Lehrstuhl für Technische Elektrophysik
Fakultät für Elektrotechnik und Informationstechnik
Technische Universität München

Numerische Weiterentwicklung des „Contamination-Flow“ Teilchentransportcodes zur Simulation von Kontaminationsübertrag in Vakuumanwendungen

Ingenieurpraxis

Autor: Berke Karakin

Betreuer: Rudolf Schönmann
Prüferin: Prof. Dr. rer. nat. habil. Gabriele Schrag

Datum der Abgabe: 14. März 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die optischen Instrumente sind das entscheidende Element für die Erdbeobachtungssatelliten in der Erdlaufbahn, die für die optische Fernerkundung eingesetzt sind, bzw. für die Raumsonden, die für die Weltraumforschung eingesetzt sind [1]. Insbesondere spielen die optischen Instrumente der Umweltsatelliten eine wichtige Rolle für die hochauflösende Beobachtung der Erde mittels Bildaufnahme von der Erdoberfläche [2]. Die erhaltenen Daten werden dann in verschiedenen Disziplinen ausgenutzt. Jedoch ist es bekannt, dass die Oberflächen der optischen Instrumente der Raumflugkörper empfindlich auf die flüchtigen Substanzen, die eine Kontamination verursachen, reagieren [3]. Der Grund für die Kontamination der Optiken, die nie komplett verhindert werden kann, sind die Wassermoleküle oder die organischen Materialien, die durch die Ausgasung der Satellitenkomponenten entstehen und an den kritischen Oberflächen adsorbieren [4]. Infolgedessen entsteht an den Oberflächen der optischen Instrumente eine Schicht von unerwünschten Substanzen, die die Leistung des Instruments erniedrigen kann. Diese Kontaminationsschicht kann sogar zum totalen Ausfall des Instruments führen und somit den Ausfall des Satelliten in seiner Laufbahn verursachen [3].

Das Programm „ContaminationFlow“ basiert auf dem Monte-Carlo-Simulator Molflow+, der für die Analyse und für den Entwurf der Hochvakuumsysteme und ihrer Komponenten dient [5]. ContaminationFlow erweitert dieses Programm mit einem Kontaminationsmodell, um die zeitliche Entwicklung der Kontaminationsschicht an den Oberflächen einer Geometrie, in der das Ultrahochvakuum herrscht, durch eine Monte-Carlo-Simulation abzuschätzen. Das Ziel bei der Entwicklung des Programms ist die Schaffung eines Tools zur Vorhersage der Kontamination und die Klassifizierung des Risikos, die bei dem Entwicklungsprozess der optischen Satelliten eingesetzt werden kann [6].

Wie es in folgenden Kapiteln detailliert erwähnt wird, berechnet das Programm die Bedeckung der Oberflächen in exponentiell wachsenden Zeitschritten. In bestimmten Fällen führt eine große Schrittweite zur Oszillation der Bedeckung, die dann zu unrealistischen Ergebnissen führt. In dieser Ingenieurpraxis geht es um die Erweiterung des iterativen Algorithmus von ContaminationFlow für die Berechnung der Bedeckung bzw. des Bedeckungsgrads mit dem Prädiktor-Korrektor-Verfahren, um diese Angelegenheit zu verhindern. Das Verfahren ist analog zu dem Heunverfahren zur numerischen Lösung eines Anfangswertproblems. Da aber das Programm

eigentlich keine Differenzialgleichung integriert, muss das Verfahren anders in den Code implementiert werden. In diesem Bericht wird auf zwei möglichen Ansätze für die Implementierung des Prädiktor-Korrektor-Verfahrens in dem Algorithmus eingegangen.

In Kapitel 2 wird ein Hintergrund gegeben, damit das Funktionsprinzip des Programms klar wird. In Kapitel 3 wird die Implementierung des Prädiktor-Korrektor-Verfahrens beschrieben. Anschließend werden in Kapitel 4 die neu implementierten Methoden miteinander und gegenüber dem vorherigen Verfahren verglichen und ausgewertet. Letztlich werden in Kapitel 5 die Ergebnisse zusammengefasst und ein Ausblick wird gegeben.

2 Hintergrund

In diesem Kapitel wird ein Hintergrund zu ContaminationFlow gegeben. Die hier vermittelten Informationen basieren auf der Doktorarbeit von Marton Ady [7] sowie auf der Dokumentation von ContaminationFlow.

2.1 ContaminationFlow

Das Programm ContaminationFlow basiert auf Molflow+. Molflow+ benutzt die Monte-Carlo-Methode, um die technisch relevante Größen für die Oberflächen eines beliebig komplizierten Vakuumsystems abzuleiten, deren analytische Berechnung schwer bis unmöglich ist. Als Beispiel zu diesen Größen kann der Druck, die Teilchendichte, die Adsorption, die Desorption sowie die Geschwindigkeit- und Winkelprofilen gegeben werden. Bei der Monte-Carlo-Methode wird nur eine beschränkte Anzahl der Teilchen, die als virtuelle Teilchen oder als Testteilchen bezeichnet wird, betrachtet und ihre Bewegung in einem System, in dem Ultrahochvakuum herrscht (korrespondiert zu einem Druckwert kleiner als 10^{-7} mbar), simuliert. Unter Anwendung der aus der Simulation erhaltenen Ergebnisse wird eine Abschätzung für die oben genannten technischen Größen gemacht. Übrigens, weil es in dem System Ultrahochvakuum herrscht, ist die mittlere freie Weglänge der Teilchen viel größer als die Größe der Geometrie. Das heißt, es herrscht näherungsweise die freie molekulare Strömung im Inneren des Systems. Als Konsequenz kann vereinfacht angenommen werden, dass einzelne Teilchen nicht mit anderen Teilchen in dem System stoßen, was unter normalem Druck sehr wahrscheinlich wäre. Diese Annahme erlaubt die Parallelisierung der Simulation, weil die simulierten Testteilchen während ihrer Flugzeit nur mit den Oberflächen der Geometrie wechselwirken. Außerdem ist die Simulation ereignisgesteuert. Das heißt, die physikalische Zeit in dem System vergeht nur nach bestimmten Ereignissen, wie zum Beispiel der Stoß mit einer Oberfläche.

ContaminationFlow erweitert Molfow+ mit einem Kontaminationsmodell, um die Kontamination an den kritischen Oberflächen der Satelliten abschätzen zu können. Infolgedessen wird zu den oben genannten relevanten Größen auch die Bedeckung (θ^*) bzw. der Bedeckungsgrad (θ) hinzugefügt. Mit der Bedeckung wird die Anzahl der Teilchen, die sich an einer Oberfläche der Geometrie befinden, ausgedrückt bzw. mit der Bedeckungsgrad wird die Anzahl der Schichten, die die Teilchen an einer Oberfläche bilden, ausgedrückt. Für die Modellierung der Kontamination wird ei-

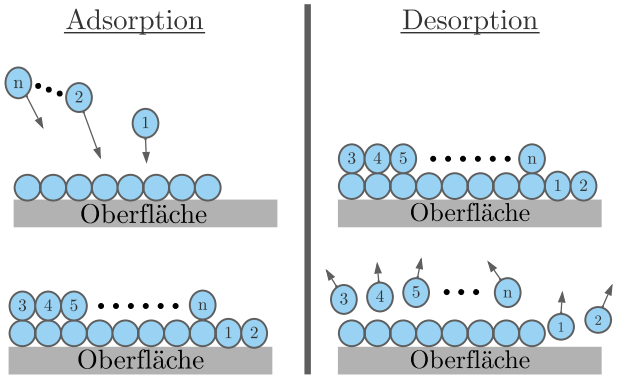


Abbildung 2.1: Kontaminationsmodell

ne Vereinfachung gemacht und es wird davon ausgegangen, dass alle Teilchen, die an einer Oberfläche adsorbieren, zuerst eine Kontaminationsschicht komplett füllen, bevor sie mit der nächsten Schicht weitermachen. Im Falle der Desorption können zuerst die oberste Teilchen desorbieren. Dieser Fall wird in der Abbildung 2.1 illustriert. Die Satelliten können in ihrer Laufbahn für mehrere Jahre tätig sein [3]. Aus diesem Grund muss das Programm eine physikalische Zeit von mehreren Jahren simulieren. Um diese Simulation rechnerisch zu ermöglichen, werden die Zeitschritte exponentiell wachsend gewählt. Das Programm benutzt mehrere Prozesse. Somit können innerhalb der gleichen Rechenzeit mehr Testteilchen simuliert werden. Da die Monte-Carlo-Methode ein stochastisches Verfahren ist, wird die Genauigkeit der Ergebnisse mit der zunehmenden Anzahl der simulierten Testteilchen auch zunehmen [8]. Die Prozesse in dem Programm werden in Hauptprozess und Subprozesse unterteilt. Die K verschiedenen Subprozesse sind für die Simulation der Testteilchen zuständig. Sie benutzen keine gemeinsamen Variablen, das heißt, dass jeder Subprozess eine andere Kopie des Programms durchführt. Während der Simulation der Testteilchen in den Subprozessen werden bestimmten Zählern je nach dem geschehenen Ereignis gezählt. Diese Zähler werden dann für die Berechnung der physikalischen Größen gemäß den entsprechenden Formeln, die hier nicht erwähnt werden, zum Hauptprozess geschickt. Es soll beachtet werden, dass in den Subprozessen im Allgemeinen nicht so viele Teilchen, wie sie sich in einem realen System befinden, simuliert werden können. Deswegen wird angenommen, dass jedes simulierte Teilchen mehrere realen Teilchen darstellt. Der Faktor, der dieses Verhältnis beschreibt, wird $K_{real/virtual}$ genannt. Am Ende der Subprozesssimulationen werden ihre Zählerwerte mit diesem Faktor multipliziert, um auf dem realen Ergebnis zu kommen. Der Hauptprozess dient für die Steuerung des Programms sowie für die Akkumulation (Summierung) der in den Subprozesssimulationen erhaltenen Zählerwerte. Am Anfang des Programms werden die Anfangsbedeckungen sowie die Geometrieparameter von dem Hauptprozess zu den Subprozessen geschickt. Am Ende der Subprozesssimulationen werden die erhaltenen Zählerwerte mit dem Faktor

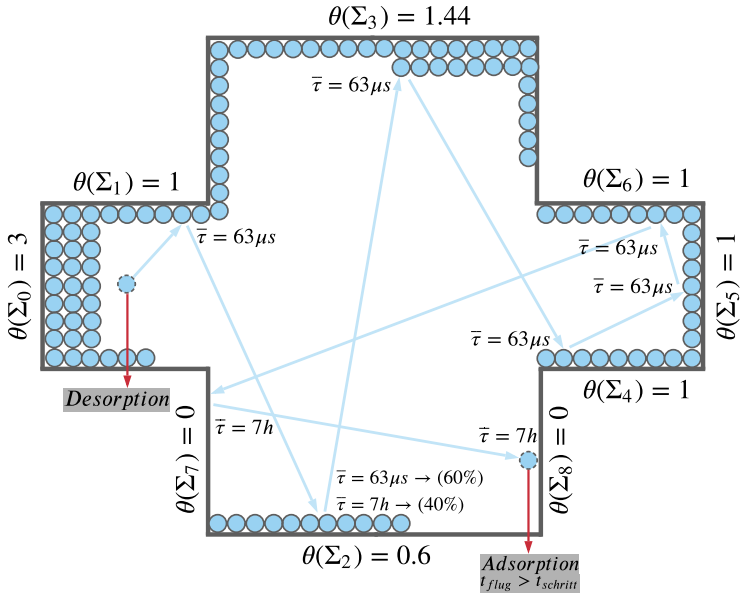


Abbildung 2.2: Simulation eines Testteilchens in einem Subprozess

$K_{real/virtual}$ skaliert. Danach wird der tatsächliche Wert der oben genannten physikalischen Größen für den nächsten Zeitpunkt berechnet. Im Falle der Bedeckung wird der Zählerwert zuerst vom Zählerwert am Anfang des Zeitschritts subtrahiert und diese Differenz wird mit dem Faktor skaliert. Letztlich wird die skalierte Differenz zur Bedeckung am Anfang des Zeitschritts addiert. Schließlich werden die Subprozessen mit den neu berechneten Größen aktualisiert und das Programm macht mit der nächsten Iteration weiter.

2.2 Simulation eines Testteilchens

In der Abbildung 2.2 wird die Simulation eines Testteilchens für eine Schrittweite von 10 Stunden grob skizziert. Dabei wurden die Geometrie und ihre Bedeckungsgrade beispielhaft gewählt. Am Anfang wird ein Testteilchen in einer zufällig gewählten Oberfläche (in der Abbildung ist sie die Oberfläche Σ_0) generiert. Die Wahrscheinlichkeit für das Wählen einer bestimmten Oberfläche hängt von ihrer Desorption, die von ihrer Bedeckung abhängt, und von ihrer Ausgasung ab. Nach der Wahl der Oberfläche wird der Startpunkt des Testteilchens an dieser Oberfläche ausgesucht. Die Zufallsvariablen für das Aussuchen eines Startpunkts sind gleichverteilt. Das heißt, jeder Punkt kann mit gleicher Wahrscheinlichkeit gewählt werden. Anschließend wird abhängig von der Desorption und von der Ausgasung der gewählten Oberfläche zufällig entschieden, ob das generierte Testteilchen durch Ausgasung oder durch Desorption entsteht. Diese Unterscheidung ist wichtig, denn im Falle einer Desorption wird davon ausgegangen, dass das Testteilchen aus der Kontaminati-

onsschicht stammt und der Zähler für die Bedeckung der Oberfläche wird um eins erniedrigt. Im Falle der Ausgasung wird angenommen, dass das Teilchen außerhalb des Systems stammt und der Zähler für die Bedeckung bleibt gleich. Nach dieser Unterscheidung wird noch die Verweilzeit des Testteilchens an der Oberfläche zu seiner gesamten Flugzeit addiert und damit die Startzeit des Teilchens bestimmt. In diesem Bericht wird auf die Berechnung der Startzeit nicht eingegangen, weil sie komplex ist. Jedoch kann angenommen werden, dass für einen Bedeckungsgrad größer als zwei jedes desorbierende Teilchen an anderen Adsorbat gebunden ist. Damit berechnet sich die Startzeit mit einer kurzen Verweilzeit. Wenn der Bedeckungsgrad kleiner als eins ist, wird angenommen, dass jedes desorbierende Teilchen an dem Adsorbens gebunden ist. Damit berechnet sich die Startzeit mit einer langen Verweilzeit. Ansonsten wird zufällig entschieden, ob das desorbierende Teilchen mit dem Adsorbens oder mit dem Adsorbat gebunden ist. Denn bei einem Bedeckungsgrad zwischen eins und zwei sind die beiden Bindungstypen des desorbierenden Teilchen möglich. Letztlich wird die Richtung des Testteilchens nach dem Knudsen-Kosinus-Gesetz und die Geschwindigkeit des Teilchens nach der Maxwell-Boltzmann-Verteilung zufällig entschieden. Als Nächstes findet der Raytracing-Algorithmus die nächste Oberfläche, an der das Testteilchen stoßen wird. Die Zeit, die unterwegs vergeht, wird zu der Flugzeit des Testteilchens addiert. Nach dem Stoß mit der Oberfläche verweilt das Testteilchen für eine bestimmte Zeit. Wie es in der Abbildung gezeigt wird, hängt die Verweilzeit von dem Bedeckungsgrad der Oberfläche ab. Bei einem Bedeckungsgrad größer als eins bindet das Testteilchen mit dem Adsorbat. Dagegen wird bei einem Bedeckungsgrad kleiner als eins zufällig entschieden, ob das Teilchen mit dem Adsorbens (Oberfläche) oder mit dem Adsorbat bindet. Wenn das Testteilchen direkt mit der Oberfläche bindet, dann wird die Bindungsenergie zwischen dem Adsorbens und dem Teilchen (E_{de}) als die Aktivierungsenergie benutzt. Im Falle der Bindung zwischen einem Wassermolekül und Edelstahl korrespondiert ihre Bindungsenergie (1 eV) [9] bei einer Temperatur von 20 °C zu einer mittleren Verweilzeit ($\bar{\tau}$) von ca. 7 Stunden. Für kurze Schrittweiten wäre die Annahme, dass das Testteilchen an der Oberfläche stecken bleibt, nicht falsch. Anderseits, falls das Testteilchen mit dem Adsorbat bindet, dann wird die Verdampfungsenthalpie (H_{vap}) als die Aktivierungsenergie benutzt. Im Falle einer Bindung zwischen zwei Wassermolekülen, korrespondiert ihre Bindungsenergie (0.5 eV) [9] bei einer Temperatur von 20 °C zu einer mittleren Verweilzeit von ca. 63 µs. Das heißt, dass das Testteilchen nach einer sehr kurzen Zeit wieder frei wird. Nach dem Addieren der Verweilzeit zu der Flugzeit des Testteilchens werden wieder seine Richtung und seine Geschwindigkeit entschieden. Der gleiche Prozess wiederholt sich, bis die Flugzeit des Testteilchens größer als die Schrittweite ist. In diesem Fall endet seine Simulation. Für den nächsten Zeitpunkt adsorbiert es an der Oberfläche, an der es gerade befindet (in der Abbildung ist sie die Oberfläche Σ_8) und der Zähler für die Bedeckung dieser Oberfläche erhöht um eins. Anschließend wird mit der Simulation eines anderen Testteilchens weitergemacht, bis es genug Testteilchen simuliert wurden.

3 Durchführung

In diesem Kapitel geht es um die Implementierung des Prädiktor-Korrektor-Verfahrens in den Simulationscode. Zuerst wird auf das Oszillationsproblem der Simulationsergebnisse eingegangen. Danach wird die Motivation für die Wahl des Prädiktor-Korrektor-Verfahrens erklärt. Diesbezüglich werden das explizite Eulerverfahren und das Heunverfahren, die zur numerischen Lösung eines Anfangswertproblems verwendet werden kann [10], miteinander verglichen. Schließlich wird die tatsächliche Implementierung des Prädiktor-Korrektor-Verfahrens in den Simulationscode beschrieben. Dazu wird auf zwei verschiedenen Ansätzen eingegangen.

3.1 Oszillation der Simulationsergebnisse

Für die Darstellung des Oszillationsproblems wird eine einfache Würfelgeometrie betrachtet, wobei der Anfangsbedeckungsgrad der 0. Oberfläche 5.0 ist und die Anfangsbedeckungsgrade der anderen Oberflächen 0 sind. Es wird mindestens 10^4 Testteilchen simuliert. Außerdem gasen keine Teilchen aus den Oberflächen aus. In diesem Fall ist zu erwarten, dass die Bedeckungsgrade aller Oberflächen im Gleichgewicht auf $0.8\overline{3}$ annähern. Dieser Fall ist in der Abbildung 3.1 auf der Seite 12 zu sehen. Die Abbildung zeigt den Bedeckungsgrad der 0. Oberfläche abhängig der Zeit. Hierbei wird eine logarithmische Zeitachse gewählt und die Simulation wird für zwei Jahren mit 1000 Iterationen ausgeführt. Daneben wird auf die zeitabhängigen Bedeckungsgrade der anderen Oberflächen nicht eingegangen, weil sie analog zur 0. Oberfläche sind. Die Abbildung zeigt, dass ab dem Zeitpunkt 10^6 s, der zum 116. Tag des simulierten Systems korrespondiert, der Bedeckungsgrad stark oszilliert. Dieses Verhalten ist unphysikalisch. Die Schrittweite in diesem Zeitpunkt beträgt ungefähr 18 Stunden. Die Ursache für dieses Verhalten ist nicht einfach zu erklären. Die einfachste Begründung wäre, dass die Monte-Carlo-Methode seine Ursache war, weil sie ein stochastisches Verfahren ist und die Ergebnisse immer eine Ungenauigkeit enthalten. Da am Ende der Subprozesssimulationen die Ergebnisse noch mit dem Faktor $K_{real/virtual}$ skaliert werden, werden auch die Ungenauigkeiten mitskaliert. Aus diesem Grund ist zu überlegen, dass diese Sache der Anlass der Oszillation ist. Aber in diesem Fall ist zu erwarten, dass das Problem mit der Zunahme der Stichprobe gelöst wird. Jedoch passiert das tatsächlich nicht. In der Tat bleibt der Fehler in der gleichen Größenordnung, wenn die Anzahl der simulierten Testteilchen verzehnfacht (10^5 Testteilchen) oder verhundertfacht (10^6 Testteilchen) wird.

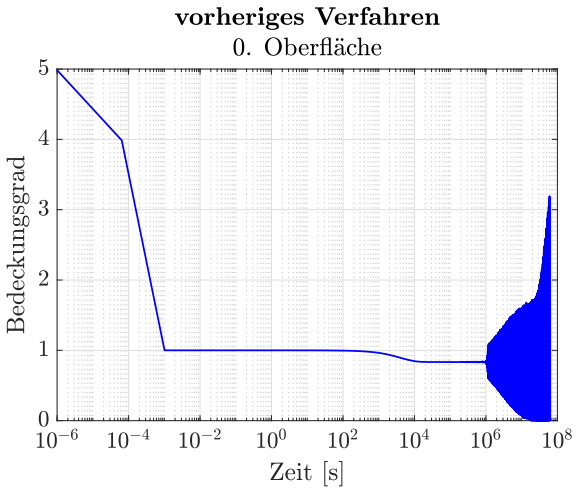


Abbildung 3.1: Oszillation bei größeren Schrittweiten

Diese Angelegenheit legt die Vermutung nahe, dass ein Effekt gibt, die die homogene Verteilung der Testteilchen in dem System verhindert. In dem betrachteten Beispiel kann davon ausgegangen werden, dass die Schrittweite in dem Zeitpunkt des Oszillationsbeginns so lang ist, dass ungefähr alle Teilchen an den Oberflächen desorbieren. Wenn aber eine Oberfläche wegen der statistischen Ungenauigkeiten ein bisschen größere Bedeckung hat, dann adsorbiert sie weniger Teilchen. Wogegen, wenn eine Oberfläche ein bisschen niedrigere Bedeckung hat, dann adsorbiert sie mehr Teilchen. In großen Schrittweiten führt diese Asymmetrie irgendwie zu einem selbstverstärkenden Effekt, sodass der in der Abbildung illustrierte Verhalten beobachtet wird. Übrigens ist es möglich, dass die Bedeckung wegen der Oszillation negativ ist. In diesem Fall wird die negative Bedeckung mit 0 ersetzt. Das führt zur Zunahme der Gesamtteilchenzahl in dem System. Die Zunahme der Gesamtteilchenzahl kann mit der Erhöhung der Anzahl der Testteilchen verlangsamt werden.

3.2 Motivation für das Prädiktor-Korrektor-Verfahren

Betrachtet wird ein Anfangswertproblem in der Form:

$$y'(t) = f(t, y(t))$$
$$y(t_0) = y_0 \text{ und } y : \mathbb{R} \to \mathbb{R}$$

Die Abbildung 3.2 zeigt die zwei verschiedenen Algorithmen, die zur numerischen Lösung des oben gezeigten Anfangswertproblems benutzt werden können, und deren Anwendung auf einer Beispielfunktion. Die Verfahren approximieren die Funktion in den diskreten Zeitpunkten. Der linke Graph illustriert die Anwendung des expliziten Eulerverfahrens für einen Schritt und dabei entstandener Diskretisierungsfehler. Zudem ist der Punkt (t_j, y_j), was in dem letzten Schritt berechnet wurde. Es wird davon

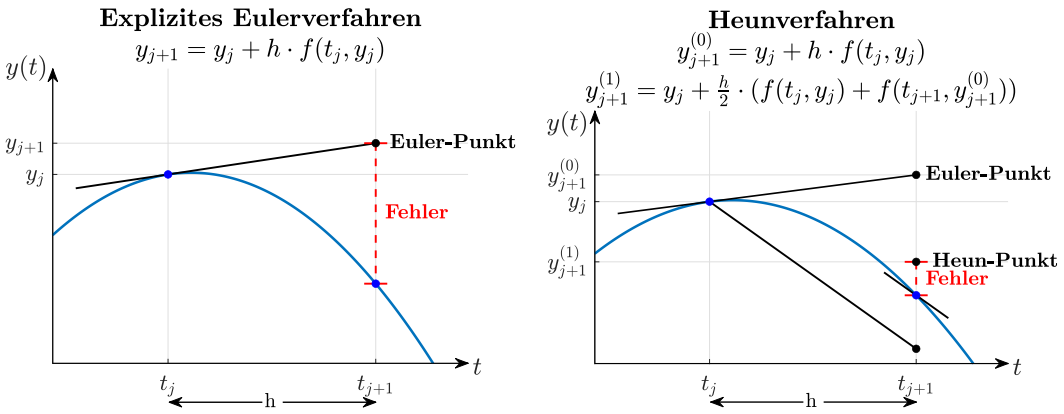


Abbildung 3.2: Vergleich vom expliziten Eulerverfahren und Heunverfahren

ausgegangen, dass das Verfahren in dem letzten Schritt eine perfekte Approximation ($e_j = |y(t_j) - y_j| = 0$) machte. Außerdem wird die Schrittweite h trotz der schnellen Dynamik der Funktion groß gewählt. Bei dem Eulerverfahren wird die Approximation für den nächsten Schritt (t_{j+1}, y_{j+1}) unter Anwendung der im linken Graph gezeigten Formel berechnet. Die Formel kann geometrisch so interpretiert werden, dass aus $f(t, y(t))$ die Steigung der Funktion in dem Punkt (t_j, y_j) berechnet wird. Diese Steigung wird dann bis zum nächsten Zeitpunkt gefolgt und der Wert der Funktion in diesem Zeitpunkt mit dem y-Wert des erreichten Punkts approximiert. In dem rechten Graph wird die Anwendung des Heunverfahrens (Prädiktor-Korrektor-Verfahren mit einem Korrektor-Schritt) für einen Schritt und der dabei entstandener Diskretisierungsfehler dargestellt. Die Formel, die im rechten Graph gezeigt ist, kann geometrisch unterschiedlich interpretiert werden. Eine dieser Interpretationen, auf der die Implementierung des Verfahrens in den Simulationscode basiert, wäre, dass es zuerst das explizite Eulerverfahren verwendet wird. Der Punkt, der dadurch erreicht wird, wird für die Approximation der Steigung der Funktion in dem nächsten Zeitpunkt benutzt. Dieser Schritt wird als Prädiktor-Schritt genannt. Da das Eulerverfahren Fehler enthält, weicht die berechnete Steigung von der tatsächlichen Steigung ab. Aus diesem Grund entspricht die Gerade, die durch den Punkt (t_{j+1}, y_{j+1}) geht, keine genaue Tangente. In dem Korrektor-Schritt wird die neu berechnete Steigung vom Ausgangspunkt bis zum nächsten Zeitpunkt gefolgt. Schließlich werden die y-Werte von dem erreichten Punkt und der Punkt aus dem Eulerschritt gemittelt und der Wert der Funktion in dem nächsten Zeitpunkt mit dem y-Wert des neuen Punkts approximiert. Es können auch mehrere Korrektor-Schritte ausgeführt werden, um die Genauigkeit des Verfahrens zu verbessern [11]. In der Abbildung ist zu merken, dass bei dem Heunverfahren der Approximationswert in zwei Schritten berechnet wird. Zwar ist die Rechenzeit dadurch länger, aber der Fehler, der bei der Approximation entsteht, ist deutlich kleiner. Demgegenüber ist der Fehler, der bei dem expliziten Eulerverfahren entsteht, viel größer. Wenn trotz dieses Fehlers

mit dem Verfahren weitergemacht würde, dann wäre es sehr wahrscheinlich, dass der Fehler in den nächsten Zeitpunkten immer größer wird. Das bedeutet, dass das Verfahren mit wachsender Amplitude oszilliert und unvernünftige Ergebnisse gibt. Obwohl das Heunverfahren im Allgemeinen bessere Ergebnisse liefert, ist es nicht garantiert, dass es immer so sein wird. Es können auch Anfangswertprobleme konstruiert werden, für die das explizite Eulerverfahren besser geeignet ist. Jedoch ist es garantiert, dass der Fehler des Heunverfahrens quadratisch abnimmt ($\mathcal{O}(h^2)$), wenn die Schrittweite verkleinert wird. Dagegen nimmt der Fehler des Eulerverfahrens mit der abnehmenden Schrittweite nur linear ab ($\mathcal{O}(h)$) [10].

3.3 Implementierung

Im Abschnitt 3.2 wurde gezeigt, dass das Prädiktor-Korrektor-Verfahren gegenüber dem Eulerverfahren genauere Ergebnisse geben kann. Dieses Ereignis war die Motivation für die Implementierung des Prädiktor-Korrektor-Verfahrens in den Simulationscode. Damit wurde gehofft, dass das neue Verfahren das Oszillationsproblem verbessert. Im Kontext der Simulation wird in der Gegenwart eine Methode ähnlich zum expliziten Eulerverfahren benutzt, um die Bedeckung für den nächsten Zeitschritt zu berechnen. Das heißt, die Bedeckungswerte, die nach den Subprozesssimulationen berechnet werden, werden als die endgültige Bedeckung für den nächsten Zeitschritt akzeptiert. Anderseits, im Prädiktor-Korrektor-Verfahren werden diese Bedeckungswerte als die Werte des Prädiktor-Schritts verwendet. Unter Anwendung dieser Werte wird die Simulation für den derzeitigen Zeitschritt noch einmal ausgeführt und damit wird der Korrektor-Schritt realisiert. Da das Prädiktor-Korrektor-Verfahren im Vergleich zum Eulerverfahren den Approximationswert in zwei Schritten berechnet, verdoppelt sich die Rechenzeit der Simulation. Für die Implementierung des Prädiktor-Korrektor-Verfahrens in den Code wird in der Hauptschleife eine zusätzliche Schleife definiert. Die sogenannte Prädiktor-Korrektor-Schleife kann gegebenenfalls mehrmals durchgelaufen werden (analog zum Prädiktor-Korrektor-Verfahren mit mehreren Korrektor-Schritten). Jedoch wurde festgestellt, dass es keinen signifikanten Vorteil bringt. Im Prädiktor-Schritt werden identisch zum vorherigen Verfahren zuerst die relevanten Größen, die für den Anlauf der Simulation notwendig sind, berechnet. Diese Größen sind die Schrittweite zwischen dem derzeitigen Zeitpunkt und dem nächsten Zeitpunkt, für den die Bedeckung sowie die anderen physikalischen Größen berechnet wird, die Desorption und die Ausgasung pro Oberfläche, sowie die anderen notwendigen Parameter, auf die hier nicht eingegangen wird. Anschließend werden die Subprozesssimulationen, wie es in Kapitel 2 erklärt wurde, ausgeführt. Im Gegensatz zum vorherigen Verfahren werden die Bedeckungswerte am Ende der Subprozesssimulationen nicht als die endgültigen Werte berücksichtigt. Im vorherigen Verfahren werden diese Werte direkt in der Liste `coveringList.currentList` gespeichert. Anschließend

wird mit der Simulation des nächsten Zeitschrittes weitergemacht. Stattdessen wird im Prädiktor-Korrektor-Verfahren die neuen Bedeckungswerte als Schätzwerte benutzt und die gleiche Iteration wird noch einmal durchgeführt. Zu diesem Zweck werden die Schätzwerte nicht in `currenList`, sondern in einer analogen Liste, die als `predictList` genannt wird, gespeichert. Somit ist sichergestellt, dass die Bedeckungswerte des gegenwärtigen Zeitpunkts im Korrektor-Schritt sich nicht ändern und die gleiche Iteration problemlos noch einmal durchgeführt werden kann. Diesbezüglich werden die für die Simulation nötigen Parameter im Korrektor-Schritt nicht noch einmal berechnet. Ferner, im Korrektor-Schritt, werden, wie es in der Abbildung 3.3 auf der Seite 17 gezeigt wird, in der `PerformBounce()` Funktion, die den Stoß und die Reflexion des Testteilchens aus einer Oberfläche simuliert, in der ersten Hälfte der Schrittweite ($t_{flug} \leq \frac{t_{schritt}}{2}$), die Bedeckungsgrade am Anfang des Zeitschritts verwendet. Das heißt, sie werden aus `currentList` gelesen. Demgegenüber wird in der zweiten Hälfte des Zeitschritts ($t_{flug} > \frac{t_{schritt}}{2}$) die Schätzwerte, die am Ende des Prädiktor-Schritts berechnet wurden, als der Bedeckungsgrad der Oberflächen benutzt. Das heißt, sie werden aus `predictList` gelesen. Am Ende des Korrektor-Schritts werden die neu berechneten Bedeckungswerte als die endgültigen Werte für den nächsten Zeitpunkt berücksichtigt, in `currentList` gespeichert und das Programm geht mit der Ausführung der nächsten Iteration vor. Es ist aber wichtig, wie die am Ende des Prädiktor-Schritts und am Ende des Korrektor-Schritts berechneten Werte in den jeweiligen Listen gespeichert werden. Dazu werden, wie es in der Abbildung 3.4 auf der Seite 17 gezeigt wird, auf zwei verschiedenen Ansätzen (P.K. Verfahren v1 und v2) eingegangen. Im ersten Ansatz werden die Bedeckungen, die am Ende des Prädiktor-Schritts berechnet wurden, und die Bedeckungen, die schon in `currentList` befinden, arithmetisch gemittelt und in `predictList` gespeichert. Dabei enthält die Variable `covering_phys` die neu berechnete Bedeckung am Ende der Iteration. Als Nächstes wird die gleiche Iteration mit dem Korrektor-Schritt wiederholt. Die Bedeckungen, die am Ende des Korrektor-Schritts erhalten werden, werden dann direkt in `currentList` gespeichert. Im zweiten Ansatz, werden die Bedeckungen, die am Ende des Prädiktor-Schritts berechnet wurden, direkt in `predictList` gespeichert und die Bedeckungen, die am Ende des Korrektor-Schritts erhalten wurden, werden mit den Bedeckungen in `predictList` arithmetisch gemittelt und in `currentList` gespeichert. Es ist leicht zu merken, dass die zweite Variante die genaue Implementierung des Heunverfahrens entspricht. In Kapitel 4 wird gezeigt, dass der zweite Ansatz tatsächlich besser funktioniert. Jedoch ist der erste Ansatz sicherer, weil es keinen direkten Einfluss auf die Bedeckungen, die am Ende der Subprozesssimulationen erhalten werden, gibt. Denn sie werden am Ende des Korrektor-Schritts direkt in `currentList` gespeichert. Demgegenüber wird im zweiten Ansatz die erhaltenen Bedeckungen am Ende des Verfahrens arithmetisch gemittelt. Solange die Simulationsergebnisse vernünftig sind, das bedeutet, dass es keine starke Oszillation zu bemerken ist, ist das kein Problem. Aber im anderen Fall ist es in bestimmten Fällen möglich, dass die Bedeckungsgrade aller

Oberflächen, wegen der zunehmenden Gesamtteilchenzahl, größer als eins werden. In diesem Fall ist die mittlere Verweilzeit bei der Simulation eines Testteilchens in der Größenordnung von µs wobei die Schrittweite des aktuellen Zeitschritts in der Größenordnung von mehreren Tagen ist. Als Konsequenz friert das Programm gewissermaßen ein, weil eine Schrittweite von mehreren Tagen nur mit µs-Schritten simuliert wird. Um dieses Ereignis zu verhinden, kann die Anzahl der simulierten Testteilchen erhöht werden. Übrigens ist es nicht garantiert, dass das Prädiktor-Korrektor-Verfahren überhaupt ein Vorteil gegenüber dem vorherigen Verfahren bringt. Wie es im nächsten Kapitel erwähnt wird, gibt es Anfangsprobleme, bei denen das vorherige Verfahren in größeren Schrittweiten tolerierbare Ergebnisse gibt. Die Anwendung des Prädiktor-Korrektor-Verfahrens für solche Anfangsprobleme ist nicht vernünftig, weil es unnötiger Aufwand, wegen der verdoppelten Rechenzeit, verursacht. Aus diesem Grund ist dessen Verwendung optional. Es wird nur dann benutzt, wenn der Parameter `usePCMethod` explizit in der Eingangsdatei auf eine Zahl ungleich `0` gesetzt wird. Letztlich fasst die Tabelle 3.1 auf der Seite 18 alle Erweiterungen zum Simulationscode zusammen.

```
1  if (simHistory->pcStep == 0) {
2      //Im Praediktor-Schritt
3      coverage = calcCoverage(iFacet);
4  } else if (simHistory->pcStep == 1
5              && flightTime <= simHistory->stepSize/2) {
6      //Im Korrektor-Schritt und in der ersten Haelfte des Zeitschritts
7      coverage = calcCoverage(iFacet);
8  } else if (simHistory->pcStep == 1
9              && flightTime > simHistory->stepSize/2) {
10     //Im Korrektor-Schritt und in der zweiten Haelfte des Zeitschritts
11     coverage = calcPredictedCoverage(iFacet);
12 }
```

Abbildung 3.3: Erweiterung von PerformBounce() mit P.K. Verfahren

```
1  if (p->usePCMethod == 0) { //P.K. Verfahren wird nicht benutzt
2    simHistory->coveringList.setCurrent(&f, (covering_phys));
3  }
4  else if (p->usePCMethod == 1) { //P.K. Verfahren v1 wird benutzt
5    if (simHistory->pcStep == 0) { //Im Praediktor-Schritt
6      boost::multiprecision::uint128_t currentVal;
7      currentVal = simHistory->coveringList.getCurrent(&f))
8      simHistory->coveringList.setPredict(&f, (covering_phys
9                                              + currentVal)/2);
10   }
11   else if (simHistory->pcStep == 1) { //Im Korrektor-Schritt
12     simHistory->coveringList.setCurrent(&f, covering_phys);
13   }
14 else { //P.K. Verfahren v2 wird benutzt
15   if (simHistory->pcStep == 0) { //Im Praediktor-Schritt
16     simHistory->coveringList.setPredict(covering_phys);
17   }
18   else if (simHistory->pcStep == 1 { //Im Korrektor-Schritt
19     boost::multiprecision::uint128_t predictVal;
20     predictVal = simHistory->coveringList.getPredict(&f))
21     simHistory->coveringList.setCurrent(&f, (covering_phys
22                                             + predictVal)/2);
23   }
24 }
```

Abbildung 3.4: Erweiterung von UpdateCovering() mit P.K. Verfahren

molflowlinux_main.cpp	
main()	Erweitert mit der Prädiktor-Korrektor-Schleife

SimulationLinux.h	
HistoryList	
predictList	Die Liste, die die Bedeckungswerte für P.K. Verfahren enthält
HistoryList()	Erweitert mit der Initialisierung von predictList als leere Liste
reset()	Erweitert mit der Rücksetzung von predictList
initPredict()	Initialisierung von predictList mit Nullen
printPredict()	Drucke predictList auf das Terminal
setPredict()	Lege den Wert der gewünschten Oberfläche in predictList fest
getPredict()	Bekomme den Wert der gewünschten Oberfläche von predictList

SimulationLinux.cpp	
ProblemDef	
usePCMethod	1, 2: Benutze das P.K. Verfahren v1, v2; 0: Benutze es nicht
ProblemDef()	Erweitert mit der Initialisierung von usePCMethod mit 0
readInputFile()	Lese den Wert für usePCMethod aus der Eingangsdatei
printInputFile()	Drucke usePCMethod in der Ausgangsdatei
SimulationHistory	
pcStep	Gegenwärtiger Schritt des P.K. Verfahrens
SimulationHistory()	Erweitert mit der Initialisierung von pcStep mit 0
updateHistory()	Erweitert mit pcStep

SimulationCalc.cpp	
getPredictedCovering()	Bekomme die Bedeckung von predictList
calcPredictedCoverage()	Berechne den Bedeckungsgrad aus der Bedeckung in predictList

SimulationMC.cpp	
PerformBounce()	Siehe Abbildung 3.3 auf der Seite 17

UpdateMainProcess.cpp	
UpdateCovering()	Siehe Abbildung 3.4 und Abbildung 3.4 auf der Seite 17

Tabelle 3.1: Erweiterungen zum Simulationscode

4 Auswertung

In diesem Kapitel werden die Simulationsergebnisse gezeigt und damit die zwei verschieden Ansätze für die Implementierung des Prädiktor-Korrektor-Verfahrens miteinander und gegenüber dem vorherigen Verfahren verglichen.

4.1 Vorabinformationen

Alle Simulationen wurden in einer Würfelgeometrie, die in der Abbildung 4.1 skizziert ist, durchgeführt. Die Wahl so einer einfachen Geometrie ermöglicht, dass die Ergebnisse im Voraus sehr einfach analytisch berechnet werden kann, um sie mit den Simulationsergebnissen zu vergleichen. Die Simulationszeit beträgt 2 Jahre und es werden 1000 Iterationen benutzt. Pro Iteration werden mindestens 10^4 Testteilchen simuliert. Als der Anfangsbedeckungsgrad der 0. Oberfläche wird der Wert θ_0, der größer als 0 ist, gewählt. Wogegen die Anfangsbedeckungsgrade der anderen Oberflächen als 0 gewählt werden. Es werden die Simulationsergebnisse von drei verschiedenen Anfangswertproblemen gezeigt. Der Anfangsbedeckungsgrad der 0. Oberfläche wird in dem ersten Anfangswertproblem auf 2.0, in dem zweiten Anfangswertproblem auf 3.5 und in dem letzten Anfangswertproblem auf 5.0 gesetzt. Die Temperatur aller Oberflächen beträgt 20 °C. Außerdem gast keine Oberfläche aus. Das heißt, die Gesamtteilchenzahl in dem System bleibt erhalten. Folglich wird erwartet, dass der Bedeckungsgrad aller Oberflächen im Gleichgewicht $\frac{\theta_0}{6}$ wird. Für diese Geometrie ist die Wahl eines Anfangsbedeckungsgrads größer als 6.0 ist nicht

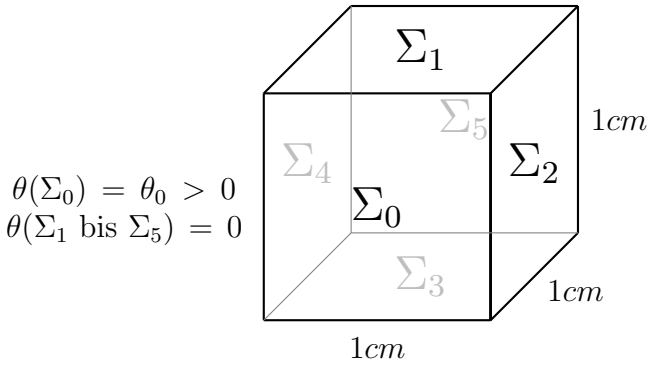


Abbildung 4.1: Die simulierte Geometrie und ihre Anfangsbedeckungsgrade

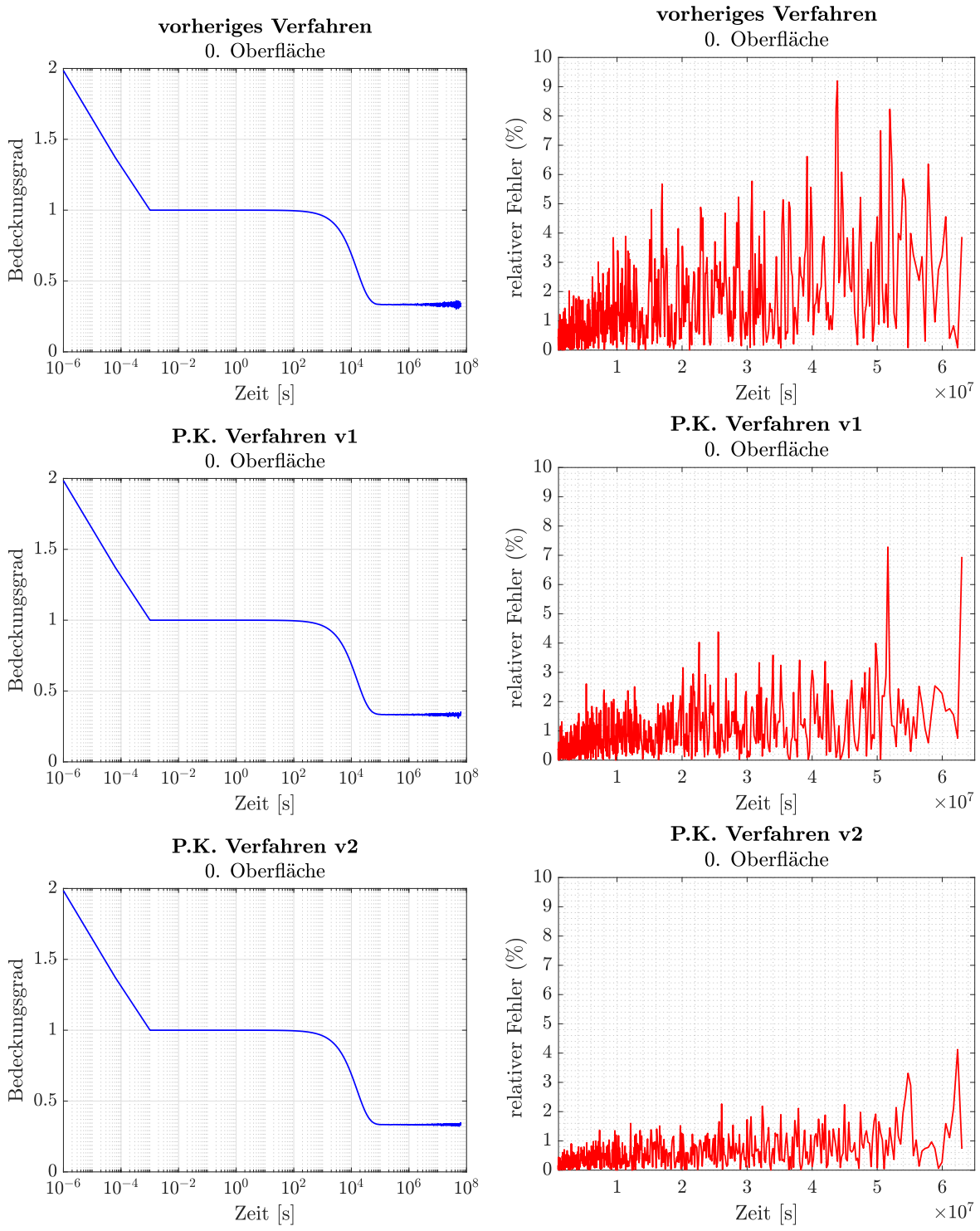
zielführend, weil in diesem Fall alle Oberflächen im Gleichgewicht einen Bedeckungsgrad größer als eins haben. Wie am Ende des Abschnitts 3.3 erwähnt, führt diese Sache gewissermaßen zum Einfrieren der Simulation. Für die Darstellung der Simulationsergebnisse werden zwei Graphen benutzt. Der erste Graph mit der blauen Kurve zeigt immer den Bedeckungsgrad der 0. Oberfläche, abhängig von der Zeit. Unterdessen ist die Zeitachse logarithmisch skaliert. Der zweite Graph mit der roten Kurve zeigt immer den relativen Fehler (die Abweichung vom Sollwert) im Prozent abhängig von der Zeit. Dabei ist die Zeitachse linear skaliert und es wird ab dem Zeitpunkt, an dem die Simulation konvergiert, betrachtet. Da in dem Graph der relative Fehler betrachtet wird, können die Simulationsergebnisse der anderen Anfangswertprobleme miteinander verglichen werden. Außerdem werden in den Graphen immer die Simulationsergebnisse der 0. Oberfläche gezeigt, denn im Gleichgewicht verhalten sich die anderen Oberflächen analog zur 0. Oberfläche und ihre Simulationsergebnisse zu zeigen wäre unnötig. Letztlich werden die wichtigsten Resultate aus den Simulationen in der Tabelle unter den Graphen dargestellt. Diese Resultate sind die Schrittweite, ab der der relative Fehler 1 % bzw. 10 % überschreitet und der maximale relative Fehler. Sie wurden natürlich aus den Simulationsergebnissen aller Oberflächen gesammelt.

4.2 Simulationsergebnisse

Die Abbildung 4.2 auf der Seite 22 illustriert die Simulationsergebnisse des vorherigen Verfahrens und der zwei verschiedenen Ansätze des Prädiktor-Korrektor-Verfahrens bei einem Anfangsbedeckungsgrad von 2.0 für die 0. Oberfläche und 0 für die anderen Oberflächen. Im linken Graph ist zu sehen, dass der Bedeckungsgrad in der kurzen Zeit auf eins fällt. Danach fällt er ziemlich langsam auf den Sollwert. Diese Tatsache ist mit den unterschiedlichen mittleren Verweilzeiten einer Wasser-Wasser-Bindung (63 µs) und einer Wasser-Edelstahl-Bindung (7 Stunden) zu erklären. In allen Simulationen ist das Rauschen ab dem Zeitpunkt von ungefähr 10^6 s erkennbar. Hierbei war die Schrittweite ca. 3 Stunden. Daneben ist in den Graphen zu merken und in der Tabelle unter den Graphen zu bestätigen, dass die beiden Ansätze des Prädiktor-Korrektor-Verfahrens das vorherige Verfahren verbessern. Jedoch ist die Verbesserung des ersten Ansatzes vernachlässigbar klein, wogegen die Verbesserung des zweiten Ansatzes bemerkbar ist. Dabei konnte die Schrittweite dreifach erhöht werden, trotzdem blieb der Fehler unter 1 %. Übrigens ist der maximale Fehler stets unter 5 %. Die Striche unter dem 10 % Fehler bedeuten, dass der relative Fehler 10 % nie überschreitet. Deshalb kann schlussgefolgert werden, dass die Ergebnisse des vorherigen Verfahrens sogar in den größeren Zeitschritten tolerierbar sind. Infolgedessen kann die Anwendung des Prädiktor-Korrektor-Verfahrens verzichtet werden, um Rechenzeit zu sparen.

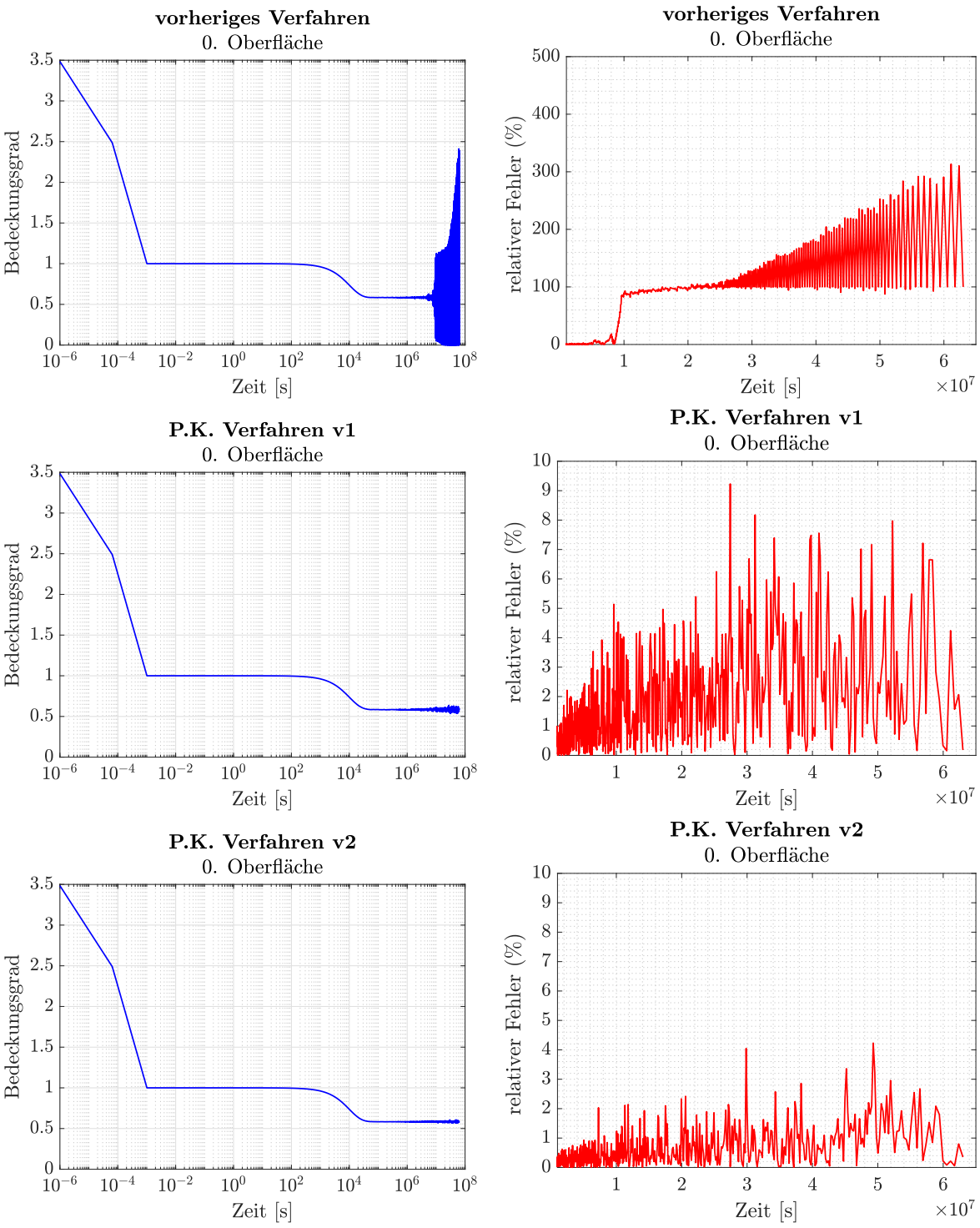
Die Abbildung 4.3 auf der Seite 23 stellt die Simulationsergebnisse bei einem Anfangsbedeckungsgrad von 3.5 für die 0. Oberfläche und 0 für die anderen Oberflächen dar. Hier ist deutliche Verbesserung des Prädiktor-Korrektor-Verfahrens gegenüber dem vorherigen Verfahren bemerkbar. Für große Schrittweiten oszillieren die Simulationsergebnisse des vorherigen Verfahrens so stark und dabei entstandener Fehler ist so groß, dass sie nicht mehr akzeptierbar sind. Demgegenüber sind die Ergebnisse der beiden Ansätzen des Prädiktor-Korrektor-Verfahrens deutlich besser, wobei der zweite Ansatz noch besser ist. Diese Feststellung kann in der Tabelle unter den Graphen bestätigt werden. Obwohl die Zunahme der Schrittweite, ab der der Fehler 1 % überschreitet, nicht groß ist, ist sie bei dem 10 % Fehler deutlich sichtbar. Bei dem zweiten Ansatz überschreitet der Fehler sogar 10 % nicht und ist stets unter 5 %. Als Schlussfolgerung kann gesagt werden, dass sich die Anwendung des Prädiktor-Korrektor-Verfahrens für dieses Anfangswertproblem lohnt.

Letztlich zeigt die Abbildung 4.4 auf der Seite 24 die Simulationsergebnisse bei einem Anfangsbedeckungsgrad von 5.0 für die 0. Oberfläche und 0 für die anderen Oberflächen. In diesem Fall hilft die Anwendung des Prädiktor-Korrektor-Verfahrens nicht viel, weil die Simulationsergebnisse der beiden Ansätze auch oszillieren, wobei sie in dem zweiten Ansatz ein bisschen besser sind, weil der Fehler meistens unter 100 % bleibt. Außerdem ist die Schrittweite des 10 % Fehlers bei dem Prädiktor-Korrektor-Verfahren ist zwar verdoppelt, aber unter Berücksichtigung der verdoppelten Simulationszeit bringt das auch nicht viel. Es kann schlussgefolgert werden, dass in bestimmten Anfangsproblemen sogar das Prädiktor-Korrektor-Verfahren die Oszillation nicht verhindern kann. Die Simulationsergebnisse mit verschiedenen Anfangsbedeckungsgraden zeigen, dass das vorherige Verfahren ab einem Anfangsbedeckungsgrad von 3.0, der erste Ansatz des Prädiktor-Korrektor-Verfahrens ab 4.0 und der zweite Ansatz ab 4.5, wegen der starken Oszillation, unvernünftige Ergebnisse geben. Außerdem wurde festgestellt, dass das Prädiktor-Korrektor-Verfahren mit mehreren Korrektor-Schritten für dieses Anfangswertproblem keine signifikante Verbesserung gegenüber dem Verfahren mit einem Korrektor-Schritt bringt.



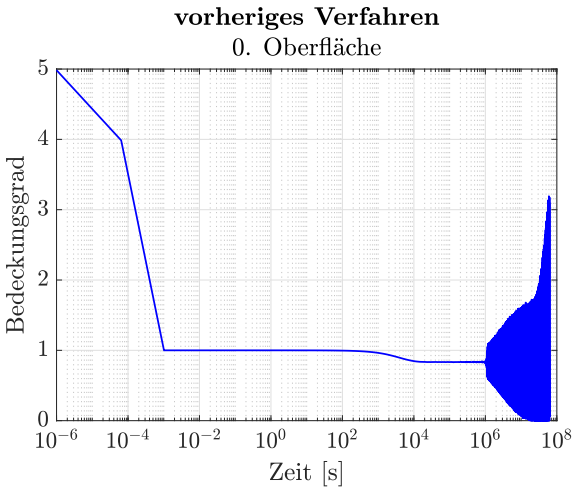
$\theta_0 = 2.0$	1 % Fehler	10 % Fehler	Max. rel. Fehler
Vorheriges Verfahren	7735 s	–	9.53 %
P.K. Verfahren v1	12780 s	–	8.24 %
P.K. Verfahren v2	23330 s	–	4.70 %

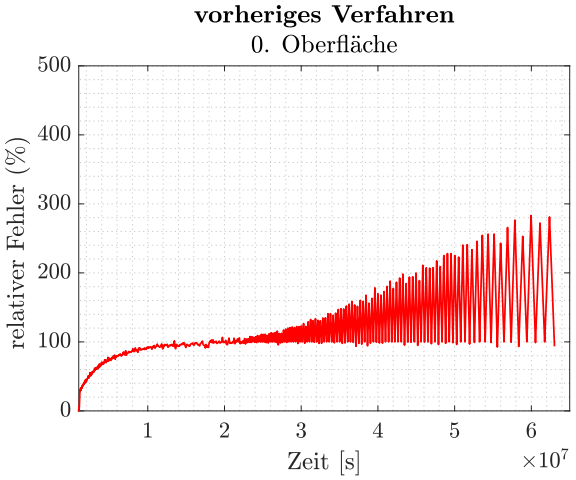
Abbildung 4.2: Simulationsergebnisse für $\theta_0 = 2.0$

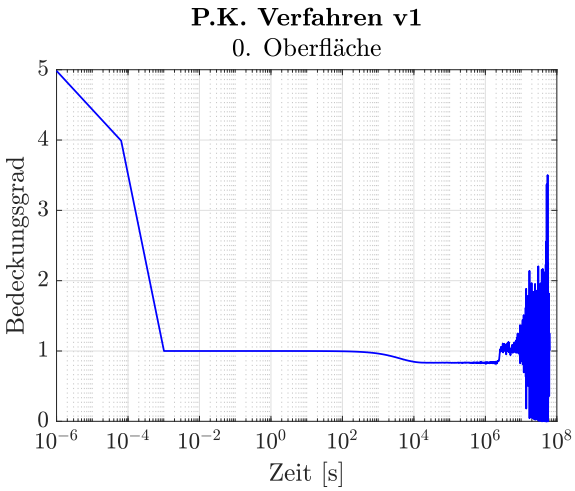


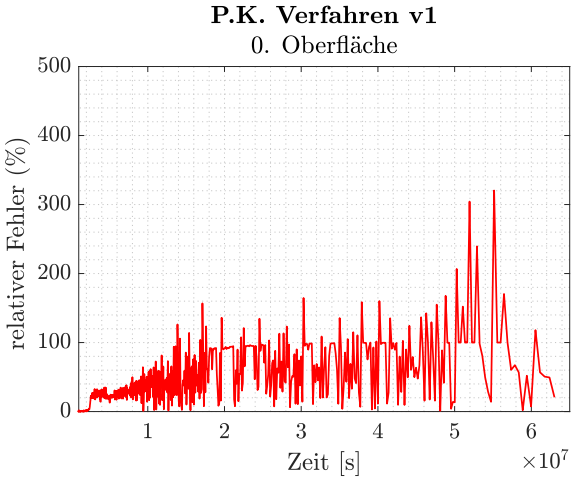
$\theta_0 = 3.5$	1 % Fehler	10 % Fehler	Max. rel. Fehler
Vorheriges Verfahren	10007 s	44870 s	316.41 %
P.K. Verfahren v1	7017 s	165900 s	11.98 %
P.K. Verfahren v2	29730 s	–	4.23 %

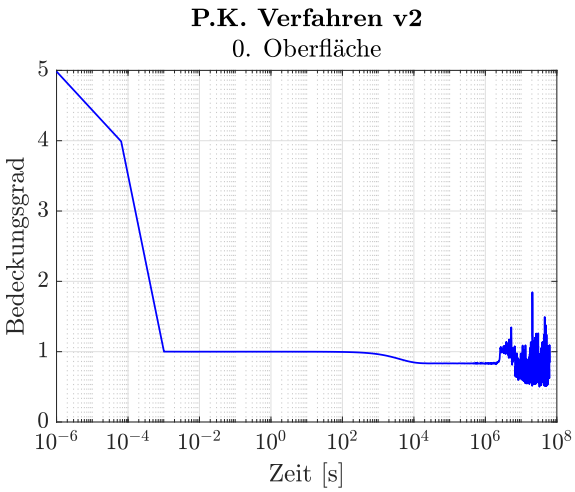
Abbildung 4.3: Simulationsergebnisse für $\theta_0 = 3.5$

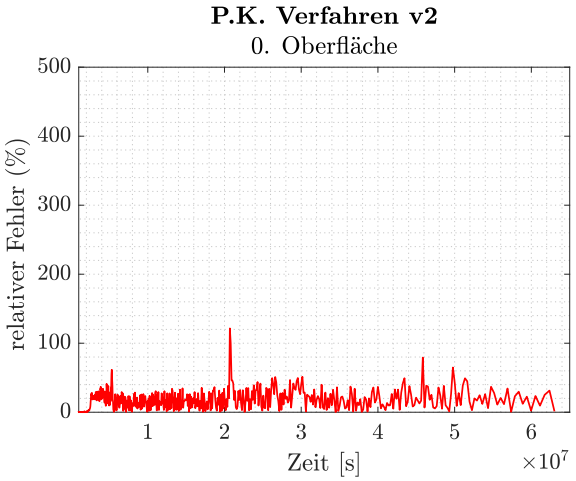












$\theta_0 = 5.0$	1 % Fehler	10 % Fehler	Max. rel. Fehler
Vorheriges Verfahren	7885 s	10821 s	291.05 %
P.K. Verfahren v1	8985 s	21220 s	403.05 %
P.K. Verfahren v2	13860 s	22560 s	174.84 %

Abbildung 4.4: Simulationsergebnisse für $\theta_0 = 5.0$

5 Abschluss und Ausblick

In dieser Ingenieurpraxis wurde die Implementierung des Prädiktor-Korrektor-Verfahrens in dem Programm „ContaminationFlow“ für die Berechnung der Bedeckung erklärt. Dazu wurde hauptsächlich auf zwei verschiedenen Implementierungsansätzen eingegangen. Die Simulationsergebnisse zeigten sich, dass das Prädiktor-Korrektor-Verfahren für manche Anfangswertprobleme eine deutliche Verbesserung im Vergleich zu dem vorherigen Verfahren bringt. Jedoch gibt es auch die Anfangswertprobleme, für die die Anwendung des neuen Verfahrens nicht viel bringt. In Kapitel 4 wurde gezeigt, dass die Anwendung des Prädiktor-Korrektor-Verfahrens in einer Würfelgeometrie mit den Anfangsbedeckungsgraden von 2.0 und 5.0 für die 0. Oberfläche und 0 für die anderen Oberflächen, nicht unbedingt nötig war, weil das vorherige Verfahren entweder tolerierbare Ergebnisse gab oder das neue Verfahren das Oszillationsproblem nicht lösen konnte. Es wurde festgestellt, dass das Prädiktor-Korrektor-Verfahren keinen Vorteil mehr bringt, wenn die Bedeckungsgrade der Oberflächen sich im Gleichgewicht auf eins annähren. Aus diesem Grund wurde die Anwendung des Prädiktor-Korrektor-Verfahrens in der Simulation optional gewählt. Die beiden Implementierungsansätze des Prädiktor-Korrektor-Verfahrens wurden gegenübergestellt. Dabei wurde es festgestellt, dass der zweite Ansatz, in dem das Prinzip des Heunverfahrens direkt in den Code eingebaut wird, tatsächlich besser als der erste Ansatz funktioniert. Nichtsdestotrotz kann es in dem zweiten Ansatz vorkommen, dass die Simulation einfriert, weil die Simulationsergebnisse so oszillieren, dass alle Oberflächen plötzlich über einen Bedeckungsgrad von mehr als eins verfügen. Das Auftreten dieses Problems ist doch sehr unwahrscheinlich und kann mit der Erhöhung der Anzahl der simulierten Teilchen gelöst werden. Als Ausblick empfehle ich, dass das Prädiktor-Korrektor-Verfahren und das vorherige Verfahren nicht in einer sehr einfachen Würfelgeometrie, sondern in einer komplexeren Geometrie verglichen werden soll. Folglich kann deduziert werden, ob das Verfahren bei einer realistischeren Simulation immer noch ein Vorteil gegenüber dem vorherigen Verfahren bringt. Da die Simulation in einer komplexeren Geometrie ausgeführt wird, wäre es nicht einfach, die Sollwerte der Bedeckungen vorweg zu wissen, bzw. analytisch zu berechnen. Demnach wäre es vernünftig, dass die gleiche Geometrie in einem realen Experiment beobachtet wird und dabei möglichst viele Messwerte gesammelt wird. Außerdem kann es sein, dass die Anwendung eines Algorithmus, die zur numerischen Lösung einer Differenzialgleichung entwickelt wurde, nicht zielführend in einer ereignisgesteuerten Simulation ist. Aus diesem Grund soll der Algorithmus eventuell für die ereignisgesteuerten Simulationen angepasst werden.

Literaturverzeichnis

[1] Arvid Hammar. *Optics For Earth Observation Instruments*. Chalmers University of Technology, 2019.

[2] Yun Zhang and Norman Kerle. Satellite remote sensing for near-real time data collection. pages 75–102, 01 2008.

[3] Zilong Jiao, Lixiang Jiang, Jipeng Sun, Jianguo Huang, and Yunfei Zhu. Outgassing environment of spacecraft: An overview. *IOP Conference Series: Materials Science and Engineering*, 611(1):012071, oct 2019. doi:10.1088/1757-899x/611/1/012071.

[4] Bernhard Schlappi, Kathrin Altwegg, Hans Balsiger, Myrtha Hassig, A. Jackel, P. Wurz, B. Fiethe, Martin Rubin, S. A. Fuselier, Jean-Jacques Berthelier, J. De Keyser, H. Reme, and U. Mall. Influence of spacecraft outgassing on the exploration of tenuous atmospheres with in situ mass spectrometry. *Journal of Geophysical Research Space Physics*, 115:A12313, 2010. URL: https://hal.archives-ouvertes.fr/hal-00511719, doi:10.1029/2010JA015734.

[5] Roberto Kersevan and J.-L Pons. Introduction to molflow+: New graphical processing unit-based monte carlo code for simulating molecular flows and for calculating angular coefficients in the compute unified device architecture environment. *Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films*, 27:1017 – 1023, 08 2009. doi:10.1116/1.3153280.

[6] Predictive simulation of contamination effects on satellites. URL: https://www.ei.tum.de/tep/forschung/forschungsprojekte/satellite-contamination/.

[7] Marton Ady. Monte Carlo simulations of ultra high vacuum and synchrotron radiation for particle accelerators, May 2016. Presented 03 May 2016. URL: https://cds.cern.ch/record/2157666.

[8] Gurkan Sin and Antonio Espuña. Editorial: Applications of monte carlo method in chemical, biochemical and environmental engineering. *Frontiers in Energy Research*, 8, 2020. URL: https://www.frontiersin.org/article/10.3389/fenrg.2020.00068, doi:10.3389/fenrg.2020.00068.

[9] Rebecca Grinham and Andrew Chew. A review of outgassing and methods for its reduction. *Applied Science and Convergence Technology*, 26:95–109, 09 2017. doi:10.5757/ASCT.2017.26.5.95.

[10] Michael Ulbrich. Numerische Mathematik EI [MA9410], May 2020. URL: https://www.moodle.tum.de/.

[11] *Numerical Differential Equation Methods*, chapter 2, pages 55–142. John Wiley & Sons, Ltd, 2016. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119121534.ch2, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119121534.ch2, doi:https://doi.org/10.1002/9781119121534.ch2.

Erklärung

Hiermit erkläre ich, Berke Karakin, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle aus fremden (einschließlich elektronischer) Quellen direkt oder indirekt übernommenen Gedanken habe ich in dieser Ausarbeitung als solche kenntlich gemacht.

München, 14. März 2022

.....
Berke Karakin