simulation. This behavior can be explained via the interchange with gas molecules from other species, which carry nitrogen molecules along until an equilibrium is reached for all components. Independently of the unfortunate scaling issue, the main feature of the Maxwell-Stefan equations is reflected quite well for a complicated geometry.




(a) 187 DOF triangulation




(b) 188 DOF triangulation




(c) 201 DOF triangulation

Figure 16: The triangulation of the computational domain as sketched in figure 15.

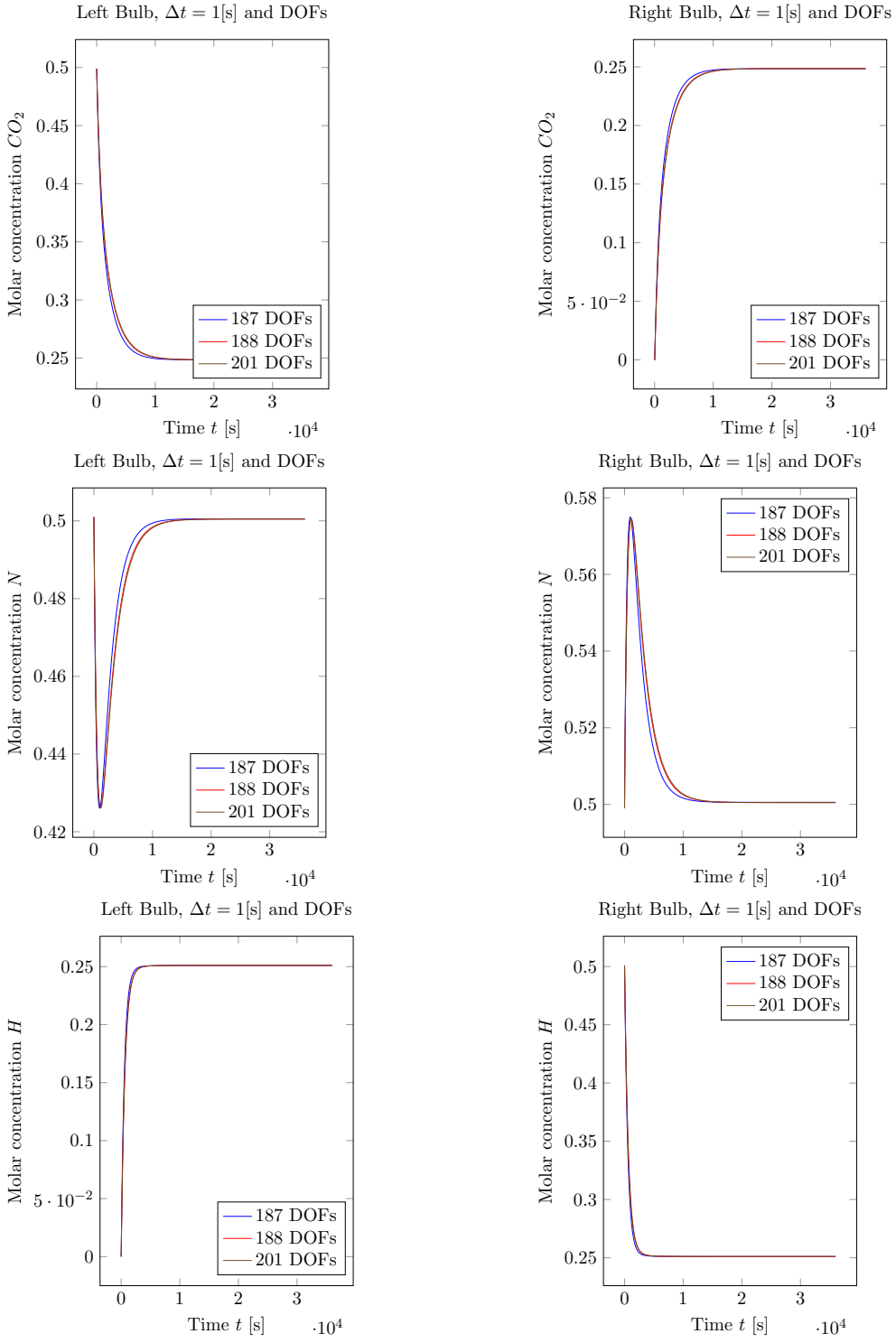



Figure 17: Results for the molar concentrations of the chemical constituents in the Duncan-Toor experiment in the center of the left and right bulb of the geometry (cp. figure 15). Observe the uphill diffusion phenomena for nitrogen.

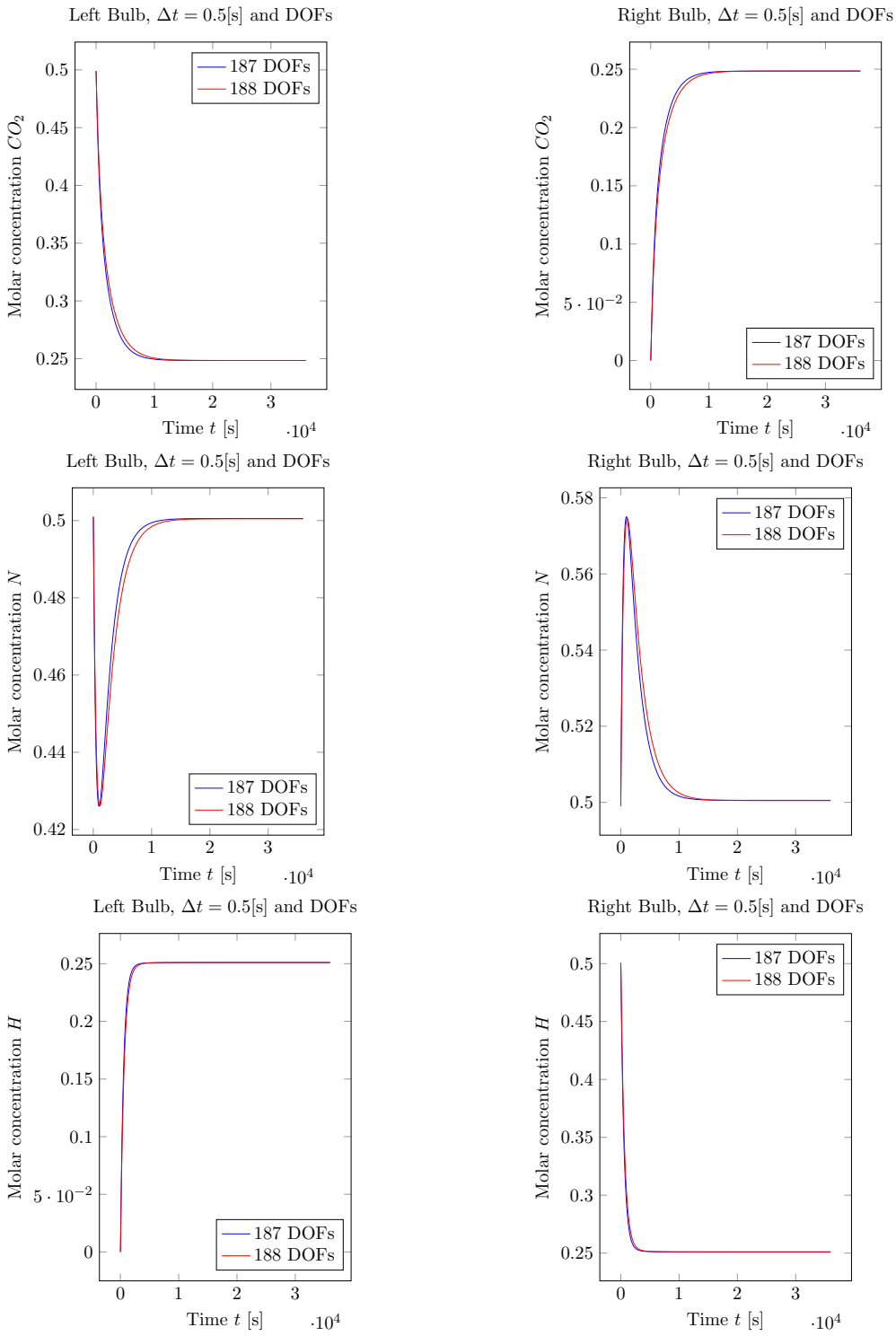



Figure 18: Results for the molar concentrations of the chemical constituents in the Duncan-Toor experiment. Observe the uphill diffusion phenomena for nitrogen.

# 6 Conclusions and outlook

In this thesis, the modeling and the necessary mathematical properties of the existence of a solution to the Maxwell-Stefan equations have been exemplified to great detail.
Still there are many open questions regarding Maxwell-Stefan type systems, mathematically e.g. a proof for the uniqueness of the solution (as stated in [40]), but also regarding the Boltzmann equations and its limits leading to continuum-mechanical descriptions, e.g. the Euler or Navier-Stokes equations (cf. [58]) or as demonstrated the Maxwell-Stefan equations.
As discussed in section 4, a new method for computing the Maxwell-Stefan equations in entropy variables, while using backwards Euler in time and conforming P1-FEM in space has been derived. Various caveats and implementational details of writing code solving the Maxwell-Stefan equations have been addressed, which will require further investigations beyond the proof-of-concept scope in this thesis.
From the numerical experiments as performed in section 5, it has been carried out that the Python implementation written for this thesis using the proposed method can actually solve Maxwell-Stefan type problems. With the benchmark developed by using the Method of Manufactured Solutions, which to the best of my knowledge has not been carried out so far on Maxwell-Stefan type systems, a new benchmark has been presented, which can be used for solver verification throughout the field.
The numerical experiments have also made clear, that the quest for a stable all-purpose solver for the Maxwell-Stefan equations will require some further features in the code as well as a further theoretical understanding of the discretization process.
The most prominent issue arising from my experiments would be an adaptive timestepping scheme, which should reflect that in case of a strongly changing solution, the timestep resolution shall be very small, while in case of an almost stationary solution, the timestep should be chosen as large as possible to significantly reduce computation time.
Furthermore, even though a damped Newton method has been implemented, in some cases for certain combinations of $Đ$, $\Delta t$ and $\Omega$, the Newton solver had troubles converging. This held also for very small timesteps while larger timesteps were converging fine, which is similar to the observations in [70]. Hence it needs more analysis on the numerical scheme on being less reliant on informed guesses and parameter studies, such that the code will adapt the optimal timestep and mesh refinement automatically.
The code present would already support higher-order discretizations in space, which may be interesting to further investigate whether higher-order methods can be successfully applied to solve the Maxwell-Stefan equations as well.
There have also been some interesting developments in using mixed FEM on the Maxwell-Stefan problem as proposed in [48], which may also be an interesting undertaking in formulating an entropy variables formulation in this setting.
Hence I think the quest of finding more performant and stable solvers for the Maxwell-Stefan equations has still a lot of productive research questions left to offer, which will hopefully incite the interested reader to fruitful further investigations.

# A Appendix

## A.1 Results from Linear Algebra

### A.1.1 Important Definitions

Let us first re-state the well-known notions of the the spectral bound and the spectral radius:

**Definition A.1 (Spectral radius).** *Let $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ be a matrix with the associated eigenvalues $\lambda_i \in \mathbb{C}$ and $1 \leq i \leq n$ (eigenvalues are counted repeatedly by algebraic multiplicity). The spectral radius of $A$ denoted by $\rho(A)$ is then given by*

$$\rho(A) := \max_{1\leq i\leq n} |\lambda_i|$$

**Definition A.2 (Spectral bound).** *Let $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ be a matrix with the associated eigenvalues $\lambda_i \in \mathbb{C}$ and $1 \leq i \leq n$ (eigenvalues are counted repeatedly by algebraic multiplicity). The spectral bound of $A$ denoted by $s(A)$ is then given by*

$$s(A) := \max_{1\leq i\leq n} \Re(\lambda_i)$$

*where $\Re$ denotes the real part of the (possibly) complex eigenvalues. Thus the spectral bound is given by the largest real part of the eigenvalues.*

The following definition of an (ir)reducible matrix has been given in [61, Sec. 3.11]:

**Definition A.3 ((Ir)reducible Matrix).** *A matrix $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ is called reducible, if there exists a disjoint partition $\{1,\dots,n\} = I \cup J$ such that $(i,j) \in I \times J$ implies $a_{ij} = 0$. Equivalently, $A$ is reducible if it can be brought in a block-triangular form*

$$\hat{A} = \begin{pmatrix} B & C \\ \boldsymbol{0}_{p,n-p} & D \end{pmatrix}$$

*by a permutation matrix $\Pi \in \mathbb{R}^{n\times n}$ such that $\hat{A} = \Pi A \Pi^{-1}$ (for sub-matrices $B, C, D$ with appropriate dimensions). If $A$ is not reducible, it is called irreducible.*

The following definition of a M-Matrix has been taken from [50]:

**Definition A.4 (M-Matrix).** *A matrix $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ is called M-Matrix iff there exists an $\alpha \in \mathbb{R}$ and a matrix $B \in \mathbb{R}^{n\times n}$ such that $A$ can be written in the form $A = \alpha I_n - B$ with $(B)_{ij} \geq 0$ and $\alpha \geq \rho(B)$, where $\rho(B)$ denotes the spectral radius of $B$ and $I_n \in \mathbb{R}^{n\times n}$ the identity matrix. If $\alpha = \rho(B)$, then $A$ is a singular M-Matrix.*

**Definition A.5 (Quasi-positive Matrix).** *A non-trivial matrix $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ is called quasi-positive (or essentially non-negative), if all $a_{ij} := (A)_{ij}$ are non-negative except on the main-diagonal, i.e. there holds $a_{ij} \geq 0$ for $i \neq j$ and $1 \leq i, j \leq n$.*

Furthermore we make use of the outer matrix product quite some time. As there are various different notations throughout mathematics and physics, the notation used here is explicitly given:

**Definition A.6 (Outer product).** *Let $\underline{x}, \underline{y} \in \mathbb{R}^n$. Then the outer product matrix $\Xi = \underline{x} \otimes \underline{y}$ is given via*

$$\Xi_{ij} = x_{(i)} y_{(j)} \quad 1 \leq i, j \leq n.$$

**Definition A.7 (Adjugate Matrix).** *Let $A \in \mathbb{R}^{n\times n}$ be a real matrix and $n \in \mathbb{N}^+$. Then the adjugate matrix (or adjunct) of $A$ is given by the transpose of the cofactor matrix of $A$, i.e. by*

$$\operatorname{adj}(A) = \operatorname{Cof}(A)^T = \tilde{A}^T = \begin{pmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \cdots & \tilde{a}_{1n} \\ \tilde{a}_{21} & \tilde{a}_{22} & & \tilde{a}_{2n} \\ \vdots & & \ddots & \vdots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \cdots & \tilde{a}_{nn} \end{pmatrix}^T = \begin{pmatrix} \tilde{a}_{11} & \tilde{a}_{21} & \cdots & \tilde{a}_{n1} \\ \tilde{a}_{12} & \tilde{a}_{22} & & \tilde{a}_{n2} \\ \vdots & & \ddots & \vdots \\ \tilde{a}_{1n} & \tilde{a}_{2n} & \cdots & \tilde{a}_{nn} \end{pmatrix}. \tag{A.1a}$$

*Note that at position $(j, i)$ there is the cofactor $\tilde{a}_{ij}$ given. The cofactors $\tilde{a}_{ij}$ can be represented by the minors $M_{ij}$ which arise by deleting the $i$-th row and the $j$-th column, thus*

$$\tilde{a}_{ij} = (-1)^{i+j} \det \underbrace{\begin{pmatrix} a_{1,1} & \cdots & a_{1,j-1} & a_{1,j+1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,j-1} & a_{n,j+1} & \cdots & a_{n,n} \end{pmatrix}}_{=:M_{ij}}. \tag{A.1b}$$

**Definition A.8 (Principal Minor).** *Let $A \in \mathbb{R}^{n\times n}$ and $n \in \mathbb{N}^+$. Then for $1 \leq k \leq n$ the $k$-th principal minor is given by the determinant of the submatrix $A_k \in \mathbb{R}^{k\times k}$*

$$A_k = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \dots & \dots & \dots & \dots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{pmatrix} \tag{A.2}$$

**Definition A.9 (Matrix-Matrix summation).** *Let $A, B \in \mathbb{R}^{n\times n}$ and $n \in \mathbb{N}^+$, then the Matrix-Matrix sum : is a short hand for*

$$A : B = \sum_{i,j=1}^{N} A_{ij} B_{ij}. \tag{A.3}$$

### A.1.2 Important lemmas and theorems

The Sherman-Morrison formula, which was first proven in [62] and can be found in [61, Proposition 3.21] is used to compute the inverse of a matrix $A$, which is perturbed by a rank-one matrix (represented by an outer product):

**Lemma A.1 (Sherman-Morrison formula).** *Let $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ and $\underline{u}, \underline{v} \in \mathbb{R}^n$. If $A$ and $A + \underline{u} \otimes \underline{v}$ are non-singular, then there holds*

$$(A + \underline{u} \otimes \underline{v})^{-1} = A^{-1} - \frac{1}{1 + \underline{v}^T A^{-1} \underline{u}} A^{-1} (\underline{u} \otimes \underline{v}) A^{-1}. \tag{A.4}$$

*Proof.* This is e.g. proven in [61, p. 53f]. □

An useful corollary to the Sherman-Morrison formula is the following lemma to compute the determinant of a matrix perturbed by a rank one matrix (cf. [61, Prop. 3.21]):

**Lemma A.2 (Matrix determinant lemma).** *Given a non-singular matrix $A \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ and $\underline{u}, \underline{v} \in \mathbb{R}^n$, then there holds*

$$\det(A + \underline{u} \otimes \underline{v}) = (1 + \underline{v}^T A^{-1} \underline{u}) \det(A) = \det(A)(\underline{u}^T \operatorname{adj}(A) \underline{v}), \tag{A.5}$$

*where $\operatorname{adj}(A)$ denotes the adjunct matrix of $A$ (cf. definition A.7).*

An important theorem is the Perron-Frobenius theory for non-negative matrices, which results can e.g. be found in [61, Sec 8.3] or [33, Sec. 8.3]:

**Theorem A.1 (Perron-Frobenius).** *Let $A \in \mathbb{R}^{n\times n}$ be a quasi-positive and irreducible matrix. Then its spectral bound $s(A)$ is a simple eigenvalue of $A$ associated with a strictly positive eigenvector and $s(A) > \Re(\lambda_i)$ for all eigenvalues $\lambda_i \in \sigma(A)$ and $\lambda_i \neq \rho(A)$. All eigenvectors of $A$ different from $\rho(A)$ have no positive eigenvector.*

The following theorem (corollary to the Courant-Fisher min-max theorem for hermitean/symmetrical matrices) can be found in [33, Thm. 4.3.1].

**Theorem A.2 (Weyl).** *Let $A, B \in \mathbb{R}^{n\times n}$ for $n \in \mathbb{N}^+$ be symmetric matrices and let the eigenvalues $\lambda_i(A), \lambda_i(B)$ and $\lambda_i(A+B)$ be arranged in increasing order, i.e. of the form*

$$\lambda_{\min} = \lambda_1 \leq \ldots \leq \lambda_n = \lambda_{\max}.$$

*Then for every pair of integers $j, k$ such that $1 \leq j, k \leq n$ and $j + k \geq n + 1$ there one has*

$$\lambda_{j+k-n}(A+B) \leq \lambda_j(A) + \lambda_k(B)$$

*As a corollary there holds*

$$\lambda_i(A) + \lambda_1(B) \leq \lambda_i(A+B) \leq \lambda_i(A) + \lambda_n(B)$$

The following lemma for the outer product of two vectors is given (note that outer products produce a matrix with maximal rank of 1). This is e.g. proven in [61, p. 53]:

**Lemma A.3 (Outer product matrix spectrum).** *Let $\underline{x}, \underline{y} \in \mathbb{R}^n$, and $n \in \mathbb{N}^+$. Then there holds for the spectrum of the outer product matrix $\Xi \in \mathbb{R}^{n\times n}$ with $\Xi = \underline{x} \otimes \underline{v}$*

$$\sigma(\underline{x} \otimes \underline{y}) = (\underbrace{0, \ldots, 0}_{n-1}, \underline{x} \cdot \underline{y}), \tag{A.6}$$

*i.e. 0 is an eigenvalue of algebraic multiplicity $n-1$ and the result of the scalar product of $\underline{x} \cdot \underline{y}$ is the only (possibly) non-trivial eigenvalue.*

**Lemma A.4 (Inverse representation by Cramer's rule).** *Let $A \in \mathbb{R}^{n\times n}$ with $n \in \mathbb{N}^+$ be an invertible matrix. By Cramer's rule, there holds*

$$A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A) \tag{A.7}$$

*where $\operatorname{adj}(A)$ denotes the adjugate of $A$ (cf. definition A.7)*

The following result is useful to compute bounds on the determinant of a matrix, which can be e.g. found in [61, sec. 6.6.1]:

**Theorem A.3 (Hadamard's inequality).** *Let $A \in \mathbb{C}^{n\times n}$ for $n \in \mathbb{N}^+$. Then there holds*

$$|\det(A)| \leq \prod_{i=1}^{n} \left( \sum_{j=1}^{n} |a_{ij}|^2 \right)^{\frac{1}{2}}, \tag{A.8}$$

*i.e. the determinant can be bounded by the product of the Hermitean norm in $\mathbb{C}^n$ of the rows.*

The following result of Theorem A.3 is useful to us:

**Corollary A.1.** *Let $A \in \mathbb{R}^{n\times n}$ a positive semi-definite matrix with real eigenvalues ($\lambda_i \geq 0$ for $1 \leq i \leq n$) and $n \in \mathbb{N}^+$. If there holds $|a_{ij}| \leq \delta$ for a constant $\delta \in \mathbb{R}^+$, then there holds*

$$\det(A) \leq \delta^n n^{n/2} \tag{A.9}$$

*Proof.* As $A$ has eigenvalues which satisfy $\lambda_i \geq 0$ and the determinant can be represented as product of all eigenvalues, there holds $|\det(A)| = \det(A)$. As $|a_{ij}| \leq \delta$, there holds by (A.8)

$$\det(A) \leq \prod_{i=1}^{n} \left( \sum_{j=1}^{n} \delta^2 \right)^{\frac{1}{2}} = \prod_{i=1}^{n} \delta n^{\frac{1}{2}} = \delta^n n^{\frac{n}{2}}. \qquad \square$$

**Theorem A.4.** *Let $A \in \mathbb{R}^{n\times n}$ and $n \in \mathbb{N}^+$ be symmetric and positive definite and $B \in \mathbb{R}^{n\times n}$ be symmetric. Then the number of positive eigenvalues of the product $AB$ equals that for $B$.*

*Proof.* The proof can be found by applying the result in [61, Prop 6.1] for hermitean matrices to symmetric ones. $\square$

**Lemma A.5.** *If $A \in \mathbb{R}^{n\times n}$ is a positive definite, symmetric matrix, then so is its inverse $A^{-1}$.*

*Proof.* As $A$ is positive definite and symmetric ($A = A^T$), it can be inverted since the determinant can be represented as the product of the eigenvalues $\lambda_i > 0$. As $A$ has full rank it is onto $\mathbb{R}^n$, thus an arbitrary $\underline{y} \in \mathbb{R}^n$ can be represented by an appropriate $\underline{x} \in \mathbb{R}^n$ and $\underline{y} = K\underline{x}$. By definition of positive definiteness there holds $x^T Ax > 0$ for any $\underline{x} \in \mathbb{R}^n$. Then there holds

$$y^T A^{-1} y = x^T A^T A^{-1} A x = x^T A x > 0,$$

thus $A^{-1}$ is positive definite. The symmetry $A^{-1} = (A^{-1})^T$ follows by

$$I_n = AA^{-1} = (A^T)^{-1} A^T = (A^{-1})^T A^T = (A^{-1})^T A = A(A^{-1})^T$$

where we made use of the fact that the left- and right-inverse is the same matrix (if *the* inverse matrix exists) as well as the simple-to-prove fact that $(A^{-1})^T = (A^T)^{-1}$. Thus $A^{-1}$ is SPD. $\square$

## A.2 Results from Analysis

### A.2.1 Important definitions

The most important concept in the analytical treatment of PDEs is the notion of Sobolev spaces and its associated norms and seminorms. First of we define the space of smooth functions with compact support in Omega, where for a continuous function $u \in C(\Omega)$

$$\operatorname{supp}(u) = \overline{\{x \in \Omega : u(x) \neq 0\}} \tag{A.10}$$

and

$$C_c^\infty(\Omega) = \{u \in C^\infty(\Omega) : \mathrm{supp}(u) = K \text{ compact} \subset \Omega\} \tag{A.11}$$

Furthermore we use the concept of $L^p$-spaces, which will be understood as $L^p(\Omega) = L^p(\Omega, \mathfrak{B}_d, \lambda^d)$ with respect to the $d$-dimensional Lebesgue measure $\lambda^d$ and the Borel $\sigma$-algebra $\mathfrak{B}_d$ induced by the Euklidean topology in $\mathbb{R}^d$ and restricted on $\Omega$ if not otherwise specified. It is well established, that the Lebesgue spaces for $1 \leq p < \infty$ form a Banach space [9, Th. 4.1.3] with their associated norms[43]

$$\|f\|_{L^p(\Omega)} := \begin{cases} \left(\int_\Omega |f|^p d\lambda^d(\underline{x})\right)^{\frac{1}{p}} & p < \infty \\ \text{ess sup } f = \inf\left\{\sup\{|f(x)|\colon x \in \Omega \backslash N\} : \lambda^d(N) = 0\right\} & p = \infty \end{cases} \tag{A.12}$$

An important notion is the one of weak derivatives [25, p. 242f], which are distributional derivatives that can still be represented in a function space, i.e. in

$$L^1_{\mathrm{loc}}(\Omega) := \left\{u \in L^1(K), \forall K \text{ compact} \subset \Omega\right\} \tag{A.13}$$

**Definition A.10 (Weak derivative).** *Let $u, v \in L^1_{loc}(\Omega)$ where $\Omega \subset \mathbb{R}^d$ and $d \in \mathbb{N}^+$. Then $v$ is called the $\alpha$-th weak derivative of $u$ (denoted as $D^\alpha_w$), iff*

$$\int_\Omega (D^\alpha_w u)\phi d\underline{x} = (-1)^{|\alpha|} \int_\Omega v(D^\alpha \phi) d\underline{x} \quad \forall \phi \in C_c^\infty(\Omega) \tag{A.14}$$

*which is unique up to sets of measure zero [25, p. 243]. Hereby $D^\alpha$ is a short hand for $\underline{\alpha} = (\alpha_1, \ldots, \alpha_d) \in \mathbb{N}^d$ and $D^\alpha = \frac{\partial^{|\underline{\alpha}|}}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}$.*

With the notion of weak derivatives $D^\alpha_w u$, one can define Sobolev spaces as follows (cp. [17, p. 29])[44]:

**Definition A.11 (Sobolev spaces).** *Let $k$ be a nonnegative integer and $1 \leq p < \infty$, one can define the Sobolev norm*

$$\|f\|_{k,p,\Omega} := \|f\|_{W^{k,p}(\Omega)} = \begin{cases} \left(\sum_{|\alpha| \leq k} \|D^\alpha_w f\|^p_{L^p(\Omega)}\right)^{\frac{1}{p}} & p < \infty \\ \max_{|\alpha| \leq k} \|D^\alpha_w f\|_{L^\infty(\Omega)} & p = \infty \end{cases} \tag{A.15a}$$

*and define the Sobolev spaces in either case with*

$$W^{k,p}(\Omega) := \left\{f \in L^1_{loc}(\Omega) : \|f\|_{W^{k,p}(\Omega)} < \infty\right\} \tag{A.15b}$$

*For $p = 2$ one obtains a Hilbert space $H^k(\Omega) = W^{k,2}(\Omega)$ with an inner product for $f, g \in H^k(\Omega)$*

$$(f, g)_{k,\Omega} = (f, g)_{H^k(\Omega)} = \sum_{|\alpha| \leq k} (D^\alpha_w f, D^\alpha_w g)_{L^2(\Omega)} \tag{A.15c}$$

$$\|f\|_{H^k(\Omega)} := \|f\|_{W^{k,2}(\Omega)} \tag{A.15d}$$

[43] Taking into account that we denote with $L^p(\Omega)$ the equivalence class of all functions which coincide $\lambda^d$-a.e. on $\Omega$, cp. further e.g. [9, p. 139f]

[44] There are alternative definitions which lead to the same spaces under moderate restrictions, see e.g. [60, p. 27f].

The associated Sobolev semi-norms for $k \in \mathbb{N}$ and $1 \leq p < \infty$ will be denoted and defined as follows:

$$|f|_{k,p,\Omega} = \begin{cases} \left(\sum_{|\alpha|=k} \|D_w^\alpha f\|_{L^p(\Omega)}^p\right)^{\frac{1}{p}} & p < \infty \\ \max_{|\alpha|=k} \|D_w^\alpha f\|_{L^\infty(\Omega)} & p = \infty \end{cases} \tag{A.16a}$$

For $p = 2$, we will also use the short hand

$$|f|_{H^k(\Omega)} = \left(\sum_{|\alpha|=k} \|D_w^\alpha f\|_{L^2(\Omega)}^2\right)^{\frac{1}{2}}. \tag{A.16b}$$

As we study parabolic equations, we need to extend the notion of Lebesgue spaces to Hilbert-space valued functions. This leads to the following definition (taken from [39, Def. 6.23]):

**Definition A.12 (Hilbert space valued Lebesgue spaces).** *Let $H$ be a Hilbert space and $T > 0$. Then the space denoted by $L^p(0,T;H)$ for $1 \leq p \leq \infty$ is the set (of all equivalence classes) of measurable functions $u : (0,T) \to H$, such that*

$$\|u\|_{L^p(0,T;H)} = \begin{cases} \left(\int_0^T \|u(t)\|_H^p dt\right)^{\frac{1}{p}} & p < \infty \\ \text{ess sup}_{0<t<T} \|u(t)\|_H & p = \infty \end{cases} \tag{A.17}$$

As we are using Cartesian products of Sobolev spaces in this thesis, we should also define the induced norms explicitly for the reader's convenience (taken from [16, p. 22/45]):

**Definition A.13 (Vector valued Lebesgue spaces).** *Let $\underline{u} = (u_1, \ldots, u_n)$ be a function $\underline{u} : \Omega \subset \mathbb{R}^d \to \mathbb{R}^n$ for $d, n \in \mathbb{N}^+$. Then the cartesian product of the Lebesgue spaces $L^p(\Omega)$ is defined componentwise, i.e. by*

$$[L^p(\Omega)]^n = L^p(\Omega; \mathbb{R}^n) = \{u : \Omega \to \mathbb{R}^n : u_{(i)} \in L^p(\Omega)\ \forall 1 \leq i \leq n\},$$

*where $u_{(i)}$ denotes the $i$-th coordinate of $\underline{u}$. The associated $L^p$-norm is defined by*

$$\|\underline{u}\|_{L^p(\Omega;\mathbb{R}^N)} = \begin{cases} \left(\sum_{k=1}^n \|u_{(k)}\|_{L^p(\Omega)}^p\right)^{\frac{1}{p}} & p < \infty \\ \max_{1\leq k\leq n} \|u_{(k)}\|_{L^\infty(\Omega)} & p = \infty \end{cases} \tag{A.18}$$

**Definition A.14 (Vector valued Sobolev spaces).** *Let $\underline{u} = (u_1, \ldots, u_n)$ be a function $\underline{u} : \Omega \to \mathbb{R}^n$ for $n \in \mathbb{N}^+$. Then the cartesian product of the Sobolev space $W^{k,p}(\Omega)$ is defined componentwise, i.e. by*

$$[W^{k,p}(\Omega)]^n = W^{k,p}(\Omega; \mathbb{R}^n) = \{u \in [L^p(\Omega)]^n : u_{(i)} \in W^{k,p}(\Omega)\ \forall 1 \leq i \leq n\}.$$

*The associated Sobolev norm is defined by*

$$\|\underline{u}\|_{[W^{k,p}(\Omega)]^n} = \|\underline{u}\|_{W^{k,p}(\Omega;\mathbb{R}^n)} = \begin{cases} \left(\sum_{k=1}^n \|u_{(k)}\|_{W^{k,p}(\Omega)}^p\right)^{\frac{1}{p}} & p < \infty \\ \max_{1\leq k\leq n} \|u_{(k)}\|_{W^{k,\infty}(\Omega)} & p = \infty \end{cases} \tag{A.19}$$

The following definition is taken from [25, p.254]:

**Definition A.15 (Hölder spaces).** *i. If $u : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ is bounded and continuous, we write*

$$\|u\|_{C(\overline{\Omega})} := \sup_{x\in\Omega} |u(x)|.$$

*ii. The $\gamma$-th Hölder seminorm of $u : \Omega \to \mathbb{R}$ is given by*

$$|u|_{C^{0,\gamma}(\overline{\Omega})} := \sup_{\substack{x,y\in\Omega \\ x\neq y}} \frac{|u(x) - u(y)|}{|x - y|^\gamma}.$$

*The Hölder space $C^{k,\gamma}(\Omega)$ then consists of all functions $u \in C^k(\overline{\Omega})$ for which*

$$\|u\|_{C^{k,\gamma}(\overline{\Omega})} := \sum_{\alpha\leq k} \|D^\alpha u\|_{C(\overline{\Omega})} + \sum_{\alpha=k} |D^\alpha u|_{C^{0,\gamma}(\overline{\Omega})} < \infty. \tag{A.20}$$

The following definition is taken from [25, p. 724]

**Definition A.16 (Compact linear operator).** *Let $X$ and $Y$ be Banach spaces. A bounded linear operator $K : X \to Y$ is called compact provided for each bounded sequence $\{u_k\}_{k=1}^\infty \subset X$, the sequence $\{Ku_k\}_{k=1}^\infty$ is precompact in $Y$; that is, there exists a subsequence $\{u_{k_j}\}_{j=1}^\infty$ such that $\{Ku_{k_j}\}_{j=1}^\infty$ converges in $Y$.*

The following definition is taken from [25, p. 286]:

**Definition A.17 (Compact embedding).** *Let $X$ and $Y$ be Banach spaces and $X \subset Y$. $X$ is compactly embedded in $Y$, written as*

$$X \subset\subset Y$$

*if*

*i. $\|u\|_Y < C\|u\|_X$ for all $u \in X$ for some positive constant $C > 0$.*

*ii. Each bounded sequence in $X$ is precompact in $Y$.*

**Definition A.18 (Weak convergence on Banach spaces).** *Let $X$ be a Banach space and $X'$ denotes the topological dual space to $X$ (linear continuous functionals on $X$). We say the sequence $\{u_i\}_{i=0}^\infty \subset X$ converges weakly to $u \in X$ (denoted by $u_i \rightharpoonup u$), if there holds*

$$\langle v, u_i\rangle_{X'\times X} \to \langle v, u\rangle_{X'\times X} \quad \forall v \in X', \tag{A.21}$$

*where $\langle\cdot,\cdot\rangle_{X'\times X}$ denotes the dual pair $\langle\cdot,\cdot\rangle_{X'\times X} : X' \times X \to \mathbb{R}$.*

**Definition A.19 (Relative compactness).** *Let $(X, \mathcal{T})$ be a topological space, then $K \subseteq X$ is called relatively compact, if its closure w.r.t. to the topology $\mathcal{T}$ is compact, i.e. each open cover there is a finite covering with elements from $\mathcal{T}$.*

### A.2.2 Important lemmas and theorems

The following theorem is taken from [8, Satz 4.3.1]:

**Theorem A.5 (Transformation theorem for $C^1$-Diffeomorphisms).** *Let $D$ be a measurable set in $\mathbb{R}^n$ for $n \in \mathbb{N}^+$, $\phi : D \to \phi(D)$ a $C^1$-Diffeomorphism and let $f$ be a measurable Function on $\phi(D)$. Then there holds*

$$\int_{\phi(D)} f(\underline{x}) d\lambda^n(\underline{x}) = \int_D f(\phi(\underline{y}))|\det(d\phi(y))| d\lambda^n(\underline{y}) \tag{A.22}$$

The following important fixed point theorem which is convenient for many arguments in (nonlinear) PDE theory has e.g. been stated in [35, Th. A.4],[38, Satz 2.13] and is e.g. proven in [27, Th. 11.6]:

**Theorem A.6 (Leray-Schauder fixed point theorem).** *Let $(B, \|\cdot\|_B)$ be a Banach space and $S : B \times [0,1] \to B$ a continuous and compact operator with $S(v,0) = 0$. If for all $v \in B$ there exists a $C > 0$, such that for all $u \in B$ and $\sigma \in [0,1]$ with $S(u,\sigma) = u$ there holds*

$$\|u\|_B \leq C,$$

*then $v \mapsto S(v,1)$ posesses a fixed point.*

On order to prove existence and uniqueness for a linear variational problem the Lax-Milgram lemma can be employed, which can e.g. be found in [17, Th. 2.7.7] and [25, p. 315f]:

**Theorem A.7 (Lax-Milgram).** *Given a Hilbert space $(V, (\cdot,\cdot)_V)$, a bilinear form $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ which is*

- *continuous, i.e. there exists a $C_1 > 0$ for all $u, v \in V$ such that*

$$a(u,v) \leq C_1 \|u\|_V \|v\|_V, \tag{A.23a}$$

- *coercive, i.e. for a $C_2 > 0$ and all $u \in V$ there holds*

$$a(u,u) \geq C_2 \|u\|_V^2, \tag{A.23b}$$

*as well as a linear form $F(\cdot) \in V'$, which is*

- *continuous, i.e. there holds for a $C > 0$ for all $v \in V$*

$$F(v) \leq C\|v\|_V. \tag{A.23c}$$

*Then there exists a unique $u \in V$ that solves the variational problem*

$$a(u, v) = F(v) \quad \forall v \in V. \tag{A.24}$$

The following generalized version of the Poincaré inequality is stated in [35, Th. A.1]:

**Theorem A.8 (Generalized Poincaré inequality).** *Let $\Omega \subset \mathbb{R}^d$ for $d \in \mathbb{N}^+$ be a bounded set with Lipschitz boundary (i.e. $\partial\Omega \in C^{0,1}$) and let $p$ be a continuous semi-norm on $H^m(\Omega)$ for $m \in \mathbb{N}^+$, such that $p(u) = 0$ for any polynomial $u \in \mathbb{P}^{m-1}$, i.e. a polynomial of degree $m-1$, implies $u = 0$. Then there exists a constant $C_p > 0$ such that for all $u \in H^m(\Omega)$ there holds*

$$\|u\|_{H^m(\Omega)} \leq C_p \left( \sum_{|\underline{\alpha}|=m} \|D^\alpha u\|_{L^2(\Omega)} + p(u) \right) = C_p \left( |u|_{H^m(\Omega)} + p(u) \right) \tag{A.25}$$

Jüngel furthermore states that examples for $p(u)$ are $p(u) = \|u\|_{L^2(\Omega)}$, $p(u) = \|u\|_{L^2(\partial\Omega)}$ and for $m = 1$ also $p(u) = \|u\|_{L^1(\Omega)}$.
The following inequality can e.g. be found in [27, p. 145]:

**Lemma A.6 (Young's inequality for Products).** *Let $a, b \in \mathbb{R}$ and $a, b \geq 0$. Then there holds for $1 < p \leq q < \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$ that*

$$ab \leq \frac{a^p}{p} + \frac{b^q}{q}. \tag{A.26}$$

The following Lemma is actually useful to us various times to find lower bounds for the vector valued $H^k$-norm (thanks to Oliver Leingang for the hint):

**Lemma A.7.** *Let $a_i \in \mathbb{R}$ with $a_i \geq 0$ for $1 \leq i \leq n$ and $n \in \mathbb{N}^+$. Then there holds*

$$\sum_{i=1}^{n} a_i^2 \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i \right)^2. \tag{A.27}$$

*Proof.* This can be proven by induction. For $n = 1$, the equality holds trivially, as $a_1^2 \geq \frac{a_1^2}{2}$. We now perform the induction step $n \to n+1$. Let $y := \sum_{i=1}^{n} a_i$. By expanding (A.27), we need to prove

$$\frac{y^2 + a_{n+1}^2}{2} + a_{n+1} y \leq y^2 + a_{n+1}^2.$$

Further simplification leads to

$$a_{n+1}y \leq \frac{y^2 + a_{n+1}^2}{2}.$$

This is exactly Young's inequality (see Lemma A.6) with $p = q = 2$ and as $a_{n+1}, y \geq 0$ (A.27) holds. □

In order to prove compact embedding between certain function spaces, the following result taken from [27, Th. 7.26] is useful:

**Theorem A.9 (Rellich-Kondrachov).** *Let $\Omega \subset \mathbb{R}^d$ be a bounded open subset with $\partial\Omega \in C^{0,1}$. Then the following statements hold for $m \in \mathbb{N}^+$, $j \in \mathbb{N}$:*

*i. If $mp < d$, $1 \leq q < \frac{dp}{d-mp}$, then*

$$W^{m,p}(\Omega) \subset\subset L^q(\Omega),$$

*ii. If $0 \leq k < m - \frac{d}{p} < k + 1$ then for $\alpha = m - \frac{d}{p} - k$ and any $0 < \beta < \alpha$*

$$W^{m,p}(\Omega) \subset\subset C^{k,\beta}(\overline{\Omega}),$$

*where $C^{k,\beta}(\overline{\Omega})$ is defined as in definition A.15.*

The following theorem has been proven by Dreher/Jüngel in [23, Th. 1]:

**Theorem A.10 (Dreher/Jüngel).** *Let $X, B$ and $Y$ be Banach spaces such that the embedding $X \hookrightarrow B$ is compact and the embedding $B \hookrightarrow Y$ is continuous. Furthermore, let either*

*i. $1 \leq p < \infty$, $r = 1$ or*

*ii. $p = \infty$, $r > 1$,*

*and let $u^{(\tau)}$ be a sequence of functions, which are constant on each subinterval $(t_{k-1}, t_k)$ satisfying*

$$\tau^{-1}\|u^{(\tau)} - \sigma_\tau u^{(\tau)}\|_{L^r(\tau,T;Y)} + \|u^{(\tau)}\|_{L^p(0,T;X)} \leq C_0 \quad \forall \tau > 0, \tag{A.28}$$

*where $C_0 > 0$ is a constant which is independent of $\tau$. Then the following holds:*

*i. If $p < \infty$, then $(u^{(\tau)})$ is relatively compact in $L^p(0,T;B)$.*

*ii. If $p = \infty$, there exists a subsequence of $(u^{(\tau)})$ which converges to each space $L^q(0,T;B)$ for $1 \leq q < \infty$ to a limit which belongs to $C^0([0,T];B)$.*

The following result is proven in [7, Korollar 1.5]:

**Theorem A.11 (Eberlein-Šmuljan for reflexive Banach spaces).** *Let $(B, \|\cdot\|_B)$ be a reflexive Banach space. Then each norm bounded sequence in $B$, i.e. $\{u_i\}_{i=0}^{\infty}$ and $\|u_i\|_B \leq C$ for $u_i \in B$, has a weakly convergent subsequence and $u \in B$, i.e.*

$$u_{i_j} \rightharpoonup u.$$

## A.3 Results from Measure Theory

### A.3.1 Important theorems and lemmas

As we've denoted $L^p$-spaces and norms always w.r.t. to the measure space $(\Omega, \mathfrak{B}_d, \lambda^d)$ with the $d$-dimensional Borel $\sigma$-algebra $\mathfrak{B}_d$ and $\lambda^d$ denoting the $d$-dimensional Lebesgue measure, we introduce in this section the notation for an arbitrary measure space $(\Omega, \mathfrak{S}, \mu)$ with an appropriate $\sigma$-algebra $\mathfrak{S}$ and a measure $\mu$

$$\|f\|_{L^p_\mu(\Omega)} = \left( \int_\Omega |f|^p d\mu \right)^{\frac{1}{p}}$$

**Theorem A.12 (Hölder inequality).** *Let $(\Omega, \mathfrak{S}, \mu)$ be a measure space, and let $f, g$ be measureable functions w.r.t. to the measure space. Then there holds for $1 \leq p \leq q \leq \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$ (Hölder-conjugated)*

$$\|fg\|_{L^1_\mu(\Omega)} \leq \|f\|_{L^p_\mu(\Omega)} \|g\|_{L^q_\mu(\Omega)} \tag{A.29}$$

*Proof.* See [44, Satz 13.4/Bem. 13.7] or [9, Th. 2.11.1]. □

For $p = q = 2$ the Hölder inequality yields the Cauchy-Schwarz inequality. By employing the counting measure and the measure space $(\mathbb{N}, 2^{\mathbb{N}}, \mu)$ with

$$\mu(A) = \begin{cases} |A| & \text{A is finite} \\ \infty & \text{A is infinite} \end{cases}$$

one can prove for an arbitrary set $I \subseteq \mathbb{N}$ the discrete version of the Hölder inequality

$$\sum_{i \in I} |a_i b_i| \leq \left( \sum_{i \in I} |a_i|^p \right)^{\frac{1}{p}} \left( \sum_{i \in I} |b_i|^q \right)^{\frac{1}{q}}. \tag{A.30}$$

The following inclusion relation for $L^p$-spaces is quite useful, especially as we consider bounded domains $\Omega \subset \mathbb{R}^d$, which form a finite measure space w.r.t. $(\Omega, \mathfrak{B}_d, \lambda^d)$ :

**Lemma A.8 ($L^p$ inclusion).** *Let $(\Omega, \mathfrak{S}, \mu)$ be a finite measure space. Then there holds for $1 \leq p \leq q \leq \infty$ that $L^q_\mu(\Omega) \subseteq L^p_\mu(\Omega)$. Furthermore each $L^q_\mu$-convergent sequence $(f_n) \in L^q_\mu$ converges in $L^p_\mu$, i.e.*

$$\lim_n \|f_n - f\|_{L^q_\mu(\Omega)} = 0 \Rightarrow \lim_n \|f_n - f\|_{L^p_\mu(\Omega)} = 0$$

*Proof.* This lemma is straightforward to prove by Theorem A.12. There holds for $f \in L^p_\mu$ for $r := \frac{q}{p} > 1$ and $s := \frac{q}{q-p}$ that

$$\| |f|^p 1 \|^p_{L^1_\mu(\Omega)} \overset{(A.29)}{\leq} \| |f|^p \|_{L^r_\mu(\Omega)} \text{meas}(\Omega)^{\frac{1}{s}} = \|f\|^p_{L^q_\mu(\Omega)} \text{meas}(\Omega)^{\frac{1}{s}}.$$

By taking the $p$-th root of the inequality there holds due to $\text{meas}(\Omega) < \infty$ that $L^q_\mu(\Omega) \subseteq L^p_\mu(\Omega)$. The second part of the lemma can be found in [44, Satz 13.25]. □

The following theorem is e.g. proven in [9, Th. 2.8.1]:

**Theorem A.13 (Lebesgue dominated convergence).** *Suppose that $\mu$-integrable functions $f_n$ converge a.e. to a function $f$. If there exists a $\mu$-integrable function $\Phi$ such that*

$$|f_n| \leq \Phi \quad \mu - a.e. \quad \forall n$$

*then the function $f$ is integrable and*

$$\int_\Omega f d\mu = \lim_{n\to\infty} \int_\Omega f_n d\mu$$

*In addition*

$$\lim_{n\to\infty} \int_\Omega |f - f_n| d\mu = 0$$

## A.4 Code used

### A.4.1 Analytical Results

Listing 2: Computation of the MMS results in section 5.1.1

```
from __future__ import print_function
import sympy as sy
import numpy as np
from sympy.utilities.autowrap import ufuncify

def gradient(symbolic_expr, der_symbols):
    """
    This computes the gradient of an symbolic expression with respect to the
    variables given.

    :param symbolic_expr: A Sympy expression containing a symbolic
        representation of differentiable functions.
    :param list der_symbols: A list of symbols to compute the derivatives w.r.
        t.
        them.
    :return: A list containing the components of the gradient of the given
        expression.
    """
    loc_gradient = []

    for der_var in der_symbols:
        res = sy.diff(symbolic_expr, der_var)
```

```
22          loc_gradient.append(sy.simplify(res))

24      return loc_gradient


27  def divergence(vector_expr, der_symbols):
28      """
29      This computes the divergence of a given vector w.r.t. the given
            derivatives
30      symbols.

32      :param list vector_expr: A list of symbolic expressions representing a
33          vector of dimension d.
34      :param der_symbols: The variables, where the derivatives w.r.t. them
            should
35          be computed. You need d derivatives to compute the divergence.
36      :return: The divergence of the vector expression as computed by sympy
37          (simplified expression).
38      """
39      if len(vector_expr) != len(der_symbols):
40          print(vector_expr)
41          print(der_symbols)
42          raise ValueError(
43              "Can only compute the divergence if there are as many derivatives
                    "
44              "as in the vector.")

46      expr = 0

48      for i in range(len(der_symbols)):
49          expr += sy.diff(vector_expr[i], der_symbols[i])

51      expr = sy.simplify(expr)

53      return expr


56  def sym_multiply(sym_coeff, sym_vec):
57      """
58      This provides a "scalar" multiplication of vector valued sympy expression
59      with a float or a scalar sympy expression.

61      :param sym_coeff: The "scalar" coefficient.
62      :param sym_vec: The vector valued Sympy expression.
63      :return:
64      """
65      res_vec = []
66      for curr_elem in sym_vec:
67          res_vec.append(sy.simplify(sym_coeff*curr_elem))

69      return res_vec


72  def sym_add(vec1, vec2):
73      """
74      This adds to vector valued Sympy expressions.

76      :param vec1: The first vector containing sympy expressions.
77      :param vec2: The second vector containing sympy expressions.
78      :return: The vector created from elementwise addition of the elements.
```

```
    """
    if len(vec1) != len(vec2):
        raise ValueError("Vector dimensions must agree!")

    res_vec = []

    for i in range(len(vec1)):
        res_vec.append(sy.simplify(vec1[i]+vec2[i]))

    return res_vec


def maxwell_stefan_analytical_rhs(sol1, sol2, diffusion_matrix, variables):
    """
    This implements a manifactured solution method as the MS are computed in 2D,
    as given in Boudin, Grec, Salvarani (2010): A mathematical and Numerical
    analysis of the MS diffusion equations, p. 6, (18)-(19).

    :param sol1: The solution ansatz function which is desired for the first
        concentration (needs to satisfy BCs!)
    :param sol2: The solution ansatz function which is desired for the second
        concentration (needs to satisfy BCs!)
    :param diffusion_matrix: The Matrix of the binary diffusion coefficients.
        Can consist of symbols as well.
    :param list variables: The list of symbols containing the symbols for the
        derivatives.
    :return: The two right hand sides obtained for the manifactured solution
        problem.
    """
    rhs1 = sy.simplify(sy.diff(sol1, "t")-diffusion_matrix[0, 1]*divergence(
        gradient(sol1, variables), variables))

    # alpha = 1.0/diffusion_matrix[0, 1]-1.0/diffusion_matrix[0, 2]
    beta = 1.0/diffusion_matrix[0, 1]-1.0/diffusion_matrix[1, 2]
    time_derivative = sy.diff(sol2, "t")
    first_term = (1.0/diffusion_matrix[1, 2]+beta*sol1)**(-1)
    second_term = gradient(sol2, variables)
    third_term = sym_multiply(
        beta*diffusion_matrix[0, 1]*sol2, gradient(sol1, variables))
    together = sym_multiply(first_term, sym_add(second_term, third_term))
    rhs2 = sy.simplify(time_derivative-divergence(together, variables))

    return rhs1, rhs2


def eval_sympy_fun(sympy_expr, eval_vec):
    """
    Idea taken from https://stackoverflow.com/questions/15478449/evaluating-a-
    function-at-a-point-in-sympy.

    .. warning::
        Might be obsolete as of more performant new functions using numpy
        backends for sympy.

    :param sympy_expr: An arbitrary sympy expression.
    :param eval_vec: The vector of a given point that evaluates the free
        symbolic variables.
    :return: The numerical value of the sympy function evaluated at the given
        point.
```

```
    """
    free_vars = sorted(sympy_expr.free_symbols, key=lambda a: str(a))
    if len(free_vars) != len(eval_vec):
        raise ValueError(
            "The evaluation points need to be equal to the symbolic vars.")

    evaluated = sy.simplify(sympy_expr.subs(zip(free_vars, eval_vec))).evalf()

    return evaluated


def compute_rhs_unit_square_time_dependent():
    r"""
    This computes the symbolic RHS obtained by the Method of Manifactured
    Solutions (MMS) in 2D on a unit square (0,1)^2.

    This uses the ansatz function

    .. math::
        \phi(x,y,t) = p [1+\cos^2(\pi x) \cos^2(\pi y) e^{-kt}] \; k > 0

    to compute the RHS as obtained by substituting \phi in the equations given
    in Boudin, Grec, Salvarani (2010): A mathematical and Numerical
    analysis of the MS diffusion equations, p. 6, (18)-(19).
    """
    x, y, t = sy.symbols("x y t")
    k, p = sy.symbols("k, p")
    gl1 = p + p * sy.cos(sy.pi * x)**2*sy.cos(sy.pi * y)**2*sy.exp(
        -k * t)
    print(sy.latex(gl1))

    D = np.array([[0, 0.833, 0.833], [0, 0, 0.168], [0, 0, 0]])
    D += D.transpose()

    d00, d01, d02, d10, d11, d12, d20, d21, d22 = sy.symbols(
        "D11, D12, D13, D21, D22, D23, D31, D32, D33")
    dsym = np.array([[d00, d01, d02], [d10, d11, d12], [d20, d21, d22]])

    res_1, res_2 = maxwell_stefan_analytical_rhs(gl1, gl1, dsym, [x, y])

    return res_1, res_2


def compute_rhs_unit_interval_time_dependent():
    r"""
    This computes the symbolic RHS obtained by the Method of Manifactured
    Solutions (MMS) in 1D on the unit interval (0,1).

    This uses the ansatz function

    .. math::
        \phi(x,y,t) = p [1+\cos^2(\pi x) \cos^2(\pi y) e^{-kt}] \; k > 0

    to compute the RHS as obtained by substituting \phi in the equations given
    in Boudin, Grec, Salvarani (2010): A mathematical and Numerical
    analysis of the MS diffusion equations, p. 6, (18)-(19).
    """
    x, t = sy.symbols("x t")
    k, p = sy.symbols("k, p")
    gl1 = p + p * sy.cos(sy.pi * x)**2*sy.exp(-k*t)
```

```
    print(sy.latex(gl1))

    d00, d01, d02, d10, d11, d12, d20, d21, d22 = sy.symbols(
        "D11, D12, D13, D21, D22, D23, D31, D32, D33")
    dsym = np.array([[d00, d01, d02], [d10, d11, d12], [d20, d21, d22]])

    res_1, res_2 = maxwell_stefan_analytical_rhs(gl1, gl1, dsym, [x])

    return res_1, res_2


if __name__ == '__main__':
    res1_1d, res2_1d = compute_rhs_unit_interval_time_dependent()
    res1_2d, res2_2d = compute_rhs_unit_square_time_dependent()
    print(sy.latex(res1_1d))
    print(sy.latex(res2_1d))
    print(sy.latex(res1_2d))
    print(sy.latex(res2_2d))
```

### A.4.2 Code for solving the MS-equations

Listing 3: Main code for solving the M-S equations

```
"""
Functions for solving the Maxwell-Stefan System with conforming P^k-FEM in space
and implicit Euler timestepping
"""
from __future__ import division, print_function
import warnings
import numpy as np
import scipy.sparse.linalg as splinalg
import time
import concurrent.futures

from fe_core.function_spaces.lagrange import LagrangeVectorFunction
from fe_core.function_space import FunctionSpace
from fe_core.finite_elements.lagrange_elements import LagrangeElement
from maxwell_stefan.utils import ParallizationStrategy, JacobianAssembly, \
    NewtonNotConverged

LEGACY_MODE = True

if LEGACY_MODE:
    import maxwell_stefan.assemble_ms_functions_legacy as assemble_ms
    from maxwell_stefan.assemble_bmat_legacy import calc_c_from_w,
        calc_w_from_c
else:
    import assemble_ms_functions as assemble_ms
    from maxwell_stefan.assemble_bmat import calc_c_from_w, calc_w_from_c


def assemble_plain(f, fprime, uold, u0, dt, d_mat, rhs):
    """
    This function assembles the Jacobian and the residual one after the other.

    Parameters
    ----------
    f : function
```

```
35          A function returning the resudual on all DOFs as vector
36      fprime : function
37          A function returing the jacobian of the residual.
38      uold : LagrangeVectorFunction
39          The function values of the previous iteration
40      u0 : LagrangeVectorFunction
41          The function values of the previous timestep.
42      dt : float
43          The timestep size
44      d_mat : np.ndarray
45          The Maxwell-Stefan diffusion matrix.
46      rhs : function or None
47          The (optional) source term of the Maxwell-Stefan equations, which can
                be
48          evaluated in x-space.

50      Returns
51      -------
52      The jacobian and the residual.
53      """
54      jacobian = fprime(uold, u0, dt, d_mat=d_mat)
55      fun_vec = f(uold, u0, dt, d_mat, rhs)

57      return jacobian, fun_vec


60  def assemble_futures(f, fprime, uold, u0, dt, d_mat, rhs):
61      """
62      This function assembles the Jacobian by using several independent python
63      processes to compute the jacobian and the residual concurrently.

65      As each Python interpreter holds the GIL while computing, only multiple
66      processes can make use of multiple cores, however as the process has to be
67      forked and there is no shared memory (data needs to be pickled), there is
68      some overhead involved.

70      Parameters
71      ----------
72      f : function
73          A function returning the resudual on all DOFs as vector
74      fprime : function
75          A function returing the jacobian of the residual.
76      uold : LagrangeVectorFunction
77          The function values of the previous iteration
78      u0 : LagrangeVectorFunction
79          The function values of the previous timestep.
80      dt : float
81          The timestep size
82      d_mat : np.ndarray
83          The Maxwell-Stefan diffusion matrix.
84      rhs : function or None
85          The (optional) source term of the Maxwell-Stefan equations, which can
                be
86          evaluated in x-space.

88      Returns
89      -------
90      The jacobian and the residual.
91      """
92      with concurrent.futures.ProcessPoolExecutor(max_workers=2) as executor:
```

```
93          jacobian_future = executor.submit(fprime, uold, u0, dt, d_mat=d_mat)
94          function_future = executor.submit(f, uold, u0, dt, d_mat, rhs)

96      return jacobian_future.result(), function_future.result()


99  def assemble_futures_threading(f, fprime, uold, u0, dt, d_mat, rhs):
100     """
101     This function assembles the jacobian by using python threading to compute
102     the jacobian and the residual concurrently.

104     As each Python interpreter holds the GIL while computing, the context can
105     only switch when the process releases the GIL after an "atomic" operation
106     (some numpy functions can release the GIL when entering C-Code), there is
107     probably not much speedup for computationally expensive operations.

109     Parameters
110     ----------
111     f : function
112         A function returning the resudual on all DOFs as vector
113     fprime : function
114         A function returing the jacobian of the residual.
115     uold : LagrangeVectorFunction
116         The function values of the previous iteration
117     u0 : LagrangeVectorFunction
118         The function values of the previous timestep.
119     dt : float
120         The timestep size
121     d_mat : np.ndarray
122         The Maxwell-Stefan diffusion matrix.
123     rhs : function or None
124         The (optional) source term of the Maxwell-Stefan equations, which can
                be
125         evaluated in x-space.

127     Returns
128     -------
129     The jacobian and the residual.
130     """
131     with concurrent.futures.ThreadPoolExecutor(max_workers=2) as executor:
132         jacobian_future = executor.submit(fprime, uold, u0, dt, d_mat=d_mat)
133         function_future = executor.submit(f, uold, u0, dt, d_mat, rhs)

135     return jacobian_future.result(), function_future.result()


138 def newton_converged(du, u_vals, norm, tol):
139     r"""
140     This function determines the stopping condition for the Newton method in
141     :py:func:`newton_solve`.

143     Currently the Newton iterations are stopped, when one of the following is
144     true:

146     * max_x |du/u| < tol
147     * ||residuuum||_{l^2} < tol

149     Parameters
150     ----------
151     du : np.ndarray
```

```
152         A numpy array of shape (N*NDof) containing the descent determined by
                the
153         last newton step.
154     u_vals : np.ndarray
155         A numpy array of shape (N, NDof) containing the function values at the
156         DOFs of the last Newton iteration.
157     norm : float
158         The l^2 norm of the residuum on the DOFs throughout the mesh.
159     tol : float
160         The tolerance which is accepted.

162     Returns
163     -------
164     bool
165     """
166     no_relative_change = np.amax(np.abs(du) / u_vals.reshape(du.shape)) < tol
167     converged = norm < u_vals.shape[0] * tol

169     return no_relative_change or converged


172 def newton_solve(
173         f, fprime, u0, dt, omega=1, tol=1e-8, maxiter=50, beta=0.8,
174         damped_steps=6, d_mat=None, rhs=None,
175         parallelization=ParallizationStrategy.NO_CONCURRENCY):
176     """
177     This employs Newton-Raphson such that a root of the function f near u0 is
178     searched.

180     :param f: The function (whole PDE=0) which root should be sought,
181         this should return a vector.
182     :param fprime: The derivative with respect to the DOFs, this should return
                a
183         Jacobian matrix.
184     :param ~.LagrangeVectorFunction u0: This is a vector valued function which
185         stores the values from the prevuous timestep.
186     :param float dt: The size of the timestep.
187     :param float omega:
188     :param float tol: The relative tolerance
189     :param int maxiter: The maximum number of iterations.
190     :param float beta: A parameter between 0 and 1 which is used in case
191         underrelaxation is needed for convergence
192     :param int damped_steps: How many times it should be tried to apply a
                damped
193         Newton step with damping factor beta^n for n<=damped_steps if the
194         residuum should increase after an iteration.
195     :param np.array d_mat: The diffusion matrix evaluated at the old timestep.
196     :param rhs: The source term of the Maxwell-Stefan PDE as a function.
197     :param ParallizationStrategy parallelization: Whether to run a parallel
198         stategy (assembly of Jacobian and residuum on different
199         processes/threads) or not.
200     :return: An array with the new function values for a
201         :class:`~.LagrangeVectorFunction`.
202     :rtype: np.array
203     """
204     dim = u0.dim
205     dims = u0.values.shape
206     unew = LagrangeVectorFunction(u0.function_space, dim=dim)
207     uold = LagrangeVectorFunction(u0.function_space, dim=dim)
208     unew.values[:] = u0.values[:]
```

```
210     jacobian = fprime(unew, u0, dt, d_mat=d_mat)
211     fun_eval = f(unew, u0, dt, d_mat, rhs)
212     deltau = splinalg.spsolve(jacobian, fun_eval)
214     unew.values[:] = u0.values[:]-omega*deltau.reshape(dims)
215     normold = np.linalg.norm(f(unew, u0, dt, d_mat, rhs))
216     iter_count = 0
218     while not newton_converged(deltau, unew.values, normold, tol):
219         iter_count += 1
220         if iter_count > maxiter:
221             raise NewtonNotConverged(maxiter, tol)
223         # deep-copy the values of the previous iteration
224         uold.values = unew.values[:]
226         # assemble the Newton step
227         tbegin = time.time()
228         if parallelization is ParallizationStrategy.NO_CONCURRENCY:
229             jacobian, fun_eval = assemble_plain(
230                 f, fprime, uold, u0, dt, d_mat, rhs)
231         elif parallelization is ParallizationStrategy.FUTURES_PROCESSES:
232             jacobian, fun_eval = assemble_futures(
233                 f, fprime, uold, u0, dt, d_mat, rhs)
234         elif parallelization is ParallizationStrategy.FUTURES_THREADING:
235             jacobian, fun_eval = assemble_futures_threading(
236                 f, fprime, uold, u0, dt, d_mat, rhs)
237         else:
238             raise NotImplementedError(
239                 "Parallelization stategy {} has not been implemented.".format(
240                     parallelization))
241         tend = time.time()
242         print("Wall Time Assemble = {}s".format(tend - tbegin))
244         print("Abbruchbed = %g, normf = %g, maxf = %g"
245               % (np.amax((np.abs(deltau)/unew.values.reshape(dims[0]*dims[1]))
                      ),
246                  np.linalg.norm(fun_eval), np.amax(np.abs(fun_eval))))
248         deltau = splinalg.spsolve(jacobian, fun_eval)
249         unew.values[:] = uold.values[:]-omega*deltau.reshape(dims)
251         # Damped Newton for security
252         normnew = np.linalg.norm(f(unew, u0, dt, d_mat, rhs))
253         n = 0
254         while normnew > normold:
255             print ("Rejected Newton Step, Step(%d)" % n)
256             n += 1
257             unew.values[:] = uold.values[:]-beta**n*deltau.reshape(dims)
258             normnew = np.linalg.norm(f(unew, u0, dt, d_mat, rhs))
259             if n > damped_steps:
260                 print(
261                     "Newton has to perform bad step, as damping after {} "
262                     "steps with beta={} showed did not improve "
263                     "residual!".format(damped_steps, beta))
264                 break
265         if n > damped_steps:
266             warnings.warn(
267                 'Badstep even if damped, trying new search direction.')
```

```
268         normold = normnew

270         print("deltaumax(Step {}) = {}".format(
271             iter_count, np.amax(deltau)))

273     return unew.values[:]


276 def solve_maxwell_stefan(
277         cold, tsteps, degree, d_mat=None, dt=1e-2, rhs=None,
278         jacobian_assembly=JacobianAssembly.COO_1,
279         parallelization=ParallizationStrategy.NO_CONCURRENCY,
280         max_newton_steps=50):
281     """

283     Parameters
284     ----------
285     cold : LagrangeVectorFunction
286         A function storing the molar gas concentration from the last
287         timestep.
288     tsteps : int
289         The number of timesteps to be performed
290     degree : int
291         The degree of the P^k FEM (higher or lower order)
292     d_mat : np.ndarray
293         The MS-diffusion matrix
294     dt : float
295         The timestep size.
296     rhs : LagrangeVectorFunction
297         The source term for the MS equations at current timestep discretized
298         in space.
299     jacobian_assembly : JacobianAssembly
300         The strategy with which to assemble the Jacobian.
301     parallelization : ParallizationStrategy
302         Whether to run a parallel stategy (assembly of Jacobian and
303         residuum on different processes/threads) or not.
304     max_newton_steps : int
305         The maximal number of acceptable Newton iterations until which the
306         method must have converged.

308     Returns
309     -------
310     The entropy variables and the molar concentrations after tsteps
311     timesteps.
312     """
313     # cold contains the initial values of c on the DOFs, cold.shape = N x NDof
314     mesh = cold.function_space.mesh
315     fel = LagrangeElement(mesh.cell, degree)

317     fs = FunctionSpace(mesh, fel)
318     dim = cold.dim
319     wold = LagrangeVectorFunction(fs, dim=dim)
320     wnew = LagrangeVectorFunction(fs, dim=dim)

322     wold.values[:] = calc_w_from_c(cold.values[:], d_mat=d_mat)

324     jac_function = jacobian_assembly.associated_function

326     for k in range(tsteps):
327         wnew.values[:] = newton_solve(
```

```
            getattr(assemble_ms, "maxwell_stefan_function"),
            getattr(assemble_ms, jac_function), wold,
            dt, d_mat=d_mat, rhs=rhs, parallelization=parallelization,
            maxiter=max_newton_steps)
        # print("maxdiff = %f" % np.amax(np.abs(wnew.values[:]-wold.values[:])
            ))
        wold.values[:] = wnew.values[:]
        cold.values[:] = calc_c_from_w(wnew.values[:], d_mat=d_mat)

    return wnew, cold
```

Listing 4: Assemblation of the Matrices (Cython)

```
'''
Compile with
cython AssembleFunNew.pyx
Then use
gcc -shared -pthread -fPIC -fwrapv -O2 -Wall -fno-strict-aliasing
-I/usr/include/python2.7 -o AssembleFunNew.so AssembleFunNew.c
'''

cimport cython
cimport numpy as np
import numpy as np
import scipy as sp
import scipy.sparse as sparse
import assemble_bmat
from fe_core.quadrature import GaussQuadrature


@cython.boundscheck(False)
@cython.wraparound(False)
def maxwell_stefan_function(
        u, uold, double dt, np.ndarray[np.float64_t, ndim = 2] diff_m,
        rhs=None):
    """

    :param Function u: The function object where the data of the new timestep
        shall be stored.
    :param Function uold: The function from the old timestep.
    :param float dt: The timestep in seconds.
    :param diff_m: The Matrix with the binary diffusion Coefficients.
    :param Function rhs: A numpy backend function which can be evaluated at
        numpy arrays
    :return:
    """
    fs = u.function_space
    mesh = fs.mesh

    qr = GaussQuadrature(mesh.cell, 2*fs.element.degree)
    # fel = LagrangeElement(mesh.cell, fs.element.degree)
    fel = fs.element

    cdef unsigned int d, c, no_gases, n_dof, no_cells
    cdef np.ndarray[np.float64_t, ndim = 2] jac, jac_min, uold_qp, phi, \
        uold_values, unew_values, rhs_values
    cdef np.ndarray[np.uint32_t, ndim = 2] cell_nodes
    cdef np.ndarray[np.float64_t, ndim = 3] b_mat, gradphi, jacobians
    cdef np.ndarray[np.float64_t, ndim = 1] qr_weights, cell_measures
    cdef double detJ, meas_j
```

```
    phi = fel.tabulate(qr.points)
    gradphi = fel.tabulate(qr.points, grad=True)

    no_gases = u.dim
    no_cells = mesh.entity_counts[mesh.dim]

    # order u_{1,1},..., u_{1,NDof},..., u_{N,1},..., u_{N,NDof}
    vec = np.zeros(no_gases*fs.node_count, dtype=np.float64)
    n_dof = fs.node_count

    # cache some properties to allow direct numpy access rather than the
        slower
    # Python object access in the loop
    cell_nodes = np.copy(fs.cell_nodes)
    qr_weights = np.copy(qr.weights)
    jacobians = np.copy(fs.mesh.mesh_jacobians)
    cell_measures = np.copy(fs.mesh.cell_measures)
    uold_values = np.copy(uold.values)
    unew_values = np.copy(u.values)

    if rhs is not None:
        rhs_values = np.copy(rhs.values)

    for c in range(no_cells):
        jac = jacobians[c]
        jac_min = np.linalg.inv(jac)
        meas_j = cell_measures[c]
        uold_qp = np.einsum("qi,di->dq", phi, uold_values[:, cell_nodes[c]])
        unew_qp = np.einsum("qi,di->dq", phi, unew_values[:, cell_nodes[c]])
        b_mat = assemble_bmat.assemble_mat_b_alternative(uold_qp, diff_m)

        denom_term_old = (1 + np.einsum("dq->q", np.exp(uold_qp)))**(-1)
        denom_term_new = (1 + np.einsum("dq->q", np.exp(unew_qp)))**(-1)

        for d in range(no_gases):
            vec[n_dof*d+cell_nodes[c]] -= meas_j*np.einsum(
                "q,q,qi,q->i",
                np.exp(uold_qp[d, :]), denom_term_old, phi, qr_weights)

            vec[n_dof*d+cell_nodes[c]] += dt*meas_j*np.einsum(
                "qd,di,ba,qib,ca,qjc,q->j",
                b_mat[:, d, :], unew_values[:, cell_nodes[c]], jac_min,
                gradphi, jac_min, gradphi, qr_weights)

            vec[n_dof*d+cell_nodes[c]] += meas_j*np.einsum(
                "q,q,qi,q->i",
                np.exp(unew_qp[d, :]), denom_term_new, phi, qr_weights)

            if rhs is not None:
                vec[n_dof*d+cell_nodes[c]] -= dt*meas_j*np.einsum(
                    "j,qj,qi,q->i", rhs_values[d, cell_nodes[c]], phi, phi,
                    qr_weights)

    return vec

@cython.boundscheck(False)
@cython.wraparound(False)
def maxwell_stefan_jacobian(
        u, uold, double dt, np.ndarray[np.float64_t, ndim = 2] d_mat):
    fs = u.function_space
```

```
108     mesh = fs.mesh
109     qr = GaussQuadrature(mesh.cell, 4*fs.element.degree)
110     fel = fs.element
111
112     cdef int c, j, m, no_gases, n_dof, no_cells
113     cdef np.ndarray[np.float64_t, ndim = 2] jac_min, unew_qp, uold_qp, phi
114
115     cdef np.ndarray[np.uint32_t, ndim = 2] cell_nodes
116     cdef np.ndarray[np.float64_t, ndim = 3] b_mat, grad_phi, jacobians
117     cdef np.ndarray[np.float64_t, ndim = 1] qr_weights, cell_measures
118     cdef double meas
119
120     phi = fel.tabulate(qr.points)
121     gradphi = fel.tabulate(qr.points, grad=True)
122
123     no_gases = u.dim
124     n_dof = fs.node_count
125     no_cells = fs.mesh.entity_counts[fs.mesh.dim]
126
127     # cache some properties to allow direct numpy access rather than the
        slower
128     # Python object access in the loop
129     cell_nodes = np.copy(fs.cell_nodes)
130     qr_weights = np.copy(qr.weights)
131     jacobians = np.copy(fs.mesh.mesh_jacobians)
132     cell_measures = np.copy(fs.mesh.cell_measures)
133
134     a_mat = sparse.lil_matrix(
135         (no_gases*n_dof, no_gases*n_dof), dtype=np.float64)
136
137     for c in range(no_cells):
138         jac_min = np.linalg.inv(jacobians[c])
139         meas = cell_measures[c]
140         unew_qp = np.einsum("qi,di->dq", phi, u.values[:, cell_nodes[c]])
141         uold_qp = np.einsum("qi,di->dq", phi, uold.values[:, cell_nodes[c]])
142         b_mat = assemble_bmat.assemble_mat_b_alternative(uold_qp, d_mat)
143         denom_term = (1 + np.einsum("dq->q", np.exp(unew_qp)))**(-1)
144         for j in range(no_gases):
145             for m in range(no_gases):
146                 a_mat[np.ix_(
147                     n_dof*j+cell_nodes[c], n_dof*m+cell_nodes[c])] += \
148                     dt*meas * np.einsum("q,ba,qkb,da,qmd,q->km", b_mat[:, j, m
                        ],
149                         jac_min, gradphi, jac_min, gradphi, qr_weights)
150
151                 if m == j:
152                     a_mat[
153                         np.ix_(n_dof*j+cell_nodes[c],
154                         n_dof*m+cell_nodes[c])] += meas* \
155                             np.einsum(
156                                 "q,qi,q,qj,q->ji",
157                                 denom_term, phi,
158                                 np.exp(unew_qp[m, :]), phi, qr_weights)
159
160                 a_mat[np.ix_(
161                     n_dof*j+cell_nodes[c], n_dof*m+cell_nodes[c])] -= \
162                     meas*np.einsum(
163                         "q,q,qi,q,qj,q->ji", denom_term**2,
164                         np.exp(unew_qp[m, :]), phi,
165                         np.exp(unew_qp[j, :]), phi, qr_weights)
```

```
    return a_mat.tocsr()

@cython.boundscheck(False)
@cython.wraparound(False)
def maxwell_stefan_jacobian_coo_matrix(
        u, uold, double dt, np.ndarray[np.float64_t, ndim = 2] d_mat):
    fs = u.function_space
    mesh = fs.mesh
    qr = GaussQuadrature(mesh.cell, 4*fs.element.degree)
    fel = fs.element

    cdef unsigned int cn, j, m, no_gases, n_dof, no_cells, no_cell_dofs
    cdef np.ndarray[np.uint32_t, ndim = 2] cell_nodes
    cdef np.ndarray[np.uint32_t, ndim = 1] row_array, col_array
    cdef np.ndarray[np.float64_t, ndim = 3] b_mat, grad_phi, jacobians
    cdef np.ndarray[np.float64_t, ndim = 2] jac_min, unew_qp, uold_qp, phi
    cdef np.ndarray[np.float64_t, ndim = 1] qr_weights, cell_measures
    cdef double meas

    phi = fel.tabulate(qr.points)
    gradphi = fel.tabulate(qr.points, grad=True)

    no_gases = u.dim
    n_dof = fs.node_count
    no_cell_dofs = fel.node_count
    no_cells = fs.mesh.entity_counts[fs.mesh.dim]

    # deep copy some properties to allow direct numpy access rather than the
    # slower Python object access in the loop
    cell_nodes = fs.cell_nodes[:]
    qr_weights = qr.weights[:]
    jacobians = fs.mesh.mesh_jacobians[:]
    cell_measures = fs.mesh.cell_measures[:]

    no_coo_entries = (
        no_cells*no_gases*(no_gases-1) +
        (no_gases-1)*no_gases*no_cells)*no_cell_dofs**2

    row_array = np.empty(no_coo_entries, dtype=np.uint32)
    col_array = np.empty(no_coo_entries, dtype=np.uint32)
    data = np.empty(no_coo_entries, dtype=np.float64)

    for cn in range(no_cells):
        jac_min = np.linalg.inv(jacobians[cn])
        meas = cell_measures[cn]
        unew_qp = np.einsum("qi,di->dq", phi, u.values[:, cell_nodes[cn]])
        uold_qp = np.einsum("qi,di->dq", phi, uold.values[:, cell_nodes[cn]])
        b_mat = assemble_bmat.assemble_mat_b_alternative(uold_qp, d_mat)
        denom_term = (1+np.einsum("dq->q", np.exp(unew_qp)))**(-1)
        for j in range(no_gases):
            for m in range(no_gases):
                # use loop indexing as done in
                # https://stackoverflow.com/questions/6604502/how-do-i-use-the-
                # indices-of-nested-for-loops-to-generate-a-consecutive-list-
                # of-n
                cell_idx = (m+j*no_gases+cn*no_gases**2)*no_cell_dofs**2
                if cell_idx+no_cell_dofs**2-1 >= no_coo_entries:
                    raise ValueError(
```

```
                        "Cell idx {} too large, number of entries: {}".format(
                            cell_idx, no_coo_entries))
                row_array[cell_idx:cell_idx+no_cell_dofs**2] = np.outer(
                    n_dof*j+cell_nodes[cn],
                    np.ones(no_cell_dofs, dtype=int)).reshape(no_cell_dofs**2)
                col_array[cell_idx:cell_idx+no_cell_dofs**2] = np.tile(
                    n_dof*m+cell_nodes[cn], no_cell_dofs)
                data[cell_idx:cell_idx+no_cell_dofs**2] = dt*meas*np.einsum(
                    "q,ba,qkb,da,qmd,q->km", b_mat[:, j, m],
                    jac_min, gradphi, jac_min, gradphi, qr_weights).reshape(
                        no_cell_dofs**2)
                if m == j:
                    data[cell_idx:cell_idx+no_cell_dofs**2] += meas*np.einsum(
                        "q,qi,q,qj,q->ji",
                        denom_term, phi, np.exp(unew_qp[m, :]),
                        phi, qr_weights).reshape(no_cell_dofs**2)

                data[cell_idx:cell_idx+no_cell_dofs**2] -= meas*np.einsum(
                    "q,q,qi,q,qj,q->ji", denom_term**2,
                    np.exp(unew_qp[m, :]), phi, np.exp(unew_qp[j, :]),
                    phi, qr_weights).reshape(no_cell_dofs**2)

    # convert to a CSR matrix
    jac_mat = sparse.coo_matrix(
        (data, (row_array, col_array)),
        shape=(no_gases*n_dof, no_gases*n_dof)).tocsr()
    return jac_mat


@cython.boundscheck(False)
@cython.wraparound(False)
def maxwell_stefan_jacobian_coo_matrix_2(u, uold, double dt, d_mat):
    fs = u.function_space
    mesh = fs.mesh
    qr = GaussQuadrature(mesh.cell, 4*fs.element.degree)
    fel = fs.element

    cdef unsigned int c, j, m, no_gases, n_dof, no_cells, no_cell_dofs
    cdef np.ndarray[np.float64_t, ndim = 2] jac_min, unew_qp, uold_qp, phi

    cdef np.ndarray[np.uint32_t, ndim = 2] cell_nodes
    cdef np.ndarray[np.float64_t, ndim = 3] b_mat, grad_phi, jacobians
    cdef np.ndarray[np.float64_t, ndim = 1] qr_weights, cell_measures
    cdef double meas

    phi = fel.tabulate(qr.points)
    gradphi = fel.tabulate(qr.points, grad=True)

    no_gases = u.dim
    n_dof = fs.node_count
    no_cell_dofs = fel.node_count
    no_cells = fs.mesh.entity_counts[fs.mesh.dim]

    # cache some properties to allow direct numpy access rather than the
        slower
    # Python object access in the loop
    cell_nodes = np.copy(fs.cell_nodes)
    qr_weights = np.copy(qr.weights)
    jacobians = np.copy(fs.mesh.mesh_jacobians)
    cell_measures = np.copy(fs.mesh.cell_measures)
```

```
    row_array = []
    col_array = []
    data = []

    for c in range(no_cells):
        jac_min = np.linalg.inv(jacobians[c])
        meas = cell_measures[c]
        unew_qp = np.einsum("qi,di->dq", phi, u.values[:, cell_nodes[c]])
        uold_qp = np.einsum("qi,di->dq", phi, uold.values[:, cell_nodes[c]])
        b_mat = assemble_bmat.assemble_mat_b_alternative(uold_qp, d_mat)
        denom_term = (1 + np.einsum("dq->q", np.exp(unew_qp)))**(-1)
        for j in range(no_gases):
            for m in range(no_gases):
                row_array.extend(np.outer(
                    n_dof*j+cell_nodes[c],
                    np.ones(no_cell_dofs, dtype=int)).flatten())
                col_array.extend(np.tile(
                    n_dof*m+cell_nodes[c],
                    no_cell_dofs))
                data.extend(
                    dt*meas*np.einsum("q,ba,qkb,da,qmd,q->km", b_mat[:, j, m],
                        jac_min, gradphi, jac_min, gradphi, qr_weights).
                            reshape(
                    no_cell_dofs**2))
                if m == j:
                    row_array.extend(np.outer(
                        n_dof*j+cell_nodes[c],
                        np.ones(no_cell_dofs, dtype=int)).flatten())
                    col_array.extend(np.tile(
                        n_dof*m+cell_nodes[c],
                        no_cell_dofs))
                    data.extend(
                        meas*np.einsum(
                            "q,qi,q,qj,q->ji",
                            denom_term, phi,
                            np.exp(unew_qp[m, :]),
                            phi, qr_weights).reshape(no_cell_dofs**2))
                row_array.extend(np.outer(
                    n_dof*j+cell_nodes[c],
                    np.ones(no_cell_dofs, dtype=int)).flatten())
                col_array.extend(np.tile(
                    n_dof*m+cell_nodes[c],
                    no_cell_dofs))
                data.extend(
                    -meas*np.einsum(
                        "q,q,qi,q,qj,q->ji", denom_term**2,
                        np.exp(unew_qp[m, :]), phi,
                        np.exp(unew_qp[j, :]),
                        phi, qr_weights).reshape(no_cell_dofs**2))

    # convert to a CSR matrix
    jac_mat = sparse.coo_matrix(
        (data, (row_array, col_array)),
        shape=(no_gases*n_dof, no_gases*n_dof)).tocsr()
    return jac_mat
```

Listing 5: Assemblation of the Matrices

```
import numpy as np
import scipy.sparse as sparse
```

```
3  from maxwell_stefan.assemble_bmat_legacy import assemble_mat_b_alternative
4  from fe_core.quadrature import GaussQuadrature


7  def maxwell_stefan_function(u, uold, dt, diff_m, rhs=None):
8      """
10     :param Function u: The function object where the data of the new timestep
11         shall be stored.
12     :param Function uold: The function from the old timestep.
13     :param float dt: The timestep in seconds.
14     :param diff_m: The Matrix with the binary diffusion Coefficients.
15     :param Function rhs: A numpy backend function which can be evaluated at
16         numpy arrays
17     :return:
18     """
19     fs = u.function_space
20     mesh = fs.mesh

22     qr = GaussQuadrature(mesh.cell, 2*fs.element.degree)
23     fel = fs.element

25     phi = fel.tabulate(qr.points)
26     gradphi = fel.tabulate(qr.points, grad=True)

28     no_gases = u.dim
29     no_cells = mesh.entity_counts[mesh.dim]

31     # order u_{1,1},..., u_{1,NDof},..., u_{N,1},..., u_{N,NDof}
32     vec = np.zeros(no_gases*fs.node_count, dtype=np.float64)
33     n_dof = fs.node_count

35     # cache some properties to allow direct numpy access rather than the
           slower
36     # Python object access in the loop
37     cell_nodes = np.copy(fs.cell_nodes)
38     qr_weights = np.copy(qr.weights)
39     jacobians = np.copy(fs.mesh.mesh_jacobians)
40     cell_measures = np.copy(fs.mesh.cell_measures)
41     uold_values = np.copy(uold.values)
42     unew_values = np.copy(u.values)

44     if rhs is not None:
45         rhs_values = np.copy(rhs.values)

47     for c in range(no_cells):
48         jac = jacobians[c]
49         jac_min = np.linalg.inv(jac)
50         meas_j = cell_measures[c]
51         uold_qp = np.einsum("qi,di->dq", phi, uold_values[:, cell_nodes[c]])
52         unew_qp = np.einsum("qi,di->dq", phi, unew_values[:, cell_nodes[c]])
53         b_mat = assemble_mat_b_alternative(uold_qp, diff_m)

55         denom_term_old = (1 + np.einsum("dq->q", np.exp(uold_qp)))**(-1)
56         denom_term_new = (1 + np.einsum("dq->q", np.exp(unew_qp)))**(-1)

58         for d in range(no_gases):
59             vec[n_dof*d+cell_nodes[c]] -= meas_j*np.einsum(
60                 "q,q,qi,q->i",
61                 np.exp(uold_qp[d, :]), denom_term_old, phi, qr_weights)
```

```
63              vec[n_dof*d+cell_nodes[c]] += dt*meas_j*np.einsum(
64                  "qd,di,ba,qib,ca,qjc,q->j",
65                  b_mat[:, d, :], unew_values[:, cell_nodes[c]], jac_min,
66                  gradphi, jac_min, gradphi, qr_weights)
67
68              vec[n_dof*d+cell_nodes[c]] += meas_j*np.einsum(
69                  "q,q,qi,q->i",
70                  np.exp(unew_qp[d, :]), denom_term_new, phi, qr_weights)
71
72              if rhs is not None:
73                  vec[n_dof*d+cell_nodes[c]] -= dt*meas_j*np.einsum(
74                      "j,qj,qi,q->i", rhs_values[d, cell_nodes[c]], phi, phi,
75                      qr_weights)
76
77      return vec
78
79
80  def maxwell_stefan_jacobian(u, uold, dt, d_mat):
81      fs = u.function_space
82      mesh = fs.mesh
83      qr = GaussQuadrature(mesh.cell, 4*fs.element.degree)
84      fel = fs.element
85
86      phi = fel.tabulate(qr.points)
87      gradphi = fel.tabulate(qr.points, grad=True)
88
89      no_gases = u.dim
90      n_dof = fs.node_count
91      no_cells = fs.mesh.entity_counts[fs.mesh.dim]
92
93      # cache some properties to allow direct numpy access rather than the
            slower
94      # Python object access in the loop
95      cell_nodes = np.copy(fs.cell_nodes)
96      qr_weights = np.copy(qr.weights)
97      jacobians = np.copy(fs.mesh.mesh_jacobians)
98      cell_measures = np.copy(fs.mesh.cell_measures)
99
100     a_mat = sparse.lil_matrix(
101         (no_gases*n_dof, no_gases*n_dof), dtype=np.float64)
102
103     for c in range(no_cells):
104         jac_min = np.linalg.inv(jacobians[c])
105         meas = cell_measures[c]
106         unew_qp = np.einsum("qi,di->dq", phi, u.values[:, cell_nodes[c]])
107         uold_qp = np.einsum("qi,di->dq", phi, uold.values[:, cell_nodes[c]])
108         b_mat = assemble_mat_b_alternative(uold_qp, d_mat)
109         denom_term = (1 + np.einsum("dq->q", np.exp(unew_qp)))**(-1)
110         for j in range(no_gases):
111             for m in range(no_gases):
112                 a_mat[np.ix_(
113                     n_dof*j+cell_nodes[c], n_dof*m+cell_nodes[c])] += \
114                     dt*meas * np.einsum("q,ba,qkb,da,qmd,q->km", b_mat[:, j, m
                            ],
115                         jac_min, gradphi, jac_min, gradphi, qr_weights)
116
117                 if m == j:
118                     a_mat[
119                         np.ix_(n_dof*j+cell_nodes[c],
```

```
                        n_dof*m+cell_nodes[c])] += meas* \
                            np.einsum(
                                "q,qi,q,qj,q->ji",
                                denom_term, phi,
                                np.exp(unew_qp[m, :]), phi, qr_weights)

                a_mat[np.ix_(
                    n_dof*j+cell_nodes[c], n_dof*m+cell_nodes[c])] -= \
                    meas*np.einsum(
                        "q,q,qi,q,qj,q->ji", denom_term**2,
                        np.exp(unew_qp[m, :]), phi,
                        np.exp(unew_qp[j, :]), phi, qr_weights)

    return a_mat


def maxwell_stefan_jacobian_coo_matrix(u, uold, dt, d_mat):
    fs = u.function_space
    mesh = fs.mesh
    qr = GaussQuadrature(mesh.cell, 4*fs.element.degree)
    fel = fs.element

    phi = fel.tabulate(qr.points)
    gradphi = fel.tabulate(qr.points, grad=True)

    no_gases = u.dim
    n_dof = fs.node_count
    no_cell_dofs = fel.node_count
    no_cells = fs.mesh.entity_counts[fs.mesh.dim]

    # deep copy some properties to allow direct numpy access rather than the
    # slower Python object access in the loop
    cell_nodes = fs.cell_nodes[:]
    qr_weights = qr.weights[:]
    jacobians = fs.mesh.mesh_jacobians[:]
    cell_measures = fs.mesh.cell_measures[:]
    no_coo_entries = (
        no_cells*no_gases*(no_gases-1) +
        (no_gases-1)*no_gases*no_cells)*no_cell_dofs**2

    row_array = np.empty(no_coo_entries, dtype=np.uint32)
    col_array = np.empty(no_coo_entries, dtype=np.uint32)
    data = np.empty(no_coo_entries, dtype=np.float64)

    for cn in range(no_cells):
        jac_min = np.linalg.inv(jacobians[cn])
        meas = cell_measures[cn]
        unew_qp = np.einsum("qi,di->dq", phi, u.values[:, cell_nodes[cn]])
        uold_qp = np.einsum("qi,di->dq", phi, uold.values[:, cell_nodes[cn]])
        b_mat = assemble_mat_b_alternative(uold_qp, d_mat)
        denom_term = (1+np.einsum("dq->q", np.exp(unew_qp)))**(-1)
        for j in range(no_gases):
            for m in range(no_gases):
                # use loop indexing as done in
                # https://stackoverflow.com/questions/6604502/how-do-i-use-the-
                # indices-of-nested-for-loops-to-generate-a-consecutive-list-
                # of-n
                cell_idx = (m+j*no_gases+cn*no_gases**2)*no_cell_dofs**2
                if cell_idx+no_cell_dofs**2-1 >= no_coo_entries:
```

```
                    raise ValueError(
                        "Cell idx {} too large, number of entries: {}".format(
                            cell_idx, no_coo_entries))
                row_array[cell_idx:cell_idx+no_cell_dofs**2] = np.outer(
                    n_dof*j+cell_nodes[cn],
                    np.ones(no_cell_dofs, dtype=int)).reshape(no_cell_dofs**2)
                col_array[cell_idx:cell_idx+no_cell_dofs**2] = np.tile(
                    n_dof*m+cell_nodes[cn], no_cell_dofs)
                data[cell_idx:cell_idx+no_cell_dofs**2] = dt*meas*np.einsum(
                    "q,ba,qkb,da,qmd,q->km", b_mat[:, j, m],
                    jac_min, gradphi, jac_min, gradphi, qr_weights).reshape(
                        no_cell_dofs**2)
                if m == j:
                    data[cell_idx:cell_idx+no_cell_dofs**2] += meas*np.einsum(
                        "q,qi,q,qj,q->ji",
                        denom_term, phi, np.exp(unew_qp[m, :]),
                        phi, qr_weights).reshape(no_cell_dofs**2)

                data[cell_idx:cell_idx+no_cell_dofs**2] -= meas*np.einsum(
                    "q,q,qi,q,qj,q->ji", denom_term**2,
                    np.exp(unew_qp[m, :]), phi, np.exp(unew_qp[j, :]),
                    phi, qr_weights).reshape(no_cell_dofs**2)

    # convert to a CSR matrix
    jac_mat = sparse.coo_matrix(
        (data, (row_array, col_array)),
        shape=(no_gases*n_dof, no_gases*n_dof)).tocsr()
    return jac_mat


def maxwell_stefan_jacobian_coo_matrix_2(u, uold, dt, d_mat):
    fs = u.function_space
    mesh = fs.mesh
    qr = GaussQuadrature(mesh.cell, 4*fs.element.degree)
    fel = fs.element

    phi = fel.tabulate(qr.points)
    gradphi = fel.tabulate(qr.points, grad=True)

    no_gases = u.dim
    n_dof = fs.node_count
    no_cell_dofs = fel.node_count
    no_cells = fs.mesh.entity_counts[fs.mesh.dim]

    # cache some properties to allow direct numpy access rather than the
        slower
    # Python object access in the loop
    cell_nodes = np.copy(fs.cell_nodes)
    qr_weights = np.copy(qr.weights)
    jacobians = np.copy(fs.mesh.mesh_jacobians)
    cell_measures = np.copy(fs.mesh.cell_measures)

    row_array = []
    col_array = []
    data = []

    for cn in range(no_cells):
        jac_min = np.linalg.inv(jacobians[cn])
        meas = cell_measures[cn]
        unew_qp = np.einsum("qi,di->dq", phi, u.values[:, cell_nodes[cn]])
```

```
238         uold_qp = np.einsum("qi,di->dq", phi, uold.values[:, cell_nodes[cn]])
239         b_mat = assemble_mat_b_alternative(uold_qp, d_mat)
240         denom_term = (1 + np.einsum("dq->q", np.exp(unew_qp)))**(-1)
241         for j in range(no_gases):
242             for m in range(no_gases):
243                 row_array.extend(np.outer(
244                     n_dof*j+cell_nodes[cn],
245                     np.ones(no_cell_dofs, dtype=int)).flatten())
246                 col_array.extend(np.tile(
247                     n_dof*m+cell_nodes[cn],
248                     no_cell_dofs))
249                 data.extend(
250                     dt*meas*np.einsum(
251                         "q,ba,qkb,da,qmd,q->km", b_mat[:, j, m],
252                         jac_min, gradphi, jac_min, gradphi, qr_weights).
                            reshape(
253                         no_cell_dofs**2))
254                 if m == j:
255                     data[-no_cell_dofs**2:] += meas*np.einsum(
256                             "q,qi,q,qj,q->ji",
257                             denom_term, phi,
258                             np.exp(unew_qp[m, :]),
259                             phi, qr_weights).reshape(no_cell_dofs**2)

261                 data[-no_cell_dofs**2:] -= meas*np.einsum(
262                         "q,q,qi,q,qj,q->ji", denom_term**2,
263                         np.exp(unew_qp[m, :]), phi,
264                         np.exp(unew_qp[j, :]),
265                         phi, qr_weights).reshape(no_cell_dofs**2)

267     # convert to a CSR matrix
268     jac_mat = sparse.coo_matrix(
269         (data, (row_array, col_array)),
270         shape=(no_gases*n_dof, no_gases*n_dof)).tocsr()
271     return jac_mat
```

## References


[1] Alexandre, R.; Desvillettes, L.; Villani, C. and Wennberg, B.: Entropy dissipation and long-range interactions. *Arch. Ration. Mech. Anal.*, vol. 152, no. 4:(2000), pp. 327–355. ISSN 0003-9527. URL http://dx.doi.org/10.1007/s002050000083

[2] Alexandre, Radjesvarane; Morimoto, Yoshinori; Ukai, Seiji; Xu, Chao-Jiang and Yang, Tong: Regularizing effect and local existence for the non-cutoff Boltzmann equation. *Arch. Ration. Mech. Anal.*, vol. 198, no. 1:(2010), pp. 39–123. ISSN 0003-9527. URL http://dx.doi.org/10.1007/s00205-010-0290-1

[3] Auzinger, Winfried and Praetorius, Dirk: Numerische Mathematik (2011). Lecture notes (in German)

[4] Beazley, David: Inside the Python GIL (2009). Presentation, online available at http://www.dabeaz.com/python/GIL.pdf

[5] Behnel, S.; Bradshaw, R.; Citro, C.; Dalcin, L.; Seljebotn, D.S. and Smith, K.: Cython: The Best of Both Worlds. *Computing in Science Engineering*, vol. 13, no. 2:(2011), pp. 31 –39. ISSN 1521-9615. URL http://dx.doi.org/10.1109/MCSE.2010.118

[6] Bernis, Laurent and Desvillettes, Laurent: Propagation of singularities for classical solutions of the Vlasov-Poisson-Boltzmann equation. *Discrete Contin. Dyn. Syst.*, vol. 24, no. 1:(2009), pp. 13–33. ISSN 1078-0947. URL https://doi.org/10.3934/dcds.2009.24.13

[7] Blümlinger, Martin: Der Satz von Eberlein-Šmulian. Lecture Notes (in German), online available at http://www.asc.tuwien.ac.at/~blue/Eb-Smu.pdf

[8] Blümlinger, Martin: Analysis 3 (Version WS2017/18) (2017). Lecture notes (in German), online available at http://asc.tuwien.ac.at/~blue/Ana3.pdf

[9] Bogachev, V. I.: *Measure theory. Vol. I, II.* Springer-Verlag, Berlin (2007). ISBN 978-3-540-34513-8; 3-540-34513-2. URL http://dx.doi.org/10.1007/978-3-540-34514-5

[10] Bothe, Dieter: On the Maxwell-Stefan approach to multicomponent diffusion. In: *Parabolic problems*, Birkhäuser/Springer Basel AG, Basel, *Progr. Nonlinear Differential Equations Appl.*, vol. 80, pp. 81–93 (2011). URL http://dx.doi.org/10.1007/978-3-0348-0075-4_5

[11] Boudin, Laurent and Desvillettes, Laurent: On the singularities of the global small solutions of the full Boltzmann equation. *Monatsh. Math.*, vol. 131, no. 2:(2000), pp. 91–108. ISSN 0026-9255. URL https://doi.org/10.1007/s006050070015

[12] Boudin, Laurent; Grec, Bérénice and Pavan, Vincent: The Maxwell-Stefan diffusion limit for a kinetic model of mixtures with general cross sections. *Nonlinear Anal.*, vol. 159:(2017), pp. 40–61. ISSN 0362-546X. URL https://doi.org/10.1016/j.na.2017.01.010

[13] Boudin, Laurent; Grec, Bérénice and Salvarani, Francesco: A mathematical and numerical analysis of the Maxwell-Stefan diffusion equations. *Discrete Contin. Dyn. Syst. Ser. B*, vol. 17, no. 5:(2012), pp. 1427–1440. ISSN 1531-3492. URL http://dx.doi.org/10.3934/dcdsb.2012.17.1427

[14] Boudin, Laurent; Grec, Bérénice and Salvarani, Francesco: The Maxwell-Stefan Diffusion Limit for a Kinetic Model of Mixtures. *Acta Applicandae Mathematicae*, vol. 136, no. 1:(2015), pp. 79–90. ISSN 0167-8019. URL http://dx.doi.org/10.1007/s10440-014-9886-z

[15] Boudin, L.; Götz, D. and Grec, B.: Diffusion models of multicomponent mixtures in the lung. *ESAIM: Proc.*, vol. 30:(2010), pp. "90–103". URL http://dx.doi.org/"10.1051/proc/2010008"

[16] Breit, Dominic: Sobolev-Räume (2013). Lecture notes, online available at http://www.mathematik.uni-muenchen.de/~diening/ws14/seminar/breit-sobolev.pdf

[17] Brenner, Susanne C. and Scott, L. Ridgway: *The mathematical theory of finite element methods*, *Texts in Applied Mathematics*, vol. 15. Springer, New York, 3rd edn. (2008). ISBN 978-0-387-75933-3. URL http://dx.doi.org/10.1007/978-0-387-75934-0

[18] Calogero, Simone: Lecture Notes on Boltzmann's equation (2011). Lecture notes, online available at http://www.math.chalmers.se/~calogero/Boltzman_notes.pdf

[19] Curtiss, C F and Bird, R B: Multicomponent diffusion in polymeric liquids. *Proceedings of the National Academy of Sciences*, vol. 93, no. 15:(1996), pp. 7440–7445. ISSN 0027-8424. URL http://dx.doi.org/10.1073/pnas.93.15.7440

[20] Cuvelier, François; Japhet, Caroline and Scarella, Gilles: An efficient way to assemble finite element matrices in vector languages. *BIT Numerical Mathematics*, vol. 56, no. 3:(2016), pp. 833–864. ISSN 1572-9125. URL http://dx.doi.org/10.1007/s10543-015-0587-4

[21] Dalcin, Lisandro D.; Paz, Rodrigo R.; Kler, Pablo A. and Cosimo, Alejandro: Parallel distributed computing using Python. *Advances in Water Resources*, vol. 34, no. 9:(2011), pp. 1124 – 1139. ISSN 0309-1708. URL http://dx.doi.org/10.1016/j.advwatres.2011.04.013. New Computational Methods and Software Tools

[22] DiPerna, R. J. and Lions, P. L.: On the Cauchy Problem for Boltzmann Equations: Global Existence and Weak Stability. *Annals of Mathematics*, vol. 130, no. 2:(1989), pp. 321–366. ISSN 0003486X. URL http://www.jstor.org/stable/1971423

[23] Dreher, Michael and Jüngel, Ansgar: Compact families of piecewise constant functions in $L^p(0,T;B)$. *Nonlinear Analysis*. ISSN 0362-546X

[24] Duncan, J. B. and Toor, H. L.: An experimental study of three component gas diffusion. *AIChE Journal*, vol. 8, no. 1:(1962), pp. 38–41. ISSN 1547-5905. URL http://dx.doi.org/10.1002/aic.690080112

[25] Evans, Lawrence C.: *Partial Differential Equations*. Graduate Studies in Mathematics. American Mathematical Society, 2nd edn. (2010)

[26] Geiser, Jürgen: Iterative solvers for the Maxwell–Stefan diffusion equations: Methods and applications in plasma and particle transport. *Cogent Mathematics*, vol. 2, no. 1. URL http://dx.doi.org/10.1080/23311835.2015.1092913

[27] Gilbarg, David and Trudinger, Neil S.: *Elliptic partial differential equations of second order*. Classics in mathematics. Berlin: Springer, reprint of the 1998 ed. edn.. ISBN 3540411607

[28] Giovangigli, Vincent: *Multicomponent flow modeling*. Modeling and simulation in science, engineering and technology. Boston, Mass.: Birkhäuser. ISBN 9780817640484; 0817640487; 3764340487

[29] Gorelick, Micha and Ozsvald, Ian: *High performance Python*. Sebastopol, CA: O'Reilly Media, 1st edn.. ISBN 9781449361594; 1449361595

[30] Grad, Harold: Asymptotic equivalence of the Navier-Stokes and nonlinear Boltzmann equations. In: *Proc. Sympos. Appl. Math., Vol. XVII*, Amer. Math. Soc., Providence, R.I., pp. 154–183 (1965)

[31] Gressman, Philip T. and Strain, Robert M.: Global classical solutions of the Boltzmann equation with long-range interactions. *Proc. Natl. Acad. Sci. USA*, vol. 107, no. 13:(2010), pp. 5744–5749. ISSN 1091-6490. URL `https://doi.org/10.1073/pnas.1001185107`

[32] Havlicek, Hans: *Lineare Algebra für Technische Mathematiker*. Berliner Studienreihe zur Mathematik. Lemgo: Heldermann Verlag, 2nd edn. (2008)

[33] Horn, Roger A. and Johnson, Charles R.: *Matrix analysis*. Cambridge: Cambridge University Press (1985). ISBN 0-511-81081-4

[34] Jones, Eric; Oliphant, Travis; Peterson, Pearu et al.: SciPy: Open source scientific tools for Python. URL `http://www.scipy.org/`

[35] Jüngel, Ansgar: *Entropy methods for diffusive partial differential equations*. SpringerBriefs in Mathematics. Springer (2016). ISBN 978-3-319-34218-4; 978-3-319-34219-1. URL `https://doi.org/10.1007/978-3-319-34219-1`

[36] Jüngel, Ansgar: *Transport equations for semiconductors*. Lecture notes in physics ; 773. Berlin: Springer. ISBN 3540895256; 9783540895251

[37] Jüngel, Ansgar: The boundedness-by-entropy method for cross-diffusion systems. *Nonlinearity*, vol. 28, no. 6:(2015), pp. 1963–2001. URL `http://stacks.iop.org/0951-7715/28/i=6/a=1963`

[38] Jüngel, Ansgar: Nichtlineare partielle Differentialgleichungen (2016). Lecture notes (in German), online available at `http://www.asc.tuwien.ac.at/~juengel/scripts/nPDE.pdf`

[39] Jüngel, Ansgar: Partielle Differentialgleichungen (2017). Lecture notes (in German), online available at `http://www.asc.tuwien.ac.at/~juengel/scripts/PDE.pdf`

[40] Jüngel, Ansgar and Stelzer, Ines V.: Existence Analysis of Maxwell-Stefan Systems for Multicomponent Mixtures. *SIAM J. Math. Anal.*, vol. 45, no. 4:(2013), pp. 2421–2440. URL `http://dx.doi.org/10.1137/120898164`

[41] Knuth, Donald E.: Computer Programming As an Art. *Commun. ACM*, vol. 17, no. 12:(1974), pp. 667–673. ISSN 0001-0782. URL `http://dx.doi.org/10.1145/361604.361612`

[42] Kolesnikov, A. F. and Tirskii, G. A.: The Stefan-Maxwell equations for diffusion fluxes of plasma in a magnetic field. *Fluid Dynamics*, vol. 19, no. 4:(1984), pp. 643–649. ISSN 1573-8507. URL `http://dx.doi.org/10.1007/BF01091091`

[43] Krishna, R. and Wesselingh, J.A.: The Maxwell-Stefan approach to mass transfer. *Chemical Engineering Science*, vol. 52, no. 6:(1997), pp. 861 – 911. ISSN 0009-2509. URL http://dx.doi.org/10.1016/S0009-2509(96)00458-7

[44] Kusolitsch, Norbert: *Maß- und Wahrscheinlichkeitstheorie*. Wien: Springer-Verlag, 1st edn. (2011)

[45] Lions, P.-L.: On Boltzmann and Landau equations. *Philos. Trans. Roy. Soc. London Ser. A*, vol. 346, no. 1679:(1994), pp. 191–204. ISSN 0962-8428. URL https://doi.org/10.1098/rsta.1994.0018

[46] Logg, Anders; Mardal, Kent-Andre and Wells, Garth N. (Editors): *Automated solution of differential equations by the finite element method*, *Lecture Notes in Computational Science and Engineering*, vol. 84. Springer, Heidelberg (2012). ISBN 978-3-642-23098-1; 978-3-642-23099-8. URL http://dx.doi.org/10.1007/978-3-642-23099-8. The FEniCS book

[47] Maxwell, James C.: On the Dynamical Theory of Gases. *Philosophical Transactions of the Royal Society of London*, vol. 157:(1867), pp. 49–88. URL http://dx.doi.org/10.1098/rstl.1867.0004

[48] McLeod, Michael and Bourgault, Yves: Mixed finite element methods for addressing multi-species diffusion using the Maxwell–Stefan equations. *Computer Methods in Applied Mechanics and Engineering*, vol. 279:(2014), pp. 515 – 535. ISSN 0045-7825. URL http://dx.doi.org/10.1016/j.cma.2014.07.010

[49] Meurer, Aaron; Smith, Christopher P. et al.: SymPy: symbolic computing in Python. *PeerJ Computer Science*, vol. 3:(2017), p. e103. ISSN 2376-5992. URL http://dx.doi.org/10.7717/peerj-cs.103

[50] Meyer, C.D. and Stadelmaier, M.W.: Singular M-matrices and inverse positivity. *Linear Algebra and its Applications*, vol. 22:(1978), pp. 139 – 156. ISSN 0024-3795. URL http://dx.doi.org/https://doi.org/10.1016/0024-3795(78)90065-4

[51] Monroe, Charles W. and Newman, John: Onsager Reciprocal Relations for Stefan-Maxwell Diffusion. *Industrial & Engineering Chemistry Research*, vol. 45, no. 15:(2006), pp. 5361–5367. URL http://dx.doi.org/10.1021/ie051061e

[52] Oberkampf, William L. and Roy, Christopher J.: Exact solutions. In: *Verification and Validation in Scientific Computing*, Cambridge University Press, p. 208–248 (2010). URL http://dx.doi.org/10.1017/CBO9780511760396.009

[53] Rebhan, Eckhard: *Theoretische Physik: Thermodynamik und Statistik*. Berlin/Heidelberg: Springer (2010). ISBN 978-3-8274-2654-3

[54] Ribeiro, Cláudio P.; Freeman, Benny D. and Paul, Donald R.: Modeling of multicomponent mass transfer across polymer films using a thermodynamically consistent formulation of the Maxwell–Stefan equations in terms of volume fractions. *Polymer*, vol. 52, no. 18:(2011), pp. 3970 – 3983. ISSN 0032-3861. URL http://dx.doi.org/10.1016/j.polymer.2011.06.042

[55] Roache, P.J.: Code verification by the method of manufactured solutions. *Journal of Fluids Engineering, Transactions of the ASME*, vol. 124, no. 1:(2002), pp. 4–10. ISSN 00982202. URL http://dx.doi.org/10.1115/1.1436090

[56] Roubicek, Tomas: *Nonlinear partial differential equations with applications*. International series of numerical mathematics ; 153. Basel: Birkhäuser. ISBN 3764372931; 9783764372934

[57] Roy, Christopher J.: Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics*, vol. 205, no. 1:(2005), pp. 131 – 156. ISSN 0021-9991. URL http://dx.doi.org/10.1016/j.jcp.2004.10.036

[58] Saint-Raymond, Laure: *Hydrodynamic limits of the Boltzmann equation*. Lecture notes in mathematics ; 1971. Berlin: Springer (2009). ISBN 9783540928461. URL http://dx.doi.org/10.1007/978-3-540-92847-8

[59] Salari, Kambiz and Knupp, Patrick: Code verification by the Method of Manifactured Solutions. Tech. Rep., Sandia National Laboratories, Albuquerque (2000). URL https://www.osti.gov/scitech/servlets/purl/759450

[60] Schöberl, Joachim: Numerical Methods for Partial Differential Equations (2009). Lecture notes, online available at http://www.asc.tuwien.ac.at/~schoeberl/wiki/lva/notes/numpde.pdf

[61] Serre, Denis: *Matrices : theory and applications*. Graduate texts in mathematics. New York, NY: Springer (2002). ISBN 0387954600

[62] Sherman, Jack and Morrison, Winifred J.: Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *Ann. Math. Statist.*, vol. 21, no. 1:(1950), pp. 124–127. URL http://dx.doi.org/10.1214/aoms/1177729893

[63] Smith, Kurt W.: *Cython*. Sebastopol, California: O'Reilly Media. ISBN 1-4919-0176-4

[64] Stefan, Josef: Über das Gleichgewicht und die Bewegung, insbesondere die Diffusion von Gasmengen. *Sitzungsber. Akad. Wiss. Wien*, vol. 63:(1871), pp. 63–124

[65] Stelzer, Ines: *Existence analysis of cross-diffusion systems in biology*. Ph.D. thesis, Technische Universität Wien, Wien (2013). URL https://resolver.obvsg.at/urn:nbn:at:at-ubtuw:1-54056

[66] Sutherland, James C.: The Maxwell-Stefan Equations (2013). Lecture slides, online available at https://sutherland.che.utah.edu/6603Notes/GMS.pdf

[67] Taylor, R. and Krishna, R.: *Multicomponent mass transfer*. New York: Wiley (1993). ISBN 0-471-57417-1

[68] Villani, Cédric: Conservative forms of Boltzmann's collision operator: Landau revisited. *M2AN Math. Model. Numer. Anal.*, vol. 33, no. 1:(1999), pp. 209–227. ISSN 0764-583X. URL http://dx.doi.org/10.1051/m2an:1999112

[69] Wesselingh, J.A. and Krishna, R.: *Mass Transfer in Multicomponent Mixtures*. Delft: VSSD (2006). ISBN 90-71301-58-3

[70] Yang, Chaodong and Gu, Yongan: Minimum time-step criteria for the Galerkin finite element methods applied to one-dimensional parabolic partial differential equations. *Numerical Methods for Partial Differential Equations*, vol. 22, no. 2:(2005), pp. 259–273. URL http://dx.doi.org/10.1002/num.20097