Ausbau und Modernisierung der numerischen Verfahren in den Systemcodes ATHLET, ATHLET-CD, COCOSYS und ASTEC

GRS - 469

Gesellschaft für Anlagen-
und Reaktorsicherheit
(GRS) gGmbH

Ausbau und Modernisierung der numerischen Verfahren in den Systemcodes ATHLET, ATHLET-CD, COCOSYS und ASTEC

Tim Steinhoff
Volker Jacht

Juli 2017

Anmerkung:

Das diesem Bericht zugrunde liegende FE-Vorhaben wurde mit Mitteln des Bundesministeriums für Wirtschaft und Energie (BMWi) unter dem Kennzeichen RS 1530 durchgeführt.

Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Auftragnehmer.

Der Bericht gibt die Auffassung und Meinung des Auftragnehmers wieder und muss nicht mit der Meinung des Auftraggebers übereinstimmen.

GRS - 469
ISBN 978-3-946607-52-6

Deskriptoren

ATHLET, ATHLET-CD, Differentialgleichungen, FiterRK, lineare Algebra, MPI, MUMPS, Numerical Toolkit, PETSc Numerik, NuT

Kurzfassung

Allen im Titel des Vorhabens genannten Systemcodes ist es gemein, dass die simulierten Prozesse zu nukleartechnischen Störfällen verschiedener Art und Ausdehnung auf mathematischer Ebene durch partielle Differentialgleichungen repräsentiert werden. Mittels einer räumlichen Diskretisierung transformieren sich diese zu einem System gewöhnlicher Differentialgleichungen, die dann zur Laufzeit der Codes approximativ durch Anwendung entsprechender numerischer Methoden gelöst werden. Hierbei kommen zum Teil Algorithmen- und Datenstrukturen zum Einsatz, welche vor mehreren Dekaden implementiert wurden.

Da die betrachteten Differentialgleichungen steif sind, bedarf es einer (semi-)impliziten Behandlung, was wiederum die Notwendigkeit generiert, lineare Gleichungssysteme lösen zu müssen. Diese involvieren die Jacobimatrix zur rechten Seite des Systems von Differentialgleichungen, welche mitunter groß und in diesem Kontext stets dünnbesetzt ist.

In diesem Vorhaben wurden im Rahmen des Systemcodes ATHLET numerische Modernisierungen eingebracht, welche auf effizienten Datenstrukturen fußen und aktuelle Algorithmen zur Behandlung der linearen Algebra bereitstellen. Optional kann auch verteilt gerechnet werden, so dass ebenfalls sehr hochdimensionale Probleme einer Simulation zugänglich gemacht werden konnten. Die etablierte Softwarearchitektur greift auf die externe Numerik-Bibliothek PETSc (Portable, Extensible Toolkit for Scientific computing) zu, welche neben der Bereitstellung von Datenstrukturen und Algorithmen auch den Zugriff auf den direkten Löser MUMPS (a MUltifrontal Massively Parallel sparse direct Solver) ermöglicht. Sämtliche neue Numerik ist in dem für dieses Vorhaben geschaffene *Numerical Toolkit* (`NuT`) gekapselt. Softwaretechnisch handelt es sich um ein eigenständiges Programm, welches mit ATHLET innerhalb der Architektur über das Konzept der Inter-Process-Communication zusammenarbeitet. Diese Art der Kommunikation wurde mithilfe des Message Passing Interface (MPI) realisiert und liefert ein hohes Maß an Flexibilität. Dies wird durch das Schichtmodell innerhalb des Numerical Toolkits unterstützt. Es findet eine klare Trennung zwischen Kommunikation, mathematischer Aufgabe und implementierendem Code, der auf die externen Bibliotheken zugreift, statt. Somit wird auch der Anforderung der leichten Wartbar- und Verständlichkeit durch den `NuT`-Code entsprochen. Die ATHLET-Seite der Kommunikation wird mittels eines Plug-Ins zu ATHLET gekapselt. Dieses Konzept ermöglicht ein minimal-invasives Vorgehen hinsichtlich des bestehenden Systemcodes, so dass andere Entwicklungen möglichst wenig beeinträchtigt werden.

Aufgrund dessen, dass durch die generelle Code-Entwicklung zum ATHLET während der Laufzeit des Vorhabens auch ATHLET-CD über das Plug-In-Konzept gekoppelt wurde, ist die `NuT`-Architektur ebenso in diesem Kontext direkt nutzbar. Den Entwicklungsarbeiten zu den anderen Codes stehen die Ergebnisse dieses Vorhabens zur Einsicht und Orientierung zur Verfügung.

Es wurden Testläufe zu verschiedenen Modellen ausgeführt. Um einen grundlegenden Eindruck zur Funktionstüchtigkeit der Neuerungen zu haben, wurde das ATHLET-Jenkins-System zur kontinuierlichen Integration genutzt. Hierbei ergab sich ein leicht verbessertes Abschneiden im Vergleich zum Standard-ATHLET. Zusätzliche Messungen zur Performance haben gezeigt, dass für jedes der dort betrachteten Probleme eine Verbesserung in der Laufzeit erzielt wurde, welche gerade für große Probleme von signifikantem Ausmaß war, da hier die genutzten externen Bibliotheken ihr volles Potential ausspielen konnten.

Der Differentialgleichungslöser in ATHLET ist durch eine Extrapolation basierend auf dem linear-impliziten Euler gegeben. Diese Methode wurde analysiert, und es ergab sich Verbesserungspotential, welches zu der Entwicklung einer Theorie zu s. g. Finite iteration Runge-Kutta-Methoden ($FiterRK$) führte. Diese nutzend wurden verschiedenen Methoden konstruiert, welche vorteilhafte Eigenschaften wie A-Stabilität, ein Einpunktspektrum der Koeffizientenmatrix sowie eine hohe Stufenordnung aufweisen. Im Rahmen der Lösung allgemeiner steifer Probleme aus der Literatur zeigte sich eine deutliche Überlegenheit der neuen Konzepte gegenüber der Extrapolation.

Das Numerical Toolkit wurde erweitert, um für eine gegebene Schrittweite einen Schritt einer Methode vom $FiterRK$-Typ zum approximativen Lösen der Differentialgleichungen in ATHLET auszuführen. Dies schließt den Extrapolationsansatz mit ein. Zusätzlich wurde bereits eine Teilmenge der Kontrollstrukturen zur Zeitintegration im ATHLET-Kontext berücksichtigt. Die DGL-Erweiterungen des Toolkits fanden keine Anwendung für die Simulationsläufe zur Validierung und Performancemessung. Jedoch liefern die erarbeiteten theoretischen Ergebnisse sowie die bis dato getätigten Implementierungsarbeiten eine fundierte Basis für etwaige weiterführende Arbeiten, welche darauf abzielen, sämtliche Kontrollstrukturen einzubinden und somit einen stabilen Integrationsablauf im ATHLET-Kontext zu gewährleisten.

Abstract

All the codes that are stated in the title of this project have in common that the intrinsic dynamics of the simulated accidents of different type and scale are represented by a set of partial differential equations. By means of some spartial discretization approach these yield a system of ordinary differential equations (ODE), which are solved numerically by appropriate time integration methods. The code for these methods is based on an algorithmic framework and on data structures, that were established several decades ago.

The ODE-system is considered to be stiff which implies a treatment via (semi-)implicit methods. This, in turn, leads to the demand to solve linear systems of equations. The Jacobian of the right hand side of the ODE-system is involved in this linear algebra task. Due to the underlying network structure of the spartial discretization the Jacobian is sparse and for problems of large dimensions of considerable size.

For this project new numerical facilities were developed in the framework of the ATHLET code. These are based on efficient data structures and employ modern algorithms to solve the aforementioned linear systems. Optionally, it is possible to make use of distributed computing to improve the performance or to make some problems of high dimensions available for simulation. The established software architecture uses the numerical library PETSc (Portable, Extensible Toolkit for Scientific computing) which supplies data strucures and numerical algorithms as well as an interface to the direct solver MUMPS (a MUltifrontal Massively Parallel sparse direct Solver). The architecture is labeled by the term *Numerical Toolkit* (`NuT`). On the level of programming the Numerical Toolkit is a self-contained executable, which cooperates with ATHLET via the concept of an inter process communication (IPC) based on the Message Passing Interface (MPI). This leads to a high degree of flexibility, which is further supported by the `NuT`-internal layer model for processing tasks. On both logical and programming level there is a strict distinction between communication, mathematical function, and mapping to external functions given by the external libraries. This approach greatly improves maintenance and readability of the new code. Regarding the ATHLET-part of the overall architecture the IPC is encapsualted in a plug-in that is loaded into the memory during the runtime of ATHLET. Exploiting such a plug-in concept gives rise to a minimally invasive influence on the already existing code and therefore on other concurrent development processes. Due to the general code development during the reporting period of this project the coupling between ATHLET and ATHLET-CD was also improved by applying the plug-in concept on ATHLET-CD. Hence, the Numerical Toolkit is directly available in this context

too. Considering the other codes, the results of this project may be shared with the correspondiong developer teams as needed.

Some test problems were executed for the sake of validation and performance measurement. Regarding the validation part the ATHLET Jenkins system for continous integration was used. A slight improvement compared to standard ATHLET was achieved. For all problems that were taken into consideration for the perfomance tests there was also an improvement in runtime. Especially for problems of higher dimensions the enhancements were significant.

The standard ODE solver in ATHLET is given by an extrapolation ansatz based on the linearly implicit Euler method. An analysis of this method revealed that there is room for improvement on a theoretical level. This led to the development of a theory for so-called Finite iteration Runge-Kutta methods ($FiterRK$). By means of the theory several methods were constructed, that show some amiable properties like A-stability, a single point spectrum of the coefficient matrix, and high stage order. Applying these methods to well-known stiff problems from the literature showed some clear supremacy of the new concepts in terms of performance and in comparison to the extrapolation ansatz.

The Numerical Toolkit was extended to provide options for executing one step of a $FiterRK$ method w.r.t. the ATHLET ODE system and for a given step size. The extrapolation ansatz is also included into this framework. Additionally, a subset of control mechanism for the time integration within the ATHLET context was taken into account. Validation and performance tests do not make use of these ODE-related extensions. On the other hand, the developed theoretical background and the written code so far provide a good starting point for further development that aims to include all necessary control mechanisms to ensure a stable execution of the integration process.

Inhaltsverzeichnis

1 Einleitung

Die GRS entwickelt und validiert das Programmsystem AC^2 bestehend aus den Komponenten ATHLET (Analyse der Thermo-Hydraulik von Lecks und Transienten), ATHLET-CD (Core Degradation) und COCOSYS (Containment-Codesystem). Hierbei dient ATHLET zur Simulation des Kreislaufverhaltens von Transienten und Störfällen. Mit der Erweiterung ATHLET-CD können Unfälle mit Kernzerstörung simuliert werden, und mittels COCOSYS wird die Berücksichtigung von Stör- und Unfallabläufen in Containments ermöglicht. Teile des Programmes werden als Modul auch in dem gemeinsam mit dem IRSN entwickelten Integralcode ASTEC (Accident Source Term Evaluation Code) eingesetzt.

Allen genannten Programmen ist es gemein, dass die physikalischen Prozesse über ein System von partiellen Differentialgleichungen abgebildet werden, welche nach einer adäquaten räumlichen Diskretisierung zu einem System gewöhnlicher Differentialgleichungen führen. Dieses kann anschließend als Anfangswertproblem gelöst werden. Zum Lösen dieses Problems kam bisher proprietäre Numerik zum Einsatz, welche zum Teil vor 20-30 Jahren etabliert wurde.

Für diese Arbeit dient der Code ATHLET als Entwicklungsbett. Die Motivation für dieses Projekt lag darin, strukturiert aktuelle Numerik zur Berechnung der Systemgrößen während eines Simulationslaufes zum Einsatz zu bringen, um die Laufzeit, die Verständlichkeit und Flexibilität des Codes zu verbessern. Hierfür wurde auf verschiedene freie Numerik-Bibliotheken zurückgegriffen, s. Kapitel 2.

Das Projekt umfasst drei Hauptarbeitspunkte, wobei sich die Ziele zum dritten Arbeitspunkt im Laufe des Projektes gewandelt haben, um den aktuellen Bedürfnissen und Gelegenheiten Rechnung zu tragen. Entsprechend sind die Inhalte auf vier Kapitel aufgeteilt:

- Kapitel 2 – AP1: *Erstellen standardisierter Schnittstellen und Implementierung von Verfahren aus offenen mathematischen Bibliotheken zur Lösung der involvierten linearen Gleichungssysteme*
- Kapitel 3 – AP2: *Analyse der Numerik zur Integration der DGL-Systeme und Test von neuen innovativen Lösungsverfahren*
- Kapitel 4 – AP3: *Kopplung mit anderen Codes – Erstellung einer stabilisierten Schnittstelle, Test iterativer Solver für Kopplungen*
- Kapitel 5 – AP3b: *Berücksichtigung paralleler Datenstrukturen und Operationen sowie Anbindung der numerischen Strukturen mittels MPI*

In diesen Kapiteln werden jeweils zunächst die Anforderungen genannt und die Ergebnisse hierzu kompakt dargelegt. Anschließend werden die fachlichen Resultate und Ausarbeitungen ausführlich in einer nach dem entsprechenden Kontext strukturierten Weise präsentiert. In Kapitel 5 wird zusätzlich auf die Hintergründe zum Wechsel des Themenkomplexes eingegangen. Als weiterer wesentlicher Abschnitt dieser Arbeit ergibt sich Kapitel 6 *Validierung der Modifikationen*, in dem verschiedene Simulationsläufe zum Austesten der Neuerungen diskutiert werden.

Als primäres Ergebnis des Projektes kann die Etablierung des `Numerical Toolkits` (kurz: `NuT`) gesehen werden, welches numerische Funktionalitäten kapselt und aggregiert. Die Anbindung an bestehenden Code sowie interner Aufbau werden eingehend durch die Inhalte in Kapitel 2 und Kapitel 5 erläutert. An dieser Stelle sei aber bereits auf das Flexibilitätspotential der Architektur und die Unabhängigkeit des Numerical Toolkits hingewiesen: De facto lässt sich das Toolkit als weitere Komponente im Gefüge der GRS-Codes interpretieren, vgl. Abb. 1.1, welche mit ATHLET verknüpft wurde.

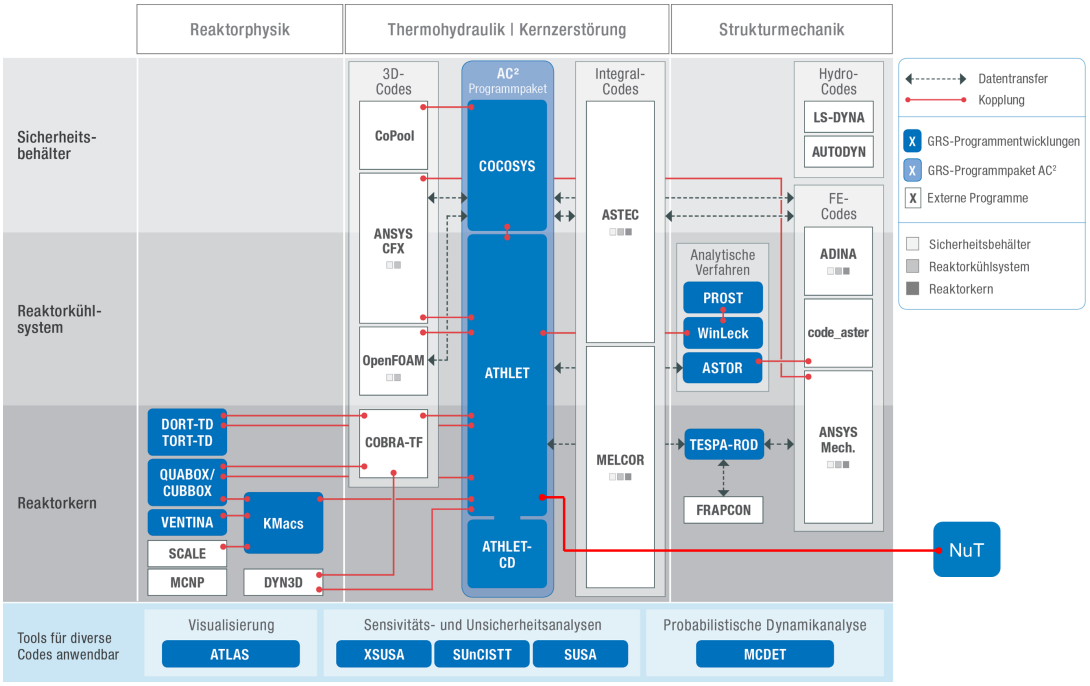


Abb. 1.1 Das Numerical Toolkit (NuT) in der GRS-Codeübersicht

1.1 Fachlicher Hintergrund und Bemerkungen

Im ATHLET-Kontext wird ausgehend von den Erhaltungsgleichungen für Masse, Energie und Impuls die räumliche Diskretisierung mittels spezieller finiter Volumen durchgeführt

und ein nicht-autonomes Anfangswertproblem

$$y'(t) = f(t, y), \quad y(t_0) = y_0, \tag{1.1}$$

in den Volumengrößen Temperatur, Druck und relativer Masseanteil sowie Massesträmen zwischen den Volumina formuliert, vgl. Abbildung 1.2. Die Komplexität der Modellierung

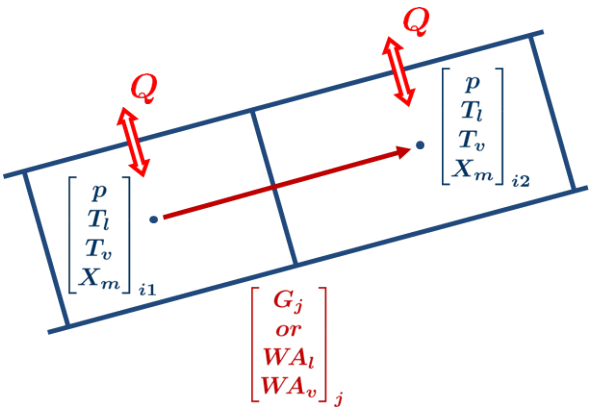


Abb. 1.2 Versetztes Gitter aus Kontrollvolumina und Junction

richtet sich nach der Feinmaschigkeit des Netzes, deren Verzweigungsgrad sowie der evtl. Aufgliederung der Massenströme nach Gas und Füssigkeit. Das System (1.1) ist im Sinne einer numerischen Lösungsstrategie als steif zu betrachten, s. /§ 6.1 AUS 16a/. Somit muss auf (semi-)implizite Verfahren zur schrittweisen Zeitintegration von (1.1) zurückgegriffen werden, welche sich intern eines Newton-Prozesses bedienen. Das Problem (1.1) ist aus algorithmischer und numerischer Sicht sehr vielschichtig. Diesem Aspekt wird in dieser Arbeit Rechnung getragen. Es sei erwähnt, dass manche stehenden Begriffe innerhalb der Numerik zu gewöhnlichen Differentialgleichungen, linearen Gleichungssystemen oder zur Eigenwerttheorie nicht näher erläutert werden. Zur Klärung dieser Begriffe sei für die einzelnen Themenfelder auf folgende Literatur verwiesen:

- Numerik gewöhnlicher Differentialgleichungen

  /HAI 93/,/HAI 96/,/BUT 16/,/VOSS 07b/,/DEU 13/,/SHA 94/

- Numerik linearer Gleichungssysteme
  /MEI 11/,/ROO 99/,/FRE 07b/,/SAA 03/

- numerische Eigenwertberechnung
  /VOSS 07a/,/SAA 11/

- Grundlagen der Numerik
  /VOSS 07a/,/FRE 07b/,/ROO 99/,/DEU 08/

Des Weiteren seien einige allgemeine Anmerkungen gemacht:

- Aufgrund von Code-Entwicklungen, die an anderer Stelle aber parallel zu diesem Projekt stattgefunden haben, ist trotz des Fokus auf ATHLET auch bereits ATHLET-CD durch die Neuerungen abgedeckt. Näheres hierzu findet sich in Bemerkung 2.1.

- Für dieses Projekt entstanden ca. 6300 Zeilen Code. Diese sind nicht dem Bericht beigefügt. Lediglich in Anhang A.1 sind die Signaturen der durch das Plug-In an ATHLET zur Verfügung gestellten Methoden aufgelistet. Zusätzlich werden in Unterabschnitt 2.3.1.3 einige Codeausschnitte dargelegt, um ein exemplarisches Durchlaufen der aufgebauten Architektur nachvollziehen zu können.

- Sämtliche Modifikationen und Erweiterungen kommen indirekt allen hierauf basierenden Codes zugute. Bei Bedarf steht der für diese Arbeit erstellte Code GRS-intern zur Einsicht bereit oder kann als Basis für weitere Vorhaben in diese Richtung genutzt werden.

- Ein Austesten automatisierter Dokumentationstools zu Codeprojekten ist keinem speziellen Arbeitspunkt zugeordnet, war aber Teil des Arbeitsplanes. Der Status quo zu Beginn des Projektes zeigte sich derart, dass auf eine auf FORTRAN basierende Eigenentwicklung zurückgegriffen wurde, die auf das kommerzielle Tool *FORCHECK*, /FOR 17/, angewiesen war. Parallel zu den Arbeiten in diesem Projekt wurde eine Python-basierte Modifikation hiervon entwickelt, die eine solche Abhängigkeit nicht mehr aufweist und dennoch den Anforderungen des speziellen Code-Desings im ATHLET genügen kann. Hierzu zählt das Auslesen und Verarbeiten besonderer Header-Kommentare sowie die Handhabung einer Vielzahl von Funktionen, die nicht in Modulen gekapselt sind. In einem Evaluierungslauf zum Dokumentationstool *Doxygen*, /HEE 17/, zeigte sich, dass dieses besagte Besonderheiten nicht befriedigend berücksichtigen kann. Auch solche Konzepte wie generische Interfaces, /MET 11, §5.18/, machten Probleme und führten zu einer aufgeblähten Ausgabe. Somit zeigt sich die Python-basierte, proprietäre Lösung eindeutig im Vorteil, was zum Ergebnis hat, dieses auch zukïnftig zum Einsatz zu bringen.

- Die aufgebaute `NuT`-Architektur bietet vielerlei Lösereinstellungen. Auch wenn nicht alle zur Verfügung stehenden Optionen in allen Kombinationen getestet wurden, bietet die Architektur dem Nutzer die Möglichkeit, auf Kommandozeilenebene zu selektieren und somit ein gegebenes Problem mit verschiedenen Löserkonfigurationen anzugehen, s. auch Unterabschnitt 2.3.1.2. Keine Codemodifikationen und kein erneuter Compilierungsprozess sind vonnöten.

- Das Numerische Toolkit und die umgebende Architektur wurden zur Projektzeit in einem Entwicklungszweig etabliert, der regelmäßig mit anderen Änderungen am Code abgeglichen wurde. Die Neuerungen werden vermutlich in der nächsten Nebenversion (minor release) 3.2 Berücksichtigung finden und somit der allgemeinen Nutzerschaft zur Verfügung stehen.

2 Erstellen standardisierter Schnittstellen und Implementierung von Verfahren aus offenen mathematischen Bibliotheken zur Lösung der involvierten linearen Gleichungssysteme

2.1 Anforderungen und Ergebnisse

Ziel dieses Arbeitspaketes ist es, zum GRS-Systemcode ATHLET eine Schnittstellenarchitektur zu etablieren, die den Einsatz externer numerischer Bibliotheken zum Lösen der im ATHLET-Kontext auftretenden linearen Gleichungssysteme ermöglicht. Ein wesentliches Augenmerk liegt hierbei auf der effizienten Handhabung der in den besagten Gleichungssystemen involvierten und dünnbesetzten Jacobimatrix zur rechten Seite in (1.1).

Aufgrund der gewählten Architektur, s. Kapitel 1 sowie Abschnitt 2.3, besteht eine enge Verbindung zum Arbeitspunkt 3b. Die dort durchgeführten Arbeiten hinsichtlich einer MPI-basierten Kommunikation zur Anbindung und effizienten Nutzung externer numerischer Bibliotheken können jedoch wohl getrennt von den Aufgabenstellungen dieses Arbeitspunktes gesehen werden, da auf unterschiedlichen Abstraktionsebenen gearbeitet wird. Der für das Projekt entwickelte Code ist derart aufgebaut, dass der MPI-Anteil transparent gehalten wird. So kann sich bei der Implementierung der hier geforderten Funktionalitäten auf die tatsächliche algorithmische Anforderung konzentriert werden.

Als zusätzliches Ziel dieses Arbeitspaktes ergibt sich eine Erweiterung der Architektur dahingehend, dass ein schrittweises Ausführen verschiedener Methoden zum approximativen Lösen der im ATHLET-Kontext genutzten gewöhnlichen Differentialgleichungen möglich sei. An dieser Stelle ist ausschließlich die softwaretechnische Ausprägung von Relevanz. Details zur Differentialgleichungsnumerik im ATHLET-Kontext finden sich in Kapitel 3.

Geleistetes

Die durchgeführten Arbeiten etablieren ein vollständiges Abhandeln der numerischen linearen Algebra mit Hilfe der externen Bibliotheken *PETSc*, /BAL 16/, *MUMPS*, /AME 16/, und *METIS*, /KAR 16/, sowohl unter Linux als auch Windows im Rahmen der `NuT`-Architektur. Hierbei ist ebenfalls das effiziente Erstellen und Speichern der Jacobimatrizen zu (1.1) inklusive partieller Updates berücksichtigt. Die Arbeiten zur (Teil-)Generierung der Jacobimatrix geschehen im Zusammenspiel mit existierenden ATHLET-Routinen, welche entsprechend angepasst wurden, aber nach wie vor den Vorteil bieten, modellspezifische

Aspekte einfließen zu lassen. Im Tandem mit dem ATHLET-eigenen Differentialgleichungslöser FEBE ist eine Architektur geschaffen worden, welche abhängig vom betrachteten Problem zum Teil signifikante Verbesserungen bezüglich der benötigten Rechenzeit von Simulationsläufen liefert, s. hierzu Kapitel 6. Einher geht hiermit die geforderte Flexibilität in der Wahl der Lösungsverfahren zu den linearen Gleichungssystemen. Während PETSc in der Sparte der iterativen Löser viele der üblichen Krylov-Unterraum-Methoden, s. /BAL 16/ sowie z. B. /MEI 11/, direkt anbietet, steht mit der Kombination aus MUMPS und METIS ein leistungsstarker direkter Löser zur Verfügung. Eine Kombination beider Ansätze ist über das Präkonditionierungskonzept zu Krylov-Unterraum-Methoden ebenso möglich. Matrixfreie Versionen iterativer Verfahren wurden nicht berücksichtigt, was hauptsächlich auf die sehr spezielle Weise, wie mit f aus (1.1) umzugehen sei, zurückzuführen ist, s. hierzu auch Bemerkung 3.70 in Abschnitt 3.6.

Ein großes Plus der Architektur ist die flexible Wahl der Lösungsverfahren, welche über Befehlszeilenkommandos selektiert werden können, s. Unterabschnitt 2.3.1.2.

Ausführlichere Informationen zu den gewählten Bibliotheken, deren Einsatz im Projekt sowie zur Motivation, diese einzubinden, finden sich in Abschnitt 2.2. Für den Auswahlprozess war nicht nur die Funktionsmächtigkeit, sondern auch das Potential eines vereinheitlichten Zuganges sowie ein GRS-kompatibles Lizenzmodel von entscheidender Bedeutung. Die Summe der Anforderungen macht bereits klar, dass Forschungscodes und experimentelle Algorithmen nicht in Frage kommen. Insofern wird z. B. das IDR(s)-Verfahren, /SON 08/, zur Lösung der linearen Systeme nicht betrachtet. Ebenso ist es nicht sinnig, ein buntes Potpourri aus vielen kleinen Einzelprogrammen zu ertüchtigen, wenn die Datenstrukturen heterogen und nicht bewusst auf Effizienz ausgelegt sind.

Es erwies sich als sinnvoll, die Bestimmung der zu nutzenden Bibliotheken *vor* einer Konkretisierung des Schnittstellen-Codes zu treffen. Dementsprechend ist die Diskussion hierzu den Aussagen zur Schnittstellenarchitektur vorgezogen. Das Architekturmodell wird danach in Abschnitt 2.3 erörtert.

Der Ausbau der Architektur zur Berücksichtigung weiterer Methoden, um (1.1) schrittweise numerisch zu lösen, ist grundsätzlich funktionstüchtig. Das Entitätenmodell der Architektur macht es leicht, eine Verbindung zwischen linearer Algebra und Differentialgleichungsnumerik herzustellen, s. Unterabschnitt 2.3.1.1. Der jetzige Entwicklungsstand bietet aber noch nicht hinreichende Stabilität, um den Anforderungen eines üblichen ATHLET-Simulationslaufes zu genügen. Dies ist den Besonderheiten geschuldet, welche die teils sehr enge Verzahnung des in FEBE genutzten Extrapolationsansatzes mit dem umgebenden ATHLET-Code bewirkt, s. auch Abschnitt 3.6. Die Komplexität der Verzah-

nung und die Folgen hieraus waren im Vorfeld nicht zur Gänze über die begleitende Dokumentation zu ATHLET absehbar, da durch eine Vielzahl von Detailanpassungen FEBE im aktuellen Stand nicht mehr dem Charakter eines *general purpose ODE-solvers* entspricht, wie es in /AUS 16a, §6.1/ proklamiert wird. Man beachte, dass die Methode der Extrapolation basierend auf dem linear-impliziten Euler sehr wohl generisch ist, jedoch ist dies nicht der Fall für die Implementierung in ATHLET. Gleiches gilt für die Steuerung zur Schrittweitenwahl und die Aufdatierung der Jacobimatrix. Trotz erheblichen Mehraufwandes, auch auf Seiten der Theorie, s. Abschnitt 3.6, wurde für die Architektur das Ziel erreicht, dass sich weiterführende Arbeiten zu diesem Thema hauptsächlich auf Modifikationen im ATHLET-Code fokusieren können. Die umgebende Architektur ist de facto fertiggestellt.

Der gesamte für dieses Projekt entwickelte Code zur konstruierten Softwarearchitektur ist in FORTRAN geschrieben. Hierbei wurde die *free source form* genutzt. Es treten keine implizit definierten Variablen auf und generell wird nicht über globale Variablen kommuniziert. Lediglich im neu erstellten Modul `cnut` werden einige wenige globale Variablen gekapselt, um sich ATHLET-spezifische Strukturinformation zur Jacobimatrix über evtl. mehrere Integrationsschritte hinweg zu merken. Der Code bedient sich Direktiven des Standards *Fortran 2003* – Stichwort: *procedure pointers*, s. auch Code 2.2 oder Code 2.3. In der Linux-Version wird aufgrund des OpenMPI-Moduls `mpi_f08` auch indirekt auf Konstrukte des Standards *Fortran 2008* zugegriffen.

Bemerkung 2.1. Parallel zu den Arbeiten in diesem Projekt wurde der GRS-Code ATHLET-CD über ein Plug-In-Konzept mit ATHLET verknüpft. Wie im Stand-Alone-Modus wird zur Ausführung dieses Code-Tandems ATHLET aufgerufen. Lediglich ein angepasster Datensatz muss vorliegen sowie ATHLET-CD als shared library. Da der für dieses Projekt relevante Numerik-Code eine etwaige Abhängigkeit von ATHLET-CD nur über f aus (1.1) erfährt, ist eine Anwendung der hier vorgestellten Neuerungen direkt auch im Kontext ATHLET/CD möglich. Im Folgenden wird lediglich der Bezeichner *ATHLET* verwandt. Dieser könnte aber auch durch die Kombination *ATHLET/CD* ersetzt werden. Ist eine gesonderte Betrachtung für ATHLET-CD angebracht, so wird explizit darauf hingewiesen, s. Abschnitt 2.3 und auch Abschnitt 6.1.

2.2 Externe Bibliotheken

Das Auslagern von bisher proprietär gehaltenen numerischen Funktionalitäten geht mit einem zweiteiligen Paradigmenwechsel einher. Zum einen verschieben sich Verantwort-

lichkeiten hin zu externem Programmcode. Zum anderen tritt GRS-spezifischer Code als Nutzer von Bibliotheksfunktionen auf, die über fest definierte Signaturen zur Verfügung stehen. Individualisierungen, wie sie in einem proprietären Umfeld stattfinden können, sind nicht in beliebigem Umfang möglich. Zwar stehen für diese Arbeit (kosten-)freie Bibliotheken im Fokus, was grundsätzlich Codeeinblicke und -modifikationen zulässt, es ist jedoch ratsam, bei Bedarf das jeweils dedizierte Entwicklerteam zu kontaktieren. Denn bei numerisch orientierten Bibliotheken handelt es sich um komplexe Softwareprodukte, deren Codeaufbau und interne Abhängigkeiten nicht ad hoc ersichtlich sein müssen. Bugfixing und Einarbeitungen von Änderungen können sich dann als sehr mühselig oder praktisch nicht durchführbar erweisen. Hinzu kommt, dass persönliche Modifikationen nicht durch offizielle zukünftige Versionen einer Bibliothek Berücksichtigung finden müssen. Solche Änderungen wären also immer wieder einzupflegen. Auf der anderen Seite ist es empfehlenswert, stets aktuelle (stabile) Versionen der externen Softwareprodukte einzubinden, um auf dem neuesten Stand von Performanz, Funktionsmächtigkeit und Bugfixing zu bleiben. Die Wartung und Pflege von GRS-Software erfährt somit eine Dimensionserweiterung, welche mit Mehrarbeit einhergeht.

Der Wunsch nach Einbindung (kosten-)freier Bibliotheken impliziert offensichtlich eine nichttriviale Abhängigkeit von externen Quellen, deren Verantwortliche in keiner Erfüllungspflicht gegenüber der GRS stehen. Hinzukommt, dass der Pflegeaufwand für GRS-Code komplexer wird. Dies sollte stets bewusst sein.

2.2.1 Anforderungen an Bibliotheken

Die Anforderungen an die externen Bibliotheken lassen sich in drei Kategorien einteilen:

- Funktionsmächtigkeit,
- Bedienbarkeit,
- Verfügbarkeit (inkl. Lizenzaspekten).

Funktionsmächtigkeit

Diese Anforderung bezieht sich auf den numerischen Nutzen, der aus der Verwendung der Bibliotheken entstehen soll. Grundlegend besteht nach Kapitel 3 Bedarf an

- speicheroptimierten Datenstrukturen zu Vektor- und Matrix-Entitäten,
- Methoden zur effizienten Bestimmung der Jacobimatrix über finite Differenzen,
- leistungsstarken Algorithmen zur Lösung von großen linearen Gleichungssystemen, wobei die involvierte Matrix dünnbesetzt ist.

Obige Punkte wirken sich extrem auf die Performanz des Gesamtalgorithmus ATHLET aus und sind damit von oberster Priorität.

Bemerkung 2.2. Anforderungen an die Funktionsmächtigkeit von externen Bibliotheken beinhalten implizit die Erwartung, dass verlässliche Ergebnisse produziert werden. Um sich hiervon zu überzeugen, wurden verschiedene eigene Tests zur Überprüfung angesetzt, s. Unterabschnitt 2.3.1.6 sowie Kapitel 6. Des Weiteren lohnt es sich, in Erfahrung zu bringen, inwiefern potentielle Kandidaten bereits an anderer Stelle erfolgreich eingesetzt werden. Tauchen hierbei große und komplexe sowie ergebnissensitive Anwendungen auf, kann davon ausgegangen werden, dass grundsätzlich Verlässlichkeit sichergestellt ist.

Bedienbarkeit

Der GRS-eigene Code tritt als Nutzer der externen Funktionalitäten auf. Diese müssen hinreichend dokumentiert und verhältnismäßig einfach einzubinden sein. Zur besseren Übersicht, Wart- und Austauschbarkeit ist im Zusammenspiel mit dem in Abschnitt 2.3 erläuterten Abstraktionsmodell ein einheitliches Zugriffskonzept zu bieten. Ebenso ist es wünschenswert, dass weitestmöglich transparente Interoperabilität der externen Bibliotheken untereinander besteht.

Verfügbarkeit

Die Anforderung *Verfügbarkeit* umfasst verschiedenen Ausprägungen des Begriffs.

- Obwohl freie Software per se für jeden verfügbar ist, muss das jeweilige Lizenzmodell zu den GRS-spezifischen Anforderungen passen. Details hierzu finden sich in Abschnitt 2.2.4.
- Freie wissenschaftliche Software ist üblicherweise linuxbasiert. Für diese Arbeit ist jedoch auch eine Windows-Implementation zu berücksichtigen. Folglich müssen Mittel und Wege zur Verfügung stehen, Bibliotheksfunktionalitäten auch auf dem Windows-Betriebssystem nutzen zu können.
- Der ATHLET-Code ist in FORTRAN geschrieben, was jedoch nicht immer der Fall für numerische Software ist. Hier findet häufig C oder C++ Anwendung. Es ist somit wünschenswert, dass solche Bibliotheken eine stabile Schnittstelle zu FORTRAN aufweisen.
- Nicht selten ist wissenschaftliche Software aus dem akademischen Umfeld in seiner Finanzierung projektabhängig. Läuft das Projekt aus, ist es schwierig, Weiterentwicklung und Pflege aufrechtzuerhalten. Es ist nicht sinnvoll, sich von solchen Umständen abhängig zu machen. Somit ist auf ein verlässliches *Funding* Wert zu legen, auf aktive Entwicklung und Pflege sowie eine reiche Community.

2.2.2 Auswahl von Bibliotheken

Die Anforderungen aus dem vorherigen Abschnitt berücksichtigend wurden entsprechende Recherchen unternommen, wobei sich abschließend für folgende Kombination entschieden wurde:

- PETSc – Portable, Extensible Toolkit for Scientific Computation, /BAL 16/, Version 3.7.6
- MUMPS – a MUltifrontal Massively Parallel sparse direct Solver, /AME 16/, Version 5.1.1-p3
- (Par)METIS, Graphalgorithmen zur Fill-In-Reduktion[1], /KAR 16/, Version 5.1.0-p3 (METIS), 4.0.3 (ParMETIS)

Während der Projektlaufzeit wurde stets auf Neuerungen in den besagten Bibliotheken reagiert. Die obigen Versionsnummern spiegeln den aktuellen Stand der Softwareprodukte zum Zeitpunkt des Abschlusses des Projektes wider.

PETSc

PETSc wird am Argonne National Laboratory entwickelt und besteht aus einer Sammlung von Datenstrukturen und Prozeduren, die auf unterschiedlichen Abstraktionsebenen numerische Algorithmen realisieren. Diese erstrecken sich von Algorithmen zur Lösung linearer Gleichungssysteme bis hin zur numerischen Zeitintegration nichtlinearer, semidiskretisierter partieller Differentialgleichung. Der Nutzer hat die Möglichkeit, jedes Abstraktionsniveau direkt anzusprechen. Im Vordergrund steht die mathematische Sicht des Problems. Hardwarenahe Implementierungen werden transparent gehalten. Dies ist möglich über die Einführung gewisser abstrakter Datentypen für Vektoren (`Vec`), Matrizen (`Mat`) und Indexmengen (`IS`), s. a. Abb. 2.1. Über interne Nutzung von MPI-Direktiven wird Skalierbarkeit der Algorithmen ermöglicht. Es wurden bereits implizite Probleme mit über $500 \cdot 10^9$ Unbekannten gerechnet oder über $1.5 \cdot 10^6$ Rechenkerne effizient genutzt, /KNE 16/. Bei Bedarf ist PETSc aber auch ohne MPI-Unterstützung kompilierbar. Wie Funktionen aufgerufen und Variablen genutzt werden bleibt hiervon unbetroffen. Auf Skalierbarkeit gilt es dann jedoch zu verzichten.

PETSc ist *portable* in zweierlei Hinsicht: Zum einen existieren Schnittstellen zu wesentlichen Programmiersprachen wie C, C++, FORTRAN und Python. Zum anderen wurde Wert auf eine gewisse Plattformunabhängigkeit gelegt. Nativ in Linux angesiedelt, steht PETSc mit Hilfe von Cygwin, /SOL 16/, auch in einer Windows-Umgebung zur Verfügung.

[1] benannt nach der gleichnamigen Titanin aus der griechischen Mythologie; griechisch für *Weisheit*

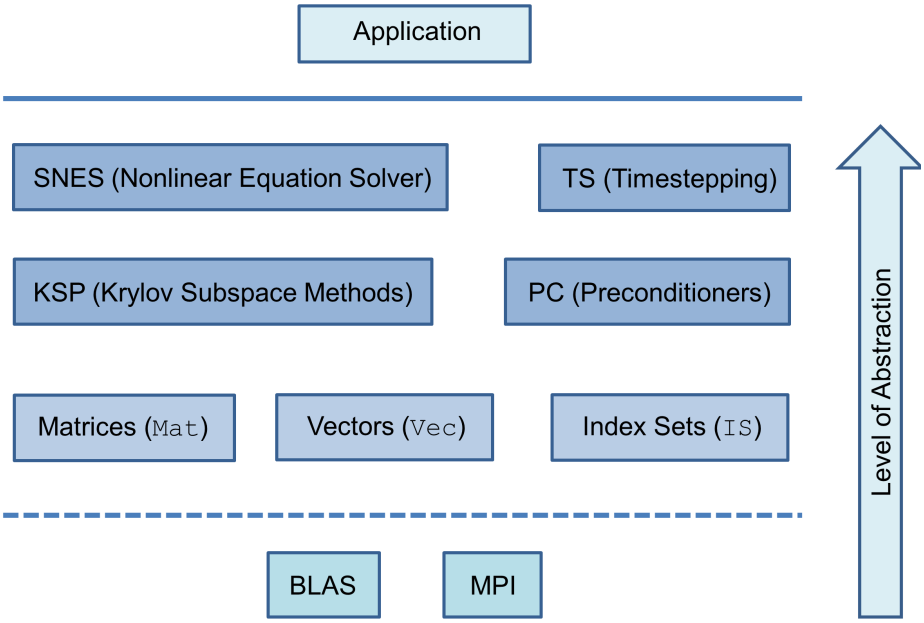


Abb. 2.1 PETSc-Architektur, s. auch das Diagramm auf /BAL 16/

PETSc ist *extensible*; das heißt, dass grundsätzlich andere eigenständige Bibliotheken an PETSc gekoppelt werden können. Dies ist auch bereits mit einer Vielzahl von Bibliotheken geschehen, s. /BAL 16/. Für den Nutzer wird diese Kopplung transparent gehalten. Das Nutzen von Funktionen der weiteren Bibliothek erfolgt mittelbar über PETSc-Strukturen. Dies macht es leicht, eine fest definierte mathematische Problemstellung, deren Datenstrukturen bereits in PETSc etabliert sind, mit verschiedenen Lösungsalgorithmen anzugehen. Im Rahmen dieser Arbeit bezieht sich dies auf das Lösen linearer Gleichungssysteme: PETSc bietet verschiedene iterative Krylov-Unterraum-Methoden wie GMRES, Bi-CG-Stab oder TFQMR – für eine mathematische Erläuterung dieser Verfahren siehe z. B. /MEI 11/. Ein skalierbarer direkter Löser wird nicht direkt von PETSc gestellt. Es wird aber u. a. eine Schnittstelle zum direkten Löser MUMPS angeboten, welcher wie bereits erwähnt für diese Arbeit ebenfalls berücksichtigt wird. Zur näheren Erläuterung der Software siehe unten.

Die Entwicklung zu PETSc kann auf eine 25-jährige Geschichte zurückblicken. Gefördert wird das Projekt vom U. S. Department of Energy, der National Science Foundation sowie dem Intel Parallel Computing Center, /KNE 16/. Somit ist auch in der Zukunft mit einem Erhalt der Bibliothek zu rechnen. Ihr Einsatz erfolgt in zahlreichen Softwareprodukten zur finiten Element/Volumen-Methode, zur Eigenwertberechnung oder auch in Simulationscodes zur Physiologie im menschlichen Körper, /BAL 16/. Dies hat eine diverse und breite Community zur Folge. Die Dokumentation ist vielfältig und mit vielen Beispielen unterlegt.

Für einen direkten Kontakt zu den Entwicklern existiert eine umfangreiche und zeitnah gepflegte Mailing-Liste. Ebenso werden seit einigen Jahren User-Meetings angeboten, an denen ebenfalls eine Vielzahl der Entwickler teilnimmt. So besteht ein direkter Austausch mit der Community.

Bemerkung 2.3. Im Rahmen des PETSc User-Meeting 2017 wurden die in diesem Projekt erzielten Ergebnisse in einem Vortrag vorgestellt, /JAC 17b/, welcher von den Entwicklern als sehr positiv beurteilt wurde, s. /STE 17/.

Betrachtet man die Summe der oben erläuterten Vorzüge, so eignet sich die Software PETSc als *Basisbibliothek*, um hierüber dem GRS-eigenen ATHLET-Code über die in Abschnitt 2.3 diskutierte Schnittstellenarchitektur effiziente Datenstrukturen und Algorithmen zur Verfügung zu stellen. Den in Abschnitt 2.2.1 formulierten Anforderungen hinsichtlich der effizienten Bestimmung der Jacobimatrix über finite Differenzen kann ebenfalls entsprochen werden, da in der verwandten PETSc-Version entsprechende Hilfsalgorithmen zur Verfügung stehen, s. a. Bemerkung 2.5 sowie Unterabschnitt 2.3.1.4. Zur Lizenzfrage s. Abschnitt 2.2.4.

Bemerkung 2.4. Zwar bietet die PETSc-Bibliothek auch DGL-Löser an, jedoch wird zum einen keine (verallgemeinerte) W-Methode mit hoher Stufenordnung angeboten. Und zum anderen erweisen sich jene Löser in ihrer General-Purpose-Funktionsweise nicht als besonders geeignet im ATHLET-Kontext, s. hierzu auch die Diskussion zu Beginn von Kapitel 3 sowie Abschnitt 3.6.

Bemerkung 2.5. In Bezug auf Hilfsalgorithmen zur Bestimmung der Jacobimatrix über finite Differenzen wurde für diese Arbeit zunächst der Bibliothek COLPACK, /GEB 13/, in der Version 1.0.9 der Vorzug gegeben. Dies lag daran, dass die entsprechenden Algorithmen in PETSc zu jener Zeit noch nicht in der jetzigen klar ausformulierten und intuitiv nutzbaren Form zur Verfügung standen. COLPACK bietet weitere effiziente Heuristiken an, welche jedoch nur gewinnbringend ausgenutzt werden können, wenn zur Bestimmung der Jacobimatrixeinträge der Rückwärtsmodus der Automatischen Differentiation (AD) zur Verfügung stünde. Siehe /GEB 13/ und /GRI 08/ zu Theorie und Techniken. Es ist jedoch nicht Teil dieser Arbeit, sich der komplexen Aufgabe zu widmen, ATHLET AD-kompatibel zu gestalten. COLPACK bringt somit im Vergleich zur aktuellen PETSc-Version keinen Mehrwert. Darüber hinaus hat es den Anschein, als habe keine wesentliche Entwicklung bzgl. des COLPACK-Projektes seit 2013 stattgefunden. Teils werden Anweisungen genutzt, die nicht dem aktuellen 4.x-Standard für OpenMP entsprechen, was eine Kompilierung erschwert. Auch ist keine native FORTRAN-Anbindung vorhanden, die erst

eigenständig ausgearbeitet werden musste. Für den weiteren Verlauf der Arbeit wurde davon abgesehen, COLPACK zu berücksichtigen.

Bemerkung 2.6. Über die in Abschnitt 2.3.1 diskutierte Architektur wird die Jacobimatrix in der PETSc-eigenen Datenstruktur `Mat` abgelegt. Weitere Formate bereitzustellen, ist nicht notwendig, da der einheitliche Weg über PETSc-Direktiven gewählt wurde, um verschiedene Löser aufzurufen. Um eine bestimmte Jacobimatrix auch außerhalb des ATHLET-Kontextes nutzen zu können, wurde über die Architektur der Zugriff auf die PETSc-Routinen zur Ablage einer Matrix in einer Datei im COO-Format (coordinate list) etabliert. Hiermit ist es ein Leichtes, die Matrix in numerischen Programmen wie Scilab, /ENT 16/, einzulesen. Es besteht somit eine Möglichkeit, das Potential solcher Programme hinsichtlich der Untersuchung von numerischen Eigenschaften der Jacobimatrix auszunutzen. Ebenso können bei bilateralem Interesse dem akademischen Umfeld Matrizen zu Forschungszwecken zur Verfügung gestellt werden.

Bemerkung 2.7. Auf die PETSc-Bibliothek wurde im ATHLET-Kontext bereits in /JAC 13/ zugegriffen. Innerhalb jenes Rahmens wurde sich ausschließlich auf das Lösen der im ATHLET auftretenden linearen Gleichungssysteme konzentriert und erste Tests hinsichtlich Performance-Verbesserungen durchgeführt. Hierfür kam auch der an PETSc koppelbare, direkte Löser UMFPACK, /DAV 04/, zum Einsatz, der jedoch aus Lizenzgründen nicht für Produktiv-Codes geeignet ist. Die damaligen Code-Modifikationen sind nur sehr rudimentär nützlich für diese Arbeit, da sehr viel später im Code in die PETSc-Bibliothek verzweigt wurde und redundante ATHLET-Strukturen aufrechterhalten werden mussten. Der damalige gute Eindruck zur Handhabung und Effizienz von PETSc nützt aber auch an dieser Stelle im Sinne der Bestärkung der Entscheidung, neue Strukturen auf PETSc fußen zu lassen.

Bemerkung 2.8. Das Konzept, abstrakte Datenstrukturen und Funktionalitäten über (externe) Pakete zur Verfügung zu stellen, ist auch dem Projekt *Trilinos*, /HER 03/, zu eigen. Der Aufbau und die Nutzung erscheint den Autoren dieser Arbeit jedoch weniger intuitiv zugänglich. Ebenso steht das objekt-orientierte FORTRAN-Interface *ForTrilinos* noch nicht stabil in Konjunktion mit Intel-Compilern zur Verfügung, s. hierzu die Beschreibung von *ForTrilinos* auf /SAN 16/. Zusätzlich ist die Unterstützung von direkten Lösern wie MUMPS über das Trilinos-Paket *Amesos* zurzeit nicht an die aktuellen Versionen angepasst. Aufgrund dieser Mali findet Trilinos für diese Arbeit keine Berücksichtigung.

MUMPS

Durch PETSc selbst wird bereits eine hinreichend große Gruppe iterativer Krylov-Unterraum-Methoden zur Verfügung gestellt. Ein auf Skalierbarkeit ausgelegter direkter Löser

fehlt jedoch. Nach /LI 13/ stehen hierfür generell eine Reihe von Kandidaten zur Verfügung. Da sich für den einheitlichen Zugang über PETSc-Datenstrukturen entschieden wurde, verkleinert sich die Liste im Wesentlichen auf die Programme MUMPS, /AME 16/ und SuperLU, /DEM 99/. Bis dato wird nur auf MUMPS zugegriffen, da SuperLU Probleme beim Kompilieren machte. Während iterative Methoden jeweils andere Maße zur Steuerung der Iterationszahl nutzen, sind Unterschiede in der Ergebnisapproximation selbst bei exakter Arithmetik sehr viel eher zu erwarten, als es für direkte Verfahren der Fall ist. Denn letztere basieren auf dem Gauß-Algorithmus und sollten ohne Berücksichtigung von Rundungsfehlern identische Ergebnisse liefern. Insofern ist es nicht allzu kritisch, sich auf einen Kandidaten zur Klasse der direkten Löser zu beschränken.

Bemerkung 2.9. Der Gauß-Algorithmus in Matrixnotation führt zur LR-Zerlegung der betrachteten Matrix. Diese kann dann in einer Rückwärts- und Vorwärtssubstitution genutzt werden, um effizient lineare Gleichungssysteme zu lösen. Man beachte, dass im Englischen der R-Faktor (abkürzend für *Rechts*) mit U (abkürzend für *Upper*) bezeichnet wird.

Die initiale Entwicklung von MUMPS fand im Zeitraum 1996-1999 für das europäische Projekt PARASOL statt. Die aktuelle Version wird von Mitarbeitern der Institutionen CERFACS, ENS Lyon, INPT(ENSEEIHT)-IRIT, Inria und der Université de Bordeaux entwickelt, /AME 17/. MUMPS erfährt eine Teilfinanzierung als *CEC ESPRIT IV long term research project – No. 20160 (PARASOL)*, /AME 16/. Die Software wird in vielerlei Projekten aus Wissenschaft und Industrie erfolgreich eingesetzt, s. ebenfalls /AME 16/. Besonders erwähnenswert ist hierbei der auf PETSc aufbauende Eigenwertlöser SLEPc, /ROM 16/, da in diesem Kontext mitunter sehr schlecht konditionierte Matrizen auftreten und MUMPS offensichtlich den Anforderungen an einen stabilen Algorithmus genügt.

Zur Lösung von dünnbesetzten linearen Gleichungssystemen implementiert MUMPS den Multifrontal-Ansatz zur LR-Zerlegung, s. /AME 00/ sowie /DAV 06/ für theoretische Erörterungen hierzu. Dieser ist nativ auf Parallelität ausgelegt. In MUMPS wird eine solche Parallelität über MPI realisiert. Auf eine Unterstützung hiervon kann aber auch während der Kompilierung verzichtet werden. Interne Skalierung und optionales Iterative-Refinement, welches eine Schätzung des Rückwärtsfehlers berücksichtigt, tragen zur Robustheit des Lösers bei. Man beachte, dass diese beiden Features zurzeit nur im Ein-Prozess-Modus im vollen Umfang zur Verfügung stehen, da die Matrix in diesem Fall nicht verteilt gespeichert ist. Eine pro Prozess lokale Skalierung exisitert aber auch bei verteilter Speicherweise.

Nach /LI 13/ besteht die Anwendung eines direkten Lösers für dünnbesetzte Systeme aus vier Schritten:

1. Permutationsschritt – Zeilen und Spalten der Matrix werden derart umsortiert, dass möglichst wenig Fill-In während einer LR-Zerlegung produziert wird.

2. Analyseschritt – eine symbolische Zerlegung wird durchgeführt, um die Nicht-Null-Strukturen der Faktoren L und R zu ermitteln und anschließend geeignete Datenstrukturen hierzu zu etablieren.

3. Zerlegungsschritt – eine numerische Faktorisierung findet statt, um konkret L und R zu bestimmen.

4. Lösungsschritt – zu gegebenen rechten Seiten wird das lineare Gleichungssystem mit Hilfe der Faktoren L und R über eine Rückwärts- und anschließende Vorwärtssubstitution gelöst.

Durch MUMPS werden alle vier Schritte abgedeckt, wobei für den ersten Schritt auch externe Bibliotheken eingebunden werden können, s. nächsten Abschnitt.

Bemerkung 2.10. Es gilt zu beachten, dass der Permutationsschritt lediglich auf der Besetztheitsstruktur der Jacobimatrix arbeitet. Aufgrund von (partial) pivoting kann es zu weiteren Permutationen im 3. Schritt kommen, um ein numerisch stabileres Vorgehen in Schritt 4 zu ermöglichen. Hierdurch können sich im Falle von dünnbesetzten Matrizen signifikante Änderungen an der durch Schritt 2 prognostizierten Besetztheitsstruktur des L und R-Faktors ergeben, was zu einer merklichen Erhöhung des Aufwandes und Speicherbedarfs für Schritt 3 führen kann, s. hierzu auch Abschnitt 6.2.3. MUMPS bietet daher die Option des s. g. *partial threshold pivotings*, /AME 17, §3.9/, welches den Pivotisierungsprozess über einen durch den Nutzer wählbaren Threshold-Parameter $u \in [0, 1]$ weniger stringent gestaltet. Für $u = 0$ findet gar keine Pivotisierung statt, während $u = 1$ das klassische partial pivoting forciert. Abhängig vom Anwendungszenario wurden für die Rechenläufe zu diesem Projekt die in /AME 17/ empfohlenen Werte $u = 10^{-k},\ k = 1, 2$, fürs direkte Lösen sowie der Wert $u = 10^{-4}$ im Sinne der Präkonditionierung beim Hybridansatz genutzt.

(Par)METIS

Der Permutationsschritt zur Fill-In-Reduzierung in obiger Aufzählung zu den einzelnen Phasen eines direkten Lösers sieht sich einem NP-schweren Problem gegenüber, s. den Verweis in /SCH 02/. Folglich sind Heuristiken zu bemühen. Diese stammen aus

der Graphentheorie, da sich das Fill-In-Problem äquivalent in diesem Kontext darstellen lässt, s. z. B. /SCH 02/. MUMPS bietet hierfür eine Vielzahl von Algorithmen: *Approximate Minimum Degree* (AMD), *Approximate Minimum Degree with automatic quasi-dense row detection* (QAMD), *Approximate Minimum Fill* (AMF) sowie Schnittstellen zu den Graphpartitionierern SCOTCH, PORD und METIS. Quellenangaben zu den einzelnen Algorithmen sind in /AME 17/ zu finden.

Für diese Arbeit wurde sich dafür entschieden, standardmäßig den Permutationsschritt über den Graphpartitionierer METIS von der University of Minnesota durchzuführen, welcher in /AME 17/ als *strongly recommended* gekennzeichnet ist. METIS basiert auf dem *multilevel nested dissection paradigm*, /KAR 99/, und laut eigener Aussage auf /KAR 16/ sind deutlich bessere Ergebnisse sowohl hinsichtlich der Fill-In-Generierung als auch bzgl. der Laufzeit zu erwarten – gerade im Vergleich zu verschiedenen Ausprägungen des Minimum Degree Algorithmus. Zu METIS existiert auch eine auf MPI basierende parallele Version ParMETIS. Beide Versionen sind direkt über entsprechende Flags in MUMPS ansprechbar.

2.2.3 Parallelität

Die drei aufgeführten Bibliotheken sind initial ohne MPI-Unterstützung kompiliert worden. Mit der Anpassung der Inhalte des Arbeitspaketes 3 hin zu einer MPI-basierten Softwarearchitektur ergab sich aber die Möglichkeit, auf das volle Potential der Bibliotheken bezüglich verteilter Speicherung und Berechnung zugreifen zu können. Hierbei ist die Wahl der Implementierungen des MPI-Standards plattformabhängig:

- Linux – OpenMPI 2.0.1 (MPI Rev. 3.1)
- Windows – Microsoft MPI v8 (MPI Rev. 2)

Die MPI-Implementierung von Microsoft ist de facto der Standard auf der MS/Windows-Plattform. OpenMPI war bereits auf dem Linux-Cluster der GRS installiert und wird dort aktuell gehalten. Beide Implementierungen werden kostenfrei zur Verfügung gestellt. Zu Lizenzbetrachtungen s. nächsten Abschnitt.

Aufgrund der unterschiedlichen Standards ist es plattformabhängig, wie die MPI-Module in den eigenen Code einzubinden sind. Ebenso gibt es Abweichungen in den verwendeten Datentypen. Da die Diskrepanzen sehr überschaubar sind, wird eine Kompatibilität zu beiden Standards über Präprozessor-Direktiven im Code sichergestellt. Für beide Plattformen wird somit der gleiche FORTRAN-Code genutzt, was der Pflege und Wartung sehr zugutekommt.

Bemerkung 2.11. Für die kostenpflichtigen MPI-Bibliotheken von Intel standen für dieses Projekt keine Lizenzen zur Verfügung.

Bemerkung 2.12. Im ATHLET-Code wird bereits eine Form der parallelen Verarbeitung optional genutzt. Innerhalb der Routinen zur Auswertung von f Abschnitt 2.3.1 aus (1.1) sind einige Bereiche durch OpenMP-Direktiven, /TIR 17/, erweitert, so dass hier über temporäre Prozessausdehnung eine Beschleunigung in der Berechnung von f stattfindet. Die für diese Arbeit in Abschnitt 2.3.1 erläuterte Architektur ist vollständig kompatibel hierzu, s. speziell Unterabschnitt 2.3.1.2.

2.2.4 Lizenzbetrachtungen

Die Weitergabe und Nutzungsvereinbarungen zum GRS-Code ATHLET werden auf Basis von /SCH 13/ geregelt. Auszugsweise seien folgende Zitate gegeben:

- Abschnitt 2.2 – *Inhalte der Lizenzverträge zur Codeüberlassung*

  > Nach positiver Entscheidung durch die Geschäftsführung und dem zuständigen Bereichsleiter über die Codeüberlassung schließt die GRS mit der anfragenden Stelle eine „VEREINBARUNG ZUR NUTZUNG VON DV-PROGRAMMEN“ (bzw. „Software License Agreement“, ANHANG 3). Diese regelt die sich aus der Überlassung ergebenden Rechte und Pflichten des künftigen Nutzers und der GRS unter anderem hinsichtlich Haftung, Gewährleistung, Geheimhaltung und Weiterentwicklung des Codes.

- Abschnitt 5 – *Zustimmung durch das BMWi*

  > Die GRS holt vor einer Weitergabe von Codes an ausländische Stellen eine schriftliche Zustimmung des BMWi ein, bei der Überlassung von Codes an nationale Stellen wird das BMWI lediglich informiert.

- Abschnitt 7 – *Regelungen für die Überlassung von Sourcecodes oder Teilmodulen hiervon*

  > Es ist eine Kooperation seitens der GRS mit dem „Anfragenden“ vorgesehen, die auf eine Weiterentwicklung von Modellen des Codes durch den „Anfragenden“ hinausläuft. Hier ist im Einzelfall zu entscheiden, in welchem Umfang der Sourcecode herausgegeben wird. Grundsätzlich sollte nie der ganze Sourcecode herausgegeben werden, da nur die GRS die jeweils aktuelle Codeversion verwalten und handhaben sollte.

Dieser Rahmen liefert gewisse Einschränkungen, auf welche (kostenfreie) externe Software zugegriffen werden kann.

Bemerkung 2.13. Nachfolgende Ausführungen über Lizenzen zu Softwareprodukten sind das Ergebnis entsprechender Recherchen der Autoren dieser Arbeit. Hinsichtlich der Kompatibilität zu den Bestimmungen der GRS sind die aufgeführten Aussagen als Einschätzung zu verstehen. Juristisch abgesicherte Bewertungen werden nicht gegeben.

Copyleft

Ausgeschlossen werden muss Software, welche mit einem starken *Copyleft* versehen ist. Dieser Begriff wurde im Kontext der *GNU General Public License* (GPL) geprägt, /STA 15/,/WIK 16b/, und besagt, dass ein abgeleitetes Softwareprodukt ebenfalls zu den Bedingungen der GPL zu lizenzieren sei, falls dieses an Dritte weitergegeben werde. Unter GPL stehende Softwareprodukte gelten als *freie Software*, welches ein Offenlegen des Sourcecodes einschließt. *Würde ein GRS-Code auf externe Software unter GPL zugreifen, müsste aufgrund des Copylefts auch der GRS-Code seine Quelldateien offen legen.* Dies ist nach obigen Anforderungen nicht kompatibel zu den GRS-Richtlinien. Beispielweise heißt dies, dass der unter GPL stehende direkte Löser UMFPACK nicht in GRS-Produktiv-Code zum Einsatz kommen kann.

Lizenzen ohne starkes Copyleft

Als Erweiterung zur GPL wurde 1991 die *GNU Lesser General Public License* (LGPL) entwickelt, /FRE 07a/, welche nur noch ein *schwaches* Copyleft aufweist. Praktisch heißt dies, dass auch in proprietärer Software auf Bibliotheken mit dieser Lizenzvereinbarung zugegriffen werden kann, sofern diese dynamisch verlinkt werden und somit für einen Dritten klar die Trennung zwischen den einzelnen Softwareprodukten ersichtlich ist. Des Weiteren existieren zahlreiche weitere Lizenzmodelle, denen keinerlei Copyleft zu eigen ist. Hierzu gehören nach Ansicht der Autoren die Lizenzen, welche die für diese Arbeit verwandten Bibliotheken/MPI-Implementierungen abdecken. Informationen bezgüglich der numerischen Bibliotheken finden sich in Tab. 2.1. Zu den verwandten MPI-Implementierungen sind Informationen dieser Art in Tab. 2.2 aufgelistet.

Bemerkung 2.14. Zur Präkonditionierung im Falle von Krylov-Unterraum-Methoden stehen auch Funktionen aus der Bibliothek Hypre, /LAB 16/, zur Verfügung. Auf den Einsatz wurde bisher verzichtet und stattdessen MUMPS als optionaler Präkonditionierer bei verteilter Rechnung gewählt. Hypre sei aber als Beispiel für eine potentiell interessante Bibliothek genannt, welche unter der LGPL steht.

Tab. 2.1 Bibliotheken und ihre Lizenzvereinbarungen

	PETSc 3.7.6	MUMPS 5.1.1-p3	METIS 5.1.0-p3 ParMETIS 4.0.3
Lizenz	2-clause BSD	CeCill-C	Apache 2.0
Verweis	/WIK 16a/	/CEA 06/	/THE 04/

BSD: Berkeley Software Distribution
CeCiLL: *CEA CNRS INRIA* Logiciel Libre

Bemerkung 2.15. Die aufgeführten numerischen Bibliotheken greifen für grundlegende Operationen der linearen Algebra auf LAPACK (Linear Algebra PACKage) und hierin auf BLAS (Basic Linear Algebra Subroutines) zurück. Dabei wird für den sequentiellen Fall die Implementierung von Intel in deren Math Kernel Library (MKL) genutzt, da hierfür eine Lizenz vorhanden ist. Für verteilte Rechnungen wird das über netlib zur Verfügung gestellte Paket ScaLAPACK (Scalable Linear Algebra PACKage) inklusive BLACS (Basic Linear Algebra Communication Subprograms) angewandt, s. /TEN 17/ für weitere Informationen zu ScaLAPACK. Ebenfalls nach /TEN 17/ steht diese Software unter der *modified BSD license*, /WIK 16a/, was keine Probleme bei der Verwendung bereiten sollte.

Tab. 2.2 MPI-Implementierungen und ihre Lizenzvereinbarungen

	MSMPI v8	OpenMPI 2.0.1
Lizenz	MS Software License Terms	2-clause BSD
Verweis	der Installation beigefügt	/WIK 16a/

Windows/Cygwin

PETSc ist nicht nativ auf Windows ausgelegt. Auf /BAL 16/ wird in der Installationsanleitung auf die Nutzung des Tools *Cygwin* verwiesen, welches eine linuxähnliche Umgebung unter Windows bereitstellt. Zurzeit wird die Erstellung einer dynamischen Bibliothek innerhalb einer Windows-Umgebung von PETSc nicht unterstützt. Dies ist erst mit der nächsten Nebenversion zu erwarten. Ob der in Tab. 2.1 genannten Lizenzmodelle sollte aber keine lizenztechnisch problematische Abhängigkeit generiert werden, auch wenn die drei Bibliotheken statisch gelinkt werden. Eine endgültige Klärung steht jedoch noch aus.

MPI-basierte Softwarearchitektur

Die konstruierte Architektur basiert auf einer MPI-gesteuerten Interprozesskommunikation zweier voneinander getrennter Programme (executables). Hierbei steht ATHLET auf der

einen Seite, während der Gegenpart durch das in Abschnitt 2.3 erläuterte Numerical Toolkit gegeben ist. Sämtliche numerischen Bibliotheken sind mit diesem Toolkit verknüpft. Lizenzfragen zu diesen Bibliotheken beziehen sich also ausschließlich auf das Toolkit. Durch die Interprozesskommunikation wird also auch im Nebeneffekt eine Art „Lizenzbarriere“ hinsichtlich bestehender GRS-Software etabliert, s. auch Abb. 2.2.

2.2.5 Kompilierung und Initialtest

Über die Konfigurationseinstellungen von PETSc lassen sich die einzelnen Bibliotheken in einem gemeinsamen Kompilierungsprozess erstellen und je nach Einstellung als dynamische oder statische Gesamtbibliothek zusammenfassen. Für die Windows-Plattform gilt zurzeit leider nur die zweite Option. Der Kompilierungsprozess der ausgewählten Bibliotheken wird zentral über die Konfiguration von PETSc gesteuert. Unter Linux wurde eine dynamische Bibliothek gebaut, während unter Windows ob der vorgegebenen Restriktion eine statische Bibliothek angelegt wurde.

Um den grundsätzlichen Zugriff auf die Gesamtbibliothek zu testen, wurde sowohl unter Windows als auch Linux erfolgreich das PETSc-eigene Beispiel *ex19* (Nonlinear PDE on a structured grid) ausgeführt.

Bemerkung 2.16. Die teils recht umfangreichen Konfigurationsanweisungen können auf Anfrage zur Verfügung gestellt werden.

2.3 Schnittstellenarchitektur

Zum besseren Verständnis bietet es sich an, den ATHLET-Code aus der Sicht eines DGL-Lösers zu betrachten. Die Modellierungskompetenz und das implementierte physikalische Wissen finden sich in der rechten Seite von (1.1) wieder. Das anstehende AWP wird dann vom ATHLET-eigenen Löser FEBE (Forward Euler/Backward Euler) angegangen, welcher ein Extrapolationsverfahren auf Basis des linear-impliziten Eulers realisiert, s. Abschnitt 3.2 für eine ausführliche mathematische Diskussion des Verfahrens. Dieser Löser ist in seinem Kontrollverhalten stark an den ATHLET-Kontext angepasst. Solche Anpassungen sind bereits zu einem Teil für dieses Projekt berücksichtigt worden. Betrachtet man ausschließlich den Anteil der linearen Algebra im Löser, so wird diese bereits durch den neuen Code komplett abgehandelt mit Hilfe der im vorherigen Abschnitt diskutierten Bibliotheken.

Bemerkung 2.17. Ursprünglich wurde FEBE als paritionierende Methode konstruiert. Hierbei wird das Gesamtsystem (1.1) und somit y in zwei Teile aufgeteilt. Ein Teil wird als nicht-steif angesehen und mit einer expliziten Methode bearbeitet. Im Falle von FEBE wäre dies der explizite Euler. Der zweite Teil wird als steif betrachtet und implizit behandelt. Im FEBE-Kontext kommt hierfür der linear-implizite Euler zum Einsatz. Dies erklärt die Bezeichnung des DGL-Lösers, da im Englischen der explizite Euler auch als Forward Euler und der (linear-)implizite Euler als Backward Euler bezeichnet werden kann. Vom algorithmischen Ablauf her hat es für die anschließende Extrapolation keinen Einfluss, ob partitioniert wurde oder nicht. Nach /HAI 96, § IV.10/ zeigt sich bei diesem Ansatz die Schwierigkeit, welche Gleichungen als steif zu betrachten seien. Ebenso sei es wahrscheinlich, dass sich Steifheit in Unterräumen ausbildet, die nicht parallel zu den Koordinatenachsen liegen. Der Ansatz der partitionierenden Methode bietet demnach gewisse Mängel. Innerhalb des ATHLET-Kontextes wird dem dadurch Rechnung getragen, dass in der aktuellen Fassung keine Partitionierung stattfindet und standardmäßig auf den linear-impliziten Euler zurückgegriffen wird. Eine Ausnahme tritt auf, wenn die Jacobimatrix J nicht über finite Differenzen konstruiert werden kann, weil die gestörten Auswertungen von f in (2.1) oder (2.2) kein valides Ergebnis liefern. In diesem Fall wird auf den expliziten Euler gewechselt, was implizit der Wahl $J = 0$ entspricht.

ATHLET – Ausgangssituation und Hemmnisse

Die Aspekte der Schrittweiten- und Fehlerkontrolle des DGL-Lösers sowie ein gewisses Überwachen der Qualität der Jacobimatrix findet sich hauptsächlich in der Datei `febe.f90` wieder. Die Logik zur linearen Algebra ist auch bereits im ATHLET formal ausgelagert und wird zentral durch `ftrix.f90` gesteuert. De facto wird dort ein direkter Löser für dünnbesetzte Systeme implementiert, der im Wesentlichen der allgemeinen Arbeitsschrittfolge auf Seite 17 genügt. Sowohl `febe.f90` als auch `ftrix.f90` werden durch *zahlreiche* Hilfsfunktionen unterstützt.

Das diffizile Zusammenspiel der einzelnen ATHLET-Funktionalitäten im `febe`/`ftrix`-Kontext in einem hinreichenden Maß zu fassen und zu durchdringen stellt sich als ein sehr zeitraubender Prozess dar. Zwar ist aufgrund der ausführlichen Literatur, s. /AUS 16a/, /AUS 16b/, /LER 16a/, recht schnell erkennbar, was grundsätzlich im Code passiert. Jedoch greifen die in dieser Arbeit anzugehenden Modifikationen auf Detailebene an. Die nachfolgenden Punkte stellen es teils als sehr herausfordernd dar, wo genau anzusetzen und welcher globale oder lokale Kontext zu berücksichtigen ist:

- Die letzten Codeänderungen im `febe`/`ftrix`-Kontext sind überwiegend vor ca. 20 Jahren geschehen. Erstimplementationen sind teils bis zu 30 Jahre alt. Somit ist

FORTRAN77-Stil vorherrschend. Dies geht einher mit alten Programmierparadigmen und Konventionen:

  - Es findet eine exzessive Nutzung von `goto`-Anweisungen und Sprungmarken statt – eine Nachvollziehbarkeit, was der Code in welcher Reihenfolge macht, wird hierdurch erschwert, s. hierzu auch /DIJ 68/.
  - Der Namensraum ist global. Aus einer Routine heraus lassen sich ohne explizite Einbindung andere Funktionen aufrufen. Nicht selten sind diese Routinen ohne oder nur mit rudimentärer Argumentliste ausgestattet. Ein Großteil der Kommunikation läuft über in Modulen abgelegte globale Variablen. Aus einem lokalen Kontext wie „`subroutine a` ruft `subroutine b` auf“ lässt sich nicht nachvollziehen, welche Änderungen am Status des Programms hierdurch vollzogen werden.
  - Die Bezeichner von Variablen und Funktionen sind oft kryptisch, was vermutlich einer Maxime geschuldet ist, nicht mehr als sechs Zeichen hierfür zu verwenden. Hinweise auf Einsatz und Aufgabe werden hierdurch oft nicht geliefert. Hinzukommt, dass alle Bezeichner konsequent in Majuskel-Form geschrieben sind.

- Die meisten Entwickler für den numerischen Code im `febe`/`ftrix`-Kontext stehen für einen persönlichen Austausch nicht mehr zur Verfügung, da sie aus der GRS ausgeschieden sind. Ohne direkte Konsultation der Programmierer ist die schriftliche Dokumentation zu Details nach heutigem Stand verbesserungsfähig.
- Die Kommentare sind in der Regel hilfreich. Zu manchen Code-Anweisungen sind diese jedoch nicht zielgerichtet verständlich und folgen keinem modernen Standard. Gerade hinsichtlich numerisch geprägter Abfragen in `febe` wäre oft eine ausführlichere Begründung für die Motivation des Vorgehens hilfreich.

Bemerkung 2.18. Der ATHLET-Code bietet im Bereich der Numerik erhebliches Verbesserungspotential hinsichtlich Effizienz, Flexibilität, Verständlichkeit und Wartbarkeit. Hierbei ist jedoch nicht außer Acht zu lassen, dass es sich um einen funktionierenden Code handelt. Insofern ist gerade bei den speziell angepassten Kontrollstrukturen im Ablauf von FEBE darauf zu achten, diese nicht zu übergehen, sondern so gut es geht in adäquat verallgemeinerter Form zu berücksichtigen. Dieser Prozess ist im Rahmen dieses Projektes weit vorangeschritten, jedoch im Sinne der Nutzung externer DGL-Numerik noch ausbaufähig. Näheres hierzu findet sich in der Einleitung zu Kapitel 3.

Im Sinne der Nachvollziehbarkeit und der klaren Trennung von semantisch unterschiedlich belegten Codebereichen, kann es nicht Anliegen dieser Arbeit sein, dem bunten Reigen an Funktionen und Modulen in gleicher wie oben beschriebener Weise weitere zur

Seite zu stellen. Erweiterungen dieser Art müssen auf ein Minimum beschränkt werden. Glücklicherweise steht ein Weg zur Verfügung, der dieser Anforderung genügt.

Ausnutzen des ATHLET-Plug-In-Konzept – Kapselung des Numerical Toolkits
Parallel zu den Arbeiten an diesem Projekt wurde im Rahmen anderer Aufgabenfelder in der GRS die Entwicklung der Bibliothek *libadt* initiiert. Der Ausbau des Funktionsumfanges läuft fortwährend. Die Abkürzung *adt* steht für *abstract data types*. Mittels dieser Bibliothek ist es u. a. möglich, Plug-Ins für den ATHLET-Code zu nutzen. Ein Plug-In zu schreiben und einzubringen, birgt einige wichtige Vorteile im Rahmen dieses Arbeitspaketes:

- Sämtliche rein numerische Funktionalitäten können vom ATHLET-Kontext weggekapselt werden. Zusätzlich besteht die Möglichkeit, die direkte Kommunikation mit dem Plug-In seitens ATHLET in einem Modul zusammenzufassen. Durch Einbinden dieses Moduls steht dann der Funktionsumfang den üblichen ATHLET-Routinen zur Verfügung.
- Die Kommunikation zwischen dem ATHLET-Code und dem Plug-In läuft über fest definierte Schnittstellenfunktionen. Ein Datenaustausch findet ausschließlich hierüber statt.
- Das Plug-In kann unabhängig von ATHLET kompiliert und gepflegt werden. Wesentlich für die Nutzung ist nur, dass es zur Laufzeit zur Verfügung steht. Es kann auch dynamisch entschieden werden, ob es Anwendung finden soll oder nicht. Des Weiteren vererben sich keine Abhängigkeiten von weiterer Software auf das Hauptprogramm, i. e. ATHLET.
- Das Plug-In-Konzept zum ATHLET-Code wurde bereits an anderer Stelle genutzt. Als prominentes Beispiel sei an dieser Stelle der GRS-Code ATHLET-CD genannt, welcher in der Form eines Plug-Ins zur Laufzeit mit ATHLET gekoppelt wird. Die Implementierung des Plug-In-Konzeptes war somit bereits auf einem stabilen Stand, als mit den konkreten Arbeiten zum hier genutzten Plug-In begonnen wurde. Keine besonderen Maßnahmen der Anpassung waren vonnöten.
- Das Plug-In ist nicht an die Bezeichnerkonvention von ATHLET gebunden. Damit stehen mehr Zeichen sowie die Verwendung von Binnenmajuskeln zur Verfügung, um die Semantik von Variablen und Funktionen rascher erfassen zu können.

Da numerische Aufgaben durch das Plug-In gekapselt werden, wird die zugehörige Architektur mit dem Begriff *Numerical Toolkit*, kurz NuT, gekennzeichnet.

Bemerkung 2.19. Obige Punkte zeigen, wie überaus sinnvoll sich die Nutzung der libadt zeigt, um über das Plug-In-Konzept externe Numerik zur Verfügung zu stellen. Zu Beginn des Projektes waren diese vorteilhaften Rahmenbedingungen noch nicht gegeben. Da jedoch bekannt war, dass parallel an einer generellen Verbesserung der Handhabbarkeit des ATHLET-Codes gearbeitet wurde, war es eine bewusste Entscheidung, diese beobachtend abzuwarten. Dies hat sich für das Projekt dahingehend ausgezahlt, dass der erstellte Code *von Anfang an* den aktuellen Entwicklungsparadigmen genügt. Keine Parallel- oder Altstrukturen müssen aufgrund verfrühter Programmiertätigkeiten berücksichtigt werden.

2.3.1 NuT-Architektur

Bei der Erstellung der NuT-Architektur wurde sich an folgenden Paradigmen orientiert:

- *Sei minimal-invasiv.*
  Der allgemeine Entwicklungsprozess am Hauptprogramm sollte durch die numerischen Neuerungen so wenig wie möglich beeinflusst werden.
- *Jede Komponente sollte nur genau die Informationen kennen, die sie zwingend benötigt.*
  Globale Variablen sind weitestgehend zu verhindern, Funktionen sind in Modulen gekapselt. Eine Trennung zwischen konkreter Anforderung (ATHLET), mathematischer Funktionalität (NuT) und Umsetzung (PETSc) ist aufrechtzuerhalten.
- *Vergeude keine Systemressourcen.*
  Zum einen bezieht sich dieser Punkt auf die Nutzung effizienter Datenstrukturen; zum anderen ist hiermit auch gemeint, ein verteiltes Rechnen zu ermöglichen, sofern dies aufgrund entsprechender Problemgröße als vorteilhaft angesehen wird.
- *Sei flexibel.*
  Der Vorteil der Numerik ist, dass ab einem gewissen Level sehr gut vom Kontext abstrahiert werden kann. Dies macht einen Einsatz für vielfältige Problemstellungen möglich. Die Softwarearchitektur sollte diese Flexibilität unterstützen.

Aus diesen Anforderungen entwickelte sich eine Architektur, wie sie in Abb. 2.2 zu sehen ist. Im Wesentlichen besteht diese aus der Kopplung zweier Programm(teil)e. Auf der einen Seite steht ATHLET mit dem angedockten Plug-In. Auf der anderen Seite stehen die NuT-Worker-Prozesse, welche die gewünschten Funktionalitäten durch Aufrufe an PETSc realisieren. Flexibilität der NuT-Architektur sowie Skalierbarkeit der Rechenoperationen wird durch den Einsatz von MPI ermöglicht:

- Es können mehrere NuT-Worker-Prozesse gestartet werden, die dann gemeinsam über entsprechende PETSc-Aufrufe numerische Berechnungen durchführen, s. auch Unterabschnitt 2.3.1.2. Generell ist es somit sinnvoll, sich des Plurals zu bedienen. Auf der anderen Seite besteht natürlich stets die Option, nur einen NuT-Worker-Prozess zu starten. Gerade hinsichtlich kleinerer Probleme sollte dies im Hinterkopf behalten werden, da eine verteilte Rechnung erst ab einer gewissen Problemgröße lohnenswert ist, s. auch Kapitel 6.
- Aufgrund der MPI-basierten Interprozesskommunikation besteht eine gewisse Unabhängigkeit des Numerical Toolkits. Sofern die Gegenseite über das passende Interface verfügt, kann das Toolkit zum Einsatz gebracht werden.

Hinsichtlich weiterer Informationen zum Einsatz von MPI s. Kapitel 5.

Bemerkung 2.20. Die etablierte NuT-Architektur bedarf zwingend einer Hardwareumgebung, welche mindestens zwei Prozesse zur Verfügung stellen kann. nach heutigen Standards ist dies jedoch keine Einschränkung, da selbst Desktop-PCs/Laptops über Mehrkernprozessoren verfügen.

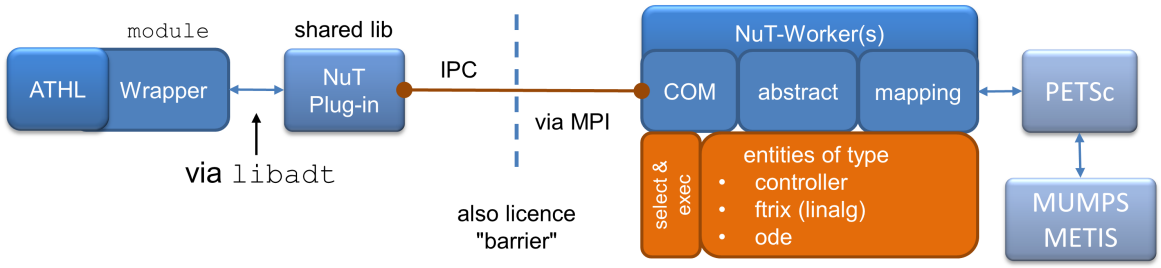


Abb. 2.2 NuT-Architektur – ATHLET und das Numerical Toolkit bedienen sich einer über MPI etablierten *Inter-Process-Communication* (IPC). Die blaue gestrichelte senkrechte Linie zeigt sowohl die logische als auch softwaretechnische Trennung der zwei Programme.

2.3.1.1 Komponenten innerhalb der Architektur

Jeder Komponente der in Abb. 2.2 dargestellten NuT-Architektur kommt eine abgegrenzte, fest definierte Rolle zu. Allen gemein ist die individuelle Wartbarkeit. Trennungen auf logischer Ebene gehen weitgehend mit Trennungen auf softwaretechnischer Ebene einher. Nachfolgend einige Erläuterungen zu den Komponenten.

ATHLET

ATHLET ist im Kontext der Architektur ein Programm, welches konkrete numerische Anforderungen hat, deren sich nach Übergabe der notwendigen Daten an anderer Stelle angenommen werden soll.

Wrapper

Der Wrapper kapselt den Teil der Architektur, der in Abb. 2.2 rechts von ATHLET angegeben ist. Programmiertechnisch ist der Wrapper als FORTRAN-Modul angelegt, welches leicht von bestehenden ATHLET-Routinen eingebunden werden kann. Er ist der ATHLET-Codebasis zugefügt, bedarf aber keiner gesonderten Behandlung. Lediglich der Zugriff auf die libadt muss gewährleistet sein, um Funktionen aus dem Plug-In über *procedure pointers* zur Laufzeit binden und nutzen zu können.

NuT-Plug-In

Das NuT-Plug-In kommuniziert in Richtung ATHLET ausschließlich mit dem Wrapper und stellt diesem einen Katalog an Funktionalitäten zur Verfügung. Das Plug-In ist konzeptuell als shared library angelegt und wird zur Laufzeit geladen. Im Plug-In selbst findet *keine* Implementierung numerischer Funktionalitäten statt. Stattdessen kapselt das Plug-In die MPI-basierte Inter-Process-Communication zu den NuT-Workers. Da es unabhängig von allen anderen Komponenten der Architektur erstellt wird, sind sowohl ATHLET als auch der Wrapper nicht von den zusätzlichen MPI-Abhängigkeiten betroffen, auf die das Plug-In angewiesen ist. Somit wird der eigentliche ATHLET-Entwicklungsprozess möglichst wenig beeinflusst. Die Signaturen zum Funktionenkatalog, welches das Plug-In anbietet, sind in Anhang A.1 zu finden. Die Einteilung orientiert sich an den Entitäten-Typen, die in den NuT-Workers genutzt werden.

NuT-Worker(s)

Ein NuT-Worker-Prozess ist logisch und programmiertechnisch in drei Teile aufgeteilt. Die interne Architektur ist auf dem Konzept von Entitäten aufgebaut. Diese verhalten sich zu einem gewissen Grad wie Objekte, wobei sich bewusst keiner objekt-orientierten Sprachkonstrukte in FORTRAN bedient wurde. Somit sind z. B. Polymorphismen oder Vererbung nicht Teil der Entitätenstruktur. Diese Konzepte werden aber auch nicht benötigt, und man umgeht etwaige Probleme durch eine nicht gänzlich ausgereifte Unterstützung einer objekt-orientierten Programmierung in FORTRAN seitens der Compiler.

Die Entitäten sind aufgabenorientiert typisiert, entsprechend der Aufteilung in Abb. 2.2. Der Controller-Entität fällt eine gesonderte Rolle zu, da diese stets die Erstkommunikation einer Anfrage übernimmt, in der die Selektion der Arbeitsentität sowie der konkrete Auftrag

an diese übermittelt wird. Der Controller schließt auch die MPI-Kommunikation ab, wenn der Worker-Prozess beendet wird.

Nach der Selektion übernimmt die gewählte Arbeitsentität die weitere aufgabenspezifische Kommunikation, um alle notwendigen Daten zu sammeln und anschließend nach den Berechnungen die Ergebnisse zu versenden. Die Abarbeitung einer konkreten Aufgabe erfolgt auf zwei Ebenen. Diese sind in Abb. 2.2 mit *abstract* und *mapping* gekennzeichnet. Hintergrund ist der, die abstrakte mathematische Anforderung von einer Implementierung hierzu zu trennen. Dies wird durch die Verwendung abstrakter Datentypen erreicht. In Anlehnung an die PETSc-Notation wurden diese mit `nutVec`, `nutMat` und `nutIS` bezeichnet, s. a. Tab. 2.3. Somit stehen Handles zu Vektoren, Matrizen und Indexmengen (`Index Sets`) zur Verfügung, ohne die tatsächlich dahinter stehenden Größen unmittelbar zur Verfügung haben zu müssen. Ermöglicht wird dies über das Arbeiten mit Adressverweisen.

Tab. 2.3 Abstraktionsebenen in der ATHLET-`NuT`-PETSc-Konstellation

Ort	Hauptdatenstrukturen
ATHLET	intrinsische Datentypen
`NuT`-(COM)	intrinsische Datentypen
`NuT`-(abstract)	`nutVec`, `nutMat`, `nutIS`, intrinsische Datentypen
`NuT`-(mapping)	`nutVec`, `nutMat`, `nutIS`, `Vec`, `Mat`, `IS`
PETSc	`Vec`, `Mat`, `IS`, intrinsische Datentypen

Innerhalb `NuT`-(abstract) besteht keine Kenntnis von MPI-, ATHLET- oder PETSc-Sepzifika. Eine Aufgabe wird hier in eine oder mehrere abstrakte `NuT`-Funktionen aufgeteilt, die auf den abstrakten Datentypen arbeiten. Solche Unterfunktionen bedienen sich entweder selbst abstrakter Funktionen oder übernehmen – als Teil des Abschnittes `NuT`-(mapping) – die Abbildung der abstrakten Anforderung auf eine oder mehrere von PETSc zur Verfügung gestellte Prozeduren. Rückgabewerte sind entweder wieder in den abstrakten Datentypen gehalten, um in `NuT`-(abstract) mit diesen weiterarbeiten zu können, oder liefern Daten intrinsischen Typs, weil genau solche gefordert wurden.

Diese logische und programmiertechnische Teilung innerhalb eines `NuT`-Worker-Prozesses kann als weiterer Aspekt der Flexibilität des Codes gesehen werden. Lediglich `NuT`-(mapping) muss angepasst werden, wenn sich Änderungen in den Signaturen von PETSc-Routinen ergeben oder man andere numerische Bibliotheken zum Einsatz bringen möchte.

Die verschiedenen Entitätentypen können bei weiteren Bedürfnissen seitens der Anwendung leicht erweitert werden, wenn bereits die entsprechenden Unterfunktionen in

NuT-(abstract) existieren. Ansonsten muss zusätzlicher Code in NuT-(mapping) bereitgestellt werden. Zurzeit umfassen die einzelnen Entitätentypen folgende Funktionalitäten:

- *controller*
  - Abhandeln der allgemeinen MPI-Kommunikation zur Initiierung einer Aufgabenbearbeitung seitens der NuT-Workers sowie Abschluss der MPI-Kommunikation bei Beendigung der NuT-Worker-Prozesse.
  - Bereitstellung des MPI-Barriere-Mechanismus: Schickt ATHLET eine Aufgabe an die NuT-Workers ist dies grundsätzlich ein nicht blockierender Vorgang auf ATHLET-Seite. Um zu gewährleisten, dass der Inhalt von lokalen Datenstrukturen, welche in die MPI-Kommunikation eingehen, nicht vorzeitig überschrieben wird, kann der Barriere-Mechanismus genutzt werden. Dies ist z. B. nützlich, wenn sukzessiv Spalteninformationen zur Speicherung der Jacobimatrix an die NuT-Worker-Prozesse gesendet werden.
- *ftrix (linalg)*
  - Speicherung der Jacobimatrix sowie der Nicht-Null-Struktur hierzu zur Laufzeit unter Berücksichtigung der Dünnbesetztheit. Die Speicherung erfolgt auf Elementebene.
  - Ablage der Jacobimatrix in eine Datei im COO-Format, s. auch Bemerkung 2.6.
  - Färbung zur Nicht-Null-Struktur der Jacobimatrix sowie Generierung eines zugehörigen Seeds zur Bestimmung der Jacobimatrix über finite Diferenzen.
  - Lösen von linearen Gleichungssystemen der Form (3.4), (3.96) oder (3.134). Hierbei findet bei Bedarf auch eine heuristische Fill-In-Minimierung statt.
  - Berücksichtigung von partiellen Updates der Jacobimatrix sowohl hinsichtlich des Seedings also auch der Speicherung.

  Hintergrundinformationen zu diesen Themen sind in Unterabschnitt 2.3.1.4 sowie Unterabschnitt 2.3.1.5 zu finden.
- *ode*
  - Berechnung der Stufenwerte einer $FiterRK$-Methode sowohl für Block- als auch Diagonalstufen, s. Abschnitt 3.3 zum Konzept der Finite Iteration Runge-Kutta-Methoden. Ebenso wird die Berechnung von Stufenwerten zu einem erweiterten Block unterstützt. Hierbei sind lineare Gleichungssysteme zu lösen. Dies geschieht über eine in der ode-Entität integrierten ftrix-Entität.

– Berechnung von Daten, welche zur Schätzung des Kontraktionsverhaltens des zugrunde liegenden Newtonprozesses genutzt werden können.

– Bereitstellung von Startapproximtionen zur Bestimmung von Blockstufen

– Bestimmung einer Fehlerschätzung sowie der finalen Schrittapproxmation für einen lokalen Zeitschritt $t \to t + h$.

Siehe auch Abschnitt 3.3 sowie Abschnitt 3.6 zur Begriffsklärung und zusätzlich Unterabschnitt 2.3.1.5 für weitere Informationen.

PETSc/MUMPS/METIS

Hierbei handelt es sich um die extern entwickelten Bibliotheken, welche numerische Operationen zur Verfügung stellen. Die Steuerung erfolgt zentral über PETSc-Mechanismen. Eine Anbindung an die `NuT`-Worker-Prozesse erfolgt ausschließlich im `NuT`-(mapping)-Teil, was Wartung und Pflege vereinfacht.

2.3.1.2 Aufruf der Programme

Um die `NuT`-Architektur zu nutzen, bedarf es zwingend der passenden MPI-Laufzeitumgebung, s. Abschnitt 2.2.3. Sowohl für OpenMPI als auch Microsoft MPI wird diese kostenlos vom jeweiligen Hersteller zur Verfügung gestellt.

Ein Aufruf der Programme erfolgt nach dem Muster

```
mpiexec [prozess-binding] -n k_1 pfad_ATHLET inputfile runID -td
outputdir : -n k_2 pfad_NuT [NuT/PETSc-options]
```

Hierbei geben k_1 und k_2 die Anzahl der gewünschten MPI-Prozesse für das jeweilige ausführbare Programm an. Standardmäßig gilt $k_1 = 1$ und $k_2 \geqslant 1$. Eine ATHLET-Instanz arbeitet also mit einem oder mehr `NuT`-Prozessen zusammen. Man beachte, dass bei Verwendung der bereits vor diesem Projekt in ATHLET implementierten OpenMP-Direktiven weitere Kerne reserviert werden müssen, um die temporär zusätzlich auftauchenden OpenMP-Prozesse abdecken zu können. In diesem Fall bietet es sich an, von der MPI-Standardeinstellung `-bind-to none` für das optionale `prozess-binding` zu der Angabe eines Rankfiles über zu gehen, e. g. `-rf ./rankfile`, um die MPI-Prozesse so auf die Kerne zu verteilen, dass keine Konflikte mit den OpenMP-Prozessen entstehen, s. auch Abb. 2.3. Als sehr flexible, weitere Neuerung ergibt sich die Angabe von `NuT`- bzw. direkt PETSc-Optionen in der aufrufenden Kommandozeile. Hierbei steht die komplette Bandbreite an PETSc-Kommandozeilenbefehlen zur Verfügung. Als zusätzlicher `NuT`-Befehl ist das Kommando `-solver` definiert worden. Als zugehörige Optionen stehen zurzeit `lu`,

`mumps` sowie `hybrid` zur Auswahl. Diese Presets dienen hauptsächlich dem Komfort des Nutzers, indem sie eine Reihe von PETSc-Befehlen aggregieren. Konkret ergibt sich:

- `lu`

  Hiermit wird der klassische LU-Algorithmus mit partial pivoting in der sequentiellen PETSc-Implementierung und mit Standardeinstellungen selektiert. Dies ist somit nur möglich, wenn es genau einen `NuT`-Worker-Prozess gibt. Es bietet sich an, diesen Weg zu gehen, wenn das Problem als klein eingestuft werden kann, s. auch Kapitel 6.

- `mumps`

  Der über PETSc angesprochene direkte Löser MUMPS wird mit dieser Option selektiert. Als Algorithmus zur Fill-In-Reduzierung kommt (Par)METIS zum Einsatz. Der Parameter zum partial threshold pivoting wird wie in Bemerkung 2.10 diskutiert gewählt.

- `hybrid`

  Bei dieser Option wird zum Lösen des Gleichungssystems die Krylov-Unterraum-Methode GMRES genutzt, wobei linksseitig mit MUMPS Präkonditioniert wird. Entsprechend ist der Threshold-parameter fürs pivoting großzügig bemessen, vgl. Bemerkung 2.10. Das Startiterierte und damit das Startresiduum wird durch vorherige einmalige Anwendung des Präkonditionierers bestimmt.

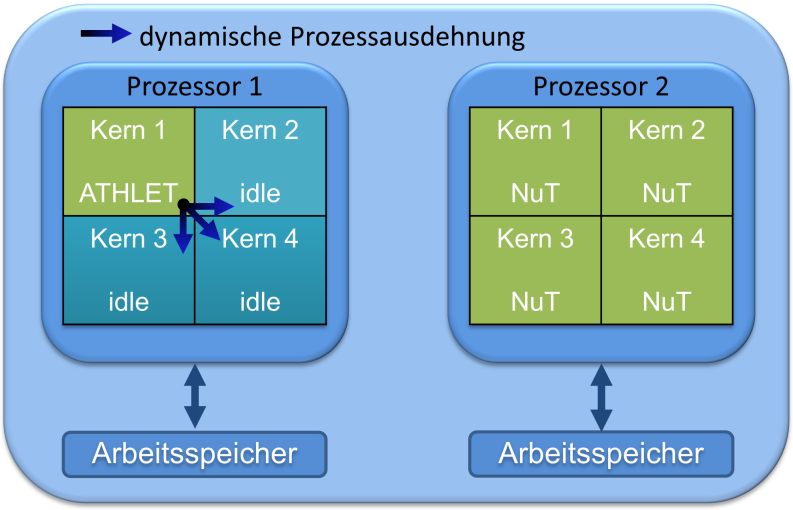


Abb. 2.3 Dynamische Prozessausdehung und hierauf angepasstes Verteilen der MPI-Prozesse

2.3.1.3 Beispiel: `linsolve` für den Legacy-Löser FEBE in der `NuT`-Architektur

Als Beispiel für das Zusammenwirken der verschiedenen Komponenten sei das das Lösen eines linearen Gleichungssystems für den im ATHLET implementierten DGL-Löser FEBE betrachtet. Es wird davon ausgegangen, dass bereits die Jacobimatrix im `NuT`-Kontext gespeichert und für numerische Rechnungen zur Verfügung steht.

Der in den folgenden Auszügen dargestellte Code stellt eine teils sehr gekürzte Fassung des Originalcodes da. Das Hauptaugenmerk liegt nicht darauf, in diesem Beispiel direkt ausführbaren Code zu präsentieren, sondern den Ablauf durch die Architektur nachvollziehbar und plausibel zu machen. Es folgen einige Anmerkungen hierzu:

- In Code 2.2 findet das dynamische Binden der im Plug-In zur Verfügung gestellten Funktionalitäten statt. An jener Stelle wäre es auch möglich, dynamisch zur Laufzeit auf ein Nichtvorhanden sein des Plug-Ins oder einer Funktion hierin zu reagieren.
- Der in ATHLET nutzbare Funktionsaufruf in den Wrapper hinein führt nach erfolgreichem Binding direkt zu einem Aufruf der Plug-In-Funktion, s. Code 2.3. Hierbei kommt das Konzept des *procedure pointers* zum Einsatz.
- Im Plug-In findet die MPI-Kommunikation mit der Gegenseite der `NuT`-Worker-Prozesse statt. Zunächst wird die allgemeine Kommunikation zur Selektion der passenden Entität/Funktionen-Kombination ausgeführt und anschließend methodenspezifische Daten an die Worker-Prozesse gesendet. In Code 2.5 und Code 2.6 ist das Gegenstück zu dieser in Code 2.4 dargestellen Vorgehensweise zu finden. Die Controller-Entität übernimmt den allgemeinen Teil und die ausgewählte ftrix-Entität handelt die benötigten Daten ab. In Zeile 11 von Code 2.6 sind alle Informationen vorhanden und die abstrakte Löserfunktion `ftrix_solve` kann aufgerufen werden.
- In Code 2.7 findet die Abarbeitung der Aufgabe mithilfe von Unterfunktionen statt, die ggf. intrinsische Datentypen berücksichtigen, generell aber auf den abstrakten `NuT`-Datentypen aus Tab. 2.3 arbeiten. Eine ftrix-Entität hält genau die Informationen zu einer Jacobimatrix, auf welche über ein `nutMat`-Objekt zugegriffen werden kann. Die abstrakten Unterfunktionen sind sehr generisch gehalten und orientieren sich an der mathematischen Anforderung. Dies macht sie unabhängig von der konkreten Implementierung in PETSc, sowohl aufgrund der gewählten Datenstrukturen als auch der benötigten Daten an sich. Ein etwaiger Wechsel zu einer anderen implementierenden Bibliothek würde sich auf dieser Ebene als sehr leicht gestalten, wie es durch Zeile 3 exemplarisch angedeutet wird.

- PETSc-Spezifika finden sich im Code 2.8 wieder, welcher zum NuT-(mapping)-Anteil der NuT-Worker-Prozesse gehört. Konkrete PETSc-Datenstrukturen werden aus den (semi)-abstrakten Größen extrahiert und den jeweiligen PETSc-Funktionen zugeführt. Die Variable A ist vom Typ nutMatP. Dieser ist bereits an PETSc gebunden, da er Variablen vom Typ Mat und KSP aggregiert. Es ist vorteilhaft, diese semi-abstrakte Datenstruktur zwischenzuschalten, da hierüber bequem notwendige Metadaten zur Matrix gespeichert werden können. Mittels KSP z. B. wird der PETSc-Löser festgelegt, der für die betrachtete Matrix benutzt werden soll. Man beachte, dass in Code 2.8 mittels PetscLogStagePush und PetscLogStagePop das Logging-Konzept von PETSc genutzt wird, um den Zeitanteil zum Lösen der linearen Gleichungssysteme zu messen. Diese Daten werden für die Diagramme in Kapitel 6 herangezogen.

Code 2.1 Anfordung in ATHLET, ein lineares Gleichungssystem zu lösen

```
use nut_wrapper

! VK: RHS Vector
! FGFEB: Diagonal Factor
! IRF: Dimension

call nut_ftrix_solve(VK, -1.0d0 / FGFEB, IRF)
```

Code 2.2 Bereitstellung der Plug-In-Funktion im Wrapper

```
! during initialization phase
subroutine bindFunction(proc_inter, plugin_name, proc_ext)
   ! ...
   ! ADT plugin dynamic function binding (+ loading)
   proc_c = plugin_try_proc(plugin_name, proc_ext)
   ! ...
   ! convert C pointer to Fortran procedure pointer
   call c_f_procpointer(proc_c, proc_inter)
   ! ...
end subroutine
```

Code 2.3 Aufruf von linsolve im Wrapper

```
subroutine nut_ftrix_solve(rhs_array, diagonalOffset, rhs_size)
   ! ...
   ! use procedure pointer stored in derived type variable <nut>
   call nut%ftrix_solve(rhs_array, diagonalOffset, rhs_size)
end subroutine
```

Code 2.4 MPI-Kommunikation im Plug-In

```
subroutine ftrix_solve(rhs_array, diagonalOffset, rhs_size)
   use nut_plugin ! MPI-definitions

   ! => MPI_Bcast containing Entity ID + Function ID
   call execute(tag_ftrix, tag_solve)

   ! function specific data sent via inter_comm
   call MPI_Bcast(rhs_size,...,inter_comm)
   call MPI_Bcast(rhs_array,rhs_size,...,inter_comm)
   call MPI_Bcast(diagonalOffset,...,inter_comm)

   ! receive result
   call MPI_Recv(rhs_array, rhs_size,...,inter_comm)
end subroutine
```

Code 2.5 NuT-(COM), allgemein

```
subroutine startWorker()
   ! ...
   do ! listen

      ! Receive entity ID and function ID via controller entity
      ! race conditions may be prevented via additonal tagged send/
         receive wrt to rank0
      call execute(modules(tag_controller), tag_receive)
      ! ...
      ! generic exec maps onto specific one
      call execute(modules(tag(1)), tag(2))
      ! ...
   end do
   ! ...
end subroutine
```

Code 2.6 NuT-(COM), methodenspezifisch

```
subroutine execute(this, func) ! ftrix-entity
   select case (func)
      case(tag_solve) ! solve linear system

        ! receive and prepare function specific data
        call MPI_Bcast(i_1,...,inter_comm)
        allocate(r_array_1(1:i_1))
        call MPI_Bcast(r_array_1, i_1,...,inter_comm)
```

```
        call MPI_Bcast(r_1,...,inter_comm,)
        ! local NuT counterpart
        call ftrix_solve(data, r_array_1, r_1)
        ! send back result
        if (nut_rank .eq. 0) then
           call MPI_Send(r_array_1, i_1,inter_comm)
        end if
        deallocate(r_array_1)

   end select
end subroutine
```

Code 2.7 Abarbeitung von linsolve in NuT-(abstract)

```
module ftrix_nut
   use nut_petsc
   ! use nut_lib_almost_as_awesome_as_petsc

   type :: ftrix_data
      ! ...
      ! Matrix J: Jacobian matrix
      nutMat :: Jac
   end type
   ! ...
   subroutine ftrix_solve(this, rhs_array, diagonalOffset)
      ! ...
      call nut_arrayToVec(rhs_array, x)
      call nut_matShift(this%Jac, diagonalOffset)
      call nut_solve(this%Jac, x) ! <- solve lin sys
      call nut_vecToArray(x, rhs_array)
      call nut_vecDestroy(x)
   end subroutine
   ! ...
end module
```

Code 2.8 Abarbeitung von linsolve in NuT-(mapping)

```
subroutine nut_solve(nut_A, nut_v)
   ! ...
   ! start profiling
   call PetscLogStagePush(stageSolve, ierr)
   ! abstract data -> PETSc related data
   call c_f_pointer(nut_A%ptr, A)
   call c_f_pointer(nut_v%ptr, v)
```

```
                                                                    8
   call nut_assembleMat_intern(A, 1)                                9
   call nut_assembleVec_intern(v)                                   10
   ! use KSP-context to solve linear system                        11
   call KSPSolve(A%solver,v%a,v%a,ierr)                             12
   ! end profiling                                                  13
   call PetscLogStagePop(ierr)                                      14
end subroutine                                                      15
                                                                    16
type :: nutMatP ! PETSc related matrix data type                    17
   Mat :: a = 0                                                     18
   KSP :: solver = 0                                                19
   nutInteger :: assembled = TAG_UNALLOCATED                        20
   nutReal :: shiftOffset = 0.0d0                                   21
   nutInteger :: handles = 1                                        22
end type                                                            23
```

Bemerkung 2.21. Auch bei absehbarer Fertigstellung der DGL-Numerik im Numerical Toolkit ist es im Sinne der Vergleichbarkeit sinnvoll, den bisherigen Löser FEBE in einer Legacy-Option erst einmal weiter zur Verfügung zu stellen. Dies rechtfertigt, an dieser Stelle als Beispiel das Lösen eines linearen Gleichungssystems zu wählen, wobei diese Anforderung seitens ATHLET generiert wird. Für über die NuT-Architektur zur Verfügung gestellte Methoden findet der komplette Ablauf der linearen Algebra direkt in den Worker-Prozessen statt.

2.3.1.4 Die rechte Seite f und die Jacobimatrix J im ATHLET-Kontext

Da das Problem (1.1) mit (semi-)impliziten Verfahren angegangen wird, also mehrere Newton(-ähnliche) Schritte pro Zeitschritt ausgeführt werden, vgl. (3.96) oder (3.134), bedarf es der Möglichkeit, Ableitungsinformationen von f zu generieren. Da solche Informationen nicht analytisch im ATHLET-Code abgelegt sind, werden diese per finite Differenzen approximativ bestimmt. Konkret geht es um die Matrixdarstellungen der partiellen Ableitungen

$$\partial f(\hat{t},\hat{y})/\partial y \qquad \text{sowie} \qquad \partial f(\hat{t},\hat{y})/\partial t$$

für geeignetes $\hat{t} \in \mathbb{R}$ und $\hat{y} \in \mathbb{R}^n$ zur kanonischen Basis im $\mathbb{R}^n$. Ist f hinreichend glatt, gilt

$$\frac{f(\hat{t}+\tau,\hat{y}) - f(\hat{t},\hat{y})}{\tau} = \frac{\partial f(\hat{t},\hat{y})}{\partial t} + \mathcal{O}(\tau) \tag{2.1}$$

für $\tau \to 0$ sowie

$$\frac{f(\hat{t}, \hat{y} + \sigma_i d) - f(\hat{t}, \hat{y})}{\sigma_i} = \frac{\partial f(\hat{t}, \hat{y})}{\partial y} \cdot d + \mathcal{O}(\sigma_i) \tag{2.2}$$

für $\sigma_i \to 0$ und beliebiges $d \in \mathbb{R}^n$. Die partielle Ableitung von f nach t steht also bereits mit nur einer weiteren f-Auswertung zur Verfügung. Wird $d = e_i$ gesetzt, wobei e_i den i-ten Einheitsvektor im $\mathbb{R}^n$ bezeichnet, lässt sich nach (2.2) somit approximativ die i-te Spalte der partiellen Ableitung von f nach y bestimmen. Läuft i von 1 bis n, so steht die gesamte Jacobimatrix näherungsweise zur Verfügung. Sei diese mit J bezeichnet. Dann gilt also

$$J = \frac{f(\hat{t}, \hat{y})}{\partial y} + \mathcal{O}(\max_i \sigma_i). \tag{2.3}$$

Dieser naive Ansatz bedarf n zusätzlicher Funktionsauswertungen und ist in der Form nur zu rechtfertigen, wenn J voll besetzt ist. Ist die Matrix dünnbesetzt, lässt sich mitunter deutlich effizienter vorgehen.

Im Folgenden wird erläutert, dass J tatsächlich einer gewissen Dünnbesetztheitsstruktur entspricht und wie hiermit im Sinne der Bestimmung und Nutzung von J im ATHLET-Kontext umgegangen wird. Dies zu klären, ist wichtig für den nächsten Abschnitt, in dem Ansatzpunkte zum Übergang ATHLET/`NuT` diskutiert werden.

Auf Modellierungsebene wird im ATHLET ein Netzwerk bestehend aus Junctions und Kontrollvolumina wie in der in Abb. 1.2 dargestellten Weise aufgebaut. Diese Abhängigkeiten lassen sich mittels einer Matrixstruktur

$$B = (b_{ij})_{i,j=1,\dots m}, \quad b_{ij} \in \{0, 1\}, \tag{2.4a}$$

veranschaulichen:

$$b_{ij} = 1 \qquad \Leftrightarrow \qquad \text{Netzwerkelement } i \text{ steht in Beziehung mit Element } j. \tag{2.4b}$$

In der ATHLET-Modellierung ist diese Relation reflexiv und symmetrisch. Es gilt also

$$b_{ii} = 1 \qquad \text{und} \qquad b_{ij} = 1 \quad \Rightarrow \quad b_{ji} = 1. \tag{2.4c}$$

Die Matrix B ist somit ebenfalls symmetrisch, i. e.,

$$B = B^T. \tag{2.4d}$$

Innerhalb ATHLETs wird die Dimension von B als `NBLO` bezeichnet, i. e., $m =$ `NBLO`. Bei weitem ist nicht jedes Kontrollvolumen mit jedem anderen über eine Junction verbunden.

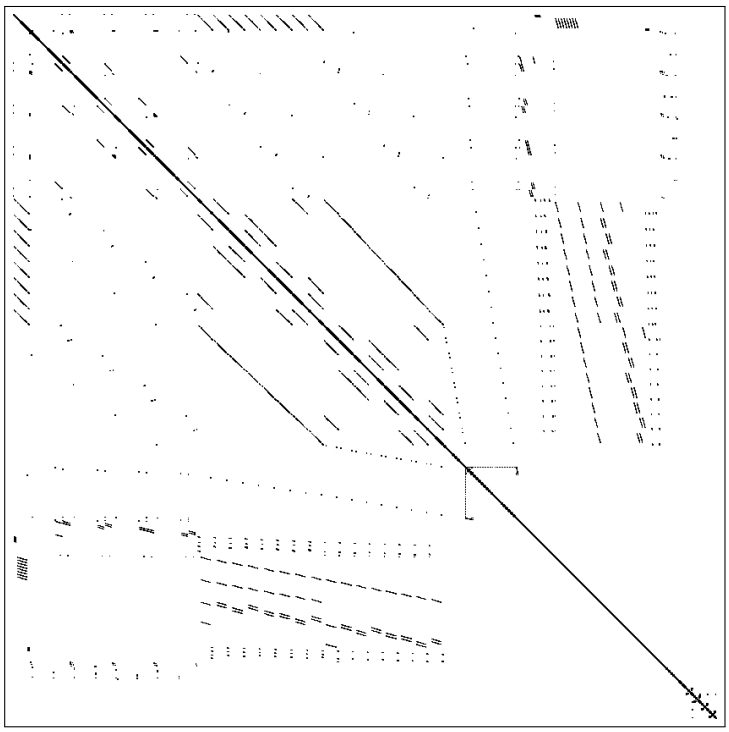


Abb. 2.4 Dünnbesetztheitsstruktur der Jacobimatrix auf Blockebene an $t = 0$ zum ATHLET-Beispiel `PWR-3D`

Die Matrix B ist somit dünnbesetzt, s. Abb. 2.4. Und da in y die Systemvariablen zu den Kontrollvolumen und Junctions zusammengefasst sind, besitzt die Jacobimatrix eine Dünnbesetztheitsstruktur, die genau der von B entspricht, wobei jedoch jeder Wert ungleich Null in B durch einen Block von Werten ersetzt wird. Blöcke, die zu verschiedenen Spalten von B gehören, sind nicht notwendig von gleicher Größe, da es von dem jeweiligen Netzwerkelement abhängt, durch wie viele Systemvariablen dieses repräsentiert wird.

In diesem Zusammenhang treten gewisse Besonderheiten und Ausnahmen auf:

- Es können im Netzwerk Elemente vorhanden sein, deren Variablen oder eine Teilmenge hiervon nicht durch Zeitintegration von (1.1) bestimmt werden, sondern stets durch algebraische Ausdrücke festgelegt sind. Diese tauchen *nicht* im Kontext der linearen Algebra auf. Da sie aber Teil des Modellierungsnetzwerkes sind, werden diese Variablen ebenfalls im ATHLET als Systemgrößen gehandhabt. Zur Unterscheidung zwischen der Menge aller Systemgroßen und der Menge der Werte, die im Ablauf zur Zeitintegration berücksichtigt werden, wird sich für diese Arbeit einer angepassten Notation bedient. Die Gesamtanzahl aller Variablen ist im ATHLET mit `IAD` oder wahlweise `IADH` bezeichnet. Der entsprechende Vektor sei durch y gekennzeichnet. ATHLET stellt nur eine rechte Seite zur Verfügung, welche auf dieser Eingabe ope-

riert. Diese sei mit `f` bezeichnet. In Abgrenzung hierzu sind die in die Integration involvierten Systemvariablen klassisch und passend zu (1.1) mit y bezeichnet, die hierfür relevante rechte Seite mit f. Die zugehörige Dimension wird im ATHLET mit `IRF` benannt. Diese entspricht dem üblichen Dimensionsbezeichner n. Formal lässt sich der Zusammenhang wie folgt darstellen:

$$\begin{aligned} &\exists \pi : \{1, \ldots, n\} \to \{1, \ldots, \texttt{IAD}\} \quad \text{mit} \quad y_{(i)} = \texttt{y}_{(\pi(i))} \\ &\text{und} \quad n = \texttt{IRF} \leqslant \texttt{IAD} = \texttt{IADH}. \end{aligned} \tag{2.5}$$

- Während der Integration können Komponenten in y ihren Status hinsichtlich des Integrationsprozesses variieren. Z. B. kann das Erreichen gewisser Grenzwerte dazu führen, dass eine Variable im nächsten Schritt über einen algebraischen Ausdruck bestimmt wird. Sollten sich die Umstände später ändern, wird die fragliche Komponente wieder per Integration bestimmt. Formal führt dies dazu, dass das System in (1.1) einer zeit- oder zustandsabhängigen Dimensionierung unterliegt. Würde man dies berücksichtigen, wären die auftretenden linearen Gleichungssysteme von entsprechend variierender Dimension. Sämtliche der auf Seite 17 dargestellten Arbeitsschritte eines direkten Lösers müssten in `ftrix` durchgeführt werden. Dies wird vermieden im ATHLET. Stattdessen werden stets lineare Gleichungssysteme der Dimension `IRF` betrachtet. Diese Systeme berechnen gewisse Newtoninkremente, s. z. B. (3.133). Wird $y_{(i)}$ zu einem Zeitpunkt algebraisch bestimmt, so wird explizit

$$\begin{aligned} &J_{(i,:)} = e_i^T \quad \text{und} \quad J_{(j,i)} = 0 \quad \text{für } i \neq j \\ &\text{sowie für die rechte Seite des Systems} \quad rhs_{(i)} = 0 \end{aligned} \tag{2.6}$$

  gesetzt. Die i-te Komponente des Newtoninkrements ergibt sich dann zu Null. Es findet keine Veränderung an $y_{(i)}$ statt und die ATHLET-interne Funktionsroutine kann sich um die algebraische Bestimmung kümmern. Die Motivation zu diesem Vorgehen wird in Paragraph *Fill-In-Minimierung* erläutert.

- Die Dimensionierung der Blöcke in J kann zeitvariant sein. Liegt in einem Netzwerkelement ein Gemischspiegel vor, so vergrößert sich die Anzahl an zugehörigen Systemvariablen. Wandert dieser Spiegel im Laufe der Zeit durch das Netzwerk, so zeigt sich dies direkt in einer Veränderung der Blockgrößen. Zwar bleibt B hierbei gleich, die Jacobimatrix ändert sich aber strukturell. Auch in diesem Fall müssten alle vier Arbeitsschritte eines direkten Lösers abgearbeitet werden. Die Strategie im ATHLET umgeht dies, indem die Invarianz von B ausgenutzt wird, s. wiederum Paragraph *Fill-In-Minimierung*.

- Kommt ein *Separator*, s. /LER 16a, § 9.7/ zum Einsatz, sind gewisse Teile des Netzwerks an zeitvariante Flüssigkeitsniveaus ausgerichtet. Es besteht die Möglichkeit, dass dieses Niveau in andere Netzwerkelemente wandert, die Verbindung hierzu aber beibehalten wird. Somit kommt es zu einer Veränderung in der Besetztheitsstruktur von B. In diesem Fall muss die komplette Matrixstruktur neu aufgebaut werden.

Als Übersicht zu den Besonderheiten in der Dimensionierung von ATHLET-Systemen sei Tab. 2.4 gegeben.

Tab. 2.4 Dimensionierungsvarianzen im ATHLET-Kontext

Art	Auftreten	Beispiel
Elementstruktur von B aus (2.4) ändert sich	selten	Separator
Blockgrößen variieren	Problem abhängig/häufig	Gemischspiegel wandert
Gleichungen werden an- oder abgeschaltet	Problem abhängig/häufig	Grenzwerte erreicht/unterschritten

Speicherung von J

Korrespondierend zu den obigen Ausführungen existieren *zwei* Datenstrukturen zur Speicherung der Jacobimatrixinformation. Zum einen wird die Matrix B in einem Format gespeichert, welches dem klassischen CSR-Format, /SAA 03/, entspricht, abgesehen von einigen zusätzlichen, abgeleiteten Informationen. Zum anderen existiert eine Speicherung der Elementstruktur – inklusive Werte –, welche sich logisch an der Blockstruktur orientiert. Eine Ähnlichkeit zu bestehenden Formaten ist nicht zu erkennen; somit wird es als grundlegend proprietär angesehen.

Bemerkung 2.22. Die just dargelegte Speicherstrategie beschreibt das Vorgehen, welches standardmäßig im ATHLET-Code durchgeführt wird. Es werden weitere Strategien angeboten, die aus legacy-technischen Gründen existieren und nicht oder nur teils die Eigenschaft der Jacobimatrix ausnutzen, dünnbesetzt zu sein. Im Rahmen dieser Arbeit wird auf jene nicht weiter eingegangen, da sie weder im Code fortlaufend gepflegt noch für heutige Probleme entsprechend angewachsener Dimension ausgelegt sind. Es wird stets davon ausgegangen, dass das Standardformat vorliegt.

Bestimmung von J über Seeding

Eine Seed-Matrix S zu einer Matrix A ist dergestalt, dass in der Multiplikation $A \cdot S$ sämtliche Elementinformationen von A extrahierbar sind. Der triviale Seed ist durch $S = I$ gegeben, wobei I die Einheitsmatrix passender Dimension bezeichnet. Ziel ist es, eine Matrix S zu finden, für welche die Anzahl der Spalten $\overline{s}$ möglichst klein ist. Ist A eine Jacobimatrix zur Funktion f an $\hat{y}$, i. e. $A = \partial f(\hat{y})/\partial y$, so korrespondiert $\overline{s}$ direkt mit der Anzahl an benötigten f-Auswertungen in einem Finite-Differenzen-Ansatz über (2.2) zur Bestimmung von J aus (2.3).

Die Störungsparameter σ werden im ATHLET-Kontext teils sehr spezifisch über die Funktion `FMADEL` gewählt. Hier wird auch eine sehr spezielle Eigenschaft der Spalten von S, welche je formal als Kandidat für d in (2.2) dienen, ausgenutzt. Um eine Idee hiervon zu geben, sei folgende kurze Herleitung betrachtet. Es gilt

$$S = (s_{ij})_{i,j}, \quad s_{ij} \in \{0,1\}, \qquad \text{sowie} \qquad Se = e$$

mit dem Eins-Vektor $e \in \mathbb{R}^{\overline{s}}$. Es wird also jede Spalte von $\partial f/\partial y$ genau einmal selektiert. Somit lässt sich Spalte s_j mit einem $l_j \in \mathbb{N}$ schreiben als

$$s_j = \sum_{k=1}^{l_j} e_{j(k)}$$

unter Zuhilfenahme einer entsprechenden injektiven Indexfunktion $j : [1, \ldots, l_j] \to [1, \ldots, n]$. Des Weiteren gilt

$$e_i^T \cdot \partial f/\partial y \cdot s_j = \begin{cases} 0 & \text{oder} \\ \partial f/\partial y_{(i,\iota)} \neq 0 & \text{für ein } \iota \in [1, \ldots, n]. \end{cases} \tag{2.7}$$

Daraus folgt mit gewissen Störungsparametern σ_k für

$$\tilde{s}_j = \sum_{k=1}^{l_j} \sigma_k e_{j(k)}$$

die Beziehung

$$f(\hat{t}, \hat{y} + \tilde{s}_j) = (I - \sum_{k=1}^{l_j} P_k) f(\hat{t}, \hat{y}) + \sum_{k=1}^{l_j} P_k f(\hat{t}, \hat{y} + \sigma_k e_{j(k)}), \tag{2.8}$$

wobei die P_k Diagonalmatrizen sind, für die

$$(P_k)_{(dd)} = \begin{cases} 1 & \text{falls } \partial f/\partial y_{(d,j(k))} \neq 0 \\ 0 & \text{sonst} \end{cases}$$

Gültigkeit hat. Da die P_k idempotent sind sowie $P_k P_j = 0$ für $j \neq k$ gilt, lässt sich der Ausdruck (2.8) äquivalent weiter umformen zu

$$\begin{aligned}\Big(\sum_{k=1}^{l_j} \sigma_k^{-1} P_k\Big)\Big(f(\hat{t}, \hat{y} + \tilde{s}_j) - f(\hat{t}, \hat{y})\Big) &= \sum_{k=1}^{l_j} \sigma_k^{-1} P_k\Big(f(\hat{t}, \hat{y} + \sigma_k e_{j(k)}) - f(\hat{t}, \hat{y})\Big) \\ &= \sum_{k=1}^{l_j} P_k\Big[\partial f / \partial y(\hat{t}, \hat{y}) e_{j(k)} + \mathcal{O}(\sigma_k)\Big].\end{aligned} \tag{2.9}$$

Die linke Seite in (2.9) ist ein mathematischer Ausdruck dessen, was in ATHLET zur Bestimmung der notwendigen Ableitungsinformationen gerechnet wird. Die rechte Seite zeigt, dass dies eine valide Operation ist. Der große Vorteil in diesem Vorgehen liegt darin, dass pro Seed-Spalte s_j nicht wie im allgemeinen Verfahren (2.2) ein einzelner Störungsparameter so gut es geht allen durch s_j gestörten Komponenten von $\hat{y}$ gerecht werden muss. Vielmehr kann auf jede dieser Komponenten individuell eingegangen werden. Im Umkehrschluss heißt dies für dieses Projekt, dass es keinen Mehrwert hat, weitere Optionen für die Wahl der σ_k zur Verfügung zu stellen; `FMADEL` liefert bereits in einer Kombination aus Empirik und klassischer Störungsanalyse sehr gute Werte.

Bemerkung 2.23. Man beachte, dass die Darstellung (2.9) indirekt in ATHLET ausgenutzt wird, dies jedoch in keinster Weise dokumentiert ist. Zwar kann dieses Projekt aus obiger schlüssiger Begründung heraus nicht mit weiteren Optionen zur Wahl der Störungsparameter σ aufwarten, jedoch ist es Verdienst dieses Projektes aus dem Bestands-Code heraus eine klar formuliert mathematische Rechtfertigung für das Vorgehen (2.9) erarbeitet zu haben.

Bemerkung 2.24. Man beachte, dass eine Aufteilung der Form (2.9) nicht von allgemeiner Gültigkeit ist. Hierbei fließt die spezielle Seeding-Information ein, die durch (2.7) charakterisiert ist.

Das Seed-Problem lässt sich graphenalgorithmisch als Färbeproblem formulieren. Die Anzahl benötigter Farben korrespondiert direkt mit der Anzahl an Spalten in der Seed-Matrix S. Folglich ist das Ziel, so wenig Farben wie möglich zu verbrauchen. Da das Problem NP-schwer ist, /GEB 05/, bedarf es Heuristiken, um das Färbeproblem anzugehen.

In ATHLET wird der Algorithmus von Courtis, Powell und Reid mit *smallest last ordering* (SLO) in der spaltenbezogenen 1-dist-Variante genutzt. Zur Erläuterung des Algorithmus s. z. B. /COL 83/. Dort wird auch bewiesen, dass dieser optimal für Bandmatrizen ist, was vermutlich damals die Motivation war, diesen einzusetzen. Zwei Besonderheiten treten in diesem Kontext auf:

- Die Bestimmung der Seed-Matrix S wird bzgl. der Matrix B ausgeführt. Dies hat den Vorteil, dass der Färbealgorithmus nur selten ausgeführt werden muss, da sich B nicht oft ändert, vgl. Tab. 2.4. In Anbetracht dessen, dass die ersten Arbeiten hierzu vor ca. 30 Jahren stattgefunden haben, ist diese Entscheidung nachvollziehbar.
- Der in ATHLET genutzte Algorithmus stellt eine leichte Modifikation des klassischen Vorgehensweise dar. Treten im *ordering*-Schritt zwei im Sinne der Ordnung gleichwertige Spalten in B auf, werden diese derart sortiert, dass die Spalte, welche weniger *Element*spalten repräsentiert, nach hinten sortiert wird. Die Blockgrößen gehen also in den Algorithmus mit ein. Eine solche Anpassung kann nicht durch Standardalgorithmen Berücksichtigung finden, was zu unterschiedlichen Färbungen führen kann, auch wenn der gleiche Basisalgorithmus genutzt wird.

Bemerkung 2.25. Im Färbealgorithmus wird nicht berücksichtigt, ob eine Systemvariable $y_{(i)}$ zurzeit algebraisch bestimmt wird oder nicht, da auf Blockebene agiert wird. Im anschließenden Eruieren der Elemente von J über (2.2) kann dies jedoch durchaus eine Rolle spielen, da die Zuweisungen in (2.6) gelten. Ergo wird $y_{(i)}$ nicht für die finite Differenz berücksichtigt. Dies kann zu einer Reduzierung der benötigten `f`-Auswertungen führen.

Bemerkung 2.26. Da der Seed S zu B erstellt wird, muss eine nachträgliche Transformation von S durchgeführt werden. Denn die Spalten von B sind auf Elementebene betrachtet verschieden breit, da sie eine unterschiedliche Anzahl an Systemvariablen kumulieren. Dies sauber im Detail zu berücksichtigen (und dies anhand des bestehenden Codes zu erkennen!), ist recht mühsam. Für das Verständnis erschwerend kommt hinzu, dass hierfür in ATHLET eine recht sperrige Datenstruktur aufgebaut wird.

Bemerkung 2.27. Neben dem klassischen einseitigen Seeding von rechts existieren auch Algorithmen, /GEB 05/,/GEB 13/, welche formal ein zweite Matrix W konstruieren, so dass sich die gesamte Jacobimatrixinformation aus den Produkten $W^T \cdot J$ und $J \cdot S$ extrahieren lässt. Dieses Vorgehen kann mitunter deutlich effizienter sein als einseitiges Seeding. Nachteil ist, dass hierfür zwingend Techniken der Automatischen Differentiation zur Verfügung stehen müssen. Diese werden weder zurzeit durch den ATHLET-Code unterstützt noch ist es Teil dieser Arbeit, diese zu etablieren, vgl. Bemerkung 2.5.

Partielle Updates von J

Im ATHLET-Code sind Kriterien und Algorithmen zu *partiellen* Updates der Jacobimatrix angegeben. Hierbei werden vier Komplexitäts-Grade des Updates unterschieden. Dies reicht von dem Update einer einzelnen Elementspalte bis hin zum Neuaufbau mehrerer

Blockzeilen und -stufen. Im Standardcode greifen partielle Updates nicht auf den Färbealgorithmus zur Optimierung der Anzahl an `f`-Auswertungen zu. Stattdessen wird (2.2) für $d = e_i$ mit passenden Indizes $i \in \{1, \ldots, n\}$ ausgeführt.

Fill-In-Minimierung

Punkt 1 in der Aufzählung auf Seite 17 nimmt sich des Problems an, die Jacobimatrix derart zu permutieren, dass in einer anschließenden LR-Zerlegung der Fill-In niedrig gehalten wird. Es lässt sich sehr viel Gewinn hinsichtlich Speicherbedarf und Rechenzeit herausholen, wenn die Permutation adäquat gewählt ist. Die Suche nach einer optimalen Permutation führt wieder auf ein NP-schweres Graphen-Problem, s. /SCH 02/, so dass abermals Heuristiken zum Einsatz kommen müssen.

Im ATHLET-Code wird hierfür der *Minimum Deficiency* Algorithmus von Tinney und Walker, /TIN 67/, genutzt. Nach /SCH 02/ liefert diese Heuristik gute Ergebnisse hinsichtlich Fill-In-Vermeidung und benötigter Operationen während der Gauß-Elimination. Nachteil sei, dass sowohl der Implementierungsaufwand beachtlich sowie die Rechenzeit im Vergleich zur viel genutzten *Minimum-Degree*-Heuristik deutlich höher sei. Diese Nachteile motivierten vermutlich die Entscheidung, dass im ATHLET-Code der Algorithmus zur Fill-In-Reduzierung auf die Blockmatrix B angewandt wird. Diese Matrix ist zum einen kleiner als J, und zum anderen ändert sie sich selten, vgl. Tab. 2.4. Die Gesamtlaufzeit der Integration wird somit möglichst wenig beeinflusst. Dennoch zeigt die Erfahrung in der Praxis, dass gerade bei größeren Problemen merklich Rechenzeit zur Ausführung benötigt wird, s. hierzu auch Abschnitt 6.2.3.

Dem Algorithmus zur Fill-In-Vermeidung liegt lediglich mittels B die (Block-)Besetzungsstruktur von J vor. Folglich stehen keine Informationen über die relativen Größen der (Element-)Einträge der Jacobimatrix zur Verfügung, so dass keine Möglichkeit exisitiert, Überlegungen zur numerischen Stabilität der anschließenden Gauß-Operationen in den Algorithmus einfließen zu lassen.

Bemerkung 2.28. Da der Fill-In hinsichtlich der Blockstruktur geschätzt wird, ist es nur konsequent, dass im ATHLET-Code die LR-Zerlegung von J auf Blockebene geschieht, de facto also eine Block-LR-Zerlegung von J vorliegt. Rückwärts- und Vorwärtssubstitution werden dann blockweise ausgeführt. Die inverse Anwendung der Diagonalblöcke wird über eine (Element-)LR-Zerlegungen mit Zeilenpivotisierung realisiert.

2.3.1.5 Etablierte Ansatzpunkte zum Übergang ATHLET/NuT

Die Aufgabe der NuT-Architektur ist es, numerische Funktionalitäten zur Verfügung zu stellen, ohne intern explizite Kenntnisse über den aufrufenden ATHLET-Kontext zu haben. Es bietet sich also nicht an, bereits Informationen über das Modellierungsnetzwerk oder ATHLET-spezifische Entscheidungen und Richtwerte des DGL-Lösers auszulagern. Des Weiteren besteht die Tatsache, dass sich `f` bzw. f aus (1.1) in seiner Eigenschaft, die Modellierungsmächtigkeit von ATHLET im Kontext des DGL-Lösers zu kapseln, im internen Aufbau und Verzweigungsrad zu weiteren Routinen sehr komplex darstellt. Zahlreiche Abhängigkeiten von globalen Modulvariablen ergänzen das Gesamtbild derart, dass f als monolithisch und nicht auslagerbar zu betrachten ist. Auf der anderen Seite ist es sehr wohl sinnvoll, sämtliche Datenstrukturen zur Jacobimatrix im NuT-Kontext zu etablieren. Denn nur dort werden sie tatsächlich für die lineare Algebra benötigt.

Lineare Algebra und Jacobimatrix

Zu diesem Thema sind folgende Ansatzpunkte etabliert worden:

- Der Übergang von Netzwerk-Datenstrukturen zur Blockmatrix B bleibt unangetastet. Ebenso bleibt die Hoheit zum Verändern der Blockgrößen sowie die Regelung zum Ab- oder Anschalten von Gleichungen beim ATHLET-Code. Aus den bestehenden Strukturen werden Informationen zu B, welche tatsächlich einem CSR-Format entsprechen, sowie Informationen zu Blockgrößen extrahiert. Diese Prozedur wird jedes Mal, wenn eines der erwähnten Ereignisse eintritt, ausgeführt.
- Mit den Informationen zur Blockmatrix und den ATHLET-Daten über abgeschaltete Gleichungen wird der benötigte Speicher zur Ablage von J festgelegt.
- Die Generierung von J per Seeding und finite Differenzen erfolgt zweiteilig.
  - Das Seeding findet in den NuT-Worker-Prozessen durch interne Aufrufe von entsprechenden PETSc-Funktionen statt. Im Gegensatz zur ATHLET-Vorgehensweise wird präferiert, den Färbealgorithmus auf der Elementstruktur von J, welche durch die Blockgrößen und den Informationen über abgeschaltete Gleichungen induziert wird, ablaufen zu lassen. Zum einen ergeben sich deutlich einfacher zu handhabende Datenstrukturen, da nicht in der Auswertung der finiten Differenzen unterschiedliche Blockgrößen berücksichtigt werden müssen. Zum anderen kann nicht durch einen allgemein nutzbaren Algorithmus dieses ATHLET-Spezifikum ausgenutzt werden, wenn auf Blockebene agiert wird. Die Zusatzinformation „Blockgröße“ ist nicht vorgesehen. Für die Färbung stehen in PETSc optimierte Algorithmen zur Verfügung. Als Standardwahl ergab sich nach einigen Tests der

Algorithmus von Courtis, Powell and Reid mit *incidence degree ordering* (IDO), s. /COL 83/ für nähere Informationen hierzu.

  - Da `f` nur direkt im ATHLET-Kontext ohne Schwierigkeiten nutzbar ist, wird nach dem Seeding das Bestimmen der Elemente von J über finite Differenzen auf ATHLET-Seite durchgeführt. Hierfür kommt die neu erstellte Routine `fmanut` zum Einsatz, welche sich als eine angepasste und sehr entschlackte Version der Standard-ATHLET-Routine `fmab3b` darstellt. Über die `NuT`-Architektur erhält `fmanut` die Seed-Information in einem CSC-Format (compressed sparse coloumn) und sendet die Ergebnisse aus den finiten Differenzen zu den `NuT`-Worker-Prozessen, welche dann die Sortierung der erhaltenen Werte in die Jacobimatrix übernehmen.

  - Partielle Updates werden von der Logik her wie im bisherigen ATHLET gehandhabt und ebenfalls in `fmanut` abgehandelt. Im Gegensatz zur Standardberechnung wird aber ebenfalls auf den Färbealgorithmus zurückgegriffen, um die Anzahl benötiger f-Auswertungen zu reduzieren. Hierfür ist allerdings ein spezielels Vorgehen nötig. PETSc-Algorithmen zur Färbung sind *nicht* auf rechteckige Matrizen ausgelegt, wie sie natürlicher Weise bei partiellen Updates auftreten. Diese Problematik wurde umgangen, indem die selektierten Spalten zur Färbung an ihrer ursprünglichen Position in der Jacobimatrix belassen wurden, während die übrigen Spalten wie in der Einheitsmatrix gewählt wurden. Nach der Färbung wird in den `NuT`-Worker-Prozessen ein Filterschritt ausgeführt, der jedweden Einfluss der Einheitsmatrix-Spalten wieder herausnimmt.

- Die Dimensionierung von J wird analog zur ATHLET-Strategie fest zu $n = \texttt{IRF}$ gewählt. Das Vorgehen (2.6) wird übernommen. Dies spart die Einführung einer weiteren Projektion wie (2.5). Es werden in J nur unwesentlich mehr Einträge generiert. Die Seed-Matrix wird hierdurch jedoch nicht größer. Auch auf die Fill-In-Vermeidung hat dies keinen negativen Einfluss. Der Lösungsprozess übernimmt die trivialen Zuweisungen, die sonst mittels der Projektion hätten durchgeführt werden müssen. In der Summe ergibt sich somit kein nennenswerter Mehraufwand.

DGL-Löser

Wie in Unterabschnitt 2.3.1.1 erläutert, soll der rein numerische Aspekt des DGL-Lösers Teil der `NuT`-Architektur sein. Hierbei liegt der Fokus zunächst auf der Ausführbarkeit einer $FiterRK$-Methode. Die Schrittweiten- sowie die Steuerung zur Jacobimatrix werden auf Seiten ATHLETs belassen.

Um einen aufgabenorientierten und den Umständen angepassten Übergang zu schaffen, wurde ausgehend von der Standard-FEBE-Datei eine adaptierte Version `febe_nut` entwickelt, welche mit den `NuT`-Worker-Prozessen über ein *Reverse-Communication-Interface* interagiert. Immer dann, wenn auf `NuT`-Seite eine Information fehlt – hauptsächlich handelt es sich hierbei um eine benötigte f-Auswertung –, merkt sich die entsprechende ode-Entität den Status der Berechnung, terminiert und gibt an die ATHLET-Seite eine passende Bedarfsmeldung weiter.

In dem Bereich von `febe_nut`, der für eine Auswertung der $FiterRK$-Methode zuständig ist, läuft so lange eine do-Schleife, bis die ode-Entität die Fertigstellung der Berechnung der Methodenwerte signalisiert. Bedarf es weiterer Informationen generiert ATHLET diese analog zur Standardversion und schickt sie dann an die ode-Entität, welche durch das Memorieren des Zustandes an der korrekten Position die Arbeit weiterführt.

Der Vorteil an diesem Vorgehen ist, dass keine blockierende Kommunikation stattfindet. Rechnet ATHLET gerade angeforderte Informationen aus, befinden sich die `NuT`-Worker-Prozesse wieder im Standardzustands des Lauschens über die controller-Entität.

Das grundsätzliche Zusammenspiel über das Reverse-Communication-Interface ist funktionstüchtig. Der Code liefert jedoch noch kein hinreichend stabiles Verhalten hinsichtlich der Berücksichtigung aller notwendigen Kontrollstrukturen zu den Themen Unstetigkeiten, Updates der Jacobimatrix, f-Auswertungen, Fehler- und Kontraktionsschätzung sowie Massenkorrektur. Das Ineinandergreifen und die algorithmische Ausprägung hiervon im ATHLET-Code sind sehr komplex und bieten vielerlei Potential zur Verbesserung. Es ergibt sich als weiteres Erschwernis, dass die Kontrollstrukturen genau auf den Einsatz des Extrapolationsansatzes basierend auf dem linear-impliziten Euler zugeschnitten sind. Wie hiermit im Rahmen eines verallgemeinerten $FiterRK$-Ansatzes umgegangen werden kann, ist in Abschnitt 3.6 diskutiert. Generell bedarf es eines äußerst vorsichtigen Vorgehens, nichts Wesentliches unberücksichtigt zu lassen. Im Rahmen des möglichen Aufwandes wurde bereits damit begonnen, Kontrollstrukturen auszulagern. Hierbei hat sich die Erkenntnis ergeben, dass sich ob der verschiedenen zu berücksichtigenden Themen eher ein exponentielles statt additives Wachstum im Aufwand zeigt. Dies war im Vorfelde so nicht abschätzbar.

2.3.1.6 Testdatensätze

Zum Austesten der sukzessiv vorgebrachten Modifikationen, welche die `NuT`-Architektur etablieren, bedarf es adäquater Beispiele. In Tab. 2.5 sind drei ausgewählte Probleme zu finden. Diese sind nach aufsteigender Komplexität sortiert. Die Probleme `PWR-3D-SYM` und

K3-NK-54H41N finden sich auch in Kapitel 6 zur Performancemessung wieder. Sample1 wird ausschließlich genutzt, um schnell zu überprüfen, ob eine Modifikation nicht grundsätzlich fehlerhaft ist. Das Beispiel ist derart klein, dass bereits im Standard-ATHLET nur wenige Sekunden Rechenzeit benötigt werden. Somit ist hier kein Performancegewinn zu erwarten. Dies ist aber in Anbetracht der Laufzeit auch nicht relevant.

Bemerkung 2.29. Da ein weiteres Beispiel der K3-Reihe in Kapitel 6 betracht wird, läuft dort das Problem K3-NK-54H41N unter dem Stichwort K3-2.

Tab. 2.5 Testprobleme, Dimensionsvariablen sind in Unterabschnitt 2.3.1.4 erläutert

	IRF	IAD	NBLO	Verzw.	
SAMPLE1	167	175	64	niedrig	†
PWR-3D-SYM	6009	6027	2058	mittel	†
K3-NK-54H41N	128673	133581	51970	hoch	/TER 08/

†: Beispiel zur ATHLET-Installation
Verzw.: Verzweigungsgrad

3 Analyse der Numerik zur Integration der DGL-Systeme und Test von neuen innovativen Lösungsverfahren

3.1 Anforderungen und Ergebnisse

Die Intention hinter diesem Arbeitspunkt ist es, den Einsatz weiterer DGL-Löser über die zu Arbeitspunkt 1 in Kapitel 2 etablierte `NuT`-Architektur zu ermöglichen. Die hierfür notwendige Erweiterung der Architektur wird thematisch ebenfalls in Kapitel 2 abgehandelt. Zur Auswahl der Verfahren ist eine Analyse des Ist-Zustandes Teil dieses Arbeitspunktes. Diese bezieht sich zum einen auf die Methode, welche dem ATHLET-Löser FEBE zugrunde liegt und durch eine Extrapolation basierend auf dem linear-impliziten Euler gegeben ist. Zum anderen geht die Analyse auf Besonderheiten ein, welche während der Zeitintegration im ATHLET-Kontext auftreten. Die neuen Methoden sind zu implementieren und an geeigneten Problemen auszutesten.

Geleistetes

Der in ATHLET genutzte Extrapolationsansatz wurde einer umfassenden theoretischen Analyse unterzogen, um Verbesserungspotentiale für neue Methoden zu erkennen. Als Resultat ergab sich, dass solche Potentiale hinsichtlich gewisser Stabilitätseigenschaften bestehen, s. Abschnitt 3.2 und insbesondere Abschnitt 3.2.4. Positiv zu bemerken ist die Tatsache, dass der Extrapolationsansatz eine W-Methode realisiert – s. (3.17) und (3.21) –, pro Stufe also nur ein lineares Gleichungssystem zu lösen ist. Nach jetzigem Stand ist keine etablierte Methode bekannt, die sowohl W-Methoden-Charakter besitzt als auch Stabilitätseigenschaften wie Steifakkuratesse sowie hohe Stufenordnung aufweist. Dies gilt ebenso für die bereits in PETSc implementierten Verfahren. Des Weiteren sind jene durch ihre Kapselung im PETSc-Code nur schwerlich auf die speziellen Umstände der ATHLET-Umgebung anpassbar, s. a. Abschnitt 3.6.

Die genannten Umstände motivierten, eine Erweiterung der Theorie zu W-Methoden zu etablieren und hierauf basierend neue Methoden zu konstruieren. Die Theorie entwickelt das Konzept der *Finite Iteration Runge-Kutta-Methoden* ($FiterRK$) und wird eingehend in Abschnitt 3.3 präsentiert. Da die asymptotische Ordnung dieser Methoden unabhängig von der Güte der verwandten Jacobimatrixapproximation ist, s. Satz 3.26, gehören solche Verfahren zur Klasse der AMF-Methoden, s. /HUN 03/ für Grundlagen.

Eingebettet in die Theorie wird auch der bisher genutzte Extrapolationsansatz betrachtet, s. Abschnitt 3.3.6. Dies ermöglicht eine Bereitstellung des gewohnten Vorgehens im Rahmen der neuen Konzepte, so dass auch in dieser Situation von dem implementierungsnahen

Teil der Theorie in Abschnitt 3.3.4 gewinnbringend Gebrauch gemacht werden kann.

Im Ganzen ist die Theorie von allgemeinem Charakter und bietet das Potential, auch an anderer Stelle vorteilhaft eingesetzt zu werden. Gleiches gilt für die zum Austesten der neuen Konzepte konstruierten Methoden, welche in Abschnitt 3.4 vorgestellt sowie in Abschnitt 3.5.2 auf verschiedene steife Probleme aus der Literatur angewandt und vergleichend dem Extrapolationsansatz gegenüber gestellt werden. Hierbei kommt eine auf regelungstechnischen Überlegungen fußende Schrittweitensteuerung zum Einsatz. Diese wird ergänzt durch eine zusätzliche Update-Strategie zur Jacobimatrix, s. Alg. 3.7 und die Verweise hierin. Der Testalgorithmus ist in Scilab, /ENT 16/, geschrieben, da hierbei das generische Überprüfen der Neuerungen im Vordergrund stand.

Die für den Scilab-Algorithmus implementierten Algorithmen 3.2-3.4 zur Berechnung von Stufenwerten wurden ebenfalls als Grundlage für die Implementation in der NuT-Architektur genutzt. Des Weiteren wurden zusätzliche Methoden entwickelt, welche dem ATHLET-Kontext angepasst sind, s. Abschnitt 3.6 und hierin Abschnitt 3.6.2 zu ATHLET-Anforderungen bzw. Methoden. Während der Konstruktion der angepassten Methoden konnte gewinnbringend das Wissen zum Einsatz gebracht werden, welches sich aus der Konstruktion der allgemeinen Methoden ergab.

Die Analyse zu den Besonderheiten von ATHLET hat sich als schwierig herausgestellt, da wesentliche Aspekte hiervon nur durch den FORTRAN-Code selbst „dokumentiert“ und auf den Extrapolationsansatz zugeschnitten sind. Hierauf und auf mögliche Vorgehensweisen wird in Abschnitt 3.6 sowie zusätzlich in Unterabschnitt 2.3.1.5 von Kapitel 2 eingegangen. Es wurde bereits eine Teilmenge der ATHLET-spezifischen Kontrollstrukturen in der NuT-Architektur berücksichtigt. Die Performancemessungen und Validierungsläufe in Kapitel 6 beziehen sich auf die in der NuT-Architektur etablierte lineare Algebra im Zusammenspiel mit FEBE. Die Entwicklungsarbeit hat gezeigt, dass sämtliche Kontrollstrukturen einzubinden sind, um einen stabilen Simulationslauf gewährleisten zu können. Es bietet sich an, dies in weiterführenden Arbeiten in Angriff zu nehmen. Positiv zu vermerken ist, dass die generischen Beispiele aus Abschnitt 3.5.2 auf eine Überlegenheit der neuen Methoden-Konzepte gegenüber der Extrapolation hindeuten. Zusätzlich sind die noch fehlenden Komponenten in der Architektur hauptsächlich auf der ATHLET-Seite angesiedelt. Die MPI-Interaktion mit den NuT-Worker-Prozessen ist im Wesentlichen fertiggestellt.

Eine Adaption der Störungsparameter zur approximativen Berechnung der Jacobimatrix über finite Differenzen war nicht nötig. Zur Erläuterungen s. die Herleitung zu (2.9) sowie Bemerkung 2.23 in Kapitel 2.

3.2 Allgemeines und Analyse des Extrapolationsansatzes

Für die theoretischen Erörterungen in diesem und den folgenden Abschnitten wird das Problem (1.1) losgelöst vom physikalischen Hintergrund betrachtet. Es sei also für geeignete offene und nichtleere Mengen D_t und D_y die Funktion $y : \mathbb{R} \supset D_t \to \mathbb{R}^n$ über die Anfangswertaufgabe

$$y'(t) = f(t, y), \quad y(t_0) = y_0 \tag{3.1}$$

definiert, wobei $y_0 \in D_y \subset \mathbb{R}^n$ gilt und $f : D_t \times D_y \to \mathbb{R}^n$ als hinreichend glatt angenommen wird. Entsprechend der Erörterungen in der Einleitung wird das System als steif betrachtet, und nach /AUS 16a, § 6.1/ ist sowohl klassische als auch oszillatorische Steifheit zu berücksichtigen.

Bemerkung 3.1. Die genannten Typen von Steifheit beziehen sich auf das Auftreten betragsmäßig großer Eigenwerte der Jacobimatrizen $\partial f/\partial y$, welche sich in der linken Hälfte der Gaußschen Zahlenebene aufhalten und zu rasch abklingenden Prozessen korrelieren bzw. *künstliche* hochfrequente und gemäßigt abklingende Lösungsanteile beschreiben. Um solche Eigenheiten der numerischen Lösung zu berücksichtigen, reicht es aus, eine *lineare* Stabilitätsanalyse durchzuführen. Nichtlineare Stabilitätskonzepte wie z. B. B-Stabilität werden weder zurzeit im ATHLET noch für die Erörterungen hier herangezogen. Des Weiteren sei angemerkt, dass Stabilitätsbetrachtungen stets von exakten Systeminformationen ausgehen. In der Praxis werden Jacobimatrizen nur näherungsweise zur Verfügung stehen, so dass bei linear-impliziten Verfahren auch Stabilitätseigenschaften nur näherungsweise erhalten bleiben. In Abschnitt 3.5.1.3 werden Strategien vorgestellt, wann es lohnenswert erscheint, eine neue Jacobimatrix zu berechnen. Im ATHLET-Code sind auch bereits verschiedene Kriterien angegeben. Besonders hervorzuheben ist hierbei die Option eines Teilupdates der Jacobimatrix. Die Steuerung hierzu bedient sich u. a. des Modellkontextes und ist insofern nicht verallgemeinerbar.

Bemerkung 3.2. Es wird nicht davon ausgegangen, dass sich die Eigenräume der für die Steifheit verantwortlichen Eigenwerte parallel zu den Koordinatenachsen ausrichten. Eine a priori Aufteilung des Differentialgleichungssystems in einen steifen und einen nichtsteifen Anteil, welche je mit angepassten Methoden behandelt werden, findet also nicht statt. Zwar bietet der in ATHLET implementierte Löser FEBE eine solche Option, in der Praxis wird jedoch stets das ganze System als steif angesehen.

Bemerkung 3.3. Innerhalb des ATHLET-Kontextes kann es zu nicht-differenzierbaren oder sogar unstetigen Änderungen in f kommen. In der Praxis ist f also höchstens abschnittsweise als glatt zu betrachten. Eine algorithmische Anpassung an dieses Verhalten

geschieht durch den Neustart des DGL-Lösers mit dem bisher als letztes berechneten Systemstatus als Anfangswert. Als weitere Konsequenz dieser Charakteristik von f wird auf die Betrachtung von Mehrschrittverfahren verzichtet, da diese auf eine Historie von Werten zurückgreifen und nach jedem Neustart des DGL-Lösers diese erst wieder mittels eines Einschrittverfahrens aufbauen müssen.

Bemerkung 3.4. Um gewisse Sachverhalte von der Notation her einfach zu halten, wird zum Teil im Folgenden auch ein autonomes System betrachtet, welches also keine explizite Zeitabhängigkeit in f aufweist. Man beachte, dass hierdurch keine Informationen verloren gehen, da vermittels $t' = 1$ das nicht-autonome System (3.1) zu einem autonomen der Größe $n + 1$ äquivalent erweitert werden kann.

Zur numerischen Berechnung einer Lösung von (3.1) wird in ATHLET ein Extrapolationsansatz gewählt. Hierbei wird ausgenutzt, dass sich der globale Fehler des zugrunde liegenden Verfahrens für beliebig glattes f in einer Potenzreihe der Schrittweite h entwickeln lässt, s. /HAI 93, Satz II.8.1/. Idee ist es, für eine gegebene Folge von Teilungsfaktoren $\{m_i\} \subset \mathbb{N}$ für jedes m_i genau so viele Schritte der Länge h/m_i mit dem Basisverfahren zu gehen und die resultierenden numerischen Approximationen $T_{i,1}$ für $y(t_0 + h)$ über einen Interpolationsansatz zu nutzen, um die Differenz von $y(t_0 + h)$ und der Potenzreihe des Fehlers mittels eines Polynoms q in h zu approximieren und schließlich $q(0)$ als finale Näherung für $y(t_0 + h)$ zurückzugeben. Der Begriff *Extrapolation* ist gerechtfertigt, da das Polynom q über den erwähnten Interpolationsansatz vermittels der Schrittweiten $h/m_i > 0$ (implizit) konstruiert wird, jedoch für $h = 0$, also außerhalb des Interpolationsintervalls, ausgewertet wird. Analog zu den Startdaten $T_{i,1}$ bezeichnet $T_{j,k}$ die Auswertung $q(0)$ desjenigen Interpolationspolynomes, welches auf den Daten $T_{j-k+1,1}, \ldots, T_{j,1}$ fußt. Die Auswertung erfolgt effizient über den Algorithmus von Aitken und Neville:

$$T_{j,k+1} = T_{j,k} + \frac{T_{j,k} - T_{j-1,k}}{(m_j/m_{j-k}) - 1}. \tag{3.2}$$

Nach /HAI 93, Satz II.9.1/ repräsentiert $T_{j,k}$ das Ergebnis einer Methode der Ordnung $p + k - 1$, wenn die Basismethode die Ordnung p besitzt. Das heißt also, dass sich pro hinzugenommene Teilschrittweite h/m_i die Ordnung der Approximation um Eins verbessert.

Bemerkung 3.5. Ein Verfahren der Ordnung p liefert ausgehend von $y(t_0)$ eine Approximation y_1 der Ordnung p für die Lösung $y(t_0 + h)$ genau dann, wenn für den *lokalen* Fehler

$$y_1 - y(t_0 + h) \in \mathcal{O}(h^{p+1}) \tag{3.3}$$

gilt. Damit lässt sich unter milden Voraussetzungen zeigen, s. z. B. /HAI 93/, dass der *globale* Fehler $y_{end} - y(t_{end})$, welcher sich auf das Ende des gesamten Integrationsintervall $[t_0, t_{end}]$ bezieht, in $\mathcal{O}(h^p)$ liegt.

Im ATHLET wird als Basismethode das linear-implizite Euler-Verfahren

$$(I - hJ)(y_1 - y_0) = hf(t_0, y_0) + h^2 \frac{\partial f_0}{\partial t} \tag{3.4}$$

mit $f_0 = f(t_0, y_0)$ und $J \approx \partial f_0 / \partial y$ genutzt[1], welches sich aus der Linearisierung des (voll) impliziten Euler-Verfahrens

$$y_1 - y_0 = hf(t_1, y_1) \tag{3.5}$$

ergibt. Taylorabgleich mit der echten Lösung liefert die Ordnung $p = 1$ für beide Methoden. Zur Veranschaulichung der Extrapolationsvorgehensweise siehe Abb. 3.1. Der dort dargestellte Fall entspricht auch der üblichen Einstellung in ATHLET. Man beachte, dass die

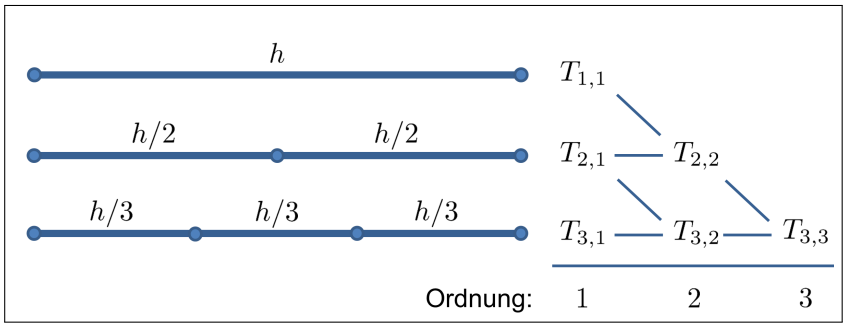


Abb. 3.1 Extrapolationsschema für den linear-impliziten Euler und für die in ATHLET verwandte Folge $\{m_i\} = \{1, 2, 3\}$. Die $T_{j,k}$ berechnen sich über (3.2).

Ordnung des linear-impliziten Eulers unabhängig von der Wahl des J ist. Hinsichtlich der asymptotischen Fehlerbetrachtung kann J also beliebig gewählt werden. Dies qualifiziert den linear-impliziten Euler, eine W-Methode darzustellen, s. (3.21). Da die Extrapolation im Wesentlichen nur eine Anwendung der Rekursion (3.2) ist, beschreibt auch diese für jede feste Extrapolationstiefe eine W-Methode. Für eine Standarddarstellung des Extrapolationsansatzes als W-Methode s. (3.25), (3.26) und (3.27) sowie Alg. 3.1.

Hinsichtlich der Stabilität ist es für den linear-impliziten Euler sehr entscheidend, dass mittels J die zurzeit dominanten Eigenwerte der echten Jacobimatrix gut widergespiegelt werden. Die voll implizite Variante ist deutlich genügsamer. Die Güte von J ist hauptsächlich hinsichtlich der Kontraktivität des Newtonprozesses, welcher zur approximativen Lösung von (3.5) eingesetzt wird, von Relevanz. Auf der anderen Seite ist der Aufwand

[1] Der Ausdruck $\partial f_0 / \partial y$ steht abkürzend für $\partial f(t, y) / \partial y_{|t=t_0, y=y_0}$.

pro Schritt in der Regel deutlich höher, da im Gegensatz zum linear-impliziten Euler, der genau *eine* Newton-Iteration pro Zeitschritt durchführt, die voll implizite Variante abhängig von der Nichtlinearität von f *und* den Genauigkeitsanforderungen des Nutzers eine vorher nicht näher bestimmte Anzahl von Newtoniterationen benötigt. Letztendlich kommt es in der Praxis darauf an, wie kostspielig sich eine Neuberechnung oder ein Update der Jacobimatrix darstellt im Gegensatz zum Lösen weiterer Gleichungssysteme. Dies ist problemabhängig. Durch gewisse interne Kopplungen gestaltet sich aber der Einsatz voll impliziter Methoden im ATHLET-Kontext generell als schwierig, s. Abschnitt 3.6

Bemerkung 3.6. Eine sinnvolle Teilungsfaktorfolge ergibt sich durch die harmonische Folge $\{m_i\} = \{1, 2, 3, 4, \dots\}$, da hierdurch die Anzahl an notwendigen Funktionsauswertungen nur sehr langsam ansteigt. Die in /HOF 81/ propagierte Bulirsch-Folge $\{1, 2, 3, 4, 6, 8, 12, 16, 24, 32, \dots\}$ wächst deutlich schneller und ist praktisch nur von Vorteil, wenn reine Quadratur anwendbar ist, also $f(t)$ statt $f(t, y)$ vorliegt, s. die Diskussion in /HAI 93, §II.9/. Im ATHLET ergibt sich jedoch praktisch kein Unterschied, da zurzeit nur bis $m_i = 3$ ausgewertet wird.

Zur Schrittweitensteuerung bedarf es einer Schätzung des lokalen Fehlers (3.3), welche ins Verhältnis zu den vom Benutzer vorgegebenen Toleranzanforderungen gestellt wird. Im ATHLET wird sich nach /AUS 16a, §6.2.1/ standardmäßig der Differenz

$$err_{est}^{loc} = T_{2,2} - T_{3,3} \tag{3.6}$$

bedient, um den Fehler von $T_{2,2}$ zu schätzen. Entsprechend der Ordnung von $T_{2,2}$ liegt der echte Fehler in $\mathcal{O}(h^{2+1})$. Und da $T_{3,3}$ eine um Eins höhere Ordnung besitzt als $T_{2,2}$ gilt dies auch für die Schätzung err_{est}^{loc}. Man beachte, dass das Fehler-Konzept auf der Asymptotik $h \to 0$ beruht. Steifheit im System kann zu einer Verzerrung der Verhältnisse führen, s. die Diskussion in Abschnitt 3.2.2 zu der Testgleichung von Prothero & Robinson und der Transportgleichung.

Bemerkung 3.7. Nach entsprechender Inspektion des Codes zur ATHLET-Integrationsmethode FEBE ergibt sich der Eindruck, dass statt auf $T_{2,2}$ auf $T_{3,2}$ zur Fehlerschätzung zurückgegriffen wird. Eine Dokumentation im Code hierzu fehlt. $T_{3,2}$ ist mit einer etwas besseren Fehlerkonstanten ausgestattet, was die Fehlerschätzung ein wenig sensitiver macht. Im weiteren Verlauf des Dokumentes wird bei Bedarf auf beide Extrapolationswerte eingegangen. Zum Thema *Fehlerkonstanten* s. Abschnitt 3.2.3.

3.2.1 Darstellungen im Butcher-Tableau

Als Vorbereitung zu linearen Stabilitätsuntersuchungen ist es sinnvoll, die vorgestellten Euler-Methoden (3.4) sowie (3.5) und die darauf basierende Extrapolation in die Notation von Butcher-Tableaus zu bringen. Hierfür wird zunächst das Konzept der Tableaus bezüglich der allgemeinen Klassen der RK- und W-Methoden vorgestellt.

Runge-Kutta-Methoden

Angelehnt an das Konzept und die Notation von Quadraturformeln lässt sich eine s-stufige Runge-Kutta-Methode (RK-Methode) schreiben als

$$k_i = hf\Big(t_0 + c_i h, y_0 + \sum_{j=1}^{s} a_{ij} k_j\Big), \quad i = 1, \dots, s, \qquad y_1 = y_0 + \sum_{j=1}^{s} b_j k_j \tag{3.7}$$

wobei die Koeffizienten $c := (c_1, \dots, c_s)^T \in [0,1]$, $\mathcal{A} := (a_{ij})_{i,j=1,\dots,s} \in \mathbb{R}^{s\times s}$ sowie $b := (b_1, \dots, b_s)^T \in \mathbb{R}^s$ eindeutig eine konkrete Methode beschreiben. Jeder Stufe ist nicht nur eine „Steigung“ k_i zugeordnet, sondern auch ein Stufenwert Y_i über

$$Y_i = y_0 + \sum_{j=1}^{s} a_{ij} k_j \tag{3.8}$$

als Approximation zu $y(t_0 + c_i h)$. Nach Konstruktion gilt offenbar

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_s \end{pmatrix} - \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \otimes y_0 = (\mathcal{A} \otimes I) \begin{pmatrix} k_1 \\ \vdots \\ k_s \end{pmatrix}. \tag{3.9}$$

Dieser Zusammenhang wird in Abschnitt 3.3.4 ausgenutzt, um effizient die für den Fortschritt der Integration benötigten Größen bereitzustellen. Als allgemeines Butcher-Tableau ergibt sich aus (3.7)

$$\begin{array}{c|c} c & \mathcal{A} \\ \hline & b^T \end{array} \tag{3.10}$$

Ist $\mathcal{A}$ eine strikte untere linke Dreiecksmatrix, so handelt es sich um eine *explizite* Runge-Kutta-Methode, andernfalls um eine *implizite* RK-Methode. Im Folgenden wird davon ausgegangen, dass

$$\mathcal{A}e = Ce \qquad \text{für} \quad C = \operatorname{diag}(c_1, \dots, c_s) \text{ und } e = (1, \dots, 1)^T \in \mathbb{R}^s \tag{3.11}$$

gilt. Dies impliziert, dass f nur an Punkten ausgewertet wird, welche die Lösung mindestens von der Ordnung Eins approximieren.

Bemerkung 3.8. Mit der Annahme (3.11) ist eine RK-Methode bereits komplett durch $\mathcal{A}$ und b beschrieben. Dennoch werden, wie es üblich ist, im weiteren Verlauf Butcher-Tableaus in ihrer vollständigen Form (3.10) angegeben.

voll impliziter Euler

Offensichtlich lässt sich der voll implizite Euler (3.5) äquivalent schreiben als

$$\begin{aligned} k_1 &= hf(t_0 + 1 \cdot h, y_0 + 1 \cdot k_1) \\ y_1 &= y_0 + 1 \cdot k_1, \end{aligned} \tag{3.12}$$

was zu einem Butcher-Tableau der folgenden überschaubaren Form führt:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{3.13}$$

W-Methoden

Sei zunächst durch f eine autonome Differentialgleichung beschrieben. Beschränkt man sich in (3.7) auf Koeffizientenmatrizen $\mathcal{A}$, welche durch eine untere linke Dreiecksmatrix dargestellt sind, so hängt k_i nicht von k_j für $i < j$ ab. Führt man nun für jede Stufe sukzessiv genau eine Newtoniteration durch, so ergibt sich

$$\Big(I - ha_{ii}J\Big)(k_i^1 - k_i^0) = -k_i^0 + hf\Big(y_0 + \sum_{j=1}^{i-1} a_{ij}k_j^1 + a_{ii}k_i^0\Big), \quad i = 1, \ldots, s, \tag{3.14}$$

mit $J \approx \partial f_0/\partial y$ und den Startnäherungen k_i^0. Werden diese Näherungen mittels einer Linearkombination der vorher berechneten Newtoniterierten k_j^1 konstruiert, i. e.

$$\gamma_{ii}k_i^0 = \sum_{j=1}^{i-1} -\gamma_{ij} \cdot k_j^1, \qquad \gamma_{ii} := a_{ii}, \tag{3.15}$$

so ergibt sich nach ein wenig Umformen und mit der Beziehung

$$a_{ij} = \alpha_{ij} + \gamma_{ij} \tag{3.16}$$

die aus der Literatur bekannte Darstellung einer s-stufigen W-Methode, s. z. B. /HAI 96, § IV.7/,

$$\begin{aligned} k_i^1 &= hf\Big(y_0 + \sum_{j=1}^{i-1} \alpha_{ij}k_j^1\Big) + hJ\sum_{j=1}^{i} \gamma_{ij}k_j^1, \quad i = 1, \ldots, s, \\ y_1 &= y_0 + \sum_{j=1}^{s} b_j k_j^1. \end{aligned} \tag{3.17a}$$

Leitet sich das autonome System aus einem nicht-autonomen über die Hinzunahme von $t' = 1$ ab, so liefert ein explizites Berücksichtigen hiervon aus obiger Relation für $i = 1, \ldots, s$ die Adaption

$$\begin{aligned} k_i^1 &= hf\Big(t_0 + \alpha_i h, y_0 + \sum_{j=1}^{i-1} \alpha_{ij}k_j^1\Big) + \gamma_i h^2 \frac{\partial f_0}{\partial t} + hJ\sum_{j=1}^{i} \gamma_{ij}k_j, \\ y_1 &= y_0 + \sum_{j=1}^{s} b_j k_j^1 \end{aligned} \tag{3.17bi}$$

mit

$$\alpha_i := \sum_{j=1}^{i-1} \alpha_{ij} \quad \text{und} \quad \gamma_i := \sum_{j=1}^{i} \gamma_{ij}. \tag{3.17bii}$$

Definierende Größen in diesem Kontext sind somit wie zuvor b sowie $A := (\alpha_{ij})_{i,j=1,\dots,s}$ und $\Gamma := (\gamma_{ij})_{i,j=1,\dots,s}$, wobei $\alpha_{ij} = 0$ für $j \geqslant i$ und $\gamma_{ij} = 0$ für $j > i$ gilt. Über (3.16) und mit der Annahme (3.11) ist der W-Methode zusätzlich eine RK-Methode mit $c = \mathcal{A}e$ zugeordnet. Wie sich im nächsten Unterabschnitt zeigt, ist es sinnvoll, diese Beziehung herauszustellen. Somit wird in dieser Arbeit eine W-Methode über das folgende erweiterte Butcher-Tableau charakterisiert:

$$\begin{array}{c|c} c & \mathcal{A} \\ \hline & b^T \end{array}, \quad A, \quad \Gamma \tag{3.18}$$

mit $\mathcal{A} = A + \Gamma, \quad c = \mathcal{A}e.$

Bemerkung 3.9. Analog zur Stufenwert-Definition (3.8) für Runge-Kutta-Methoden bietet sich auf natürliche Weise für W-Methoden die Definition

$$Y_i^1 = y_0 + \sum_{j=1}^{s} a_{ij} k_j^1 \tag{3.19}$$

an, welche die Grundidee des W-Methoden-Konzeptes, genau eine Newton-Iteration pro Stufe durchzuführen, widerspiegelt.

Linear-impliziter Euler

Vergleich von (3.4) und (3.17b) zeigt auf, dass sich der linear-implizite Euler darstellen lässt als

$$\begin{aligned} k_1^1 &= hf(t_0, y_0) + 1 \cdot h^2 \frac{\partial f_0}{\partial t} + hJ \cdot 1 \cdot k_1^1 \\ y_1 &= y_0 + 1 \cdot k_1^1. \end{aligned} \tag{3.20}$$

Dies liefert sofort das erweiterte Butcher-Tableau

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}, \quad A = \begin{pmatrix} 0 \end{pmatrix}, \quad \Gamma = \begin{pmatrix} 1 \end{pmatrix}. \tag{3.21}$$

Es ist nicht überraschend, dass der voll implizite Euler die zugeordnete RK-Methode ist, da beide Euler-Verfahren nach Definition über Linearisierung zusammenhängen, s. (3.4) und (3.5).

Extrapolation

Um die Größen $T_{i,1}$ für den Extrapolationsansatz zu berechnen, ist die Basismethode – hier der linear-implizite Euler – mit Schrittweiten der Länge h/m_i für die Folge von Teilungsfaktoren $\{m_i\}$ auf das Integrationsintervall $[t_0, t_0+h]$ anzuwenden. Jeder einzelne Teilschritt lässt sich als Stufe in einer W-Methode interpretieren. An dieser Stelle wird dies exemplarisch für $m_i = 3$ zur Berechnung von $T_{3,1}$ gezeigt. Hieraus lässt sich leicht der allgemeine Fall mittels eines Induktionsargumentes herleiten.

Um die k_1^1 der einzelnen Eulerschritte in der Notation wohl unterscheidbar zu machen, wird diesen der Index des y-Startwertes hinzugefügt. Die Ableitungsgrößen werden nicht pro Teilschritt neu ausgewertet. Dies entspricht der Denkweise, die berechneten Größen auf den Gesamtschritt $t_0 \to t_0 + h$ zu beziehen. Somit ergibt sich zur Berechnung von $T_{3,1}$ das folgende Vorgehen:

1. $t_0 \to t_0 + h/3$

$$\begin{aligned} k_{1,0}^1 &= \tfrac{1}{3}hf(t_0, y_0) + (\tfrac{1}{3})^2 h^2 \frac{\partial f_0}{\partial t} + \tfrac{1}{3}hJk_{1,0}^1 \\ y_1 &= y_0 + k_{1,0}^1, \qquad t_1 = t_0 + \tfrac{1}{3}h \end{aligned} \tag{3.22a}$$

2. $t_1 \to t_1 + h/3$

$$\begin{aligned} k_{1,1}^1 &= \tfrac{1}{3}hf(t_1, y_1) + \left(\tfrac{1}{3}\right)^2 h^2 \frac{\partial f_0}{\partial t} + \tfrac{1}{3}hJk_{1,1}^1 \\ y_2 &= y_1 + k_{1,1}^1, \qquad t_2 = t_1 + \tfrac{1}{3}h \end{aligned} \tag{3.22b}$$

3. $t_2 \to t_2 + h/3$

$$\begin{aligned} k_{1,2}^1 &= \tfrac{1}{3}hf(t_2, y_2) + \left(\tfrac{1}{3}\right)^2 h^2 \frac{\partial f_0}{\partial t} + \tfrac{1}{3}hJk_{1,2}^1 \\ y_3 &= y_2 + k_{1,2}^1, \qquad t_3 = t_2 + \tfrac{1}{3}h \\ T_{3,1} &= y_3 \end{aligned} \tag{3.22c}$$

Wird nun $k_i^1 = 3 \cdot k_{1,i-1}^1$ und analog hierzu $Y_i = y_i$ für $i = 1, 2, 3$ gesetzt und (3.22) in diesen Größen formuliert ergibt sich:

1. $t_0 + 1/3 \cdot h$

$$\begin{aligned} k_1^1 &= hf(t_0, y_0) + \left(\tfrac{1}{3}\right) h^2 \frac{\partial f_0}{\partial t} + \tfrac{1}{3}hJk_1^1 \\ Y_1 &= y_0 + \tfrac{1}{3}k_1^1, \qquad t_1 = t_0 + \tfrac{1}{3}h \end{aligned} \tag{3.23a}$$

2. $t_0 + 2/3 \cdot h$

$$\begin{aligned} k_2^1 &= hf(t_0 + \tfrac{1}{3}h, y_0 + \tfrac{1}{3}k_1^1) + \left(\tfrac{1}{3}\right) h^2 \frac{\partial f_0}{\partial t} + \tfrac{1}{3}hJk_2^1 \\ Y_2 &= y_0 + \tfrac{1}{3}k_1^1 + \tfrac{1}{3}k_2^1, \qquad t_2 = t_0 + \tfrac{2}{3}h \end{aligned} \tag{3.23b}$$

3. $t_0 + h$

$$
\begin{aligned}
k_3^1 &= \tfrac{1}{3} h f(t_0 + \tfrac{2}{3}h, y_0 + \tfrac{1}{3}k_1^1 + \tfrac{1}{3}k_2^1) + \left(\tfrac{1}{3}\right) h^2 \frac{\partial f_0}{\partial t} + \tfrac{1}{3} h J k_3^1 \\
Y_3 &= y_0 + \tfrac{1}{3}k_1^1 + \tfrac{1}{3}k_2^1 + \tfrac{1}{3}k_3^1, \qquad t_3 = t_0 + h \\
T_{3,1} &= Y_3
\end{aligned}
\tag{3.23c}
$$

Vergleich mit (3.17b) und (3.19) zeigt, dass sich die Berechnung von $T_{3,1}$ als W-Methode mit folgendem Tableau ergibt

$$
T_{3,1}: \quad \begin{array}{c|ccc} \frac{1}{3} & \frac{1}{3} & & \\ \frac{2}{3} & \frac{1}{3} & \frac{1}{3} & \\ 1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \hline & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{array}, \quad A = \begin{pmatrix} & & \\ \frac{1}{3} & & \\ \frac{1}{3} & \frac{1}{3} & \end{pmatrix}, \quad \Gamma = \begin{pmatrix} \frac{1}{3} & & \\ & \frac{1}{3} & \\ & & \frac{1}{3} \end{pmatrix}. \tag{3.24}
$$

Die zugeordnete RK-Methode ist das „gedrittelte“ voll implizite Euler-Verfahren, welches sich exakt analog zu (3.22)/(3.23) erschließt, wenn k_i^1 aus (3.17b) mit k_i aus (3.7) ersetzt wird. Nach Induktion ergibt sich allgemein für $T_{i,1}$ mit $i \in \mathbb{N}$ und zugehörigem $m_i \in \mathbb{N}$

$$
T_{i,1}: \quad \begin{array}{c|cccc} \frac{1}{m_i} & \frac{1}{m_i} & & & \\ \frac{2}{m_i} & \frac{1}{m_i} & \ddots & & \\ \vdots & \vdots & & \ddots & \\ 1 & \frac{1}{m_i} & \cdots & \cdots & \frac{1}{m_i} \\ \hline & \frac{1}{m_i} & \cdots & \cdots & \frac{1}{m_i} \end{array}, \quad A = \frac{1}{m_i} \cdot \begin{pmatrix} & & & \\ 1 & & & \\ \vdots & \ddots & & \\ 1 & \cdots & \cdots & 1 \end{pmatrix}, \quad \Gamma = \tfrac{1}{m_i} \cdot I. \tag{3.25}
$$

Zur Berechnung von $T_{j,k}$ bedarf es $T_{1,1}$ bis $T_{j,1}$. Deren Koeffizienten legen bereits direkt die Größen $\mathcal{A}$, A und Γ im erweiterten Butcher-Tableau (3.18) von $T_{j,k}$ fest. Ausschließlich auf b aus (3.18) nimmt der Algorithmus von Aitken und Neville, (3.2), Einfluss. Genau hier spiegelt sich das Extrapolationsprinzip wieder, s. Alg. 3.1. Als Beispiel und zur späteren Referenzierung seien die Tableaus zu $T_{2,2}$ und $T_{3,3}$ zur ATHLET-Folge $\{m_i\} = \{1, 2, 3\}$ in (3.26) bzw. (3.27) gegeben.

$$
T_{2,2}: \quad \begin{array}{c|ccc} 1 & 1 & & \\ \frac{1}{2} & & \frac{1}{2} & \\ 1 & & \frac{1}{2} & \frac{1}{2} \\ \hline & -1 & 1 & 1 \end{array}, \quad A = \begin{pmatrix} & & \\ & & \\ & \frac{1}{2} & \end{pmatrix}, \quad \Gamma = \begin{pmatrix} 1 & & \\ & \frac{1}{2} & \\ & & \frac{1}{2} \end{pmatrix}. \tag{3.26}
$$

$$
T_{3,3}: \quad
\begin{array}{c|cccccc}
1 & 1 & & & & & \\
\frac{1}{2} & & \frac{1}{2} & & & & \\
1 & & \frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{3} & & & & \frac{1}{3} & & \\
\frac{2}{3} & & & & \frac{1}{3} & \frac{1}{3} & \\
1 & & & & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
\hline
 & \frac{1}{2} & -2 & -2 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2}
\end{array}
\tag{3.27a}
$$

$$
A = \begin{pmatrix}
 & & & & & \\
 & \frac{1}{2} & & & & \\
 & & & & & \\
 & & & \frac{1}{3} & & \\
 & & & \frac{1}{3} & \frac{1}{3} &
\end{pmatrix}, \quad
\Gamma = \begin{pmatrix}
1 & & & & & \\
 & \frac{1}{2} & & & & \\
 & & \frac{1}{2} & & & \\
 & & & \frac{1}{3} & & \\
 & & & & \frac{1}{3} & \\
 & & & & & \frac{1}{3}
\end{pmatrix}. \tag{3.27b}
$$

Bemerkung 3.10. Man beachte, dass für $T_{i,1}$ die Jacobimatrix J mit h/m_i gewichtet wird. Jedes $T_{i,1}$ ist also mit einer anderen Iterationsmatrix $I - h/m_i J$ verknüpft. Um die linearen Gleichungssysteme in (3.17) zu lösen, bedarf es einer Zerlegung der Iterationsmatrix (oder eines zugehörigen Präkonditionierers). Da, wie bereits auf Seite 55 erwähnt, die Qualität von J im linear-impliziten Kontext entscheidend für die Stabilität des Verfahrens ist, ist anzuraten, dass pro $T_{i,1}$ neu zerlegt wird, da sich der Gewichtungsfaktor h/m_i ändert. Somit sind für $T_{k,k}$ k Zerlegungen notwendig. Die in Abschnitt 3.4 und Abschnitt 3.6.2 vorgestellten Methoden umgehen diesen Zeit- und ggf. Speicheraufwand, indem $\mathcal{A}$ so gewählt ist, dass ein Einpunktspektrum vorliegt.

Bemerkung 3.11. Wie in Abschnitt 3.6.1 skizziert bedarf es für die Bereitstellung der Rechendaten einer $FiterRK$-Methode mit einem Einpunktspektrum der Koeffizientenmatrix und zum relevanten Ordnungspaar 3(2) mehr f-Auswertungen als für den Extrapolationsansatz, insbesondere wenn Flexibilität bzgl. des Poles der Stabilitätsfunktion entsprechend Satz 3.42 gewünscht wird. Auf der anderen Seite ist nach obiger Bemerkung nur eine Zerlegung der Iterationsmatrix/des Präkonditionierers vonnöten. Tests zu den in Kapitel 6 vorgestellten Problemen haben gezeigt, dass der Mehraufwand in der Berechnung weiterer f-Evaluationen im Vergleich zu mehreren Zerlegungen kompensiert werden kann, wenn das Problem groß genug ist. Besonders hat sich dies bei den Problemen aus der `K3`-Reihe bemerkbar gemacht. Man beachte, dass selbst bei mehr Rechenzeit

pro Schritt nicht zwingend gefolgert werden kann, dass die Gesamtrechnung mehr Zeit benötigt. Denn wenn sich die Gesamtzahl an diskreten Zeitschritten durch den Einsatz einer qualitativ hochwertigeren Methode nach unten korrigiert, kann in der Summe ein Performancegewinn entstehen. Hierauf deuten auch die Ergebnisse zu den allgemeinen Problemen in Abschnitt 3.5.2 hin.

Alg. 3.1 Berechnung von b aus (3.18) für $T_{j,k}$

Data : Folge von Teilungsfaktoren $\{m_i\}_{i=1,\dots,j}$

$b(T_{i,1})$ für $i = j-k+1,\dots,j$ mit $b(T_{i,1}) = m_i^{-1}(1,\dots,1)^T \in \mathbb{R}^{m_i}$ entsprechend (3.25)

foreach $b(T_{i,1})$ **do**

  berechne $\ell = \sum_{s=j-k+1}^{i-1} m_s$ und $r = \sum_{s=i+1}^{k} m_s$;

  setze z_{dim} als Nullvektor im $\mathbb{R}^{dim}$ für $dim \in \{\ell, r\}$;

  erweitere $b(T_{i,1})$ zu $b^{i,1} := \left(z_\ell^T, b(T_{i,1}), z_r^T\right)^T$;

for $v = 1$ **to** $k-1$ **do**

  for $u = j-k+1+v$ **to** j **do**

    $b^{u,v+1} = b^{u,v} + \dfrac{b^{u,v} - b^{u-1,v}}{(m_u/m_{u-v}) - 1}$; `// Extrapolation`

setze $b = b^{j,k}$;

3.2.2 Lineare Stabilitätseigenschaften

Lineare Stabilitätsanalysen beziehen sich auf das Verhalten von numerischen Integratoren bzgl. gewisser linearer Testgleichungen/-probleme, die durch Wahl bestimmter Parameter steifes Verhalten zeigen können. Steifheit sei hier als das Phänomen charakterisiert, welches bei moderater Genauigkeitsanforderung explizite Methoden sehr kleine Schrittweite wählen lässt, obschon die Lösung ein glattes Verhalten aufweist. Lineare Stabilitätsanalyse geht davon aus, dass eine Teilmenge der Eigenwerte der Jacobinmatrix von f in (3.1) für dieses Verhalten verantwortlich zeichnet: Betragsmäßig große Eigenwerte mit negativem Realteil beschreiben exponentiell(!) abklingende Lösungskomponenten, die für den eigentlichen Verlauf der Lösung nicht entscheidend sind. Eine explizite Methode kann dies jedoch nicht erkennen und drosselt die Schrittweiten ob der raschen Veränderung.

Entsprechend der historischen Reihenfolge werden in diesem Abschnitt Stabilitätskonzepte diskutiert, die sich auf die Gleichungen von Dahlquist, (3.28), Prothero & Robinson, (3.38), sowie einer räumlichen Semidiskretisierung der Transportgleichung, (3.54),

beziehen. Jedes Testproblem hat seine Berechtigung und liefert Einsichten in das Stabilitätsverhalten von RK- und W-Methoden. Im Folgenden wird die im ATHLET verwandte Extrapolation in diesen Kontexten untersucht und basierend darauf werden Verbesserungsansätze formuliert.

Dahlquist

Das etablierteste Stabilitätskonzept hat die s. g. Testgleichung von Dahlquist zum Gegenstand.

DAHLQUIST

$$y' = \lambda y, \qquad \mathrm{Re}(\lambda) < 0, \qquad y(t_0) = y_0 \tag{3.28}$$

Die Gleichung ergibt sich aus der Linearisierung an einer Lösung $\varphi(t)$ von (3.1) und unter der Vereinfachung, dass die Jacobimatrix $J(t)$ konstant sei sowie unter der Annahme der Diagonaliserbarkeit von J. Die Gleichung (3.28) beschreibt dann die Entwicklung der *Abweichung* von der Lösung $\varphi(t)$ bei Startauslenkung y_0 in Bezug auf die Eigenvektorrichtung zu einem Eigenwert λ von J mit $\mathrm{Re}(\lambda) < 0$. Für Details zur Herleitung s. /HAI 96, § IV.2/. Da die Dahlquist-Gleichung autonom und linear in y ist, ergibt sich mit Bemerkung 3.1, dass sich eine W-Methode angewandt auf (3.28) exakt gleich zur zugeordneten RK-Methode verhält. Es kann sich also bzgl. der Dahlquist-Gleichung auf die Klasse der RK-Methoden beschränkt werden.

Bemerkung 3.12. Es mag verblüffen, dass eine solch einfache skalare Gleichung derart immense Aussagekraft über das Stabilitätsverhalten einer RK-Methode hinsichtlich der potentiell sehr komplexen vektoriellen Ausgangsgleichung (3.1) haben soll. Dies ist zum einen, wie bereits angedeutet, darin begründet, dass oftmals ein Teilspektrum der Jacobimatrizen für steifes Verhalten verantwortlich zeichnet, also eine Linearisierung des Problems eine probate Vorgehensweise darstellt. Zum anderen mag (3.28) als eine Art Prüfstein gesehen werden: Wenn bereits in diesen vereinfachten Verhältnissen eine Methode instabiles Verhalten zeigt, kann nicht erwartet werden, dass für die Ausgangsprobleme Stabilität gewährleistet sei. Es ist diese Denkweise, die z. B. in /HAI 93/,/HAI 96/,/SHA 94/,/BUT 16/,/DEU 13/ an vielerlei Stellen genutzt wird, um Instabilitäten im Ausgangsproblem zu erklären, womit die Wichtigkeit dieser Gleichung herausgestellt wird. Als großer Nachteil der Dahlquist-Gleichung erweist sich jedoch, dass sie nicht das Phänomen der Ordnungsreduktion in seiner Gänze erklären kann. Da dieses Phänomen beim Extrapolationsansatz von Relevanz ist, motiviert dies die Betrachtung weiterer Testgleichungen, s. (3.38) und (3.54).

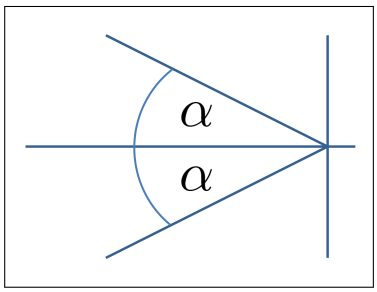


Abb. 3.2 Konzept der $A(\alpha)$-Stabilität in der Gaußschen Zahlenebene

Die Lösung von (3.28) ist

$$y(t_0 + h) = y_0 e^{\lambda h} \tag{3.29}$$

und wegen $\mathrm{Re}(\lambda) < 0$ abklingend. Die numerische Approximation einer RK-Methode zu (3.28) wird über die zugehörige Stabilitätsfunktion $R : \mathbb{C} \to \mathbb{C}$ beschrieben:

$$y_1 = R(z)y_0, \qquad z = h\lambda. \tag{3.30}$$

Eine sinnvolle Anforderung an eine RK-Methode ist offensichtlich, dass die numerische Lösung nicht ansteigt, während die exakte Lösung abklingt. Hinsichtlich R heißt dies, dass es für das Stabilitätsgebiet

$$S = \{z \in \mathbb{C} \mid |R(z)| \leqslant 1\} \tag{3.31}$$

wünschenswert wäre, die Beziehung

$$S_\alpha \subset S \tag{3.32}$$

mit

$$S_\alpha = \{z \in \mathbb{C} \mid \arg(-z) \leqslant \alpha\} \tag{3.33}$$

für $\alpha = 90\,^\circ$ zu erfüllen. Ist dies der Fall, so heißt die Methode A-stabil, gilt (3.32) nur für ein $0 \leqslant \alpha < 90\,^\circ$, so heißt die Methode $A(\alpha)$-stabil, s. Abb. 3.2 zur Veranschaulichung. Gilt zusätzlich

$$\lim_{z\to\infty} R(z) = 0, \tag{3.34}$$

so handelt es sich um eine L- bzw- $L(\alpha)$-stabile Methode.

Bemerkung 3.13. Betrachtet man die Abfolge der Berechnungen innerhalb einer expliziten RK-Methode, so ergibt sich als Stabilitätsfunktion ein Polynom in z vom Grad $\leqslant s$, s. auch (3.56). Für den expliziten Euler gilt z. B. $R(z) = 1 + z$. Somit existiert kein

α, so dass eine explizite RK-Methode $A(\alpha)$-Stabilität erfährt, denn aus $z \to \infty$ folgt unweigerlich $R(z) \to \infty$. Betrachtet man hingegen den impliziten Euler, so erhält man $R(z) = (1-z)^{-1}$ und offensichtlich wird hierdurch eine L-stabile Methode beschrieben. Da im ATHLET-System auch oszillatorische Steifheit auftritt, also betragsmäßig große Eigenwerte mit negativen Realteil dicht an der imaginären Achse liegen, stellt L-Stabilität ein hohes Gut im ATHLET-Kontext dar.

Aus der Definition der Stabilitätsfunktion in (3.30), deren konkreter Wert $R(z) = (1-z)^{-1}$ für den impliziten Euler und aus der Rekursion (3.2) ergeben sich sofort mit der Teilungsfaktorfolge $\{m_i\}$ für die Extrapolationswerte $T_{j,k}$ die zugehörigen Stabilitätsfunktionen $R_{j,k}$ über

$$R_{j,1}(z) = (1 - z/m_j)^{-m_j} \tag{3.35a}$$

$$R_{j,k+1}(z) = R_{j,k}(z) + \frac{R_{j,k}(z) - R_{j-1,k}(z)}{(m_j/m_{j-k}) - 1} \tag{3.35b}$$

Per Induktion erschließt sich

$$\lim_{z\to\infty} R_{j,k} = 0. \tag{3.36}$$

Aus der Literatur, s. z. B. /HAI 96, Tab. 9.2/, ist bekannt, dass für die harmonische Teilungsfaktorfolge $\{1, 2, 3, 4, 5, 6, \dots\}$ der extrapolierte Euler bis hohe Extrapolationsstufen sehr gute $A(\alpha)$-Stabilität mit $\alpha \in [89\,^\circ, 90\,^\circ]$ aufweist. Für einen Ausschnitt bis zur Größe $T_{3,3}$ ergibt sich

$$\begin{array}{lll} T_{1,1} & & 90\,^\circ \\ T_{2,1} \quad T_{2,2} & \overset{A(\alpha)}{\longleftrightarrow} & 90\,^\circ \quad 90\,^\circ \\ T_{3,1} \quad T_{3,2} \quad T_{3,3} & & 90\,^\circ \quad 90\,^\circ \quad 89.85\,^\circ \end{array} \tag{3.37}$$

Man beachte, dass zwar z. B. für $T_{3,3}$ der α-Kegel nicht die imaginäre Achse abdecken kann, da $\alpha < 90\,^\circ$ gilt, jedoch praktisch weite Teile der imaginären Achse im Stabilitätsgebiet liegen, s. /HAI 96, Abb. 9.5/. Ab gewissen Werten $\tilde{z} \in i\mathbb{R}$ muss dies auch der Fall sein, da (3.36) wahr ist.
Im Sinne der Dahlquist-Gleichung liefert der Extrapolationsansatz also zufriedenstellende $L(\alpha)$-Stabilität. Dennoch wäre eine vollwertige L-Stabilität wünschenswert.

Prothero & Robinson

Diese Testgleichung wurde in /PRO 74/ eingeführt und kann als Verallgemeinerung der Dahlquist-Gleichung interpretiert werden. Sie ergibt sich zu

PROTHERO & ROBINSON

$$y' = \lambda\Big(y - \varphi(t)\Big) + \varphi'(t), \qquad \mathrm{Re}(\lambda) < 0, \quad y(t_0) = \varphi(t_0) \tag{3.38}$$

mit einer nicht näher bestimmten aber hinreichend glatten Funktion $\varphi : D_t \to \mathbb{R}$.
Für $\varphi \equiv 0$ entspricht (3.38) der Dahlquist-Gleichung (3.28) mit Anfangswert Null. Als Lösung von (3.38) ergibt sich $y = \varphi$. Werden die Startwerte gestört, $y_{t_0} = \varphi(t_0) + \xi$, so verändert sich die Lösung zu

$$y(t) = \varphi(t) + \xi e^{\lambda(t-t_0)}. \tag{3.39}$$

Je steifer das System ist, desto selbstkorrigierender stellt sich also die anfangsgestörte Lösung dar. Die Herleitung von (3.38) erfolgt analog zur Herleitung der Dahlquist-Gleichung, jedoch wird die Abweichung $y - \varphi(t)$ nicht zu einer neuen Variablen zusammengefasst und spiegelt damit eher die realen Gegebenheiten wieder. Denn (3.38) ist wie das Ausgangsproblem (3.1) nicht-autonom. Somit verhält sich eine W-Methode, auch bei exakten Ableitungsinformationen bezüglich y, nicht per se genau so wie ihre zugeordnete RK-Methode für dieses Testproblem. Kritische Werte sind α_i und γ_i aus (3.17bii). Vergleich von (3.17b) und (3.7) zeigt, dass sich gleiches Verhalten einstellt, wenn

$$\alpha_i = c_i \qquad \text{und damit wegen } \mathcal{A}e = c \text{ aus (3.18)} \qquad \gamma_i = 0 \tag{3.40}$$

für $i = 1, \ldots, s$ gilt. Wie aus (3.25) leicht herzuleiten ist, ist dies für kein $T_{j,k}$, welches auf dem linear-impliziten Euler basiert der Fall. Es stellt sich die Frage, ob dies nachteilig sei. Dies wird im Folgenden diskutiert.

Als einleitendes Beispiel für die Untersuchung seien die beiden im ATHLET verwandten Extrapolations-Größen $T_{2,2}$ und $T_{3,3}$ für das nicht-steife Problem

LOTKA & VOLTERRA

$$\begin{aligned} y'_{(1)} &= y_{(1)} \cdot (p_1 - p_3 y_{(2)}) \\ y'_{(2)} &= y_{(2)} \cdot (p_2 - p_4 y_{(1)}), \end{aligned} \qquad y(t_0) = \begin{pmatrix} y_{0,1} \\ y_{0,2} \end{pmatrix}, \tag{3.41}$$

für $p_1 = 2$, $p_2 = 1$, $p_3 = 0.8$ und $p_4 = 1$ sowie $y_{0,1} = 2$ und $y_{0,2} = 1$
mit festen Schrittweiten h_j für das Zeit-Intervall $[0, 2]$ berechnet. Tabelle 3.2 zeigt die Verhältnisse aufeinanderfolgender Fehler aus Tab. 3.1, welche gut die (globalen) Ordnungen $p = 2$ und $p = 3$ von $T_{2,2}$ bzw. $T_{3,3}$ widerspiegeln, da pro Halbierung der Schrittweite der Fehler approximativ geviertelt bzw. geachtelt wird.
Hinsichtlich der Gleichung von Prothero & Robinson wird für eine fallende Folge von Schrittweiten h_j der lokale Fehler $|y_1 - \varphi(t_0 + h)|$ für die konkrete Funktion $\varphi = \cos$ und mit $\lambda = -10^5$ betrachtet. Die Resultate für $T_{2,2}$ und $T_{3,3}$ finden sich in Tab. 3.3 und Tab. 3.4. Da es sich um einen lokalen Fehler handelt, müssten in Tab. 3.4 entsprechend der Ordnungen $p = 3$ und $p = 4$ von $T_{2,2}$ bzw. $T_{3,3}$ Zahlen in der Nähe von 8 bzw. 16 stehen.

Tab. 3.1 Schrittweiten-Fehler-Relation für Problem LOTKA & VOLTERRA. Der Fehler zur numerischen Lösung $\tilde{y}$ für $t_{end} = 2$ und $atol = rtol = 10^{-10}$ ist in der Maximum-Norm gemessen. Die Zahlen sind auf eine Stelle hinter dem Komma gerundet.

	Schrittweite h			
	2^{-3}	2^{-4}	2^{-5}	2^{-6}
Err($T_{2,2}$)	$3.8 \cdot 10^{-2}$	$8.8 \cdot 10^{-3}$	$2.1 \cdot 10^{-3}$	$5.2 \cdot 10^{-4}$
Err($T_{3,3}$)	$8.1 \cdot 10^{-4}$	$1.1 \cdot 10^{-4}$	$1.5 \cdot 10^{-5}$	$1.8 \cdot 10^{-6}$

Tab. 3.2 Verhältnis aufeinanderfolgender Fehler aus Tab. 3.1 auf zwei Stellen hinter dem Komma gerundet.

	$\text{Err}_{h=2^{-3}}/\text{Err}_{h=2^{-4}}$	$\text{Err}_{h=2^{-4}}/\text{Err}_{h=2^{-5}}$	$\text{Err}_{h=2^{-5}}/\text{Err}_{h=2^{-6}}$
$T_{2,2}$	4.36	4.15	4.06
$T_{3,3}$	7.19	7.69	7.87

Die Steifheit des Problems verhindert dies jedoch. Denn nach /HAI 96, Lemma IV.15.10/ ergibt sich für den lokalen Fehler $\delta_h(t) = y_1 - \varphi(t+h)$ die Darstellung

$$\delta_h(t) = \Big[\underbrace{b^T \mathcal{A}^{-1} \operatorname{diag}(\alpha_1, \ldots, \alpha_s)^2 e}_{=:\sigma} - 1 \Big] \cdot \frac{h^2}{2} \varphi''(t) + \mathcal{O}(h^3) + \mathcal{O}(h^2/z) \tag{3.42}$$

für $h \to 0$ und $z = h\lambda \to \infty$.

In Tab. 3.5 sind die Werte von σ für verschiedene $T_{j,k}$ aufgezeigt. Dies erklärt den beobachteten Ordnungsverlust von $T_{2,2}$. Der h^2-Term in (3.42) verschwindet nicht und gibt somit die lokale Ordnung vor. Die andere Approximation zweiter Ordnung, $T_{2,3}$, wird ebenfalls vom h^2-Term dominiert, kann also nicht als Ersatz dienen. Aufgrund des h^3-Einflusses verliert zwar auch $T_{3,3}$ an lokaler Ordnung, im Sinne der Fehlerschätzung (3.6) ist es jedoch ob des Ordnungsverlustes von $T_{2,2}$ nach wie vor nutzbar. Das Gesamtverfahren verhält sich dann aber lokal wie eine Methode der Ordnung Eins. Und da die Schrittweitensteuerung am lokalen Fehler ausgerichtet ist, werden sich sehr wahrscheinlich entsprechende Schrittreduktionen einstellen. Für den globalen Fehler $\varepsilon_n := y_n - \varphi(t_n)$ zur Gleichung von Prothero & Robinson gilt nach /HAI 96, Eq. (IV.15.8) bzw. (IV.15.37)/ die Rekursion

$$\varepsilon_{n+1} = R(z)\varepsilon_n + \delta_h(t_n). \tag{3.43}$$

Da die Stabilitätsfunktionen der Extrapolations-Größen bei Unendlich verschwinden, s. (3.36), ist der globale Fehler für steife Systeme der Form (3.38) im Wesentlichen gleich

dem lokalen. Man beachte, dass durch den Faktor $R(z)$ Störungen in den Anfangswerten für einen lokalen Schritt ausgedämpft werden, ganz analog zu (3.39). Dies geschieht jedoch nicht *während* eines Schrittes in Diskrepanz zu (3.39). Es werden lediglich „Altlasten“ ausgedämpft, nicht die Störungen, die in der Approximation des lokalen Schrittes auftreten, wie die Darstellung von δ_h in (3.42) zeigt und die Experimente in Tab. 3.6 bestätigen.

Bemerkung 3.14. In der Praxis treten in der Regel nicht Eigenwerte der in Tab. 3.6 dargestellten Größe auf. Für die Betrachtungen hier wird dadurch $z \to \infty$ simuliert, welches der Aussage (3.42) zugrunde liegt, um ein qualitatives Verhalten zu beschreiben. Die Beispiele in Abschnitt 3.5.2 zeigen, dass zwar nicht von der in (3.42) dargestellten Asymptotik als Ist-Zustand ausgegangen werden kann, dass diese aber Performance-Tendenzen verständlich macht.

Modifizierter linear-impliziter Euler

Den linear-impliziten Euler auf das autonomisierte Problem, $t' = 1$, angewandt ergibt bei Verwendung exakter t-Informationen in der Jacobimatrix die Gleichung

$$\left[I - h \begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & J \end{pmatrix} \right] \begin{pmatrix} t_1 - t_0 \\ y_1 - y_0 \end{pmatrix} = h \begin{pmatrix} 1 \\ f(t_0, y_0) \end{pmatrix}. \tag{3.44}$$

Die in (3.4) gegebene Form stellt somit die Rechenschritte hinsichtlich der y-Variablen im obigen System dar. Offensichtlich entspricht das über (3.44) berechnete t_1 dem erwarteten Wert $t_0 + h$. Nach bereits einem Newton-Schritt liegt also der exakte t-Wert vor, s. (3.5). Dies ist nicht verwunderlich, da die t-Variable in (3.44) aus einem linearen Zusammenhang berechnet wird und die erste Zeile des Gleichungssystems exakte Ableitungsdaten enthält. Dies motiviert eine Modifikation des linear-impliziten Eulers, welche bereits zu Beginn mit dem exakten t-Wert arbeitet. Formal ergibt sich dann in (3.44) ein t-Inkrement von Null, und die Rechenschritte bezüglich y sind gegeben über

$$(I - hJ)(y_1 - y_0) = hf(t_0 + h, y_0). \tag{3.45}$$

Diese Vorschrift wird an dieser Stelle *modifizierter linear-impliziter Euler* genannt. Der Extrapolationsansatz lässt sich ganz analog zur unmodifzierten Variante anwenden und wird mit $modT_{j,k}$ bezeichnet. Das erweiterte Butcher-Tableau ändert sich nicht, lediglich α_i und γ_i aus (3.17bii) werden ersetzt durch c_i bzw. 0. Somit ist (3.40) erfüllt, die modifizierte Extrapolation zeigt für das Problem von Prothero & Robinson also das exakt gleiche Verhalten wie die Extrapolation basierend auf dem voll impliziten Euler. Dies ist vorteilhaft, wie es die Diskussion im folgenden Abschnitt zeigt.

Bemerkung 3.15. Man beachte, dass $modT_{j,k}$ mehr f-Auswertungen benötigt als $T_{j,k}$, da pro Teilungsfaktor m_i für den ersten Subschritt $t_0 \rightarrow t_0 + h/m_i$ die Auswertung $f(t_0 + c_i h, y_0)$ statt $f(t_0, y_0)$ nutzt. Im ATHLET-Kontext spielt dies durchaus eine Rolle, s. auch Bemerkung 3.71. Dennoch wird hier die modifizierte Variante weiter diskutiert, um das Potential im Sinne einer verbesserten Stabilität aufzuzeigen.

Stufenordnung und Steifakkuratesse

Angewandt auf das Problem von Prothero & Robinson (3.38) ergibt sich nach /HAI 96, IV.(15.7)/ für eine RK-Methode der lokale Fehler δ_h zu

$$\delta_h(t) = -zb^T(I - z\mathcal{A})^{-1}\Delta_h(t) - \Delta_{0,h}(t), \tag{3.46}$$

mit $z = h\lambda$ sowie

$$\Delta_h(t) := \varphi(t + c_i h) - Y_i \qquad \text{und} \qquad \Delta_{0,h}(t) := \varphi(t + h) - y_1,$$

wobei Y_i und y_1 aus (3.7) und (3.8) hinsichtlich der Anwendung auf

$$y' = \varphi', \qquad y(t) = \varphi(t), \tag{3.47}$$

resultieren. Bezüglich der Δ-Fehler ist also die Güte der Methode und der einzelnen Stufenwerte im Sinne einer Quadraturformel von Relevanz, und (3.46) zeigt, dass auch die Δ-Fehler der Stufen direkten Einfluss auf den Fehler δ_h haben. Dies motiviert die Definition der *Stufenordnung*: Gilt

$$\mathcal{C}(q): \quad \mathcal{A}C^{k-1} = \frac{1}{k}C^k e, \quad k = 1, \ldots, q \tag{3.48}$$

mit C und e aus (3.11), so ergibt sich für eine RK-Methode *RK* der Ordnung p nach /HAI 96, § IV.15/ die Stufenordnung über

$$StO(RK) = \min(q, p). \tag{3.49}$$

Bemerkung 3.16. Die Bedingung (3.48) ist eine der bekannten *simplifying assumptions*, die in /BUT 64/ eingeführt wurden, um Aussagen über Ordnungsbedingungen zu RK-Methoden zu formulieren. An dieser Stelle lässt sich (3.48) bereits nutzen, um auch Aussagen über die Ordnung der Stufenwerte (3.8) hinsichtlich des allgemeinen und eigentlich zu lösenden Problemes (3.1) und nicht nur bezüglich (3.47) zu treffen. Denn für hinreichend glattes f und für $i = 1, \ldots, s$ gilt nach /BUT 64, Th. 7/ die Beziehung

$$\mathcal{C}(q) \qquad \Rightarrow \qquad y(t_0 + c_i h) - Y_i \in \mathcal{O}(h^{q+1}) \text{ mit } y \text{ definiert über (3.1).} \tag{3.50}$$

Darüber hinaus erleichtert die Gültigkeit von $\mathcal{C}(q)$ immens die Konstruktion von RK-Methoden. Hiervon wird im nächsten Abschnitt Gebrauch gemacht.

Der voll implizite Euler weißt eine Stufenordnung von Eins auf, wie sich leicht aus dem Tableau (3.13) erschließen lässt. Eine hierauf basierende Extrapolation ist also ebenfalls mit einer Stufenordnung von Eins belegt. Erfreulicherweise heißt dies aber nicht, dass $\Delta_h \in \mathcal{O}(h^2)$ eine ähnliche Ordnungsreduktion im Fehler δ_h analog zum Fall des unmodifizierten linear-impliziten Eulers nach sich zieht, wie folgendes Resultat zeigt.

Proposition 3.17. *Für die Extrapolation* (3.2) *basierend auf den voll impliziten Euler* (3.13) *oder dem modifizierten linear-impliziten Euler* (3.45) *gilt für* δ_h *aus* (3.46) *die Beziehung*

$$\delta_h \in \mathcal{O}(h^2/z) \qquad \textit{für} \quad h \to 0,\ z \to \infty. \tag{3.51}$$

Beweis. Da sich der modifizierte linear-implizite Euler wie der voll implizite Euler für das Problem von Prothero & Robinson verhält, reicht es, den voll impliziten Euler zu betrachten. Für reguläres $\mathcal{A}$ gilt

$$-zb^T(I - z\mathcal{A})^{-1} = b^T\mathcal{A}^{-1} + \mathcal{O}(z^{-1}) \tag{3.52}$$

Da sich für den voll impliziten Euler $Y_s = Y_1 = y_1$ sowie $b^T\mathcal{A}^{-1} = 1$ ergibt, folgt für zugehöriges δ_h die Beziehung $\delta_h \in \mathcal{O}(\Delta_h/z)$ und mit $\Delta_h \in \mathcal{O}(h^2)$ also $\delta_h \in \mathcal{O}(h^2/z)$. Somit liefert der voll implizite Euler ein asymptotisch exaktes Ergebnis für $z \to \infty$. Die Extrapolation stellt lediglich die Anwendung der Rekursion (3.2) dar. Mittels Induktion und Standardaussagen zur Konvergenz von Folgen ergibt sich die Behauptung. □

Je steifer das Problem ist – z wächst im Betrag –, desto genauere Werte liefert also das in Proposition 3.17 betrachtete Extrapolationsverfahren. Dies schließt eine globale Sicht mit ein, da sich der globale Fehler über die Rekursion (3.43) berechnet und nach (3.36) die Stabilitätsfunktion zu den extrapolierten Größen im Unendlichen verschwindet. Dies ist eine Eigenschaft, die allgemein als *Steifakkuratesse* bezeichnet wird. Man vergleiche hierzu auch die Ergebnisse in Tab. 3.6 und Tab. 3.7.

Bemerkung 3.18. Der Begriff *Steifakkuratesse* lässt sich sinnvoll auf Methoden erweitern, für die $\lim_{z\to\infty}|R(z)| < 1$ gilt, da dann nach (3.43) ebenfalls in globaler Sicht der Fehler ausgedämpft wird, nur ggf. langsamer. Ist $\lim_{z\to\infty}|R(z)| \geqslant 1$ und trotzdem $\delta_h \to 0$ für $h \to 0$ und $z \to \infty$, dann kann von *lokaler* Steifakkuratesse gesprochen werden.

Auch wenn in Proposition 3.17 eine asymptotische Aussage im Vordergrund steht, so zeigt sich zusätzlich die Information, dass die Stufenordnung durchaus Einfluss auf den Fehler δ_h hat. Dieser verstärkt sich auf kritische Weise für das folgende Testproblem.

Tab. 3.3 Schrittweiten-Fehler-Relation für Problem PROTHERO & ROBINSON mit $\varphi = \cos$, $t_0 = 1$ und $\lambda = -10^5$. Dargestellt ist der *lokale* Fehler $|y_1 - \varphi(t_0 + h)|$ auf eine Stelle hinter dem Komma gerundet.

	Schrittweite h			
	2^{-3}	2^{-4}	2^{-5}	2^{-6}
Err($T_{2,2}$)	$1.9 \cdot 10^{-3}$	$5.0 \cdot 10^{-4}$	$1.3 \cdot 10^{-4}$	$3.3 \cdot 10^{-5}$
Err($T_{3,3}$)	$4.7 \cdot 10^{-5}$	$5.7 \cdot 10^{-6}$	$6.8 \cdot 10^{-7}$	$6.8 \cdot 10^{-8}$

Tab. 3.4 Verhältnis aufeinanderfolgender Fehler aus Tab. 3.3 auf zwei Stellen hinter dem Komma gerundet.

	$\text{Err}_{h=2^{-3}}/\text{Err}_{h=2^{-4}}$	$\text{Err}_{h=2^{-4}}/\text{Err}_{h=2^{-5}}$	$\text{Err}_{h=2^{-5}}/\text{Err}_{h=2^{-6}}$
$T_{2,2}$	3.79	3.90	3.95
$T_{3,3}$	8.25	8.46	9.91

Das Transportproblem

TRANSPORT Basierend auf

$$u_t + a u_x = 0, \qquad 0 \leqslant t \leqslant t_{end},\ \ 0 \leqslant x \leqslant 1,\ \ a > 0, \tag{3.53a}$$

und mit den Anfangs- und Randbedingungen

$$\begin{aligned} u(x,0) &:= u_0(x), \quad u_0 \in \Pi_d \\ u(0,t) &:= u_0(-at) \quad \text{(Dirichlet)} \end{aligned} \tag{3.53b}$$

ergibt sich aus einem Linienmethodenansatz $x \to (x_i)_{i=1,\dots,n}$, $x_{i+1} - x_i = \delta x$, und einer Upwind-Diskretisierung zweiter Ordnung das System

$$y'(t) = Sy + g(t), \qquad y(t_0) = \Big(u_0(x_i)\Big)_{i=1,\dots,n}, \tag{3.54a}$$

Tab. 3.5 Wert von σ aus (3.42) für verschiedene $T_{j,k}$

	$T_{2,2}$	$T_{2,3}$	$T_{k,k},\ k = 3, \dots, 7$
σ	$1/2$	$5/6$	1

Tab. 3.6 Wachsende Steifheit für Problem PROTHERO & ROBINSON mit $\varphi = \cos$, $t_0 = 1$ und $h = 2^{-5}$. Darstellung des lokalen Fehlers auf zwei Stellen hinter dem Komma gerundet.

	Eigenwert λ			
	-10^5	-10^6	-10^7	-10^8
Err($T_{2,2}$)	$1.29 \cdot 10^{-4}$	$1.29 \cdot 10^{-4}$	$1.29 \cdot 10^{-4}$	$1.29 \cdot 10^{-4}$
Err($T_{3,3}$)	$6.78 \cdot 10^{-7}$	$7.17 \cdot 10^{-7}$	$7.21 \cdot 10^{-7}$	$7.21 \cdot 10^{-7}$

Tab. 3.7 Wachsende Steifheit für Problem PROTHERO & ROBINSON mit $\varphi = \cos$, $t_0 = 1$ und $h = 2^{-5}$. Darstellung des lokalen Fehlers auf zwei Stellen hinter dem Komma gerundet.

	Eigenwert λ			
	-10^5	-10^6	-10^7	-10^8
Err($modT_{2,2}$)	$6.66 \cdot 10^{-10}$	$6.91 \cdot 10^{-11}$	$6.93 \cdot 10^{-12}$	$6.93 \cdot 10^{-13}$
Err($modT_{3,3}$)	$1.48 \cdot 10^{-11}$	$2.48 \cdot 10^{-13}$	$1.20 \cdot 10^{-14}$	$8.88 \cdot 10^{-16}$

mit

$$S = -\frac{a}{\delta x}\begin{pmatrix} 3/2 & & & & \\ -2 & \ddots & & & \\ 1/2 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & 1/2 & -2 & 3/2 \end{pmatrix} \tag{3.54b}$$

und

$$g(t) = \frac{a}{\delta x}\begin{pmatrix} 2u_0(x_1 - at) \\ -\frac{1}{2}u_0(x_1 - \delta x - at) \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.54c}$$

Bemerkung 3.19. Das hier vorgestellte Transportproblem ist eine Adaption des Beispieles in /HUN 03, § 2.1/. Während dort Quellterme und deutlich komplexere Anfangsfunktionen betrachtet werden, wurde an dieser Stelle bewusst reduziert, um polynomielle Lösungsstrukturen zu erhalten. Da das Ordnungsprinzip numerischer Verfahren auf Taylorabgleich basiert und ein Polynom gleich seiner Taylorentwicklung ist, können Sze-

narien konstruiert werden, in denen für eine Methode der Fehler ausschließlich durch die Rechengenauigkeit bestimmt werden sollte.

Bemerkung 3.20. Man beachte, dass sich das Problem (3.54a) nicht äquivalent auf einen Satz von skalaren Gleichungen der Form (3.38) bringen lässt, denn die Matrix S ist nicht diagonalisierbar. Wie sich zeigt, gelten somit auch nicht die Aussagen, welche für das Testproblem von Prothero & Robinson erarbeitet wurden.

Für die Lösung zu (3.53) gilt

$$u(x,t) = u_0(x - at) \qquad \Rightarrow \qquad u(x_{fix}, t) \in \Pi_d. \tag{3.55}$$

Das heißt, dass für $d \leqslant 2$ die Upwind-Diskretisierung zweiter Ordnung keinen Diskretisierungsfehler einbringt. Ein ODE-Verfahren der Ordnung $p \geqslant d$ sollte in diesem Fall also, abgesehen von Fehlereffekten durch endliche Rechengenauigkeit, das exakte Ergebnis reproduzieren können. Tabelle 3.9 zeigt Verhältnisse von globalen Fehlern hinsichtlich $modT_{3,3}$ und variierender Schrittweiten. Als Vergleich ist die Methode $FiterRK32b$ angegeben, welche von der Ordnung Drei mit einer Stufenordnung von Zwei ist. Details hierzu werden in Abschnitt 3.4 diskutiert. Während die Vergleichsmethode nach Tab. 3.8 numerisch exakte Ergebnisse liefert, erfährt $modT_{3,3}$ offensichtlich eine Ordnungsreduktion, s. Tab. 3.9. Werte für die urspüngliche Methode $T_{3,3}$ sind nicht angegeben, da diese sich von den Werten zur modifzierten Methode nicht wesentlich unterscheiden.

Tab. 3.8 Schrittweiten-Fehler-Relation für Problem TRANSPORT mit $u_0(x) = x^2 + x + 1$ und $a = 2$ sowie dem Diskretisierungsparameter $n = 500$. Der Fehler zur Lösung $u_0(x - at_{end})$ für $t_{end} = 2$ ist in der Maximum-Norm gemessen. Die Vergleichsmethode $FiterRK32b$ wird in Abschnitt 3.4 erläutert.

	Schrittweite h			
	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Err($modT_{3,3}$)	$2.2 \cdot 10^{-2}$	$5.6 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$	$3.5 \cdot 10^{-4}$
Err($FiterRK32b$)	$3.2 \cdot 10^{-14}$	$3.2 \cdot 10^{-14}$	$3.4 \cdot 10^{-14}$	$3.2 \cdot 10^{-14}$

Tab. 3.9 Verhältnis aufeinanderfolgender Fehler aus Tab. 3.8 für $modT_{3,3}$

$\text{Err}_{h=2^{-1}}/\text{Err}_{h=2^{-2}}$	$\text{Err}_{h=2^{-2}}/\text{Err}_{h=2^{-3}}$	$\text{Err}_{h=2^{-3}}/\text{Err}_{h=2^{-4}}$
3.66	3.99	3.99

Mit einer Erhöhung des Polynomgrades auf $d = 3$ ändert sich die Situation für den Extrapolationsansatz nicht wesentlich, sehr wohl gibt es aber eine Veränderung in der Vergleichsmethode, welche das globale Fehlerverhalten einer Methode dritter Ordnung zeigt, s. Tab. 3.11. Auf den ersten Blick deckt sich dies mit der Eigenschaft, dass $FiterRK32b$ eine Methode mit der Ordnung $p = 3$ darstellt. Die Lösung des kontinuierlichen Problems ist für festes x ein Polynom dritten Grades in t. Somit sollte sich jedoch für $FiterRK32b$ ausschließlich der fixe numerisch Fehler durch die räumliche Diskretisierung bemerkbar machen. Dieser ist unabhängig von der Zeitschrittweite h. Dies wird aber nicht von den Werten in den Tabellen 3.10 und 3.11 widergespiegelt.

Tab. 3.10 Schrittweiten-Fehler-Relation für Problem TRANSPORT mit $u_0(x) = x^3 + x^2 + x + 1$ und $a = 2$ sowie dem Diskretisierungsparameter $n = 500$. Der Fehler zur numerischen Lösung $\tilde{u}$ für $t_{end} = 2$ und $atol = rtol = 10^{-10}$ ist in der Maximum-Norm gemessen. Die Vergleichsmethode $FiterRK32b$ wird in Abschnitt 3.4 erläutert.

	Schrittweite h			
	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Err($modT_{3,3}$)	$1.5 \cdot 10^{-1}$	$4.8 \cdot 10^{-2}$	$1.4 \cdot 10^{-2}$	$3.6 \cdot 10^{-3}$
Err($FiterRK32b$)	$3.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-3}$	$5.5 \cdot 10^{-4}$	$6.8 \cdot 10^{-5}$

Tab. 3.11 Verhältnis aufeinanderfolgender Fehler aus Tab. 3.10

	$\text{Err}_{h=2^{-1}}/\text{Err}_{h=2^{-2}}$	$\text{Err}_{h=2^{-2}}/\text{Err}_{h=2^{-3}}$	$\text{Err}_{h=2^{-3}}/\text{Err}_{h=2^{-4}}$
$modT_{33}$	3.17	3.55	3.76
$FiterRK32b$	7.99	8.00	8.02

Eine Antwort auf das Verhalten beider betrachteten Methoden liefert die Analyse zur Ordnungsreduktion in /HUN 03, § II.2/. Um die wesentlichen Erkenntnisse hiervon darstellen zu können, bedarf es der Aussage folgender Proposition, s. /HAI 93, Prop. IV.3.2/,

Proposition 3.21. *Die über die Beziehung* (3.30) *definierte Stabilitätsfunktion R zu einer RK-Methode lässt sich schreiben als*

$$R(z) = \frac{\det(I - z\mathcal{A} + zeb^T)}{\det(I - z\mathcal{A})} = \frac{P(z)}{Q(z)}, \qquad P,\, Q \in \Pi_s. \tag{3.56}$$

Der globale Fehler zum Transportproblem (3.54) lässt sich rekursiv darstellen als

$$\varepsilon_{n+1} = \mathcal{R}(hS)\varepsilon_n + \delta_h(t_n) \tag{3.57}$$

mit

$$\mathcal{R}(Z) = P(Z) \cdot [Q(Z)]^{-1}.$$

und P, Q aus (3.56). $\mathcal{R}$ ist somit eine Verallgemeinerung von R im Sinne eines matrixwertigen Arguments. Mit $\delta_h(t_n)$ ist wieder der lokale Fehler bezeichnet. Es ergibt sich eine Struktur analog zu der Fehlerdarstellung (3.43) bezüglich des Testproblems von Prothero & Robinson. Der Unterschied liegt im Detail. Nach der Analyse in /HUN 03, § II.2.2/ ist der lokale Fehler durch die Stufenordnung q dominiert, i. e., $\delta_h(t_n) \in \mathcal{O}(h^{q+1})$. Mit einer kurzen Nebenrechnung ergibt sich aufgrund der Struktur von S aus (3.54b) und wegen $\lim_{z\to\infty} R(z) = 0$ für den Extrapolationsansatz und $FiterRK32b$ bezüglich der matrixwertigen Stabilitätsfunktion $\mathcal{R}$ die Beziehung

$$\mathcal{R}(hS) \to 0 \qquad \text{für festes } h \text{ und } \delta x \to 0.$$

Der globale Fehler ist also im Wesentlichen gleich dem lokalen. Und da $q = 1$ für die Euler-Extrapolation gilt und $q = 2$ für $FiterRK32b$, liefert dies eine Erklärung für die Werte in Tab. 3.11. Wird der Polynomgrad erhöht, $d > 3$, so zeigt sich kein wesentlich anderes Bild. $FiterRK32b$ entspricht dann zwar genau der Theorie, eine Methode der globalen Fehlerordnung $p = 3$ zu realisieren, man darf hierbei aber nicht vergessen, dass hierfür die lokale Fehlerordnung und somit direkt die Stufenordnung verantwortlich zeichnet.

Bemerkung 3.22. Sowohl der Extrapolationsansatz als auch $FiterRK32b$ sind als eingebettete Methoden interpretierbar. Das heißt, dass beide Verfahren Stufen verschiedener Ordnung nutzen, um den lokalen Fehler zu schätzen, und damit eine adaptive Schrittweitensteuerung ermöglichen. Obige Analyse lässt diesen Ansatz im Kontext des Transportproblems zweifelhaft erscheinen, da alle Stufen durch die Stufenordnung dominiert werden. Das ist asymptotisch korrekt. Eine genauere Betrachtung zeigt aber, dass auch Fehlerkonstanten eine gewichtige Rolle spielen, s. nächsten Abschnitt für eine Definition dieser Konstanten. Als notwendig für eine Anwendung des Prinzips der eingebetteten Methode mag somit gesetzt werden, dass die Stufe mit der höheren (klassischen) Ordnung eine Fehlerkonstante besitzt, welche mindestens um den Faktor $\frac{1}{2}$ kleiner als die Konstante der Stufe mit niedriger Ordnung ist. Folglich besteht die begründete Chance, dass das führende Bit in der Fehlerschätzung unangetastet bleibt. Die Bedingung an die Fehlerkonstante wird von beiden hier betrachteten Verfahren erfüllt, ebenso von allen anderen neu konstruierten Methoden, s. Abschnitt 3.2.3 sowie Abschnitt 3.4 und Abschnitt 3.6.2. Unabhängig hiervon findet eine Ordnungsreduktion statt, die sich an

der Stufenordnung orientiert. Bei $FiterRK32b$ fällt dies nicht gesondert ins Gewicht, da die Stufenordnung $q = 2$ ist und die zusätzliche Stufe, die zur Fehlerschätzung und zur lokalen Extrapolation genutzt wird, die (klassiche) Ordnung $p = 3$ besitzt. Eine Extrapolation basierend auf eine der Euler-Methoden wird immer auf die Stufenordnung $q = 1$ reduziert, egal welche Extrapolationstiefe genutzt wird. Dies ist ein inhärentes Problem des betrachteten Extrapolationsansatzes.

3.2.3 Fehlerkonstanten

Für ein Verfahren der Ordnung p gilt

$$e^z - R(z) \in \mathcal{O}(z^{p+1}),$$

welches sich sofort aus der Definition von R in (3.30) und des Ordnungsbegriffes ergibt (Taylorabgleich). Konkreter erhält man

$$e^z - R(z) = c_{p+1} \cdot z^{p+1} + \mathcal{O}(z^{p+2})$$

und $C_{err} := |c_{p+1}|$ heißt die *Fehlerkonstante* des Verfahrens. Für $T_{k,k}$ mit $k = 1, \ldots, 4$ ergeben sich diese wie in Tab. 3.12 gezeigt. Die Modifikationen $modT_{k,k}$ weisen die gleichen Konstanten auf, da diese bezüglich eines autonomen Problems definiert sind.

Tab. 3.12 Fehlerkonstanten für ausgewählte $T_{k,k}$

	$T_{1,1}$	$T_{2,2}$	$T_{3,3}$	$T_{4,4}$
C_{err}	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{24}$	$\frac{1}{120}$

Offensichtlich erfüllen die dargestellten $T_{k,k}$ die in Bemerkung 3.22 als sinnvoll erachtete Bedingung, dass sich die Fehlerkonstante mindestens um den Faktor $\frac{1}{2}$ bei Erhöhung der Ordnung reduziert. Die Konstanten sind verhältnismäßig groß. Die in Abschnitt 3.4 vorgestellten Methoden weisen bei gleicher Ordnung deutlich bessere, also kleinere, Fehlerkonstanten auf, s. Abschnitt 3.4.

Bemerkung 3.23. Wird statt $T_{2,2}$ der Wert von $T_{3,2}$ zur Fehlerschätzung im Tandem mit $T_{3,3}$ genutzt, so ist die in Bemerkung 3.22 aufgestellte Bedingung verletzt, da $C_{err}(T_{3,2}) = \frac{1}{18}$ gilt und somit der Faktor zu $\frac{3}{4}$ anwächst.

3.2.4 Allgemeines Résumé zum Extrapolationsansatz

Die obigen Analysen und Aussagen zusammenfassend und nutzend ergeben sich folgende Pros und Contras hinsichtlich einer Anwendung des auf dem linear-impliziten Euler basierende Extrapolationsverfahrens:

Pros

- Sämtliche Varianten des Euler-Verfahrens sind leicht zu erfassen ob der Einfachheit der Methode, welches sich auch in den entsprechend übersichtlichen Butcher-Tableaus (3.13) und (3.18) widerspiegelt. Auch der Extrapolationsansatz ist ohne viel Vorbereitung in seinen Grundzügen zugänglich.
- Extrapolation ermöglicht auf sehr einfache Weise die Konstruktion von Verfahren beliebiger Ordnung, welche rekursiv durch bereits berechnete Extrapolationsgrößen zugänglich sind, s. (3.2) sowie Abb. 3.1. Dies wird z. B. in /DEU 85/ zur adaptiven Ordnungssteuerung genutzt.
- Wegen der Rekursion (3.2) und der hiermit verbundenen Ordnungserhöhung lässt sich leicht eine eingebettete Methode zum Ordnungspaar $p + 1(p)$ konstruieren für beliebiges $p \in \mathbb{N}$, um adaptive Schrittweitensteuerung zu ermöglichen.
- Der (linear-)implizite Euler ist L-stabil, s. Bemerkung 3.13 sowie (3.37). Mit einer einfachen Modifikation, (3.45), lässt sich sogar Steifakkuratesse herstellen. Per Induktion vererbt sich dies auf die extrapolierten Werte wie Proposition 3.17 zeigt. A-Stabilität bleibt jedoch nur für geringe Extrapolationsstufen gewahrt. Höhere Extrapolationsstufen können aber zumindest mit $A(\alpha)$-Stabilität für ein $\alpha \in [89\,°, 90\,°]$ aufwarten, s. a. /HAI 96, Abb. 9.5/.
- Die zur Extrapolation benötigten Startdaten $T_{i,1}$ können im Prinzip parallel berechnet werden. Es gilt zu beachten, dass pro $T_{i,1}$ die Zerlegung der Iterationsmatrix $I - h/m_i J$ oder eines Präkonditionierers hierzu zu berechnen und zu speichern ist, s. a. Bemerkung 3.10.
- Der linear-implizite Euler bedarf nur eines Newton-Schrittes, um seine finale Approximation y_1 zu liefern. Entsprechend sind die einzelnen Stufenwerte der Extrapolation mit je nur einem Newton-Schritt ermittelbar. Steht erst einmal ein adäquates J zur Verfügung, so ist ein Zeitschritt deutlich schneller berechnet als im voll impliziten Fall.
- Der linear-implizite Euler besitzt die Ordnung $p = 1$, unabhängig von der Wahl der Matrix J. Somit wird eine W-Methode realisiert. Diese Eigenschaft vererbt sich auf den Extrapolationsansatz. Dies rechtfertigt die Nutzung von finiten Differenzen zur Approximation von $\partial f_0/\partial y$ oder auch prinzipiell Ansätze zur „Entschlackung“ der Matrix J.

Contras

- Stabilitätsaussagen zu W-Methoden beziehen sich auf den Idealfall $J = \partial f_0/\partial y$. Anhand numerischer Beispiele ist leicht zu erkennen, wie wichtig es ist, dass die

Approximation J die für Steifheit verantwortlichen Eigenwerte und -räume zu $\partial f_0/\partial y$ berücksichtigt, denn y_1 aus (3.17) hängt direkt von J ab. Im Gegensatz hierzu erweist sich eine voll implizite Methode als deutlich robuster. In der Definition (3.7) ist y_1 unabhängig von J, und in der numerischen Umsetzung mittels eines Newtonprozesses braucht J nur so gut zu sein, wie Kontraktivität (mit adäquater Kontraktionskonstante) gesichert bleibt. Mehr noch, das Kontraktionsverhalten des Newtonprozesses kann leicht geschätzt werden. Es steht also direkt ein Indikator für die Güte von J im voll impliziten Kontext zur Verfügung. Eine klassische W-Methode hat es deutlich schwieriger, da jede Stufe nur mit einem Newton-Schritt ermittelt wird, ein Hinweis auf Kontraktion aber mindestens zwei Newton-Schritte benötigt. Satz 3.26 zeigt, dass weitere Newton-Schritte durchgeführt werden können. Das wäre aber merklicher Rechenaufwand, den man hauptsächlich um der Kontraktionsinformation willen aufbrächte. Eine alternative Strategie wird in Abschnitt 3.5.1.3 diskutiert. Selbst wenn ein zuverlässiger Test zur Neuberechnung von J zur Verfügung steht, sei angemerkt, dass ob des Stabilitätsaspektes öfter ein Update von J stattfinden wird als im voll impliziten Fall. Es ist problemabhängig, ob eine frequentere Aktualisierung von J oder mitunter eine merklich größere Anzahl an zu lösenden linearen Gleichungssystemen mehr ins Gewicht hinsichtlich der Rechenzeit fällt. Hierbei ist ebenfalls zu beachten, dass aktuelle Informationen zur Jacobimatrix in der Regel den Newtonprozess beschleunigen, also weniger Iterationen vonnöten sind.

- Auch mit idealem J ist ab der Ordnung $p = 3$ keine A-Stabilität für den Extrapolationsansatz gewährleistet. Es kann also für die Dahlquist-Gleichung keine Kontraktivität der Folge der numerischen Approximationen $y_1, y_2, \ldots$ sichergestellt werden. Gleiches gilt sogar im matrixwertigen Fall, $y' = Jy$. A-Stabilität garantiert Kontraktivität der numerischen Lösung in der Euklidischen Norm, sofern die analytische Lösung kontraktiv ist, also $\mathrm{Re}\langle y, Jy\rangle \leqslant 0$ für alle $y \in \mathbb{C}^n$ und dem Euklidischen inneren Produkt gilt, s. /HAI 96, Cor. IV.11.3/. Man beachte, dass im Rahmen gewisser Vereinfachungen $y' = Jy$ Aussagekraft hinsichtlich der Abweichung von einer Lösung $\varphi(t)$ von (3.1) liefert, s. auch die Erläuterungen zur Herleitung der Dahlquistgleichung auf Seite 64 sowie /HAI 96, Gl. IV.2.(2.2')/. Im Rahmen linearer Stabilitätseigenschaften stellt somit A-Stabilität ein hohes Gut dar.

- Nach Proposition 3.17 ist zwar Steifakkuratesse für die modifizierte Extrapolation gewährleistet, dennoch zeigt sich die niedrige Stufenordnung deutlich im lokalen Fehler, s. (3.51). Für die Asymptotik $z \to \infty$ ist dies unproblematisch, was auch durch Tab. 3.7 bestätigt wird. Betrachtet man $T_{3,3}$ als Beispiel, so liegt der lokale Fehler

für nichtsteife Probleme in $\mathcal{O}(h^4)$. Dies bedeutet, dass für betragsmäßig fallendes λ in (3.38), das Problem wird also „entsteift“, die Fehlerdarstellung von $\delta_h \in \mathcal{O}(h^2/z)$ hin zu $\delta_h \in \mathcal{O}(h^4)$ wechselt. Bei milder Steifheit kann es also sein, dass man sich in dieser Grauzone befindet. Ein analytischer Zugang ist nicht trivial. Aus einem phänomenologischen Ansatz, s. Tab. 3.13, lässt sich aber bereits erkennen, dass sich die geringe Stufenordnung negativ einbringt. Man beachte, dass die modifizierte Variante bereits bessere Ergebnisse liefert, weswegen nur diese in Tab. 3.13 aufgeführt ist.

Tab. 3.13 Verhältnis aufeinanderfolgender Fehler für Problem PROTHERO & ROBINSON mit $\lambda = -10$, $\varphi = \cos$ und $t_0 = 1$. Die Vergleichsmethode $FiterRK32b$ weist eine Stufenordnung von Zwei auf und wird in Abschnitt 3.4 näher erläutert.

	$modT_{2,2}$	$modT_{3,3}$	$FiterRK32b$
$\text{Err}_{h=2^{-4}}/\text{Err}_{h=2^{-5}}$	4.86	7.90	13.83
Idealquotient nach Ordnung	8	16	16

- Das Problem TRANSPORT zeigt deutlich, dass Ordnungsreduktion durch geringe Stufenordnung nicht in jedem Fall gelindert oder verhindert werden kann. Auch wenn eine diagonalisierbare Jacobimatrixapproximation J vorliegt und somit theoretisch auf die Testgleichung von Prothero & Robinson reduziert werden könnte, besteht in der Praxis die Gefahr schlecht konditionierter Transformationsmatrizen ob etwaiger fast paralleler Eigenräume gepaart mit in Clustern konzentrierten Eigenwerten. Somit ist in einem solchen Fall eher ein Verhalten entsprechend der Untersuchungen zur Transportgleichung zu erwarten.
- Es ist zwar sehr leicht, Extrapolationen höherer Ordnung zu konstruieren, jedoch steht die Ordnung in einem quadratischen Zusammenhang zu der Anzahl benötigter Stufen. Innerhalb des ATHLET-Kontextes ist dies zu verschmerzen, da nicht über eine Extrapolationstiefe von $k = 3$ hinausgegangen wird. Dennoch bedarf es pro $T_{i,1}$ einer Zerlegung der zugehörigen Iterationsmatrix $I - h/m_i J$. Wie bei den Pro-Argumenten angesprochen kann Parallelität den Mehraufwand kompensieren. Auf der anderen Seite kann eine parallele Implementation genutzt werden, um ein einzelnes lineares Gleichungssystem zeiteffizienter zu lösen. Grundsätzlich können auch beide Konzepte kombiniert werden. Hierbei spielt eine wesentliche Rolle, auf welcher Skalierungsebene Parallelität zur Verfügung steht. Für (sehr) große Systeme, die auf einem handelsüblichen PC gerechnet werden sollen, steht nicht die Option zur Verfügung, mehrere Matrixzerlegungen im Speicher zu halten. Sehr wohl kann aber

die vorhandene Multi-Core-Architektur genutzt werden, um die lineare Algebra zu einer Matrix zu beschleunigen. Innerhalb einer Multi-Node-Umgebung mit separiertem Speicher besteht durchaus die Möglichkeit, die $T_{i,1}$ inklusive Zerlegung parallel zu berechnen. An dieser Stelle liegt das Hauptaugenmerk jedoch darauf, ein einheitliches Vorgehen zu etablieren. In Anbetracht der Berücksichtigung von Desktop-PCs folgt somit, dass ein verteiltes Rechnen ausschließlich genutzt wird, um die Numerik zu genau einer Matrix zu beschleunigen. Folglich ist die Notwendigkeit, mehrfach die Iterationsmatrix pro Zeitschritt zu zerlegen, als Negativpunkt gewertet.

- W-Methoden sind interpretierbar als gewisse voll implizite RK-Methoden, die bewusst auf die Ausführung einer Newton-Iteration pro Stufe beschränkt werden. Damit fällt den Startnäherung (3.15) besonderes Gewicht hinsichtlich der Ordnung der Methode zu, was sich in der Wahl der Parameter α_{ij} und γ_{ij} widerspiegelt. Dennoch sollte nicht außer Acht gelassen werden, dass das Kernprinzip ein Newtonprozess ist. Es erscheint somit z. B. unsinnig, eine Newton-Iterierte zu akzeptieren, wenn der Prozess divergiert. Einer W-Methode sind jedoch per se nicht die Möglichkeiten gegeben, dies (approximativ) zu überprüfen, da hierfür mindestens zwei Newtonschritte in mindestens einer Stufe nötig wären. Eine bekannte Ausnahme hiervon bildet die Extrapolation basierend auf der linear-impliziten Mittelpunktregel, s. /HAI 96, §IV.9.(9.30) und Aufg.IV.9.3/. Als potentieller Ersatz für den bisherigen Extrapolationsansatz sollte diese Methode jedoch nicht angesehen werden, da sie generell mit mehr Aufwand im Rahmen der Linearen Algebra belegt ist und auch nur eine Stufenordnung von Eins aufweist. Generell sollte als notwendige Bedingung angesehen werden, dass der zugrundeliegende Newtonprozess nicht divergiert. Dem Extrapolationsansatz bleibt somit nur die Option, mindestens einen zusätzlichen Newton-Schritt pro Zeitschritt durchzuführen.

Bemerkung 3.24. Es sei erwähnt, dass die Implementierung des Extrapolationsansatzes in ATHLET während der Berechnung von $T_{2,1}$ bereits ein zweites Newtoninkrement zur Kontraktionsschätzung berechnet und somit der Anforderung des letzten Punktes entspricht. Zur Verbesserung der Newtonapproximation wird das Inkrement jedoch nicht verwandt.

Obige Nachteile haben in der Praxis meist zur Folge, dass schlicht mehr Zeitschritte zur numerischen Lösung von (3.1) nötig sind, s. auch die Ergebnisse in Abschnitt 3.5.2. Zwar hängt der Fehlerschätzer für die Schrittweitenkontrolle von der lokalen Ordnung ab, jedoch geht diese nicht als allzu sensibler Parameter ein, s. die Diskussion in /DEU 13,

§ 5.2.2/. Auf alle Fälle führt Ordnungsreduktion zu echt größeren Fehlern in der numerischen Approximation. Ein weiteres Beiwerk hierzu ist durch die verhältnismäßig großen Fehlerkonstanten beim ATHLET-Extrapolationsansatz gegeben, s. Tab. 3.12.

Die Diskussion in diesem Abschnitt zeigt, dass L-Stabilität und Steifakkuratesse sowie eine hohe Stufenordnung sinnvolle Eigenschaften für eine Methode zur numerischen Lösung von (3.1) darstellen. Steife Lösungskomponenten sollten nicht den Fehler dominieren. Die Erfahrung zeigt, dass sich eine W-Methode nicht als unadäquat für die im ATHLET auftauchenden Differentialgleichungen erweist. Diese Erfahrung sowie die aufgeführten Eigenschaften hinsichtlich Robustheit wurden als Grundlage genommen, um neue Methoden, s. g. *Finite Iteration Runge-Kutta-Methoden*, zu konstruieren. Diese lassen sich als verallgemeinerte W-Methoden interpretieren, da zwar auch a priori die Anzahl an notwendigen Newton-Iterationsschritten festgelegt ist, um eine gewisse Ordnung zu erreichen, jedoch nicht zwingend jede Stufe mit einem einzelnen Iterationsschritt auskommt. Dies ist nicht per se nachteilig, da hierdurch auf natürliche Weise Schätzungen hinsichtlich des Kontraktionsverhaltens des Newtonprozesses zur Verfügung stehen.

3.3 Theorie zu Finite Iteration Runge-Kutta-Methoden

Wie im vorherigen Abschnitt gezeigt, kann jeder W-Methode eine RK-Methode zugeordnet werden, zu der die W-Methode eine „Finite Iteration“-Variante mit genau einem Newtonschritt pro Stufe darstellt, vgl. (3.18). Besondere Rolle spielen hierbei die Startnäherungen (3.15), da diese unmittelbar Einfluss auf die Ordnung der W-Methode haben. Untersuchungen zu RK-Methoden, die mit gegebenen Startnäherungen nur eine endliche Anzahl an Newton-Iterationen durchführen, wurden bereits in /JAC 96/ gemacht. Für die im Newtonprozess involvierte Matrix J als Approximation zu $\partial f_0/\partial y$ wurden jedoch nur Spezialfälle betrachtet, z. B. $J = 0$ oder $J = \partial f_0/\partial y$. Im Folgenden wird zunächst ein Ergebnis formuliert, Satz 3.26, welches für beliebiges $J \in \mathbb{R}^{n\times n}$ gilt und als Grundlage zur Konstruktion der Finite Iteration Runge-Kutta-Methoden, kurz $FiterRK$-Methoden, dient.

Um die Notation nicht zu überfrachten, wird zu einem autonomen System gewechselt, ggf. also $t' = 1$ hinzugefügt. Aus gleichem Grund sei für $\iota \in \mathbb{N}$ mittels $f_0^{(\iota)}$ die Ableitung $\partial^\iota f_0/\partial y^\iota$ bezeichnet. Des Weiteren sei $D_h \subset \mathbb{R}$ offen mit $0 \in D_h$ und hiermit $\Psi^{(q)}_{|h=0}$ die q-te Ableitung an $h = 0$ bezüglich einer hinreichend glatten Funktion $\Psi : D_h \to \mathbb{R}^\eta$, $\eta \in \mathbb{N}$. Man beachte, dass aus der Leibnizregel die Beziehung

$$(h\Psi(h))^{(q)}_{|h=0} = q \cdot \Psi(h)^{(q-1)}_{|h=0} \tag{3.58}$$

folgt. Für $d, r \in \mathbb{N}$, $\underline{K} : D_h \to \mathbb{R}^{r \cdot n}$ sowie $K : D_h \to \mathbb{R}^{d \cdot n}$ sei das Problem

$$K = hF\Big(\underline{K}, K\Big) \tag{3.59}$$

betrachtet und mit einer Approximation $\widetilde{\underline{K}}$ zu $\underline{K}$ eine zugehörige Newton-Iteration

$$K^{l+1} = hF\Big(\widetilde{\underline{K}}, K^l\Big) + h(A \otimes J)(K^{l+1} - K^l), \qquad l = 0, \ldots, \tag{3.60}$$

wobei

$$F(\underline{\kappa}, \kappa) = \Big(F_i(\underline{\kappa}, \kappa)\Big)_{i=1,\ldots,d}, \quad F_i(\underline{\kappa}, \kappa) := f\Big(y_0 + (\underline{A}_{(i,:)} \otimes I)\underline{\kappa} + (A_{(i,:)} \otimes I)\kappa\Big) \tag{3.61}$$

und $\underline{A} \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{d \times d}$ gilt. Man beachte, dass (3.60) für $J = 0$ eine Fixpunktiteration darstellt.

Bemerkung 3.25. Das System (3.7) zur Bestimmung von $k_1, \ldots, k_s$ ergibt sich zu (3.59) mit $d = s$ bei Weglassung von $\underline{K}$ für die Wahl

$$K := \begin{pmatrix} k_1 \\ \vdots \\ k_s \end{pmatrix} \tag{3.62}$$

und unter der Prämisse (3.11) sowie der Berücksichtigung der vorgenommenen Autonomisierung.

Satz 3.26 (Einfluss der Iteration)**.** *Sei angenommen, dass*

$$\begin{aligned} \widetilde{\underline{K}}^{(u)}_{|h=0} &= \underline{K}^{(u)}_{|h=0}, \qquad u = 0, \ldots, \mu, \\ K^{0(v)}_{|h=0} &= K^{(v)}_{|h=0}, \qquad v = 0, \ldots, \varphi, \end{aligned} \tag{3.63}$$

gelte und K^l für $l \geqslant 1$ über (3.60) *wohldefiniert sei. Dann gilt*

$$K^{l(q)}_{|h=0} = K^{(q)}_{|h=0}, \quad q = 0, \ldots, \min(\mu + 1, \varphi + \omega) \tag{3.64}$$

mit $\omega = l$ für $J \in \mathbb{R}^{n \times n}$ beliebig und mit $\omega = 2l$ für die Wahl $J = f_0^{(1)}$.

Beweis. Die Aussage ergibt sich aus der Anwendung von (3.58) und mittels eines geschachtelten Induktionsargumentes bezüglich der Iterations- und Ableitungsindizes. Für den Spezialfall $J = f_0^{(1)}$ wird zusätzlich die Beziehung

$$K^{l+1(q+1)}_{|h=0} = K^{(q+1)}_{|h=0} + (q+1)(A \otimes f_0^{(1)})\Big(K^{l(q)}_{|h=0} - K^{(q)}_{|h=0} + K^{l+1(q)}_{|h=0} - K^{l(q)}_{|h=0})\Big) = K^{(q+1)}_{|h=0}$$

ausgenutzt, welche aus der Formel von Faà di Bruno, /HAI 93, Lem. II.2.8/, und für $q \leqslant \min(\mu, \varphi + \omega - 1)$ folgt. □

Hieraus leitet sich sofort ein erstes einfaches Resultat hinsichtlich einer „Finite Iteration“-Variante für RK-Methoden ab.

Korollar 3.27. *Sei eine RK-Methode* (3.7) *der Ordnung p gegeben und $J \in \mathbb{R}^{n\times n}$ beliebig. Für den Newtonprozess* (3.60) *sei K^0 als Startapproximation für K aus* (3.62) *zu $K^0 \equiv 0$ gewählt sowie $A = \mathcal{A}$ gesetzt. Dann gilt*

$$K^{p(q)}_{|h=0} = K^{(q)}_{|h=0}, \quad q = 0, \dots, p$$

und somit

$$y(t_0 + h) - \Big(y_0 + (b^T \otimes I)K^p\Big) \in \mathcal{O}(h^{p+1}).$$

Beweis. Nach Definition in (3.59) ergibt sich $K^{(0)}_{|h=0} = 0$. Somit ist (3.63) für $\varphi = 0$ erfüllt ($\underline{K}$-Größen brauchen hier nicht berücksichtigt zu werden). Der restliche Beweis folgt aus Satz 3.26 und der Definition der Fehlerordnung über Taylorapproximationen an $h = 0$. □

Bemerkung 3.28. Üblicherweise sind im Rahmen einer impliziten RK-Methode Approximationen $Y_i^0 =: y_0 + z_i^0$ für $y(t_0 + c_i h)$ gegeben, welche als Startnäherungen zur Bestimmung der Stufenwerte Y_i dienen. Der Schritt hin zur Darstellung, wie sie in Satz 3.26 genutzt wird, ergibt sich wie folgt. Mit dem Verschubvektor

$$Z = \begin{pmatrix} z_1 \\ \vdots \\ z_s \end{pmatrix} := \begin{pmatrix} Y_1 \\ \vdots \\ Y_s \end{pmatrix} - \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \otimes y_0 \tag{3.65}$$

und der Schreibweise (3.62) lässt sich der Zusammenhang (3.9) darstellen als

$$Z = (\mathcal{A} \otimes I)K. \tag{3.66}$$

Definiert man nun analog hierzu K^0 über

$$Z^0 := \begin{pmatrix} z_1^0 \\ \vdots \\ z_s^0 \end{pmatrix} =: (\mathcal{A} \otimes I)K^0 \tag{3.67}$$

stehen formal die in Satz 3.26 genutzten Größen zur Verfügung. Man beachte, dass aus der Gültigkeit von $\mathcal{C}(q)$ die Beziehung

$$y(t_0 + c_i h) - Y_i \in \mathcal{O}(h^{q+1}), \qquad i = 1, \dots, s,$$

folgt, vgl. (3.50). Ist auf der anderen Seite

$$y(t_0 + c_i h) - Y_i^0 \in \mathcal{O}(h^{r+1}), \qquad i = 1, \dots, s, \tag{3.68}$$

mit $r \geqslant 0$ erfüllt, so liefert Differenzbildung unter Zuhilfenahme von (3.66) und (3.67) sofort

$$K - K^0 \in \mathcal{O}(h^{\min(q,r)+1}).$$

Somit sichert $\mathcal{C}(q)$ für die Approximation K^0 ein Gütemaß zu, welches der Qualität der Startnäherungen der Stufenwerte entspricht oder bei sehr hochwertigen Stufenwertapproximationen mindestens gleich der Stufenordnung ist. Die Analyse in /JAC 96/ setzt nur (3.68) voraus und ist in dem Sinne allgemeiner als der hier beschrittene Pfad. Dies sei nicht als Einschränkung empfunden, da an dieser Stelle das Interesse an $FiterRK$-Methoden der Ordnung p liegt, welche auch eine Stufenordnung von p aufweisen. Obige Diskussion zeigt, dass sich in diesem Kontext auf die RK-Größen K und K^l bzw. deren Z-Pendants beschränkt werden kann.

3.3.1 Berücksichtigung von Stufenordnung und Steifakkuratesse

Satz 3.26 und Korollar 3.27 sind bewusst allgemein gehalten und beziehen keine spezielle Struktur der Koeffizienten-Matrix $\mathcal{A}$ oder die Bedingung $\mathcal{C}(p)$ mit ein. Wie in Abschnitt 3.2.4 herausgestellt, erweist es sich für eine RK-Methode der Ordnung p als sinnvolle Eigenschaft, auch eine Stufenordnung von p zuzusichern. Des Weiteren ist es im Sinne einer praktischen Schrittweitensteuerung vorteilhaft, sich auf eingebettete Methoden zum Ordnungspaar $p+1(p)$ zu beschränken. Nach (3.52) und der Fehlerdarstellung (3.46) lässt sich Steifakkuratesse sehr leicht durch die Wahl $b^T = \mathcal{A}_{(i,:)}$, falls $c_i = 1$, erreichen. Um all dies zu berücksichtigen, sei für die weitere Diskussion die Gültigkeit folgender Bedingungen festgelegt:

$$\mathcal{C}(p) : \mathcal{A}C^{k-1} = \frac{1}{k}C^k e, \quad k = 1, \ldots, p, \qquad \text{analog zu (3.11) und (3.48)} \qquad \text{(3.69a)}$$

$$\mathcal{A}_{(s,:)}C^p e = (p+1)^{-1}c_s^{p+1}, \quad b^T = \mathcal{A}_{(s,:)}, \quad c_{s-1} = c_s = 1. \qquad \text{(3.69b)}$$

Die Bedingungen (3.69b) sind bezüglich der Stufen o.B.d.A. zu verstehen. In der dargestellten Form wird zusätzlich lokale Extrapolation genutzt. Die gewünschten Ordnungseigenschaften lassen sich mittels /BUT 64, Th. 7/ herleiten.

Bemerkung 3.29. Die Darstellung (3.52) ist nur valide, falls $\mathcal{A}$ regulär ist. In der Regel ist dies für implizite RK-Methoden der Fall. Als einzige Ausnahme seien hier Methoden zugelassen, welche von der Struktur

$$\mathcal{A} = \begin{pmatrix} 0 & 0^T \\ \tilde{a} & \tilde{\mathcal{A}} \end{pmatrix} \qquad \text{(3.70)}$$

mit $\tilde{a} \in \mathbb{R}^s$ und $\tilde{\mathcal{A}} \in \mathbb{R}^{s-1 \times s-1}$ sind, also eine explizite erste Stufe berücksichtigen. Im Sinne der Fehlerdarstellung (3.46) steuert eine solche Stufe keinen Fehler bei, so dass sich in der Diskussion hinsichtlich Steifakkuratesse auf $\tilde{\mathcal{A}}$ beschränkt werden kann. Diese wird als regulär vorausgesetzt, und die Analyse in /HAI 96, § IV.15/ zeigt, dass unter dieser Bedingung ganz analog, i. e. via (3.69b), Steifakkuratesse sichergestellt werden kann.

Um der Bedingung (3.69a) zu genügen, bedarf es eines Blockes $A_m \in \mathbb{R}^{m \times m}$ mit $m \geqslant p$ in $\mathcal{A}$, wobei eine evtl. explizite erste Stufe zum Block gezählt wird. Die Blockstufen können durch weitere Diagonalstufen ergänzt werden, welches sich positiv auf Stabilität und Fehlerkonstanten auswirkt, s. hierzu /BUT 90/,/BUT 98/. Es werden somit Koeffizientenmatrizen der folgenden Form betrachtet:

$$\mathcal{A} = \begin{pmatrix} a_{11} & \cdots & a_{1m} & & & \\ \vdots & \ddots & \vdots & & & \\ a_{m1} & \cdots & a_{mm} & & & \\ a_{m+1,1} & \cdots & \cdots & a_{m+1,m+1} & & \\ \vdots & & & & \ddots & \\ a_{s1} & \cdots & \cdots & \cdots & \cdots & a_{ss} \end{pmatrix} \qquad \begin{aligned} A_m &:= (a_{ij})_{i,j=1,\ldots,m} \\ a_{ii} &\neq 0,\ i = m+1, \ldots, s. \end{aligned} \tag{3.71}$$

Bemerkung 3.30. Die zu einer klassischen W-Methode zugeordnete RK-Methode weist keinen oder anders interpretiert einen Block der Größe $m = 1$ auf. Im Sinne der Implementation bietet sich die erste Interpretation an, da für nichttriviale Blöcke, $m > 1$ oder $m > 2$ für Methoden mit expliziter erster Stufe, zusätzliche Transformationen zu berücksichtigen sind, s. Unterabschnitt 3.3.4.1.

Die Struktur (3.71) der Matrix $\mathcal{A}$ lässt sich gewinnbringend in Bezug auf die notwendige Anzahl an Newtoniterationsschritten im $FiterRK$-Ansatz nutzen, ohne das Ergebnis von Korollar 3.27 abzuschwächen.

Die Diagonalstufen können im Newtonprozess *einzeln* abgearbeitet werden, und pro Diagonalstufe können zuvor berechnete $k_j^{l_j}$-Werte im Sinne von (3.15) genutzt werden, um qualitativ hochwertige Startnäherungen k_i^0 für Stufe $i > m$ zu konstruieren. Der hochgestellte Index l_j reflektiert hierbei die durchgeführten Newtonschritte für Stufe j. Ob der Blockstruktur in $\mathcal{A}$ stehen mindestens m solcher $k_j^{l_j}$-Werte pro Diagonalstufe zur Verfügung. In Formeln ergibt sich analog zu (3.15) und (3.17a) für den ersten Newtonschritt

in Stufe $i > m$ die Darstellung

$$\gamma_{ii} k_i^0 = \sum_{j=1}^{i-1} -\gamma_{ij} \cdot k_j^{l_j}, \qquad \gamma_{ii} := a_{ii}, \tag{3.72a}$$

$$k_i^1 = hf\Big(y_0 + \sum_{j=1}^{i-1} \alpha_{ij} k_j^{l_j}\Big) + hJ\Big(\sum_{j=1}^{i-1} \gamma_{ij} k_j^{l_j} + \gamma_{ii} k_i^1\Big). \tag{3.72b}$$

Als Vorbereitung für die nächsten Aussagen erweist sich folgendes Lemma als nützlich.

Lemma 3.31 (Kondensation)**.** *Sei $q \geqslant 1$ und gelte für eine RK-Methode* (3.7) *die Bedingung $\mathcal{C}(p)$ mit $p \geqslant q-1$. Dann existiert $\zeta_q = \zeta_q\big(f_0^{(0)}, \ldots, f_0^{(q-1)}\big) \in \mathbb{R}^n$, so dass $k_{i|h=0}^{(q)} = \zeta_q \cdot c_i^{q-1}$, $i = 1, \ldots, s$, gilt, und ζ_q ist unabhängig von i.*

Beweis. Mittels rekursiven Nutzens von (3.69a) und einiger weiteren kleineren Nebenrechnungen ergibt sich die Behauptung aus der B-Reihen-Darstellung von k_i. Siehe z. B. /HAI 93, §II.2/, insbesondere Satz 2.11 zur Herleitung der besagten B-Reihen-Darstellung. □

Das obige Lemma zeigt sehr schön, welch immense Vereinfachungen in der Darstellung der k_i das Einhalten der Bedingung $\mathcal{C}(p)$ zur Folge hat. Dies wird im folgenden Satz zur Konstruktion der Startnäherung aus (3.72a) ausgenutzt.

Satz 3.32 (Diagonalstufen)**.** *Sei $i > m$ und gelte $k_{j\,|h=0}^{l_j\,(q)} = k_{j|h=0}^{(q)}$ für $j = 1, \ldots, i-1$ und $q = 0, \ldots, \mu$ sowie $\mathcal{C}(\mu-1)$ und $\sum_{j=1}^{i} \gamma_{ij} c^{k-1} = 0$ für $k = 1, \ldots, \eta$. Sei k_i^l wohldefiniert über* (3.72b) *und* (3.60)*. Dann gilt*

$$k_{i|h=0}^{l\,(q)} = k_{i|h=0}^{(q)}, \quad q = 0, \ldots, \min\Big(\mu+1, \min(\eta, \mu) + \omega\Big)$$

mit ω wie in Satz 3.26 gewählt.

Beweis. Mit $\mathcal{C}(\mu-1)$, den Bedingungen an $k_j^{l_j}$, $j = 1, \ldots, i-1$, und Lemma 3.31 ergibt sich $k_{i\,|h=0}^{0\,(q)} = k_{i|h=0}^{(q)}$ für $q = 0, \ldots, \min(\eta, \mu)$. Die Behauptung folgt dann aus einer Anwendung von Satz 3.26 für $\underline{\widetilde{K}} = ({k_1^{l_1}}^T, \ldots, {k_{i-1}^{l_{i-1}}}^T)^T$, $A = (\mathcal{A})_{ii}$ und $K^0 = k_i^0$. □

Die Bedingung an die γ_{ij} in Satz 3.32 lässt sich mit Γ und A aus (3.18) sowie C aus (3.69a) schreiben als

$$\Gamma C^{k-1} = 0, \quad k = 1, \ldots, \eta, \qquad \overset{(3.16),(3.69a)}{\Longleftrightarrow} \qquad AC^{k-1} = \frac{1}{k} C^k e, \quad k = 1, \ldots, \eta.$$

Ganz analog zu (3.48) wird somit für die α-Stufen eine gewisse Approximationsgüte gefordert. Dies motiviert die Definition der α-*Stufenordnung*. Gilt

$$\mathcal{C}_\alpha(q): \quad AC^{k-1} = \frac{1}{k} C^k e, \quad k = 1, \ldots, q, \tag{3.73}$$

so besitzt die zugehörige $FiterRK$-Methode eine α-Stufenordnung von q.

Korollar 3.33. *Gelte $\mathcal{C}(p)$ und $\mathcal{C}_\alpha(p)$ sowie $k_j^{l_j(q)}{}_{|h=0} = k_j^{(q)}{}_{|h=0}$ für die Blockstufen $j = 1, \ldots, m$ und für $q = 0, \ldots, p$. Dann folgt*

$$k_i^{1(q)}{}_{|h=0} = k_i^{(q)}{}_{|h=0} \qquad \text{für} \quad q = 0, \ldots, p+1 \quad \text{und} \quad i = m+1, \ldots, s, \tag{3.74}$$

und die zugehörige $FiterRK$-Methode ist von der Ordnung p.

Beweis. Die Behauptung (3.74) ergibt sich sofort aus Satz 3.32 und einem Induktionsargument über die Indizes der Diagonalstufen. Die Ordnungsaussage folgt aus Korollar 3.27. □

Bemerkung 3.34. Man beachte, dass $\mathcal{C}_\alpha(p)$ stets erfüllbar ist, wenn $\mathcal{C}(p)$ erfüllt ist. Denn aus der Gültigkeit von $\mathcal{C}(p)$ folgt, dass für den $m \times m$-Block A_m in $\mathcal{A}$ aus (3.71) die Bedingung $m \geqslant p$ gilt. Somit stehen für die i-te Stufe mit $i > m$ genau $m + i - 1$ α-Werte zur Verfügung.

Korollar 3.33 is ein sehr erfreuliches Resultat. Es zeigt, dass Diagonalstufen mit einem minimalen Aufwand an Implizitheit, i. e. nur ein Newtonschritt pro Stufe, bestimmt werden können, ohne an Approximationsgüte zu verlieren. Somit stehen Diagonalstufenwerte sehr kostengünstig zur Verfügung, sofern die Blockstufen hinreichende Qualität vorweisen können.

3.3.2 Verflechtung von Stufen

Satz 3.32 nutzt die Güte vorheriger Stufenwerte k_j als hinreichende Bedingung, um Aussagen über die Qualität von k_i mit $i > j$ zu treffen. In gewisser Weise sind diese Bedingungen auch notwendig, wie folgende Betrachtung zeigt. Gelte $\mathcal{C}(p)$. Dann ergibt sich entsprechend (3.50) die Beziehung

$$\begin{aligned} & y(t_0 + c_i h) - Y_i \in \mathcal{O}(h^{p+1}), \quad i = 1, \ldots, s \\ & \qquad \Longleftrightarrow \\ & \left[\begin{pmatrix} y(t_0 + c_1 h) \\ \vdots \\ y(t_0 + c_s h) \end{pmatrix} - e \otimes y_0 \right]^{(q)}_{|h=0} - (\mathcal{A} \otimes I) K^{(q)}_{|h=0} = 0, \quad q = 0, \ldots, p, \end{aligned}$$

mit K aus (3.62). Nur diese Größe bietet sich an, um durch eine Approximation ersetzt zu werden. Zur Bewahrung der Stufenordnung, muss eine solche Näherung $\widetilde{K}$ die Bedingungen

$$(\mathcal{A} \otimes I) \widetilde{K}^{(q)}_{|h=0} = (\mathcal{A} \otimes I) K^{(q)}_{|h=0}, \quad q = 0, \ldots, p$$

erfüllen. Für reguläres $\mathcal{A}$ ist dieses äquivalent zu

$$\widetilde{K}^{(q)}_{|h=0} = K^{(q)}_{|h=0}, \quad q = 0, \dots, p. \tag{3.75}$$

Mit der Wahl $(\widetilde{K})_j = k_j^{l_j}$ ergibt es sich somit auch als notwendig, die Approximationen $k_j^{l_j}$ eins-zu-eins mit den voll impliziten Größen k_j abzugleichen.

Bemerkung 3.35. Im Falle einer expliziten ersten Stufe ist $\mathcal{A}$ singulär. Mit der Wahl $(\widetilde{K})_1 = k_1 = hf_0$ kann sich in obiger Betrachtung auf $\tilde{\mathcal{A}}$ aus (3.70) beschränkt werden, welches als regulär vorausgesetzt wurde. Somit bleibt ganz analog auch in diesem Fall die Aussage gültig.

Wie zu Beginn dieses Unterabschnittes festgelegt, liegt das Interesse an eingebetteten Methoden, vgl. (3.69). Argumentiert man ähnlich zum obigen Fall der Stufenwerte, so ergibt sich mit (3.69b) die zusätzliche Bedingung

$$(\mathcal{A}^T_{(s,:)} \otimes I)\widetilde{K}^{(p+1)}_{|h=0} = (\mathcal{A}^T_{(s,:)} \otimes I)K^{(p+1)}_{|h=0}. \tag{3.76}$$

Im Gegensatz zu (3.75) können sich die einzelnen $k_j^{l_j}$ gegenseitig unterstützen, um (3.76) zu erfüllen. An dieser Stelle werden zwei Spezialfälle betrachtet, welche zur Konstruktion der in Abschnitt 3.4 diskutierten Methoden genutzt werden.

Satz 3.36 (Auslöschung)**.** *Sei* $K_r = (k_1^T, \dots, k_r^T)^T$ *entsprechend* (3.62) *mit* $r < s$ *sowie* $\mathcal{A}_r = \mathcal{A}_{(1:r,1:r)}$ *gegeben. Sei die Iteration* (3.60) *ohne* $\underline{K}$*-Eingabe und mit einer Startnäherung* K_r^0 *wie in Satz 3.26 mit* $\varphi \leqslant p$ *und für* $A = \mathcal{A}_r$ *betrachtet. Des Weiteren seien vorausgesetzt*

$$k_{i\,|h=0}^{0(\varphi+1)} = c_i^\varphi \xi^0 \quad \textit{mit einem} \quad \xi^0 \in \mathbb{R}^n, \tag{3.77a}$$

$$\sum_{i=1}^{r} b_i c_i^p = 0 \tag{3.77b}$$

sowie $\mathcal{C}(p)$*. Falls* $K_r^{p-\varphi}$ *wohldefiniert ist über* (3.60)*, dann gilt*

$$\sum_{i=1}^{r} b_i k_i^{p-\varphi\,(p+1)}_{|h=0} = \sum_{i=1}^{r} b_i k_{i\,|h=0}^{(p+1)} = 0. \tag{3.78}$$

Beweis. Für den Fall $\varphi = p$ folgt die Aussage direkt aus den Annahmen und aus Lemma 3.31. Sei im Folgenden $\varphi < p$. Analog zum Beweis zu Satz 3.26 ergibt sich vermittels der Formel von Faà di Bruno die Beziehung $K_{r\,|h=0}^{1(\varphi+2)} = K_{r|h=0}^{(\varphi+2)} + (\varphi+2)\Big(A \otimes (J - f_0^{(1)})\Big)\Big(K_{r\,|h=0}^{1(\varphi+1)} - K_{r\,|h=0}^{0(\varphi+1)}\Big)$. Wegen (3.77a), Lemma 3.31 und (3.69a) lässt sich dieser Zusammenhang schreiben als $K_{r\,|h=0}^{1(\varphi+2)} = K_{r|h=0}^{(\varphi+2)} + C_r^{\varphi+1} e_r \otimes \xi^1$ mit $C_r = \operatorname{diag}(c_1, \dots, c_r)$, $e_r = (1, \dots, 1)^T \in \mathbb{R}^r$ und einem $\xi^1 \in \mathbb{R}^n$. Mittels Induktion und analog zur eben durchgeführten Argumentation für K_r^1 ergibt sich $K_r^{p-\varphi\,(p+1)}_{|h=0} = K_{r\,|h=0}^{(p+1)} + C_r^p e_r \otimes \xi^{p-\varphi}$ mit einem $\xi^{p-\varphi} \in \mathbb{R}^n$, welches wegen (3.77b) die Behauptung verifiziert. □

Hiermit lässt sich als direkte Erweiterung zu Korollar 3.33 die folgende Aussage formulieren.

Korollar 3.37. *Gelte $\mathcal{C}(p)$ und $\mathcal{C}_\alpha(p)$ sowie* (3.77b) *für $r = m$. Sei für die Blockstufen $K_b = (k_1^T, \ldots, k_m^T)^T$ eine Startnäherung über $K_b^0 = 0$ gegeben und seien hiermit p Schritte der Iteration* (3.60) *mit $A = A_m$ aus* (3.71) *durchgeführt. Für die Diagonalstufen $i = m+1, \ldots, s$ sei entsprechend* (3.72b) *je genau ein Newtonschritt* (3.60) *ausgeführt. Dann gilt*

$$
\begin{aligned}
&k_{i\,|h=0}^{p(q)} = k_{i|h=0}^{(q)} \qquad \textit{für} \quad q = 0, \ldots, p \quad \textit{und} \quad i = 1, \ldots, m, \\
&k_{i\,|h=0}^{1(q)} = k_{i|h=0}^{(q)} \qquad \textit{für} \quad q = 0, \ldots, p+1 \quad \textit{und} \quad i = m+1, \ldots, s, \\
&\sum_{i=1}^{m} b_i k_{i\,|h=0}^{p(q)} + \sum_{i=m+1}^{s} b_i k_{i\,|h=0}^{1(q)} = \sum_{i=1}^{s} b_i k_{i|h=0}^{(q)}, \quad q = 0, \ldots, p+1.
\end{aligned}
$$

Und unter der Voraussetzung (3.69b) *ist die zugehörige $FiterRK$-Methode von der Ordnung $p+1$ mit einer Stufenordnung von p.*

Diagonalstufen mit niedrigem Stufenindex sind durch die Vorgaben $\mathcal{C}(p)$ und $\mathcal{C}_\alpha(p)$ in ihren Freiheitsgraden stärker beschränkt als Stufen höheren Indexes. Mit den beiden Stufenordnungsbedingungen an der Hand zeigt Korollar 3.33, dass eine Approximationsgüte bis zur Ableitungsordnung $p+1$ für die Diagonalstufen erreicht wird. Um Freiheitsgrade für niedrig indizierte Stufen zu ermöglichen, um z. B. gewissen Stabilitätsanforderungen zu genügen, kann wieder (3.76) genutzt werden. Einen Spezialfall deckt folgender Satz ab.

Satz 3.38 (Diagonalverpflechtung)**.** *Gelte mindestens $\mathcal{C}(p-1)$ sowie $\mathcal{C}_\alpha(p-1)$ und seien mit $i_1, \ldots, i_\nu \in \{m+1, \ldots, s\}$ diejenigen Indizes von Diagonalstufen bezeichnet, für welche*

$$
\sum_{j=1}^{i} \gamma_{ij} c_j^{k-1} = 0 \quad \textit{für } k = 1, \ldots, p-1 \qquad \textit{aber} \qquad \sum_{j=1}^{i} \gamma_{ij} c_j^{p-1} \neq 0
$$

gilt (somit also $\mathcal{C}_\alpha(p)$ nicht erfüllt wird). Sind k_i^1 wohldefiniert über (3.72)*, dann folgt mit der Bedingung*

$$
\sum_{k=1}^{\nu} b_{i_k} \sum_{j=1}^{i_k} \gamma_{i_k j} c_j^{p-1} = 0 \tag{3.79}
$$

die Beziehung

$$
\sum_{k=1}^{\nu} b_{i_k} k_{i_k\,|h=0}^{1\;(p+1)} = \sum_{k=1}^{\nu} b_{i_k} k_{i_k\,|h=0}^{\;(p+1)}.
$$

Beweis. Das Vorgehen ist analog zu der Argumentation im Beweis von Satz 3.36. Unter Ausnutzung der speziellen Struktur der Iterierten k_i^0 und k_i^1 wie in (3.72) dargestellt, ergibt sich unter Zuhilfenahme der Formel von Faà di Bruno die Relation $k^{1\ (p+1)}_{i_k|h=0} = k^{1\ (p+1)}_{i_k|h=0} + (p+1)(J - f_0^{(1)})(\sum_{j=1}^{i_k} \gamma_{i_k j} c_j^{p-1})\xi_{p-1}$ mit einem $\xi_{p-1} \in \mathbb{R}^n$ und für $i_1, \ldots, i_\nu$. Die Behauptung folgt dann unmittelbar aus der Bedingung (3.79). □

Bemerkung 3.39. In der Bedingung (3.79) treten die Freiheitsgrade b_{i_k} und $\gamma_{i_k j}$ linear auf, womit (3.79) leicht nach einer dieser Größen umgestellt werden kann.

Bemerkung 3.40. Man beachte, dass zur Berechnung von $k^1_{i_1}, \ldots, k^1_{i_\nu}$ das gleiches J verwandt werden muss, um den Voraussetzungen von Satz 3.38 zu genügen. Satz 3.32 hingegen ist nicht darauf angewiesen. In der Praxis spielt dies jedoch meist keine Rolle, da in der Regel nur eine Approximation J pro Schritt bestimmt wird.

3.3.3 Sicherstellung eines Einpunktspektrums

Mit einer Struktur der Matrix $\mathcal{A}$ wie in (3.71) gezeigt, hängen die ersten m Stufenwerte direkt voneinander ab. Um den Voraussetzungen von Korollar 3.33 zu genügen, könnte man entsprechend Satz 3.26 p Iterationsschritte der Newtoniteration (3.60) mit $A = A_m$ durchführen. Dann wären jedoch lineare System der Dimension $m \cdot n$ zu lösen. Das ist inakzeptabel, da der Aufwand zum Lösen von linearen Systemen quadratisch von der Dimension abhängt – eine Zerlegung der involvierten Matrix vorausgesetzt. Dies betrifft auch zu einem gewissen Grad etwaige iterative Methoden, da hier oft über eine unvollständige LR-Zerlegung der Präkonditionierer konstruiert wird.

Reduzieren lässt sich der Rechen- und Speicherbedarf, wenn die Matrix A_m über bestimmte Ähnlichkeitstransformationen auf Dreiecksgestalt gebracht wird. Details hierzu finden sich in Abschnitt 3.3.4. Mittels der Dreiecksgestalt lassen sich äquivalent zum großen System sukzessive m „kleine“ lineare Systeme der Dimension n lösen. Somit ist der Aufwand vergleichbar mit dem zur Berechnung von m Diagonalstufen. Weist A_m ein Einpunktspektrum $\mathrm{spec}(A_m) = \{\gamma\}$ auf, so bedarf es zusätzlich nicht einmal eines Wechsels der in den linearen Gleichungssystemen involvierten Matrix. Wird $a_{ii} = \gamma$ für $i > m$ gesetzt, so erstreckt sich diese Ersparnis auch auf die Diagonalstufen. Ein Einpunktspektrum stellt somit eine wünschenswerte Eigenschaft dar.

Um auch Methoden mit expliziter erster Stufe zu berücksichtigen, wird die Forderung nach einem Einpunktspektrum und die zugehörige Notation entsprechend angepasst: Die Matrix $\mathcal{A}$ aus (3.71) darf singulär sein, aber nur aufgrund einer expliziten ersten Stufe, i. e., $c_1 = 0$ sowie $k_1 = f_0$. In diesem Fall wird $\mathcal{A}$ als *1-explizit* bezeichnet. Ein reguläres $\mathcal{A}$

wird *0-explizit* genannt. Somit steht die Forderung, dass ein k-explizites $\mathcal{A}$ mit $k \in \{0, 1\}$ einen $s - k$-fachen Eigenwert γ besitzen soll.
Mit den weiteren Nebenbedingungen (3.69) bedarf es eines Blockes $A_m \in \mathbb{R}^{m \times m}$ in $\mathcal{A}$ mit $m \geqslant p$ wie bereits auf Seite 86 diskutiert. Für 0-explizites $\mathcal{A}$ und $m = p$ sind die Blockkoeffizienten $a_{11}, \ldots, a_{mm}$ eindeutig über $\mathcal{C}(p)$ in Abhängigkeit von $c_1, \ldots, c_m$ definiert. In diesem Fall besitzt A_m genau dann einen einzigen verschiedenen Eigenwert γ, wenn $c_1, \ldots, c_m$ als γ-Vielfache der m verschiedenen Nullstellen des m-ten Laguerre-Polynoms $L_m^{(0)}$ gewählt werden, s. /HAI 96, Lemma IV.8.1/. Ein ähnliches Resultat lässt sich für 1-explizites $\mathcal{A}$ herleiten, jedoch bezüglich verallgemeinerter Laguerre-Polynome $L_m^{(1)}$, s. Bemerkung 3.43. Zur Definition der (verallgemeinerten) Laguerre-Polynome siehe (3.80a) für $\varphi = 1$ und $k = 0$ bzw. $k = 1$. Für $m = p$ sind also bereits alle Freiheitsgrade (bis auf die γ-Vielfachheit) und damit sämtliche Stabilitätsaspekte festgelegt. Um in dieser Hinsicht mehr Flexibilität zu erhalten, wird im Folgenden ein Block der Größe $m = p + 1$ betrachtet, und es werden Bedingungen für einen $s - k$-fachen Eigenwert für k-explizites $\mathcal{A}$ präsentiert, s. Satz 3.42. Im klassischen Fall $m = p = s$ wird die zugrundeliegende RK-Methode (3.7) für die Beweisführung als Kollokationsansatz interpretiert, s. /HAI 93, Th. II.7.8/, wohingegen für $m = p + 1 = s$ ein gestörter Kollokationsansatz zu betrachten ist. Da die Verbindung einer RK-Methode zu einem (gestörten) Kollokationsansatz im Rahmen dieser Arbeit nur für die Beweisführung zu Satz 3.42 Relevanz hat, wird an dieser Stelle auf eine Erörterung des umfangreichen Beweises verzichtet und stattdessen auf /STE 16/ verwiesen. Die Beweise zu den Hilfsaussagen Proposition 3.44 und 3.45 zur Konstruktion von Methoden mit $m = p + 1$ sind ebenfalls in /STE 16/ zu finden.

Bemerkung 3.41. Ein weiterer in der Literatur vertretener Ansatz, um mehr Flexibilität in der Wahl der relativen Schrittweiten c_i bei zeitgleicher Einhaltung von $\mathcal{C}(p)$ zu erhalten, ist vermittels des Konzeptes der *effective order* gegeben, s. /BUT 16, § 365/. Im Falle einer konstanten Schrittweite ist dieses Konzept sehr effizient umsetzbar. Jedoch ist es im Vergleich zu dem hier gewählten Weg der Blockvergrößerung deutlich weniger verträglich mit einer adaptiven Schrittweitensteuerung und wird daher nicht weiter für diese Arbeit berücksichtigt.

Im Folgenden ist eine Beschränkung auf den Block, i. e. $\mathcal{A} = A_m$, statthaft, da Diagonalstufen nur trivial auf das Spektrum von $\mathcal{A}$ wirken. Ebenso bedarf es keiner besonderen Berücksichtigung des $FiterRK$-Konzeptes, da dieses keinen Einfluss auf das Spektrum von $\mathcal{A}$ hat.

Zur Herleitung der Spektrumsbedingungen wird auf *gestörte* Laguerre-Polynome zurückgegriffen: Für $\varphi \in \mathbb{R}$ und $k \in \{0, 1\}$ sei das m-te gestörte Laguerre-Polynom

$L^{(k)}_{m,\varphi} : \mathbb{R} \to \mathbb{R}$ definiert vermittels

$$L^{(k)}_{m,\varphi}(t) := \sum_{i=0}^{m-1} \binom{m+k}{i+k} (-1)^i \frac{t^i}{i!} + (-1)^m \frac{t^m}{m!} \cdot \varphi \tag{3.80a}$$

und hierzu die algebraische Kurve

$$\Xi_{k,m} := \{(t, \varphi) \in \mathbb{R} \times \mathbb{R} \mid L^{(k)}_{m,\varphi}(t) = 0\}. \tag{3.80b}$$

Wird $\varphi_i = 1$ gesetzt, ergeben sich die bekannten Fassungen. Folglich ist nach /ABR 72, § 22.16/ die untere Schranke $m \leqslant |\Xi_{k,m}|$ garantiert. Mit den obigen Definitionen lässt sich folgende Hauptaussage zur Sicherung eines Einpunktspektrums formulieren.

Satz 3.42. *Sei $k \in \{0, 1\}$ und $m = p + 1 = s$ für $p \geqslant 1$ gegeben. Möge $\mathcal{A}$ k-explizit sein und $\mathcal{C}(p)$ für paarweise verschiedene $c_1, \ldots, c_m$ gelten. Dann besitzt $\mathcal{A}$ einen $m-k$-fachen Eigenwert $\gamma \neq 0$ genau dann, wenn*

$$\chi_F : \mathcal{A}C^p e = (p+1)^{-1} F C^{p+1} e \tag{3.81}$$

für $F := \operatorname{diag}(\varphi_1, \ldots, \varphi_m)$ mit paarweise verschiedenen Zwei-Tupeln $(c_{i+k}/\gamma, \varphi_{i+k}) \in \Xi_{k,m-k}$, $i = 1, \ldots, m-k$, und im Falle von $k = 1$ mit beliebigem $\varphi_1 \in \mathbb{R}$ gilt.

Bemerkung 3.43. Wird $F = I$, das heißt, $\varphi_1 = 1$ für $i = 1, \ldots, m$ sowie $k = 0$ gesetzt, so leitet sich direkt aus Satz 3.42 die klassische Aussage in /HAI 96, Lemma IV.8.1/ ab, da dann die Konjunktion aus $\mathcal{C}(p)$ und χ_F der Bedingung $\mathcal{C}(p+1)$ entspricht. Ein analoges Resultat kann aus obigem Satz für Methoden mit expliziter erster Stufe gefolgert werden, wenn der Fall $F = I$ und $k = 1$ betrachtet wird.

Satz 3.42 liefert das erfreuliche Ergebnis, dass die Zusatzbedingung χ_F bei der Entwicklung von Methoden recht einfach zu handhaben ist. Die Paare (c_i, φ_i) müssen nur so gewählt werden, dass sie paarweise verschieden sind und auf der Kurve $\Xi_{m,k}$ liegen. Genau hierdurch ergibt sich die durch die Vergrößerung des Blocks erkaufte Flexibilität: Alle Punkte auf der Kurve $\Xi_{k,m}$ stehen per se als Kandidaten zur Verfügung. Exemplarisch ist eine solche Kurve für $m = 3$ und $k = 0$ in Abb. 3.3 zu sehen.
Neben Satz 3.42 erweisen sich folgende Aussagen zur Konstruktion von Methoden mit $m = p + 1$ als recht nützlich.

Proposition 3.44 (Fehlerkonstante)**.** *Sei $k \in \{0, 1\}$ und $s = m$ gesetzt. Sei des Weiteren $\mathcal{A}$ k-explizit und gelte $\mathcal{C}(p)$ für paarweise verschiedene $c_1, \ldots, c_s$. Zusätzlich möge χ_F aus* (3.81) *für $F = \operatorname{diag}(\varphi_1, \ldots, \varphi_s) \in \mathbb{R}^{s \times s}$ erfüllt sein. Wird die zugehörige RK-Methode* (3.7) *auf $y' = \lambda y$ mit $t_0 = 0$ und $y_0 = 1$ angewandt, dann gilt für die Stufenwerte Y_i, $i = 1, \ldots, s$,*

$$\exp(c_i z) - Y_i = \frac{(1 - \varphi_i)}{(p+1)!} \cdot (c_i z)^{p+1} + \mathcal{O}\big((c_i z)^{p+2}\big) \qquad \textit{mit} \quad z = h\lambda. \tag{3.82}$$

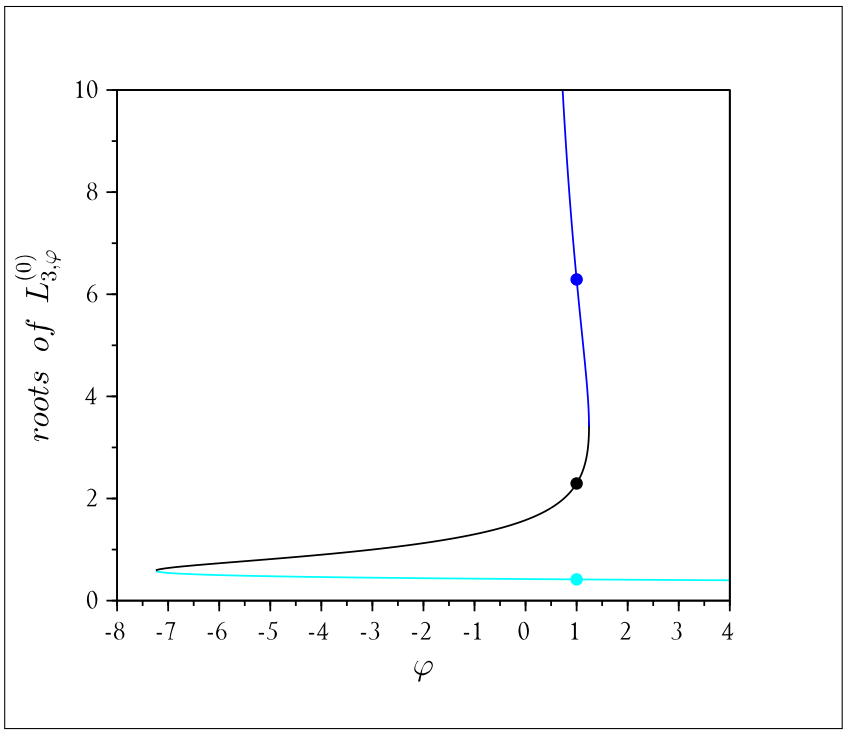


Abb. 3.3 Ausschnitt der algebraischen Kurve $\Xi_{m,k}$ für $m = 3$ und $k = 0$. Die drei Nullstellen des Laguerre-Polynoms $L_3^{(0)}$ sind je mit einem Kreis markiert. Wird φ derart gewählt, dass zwei unterschiedlich gefärbte Anteile der Kurve bei $[\varphi|c]$ aufeinandertreffen, so besitzt $L_{3,\varphi}^{(0)}$ die doppelte Nullstelle c.

Die Aussage zeigt zum einen, dass die Fehlerkonstante zum Stufenwert Y_i nur vom „zugehörigen“ φ_i, also nicht von φ_j für $j \neq i$ abhängt. Zum anderen legt sie die Empfehlung nah, die Störungsparameter φ_i nicht zu weit vom Standardwert $\varphi_i = 1$ zu wählen.

Proposition 3.45. *Sei $k \in \{0, 1\}$ sowie $s = m$ und werde eine RK-Methode wie in Proposition 3.44 betrachtet, jedoch mit $c_1, \ldots, c_s$ und F derart gewählt, dass $\mathcal{A}$ entsprechend Satz 3.42 einen $s - k$-fachen Eigenwert $\gamma = 1$ besitzt. Dann hängt die Stabilitätsfunktion R_{i+k} der Stufe $i + k$ für $i \in \{1, \ldots, s - k\}$ nicht von (c_j, φ_j) für $j \neq i + k$, sondern nur von (c_{i+k}, φ_{i+k}) ab.*

Mittels dieses Ergebnis ergibt sich eine immense Erleichterung hinsichtlich der Stabilitätsuntersuchungen zu einer zu konstruierenden Methode mit $m = p + 1$. Denn für Stufe $i + k$ steht ausschließlich das Zwei-Tupel $(c_{i+k}, \varphi_{i+k}) \in \Xi_{k,m}$ als freier Parameter zur Verfügung, um die Ausprägung von R_{i+k} und damit das Stabilitätsverhalten der Stufe $i + k$ festzulegen.

Bemerkung 3.46. Zwar hängt R_{i+k} nur von $(c_{i+k}, \varphi_{i+k}) \in \Xi_{k,m}$ ab, dies gilt jedoch nicht für $\mathcal{A}_{(:,i+k)}$. Das heißt, die Elemente von $\mathcal{A}$ werden in Betrag und Vorzeichen von allen Zwei-Tupeln $(c_j, \varphi_j) \in \Xi_{k,m}$ beeinflusst.

Proposition 3.47. *Seien die Voraussetzungen von Proposition 3.45 gegeben. Dann besitzt der $s-k$-fache Eigenwert $\gamma = 1$ von $\mathcal{A}$ eine geometrische Vielfachheit von Eins.*

Beweis. Die Aussage folgt sofort aus dem Beweis von Proposition 3.45 in /STE 16/ und der Abhängigkeit des Fehlers (3.82) von φ_i. □

Unter den Voraussetzungen von obiger Proposition besteht somit eine Jordansche Normalform von $\mathcal{A}$ aus einem einzelnen Jordankästchen. Folglich können Blockstufen nicht getrennt voneinander behandelt werden. Wie in Unterabschnitt 3.3.4.1 diskutiert wird, kann es aber vermieden werden, Systeme der Größe $m \cdot n$ lösen zu müssen.

3.3.4 Transformationen und Berechnungsdarstellungen

Die bisherigen Ergebnisse und Erörterungen haben sich in der Darstellung hauptsächlich der „Steigungen“ k_i bedient, da Runge-Kutta-Methoden als verallgemeinerte Quadraturformeln betrachtet werden können und somit die Einträge von $\mathcal{A}$ Wichtungen für die k_i darstellen, s. (3.7). Für die praktische Berechnung ist es jedoch sinnvoll, auf die Verschübe z_i aus (3.65) zurückzugreifen, da somit direkt die für die Fortführung der Integration benötigten Größen zur Verfügung stehen. Wie bereits bemerkt, s. (3.9) sowie (3.66), gilt für die implizit bestimmten Größen Z aus (3.65) und K aus (3.62) die Beziehung

$$Z = (\mathcal{A} \otimes I)K. \tag{3.83}$$

Vernachlässigt man $\underline{\widetilde{K}}$ in(3.60) und bedient sich statt (3.61) der an dieser Stelle vorteilhafteren Darstellung

$$\mathcal{F}(\zeta) = \Big(\mathcal{F}_i(\zeta_i)\Big)_{i=1,\ldots,s}, \;\; \mathcal{F}_i(\zeta_i) := f(y_0+\zeta_i), \;\; \zeta_i \in \mathbb{R}^n, \;\; \zeta := (\zeta_1^T, \ldots, \zeta_s^T)^T, \tag{3.84}$$

so lässt sich mit $A = \mathcal{A}$ der Newtonschritt (3.60) schreiben als

$$(I - h\mathcal{A} \otimes J)\Delta K^l = -K^l + h\mathcal{F}\Big((\mathcal{A} \otimes I)K^l\Big). \tag{3.85}$$

Über den Zusammenhang (3.83) lässt sich das implizite System in (3.7) für die k_i, also für K, auch in Z formulieren

$$Z - h(\mathcal{A} \otimes I)\mathcal{F}(Z) = 0. \tag{3.86}$$

Ganz analog zu (3.85) ergibt sich ein Newtonschritt zu

$$(I - h\mathcal{A} \otimes J)\Delta Z^l = -Z^l + h(\mathcal{A} \otimes I)\mathcal{F}(Z^l). \tag{3.87}$$

Bemerkung 3.48. Mit (3.87) ist die Grundform der Iterationsvorschrift zur Bestimmung der Verschübe zu den Stufenwerten für Block- und Diagonalstufen gegeben. Alle folgenden Darstellungen sind Abwandlungen und Anpassungen hiervon. Die Darstellung (3.87) ist auch hinsichtlich numerischer Stabilität vorteilhaft: Nach /SHA 93/ sind die involvierten Matrizen in der Regel schlecht konditioniert. Bei einem Lösen der zugehörigen linearen Gleichungssysteme kann also nur davon ausgegangen werden, dass die führenden Ziffern der Lösungsvariablen korrekt berechnet werden. In einer Formulierung, in der *Inkremente* ΔZ statt direkte Werte Z oder Y ermittelt werden, sind auch unter diesen Bedingungen bestmöglich die relevanten Informationen erfasst.

Mit einer konsistenten Startnäherung, i. e.,

$$Z^0 = (\mathcal{A} \otimes I)K^0 \tag{3.88}$$

folgt mittels eines einfachen Induktionsargument aus (3.85) und (3.87) die Gültigkeit der Beziehungen

$$\Delta Z^l = (\mathcal{A} \otimes I)\Delta K^l \qquad \text{und somit} \qquad Z^l = (\mathcal{A} \otimes I)K^l \qquad \text{für} \quad l = 0, 1, 2, \dots \ . \tag{3.89}$$

Obige Aussagen geben einen ersten Einblick in den Zusammenhang zwischen iterierten K- und Z-Größen, beziehen sich aber auf das simultane Iterieren *aller* Stufen. Da Matrizen $\mathcal{A}$ der Form (3.71) betrachtet werden, ist es sinnvoll, im Detail zwischen den Block- und den Diagonalstufen zu unterscheiden. Hieran anschließend folgt eine Diskussion bezüglich der Bestimmung von y_1 als Approximation zu $y(t_0 + h)$ nebst Schätzung des Fehlers.

3.3.4.1 Blockstufen

Zur Betrachtung von Blockstrukturen sei angenommen, dass nach (3.71) für k-explizites $\mathcal{A}$ mit $k \in \{0, 1\}$ ein Block A_m der Größe $m \geqslant 2 + k$ existiert, da ansonsten wie für Diagonalstufen argumentiert werden kann, s. Unterabschnitt 3.3.4.2.
Existiert ein nichttrivialer Block A_m, müssen die zugehörigen Stufen parallel iteriert werden. Somit ergibt sich für

$$Z_m := \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} \qquad \text{und} \qquad K_m := \begin{pmatrix} k_1 \\ \vdots \\ k_m \end{pmatrix} \tag{3.90}$$

sofort (3.89) unter Berücksichtigung von (3.88) und mit $\mathcal{A}$ ersetzt durch A_m, i. e.

$$Z_m^l = (\mathcal{A} \otimes I)K_m^l \qquad \text{für} \quad l = 0, 1, 2, \ldots . \tag{3.91}$$

Beschränkt man ebenfalls $\mathcal{F}$ aus (3.84) auf die relevanten Blockgrößen, i. e.

$$\mathcal{F}_{1:m}(\cdot) := \Big(\mathcal{F}_i(\cdot)\Big)_{i=1,\ldots,m},$$

lässt sich die Iterationsvorschrift (3.87) bezogen auf die Blockstufen schreiben als

$$(I - hA_m \otimes J)\Delta Z_m^l = -Z_m^l + h(A_m \otimes I)\mathcal{F}_{1:m}(Z_m^l). \tag{3.92}$$

W-Transformation und grundsätzliche Berechnungsvorschrift

In der Form (3.92) ist ein Gleichungssystem der Größe $m \cdot n$ pro Newtonschritt zu lösen. Der Aufwand lässt sich reduzieren auf m Newtonsysteme der Größe n nebst einigen Operationen in $\mathcal{O}(n)$, was dem zu erwartenden Aufwand für m Stufen entspricht. Der grundsätzliche Ansatz wird bereits in /HAI 96, § IV.8/ diskutiert und hier kurz vorgestellt, um die konkreten Rechenvorschriften zur Bestimmung der notwendigen $FiterRK$-Größen herzuleiten.

Sei zunächst $\mathcal{A}$ 0-explizit, somit A_m regulär. Dann existieren ein reguläres $T \in \mathbb{R}^{m\times m}$ und eine reguläre untere linke Dreiecksmatrix $\Lambda \in \mathbb{R}^{m\times m}$, so dass

$$A_m^{-1} = T\Lambda T^{-1} \tag{3.93}$$

gilt, s. hierzu auch den Unterabschnitt *Wahl von* T weiter unten. Sei vermittels T eine neue Variable W^0 über

$$W^0 := (T^{-1} \otimes I)Z_m^0 \tag{3.94}$$

definiert. Ein einfaches Induktionsargument liefert mittels Linksmultiplikation von (3.92) mit $(hA_m)^{-1} \otimes I$ und unter Zuhilfenahme von (3.93) die Zusammenhänge

$$\Delta W^l = (T^{-1} \otimes I)\Delta Z_m^l \qquad \text{sowie} \qquad W^l = (T^{-1} \otimes I)Z_m^l \tag{3.95}$$

mit der Berechnungsvorschrift

$$\begin{aligned}(h^{-1}\Lambda \otimes I - I \otimes J)\Delta W^l = &- h^{-1}(\Lambda \otimes I)W^l \\ &+ (T^{-1} \otimes I)\mathcal{F}_{1:m}\Big((T \otimes I)W^l\Big)\end{aligned} \tag{3.96}$$

für $l = 0, 1, 2, \ldots$. Die Einführung der W-Variablen birgt den Vorteil, dass in dem Newtonschritt (3.96) nur noch Λ als Einfluss der Methode in der Iterationsmatrix auftritt. Da Λ eine untere linke Dreiecksmatrix ist, lassen sich somit die m Komponentenvektoren

von ΔW_m^l *sukzessive* über (3.96) bestimmen. Die hierbei auftretenden m linearen Gleichungssysteme der Dimension n greifen in kanonischer Reihenfolge auf die Matrizen

$$(h^{-1}\Lambda_{(i,i)} - J), \qquad i = 1, \ldots, m \tag{3.97}$$

zu. Im Falle eines Einpunktspektrums von A_m gilt wegen (3.93) die Beziehung $\Lambda_{(1,1)} = \cdots = \Lambda_{(m,m)}$; es muss also nur eine einzige Matrix der Form (3.97) berechnet und ggf. zerlegt werden. Dies führt zu enormen Ersparnissen im Rechenaufwand, s. hierzu auch die Diskussion in Abschnitt 3.3.3.

Die Darstellung (3.97) bietet den weiteren Vorteil, dass J nur im (trivialen) Kroneckerprodukt mit der Einheitsmatrix auftritt. Im nicht-transformierten System (3.92) müsste bei jeder Änderung von h oder J das Produkt $hA_m \otimes J$ bestimmt werden.

Berücksichtigung einer expliziten ersten Stufe im Block

In diesem Falle ist $m > 2$ vorausgesetzt und A_m lässt sich schreiben als

$$A_m = \begin{pmatrix} 0 & 0^T \\ a_1 & A_u \end{pmatrix} \qquad \text{mit} \qquad a_1 = \mathcal{A}_{(2:m,1)}, \quad A_u = \mathcal{A}_{(2:m,2:m)}.$$

Für jede konsistente Startnäherung Z_m^0, s. (3.91) mit $l = 0$, gilt $z_1^0 = 0$. Dies ist bereits der exakte Wert, und die Iteration (3.92) ändert nichts daran. Insofern muss diese Komponente keine Berücksichtigung in einer konkreten Berechnung finden. Mit einer Beschränkung auf die Komponentenvektoren 2 bis m ergibt sich aus (3.92) die Vorschrift

$$\begin{aligned}(I - hA_u \otimes J)\Delta Z_{2:m}^l &= -Z_{2:m}^l + h\Big((a_1 \;\; A_u) \otimes I\Big)\mathcal{F}_{1:m}(Z_m^l) \\ &= -Z_{2:m}^l + h(A_u \otimes I)\mathcal{F}_{2:m}(Z_{2:m}^l) + ha_1 \otimes f(y_0),\end{aligned} \tag{3.98}$$

da $\mathcal{F}_1 = f(y_0)$ gilt. Die Beziehungen (3.89) bleiben bestehen, jedoch ist nun für die Transformation von $Z_{2:m}$ auf W-Variablen die Matrix A_u heranzuziehen. Seien die angepassten Größen mit dem Index u versehen, dann ergibt sich analog zu (3.96)

$$\begin{aligned}(h^{-1}\Lambda_u \otimes I - I \otimes J)\Delta W_u^l = &- h^{-1}(\Lambda_u \otimes I)W_u^l \\ &+ (T_u^{-1} \otimes I)\mathcal{F}_{2:m}\Big((T_u \otimes I)W_u^l\Big) \\ &+ \Lambda_u T_u^{-1} a_1 \otimes f(y_0).\end{aligned} \tag{3.99}$$

Wahl von T oder ggf. T_u

Die Motivation für die Einführung eines Blockes liegt in dem Wunsch, $\mathcal{C}(p)$ für $p > 1 + k$ bei k-explizitem $\mathcal{A}$, $k \in \{0, 1\}$, zu erfüllen. Die Beziehung (3.97) zeigt, dass ein solcher Block nur dann effizient genutzt werden kann, wenn ein Einpunktspektrum vorliegt. Sei dies der Fall für die Diskussion hinsichtlich der Wahl von T oder T_u.

Für $m = p$ und $k = 0$ besitzt die Matrix A_m nach /BUT 98, Th. 21/ eine Jordansche Normalform, welche bidiagonal ist. Der m-fache Eigenwert γ von A_m weist also eine geometrische Vielfachheit von Eins auf. Dies gilt ebenso für A_m^{-1} und wegen der Ähnlichkeitstransformation (3.93) auch für Λ. Damit kann Lambda nicht in zwei oder mehrere Subblöcke zerfallen, und die Berechnung muss sukzessive erfolgen. Anders ausgedrückt: Keine Wahl von T ermöglicht eine (teil-)parallele Bestimmung der Komponentenvektoren von ΔW^l. Ein ganz analoges Resultat gilt für $k = 1$ aufgrund von /BUT 98, Th. 23/. Ist $m = p + 1$, so ist wegen Proposition 3.47 ebenfalls keine Entflechtung möglich.
Dies motiviert die Entscheidung, T oder ggf. T_u so „rechenfreundlich“ wie möglich zu konstruieren, da hiermit multipliziert oder Gleichungssysteme gelöst werden, s. (3.96) und (3.99). Das wesentliche Qualitätskriterium ist also die Kondition der Transformationsmatrix.

Bemerkung 3.49. Man beachte, dass selbst bei exakt vorliegenden Transformationsmatrizen und deren Inversen, die Kondition nicht außer Acht gelassen werden darf. Denn die auftretenden rechten Seiten in den entsprechenden linearen Gleichungssystemen werden mindestens Rundungsfehler ob der endlichen Rechengenauigkeit aufweisen. Und nach /MEI 11, Satz 2.41/ können diese ins Ergebnis aufgrund einer schlechten Kondition von T bzw. T_u verstärkt eingehen.

Entsprechend /STO 05, Satz 6.4.1/ existiert eine *orthogonale* Matrix T, so dass Λ aus (3.93) die gewünschte Dreiecksgestalt besitzt. Λ heißt in diesem Fall Schursche Normalform von A_m^{-1}. In einem gewissen Sinne kann keine bessere Wahl als ein T solchen Typs gefunden werden. Denn für eine beliebige induzierte Matrixnorm $\|\cdot\|$ auf $\mathbb{R}^{\nu\times\nu}$ und jede reguläre Matix $\mathcal{M} \in \mathbb{R}^{\nu\times\nu}$ mit $\nu \in \mathbb{N}$ gilt nach /MEI 11, Lem. 2.39/

$$\operatorname{cond}(\mathcal{M}) \geqslant 1. \tag{3.100}$$

Wird $\|\cdot\|$ zur Euklidischen Norm, i. e. $\|\cdot\|_2$, gewählt, so ist für jede orthogonale Matrix $\mathcal{M}$ die Abschätzung (3.100) mit Gleichheit erfüllt, und es gilt $\mathcal{M}^{-1} = \mathcal{M}^T$. Somit bietet es sich an, dass zur Bestimmung konkreter Methoden, s. Abschnitt 3.4, die Matrizen T und Λ aus (3.93) über die Konstruktion der Schurchen Normalform von A_m^{-1} bestimmt werden. Analoges gilt für T_u und Λ_u.

Bemerkung 3.50. Die in /BUT 98/ aufgeführten Transformationsmatrizen sind über den dort diskutierten algebraischen Hintergrund konstruiert und motiviert. Deren Kondition ist auch nach entsprechender Recherche nicht direkt benennbar. Es erscheint nicht lohnenswert, das Thema weiter zu verfolgen, da im Sinne der Kondition (und bezüglich

der Euklidischen Norm) bereits wie oben angegeben eine optimale Wahl zur Verfügung steht.

Berücksichtigung nicht-autonomer ODEs im Block

Liegt ein nicht-autonomes System vor und wurde dies über Hinzunahme von $t' = 1$ autonomisiert, so kann unter Ausnutzung einer Jacobi-Struktur der Form

$$\begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & J \end{pmatrix} \tag{3.101}$$

analog zu den klassischen W-Methoden vorgegangen werden, vgl. (3.17b). In diesem Falle stehen mit den Startnäherungen τ_i^0 für $c_i h$, $i = 1, \dots, m$, nach einem Schritt der Iteration (3.92), wobei dort J ersetzt ist durch (3.101), die Inkremente

$$\Delta\tau_i^0 = -\tau_i^0 + h \sum_{j=1}^{m} a_{ij} \overset{C(1)}{=} -\tau_i^0 + c_i h \tag{3.102}$$

zur Verfügung. Mit $\tau_i^1 = \tau_i^0 + \Delta\tau_i^0$, $i = 1, \dots, m$, liegen also die exakten Lösungen $\tau_i^1 = c_i h$ für den expliziten Zeitanteil vor. Hinsichtlich des y- bzw. Z-Anteils ergibt eine direkte Berücksichtigung der Struktur (3.101) im ersten Schritt, i. e. $l = 0$, und für 0-explizites $\mathcal{A}$ die angepasste Iterationsvorschrift

$$\begin{aligned} (I - hA_m \otimes J)\Delta Z_m^0 = &- Z_m^0 + h(A_m \otimes I)\mathcal{F}_{1:m}(\tau^0, Z_m^0) \\ &+ h(A_m \otimes \partial f_0/\partial t)\Delta\tau^0 \end{aligned} \tag{3.103}$$

mit

$$\begin{aligned} &\mathcal{F}_{1:m}(\tau, \zeta) = \Big(\mathcal{F}_i(\tau_i, \zeta_i)\Big)_{i=1,\dots,m}, \quad \mathcal{F}_i(\tau_i, \zeta_i) := f(t_0 + \tau_i, y_0 + \zeta_i), \\ &\tau_i \in \mathbb{R}, \quad \tau := (\tau_1, \dots, \tau_m)^T, \quad \zeta_i \in \mathbb{R}^n, \quad \zeta := (\zeta_1^T, \dots, \zeta_s^T)^T. \end{aligned} \tag{3.104}$$

sowie $\Delta\tau^0 = (\Delta\tau_1^0, \dots, \Delta\tau_m^0)^T$. Da die Lösungen $\tau_i = c_i h$ von vornherein bekannt sind, kann man auch direkt $\tau_i^0 = c_i h$ nutzen. Dies führt auf ein Inkrementvektor $\Delta\tau^0 = 0$ und somit zu

$$(I - hA_m \otimes J)\Delta Z_m^0 = -Z_m^0 + h(A_m \otimes I)\mathcal{F}_{1:m}(h(c_1, \dots, c_m)^T, Z_m^0). \tag{3.105}$$

Entsprechend des bisher Diskutierten gilt diese Vorschrift analog auch für $l > 0$, unabhängig davon, wie τ_i^0, $i = 1, \dots, m$, gewählt sind. Wird eine Methode mit 1-explizitem $\mathcal{A}$ betrachtet, so ist stets $\tau_1^0 = 0$. Ausnutzen hiervon in (3.98) und Berücksichtigung von (3.104) liefert

$$\begin{aligned} (I - hA_u \otimes J)\Delta Z_{2:m}^0 = &- Z_{2:m}^0 + h(A_u \otimes I)\mathcal{F}_{2:m}(\tau_{(2:m)}^0, Z_{2:m}^0) \\ &+ ha_1 \otimes f(t_0, y_0) + h(A_u \otimes \partial f_0/\partial t)\Delta\tau_{(2:m)}^0, \end{aligned} \tag{3.106}$$

und es gilt (3.102) für $i = 2, \ldots, m$. Analog ergibt sich

$$\begin{aligned}(I - hA_u \otimes J)\Delta Z^0_{2:m} = &- Z^0_{2:m} + h(A_u \otimes I)\mathcal{F}_{2:m}(h(c_2, \ldots, c_m)^T, Z^0_{2:m}) \\ &+ ha_1 \otimes f(t_0, y_0)\end{aligned} \tag{3.107}$$

für $\tau^0_i = c_i h$ oder für weitere Iterationsschritte mit $l > 0$ anstelle des Iterationsindex 0.

Die Transformation zu W-Variablen erfolgt wie in (3.96) oder (3.99), wobei im Falle von (3.103) oder (3.106) der zusätzliche Term

$$(T^{-1} \otimes \partial f_0/\partial t)\Delta\tau^0 \qquad \text{bzw.} \qquad (T_u^{-1} \otimes \partial f_0/\partial t)\Delta\tau^0_{(2:m)} \tag{3.108}$$

zu berücksichtigen ist.

Obige Ausführungen zeigen zwei Strategien auf, wie im Falle einer nicht-autonomen Differentialgleichung mit dem expliziten Zeitanteil umgegangen werden kann:

- Setze $\tau^0_i = c_i h$ und nutze für alle Iterationsschritte (3.105) bzw. (3.107).
- Setze τ^0_i beliebig und nutze für den ersten Iterationsschritt (3.103) bzw. (3.106) und für die weiteren Iterationsschritte (3.105) bzw. (3.107).

Vorteile der Strategie $\tau^0_i = c_i h$

- Der Ansatz entspricht der Faustregel „Nutze sämtliche A-priori-Informationen über die Lösung“. Dies fördert die Hoffnung, mit wenig Iterationsschritten gute Approximationen zu erhalten. Im Falle einer linearen Differentialgleichung bei Vorlage exakter Jacobimatrix-Informationen reicht ein einziger Schritt aus, so dass Z^1_m mit Z_m aus(3.90) übereinstimmt.
- Falls eine Stufenordnung von $p = 2$ gefordert wird und ein T2-Update in Verbindung mit einer Startiterierten $Z^0_m = 0$ genutzt wird, reicht im $FiterRK-$Kontext ebenfalls ein Newtonschritt aus, um die Stufenordnung im Block sicherzustellen.
- Die partielle Ableitung $\partial f_0/\partial t$ muss nicht bestimmt werden.

Vorteile der Strategie τ^0_i beliebig

- Für den Spezialfall $\tau^0_i = 0$ in Verbindung mit $Z^0_m = 0$ bedarf es im ersten Newtonschritt lediglich einer einzigen f-Auswertung, i. e., $f(t_0, y_0)$. Dieser Vektor kann als wohldefiniert angesehen werden und steht im ATHLET-Kontext bereits zur Verfügung.

Abhängig von den Kosten zur Auswertung von f und des evtl. merklich eingeschränkten Definitionsbereiches von f mag der zweite Ansatz bevorzugt werden. Dies ist in der Regel im ATHLET-spezifischen Rahmen der Fall, s. zur Konsequenz hieraus auch Bemerkung 3.71. Als Standardoption im allgemeinen Fall wird der erste Ansatz, i. e. $\tau^0_i = c_i h$, empfohlen.

Bereitstellung konsistenter Startnäherungen im Block

Da in der obigen Passage der explizite Zeitanteil auch im Sinne der Startapproximationen τ_i^0 bereits diskutiert wurde, kann sich hier auf ein autonomes bzw. auf den y-Anteil eines nicht-autonomen Systems beschränkt werden.

Entsprechend Bemerkung 3.28 kann eine konsistente Startnäherung Z_m^0 über (3.67) konstruiert werden, wenn Startapproximationen Y_i^0 für die Stufenwerte vorliegen. Eine implizite RK-Methode besitzt den Vorteil, dass eine Ordnungsbedingung der Form (3.68) nicht zwingend erforderlich ist, solange der zugehörige Newtonprozess (hinreichend rasch) konvergiert. Im Falle einer $FiterRK$-Methode ist die besagte Ordnungsbedingung jedoch von großer Bedeutung, da nach Satz 3.26 hierüber direkt die Anzahl an notwendigen Newtonschritten bestimmt werden kann. Im Rahmen einer Einschrittmethode mit adaptiver Schrittweitenkontrolle ist es schwierig, eine Bedingung wie (3.68) mit der *aktuellen* Schrittweite zu erfüllen. Eine Möglichkeit hierfür wäre, eine Methode geringerer Ordnung zu bemühen. Dieses Vorgehen verlagert das Problem, da auch diese Methode in der Regel einer Startnäherung bedarf. Da sich im $FiterRK$-Umfeld sukzessive die Ordnung der berechneten Approximation pro Newtonschritt um Eins verbessert (bis zur Ordnung der zugrundeliegenden impliziten RK-Methode), kann der Newtonprozess selbst als Ausführung einer Folge von Methoden aufsteigender Ordnung gesehen werden. Diese Interpretation kann als Rechtfertigung dienen, mit einer Nullapproximation zu starten, i. e. $Z_m^0 = 0$. Konsistenz ist damit im Sinne von (3.68) für $r = 0$ sichergestellt. Gelten zusätzlich die Bedingungen von Satz 3.36 für die Blockstufen, so führt dies zu p Iterationschritten bei einer Stufenordnung von p. Legt man Wert auf eine Reduzierung der Anzahl an f-Auswertungen, so kann es durchaus sinnvoll sein, eine andere Methode kleinerer Ordnung für die Generierung von Startapproximationen zu nutzen. Dies wird für die Methoden in Abschnitt 3.6.2 ausgenutzt.

Es kann davon ausgegangen werden, dass zu Anfang eines Schrittes $f(y_0)$ vorliegt. Dieses kann genutzt werden, um über einen expliziten Eulerschritt, i.e .

$$z_i^0 = c_i h f(y_0) \quad \text{für} \quad i = 1 + k, \ldots, m, \quad \text{bei } k\text{-explizitem } \mathcal{A} \text{ mit } k \in \{0, 1\} \quad (3.109)$$

Startapproximationen der Ordnung Eins zur Verfügung zu stellen. Dies mag im Kontext einer 1-expliziten Koeffizientenmatrix $\mathcal{A}$ nicht als unadäquat angesehen werden, da bei Verwendung zugehöriger Methoden bewusst die Entscheidung getroffen wird, explizit $f(y_0)$ zu nutzen. Schwierigkeiten können sich bei den Auswertungen $f\Big(y_0 + c_i h f(y_0)\Big)$ ergeben, da bei entsprechender Steifheit ein expliziter Eulerschritt weit über das Ziel hinausschießen kann. Der anschließende Newtonschritt kompensiert dieses Verhalten.

Jedoch ist dieser nur anwendbar, wenn die f-Auswertungen wohl definiert bleiben. Bei sensitivem Definitionsbereich ist somit von der Einbindung der Ableitung $f(y_0)$ abzusehen und eine Methode mit 0-explizitem $\mathcal{A}$ zu bevorzugen.

3.3.4.2 Diagonalstufen

Diagonalstufen zeichnen sich durch zwei Besonderheiten aus:

- Die Implizitheit der Stufe ist nur auf sich selbst bezogen. Die Größen k_i bzw. z_i für $i > m$ hängen nur explizit von vorherigen Stufen ab. Nachfolgende Stufen haben keine Relevanz, s. (3.111) oder (3.113).
- Gilt $i > 1$, können aus vorherigen Stufen sehr einfach Startapproximationen k_i^0 respektive z_i^0 konstruiert werden, s. (3.127) oder (3.128), wodurch die notwendige Anzahl an Iterationen im $FiterRK$-Kontext mitunter stark reduziert werden kann, s. Satz 3.32 sowie Korollar 3.33.

Die Darstellung der nachfolgenden Rechenvorschriften geht von der Annahme aus, dass ein nichttrivialer Block vorliegt, i. e., $m \geqslant 2 + k$ für k-explizites $\mathcal{A}$, $k \in \{0, 1\}$, gilt. Ist nur ein trivialer Block gegeben, so wird die Blockstufe $1 + k$ wie eine Diagonalstufe behandelt und über die hier aufgeführten Rechenvorschriften abgedeckt. Blockbezogene Größen wie T oder W werden dann als leere Entitäten interpretiert. Zusätzlich bedarf es einer Anpassung der Dimensionierung:

$$\bar{m} := \begin{cases} m & \text{für} \quad m \geqslant 2 + k \quad \text{(nichttrivialer Block)} \\ m - 1 & \text{für} \quad m = 1 + k \quad \text{(trivialer Block).} \end{cases} \tag{3.110}$$

Zusammenhang zwischen k_i und z_i

Sei Stufe i mit $i \in \{\bar{m} + 1, \ldots, s\}$ betrachtet. Nach (3.59) lässt sich die Gleichung zur Bestimmung von k_i schreiben als

$$k_i = hf\Big(y_0 + \sum_{j=1}^{i-1} a_{ij} k_j + a_{ii} k_i\Big). \tag{3.111}$$

Werden k_j für $j = 1, \ldots, i-1$, durch die finalen Approximationen $k_j^{l_j}$ ersetzt, ergibt sich der entsprechende Newtonschritt (3.60) zu

$$(I - h a_{ii} J)\Delta k_i^l = -k_i^l + hf\Big(y_0 + \sum_{j=1}^{i-1} a_{ij} k_j^{l_j} + a_{ii} k_i^l\Big). \tag{3.112}$$

Mit $z_i = Y_i - y_0$ folgt aus (3.7) die Bestimmungsgleichung

$$z_i = \sum_{j=1}^{i-1} a_{ij} k_j + a_{ii} hf(y_0 + z_i) \tag{3.113}$$

Wiederum die finalen Approximationen der vorherigen Stufen ausnutzend ergibt sich

$$(I - ha_{ii}J)\Delta z_i^l = -z_i^l + \sum_{j=1}^{i-1} a_{ij}k_j^{l_j} + a_{ii}hf\Big(y_0 + z_i^l\Big). \tag{3.114}$$

Mit einer konsistenten Startnäherung

$$z_i^0 = \sum_{j=1}^{i-1} a_{ij}k_j^{l_j} + a_{ii}k_i^0 \tag{3.115}$$

lässt sich der erste Iterationsschritt schreiben als

$$(I - ha_{ii}J)\Delta z_i^0 = -a_{ii}k_i^0 + a_{ii}hf\Big(y_0 + z_i^0\Big). \tag{3.116}$$

Vergleich mit (3.112) für $l = 0$ zeigt

$$\Delta z_i^0 = a_{ii}\Delta k_i^0,$$

und ein Induktionsargument liefert

$$\Delta z_i^l = a_{ii}\Delta k_i^l \tag{3.117a}$$

und hiermit

$$(I - ha_{ii}J)\Delta z_i^l = -a_{ii}k_i^l + a_{ii}hf\Big(y_0 + z_i^l\Big), \qquad l = 0, 1, \dots . \tag{3.117b}$$

Diese Beziehungen ausnutzend führt ein weiteres Induktionsargument hinsichtlich des Stufenindex zu

$$\begin{pmatrix} z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_s^{l_s} \end{pmatrix} = (\mathcal{A}_{(\bar{m}+1:s,1:s)} \otimes I) = \begin{pmatrix} k_1^{l_1} \\ \vdots \\ k_s^{l_s} \end{pmatrix}, \tag{3.118}$$

und zusammen mit (3.91) ergibt sich eine Verfeinerung von (3.89) vermittels

$$\begin{pmatrix} z_1^{l_1} \\ \vdots \\ z_s^{l_s} \end{pmatrix} = (\mathcal{A} \otimes I) \begin{pmatrix} k_1^{l_1} \\ \vdots \\ k_s^{l_s} \end{pmatrix}, \tag{3.119}$$

wobei $l_1 = \cdots = l_m$ die Anzahl der Iterationen für die Blockstufen widerspiegelt. Im übernächsten Unterabschnitt wird dieser Zusammenhang zur Konstruktion von k_i^0 und z_i^0 genutzt.

Bemerkung 3.51. Im Falle einer expliziten ersten Stufe, i. e. $\mathcal{A}$ ist 1-explizit, kann $k_1^{l_1}$ in der Darstellung (3.119) durch $k_1^0 = f(y_0)$ und $z_1^{l_1}$ durch $z_1^0 = 0$ ersetzt werden, da dies bereits die bekannten exakten Werte sind und die Newtonvorschrift (3.87) diese Werte invariant lässt.

Berechnungsvorschrift zu einer Diagonalstufe

Der Newtonschritt (3.117b) lässt sich äquivalent schreiben als

$$(h^{-1}a_{ii}^{-1}I - J)\Delta z_i^l = -h^{-1}k_i^l + f(y_0 + z_i^l). \tag{3.120}$$

Somit wird wieder die Multiplikation von J mit nichttrivialen Faktoren eingespart, vgl. (3.97). Ein weiterer wesentlicher Vorteil tritt im Zusammenhang mit einem Einpunktspektrum von $\mathcal{A}$ auf, da dann wegen der Transformation (3.93) der Zusammenhang

$$\Lambda_{1,1} = \cdots = \Lambda_{m,m} = a_{m+1,m+1}^{-1} = \cdots = a_{s,s}^{-1}.$$

gilt. Es wird also nur eine Iterationsmatrix für sämtliche Stufenberechnungen benötigt. Analoges ergibt sich im Falle einer Methode mit expliziter erster Stufe aus (3.99) und $A_u^{-1} = T_u\Lambda_u T_u^{-1}$.

Für die Vorschrift (3.120) muss k_i^l zur Verfügung stehen. Mit der Berechnung von Δz_i^l geschieht das Aufdatieren mit Hilfe von (3.117a). Damit über (3.120) eine Newtoniteration ausgeführt werden kann, bedarf es Startnäherungen z_i^0 und k_i^0.

Bestimmung von k_i^0 und z_i^0

Die in Abschnitt 3.2.1 für W-Methoden eingeführten Matrizen A und Γ lassen sich für $FiterRK$-Methoden mit einem $\mathcal{A}$ der Form (3.71) in ihrer Definition formal erweitern zu

$$A = (\alpha_{ij})_{i,j=1,\ldots,s} \qquad \text{mit} \qquad \alpha_{ij} = 0 \quad \text{für} \quad \begin{cases} i = 1,\ldots,\bar{m} \quad \text{und} \quad j = 1,\ldots,s \\ i = \bar{m}+1,\ldots,s \quad \text{und} \quad j = i,\ldots,s \end{cases}$$

und

$$\Gamma = (\gamma_{ij})_{i,j=1,\ldots,s} = \mathcal{A} - A.$$

Somit gilt wieder (3.16), also

$$\mathcal{A} = A + \Gamma. \tag{3.121}$$

Zur Bestimmung von k_i^0 und z_i^0 bedarf es einer Submatrix von A, i. e.,

$$\bar{A} := A_{(\bar{m}+1:s,1:s-1)} \tag{3.122}$$

und einer Matrix $\bar{\Gamma}$, welche sich aus Γ ergibt über

$$\bar{\Gamma} := \left(\Gamma_{(\bar{m}+1:s,1:s)} - \begin{pmatrix} O_{(s-\bar{m})\times\bar{m}} & D_\gamma \end{pmatrix}\right)_{(:,1:s-1)} \tag{3.123}$$

mit der Nullmatrix $O_{(s-\bar{m}\times\bar{m})}$ in $\mathbb{R}^{(s-\bar{m})\times\bar{m}}$ sowie

$$D_\gamma := \operatorname{diag}(\gamma_{\bar{m}+1,\bar{m}+1},\ldots,\gamma_{ss}) = \operatorname{diag}(a_{\bar{m}+1,\bar{m}+1},\ldots,a_{ss}). \tag{3.124}$$

- Sei $\mathcal{A}$ 0-explizit. Dann folgt mit

  $$\mathcal{A}_{s-1} := \mathcal{A}_{(1:s-1,1:s-1)}$$

  und (3.119) die Beziehung

  $$\begin{pmatrix} k_1^{l_1} \\ \vdots \\ k_{s-1}^{l_{s-1}} \end{pmatrix} = (\mathcal{A}_{s-1}^{-1} \otimes I) \begin{pmatrix} z_1^{l_1} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix}. \tag{3.125}$$

  Aus (3.121), den Startnäherungen (3.72a) und (3.115) ergibt sich

  $$\begin{pmatrix} z_{\bar{m}+1}^0 \\ \vdots \\ z_s^0 \end{pmatrix} = (\bar{A} \otimes I) \begin{pmatrix} k_1^{l_1} \\ \vdots \\ k_{s-1}^{l_{s-1}} \end{pmatrix} \tag{3.126}$$

  und mit (3.125) sowie (3.95) folgt die Darstellung

  $$\begin{aligned} \begin{pmatrix} z_{\bar{m}+1}^0 \\ \vdots \\ z_s^0 \end{pmatrix} &= (\bar{A} \otimes I)(\mathcal{A}_{s-1}^{-1} \otimes I) \begin{pmatrix} z_1^{l_1} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix} \\ &= (\bar{A} \otimes I)(\mathcal{A}_{s-1}^{-1} \otimes I) \begin{pmatrix} T \otimes I & \\ & I_\delta \end{pmatrix} \begin{pmatrix} W^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix}, \end{aligned} \tag{3.127}$$

  wobei l_b die Iterationszahl für die Blockstufen angibt und I_δ die Identität im $\mathbb{R}^{\delta \times \delta}$ mit $\delta = n \cdot (s - 1 - \bar{m})$ (im Gegensatz zur Identität I im $\mathbb{R}^{n \times n}$). Ganz analog ergibt sich für die k_i^0-Größen

  $$\begin{aligned} \begin{pmatrix} k_{\bar{m}+1}^0 \\ \vdots \\ k_s^0 \end{pmatrix} &\overset{(3.72a)}{=} -(D_\gamma^{-1} \otimes I)(\bar{\Gamma} \otimes I) \begin{pmatrix} k_1^{l_1} \\ \vdots \\ k_{s-1}^{l_{s-1}} \end{pmatrix} \\ &\overset{\substack{(3.125) \\ (3.95)}}{=} -(D_\gamma^{-1} \otimes I)(\bar{\Gamma} \otimes I)(\mathcal{A}_{s-1}^{-1} \otimes I) \begin{pmatrix} T \otimes I & \\ & I_\delta \end{pmatrix} \begin{pmatrix} W^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix}. \end{aligned} \tag{3.128}$$

- Sei $\mathcal{A}$ 1-explizit. In diesem Fall ist es entsprechend Bemerkung 3.51 sinnvoll, in (3.119) die z_1-Größe nicht explizit zu berücksichtigen. Vielmehr bietet es sich an, mit e_1, dem

ersten Einheitsvektor in $\mathbb{R}^s$, die Beziehung (3.119) umzuschreiben zu

$$\begin{pmatrix} k_1^0 \\ z_2^{l_2} \\ \vdots \\ z_s^{l_s} \end{pmatrix} = \left((\mathcal{A} + e_1 e_1^T) \otimes I \right) \begin{pmatrix} k_1^0 \\ k_2^{l_2} \\ \vdots \\ k_s^{l_s} \end{pmatrix}. \tag{3.129}$$

Die Matrix $\mathcal{A} + e_1 e_1^T$ ist regulär und ebenso

$$Æ_{s-1} := (\mathcal{A} + e_1 e_1^T)_{(1:s-1,1:s-1)}.$$

Somit ergibt sich ähnlich zu (3.125)

$$\begin{pmatrix} k_1^0 \\ k_2^{l_2} \\ \vdots \\ k_{s-1}^{l_{s-1}} \end{pmatrix} = (Æ_{s-1}^{-1} \otimes I) \begin{pmatrix} k_1^0 \\ z_2^{l_2} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix}. \tag{3.130}$$

Auf gleiche Weise argumentierend wie für (3.127) und (3.128) folgen hieraus die Berechnungsvorschriften

$$\begin{aligned} \begin{pmatrix} z_{\bar{m}+1}^0 \\ \vdots \\ z_s^0 \end{pmatrix} &= (\bar{A} \otimes I)(Æ_{s-1}^{-1} \otimes I) \begin{pmatrix} k_1^0 \\ z_2^{l_2} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix} \\ &= (\bar{A} \otimes I)(Æ_{s-1}^{-1} \otimes I) \begin{pmatrix} I & & \\ & T_u \otimes I & \\ & & I_\delta \end{pmatrix} \begin{pmatrix} k_1^0 \\ W_u^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix}, \end{aligned} \tag{3.131}$$

und

$$\begin{aligned} \begin{pmatrix} k_{\bar{m}+1}^0 \\ \vdots \\ k_s^0 \end{pmatrix} &\overset{(3.72a)}{=} -(D_\gamma^{-1} \otimes I)(\bar{\Gamma} \otimes I) \begin{pmatrix} k_1^{l_1} \\ \vdots \\ k_{s-1}^{l_{s-1}} \end{pmatrix} \\ &\overset{\substack{(3.125) \\ (3.95)}}{=} -(D_\gamma^{-1} \otimes I)(\bar{\Gamma} \otimes I)(Æ_{s-1}^{-1} \otimes I) \begin{pmatrix} I & & \\ & T_u \otimes I & \\ & & I_\delta \end{pmatrix} \begin{pmatrix} k_1^0 \\ W_u^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_{s-1}^{l_{s-1}} \end{pmatrix}. \end{aligned} \tag{3.132}$$

Bemerkung 3.52. Im Falle eines trivialen Blockes für 0-explizites $\mathcal{A}$ gilt $m = 1$, und es liegt somit eine DIRK-Methode[2] vor. Nach (3.17a) und (3.18) ist dies die zugeordnete RK-Methode zu einer klassischen W-Methode. Entsprechend deren Definition wird auch an dieser Stelle der triviale Block wie eine Diagonalstufe behandelt. Hierbei ist zu beachten, dass sich in diesem Fall ob der kompakten und einheitlichen Notation die erste Zeile von $\bar{A}$ und $\bar{\Gamma}$ zu einer Nullzeile ergibt. Dies ist konform zum Informationsstand für die erste Stufe, da es noch keine anderen Stufeninformationen gibt, auf die zurückgegriffen werden könnte. Es gilt also $z_1^0 = 0$ und $k_1^0 = 0$. Im Sinne einer konkreten Implementierung sollte dies jedoch direkt gesetzt und nicht über $\bar{A}$ und $\bar{\Gamma}$ bzw. deren nachfolgend aufgeführten abgeleiteten Größen (3.144) und (3.145) berechnet werden, vgl. Zeile 8 in Alg. 3.3.

Berücksichtigung nicht-autonomer ODEs in den Diagonalstufen

Entsteht das autonome System aus einem nicht-autonomen durch Hinzunahme von $t' = 1$, so ist im Gegensatz zu den Blockstufen die Art der expliziten Berücksichtigung hiervon bereits durch die konsistente Startnäherung (3.115) respektive (3.126) und durch die Konstruktion von k_i^0 in (3.72a) festgelegt. Mit einer Jacobi-Struktur der Form (3.101) ergibt sich aus (3.120) analog zu (3.17b) und für $l = 0$

$$(h^{-1}a_{ii}^{-1}I - J)\Delta z_i^0 = -h^{-1}k_i^0 + f(t_0 + \alpha_i h, y_0 + z_i^0) + h\gamma_i \partial f_0/\partial t. \tag{3.133}$$

Ist $\mathcal{C}_\alpha(q)$ aus (3.73) für $q \geqslant 1$ erfüllt, so gilt $\alpha_i = c_i$ und $\gamma_i = 0$. Der explizite Zeitanteil wird also exakt durch die Startnäherung wiedergegeben, und die partielle Zeitableitung braucht nicht berücksichtigt zu werden. Dies ist spätestens auch im Allgemeinen nach der ersten Iteration der Fall und es ergibt sich für $l > 0$ stets

$$(h^{-1}a_{ii}^{-1}I - J)\Delta z_i^l = -h^{-1}k_i^l + f(t_0 + c_i h, y_0 + z_i^l). \tag{3.134}$$

Alternativ zur obigen Vorgehensweise, auch wenn $\mathcal{C}_\alpha(1)$ nicht gilt, kann direkt mit dem exakten Wert $\tau_i = c_i h$ gearbeitet werden. Konsistente Startnäherungen (3.126) beziehen sich dann nur noch auf den y- bzw. z-Anteil der Lösungsvariablen. Eine entsprechende Auswahl für τ_i ist in Alg. 3.3 vermerkt.

Ermittlung von y_1 und Fehlerschätzung

Da y_1 für eine Runge-Kutta-Methode (3.7) oder auch für eine W-Methode (3.17a) die gleiche Struktur wie ein Stufenwert besitzt, vgl. (3.8) bzw. (3.19), kann sich wieder des Zusammenhanges (3.119) bzw. (3.129) bedient werden. Im Allgemeinen ergibt sich für

[2] Diagonally Implicit Runge-Kutta

0-explizites $\mathcal{A}$ somit

$$y_1 = y_0 + (b^T \otimes I)(\mathcal{A}^{-1} \otimes I) \begin{pmatrix} T \otimes I & \\ & I_{\delta+n} \end{pmatrix} \begin{pmatrix} W^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_s^{l_s} \end{pmatrix} \tag{3.135a}$$

und für 1-explizites $\mathcal{A}$

$$y_1 = y_0 + (b^T \otimes I)\Big((\mathcal{A} + e_1 e_1^T)^{-1} \otimes I\Big) \begin{pmatrix} I & & \\ & T_u \otimes I & \\ & & I_{\delta+n} \end{pmatrix} \begin{pmatrix} k_1^0 \\ W_u^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_s^{l_s} \end{pmatrix} \tag{3.135b}$$

mit $I_{\delta+n}$ der Identität in $\mathbb{R}^{(\delta+n)\times(\delta+n)}$. Eine lokale Fehlerschätzung err_{est}^{loc} basiert in der Regel auf der Differenz von y_1 und einem $\hat{y}_1$, welches mit $\hat{b}$ anstelle von b konstruiert wird und eine andere Konsistenzordnung als y_1 aufweist. Praktisch heißt dies, dass sich err_{est}^{loc} mit $\delta b = b - \hat{b}$ berechnet zu

$$err_{est}^{loc} = (\delta b^T \otimes I)(\mathcal{A}^{-1} \otimes I) \begin{pmatrix} T \otimes I & \\ & I_{\delta+n} \end{pmatrix} \begin{pmatrix} W^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_s^{l_s} \end{pmatrix} \tag{3.136a}$$

bzw.

$$err_{est}^{loc} = (\delta b^T \otimes I)\Big((\mathcal{A} + e_1 e_1^T)^{-1} \otimes I\Big) \begin{pmatrix} I & & \\ & T_u \otimes I & \\ & & I_{\delta+n} \end{pmatrix} \begin{pmatrix} k_1^0 \\ W_u^{l_b} \\ z_{\bar{m}+1}^{l_{\bar{m}+1}} \\ \vdots \\ z_s^{l_s} \end{pmatrix}. \tag{3.136b}$$

Einige wichtige Spezialfälle lassen sich deutlich effizienter bestimmen:

- $y_1 = Y_i$ für $i \in \{1, \ldots, s\}$. Dann gilt

  $$y_1 = y_0 + z_i^{l_i}. \tag{3.137}$$

  Für $i > \bar{m}$ ist $z_i^{l_i}$ direkt vorhanden. Dies gilt ebenfalls für den formell berücksichtigten Fall $y_1 = Y_1 = y_0$ bei 1-explizitem $\mathcal{A}$. Andernfalls muss $z_i^{l_i}$ mit $1 + k \leqslant i \leqslant \bar{m}$ über

  $$\begin{aligned} z_i^{l_i} &= (e_i^T \otimes I)(T \otimes I) W^{l_b} && \text{für} \quad \text{0-explizites } \mathcal{A} \\ z_i^{l_i} &= (e_{i-1}^T \otimes I)(T_u \otimes I) W_u^{l_b} && \text{für} \quad \text{1-explizites } \mathcal{A} \end{aligned} \tag{3.138}$$

  konstruiert werden.

- $y_1 = Y_i$ sowie $\hat{y}_1 = Y_j$ für $i, j \in \{1, \ldots, s\}$, $i \neq j$. Dann gilt (3.137) und

$$err_{est}^{loc} = z_i^{l_i} - z_j^{l_j}. \tag{3.139}$$

- $y_1 = Y_i$ sowie $\hat{y}_1 = Y_j$ für $i \in \{\bar{m}+1, \ldots, s\}$, $j \in \{1, \ldots, i-1\}$ und $z_i^0 = z_j^{l_j}$ sowie $l_i = 1$. Dann gilt (3.137) und

$$err_{est}^{loc} = \Delta z_i^0. \tag{3.140}$$

Im letzten Fall bekommt man die Fehlerschätzung geschenkt, da Δz_i^0 sowieso berechnet wird. Die konstruierten Methoden, welche in Abschnitt 3.4 diskutiert werden, machen von dieser sehr effizienten Vorgehensweise Gebrauch.

3.3.4.3 Implementierung von Rechenvorschriften

Die in den Unterabschnitten 3.3.4.1 und 3.3.4.2 hergeleiteten Rechenvorschriften basieren auf Kroneckerprodukten. Dies bietet sowohl den Vorteil einer üblichen Matrix-Vektor-Notation als auch den Vorteil, die einzelnen Transformationsschritte leicht nachvollziehen zu können. Im Sinne einer Implementierung ist eine Kroneckerprodukt-Darstellung jedoch unvorteilhaft.

Zur Herleitung einer passenderen Darstellung seien $v = (v_1^T, \ldots, v_\mu^T)^T$ für $v_i \in \mathbb{R}^\eta$, $i = 1, \ldots, \mu$, und $B \in \mathbb{R}^{\xi \times \mu}$ mit $\mu, \eta, \xi \in \mathbb{N}$ beliebig. Dann gilt

$$(B \otimes I_\eta)v = (B \otimes I_\eta) \begin{pmatrix} v_1 \\ \vdots \\ v_\mu \end{pmatrix} = \mathrm{vec}\left[(v_1, \ldots, v_\mu)B^T\right]. \tag{3.141}$$

Die vec-Operation ist nur für den Erhalt der Identität in (3.141) nötig, jedoch nicht für eine algorithmische Umsetzung. Das heißt, dass Kroneckerprodukte keine explizite Berücksichtigung finden müssen – sich des Argumentes der vec-Operation in (3.135b) zu bedienen, genügt. Dabei bietet sich zusätzlich an, dass all die Faktoren in den Rechenvorschriften, welche nur Methodenentitäten enthalten, im Vorfelde berechnet und zur späteren Nutzung abgelegt werden können. Die grundlegende Verwendung dieser vorberechneten Werte zur Bestimmung der für den Fortschritt der Integration benötigten Größen wird in den Algorithmen 3.2, 3.3 und Algorithmus 3.4 dargelegt.

Im Sinne einer Vereinheitlichung der Handhabung von Größen zu k-explizitem $\mathcal{A}$ mit $k \in \{0, 1\}$ seien die folgenden Methodenentitäten als bereits ermittelt festgelegt. Übliche Rechenregeln für Kroneckerprodukte finden hierbei implizit Anwendung.

- Bezug: Blocktransformation

  Referenz: (3.96), (3.99)

$$(\mathtt{T}, \mathtt{Tinv}, \mathtt{\Lambda}) = \begin{cases} (T, T^{-1}, \Lambda) & \text{falls} \quad k = 0 \\ (T_u, T_u^{-1}, \Lambda_u) & \text{falls} \quad k = 1 \end{cases} \tag{3.142}$$

- Bezug: Berücksichtigung einer expliziten ersten Stufe im Block

  Referenz: (3.99)

$$\mathtt{LambTinva1} = \Lambda_u T_u^{-1} a_1 \tag{3.143}$$

- Bezug: Bereitstellung der z_i^0 für Diagonalstufen

  Referenz: (3.127), (3.131)

$$\mathtt{AlphAinvTI} = \begin{cases} \bar{A} \cdot \mathcal{A}_{s-1}^{-1} \cdot \begin{pmatrix} T \\ I_{s-1-\bar{m}} \end{pmatrix} & \text{falls} \quad k = 0 \\ \bar{A} \cdot \text{Æ}_{s-1}^{-1} \cdot \begin{pmatrix} 1 & T_u \\ & I_{s-1-\bar{m}} \end{pmatrix} & \text{falls} \quad k = 1 \end{cases} \tag{3.144}$$

- Bezug: Bereitstellung der k_i^0 für Diagonalstufen

  Referenz: (3.128), (3.132)

$$\mathtt{nDinvGamAinvTI} = \begin{cases} -D_\gamma \cdot \bar{\Gamma} \cdot \mathcal{A}_{s-1}^{-1} \cdot \begin{pmatrix} T \\ I_{s-1-\bar{m}} \end{pmatrix} & \text{falls} \quad k = 0 \\ -D_\gamma \cdot \bar{\Gamma} \cdot \text{Æ}_{s-1}^{-1} \cdot \begin{pmatrix} 1 & T_u \\ & I_{s-1-\bar{m}} \end{pmatrix} & \text{falls} \quad k = 1 \end{cases} \tag{3.145}$$

- Bezug: Bereitstellung zeitexpliziter Gewichte α_i und γ_i für Diagonalstufen

  Referenz: (3.17b), (3.133)

$$(\mathtt{alphsum}, \mathtt{gamsum}) = (\bar{A}e, \Gamma_{(\bar{m}+1:s,1:s)}e) \tag{3.146}$$

  mit dem Eins-Vektor e in $\mathbb{R}^{s-1}$ bzw. $\mathbb{R}^s$

- Bezug: Bereitstellung von y_1 und err_{est}^{loc}

  Referenz: (3.135), (3.136)

$$\begin{pmatrix} \mathtt{bAinvTI} \\ \mathtt{dbAinvTI} \end{pmatrix} = \begin{cases} \begin{pmatrix} b^T \mathcal{A}^{-1} \begin{pmatrix} T \\ I_{s-\bar{m}} \end{pmatrix} \\ \delta b^T \mathcal{A}^{-1} \begin{pmatrix} T \\ I_{s-\bar{m}} \end{pmatrix} \end{pmatrix} & \text{falls} \quad k = 0 \\ \begin{pmatrix} b^T (\mathcal{A} + e_1 e_1^T)^{-1} \begin{pmatrix} 1 & T_u \\ & I_{s-\bar{m}} \end{pmatrix} \\ \delta b^T (\mathcal{A} + e_1 e_1^T)^{-1} \begin{pmatrix} 1 & T_u \\ & I_{s-\bar{m}} \end{pmatrix} \end{pmatrix} & \text{falls} \quad k = 1 \end{cases} \tag{3.147}$$

Wie bereits erwähnt, sind in den obigen Ausdrücken sämtliche blockbezogenen Größen als leere Entitäten zu behandeln, wenn nur ein trivialer Block vorliegt, i. e., wenn $m = 1+k$ gilt.

Alg. 3.2 Berechnung von W- und Z-Entitäten im Block

Data : $k \in \{0,1\}$ (k-explizites $\mathcal{A}$), Z_m^0, $\tau^0 = (\tau_i^0)_{i=1,\dots,m}$, maxiterB, Methodengrößen (3.142) und (3.143) sowie t_0, y_0, $f_0 = f(t_0, y_0)$, Schrittweite h, Jacobimatrix J

if $m \geqslant 2 + k$ **then** // nichttrivialer Block

  set $\mathtt{W} = \mathtt{zeros}(n, m-k)$, $\mathtt{dW} = \mathtt{zeros}(n, m-k)$, $\mathtt{F_b} = \mathtt{zeros}(n, m-k)$, $\mathtt{Z_b} = \mathtt{zeros}(n, m)$;

  for $l = 1 : \mathtt{maxiterB}$ **do**

    if $l = 1$ **then** // Startwerte setzen

      set $\tau = \tau^0$, $\Delta\tau = (-\tau_i^0 + c_i h)_{i=1,\dots,m}$ sowie $\mathtt{Z_b}(:, i) = z_i^0$ für $i = 1+k, \dots, m$ und hiermit $\mathtt{W} = \mathtt{Z_b}(:, 1+k:m) \cdot \mathtt{Tinv}^T$;

    else // exakte bzw. iterierte Werte nutzen

      set $\tau = h(c_1, \dots, c_m)$, $\Delta\tau = 0$ sowie $\mathtt{Z_b}(:, 1+k:m) = \mathtt{W} \cdot \mathtt{T}^T$;

    set $\mathtt{F_b} = \Big[f\Big(t_0 + \tau_{1+k}, y_0 + \mathtt{Z_b}(:, 1+k)\Big), \dots, f\Big(t_0 + \tau_m, y_0 + \mathtt{Z_b}(:, m)\Big)\Big]$;

    set $\mathtt{rhs} = -h^{-1}\mathtt{W} \cdot \Lambda^T + \mathtt{F_b} \cdot \mathtt{Tinv}^T$;

    // Berücksichtigung einer etwaigen expliziten ersten Stufe

    if $k = 1$ **then set** $\mathtt{rhs} = \mathtt{rhs} + f_0 \cdot \mathtt{LambTinva1}^T$;

    // Berücksichtigung eines etwaigen zeitexpliziten Einfluss

    if $\Delta\tau(1+k:m) \neq 0$ **then set** $\mathtt{rhs} = \mathtt{rhs} + \partial f_0/\partial t \cdot \Big(\mathtt{Tinv} \cdot \Delta\tau(1+k:m)\Big)^T$;

    for $j = 1 : m-k$ **do** // sukzessives Lösen des Blocksystems

      löse $(h^{-1}\Lambda(j,j)I - J)\mathtt{dW}(:, j) = \mathtt{rhs}(:, j)$;

      if $j < m-k$ **then** // Aufdatieren von rhs für Folgesysteme

        set $\mathtt{rhs}(:, j+1:m-k) = \mathtt{rhs}(:, j+1:m-k) - h^{-1}\mathtt{dW}(:, j) \cdot \Big(\Lambda(j+1:m-k, j)\Big)^T$

    set $\mathtt{W} = \mathtt{W} + \mathtt{dW}$;

else // trivialer Block

  set $\mathtt{W} = [\,]$;

Alg. 3.3 Berechnung von Z-Entitäten in Diagonalstufen

Data : $k \in \{0,1\}$ (k-explizites $\mathcal{A}$), `W` aus Alg. 3.2, `maxiterD`, Methodengrößen (3.144)-(3.146) sowie t_0, y_0, $f_0 = f(t_0, y_0)$, Schrittweite h, Jacobimatrix J

if $s - \bar{m} > 0$ **then** `// Diagonalstufe(n) oder trivialer Block liegen vor`
  set $\texttt{ki} = \texttt{zeros}(n, 1)$;
  if $k = 0$ **then set** $\texttt{WZ} = [\texttt{W}, \texttt{zeros}(n, s - \bar{m})]$ **else set** $\texttt{WZ} = [f_0, \texttt{W}, \texttt{zeros}(n, s - \bar{m})]$;
  for $i = \bar{m} + 1 : s$ **do** `// Stufe` i
    for $l = 1 : \texttt{maxiterD}(i)$ **do**
      if $l = 1$ **then** `// Startwerte setzen`
        if $i = 1$ **then** `// trivialer Block, 0-explizites` $\mathcal{A}$
          set $\tau_i = 0$ oder $\tau_i = c_i h$; `// ki und WZ(:,i) bereits korrekt zu Null initialisiert`
        else
          set $\texttt{WZ}(:, i) = \texttt{WZ}(:, 1 : i-1) \cdot \Big(\texttt{AlphAinvTI}(i - \bar{m}, 1 : i - \bar{m} - 1)\Big)^T$;
          set $\texttt{ki} = \texttt{WZ}(:, 1 : i-1) \cdot \Big(\texttt{nDinvGamAinvTI}(i - \bar{m}, 1 : i - \bar{m} - 1)\Big)^T$;
          set $\tau_i = \texttt{alphsum}(i - \bar{m})h$ oder $\tau_i = c_i h$;
      else
        set $\tau_i = c_i h$; `// stets exakte Zeitdaten`
        set $\texttt{ki} = \texttt{ki} + (\mathcal{A}_{(i,i)})^{-1} \cdot \texttt{dzi}$;
      set $\texttt{rhsi} = -h^{-1}\texttt{ki} + f\Big(t_0 + \tau_i, y_0 + \texttt{WZ}(:, i)\Big)$;
      `// Berücksichtigung eines etwaigen zeitexpliziten Einfluss`
      if $\tau_i \neq c_i h$ **then set** $\texttt{rhsi} = \texttt{rhsi} + \texttt{gamsum}(i - \bar{m}) \cdot h\partial f_0/\partial t$;
      `// Lösen des linearen Systems`
      löse $(h^{-1}(\mathcal{A}_{(i,i)})^{-1} I - J)\texttt{dzi} = \texttt{rhsi}$;
      set $\texttt{WZ}(:, i) = \texttt{WZ}(:, i) + \texttt{dzi}$;
else `// keine Diagonalstufen und kein trivialer Block`
  if $k = 0$ **then set** $\texttt{WZ} = \texttt{W}$ **else set** $\texttt{WZ} = [f_0, \texttt{W}]$;

Alg. 3.4 Berechnung von y_1 und Fehlerschätzung

Data : $k \in \{0,1\}$ (k-explizites $\mathcal{A}$), WZ aus Alg. 3.3, Methodengrößen (3.147) sowie y_0

if $y_1 = Y_i$ *für ein* $i \in \{1,\ldots,s\}$ **then** // y_1 entspricht einem Stufenwert
  if $i \leqslant \bar{m}$ **then** // Y_i ist Blockstufe
    if $i > k$ **then** // $z_i^{l_i}$ liegt nicht vor
      set $z_i^{l_i} = \mathtt{WZ}(:, 1+k:m) \cdot \left(\mathtt{T}(i-k,:)\right)^T$; // vgl. (3.138)
    else
      set $z_i^{l_i} = 0$; // trivialer Zuwachs; hier gilt $i = k = 1$
  else
    set $z_i^{l_i} = \mathtt{WZ}(:, i)$;
  set $y_1 = y_0 + z_i^{l_i}$;
  if $\tilde{y}_1 = Y_j$ *für ein* $j \in \{1,\ldots,s\} \setminus \{i\}$ **then** // $\tilde{y}_1$ ist Stufenwert
    if $j \leqslant \bar{m}$ **then** // Y_j ist Blockstufe
      if $j > k$ **then** // $z_j^{l_j}$ liegt nicht vor
        set $z_j^{l_j} = \mathtt{WZ}(:, 1+k:m) \cdot \left(\mathtt{T}(j-k,:)\right)^T$; // vgl. (3.138)
      else
        set $z_j^{l_j} = 0$; // trivialer Zuwachs; hier gilt $j = k = 1$
    else
      set $z_j^{l_j} = \mathtt{WZ}(:, j)$;
    if $i \in \{\bar{m}+1,\ldots,s\}, j \in \{1,\ldots,i-1\}$ *und* $z_i^0 = z_j^{l_j}$ *und* $l_i = 1$ **then**
      set $err_{est}^{loc} = \Delta z_i^0$; // einziges Newtoninkrement in Stufe i
    else
      set $err_{est}^{loc} = z_i^{l_i} - z_j^{l_j}$;
  else // $\tilde{y}_1$ ist kein Stufenwert
    set $err_{est}^{loc} = \mathtt{WZ} \cdot (\mathtt{dbAinvTI})^T$;
else // y_1 ist kein Stufenwert
  set $y_1 = y_0 + \mathtt{WZ} \cdot (\mathtt{bAinvTI})^T$;
  set $err_{est}^{loc} = \mathtt{WZ} \cdot (\mathtt{dbAinvTI})^T$;

Bemerkung 3.53. Die Algorithmen 3.2-3.4 stellen eine Grundform der Berechnung der relevanten Lösergrößen dar. Anpassungen im Detail hinsichtlich der einzelnen Rechenschritte und Speicherung verschiedener Variablen sind in einer tatsächlichen Implementierung sinnvoll. Z. B. kann zu Anfang eines Schrittes $h^{-1}\Lambda$ berechnet werden, da dieses Produkt konstant ist während des Schrittes. Ebenso bietet es sich an, statt mit `ki` zu arbeiten, das Produkt $-h^{-1}\texttt{ki}$ aufzudatieren. In den dargestellten Algorithmen wird von solchen Effizienz-Maßnahmen abgesehen, um den Einfluss der festen Methodenentitäten und den von der Dynamik abhängenden Größen h und f deutlicher abzugrenzen.

Ausnahmebehandlungen bei fehlender Wohldefiniertheit von f sowie Kriterien für einen frühzeitigen Abbruch der Newtoniterationen bei hinreichender Güte der Approximationen oder bei Divergenz sind ebenfalls zu berücksichtigen. Hinsichtlich Letzterem s. a. Abschnitt 3.5.1.1 und Alg. 3.8.

3.3.5 Das T2-Update

Nach Satz 3.26 ist es vorteilhaft, $J = f_0^{(1)}$ im Newtonprozess (3.60) zu wählen, da dann pro Newtonschritt zwei Ableitungsordnungen abgeglichen werden können. In der Praxis ist die Bereitstellung von $f_0^{(1)}$ in jedem Schritt selten möglich. Satz 3.26 liefert dann die Erkenntnis, dass für beliebiges J zumindest eine Ableitungsordnung abgeglichen wird. An dieser Stelle wird ein Ansatz beschrieben, welcher mit möglichst wenig zusätzlichem Aufwand einen gewissen Zwischenweg zeichnet. Dieser Ansatz bedient sich eines $J \neq f_0^{(1)}$, modifiziert dieses aber über ein Rang-1-Update – das T2-Update – derart, dass sinnvolle Informationen der exakten Jacobimatrix $f_0^{(1)}$ einfließen können.

Ziel des Vorgehens ist es, die notwendige Iterationszahl in den Blockstufen um Eins zu verringern. Statt also bei Einhaltung der Auslöschungsbedingung (3.77b) und mit einem Startwert $K_b^0 = 0$ für eine Stufenordnung von p ebenfalls p Block-Newtonschritte durchzuführen, s. Korollar 3.37, wird die Anzahl der Block-Newtonschritte auf $p - 1$ verringert. Es ergibt sich eine Ersparnis von p Standard-Newtonschritten. Somit sind für k-explizites $\mathcal{A}$, $k \in \{0, 1\}$, mindestens $p - k$ lineare Systeme weniger zu lösen. Wie im Folgenden gezeigt wird, bedarf der Modifikationsansatz der Lösung eines zusätzlichen linearen Gleichungssystems. Das heißt, ab $p = 2 + k$ handelt es sich hinsichtlich des Pro-Schritt-Aufwandes um ein rentables Vorgehen.

Bemerkung 3.54. Das im nächsten Unterabschnitt hergeleitete T2-Update ist bei expliziter Verfügbarkeit von J wie dargestellt anwendbar. Dies ist der Fall, wenn eine direkte Methode zum Lösen der linearen Gleichungssysteme in Zeile 13 von Alg. 3.2 genutzt

wird. Hinsichtlich eines iterativen Verfahrens, i. e. Krylovraumverfahrens, gestaltet sich die Situation komplexer. Entsprechend der Theorie muss für die Blockstufen die gleiche Matrix genutzt werden. Unter gewissen Regularitätsanforderungen an die Menge der approximativen Lösungen aus dem iterativen Verfahren kann a posteriori eine Matrix J konstruiert werden, die zu einem gleichen Lösungsverhalten führt, wie es durch die iterativ ermittelten Approximationen vorgegeben wird. Schwierig hierbei ist, die Sicherstellung dieser Regularitätsbedingungen, da diese problemabhängig sein können. Es wurde davon abgesehen, sich im Detail weiter mit diesem Ansatz zu beschäftigen oder eine Alternative hierzu zu formulieren. Im Rahmen der iterativen Verfahren sei das T2-Update bei expliziter Verfügbarkeit eines Präkonditionierers in seiner Anwendung auf diesen beschränkt – mit der Hoffnung, die Konvergenz des Krylovraumverfahrens zu beschleunigen.

Bemerkung 3.55. Das T2-Update kann unter Berücksichtigungen der Aussagen in Bemerkung 3.54 ebenso für die linearen Gleichungssysteme zu den Diagonalstufen eingesetzt werden. Die Wirksamkeit des Updates beruht jedoch darauf, dass der Newtonprozess mit einer Null-Approximation gestartet wird. Dies ist bezüglich einer Diagonalstufe in der Regel nicht der Fall. Auf der anderen Seite sollte unabhängig von der Startapproximation eine Anwendung des T2-Updates nicht schaden oder sogar begünstigend wirken, da für eine gewisse Raumrichtung eine Angleichung von J an $f_0^{(1)}$ vorgenommen wird, s. (3.153).

3.3.5.1 Herleitung des T2-Updates

Für $k \in \{0, 1\}$ sei $\mathcal{A}$ k-explizit von der Form (3.71) mit A_m und $m \geqslant 1 + k$. Entsprechend der Einführung zu diesem Abschnitt wird sich auf den Block und somit auf A_m beschränkt.

Zur Motivation des T2-Updates bietet sich die Newtondarstellung (3.92) für $l = 0$ mit $Z_m^0 = 0$ also $\Delta Z_m^0 = Z_m^1$ an. Ableiten unter Anwendung von (3.58) liefert

$$\begin{aligned} &{Z_{m|h=0}^1}^{(0)} = 0, \qquad {Z_{m|h=0}^1}^{(1)} = A_m e \otimes f_0 \\ &{Z_{m|h=0}^1}^{(2)} = 2(A_m^2 e \otimes J f_0). \end{aligned} \tag{3.148}$$

mit $f_0 = f(y_0)$ und e, dem Eins-Vektor in $\mathbb{R}^m$. Für Z_m aus (3.90), welches sich implizit definiert über

$$Z_m = h(A_m \otimes I) \begin{pmatrix} f(y_0 + z_1) \\ \vdots \\ f(y_0 + z_m) \end{pmatrix}, \tag{3.149}$$

gilt

$$\begin{aligned} Z_{m|h=0}^{(0)} &= 0 \qquad Z_{m|h=0}^{(1)} = A_m e \otimes f_0 \\ Z_{m|h=0}^{(2)} &= 2(A_m^2 e \otimes f_0^{(1)} f_0) \end{aligned} \tag{3.150}$$

mit $f_0^{(1)} = \partial f(y_0)/\partial y$. Wie von Satz 3.26 vorausgesagt, ergibt sich unabhängig von der Wahl des J ein Abgleich der Ableitungen bis zur Ordnung Eins. Ist zusätzlich

$$Jf_0 = f_0^{(1)} f_0 \tag{3.151}$$

gegeben, so reicht der Abgleich bis in die zweite Ableitung. Gilt $f_0 \neq 0$ – andererseits wäre (3.151) sowieso erfüllt –, so lässt sich (3.151) sicherstellen, wenn statt J die Matrix $J + uv^T$ mit

$$v = f_0/\|f_0\|_2 \qquad \text{und} \qquad u = (f_0^{(1)} - J)v \tag{3.152}$$

genutzt wird. Ergo ergibt sich

$$(J + uv^T)f_0 = f_0^{(1)} f_0. \tag{3.153}$$

Dies motiviert, das Rang-1-Update uv^T als *T2-Update* zu bezeichnen: Bei Wohldefiniertheit aller Größen stimmen Z_m^1 und Z_m in ihren Taylorentwicklungen nach h bei $h = 0$ bis zur zweiten Ordnung überein, wenn $J + uv^T$ genutzt wird.

Bemerkung 3.56. In der Definition von u taucht die exakte Jacobimatrix $f_0^{(1)}$ auf, jedoch nur in Anwendung auf den Vektor v. Somit bedarf es lediglich einer Routine, welche das Produkt $f_0^{(1)} v$ zur Verfügung stellt. Dies kann z. B. über einen entsprechenden Finite-Differenzen-Ansatz geschehen, welche eine zusätzliche Funktionsauswertung benötigt.

Damit Wohldefiniertheit von Z_m^1 für die Wahl $J + uv^T$ gewährleistet ist, muss eine gewisse Regularitätsbedingung erfüllt sein. Hierzu sei das System (3.92) in der transformierten Form (3.96) betrachtet. Bezieht man das T2-Update mit ein, so sind die dort in den linearen Gleichungssystemen auftretenden Matrizen entsprechend (3.97) von der erweiterten Gestalt

$$h^{-1}\lambda I - (J + uv^T), \qquad \lambda \in \mathbb{R}. \tag{3.154}$$

Umformen liefert

$$h^{-1}\lambda I - \left(J + uv^T\right) = \overbrace{(h^{-1}\lambda I - J)}^{=:M} \overbrace{(I - \underbrace{M^{-1}u}_{=:\widetilde{u}} v^T)}^{=:\widetilde{I}}. \tag{3.155}$$

Unter der Voraussetzung, dass M regulär ist, ist auch die Matrix in (3.154) regulär und somit Z_m^1 wohldefiniert genau dann, wenn

$$\widetilde{I} \quad \text{ist regulär} \qquad \Leftrightarrow \qquad 1 - v^T \widetilde{u} \neq 0 \tag{3.156}$$

gilt. Letzteres folgt aus dem Determinanten-Lemma für Matrizen, s. z. B. /BRO 11/.

Bemerkung 3.57. Man beachte, dass $h \to 0$ zur Folge hat, dass $M^{-1} \to 0$. Der Einfluss von v und u (sowie J) nimmt also ab, wenn h kleiner wird. Es gibt somit ein $\bar{h}$, so dass $1 - v^T \widetilde{u} > 0$ für $0 < h < \bar{h}$ ist. Zum einen zeigt dies, dass die modifizierte Matrix (3.154) für hinreichend kleine h regulär ist. Zum anderen bietet sich über die Forderung

$$1 - v^T \widetilde{u} > \varepsilon > 0 \tag{3.157}$$

ein erstes Kriterium an, um eine Verbesserung der Approximatonsgüte von J in Erwägung zu ziehen. Dieses liefert jedoch keine Aussage über die Kondition von $\widetilde{I}$, welche im Sinne der Nutzung von $\widetilde{I}$ in der Lösung von linearen Gleichungssystemen deutlich nützlicher wäre. Ein hierzu angepasstes Kriterium findet sich in Unterabschnitt 3.3.5.3.

3.3.5.2 Verträglichkeit mit der Auslöschungsbedingung (3.77a)

Wird eine Stufenordnung von lediglich $p = 1$ gefordert und eine Methode zum Ordnungspaar $2(1)$ betrachtet, dann liefert das im vorherigen Abschnitt diskutierte und unter Einbindung des T2-Updates konstruierte Z_m^1 bereits hinreichende Approximationsgüte.

Sei der allgemeinere Fall gegeben, dass $\mathcal{C}(p)$ für $p \geqslant 2$ gilt und eine Methode zum Ordnungspaar $p+1(p)$ vorliegt. Eine Ersparnis an benötigten Block-Newtonschritten mittels einer Kombination aus T2-Update und Ausnutzen der Auslöschungsbedingung (3.77b) ergibt sich dann, wenn Z_m^1 die Auslöschungsbedingung (3.77a) impliziert (mit dem Indexverschub $1 \to 0$). Da mittels Z_m^1 die ersten zwei Ableitungsordnungen abgeglichen werden, wenn das T2-Update zum Einsatz kommt, ist nach Satz 3.36 also die dritte Ableitung von Z_m^1 an $h = 0$ zu untersuchen. Diese ergibt sich zu

$$\begin{aligned} {Z_m^1}_{|h=0}^{(3)} &= 6(A_m^3 e \otimes (J + uv^T)^2 f_0) \\ &\overset{\mathcal{C}(p)}{=} 3(A_m C_m^2 e \otimes (J + uv^T)^2 f_0) \end{aligned} \tag{3.158}$$

mit $C_m = \mathrm{diag}(c_1, \ldots, c_m)$. Ist $\mathcal{A}$ 0-explizit und somit A_m regulär, folgt wegen (3.91) sofort

$${K_m^1}_{|h=0}^{(3)} = 3(C_m^2 e \otimes (J + uv^T)^2 f_0),$$

was der Auslöschungsbedingung (3.77a) genügt. Im Falle eines 1-expliziten $\mathcal{A}$ ergibt sich die gleiche Darstellung durch direktes Ableiten von (3.85) für $l = 0$, $K_m^0 = 0$, A_m statt $\mathcal{A}$ und unter Berücksichtigung von $\mathcal{C}(p)$.

Bemerkung 3.58. Einer der Vorteil des T2-Updates ist, dass es sich mit relativ geringem Zusatzaufwand einbinden lässt, s. Unterabschnitt 3.3.5.6 für Details. Dies ändert sich, wenn höhere Ableitungen abgeglichen werden sollen. Betrachtet man von Z_m aus (3.149) z. B. die dritte Ableitung an $h = 0$, so treten Terme der Form $f_0^{(2)}[f_0]^2$ sowie $f_0^{(1)} f_0^{(1)} f_0$ auf. Diese müssten nach (3.148) durch ein Produkt der Form $J^2 f_0$ ebenfalls abgefangen werden, ohne dass dabei die Eigenschaften hinsichtlich der zweiten Ableitungsordnung verloren gehen. Dies lässt sich als Bedingungen an J formulieren. Man wird aber nicht umhinkommen, Approximationen für $f_0^{(2)}[f_0]^2$ bzw. $f_0^{(1)} f_0^{(1)} f_0$ zu finden. In einem Kontext, in dem bereits $f_0^{(1)}$ approximiert werden muss, kann es hierbei leicht zu numerischen „Verschmierungen“ kommen, welche den gewünschten Ordnungsabgleich gefährden können. Dies gilt leider auch für den alternativen Ansatz, in der Newtondarstellung (3.92) mit h^2 gewichtete explizite Ableitungsterme auf der rechten Seite einzuführen. Hinzukommt, dass diskutiert werden müsste, welchen Einfluss solche Modifikationen auf die Stabilität des Verfahrens hinsichtlich Steifheit hätten. Eine solche Betrachtung wurde für diese Arbeit nicht weiter verfolgt, so dass sich auf das T2-Update beschränkt wird.

3.3.5.3 Eigenschaften des T2-Updates

Orientiert man sich an den Beweistechniken zu /DEU 04, Th. 2.7/, so lässt sich leicht zeigen, dass das T2-Update uv^T mit u und v aus (3.152) sowie mit M und $\widetilde{I}$ aus (3.155) eine Least-change-Eigenschaft besitzt:

$$\begin{aligned}
&\arg\min_{B\in\mathcal{T}}\|B\|_2 = uv^T, \qquad \mathcal{T} := \{B \mid (J+B)v = f^{(1)}(y_0)v\} \\
&\qquad \Longleftrightarrow \\
&\arg\min_{N\in\widetilde{\mathcal{T}}}\|I - M^{-1}N\|_2 = M\widetilde{I}, \qquad \widetilde{\mathcal{T}} := \{N \mid Nv = \Big(h^{-1}\lambda I - f^{(1)}(y_0)\Big)v\},
\end{aligned} \tag{3.159}$$

und es gilt

$$\min_{N\in\widetilde{\mathcal{T}}}\|I - M^{-1}N\|_2 = \|\tilde{u}\|_2. \tag{3.160}$$

Die Beziehung (3.160) lässt sich derart interpretieren, dass der relative Abstand von M zu einem N, welches in $\widetilde{\mathcal{T}}$ liegt, im Maß der Euklidischen Norm mindestens $\|\widetilde{u}\|_2$ beträgt. Die Größe $\|\tilde{u}\|_2$ selbst lässt sich wiederum als ein relativer Abstand deuten. Denn diese lässt sich schreiben als

$$\|\tilde{u}\|_2 = \frac{\|M^{-1}\Big(M - (h^{-1}\lambda I - f_0^{(1)})\Big)f_0\|_2}{\|f_0\|_2}. \tag{3.161}$$

Im Sinne eines Gütekriteriums für M, und damit für J, erscheint es sinnvoll zu fordern, dass mindestens

$$\|\tilde{u}\|_2 < \delta < \frac{1}{2} \tag{3.162}$$

gilt, um wenigstens Übereinstimmung im führenden Bit von M und $N = M\widetilde{I}$ bzw. M und $h^{-1}\lambda I - f_0^{(1)}$ in Richtung von f_0 zu sichern.

Die Größe $\|\tilde{u}\|_2$ taucht auch in einer Konditionskontrolle zu $\widetilde{I}$ auf. Gelte nicht nur (3.156), sondern

$$\theta := \|v^T\widetilde{u}\|_2 = \|\widetilde{u}\|_2 < 1,$$

dann ergibt sich aus /DEU 04, Lem. 2.8/ die Abschätzung

$$\mathrm{cond}_2(\widetilde{I}) \leqslant \frac{1+\theta}{1-\theta} \tag{3.163}$$

und (3.162) liefert

$$\|\tilde{u}\|_2 < \delta < \frac{1}{2} \qquad \Rightarrow \qquad \mathrm{cond}_2(\widetilde{I}) \leqslant 3. \tag{3.164}$$

3.3.5.4 Stabilitätsbetrachtungen zum T2-Update

Eine sinnvolle Anwendung des T2-Updates setzt $J \neq f_0^{(1)}$ voraus. Entsprechend der Anmerkungen in /HAI 96, § IV.7/ gestaltet sich eine Stabilitätsbetrachtung in diesem Fall jedoch recht kompliziert und soll auch an dieser Stelle nicht ausgeführt werden. Es sei aber folgende Anmerkung gemacht: Wie in Unterabschnitt 3.3.5.6 gezeigt wird, bietet sich die Produktdarstellung (3.155) als Grundlage zur Implementation des T2-Updates an. Der Einfluss des Updates beschränkt sich dann auf $\widetilde{I}$. Liegt Steifheit vor, so ist der Vektor v nicht wesentlich betroffen, da dieser auf eine Euklidische Länge von Eins normiert ist. Zwar wird der Vektor u direkt von Steifheit beeinflusst, jedoch wird zur Ermittlung von $\widetilde{I}$ auf $\tilde{u}$ zurückgegriffen, welches eine inverse Anwendung von M beinhaltet, also aus einer der Steifheit entgegenwirkenden Operation resultiert. Die Güteschranke (3.162) lässt sich somit unabhängig von der Steifheit des Problems festlegen. Und bei Einhaltung von (3.162) zeigt (3.164), dass das multiplikative T2-Update stets stabil angewandt werden kann.

3.3.5.5 Berücksichtigung nicht-autonomer Systeme

Wie bereits in Unterabschnitt 3.3.4.1 für den allgemeinen Fall diskutiert, kann das bisher betrachtete System als Autonomisierung des nicht-autonomen Systems mittels Hinzunahme von $t' = 1$ interpretiert werden. Oder es kann das nicht-autonome System dadurch

berücksichtigt werden, dass die zeitexpliziten Informationen τ_i bereits in der Startnäherung exakt vorgegeben werden, i. e., $\tau_i^0 = c_i h$ für $i = 1, \ldots, m$. Der Startwert $Z_m^0 = 0$ bezieht sich dann nur noch auf den y-Anteil des Systems.

Im ersten Fall ändert sich nichts an Theorie und Herleitung, sofern – wie sonst auch angenommen – eine Jacobi-Struktur der Form (3.101) vorliegt. Alle bisher getroffen Aussagen bleiben derart erhalten. Man beachte, dass sich formal v von $f_0/\|f_0\|_2$ zu $(1, f_0^T)^T/\|(1, f_0^T)^T\|_2$ ändert. Der Vektor u wird dann über

$$\|(1, f_0^T)^T\|_2^{-1} \cdot \left[\begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & \frac{\partial f_0}{\partial t} \end{pmatrix} - \begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial y} & J \end{pmatrix} \right] \begin{pmatrix} 1 \\ f_0 \end{pmatrix} \tag{3.165}$$

beschrieben. Die spezielle Struktur erlaubt für $f_0 \neq 0$ folgende Modifikation: Wähle

$$v = \|(0, f_0^T)^T\|_2^{-1} \cdot \begin{pmatrix} 0 \\ f_0 \end{pmatrix} =: \begin{pmatrix} 0 \\ \hat{v} \end{pmatrix}$$

und entsprechend

$$u = \|(0, f_0^T)^T\|_2^{-1} \cdot \left[\begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & \frac{\partial f_0}{\partial t} \end{pmatrix} - \begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & J \end{pmatrix} \right] \begin{pmatrix} 0 \\ f_0 \end{pmatrix} =: \begin{pmatrix} 0 \\ \hat{u} \end{pmatrix}.$$

Dann gilt

$$\left[\begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & J \end{pmatrix} + uv^T \right] \begin{pmatrix} 1 \\ f_0 \end{pmatrix} = \begin{pmatrix} 0 & 0^T \\ \frac{\partial f_0}{\partial t} & J + \hat{u}\hat{v}^T \end{pmatrix} \begin{pmatrix} 1 \\ f_0 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{\partial f_0}{\partial t} + \frac{\partial f_0}{\partial y} f_0 \end{pmatrix}.$$

Die wesentliche Operation führt nach wie vor zum gewünschten Ergebnis, und die Theorie bleibt wie bisher beschrieben erhalten. Der Vorteil an der Modifikation liegt darin, dass ein direktes Berücksichtigen der zeitexpliziten Größen wie in (3.103) oder (3.106) genau so wie dort beschrieben auch bei der Anwendung eines T2-Updates vorgenommen werden kann.

Wird der zeitexplizite Anteil exakt behandelt, i. e., $\tau_i^0 = c_i h$ für $i = 1, \ldots, m$, so wird wie in der Herleitung direkt J über uv^T modifziert. Es ändert sich jedoch die Newtondarstellung zu

$$\Big(I - hA_m \otimes (J + uv^T)\Big) Z_m^1 = h(A_m \otimes I) \begin{pmatrix} f(t_0 + c_1 h, y_0) \\ \vdots \\ f(t_0 + c_m h, y_0) \end{pmatrix}.$$

Ableiten liefert

$$\begin{aligned}
&{Z_m^1}_{|h=0}^{(0)} = 0 \qquad {Z_m^1}_{|h=0}^{(1)} = A_m e \otimes f_0 \\
&{Z_m^1}_{|h=0}^{(2)} = 2(A_m^2 e \otimes (J + uv^T) f_0) + 2A_m C_m^2 e \otimes \partial f_0/\partial t \\
&{Z_m^1}_{|h=0}^{(3)} = 6\Big(A_m^3 e \otimes (J + uv^T)^2 f_0 + A_m^2 C_m e \otimes \partial f_0/\partial t\Big) + 3(A_m C_m^2 e \otimes \partial^2 f_0/\partial t^2).
\end{aligned}$$

(3.166)

Damit besteht auch in diesem Fall Übereinstimmung mit Z_m aus (3.149) bis zur zweiten Ableitungsordnung, wenn

- Z_m als autonomisiert durch Hinzunahme von $t' = 1$ interpretiert wird und mindestens $\mathcal{C}(1)$ gilt.
- Z_m exakte Zeitinformationen $\tau_i = c_i h$ nutzt, analog zu Z_m^1.

Unter der Voraussetzung $\mathcal{C}(p)$ mit $p \geqslant 2$ folgt aus der dritten Ableitung in (3.166), dass die Auslöschungsbedingung (3.77a) erfüllt ist. Die Argumentation hierzu verläuft analog zu der Diskussion in Unterabschnitt 3.3.5.2.

3.3.5.6 Das T2-Updates in der Praxis

Unabhängig davon, ob Block- oder Diagonalstufen betrachtet werden, liegt bei der Anwendung eines T2-Updates ein System mit einer Matrix der Gestalt (3.154) vor, i. e.,

$$\left[h^{-1}\lambda I - (J + uv^T)\right]x = rhs, \tag{3.167}$$

vgl. Alg. 3.2, Zeile 13, sowie Alg. 3.3, Zeile 18, wobei statt J die Summe $J + uv^T$ auftritt.

Anwendung des T2-Updates

Um nicht die Dünnbesetztheitsstruktur von $M = h^{-1}\lambda I - J$ zu zerstören oder sich der Möglichkeit zu berauben, eine Zerlegung hiervon über mehrere Zeitschritte effizient zu nutzen, bietet sich die Produktdarstellung (3.155) an. Damit ergibt sich

1. löse $M\mathring{x} = rhs$,
2. löse $(I - \widetilde{u}v^T)x = \mathring{x}$,

wobei sich $\widetilde{u}$ aus

$$M\widetilde{u} = u \overset{(3.152)}{=} (f_0^{(1)} - J)v \tag{3.168}$$

ermittelt. Das System zur Bestimmung von x lässt sich unter der Bedingung (3.156) leicht vermittels der Sherman-Morrison-Formel, s. z. B. /BRO 11/, lösen:

$$x = \left[I + \frac{\widetilde{u}v^T}{1 - v^T\widetilde{u}}\right]\mathring{x}. \tag{3.169}$$

Bezüglich der Bestimmung des Vektors $\widetilde{u}$ lässt sich die Darstellung von $\widetilde{u}$ in (3.161) gewinnbringend nutzen:

1. löse $M\mathring{u} = f_0^{(1)} v - h^{-1}\lambda v$,

2. setze $u = v + \mathring{u}$.

Dies spart eine Multiplikation mit J ein. Das beschriebene Vorgehen wird in den Algorithmen 3.5 und 3.6 zusammengefasst.

Alg. 3.5 Berechnung von $\widetilde{u}$ aus der Produktdarstellung des T2-Updates in (3.155)

Data : v aus (3.152), λ, h und J aus (3.167) sowie eine Prozedur `Dfdirec` zur (approximativen) Bestimmung von Richtungsableitungen $f_0^{(1)} \cdot d$

1 löse $(h^{-1}\lambda I - J)\mathring{u} = \texttt{Dfdirec}(v) - h^{-1}\lambda v$;

2 **set** $u = v + \mathring{u}$;

Alg. 3.6 Berechnung von x im vom T2-Update modifizierten System (3.167)

Data : $\widetilde{u}$ aus Alg. 3.5, v aus (3.152) sowie λ, h, J und rhs aus (3.167) und ggf. `1mvtu` aus einem vorherigen Aufruf dieses Algorithmus an t_0

1 **if** isempty(`1mvtu`) **then**

2 **set** $\texttt{1mvtu} = 1 - v^T\widetilde{u}$; `// für jede Stufe gleich → nur einmal berechnen`

3 löse $(h^{-1}\lambda I - J)\mathring{x} = rhs$;

4 **set** $x = \mathring{x} + \dfrac{v^T\mathring{x}}{\texttt{1mvtu}} \cdot \widetilde{u}$;

Kosten des T2-Updates

Als wesentliche Kosten zur Einbindung des T2-Updates ergeben sich das Ausführen des linearen Gleichungssystems zur Bestimmung von $\mathring{x}$ sowie die Bestimmung des Produktes $f_0^{(1)} v$, was einer zusätzlichen Auswertung von f entspricht, falls finite Differenzen bemüht werden, vgl. Bemerkung 3.56. Beide Operationen sind genau einmal pro Zeitintegrationsschritt auszuführen, unabhängig von der Anzahl der Stufen, in denen das T2-Update genutzt wird. Ändert sich die Schrittweite, aber nicht der Bezugspunkt t_0, so ist nur das System zur Bestimmung von $\mathring{x}$ neu zu lösen. Die Berechnung von x nach der Sherman-Morrison-Formel in (3.169) muss pro Stufe ausgeführt werden, welche das T2-Update nutzt, ist aber in $\mathcal{O}(n)$ Operationen durchführbar.

3.3.6 Extrapolationsansatz als FiterRK-Methode

Wie bereits in Abschnitt 3.2 erwähnt, ist für jede feste Extrapolationstiefe der Extrapolationsansatz basierend auf dem linear impliziten Euler als W-Methode interpretierbar.

W-Methoden sind spezielle $FiterRK$-Methoden mit trivialem Block und genau einem Newtonschritt pro Stufe. Da der linear-implizite Euler sich nicht einer expliziten ersten Stufe bedient, ist der triviale Block von der Größe Eins. Zur Berechnung der Stufenwerte ist Alg. 3.3 heranzuziehen, wobei über die Wahl von τ_i in Zeile 8 bzw. Zeile 12 von Alg. 3.3 die Wahl zwischen der klassischen Extrapolationsgröße $T_{i,1}$ oder der in Abschnitt 3.2.2 eingeführten modifzierten Extrapolationsgröße $modT_{i,1}$ getroffen wird. Es ergibt sich also entweder die Berechnungsvorschrift (3.133) oder (3.134) (mit $l = 0$). Erstere entspricht der Darstellung in /AUS 16a, § 6.2.1/, wobei dort bereits die spezielle Struktur von Γ, s. (3.25), berücksichtigt ist, da dann nach (3.15) $k_i^0 = 0$ gilt. Damit ist zwar $\mathcal{C}(1)$, vgl. (3.25), erfüllt, jedoch nicht $\mathcal{C}_\alpha(1)$ aus (3.73). Mit der Wahl k_i^0 handelt es sich um eine zu Satz 3.26 kompatible Startnäherung für $\varphi = 0$ in (3.63) (etwaige vorherige Stufen gehen mit $\mu = 1$ ein). Damit liegt nach einem Newtonschritt eine Approximation k_i^1 vor, welche von erster Ordnung ist, i. e., $k_i^1 - k_i \in \mathcal{O}(h^2)$. Damit ist der geringen Stufenordnung von Eins genüge getan. In einem solchen Kontext ist es nicht von Nachteil, dass die Informationen der vorherigen Stufen nicht für die Startapproximation k_i^0 über (3.15) genutzt werden. Entsprechend der Wahl $k_i^0 = 0$ für alle Stufen, ist $\bar{\Gamma}$ aus (3.123) identisch Null. Gleiches gilt folglich für die algorithmische Größe `nDinvGamAinvTI` aus (3.145). Die Variable `ki` in Zeile 11 von Alg. 3.3 wird somit korrekt zu Null initialisiert. Auch `AlphAinvTI` aus (3.144) stellt sich in diesem Zusammenhang als sehr einfach strukturierte Matrix dar: Die Variable $\texttt{WZ}(:, i)$ in Zeile 10 von Alg. 3.3 ergibt sich entweder zu Null, falls gerade eine Integration zu einer Teilschrittweite h/m_j, $m_j \in \mathbb{N}$, begonnen wird. Oder sie ist gleich $\texttt{WZ}(:, i-1)$ falls bereits mindestens ein Integrationsschritt zu h/m_j durchgeführt wurde. Dies entspricht genau dem Vorgehen, m_j Schritte des linear-impliziten Eulerverfahrens mit der festen Schrittweite h/m_j auszuführen, um $T_{j,1}$ zu konstruieren.

Die Stufenwerte erben direkt die Stabilitätseigenschaften des linear-impliziten Eulers. In der modfizierten Variante besteht sogar Steifakkuratesse, s. Proposition 3.17. Die eigentliche Extrapolation verläuft nach Alg. 3.1, was zur Berechnung von y_1 und dem lokalen Fehlerschätzer err_{est}^{loc} vermittels Zeile 25 bzw. Zeile 26 in Alg. 3.4 führt.

3.4 Konstruierte allgemeine Methoden

Basierend auf der Theorie in Abschnitt 3.3 wurden vier $FiterRK$-Methoden konstruiert: $FiterRK32a$, $FiterRK32b$, $FiterRK43$ und $FiterRK32ex$. Wesentliche Motivation war die Erhöhung der Stufenordnung gegenüber dem Extrapolationsansatz basierend auf dem linear impliziten Euler, s. a. die Diskussion in Abschnitt 3.2.2. Für alle diese entwickelten

Methoden gelten folgende Eigenschaften:

- Es handelt sich um eine eingebettete Methode. Die Ziffernfolge im Namen gibt das Ordnungspaar $p+1(p)$ an. Da die höhere Ziffer links steht, wird standardmäßig lokale Extrapolation genutzt; y_1 ist somit eine Approximation der Ordnung $p+1$.
- Ist das Ordnungspaar zur Methode durch $p+1(p)$ gegeben, so ist die Stufenordnung gleich p, d. h., $\mathcal{C}(p)$ aus (3.69a) ist erfüllt. Zusätzlich gilt $\mathcal{C}_\alpha(1)$ aus (3.73). Es ist somit die α-Stufenordnung mindestens gleich Eins.
- Jede Diagonalstufe bedarf nur eines Newtoniterationsschrittes.
- Die Methode ist L-stabil und steifakkurat hinsichtlich der standardmäßigen Berechnung von y_1, welches gleich dem Stufenwert der letzten Stufe ist, i. e., $y_1 = Y_s^1$.
- Der lokale Fehler err_{est}^{loc} berechnet sich effizient über (3.140) und ergibt sich formal via $Y_s^1 - Y_{s-1}^1$. Für die Fehlerkonstanten gilt $C_{err,s} < \frac{1}{2} C_{err,s-1}$, s. a. Bemerkung 3.22 hierzu sowie die Tabellen in Abschnitt 3.4.1.
- Sämtliche Stabilitätsaussagen gelten ausschließlich für den Fall $J = f_0^{(1)}$. Dies ist inhärente Eigenschaft eines $FiterRK$-Ansatzes.
- Pro Zeitschritt nutzen sämtliche Newtonprozesse zur Berechnung der Stufenwerte die gleiche Iterationsmatrix. Diese ist also nicht abhängig vom Stufenindex.
- Die Matrix $\mathcal{A}$ zur Methode ist von der Gestalt (3.71). Da mindestens $\mathcal{C}_\alpha(1)$ gilt, kann auf die Berechnung von $\partial f_0/\partial t$ verzichtet werden, wenn im Block gleich zu Beginn mit den exakten Zeitdaten $t_0 + c_i h$ für $i = 1, \ldots, m$ gerechnet wird.

Bemerkung 3.59. Die Wahl der Ordnungspaare für die konstruierten $FiterRK$-Methoden orientiert sich an dem im ATHLET-Code verwandten Extrapolationsgrad. Dort wird maximal $T_{3,3}$ genutzt.

Bemerkung 3.60. Die in Abschnitt 3.3 hergeleitete Theorie bietet ein sehr leistungsstarkes Werkzeug, um in moderater Zeit ggf. weitere Methoden zu konstruieren, die mindestens der Eigenschaft genügen, dass zum Ordnungspaar $p+1(p)$ die Stufenordnung gleich p ist. Gerade durch die Gültigkeit von Satz 3.42 lassen sich weitere Freiheitsgrade ins Spiel bringen, ohne $\mathcal{C}(p)$ zu verletzen.

Bemerkung 3.61. Alle vier Methoden wurden mit Hilfe des Computeralgebrasystems *Maxima*, /MAX 15/, sowie der numerischen Software *Scilab*, /ENT 16/, konstruiert. Die Koeffizienten und Rechengrößen der Methoden wurden in 40-stelliger Genauigkeit ermittelt, was hinreichend Puffer gegenüber der 16-stelligen Genauigkeit des Testalgorithmus

in Scilab bietet. Es wird an dieser Stelle darauf verzichtet, diese ermittelten Größen tabellarisch aufzulisten, da aus den konkreten Werten wenig Erkenntnisgewinn gezogen werden kann und auch nicht das Maß an Zahlenästhetik vorliegt , wie es für den Extrapoltaionsansatz basierend auf dem linear impliziten Euler der Fall ist. Detailentscheidungen für oder wider bestimmter Methodenwerte sind in den jeweiligen Dateien zur Konstruktion der Methoden zu finden. Diese werden auf Anfrage gern zur Verfügung gestellt.

Bemerkung 3.62. Im Laufe des Projektes ergaben sich immer mehr Informationen dazu, wie die Differentialgleichungsnumerik in ATHLET gehandhabt wird, s. Abschnitt 3.6. Dies motivierte die Konstruktion weiterer Methoden, s. Abschnitt 3.6.2. Diese sind nicht in den Benchmarks in Abschnitt 3.5.2 enthalten, bedienen sich aber der gleichen Konzepte, wodurch die generelle Überlegeneheit zum Extrapolationsansatz – wie durch die anderen Methoden gezeigt – vorhanden sein sollte.

Methoden mit 0-explizitem $\mathcal{A}$

Für die Methoden $FiterRK32a$, $FiterRK32b$ und $FiterRK43$ ist das jeweilige $\mathcal{A}$ 0-explizit; es findet somit keine explizite erste Stufe Anwendung. Mittels /HAI 96, Lemma IV.8.1/ oder im Falle von $FiterRK32b$ mittels Satz 3.42 wird zugesichert, dass $\mathcal{A}$ ein Einpunktspektrum besitzt. Als Grundlage zur Konstruktion der drei Methoden dienen Korollar 3.33 und 3.37. Um die Anzahl der Newtonschritte für die Blockstufen nach oben durch p zu beschränken, findet Satz 3.36 Anwendung. Die Matrix T zur Transformation von Z- auf W-Variablen im Block wurde entsprechend der Diskussion in Unterabschnitt 3.3.4.1 auf Basis der Schurschen Normalform von $A_m - 1$ konstruiert und ist somit orthogonal.

In der Konstruktion der Methoden wurde darauf Wert gelegt, dass bereits die α-Stufenwerte $Y_i^0 = y_0 + z_i^0$, also die Startapproximationen in den Newtonprozessen für die Stufenwerte, möglichst gute Ordnungs- und Stabilitätseigenschaften besitzen. Entsprechend gilt sogar $\mathcal{C}_\alpha(p)$. Informationen zu Anzahl der Stufen, der Blockgröße und der Stabilitäten der einzelnen Stufen finden sich in Tab. 3.14-3.16. Gewisse Besonderheiten sind wie folgt.

- $FiterRK32a$
  Die zugeordnete voll implizite Methode ist S-reduzibel, da in der Newtonasymptotik die zweite Blockstufe das gleiche Ergebnis liefert wie die erste Diagonalstufe. Diese Redundanz kann genutzt werden, um eine weitere Schätzung zur Berechnung von α_N in (3.182) zur Verfügung zu stellen. Die Koeffizienten a_{sj} der letzten Stufe sind derart gewählt, dass zusätzlich (3.170) für die Stabilitätsfunktion der letzten Stufe gilt, s. a. Tab. 3.14.

- $FiterRK32b$

  Diese Methode weist einen Block der Größe $m = 3$ auf bei einer Stufenordnung von $p = 2$. Wie oben bereits erwähnt wird Satz 3.42 genutzt, damit $\mathcal{A}$ genau einen verschiedenen Eigenwert $\gamma = a_{ii}$, $i = 4, \ldots, 6$, besitzt. Die skalierten Schrittweiten $c_1/\gamma, c_2/\gamma, c_3/\gamma$ liegen auf der in Abb. 3.3 dargestellten algebraischen Kurve für

  $$\varphi_1 = -3.93267796610169507,$$
  $$\varphi_2 = -1.34433898305084742,$$
  $$\varphi_3 = 1.07857627118644062.$$

  Die Wahl der φ_i resultiert aus der Forderung, alle Blockstufen A-stabil zu machen, Positivität der Stabilitätsfunktionen im Sinne von (3.170) zu garantieren und die Fehlerkonstanten moderat zu halten. Die weiteren Freiheitsgrade wurden genutzt, um diese Eigenschaften auch für die restlichen Stufen sicherzustellen.

- $FiterRK43$

  Da der klassische Ansatz genutzt wurde, um ein Einpunktspektrum für $\mathcal{A}$ zu ermöglichen, können die erste und dritte Blockstufe nicht A-stabil konstruiert werden. Wie in /BUT 90/ gezeigt wird, sind mindestens drei Diagonalstufen nötig, um bei Wahrung von $\mathcal{C}(3)$ A-Stabilität für den vollen Schritt sicherzustellen, ohne auf ein $c_i > 1$ zugreifen zu müssen. Dass die sechs Stufen der Ordnung Drei durch eine L-stabile Stufe der Ordnung Vier erweitert werden können, wurde erst durch die Konstruktion von $FiterRK43$ in dieser Arbeit nachgewiesen.

Methode mit 1-explizitem $\mathcal{A}$ – $FiterRK32ex$

Als einzige Methode mit einer expliziten ersten Stufe, also mit einem 1-expliziten $\mathcal{A}$, ist $FiterRK32ex$ konstruiert worden. Da nur eine Stufenordnung von Zwei berücksichtigt wird und auf f_0 zurückgegriffen werden kann, ist der Block von trivialer Größe. Jede Stufe kann also als Diagonalstufe interpretiert werden, und ein Spektrum der Form $\mathrm{spec}(\mathcal{A}) = \{0, \gamma\}$ resultiert aus der Wahl $a_{ii} = \gamma$ für $i > 1$. Die zugeordnete voll implizite Methode ist somit eine ESDIRK-Methode. Konkret ergibt sich *ESDIRK 3/2 in 4 stages* aus /KVÆ 04/ mit der dortigen Wahl a) für den Parameter γ. Die in dieser Arbeit hergeleitete $FiterRK$-Version etabliert die Stufenordnung von Zwei mittels Anwendung von Satz 3.32. Für ein Abgleich in der dritten Ableitungsordnung bzgl. der letzten Stufe wurde das in Satz 3.38 beschriebene Konzept der Diagonalverflechtung für die zweite und dritte Stufe genutzt. Somit muss lediglich ein Newtonschritt pro Stufenwertberechnung erfolgen und $FiterRK32ex$ kann auch als klassische W-Methode interpretiert werden.

Die Hauptmotivation zur Konstruktion von $FiterRK32ex$ liegt darin, mit möglichst wenig

Aufwand eine Methode zum Ordnungspaar $p+1(p)$ für $p=2$ zu erzeugen, welche auch $\mathcal{C}(p)$ genügt und möglichst wenig Newtonschritte ausführt. Hierbei hilft die Hinzunahme einer ersten expliziten Stufe. Allerdings ist es deutlich schwieriger Stabilitätsbedingungen zu genügen, selbst für $J = f_0^{(1)}$, wie es die Spalten *A-stab* und $R(\infty)$ in Tab. 3.17 zeigen. Aus der Not eine Tugend machend, kann die $FiterRK32ex$-Methode auch als Prüfstein gesehen werden, ob sich geringe Stabilitätseigenschaften als merklicher Nachteil entpuppen.

Man beachte, dass nativ keine Möglichkeit besteht, eine Schätzung für α_N aus (3.182) zu berechnen, da die Newtonschritte verschiedene Stufen abdecken. Entweder führt man einen weiteren Iterationsschritt durch, oder man beschränkt sich komplett auf die anderen in Unterabschnitt 3.5.1.3 erläuterten Strategien zum Update der Jacobimatrix.

3.4.1 Tabellarische Darstellung

Die nachfolgenden Tabellen stellen in kompakter Weise wesentliche Informationen zu den vier konstruierten Methoden dar. Die Spalten sind wie folgt semantisch belegt:

- i: Angabe des Stufenindex in einer ganzeinheitlichen Zählung
- $c_i \approx$: relative Schrittweite; der Stufenwert $Y_i^{l_i}$ approximiert $y(t_0 + c_i h)$
- Stufe : *Bx* oder *Dx* bezeichnet die x-te Block- bzw. Diagonalstufe in einer lokalen Zählung. Ein hintergestelltes α bezieht sich auf den α-Stufenwert $Y_i^0 = y_0 + z_i^0$, wobei i den Index in der ganzeinheitlichen Zählung wiedergibt. Ist eine doppelte Angabe wie Dk|Djα für Stufe i zu sehen, so ist der Stufenwert $Y_i^{l_i}$ gleich dem α-Stufenwert zur Stufe, die mit Dj gekennzeichnet ist.
- A-stab: Gibt an, ob unter der Prämisse $J = f_0^{(1)}$ durch $Y_i^{l_i}$ bzw. Y_i^0 eine A-stabile Approximation von $y(t_0 + c_i h)$ gegeben ist.
- $R(\infty)$: Gibt Auskunft über $\lim_{z\to\infty} R(z)$, wobei R die Stabilitätsfunktion zur betrachteten Stufe oder α-Stufe ist.
- StA: Spiegelt wieder, ob unter der Prämisse $J = f_0^{(1)}$ durch $Y_i^{l_i}$ bzw. Y_i^0 eine steifakkurate Approximation von $y(t_0 + c_i h)$ gegeben ist. Falls $|R(\infty)| \geqslant 1$ gilt, ist etwaige Steifakkuratesse nur lokal vorhanden s. auch Bemerkung 3.18.
- $R(x<0)>0$: Dies ist eine Kurzschreibweise für

  $$\forall x \in \mathbb{R}_- : R(x) > 0. \tag{3.170}$$

  Ist ein Haken gesetzt, gilt (3.170). Andernfalls gibt es mindestens ein $x \in \mathbb{R}_-$ mit $R(x) \leqslant 0$.

- Niter: Anzahl der Newtoniterationen, um den Ordnungseigenschaften zu genügen. Hinsichtlich der Blockstufen wird hierür von einer Startnäherung $Z_m^0 = 0$ in (3.92) ausgegangen. Ebenso wird im Falle nicht-autonomer Differentialgleichungen davon ausgegangen, dass von Beginn an mit den exakten Zeitdaten $t_0 + c_i h$ gearbeitet wird.

Tab. 3.14 Stufeneigenschaften zur $FiterRK32a$-Methode

i	$c_i \approx$	Stufe	A-stab	$R(\infty)$	StA	$R(x<0)>0$	Niter
1	0.154	B1	✓	0	✓	✓	2
2	0.896	B2|D1α	✓	0	✓	–	2
3	0.896	D1	✓	0	✓	–	1
4	1	D2α	✓	$\approx \frac{16}{25}$	–	–	
		D2|D3α	✓	0	✓	–	1
5	1	D3	✓	0	✓	✓	1

$\mathcal{C}(2)$ und $\mathcal{C}_\alpha(2)$ sind erfüllt,

$C_{err}^{D2} \approx 3.84 \cdot 10^{-2}$, $C_{err}^{D3} \approx 5.75 \cdot 10^{-3}$, $\frac{C_{err}^{D3}}{C_{err}^{D2}} \approx 1.5 \cdot 10^{-1}$

Tab. 3.15 Stufeneigenschaften zur $FiterRK32b$-Methode

i	$c_i \approx$	Stufe	A-stab	$R(\infty)$	StA	$R(x<0)>0$	Niter
1	0.084	B1	✓	0	✓	✓	2
2	0.224	B2	✓	0	✓	✓	2
3	1	B3	✓	0	✓	✓	2
4	0.871	D1α	✓	$\frac{3}{4}$	–	✓	
		D1	✓	0	✓	✓	1
5	1	D2α	✓	0	–	✓	
		D2|D3α	✓	0	✓	✓	1
6	1	D3	✓	0	✓	✓	1

$\mathcal{C}(2)$ und $\mathcal{C}_\alpha(2)$ sind erfüllt,

$C_{err}^{D2} \approx 1.13 \cdot 10^{-2}$, $C_{err}^{D3} \approx 2.88 \cdot 10^{-4}$, $\frac{C_{err}^{D3}}{C_{err}^{D2}} \approx 2.5 \cdot 10^{-2}$

Tab. 3.16 Stufeneigenschaften zur $FiterRK43$-Methode

i	$c_i \approx$	Stufe	A-stab	$R(\infty)$	StA	$R(x<0)>0$	Niter
1	0.066	B1	–	0	✓	✓	3
2	0.365	B2	✓	0	✓	–	3
3	1	B3	–	0	✓	✓	3
4	0.45	D1α	✓	$\approx \frac{3}{4}$	–	✓	
		D1	✓	0	✓	–	1
5	0.711	D2α	✓	0	–	✓	
		D2	✓	0	✓	✓	1
6	1	D3α	–	0	–	✓	
		D3|D4α	✓	0	✓	✓	1
7	1	D4	✓	0	✓	✓	1

$\mathcal{C}(3)$ und $\mathcal{C}_\alpha(3)$ sind erfüllt,

$C_{err}^{D3} \approx 1.46 \cdot 10^{-3}, \qquad C_{err}^{D4} \approx 4.32 \cdot 10^{-4}, \qquad \frac{C_{err}^{D4}}{C_{err}^{D3}} \approx 3 \cdot 10^{-1}$

Tab. 3.17 Stufeneigenschaften zur $FiterRK32ex$-Methode

i	$c_i \approx$	Stufe	A-stab	$R(\infty)$	StA	$R(x<0)>0$	Niter
1	0	B1					
2	0.872	B2α	–	∞	–	–	
		B2	✓	-1	✓	–	1
3	1	D1α	–	∞	–	–	
		D1|D2α	✓	$\approx -\frac{24}{25}$	✓	–	1
4	1	D2	✓	0	✓	–	1

$\mathcal{C}(2)$ und $\mathcal{C}_\alpha(1)$ sind erfüllt,

$C_{err}^{D1} \approx 7.92 \cdot 10^{-2}, \qquad C_{err}^{D2} \approx 2.4 \cdot 10^{-2}, \qquad \frac{C_{err}^{D2}}{C_{err}^{D1}} \approx 3 \cdot 10^{-1}$

3.5 Testalgorithmus

Um eine Einschätzung über die tatsächliche Qualität der in Abschnitt 3.4 vorgestellten Methoden im Vergleich zum Euler-Extrapolationsansatz zu erhalten, ist ein in Scilab, /ENT 16/, programmierter Testalgorithmus konstruiert worden. In diesem wurden die einzelnen Methoden auf verschiedene aus der Literatur bekannte steife Probleme angewandt, s. Abschnitt 3.5.2. Zeitgleich dient der Testalgorithmus als Blaupause zur Konstruktion des

eigentlichen ATHLET-bezogenen Lösers, was enorme Zeitersparnis in der Implementierung mit sich bringen sollte.

Der Testalgorithmus setzt sich im Wesentlichen aus den folgenden Teilalgorithmen zusammen:

- Algorithmus 3.2 und 3.3
  Berechnung der Stufenwerte für Block- und Diagonalstufen
- Algorithmus 3.5 und 3.6
  Zusatzberechnungen bei Einsatz eines optionalen T2-Updates
- Algorithmus 3.4
  Berechnung von y_1 und der vektoriellen Fehlerschätzung
- Algorithmus 3.7 und 3.8 Schrittweiten- und Jacobimatrixkontrolle, s. a. Abschnitt 3.5.1

Bemerkung 3.63. Der Scilab-Algorithmus ist als experimenteller Code einzustufen und dient hauptsächlich dazu, numerische Aspekte und Methoden zu beurteilen. Losgelöst vom ATHLET-Kontext sind numerische Schwierigkeiten und Problemzonen besser identifizierbar. Der Scilab-Algorithmus wird nicht produktiv eingesetzt und ist auf Anfrage zugänglich.

Der Übersicht halber sind gewisse Aspekte des Testalgorithmus nicht in den referenzierten Algorithmen oder in Abschnitt 3.5.1 im Detail erläutert und werden im Folgenden kurz stichpunktartig diskutiert:

- Ausnahmebehandlung bei Nichtdefiniertheit von $f(y)$
  Der Testalgorithmus terminiert mit einer Fehlermeldung in diesem Fall. Dies ist im Rahmen des Testszenarios opportun. Für eine Adaption in den ATHLET-Kontext ist hier auf die höhere Logikebene zu wechseln. Man beachte, dass dies die einzige Situation ist, in der der Differentialgleichungslöser eine Ausnahme generieren kann, die er nicht selbst handhaben kann.
- Änderungen an J oder h und das T2-Update
  Der über (3.155) definierte Vektor $\widetilde{u}$ ist direkt von h und J abhängig und muss bei Änderung mindestens einer dieser beiden Größen neu berechnet werden. Eine Ausnahme besteht, wenn das neue J als exakt angesehen wird. Dann bedarf es entsprechend der Diskussion in Unterabschnitt 3.3.5.1 keines T2-Updates. Zur möglichen Qualitätskontrolle der Iterationsmatrix über Informationen aus dem T2-Update s. Unterabschnitt 3.5.1.3.

- System-Skalierung

  Mit diesem Begriff ist an dieser Stelle eine adaptive und komponentenweise Skalierung des y-Vektors gemeint. Statt also eine Komponente $y_{(i)}$ zu betrachten, wird mit

  $$y_{(i)}^{scl} := y_{(i)} / \max(TOL_{scl}, |y_{0,(i)}|) \tag{3.171}$$

  gearbeitet, wobei $y_{0,(i)}$ die i-te Komponenten von y_0 im Zeitschritt $t_0 + h$ bezeichnet und TOL_{scl} eine notwendige Abgrenzung gegen Null ist, i. e., $0 < TOL_{scl} \ll 1$. Auf Vektorebene lässt sich die Umskalierung darstellen als

  $$y_{sc} = D_{sys}^{-1} y \qquad \text{mit} \quad D_{sys} = \operatorname{diag}\Big(\max(TOL_{scl}, |y_{0,(1)}|), \ldots, \max(TOL_{scl}, |y_{0,(n)}|)\Big). \tag{3.172}$$

  Damit ergibt sich

  $$y' = f(y) \qquad \Leftrightarrow \qquad y'_{scl} = D_{sys}^{-1} f(D_{sys} y_{scl}) =: f_{scl}(y_{scl}). \tag{3.173}$$

  Die Verwendung einer Systemskalierung bietet den Vorteil der Skalierungsinvarianz (sofern TOL_{scl} nicht greift): Der numerische Algorithmus zeigt das gleiche Verhalten, auch wenn Variablen eine Einheiten-Redimensionierung erfahren, e. g., von $[s]$ zu $[h]$ oder von $[m]$ zu $[mm]$. Auf der anderen Seite geht Systemskalierung mit Mehrarbeit einher, da die Skalierung explizit in f-Auswertungen oder Jacobimatrizen berücksichtigt werden muss:

  $$f(\cdot) \quad \rightarrow \quad D_{sys}^{-1} f(D_{sys} \cdot) \overset{(3.173)}{=} f_{scl}(\cdot) \qquad \text{sowie} \qquad J \quad \rightarrow \quad D_{sys}^{-1} J D_{sys} =: J_{scl}. \tag{3.174}$$

  Werden diese Transformationen transparent gehalten, ändert sich nichts am Ablauf des Testalgorithmus. Einzige Ausnahme besteht in der notwendigen Rück- oder Umskalierung bei der Auswertung der Fehlernorm $\|\cdot\|_{err}$, da hierbei die vom Nutzer vorgegebenen Toleranzen zu den einzelnen y-Komponenten im *unskalierten* System eingehen, vgl. (3.178). Man beachte, dass die über die linearen Systeme in Zeile 13 von Alg. 3.2 und Zeile 18 von Alg. 3.3 berechneten Inkremente ebenfalls systemskaliert sind. Folglich gilt dies für alle berechneten W- und Z-Werte und eine Auswertung von $\|\cdot\|_{err}$ führt wegen der Rückskalierung zu einem von der Systemskalierung unabhängigen Ergebnis.

3.5.1 Schrittweiten- und Jacobimatrixkontrolle

Dieser Teil des Testalgorithmus basiert im Wesentlichen auf den Ideen in /GUS 97/ mit Verfeinerungen aus /VÖL 10/. Beide Quellen legen voll implizite Methoden zu Grunde. Im

Alg. 3.7 Schrittweiten- und Jacobimatrixkontrolle zur Schrittweite h bei Standardterminierung der Algorithmen zur Stufen- und Fehlerberechnung

Data : Vektor und Ordnung der lokalen Fehlerschätzung err_{est}^{loc} bzw. p; Kontraktionsschätzung α_N aus (3.182); Schrittweite h; diverse Konstanten

set $r_{cur} = \|err_{est}^{loc}\|_{err}$; `// s. Abschnitt 3.5.1.1`

set $h_r = h \cdot \left[\frac{\varepsilon}{r_{cur}}\right]^{1/(p+1)}$; `// Standardregler/Korrektor` H_0110, `s. Tab. 3.18`

set $h_\alpha = (\alpha_{ref}/\alpha_N)^{1/(p+1)} h$;

if $r_{cur} < \texttt{TOL}_{err}$ **then** `// Schritt akzeptiert`

  set `stepaccepted` $= 1$;

  if `steprestr` **then** `// s. Alg. 3.8`

    set $h_r = \min(1, h/h_{prev}) \cdot h_r$; `// zusätzliche Dämpfung, vgl. /VÖL 10/`

  else `// berechne Prädiktor`

    set $h_r =$ `calc_hpred`$(h_{prev}, h, h_r, r_{cur}, r_{prev}, p)$;

else if J *ist nicht exakt* **then** `//` $FiterRK$`-spezifisch`

  Berechne Spektralschätzung zu J, s. Abschnitt 3.5.1.3. Bei Diskrepanz konstruiere ein besseres J. Terminiere und starte Alg. 3.2 mit *unverändertem* h.

set $h_{new} = \min(h_r, h_\alpha)$;

Setze `hfacmax` nach der Prioritätenreihenfolge in Unterabschnitt 3.5.1.2;

`// Anwendung von absoluten und realtiven` min`- und` max`-Limitern`

set $h_{new} = \min(\max(\min(h_{new}, h \cdot$ `hfacmax`$), h \cdot$ `hfacmin`$, h_{min}), h_{max})$;

if $\alpha_N - |h - h_{LU}|/h_{LU} > \alpha_{Jac}$ **then**

  if `stepaccepted` **then**

    Konstruiere neues J zu Beginn des nächsten Zeitschrittes;

  else if J *ist nicht exakt* **then**

    Konstruiere besseres J. Terminiere und starte Alg. 3.2 mit $h = h_{new}$;

else if $h_{new} < c_1 h_{LU}$ *oder* $h_{new} > c_2 h_{LU}$ **then** `//` $0 < c_1 < 1 \leqslant c_2 < 2$

  Zerlege Iterationsmatrizen aus (3.97) bzw. (3.120) mit $h = h_{new}$, bevor das nächste lineare Gleichungssystems gelöst wird. Setze dann $h_{LU} = h_{new}$;

if `stepaccepted` **then**

  Berechne Spektralschätzung zu J. Bei Diskrepanz konstruiere neues J zu Beginn des nächsten Zeitschritts; `//` $FiterRK$`-spezifisch`

  set $h_{prev} = h$ sowie $r_{prev} = r_{cur}$;

  Gehe in den nächsten Zeitschritt mit $h = h_{new}$;

else `// Wiederholung des jetzigen Schrittes für neues` h

  Terminiere und starte Alg. 3.2 mit $h = h_{new}$

Alg. 3.8 Schrittweiten- und Jacobimatrixkontrolle zur Schrittweite h bei Divergenz im Newtonprozess, i. e., es gilt $\alpha_N > 1$ mit α_N aus (3.182)

Data : Schrittweite h

1 **if** J *frisch konstruiert oder exakt* **then**

2 **set** $h_{new} = \min(\frac{1}{2}h, h_\alpha)$; `// h_α wie in Z. 3 von Alg. 3.7`

3 **set** `steprestr` $= 1$;

4 starte Alg. 3.2 mit $h = h_{new}$

5 **else**

6 Konstruiere besseres J und starte Alg. 3.2 mit unverändertem h;

$FiterRK$-Kontext ist die Güte der Jacobimatrixapproximation nicht nur für die Konvergenz des Newtonprozesses relevant, sondern auch hinsichtlich Stabilität. Aus diesem Grund wird zusätzlich in Alg. 3.7 in Zeile 10 und Zeile 10 eine Spektralschätzung durchgeführt, für Details hierzu s. Unterabschnitt 3.5.1.3. In der Schrittweitensteuerung basiert die Prädiktion auf regelungstechnischen Überlegungen aus /GUS 94/,/SÖD 02/,/SÖD 03/. Dieser Teil des Algorithmus ist am ehesten als experimentell zu betrachten und erfährt ggf. eine Verfeinerung in der Art des regelungstechnischen Ansatzes, da z. B. nach aktuellem Stand der Recherche W-Methoden nicht explizit in der Literatur Berücksichtigung finden. Auch die Auswahl der Schrittweitenprädiktoren wird evtl. vergrößert oder geändert werden.

Innerhalb der generellen Schrittweiten- und Jacobimatrixkontrolle sind α_{ref} und α_{Jac} wesentliche Steuerparameter:

Mittels α_{ref} wird die Referenz-Kontraktionsrate im Newtonprozess beschrieben. Nach der Analyse in /GUS 97/ sollte $\alpha_{ref} \in [0.2, 0.4]$ gelten, um die Anzahl der Newtoniterationsschritte l pro Einheitsschrittweite, i. e. l/h, in einem gewissen Sinne optimal zu halten. Der Parameter α_{ref} taucht im Algorithmus nur zur Berechnung von h_α auf (ebenso in den zwei genannten Quellen). Es ist legitim, auch im $FiterRK$-Kontext auf diesen Steuerwert zurückzugreifen, da sich zum einen im Vergleich zu einer voll impliziten Methode das grundsätzliche Vorgehen nicht ändert. Nur die (maximale) Anzahl der Newtonschritte ist a priori bekannt. Zum anderen wird α_{ref} zur Anpassung der Schrittweite, jedoch nicht für eine Entscheidung, ob eine neue Matrix J konstruiert werden soll, herangezogen. Die Berechnung von h_α erfolgt in der in /VÖL 10/ vorgeschlagenen gefilterten Form. Dies verhindere Fluktuationen durch ein etwaiges zu starkes Herabsetzen der Schrittweite, wie es der Fall sei, wenn die in /GUS 97/ proklamierte dead-beat-Regelung $(\alpha_{ref}/\alpha_N)h$ Anwendung fände.

Eine gleichzeitige Berücksichtigung der lokalen Nichtlinearität und der Güte der Approxi-

mation J findet über α_{Jac} in Zeile 15 von Alg. 3.7 statt. Zur Herleitung der Ungleichung s. /GUS 97, § 4/. Dort wird $\alpha_{ref} = \alpha_{Jac}$ empfohlen. Um den speziellen $FiterRK$-Kontext zu berücksichtigen, wird an dieser Stelle ein deutlich kleinerer Wert $\alpha_{Jac} \in [10^{-2}, \frac{1}{2}10^{-1}]$ genutzt, um eine angemessene Sensibilität hinsichtlich der Qualität der Jacobimatrix zu wahren. Dieser Ansatz geht zurück auf eine Idee von Deuflhard, s. /DEU 85/.

Man beachte, dass es für die Abfrage in Zeile 15 von Alg. 3.7 irrelvant ist, ob h größer oder kleiner h_{LU} ist, um den Einfluss der relativen Diskrepanz auf die Konvergenzrate zu schätzen. Dies ist nicht der Fall für den Test in Zeile 20 – das Kriterium zur Berechnung einer neuen Zerlegung ist asymmetrisch. Dies spiegelt die Vorgehensweise in /HAI 96, § IV.8/ wieder und begründet sich damit, dass es im Kontext eines steifen Systems eher tragbar ist, mit $h < h_{LU}$ zu arbeiten, da dann der Einfluss der Steifheit in der Iterationsmatrix nicht unterschätzt wird. Dies passiert dann, wenn $h > h_{LU}$ gilt. Generell sollten sich im Kontext der $FiterRK$-Methoden c_1 und c_2 sehr nah bei 1 befinden. Für den Wert von c_2 ist $c_2 = 1$ zu empfehlen.

Bemerkung 3.64. Wird ein iteratives Verfahren zur approximativen Lösung der Systeme in Zeile 13 und Zeile 18 von Alg. 3.2 bzw. Alg. 3.3 genutzt, so beziehen sich Aussagen zu Zerlegungen der Iterationsmatrix und die Schrittweite h_{LU} auf den Präkonditionierer, sofern dieser als direkter Löser gegeben ist. Sollte dies nicht der Fall sein, so ist dieser Aspekt des Algorithmus nicht zu beachten.

3.5.1.1 Newton- und Fehlermodell

Der in Abschnitt 3.3 detailliert dargestellte Newtonprozess setzt eine gewisse Glattheit von f im Zusammenspiel mit der Schrittweite h voraus, um wohldefiniert zu sein und zu konvergieren. Im Wesentlichen muss hierfür das Startinkrement oder das Startresiduum hinreichend gut sowie eine affin invariante Lipschitzbedingungen an $I - h\partial f(y)/\partial y$ erfüllt sein. Gilt $J \neq f_0/\partial y$, so ist $M = I - hJ$ als invertierbar angenommen und in besagte Lipschitzbedingungen einzubinden sowie ein beschränktes affin invariantes Maß für den Abstand zwischen M und $I - h\partial f(y)/\partial y$ zu fordern. Es ist nicht schwierig, in einer Kombination aus Theorem 2.5 und Theorem 2.6 in /DEU 04/ eine affin kovariante, d. h. *fehler*orientierte, lokale Konvergenzaussage herzuleiten, wenn $J \neq f_0/\partial y$ gilt. Für $J = f_0/\partial y$ findet direkt Theorem 2.5 Anwendung. In einem affin kontravarianten Ansatz, i. e. in einer *residuen*bezogenen Sichtweise, ist Theorem 2.13 in /DEU 04/ zu adaptieren. Berücksichtigung affiner Invarianz geht auf Deuflhard zurück, s. z. B. /DEU 04/, und führt Theorie und Praxis dichter zusammen als klassische Ansätze.

Fehlerorientierung spiegelt sich darin wieder, dass Konvergenzmaße und Abbruchkriterien in Z- und ΔZ-Entitäten aus (3.87) oder entsprechenden Derivaten hiervon, i. e. (3.92) und (3.117b), formuliert werden. Eine residuenbasierte Vorgehensweise bedient sich in analoger Weise der rechten Seiten, i. e. Residuen, der just erwähnten Newtonvorschriften. In diesem Abschnitt wird das Residuum vereinfacht mit `res` bezeichnet.

Im Sinne der Konvergenzordnung ist es einerlei, welche Sichtweise man bevorzugt. Mit der gegebenen Qualität von M kann nur lokal q-lineare Konvergenz mit einer Kontraktionskonstanten $0 < \overline{\alpha}_N < 1$ erwartet werden. Die Größe von $\overline{\alpha}_N$ kann sich aber erheblich unterscheiden! Nach /HOU 85/ impliziert im Rahmen von steifen Differentialgleichungen ein kleines Residuum einen kleinen Fehler. Dafür werden für voll implizite Verfahren in der Regel mehr Iterationsschritte benötigt, um das Residuum hinreichend klein zu bekommen, als wenn man auf Basis der Fehlerschätzung r_{cur} arbeitet (bei gleichem Toleranzwert). Ein Grund hierfür kann darin liegen, dass die Jacobimatrizen im gegebenen Kontext in der Regel schlecht konditioniert sind, /SHA 93/, was zu stark achsenverzerrten Niveaumengen bzgl. der Norm des Residuums führt, vgl. /DEU 04, § 3/. Auch kann dies verantwortlich dafür zeichnen, dass der Konvergenzradius für den residuenbasierten Ansatz merklich kleiner ist – das Residuum wächst, obschon sich bzgl. der Inkremente Kontraktion zeigt.

Zwar wird in dieser Arbeit ein Finite-Iteration-Ansatz diskutiert, jedoch sind Bedingungen für einen vorzeitigen Abbruch oder Konvergenzschätzungen auch in diesem Kontext von Relevanz. Insofern ist zu diskutieren, inwiefern die zwei vorgestellten Sichtweisen Beachtung finden. In /VÖL 10/ wird sich des residuenbasierten Ansatzes bedient, wohingegen in /HAI 96, § IV.8/ der fehlerorientierte Weg beschrieben wird. Für diese Arbeit wurde sich für einen hybriden Ansatz entschieden, wobei das Hauptaugenmerk auf der fehlerorientierten Sicht liegt. Wegen der oben beschriebenen Schwächen des Residuenansatzes hinsichtlich der Kontraktionseigenschaften wird diesbezüglich mit Z- und ΔZ-Entitäten gearbeitet. Sei durch obige Sätze die Existenz einer Lösung $Z^\star$ zum Newtonprozess gesichert, so folgt aus q-linearer Konvergenz die Beziehung

$$\|\Delta Z^{l+1}\|_{err} \leqslant \overline{\alpha}_N \|\Delta Z^l\|_{err} \qquad \forall l.$$

Mit Hilfe der Dreiecksungleichung und der geometrischen Reihe ergibt sich dann

$$\|Z^{l+1} - Z^\star\|_{err} \leqslant \frac{\overline{\alpha}_N}{1 - \overline{\alpha}_N} \|\Delta Z^l\|_{err}. \tag{3.175}$$

Die Konstante $\overline{\alpha}_N$ kann über die Entitäten

$$[\alpha_N]_l = \frac{\|\Delta Z^l\|_{err}}{\|\Delta Z^{l-1}\|_{err}} \qquad \text{für} \quad l \geqslant 1 \tag{3.176}$$

geschätzt werden. Den Zusammenhang (3.175) ausnutzend ergibt sich die in /HAI 96, Eq. (8.10)/ dargestellte Abbruchbedingung

$$\eta_l \|\Delta Z^l\|_{err} \leqslant \nu_{err} \cdot \mathtt{TOL}_{err} \qquad \text{mit} \quad \eta_l = \frac{[\alpha_N]_l}{1-[\alpha_N]_l} \qquad \text{und für} \quad l \geqslant 1.$$

Z^{l+1} gilt dann als finale Approximation für $Z^\star$. In /HAI 96/ wird mit $\nu_{err} \in [10^{-2}, 10^{-1}]$ gearbeitet. Im $FiterRK$-Kontext ist es essentiell, nicht zu früh abzubrechen. Aus diesem Grund wird η_l in obiger Darstellung durch $\max(\eta_l, 1)$ ersetzt, was äquivalent zur konservativen Annahme $[\alpha_N]_l \geqslant 1/2$ ist. Zusätzlich wird das Residuum berücksichtigt, so dass sich die im $FiterRK$-Kontext genutzte Bedingung für einen vorzeitigen Abbruch zu

$$\begin{aligned} &\max(\eta_l, 1)\|\Delta Z^l\|_{err} \leqslant \nu_{err} \cdot \mathtt{TOL}_{err} \\ &\text{oder} \qquad\qquad\qquad\qquad \text{für} \quad l \geqslant 1 \\ &\|\mathtt{res}_{l+1}\|_{err} \leqslant \nu_{res} \cdot \mathtt{TOL}_{err} \end{aligned} \tag{3.177a}$$

mit

$$\nu_{err} = 2^{-3} \cdot 10^{-1} \qquad \text{und} \qquad \nu_{res} = 10^{-1} \tag{3.177b}$$

ergibt. Die Werte der Konstanten ν_{err} und ν_{res} sind konform zu /HAI 96/ bzw. /VÖL 10/ gewählt und beziehen die Ergebnisse der Diskussion in /HOU 85/ über $\nu_{err} < \nu_{res}$ mit ein.

Bemerkung 3.65. Es ist legitim, in (3.177a) η_l ausschließlich auf dem aktuellen Wert $[\alpha_N]_l$ basieren zu lassen, da jede Newtoniterierte als Startpunkt zu einem neuen Newtonprozess interpretiert werden kann, welcher wiederum durch ein eigenes Kontraktionsmaß charakterisiert wird.

Bemerkung 3.66. In /GUS 97/ wird diskutiert, ν_{err} methodenabhängig zu wählen. Bei einem Abbruch der Newtoniteration kann sich der Abbruchfehler durch $\|b^T\mathcal{A}^{-1}\|$ verstärkt auf r_{cur} auswirken. Der hier genutzte Standardwert sollte für alle betrachteten Methoden hinreichend viel Puffer bieten, dass bei einer vorzeitigen Beendigung des Newtonprozesses, der Abbruchfehler nicht den eigentlichen Schrittfehler r_{cur} dominiert.

Die Abfrage (3.177) ist erst nach mindestens zwei Iterationsschritten möglich, da sonst η_l nicht zur Verfügung steht. Bereits zum Start oder nach einer Iteration steht jedoch das Residuum $\mathtt{res}_0$ bzw. zusätzlich $\mathtt{res}_1$ zur Verfügung. Unabhängig davon, wie gut $\mathtt{res}_0$ ist, wird mindestens eine Iteration ausgeführt, um den impliziten Charakter der Methode zu wahren. Nach einer Iteration wird mit

$$\nu_{lin} = \min\Big(\nu_{cap}/\min(rtol), 10^{-1}\Big) \qquad \text{und} \qquad \nu_{cap} = 10^{-11} \tag{3.177c}$$

das Residuum $\mathtt{res}_1$ überprüft, i. e.

$$\|\mathtt{res}_1\|_{err} \leqslant \nu_{lin} \cdot \mathtt{TOL}_{err}. \tag{3.177d}$$

Somit wird grob ein Residuum mindestens von der Größenordnung ν_{cap} gefordert. Dieser Test ist besonders hilfreich im Falle linearer Systeme, sofern die Iterationsmatrix M auf exaktem J und aktuellem h basiert. Denn dann gilt in exakter Arithemtik $\mathtt{res}_1 = 0$. Der Newtonprozess kann also bereits nach einem Schritt abgebrochen werden. Dies wäre nicht mit dem gleichen Maß an Gewissheit im Falle eines rein fehlerorientierten Vorgehens möglich – hinsichtlich Heuristiken hierzu s. z. B. /HAI 96, § IV.8/.

Fehlernorm $\|\cdot\|_{err}$

Es ist sinnvoll und üblich, s. z. B. /HAI 93/,/SHA 94/,/GUS 94/,/VÖL 10/ ein komponentenweises Fehlermaß zu betrachten. Dies kann in vektorieller Form geschehen, wenn eine adäquate Skalierung eingebracht wird. Dies geht einher mit $\mathtt{TOL}_{err} \leqslant 1$, s. a. Bemerkung 3.69 zur Wahl von $\mathtt{TOL}_{err}$.

Seien mit $atol$ und $rtol$ in $\mathbb{R}^n$ die Vektoren zur komponentenweisen absoluten bzw. relativen Fehlertoleranz gegeben. Eine recht allgemeine Darstellung zur Fehlernorm eines Vektors $\xi \in \mathbb{R}^{\mu \cdot n}$ ergibt sich dann zu

$$\|\xi\|_{err} = \left\| D_{err}^{-1} \cdot \xi \right\|_{\psi}, \qquad \psi \in \{2, \infty\}, \tag{3.178a}$$

mit

$$\begin{aligned} D_{err} &= \zeta \cdot \Big(I_\mu \otimes \operatorname{diag}(atol) + \Big(I_\mu \otimes \operatorname{diag}(rtol) \Big) \cdot \mathtt{Y} \Big), \\ \zeta &= \begin{cases} \mu \cdot n & \text{falls } \psi = 2 \\ 1 & \text{falls } \psi = \infty, \end{cases} \end{aligned} \tag{3.178b}$$

wobei I_μ die Einheitsmatrix in $\mathbb{R}^{\mu \times \mu}$ bezeichnet und $\mathtt{Y}$ eines der folgenden y-Maße darstellt:

$$\begin{aligned} &\text{fix: } \mathtt{Y} = e_\mu \otimes |y_0|, \qquad \text{Schnitt: } \mathtt{Y} = \begin{pmatrix} \frac{1}{2}(|y_0| + |y_{cur_1}|) \\ \vdots \\ \frac{1}{2}(|y_0| + |y_{cur_\mu}|) \end{pmatrix}, \\ &\text{max: } \mathtt{Y} = \begin{pmatrix} \max(|y_0|, |y_{cur_1}|) \\ \vdots \\ \max(|y_0|, |y_{cur_\mu}|) \end{pmatrix}, \quad \text{voll adaptiv: } \mathtt{Y} = \begin{pmatrix} |y_{cur_1}| \\ \vdots \\ |y_{cur_\mu}| \end{pmatrix}. \end{aligned} \tag{3.178c}$$

Die Operationen $|\cdot|$ und $\max(\cdot, \cdot)$ sind komponentenweise anzuwenden.

Sei $\mathcal{A}$ k-explizit mit $k \in \{0, 1\}$ und von der Gestalt (3.71). Sei mit l der aktuelle Iterationsindex bezeichnet. Dann ergeben sich folgenden Konkretisierungen

- Newtonprozess im Block für $m \geqslant 2 + k$, vgl. (3.92) bzw. (3.98)
  Hier gilt $\mu = m - k$, und y_{cur_j} ergibt sich zu $y_{cur_j} = y_0 + z^l_{j+k}$, $j = 1, \ldots, m - k$.
- Newtonprozess in Stufe i – Diagonal- oder triviale Blockstufe, vgl. (3.117b)
  Es ist $\mu = 1$ und $y_{cur_1} = y_0 + z^l_i$.
- Schätzung des lokalen Fehlers
  Auch hier gilt $\mu = 1$ und y_{cur_1} wird zu $y_{cur_1} = y_1$ gesetzt.

Ist `Y` nicht zur festen Skalierung gewählt, so ist die Fehlernorm *adaptiv*. Offensichtlich steht y_1 während der Newtonprozesse noch nicht zur Verfügung, so dass stattdessen obige iterativ generierte Approximationen für den Grenzwert der Newtonprozesse, i. e. $y_{cur_j} = y_0 + z_{j+k}$ bzw. $y_{cur_1} = y_0 + z_i$, genutzt werden. Dies ist vertretbar, da die Stufenwerte direkt in die Berechnung des lokalen Fehlers eingehen und für die Newtonprozesse ein verschärftes aber mit der eigentlichen Fehlerabfrage über $\texttt{TOL}_{err}$ gekoppeltes Abbruchkriterium gilt, s. Zeile 4 in Alg. 3.7 sowie (3.177).

Bemerkung 3.67. Ist $\|\cdot\|_{err}$ adaptiv gewählt, so ist für $l > 1$ die Norm von ΔZ^{l-1} in (3.176) in der aktuellen Skalierung neu zu berechnen, um ein konsistentes Maß mit der Norm im Zähler von (3.176) zu erhalten und damit eine sinnvolle Kontraktionsaussage zu ermöglichen. Unter dieser Prämisse ist es valide, die Fehlernorm im Verlauf des Newtonprozesses adaptiv zu wählen, s. die Interpretation hierzu in Bemerkung 3.65.

3.5.1.2 Schrittweitenprädiktion nach regelungstechnischen Gesichtspunkten

Wird mit der aktuellen Schrittweite h die Fehlertoleranz unterschritten, so ermittelt die Funktion `calc_hpred` in Zeile 9 von Alg. 3.7 eine Prädiktion für den nächsten Zeitschritt über

$$\begin{aligned} h_r &= \left[\frac{\varepsilon}{r_{cur}}\right]^{\beta_1} \left[\frac{\varepsilon}{r_{prev}}\right]^{\beta_2} \left[\frac{h}{h_{prev}}\right]^{-\alpha_2} h \\ &= \left[\frac{\varepsilon}{r_{cur}}\right]^{\beta_1+\beta_2} \left[\frac{r_{prev}}{r_{cur}}\right]^{-\beta_2} \left[\frac{h}{h_{prev}}\right]^{-\alpha_2} h \end{aligned} \tag{3.179}$$

für eine gewisse Wahl an Parametern entsprechend der Auswahl in Tab. 3.18. Der Prädiktor mit der Bezeichnung H_0110 ist die klassische und auch im ATHLET-Code bisher verwandte Wahl. Die Konstruktion motiviert sich aus dem Modell

$$r_{cur} = \hat{\varphi}_{cur} h^{\kappa}, \qquad \kappa = p + 1, \tag{3.180}$$

zum lokalen Fehler der Methode. Die Größe r_{cur} ermittelt sich aus Zeile 1 in Alg. 3.7. Unter der Annahme, dass $\hat{\varphi}$ konstant bleibt, i. e. $\hat{\varphi}_{next} = \hat{\varphi}_{cur}$, kann obige Relation nach $\hat{\varphi}_{cur}$ umgestellt werden, um damit $\hat{\varphi}_{next}$ zu ermitteln. Der Schrittweitenregler H_0110 liefert dann im Rahmen dieser Modellannahmen $r_{next} = \varepsilon$. Es wird also eine sofortige Korrektur erzwungen. Dieses Verhalten ist als dead-beat-Regelung bekannt und wird im Kürzel H_0110 durch die tiefgestellte 0 gekennzeichnet. Die Ziffernfolge 110 ist eine spezielle Instanz des Dreitupels $p_D p_A p_F$ und bezieht sich auf gewisse Ordnungseigenschaften des Prädiktors im Sinne eines Reglers, vgl. Tab. 3.18. Nachstehend wird eine kurze Erläuterung zu diesen Eigenschaften gegeben. Detaillierte regelungstechnische Definitionen und Diskussionen lassen sich in /SÖD 03/ finden. Ergänzungen und Kommentare liefert /DIB 13/.

- Dynamik-Ordnung p_D
  Hiermit wird über $2p_D - 1$ die Anzahl an freien Parametern zur Bestimmung von h_r angegeben. Für $p_D = 2$ sind es die drei in (3.179) dargestellten Freiheitsgrade $\beta_1, \beta_2, \alpha_2$. Hierbei ist zu beachten, dass α_1 stets Eins ist, da (3.179) aus einer Gleichung für h_r/h entsteht, und somit erst durch diese Normierung eine Prädiktorwahl durch genau ein Parametertupel beschrieben wird. Da eine explizite Regelung Anwendung findet, also nicht zukünftige Fehler oder Schrittweiten eingebunden werden, kann der Wert von p_D auch dahingehend interpretiert werden, wie lang die Historie ist, auf die zugegriffen wird um h_r zu bestimmen. Für $p_D = 1$ sind es z. B. ausschließlich Werte aus dem aktuellen Schritt, für $p_D = 2$ werden zusätzlich Informationen aus dem vorherigen Schritt einbezogen, s. (3.179).

- Adaptivitätsordnung p_A
  Diese Größe spiegelt wieder, bis zu welchem Polynomgrad Änderungen in $\hat{\varphi}$ durch h_r berücksichtigt werden können. Der Grad ergibt sich zu $p_A - 1$. Entsprechend der obigen Erläuterungen zu den Modellannahmen des Standardreglers H_0110 gilt für diesen $p_A = 1$, es wird also von einem konstanten $\hat{\varphi}$ ausgegangen.

- Filter-Ordnung p_F
  Die Filter-Ordnung gibt Auskunft darüber, inwiefern der Schrittweitenregler in der Lage ist, hochfrequente Anteile in $\hat{\varphi}$ herauszufiltern. Tiefpassfilterung führt zu glatteren Schrittweitenfolgen und kann somit zu einem robusteren Verhalten des Differentialgleichungslösers beitragen. Es gilt $p_F \in \mathbb{N}$ und Filterung wird erst für $p_F > 0$ berücksichtigt. Hinsichtlich der genauen Interpretation des numerischen Wertes von p_F s. /SÖD 03/. Für den dead-beat-Regler H_0110 gilt $p_F = 0$. Es findet also keine Filterung statt. Dieses Verhalten ist plausibel, da nach (3.180) und den Erläuterungen

hierzu bzgl. H_0110 die Beziehung $\log h_r = \kappa^{-1}(\log \varepsilon - \log \hat{\varphi})$ gilt. Jede Störung von $\hat{\varphi}$ zeigt sich also direkt proportional (im logarithmischen Sinne) in h_r. Nach /SÖD 03/ neigen dead-beat-Regler generell dazu, empfindlich auf hohe Frequenzanteile in $\hat{\varphi}$ zu reagieren.

Tab. 3.18 Schrittweitenregler zur Prädiktion mit $\kappa = p + 1$ (lokale Fehlerordnung)

Parameter			Ordnungen			Kürzel	Typ	Verweis
$\kappa\beta_1$	$\kappa\beta_2$	α_2	p_D	p_A	p_F			
1			1	1	0	H_0110	standard	/HAI 93, § II.4/
2	-1	-1	2	2	0	$H_0220(PC11)$	predictive	/GUS 94/,/SÖD 02/
1/6	1/6		2	1	1	$H211PI$	low-pass filter	/SÖD 03/

Bemerkung 3.68. Obschon aus lokaler Sicht die Vorschrift (3.179) ausschließlich zur Prädiktion genutzt wird, so ist der Begriff der Regelung gerechtfertigt, wenn der gesamte Integrationsvorgang als ein zu regelnder Prozess interpretiert wird. In diesem *globalen* Rahmen ist h_r als das Ergebnis einer Regelungs*korrektur* identifizierbar. Eine *lokale* Regelschleife bei Überschreitung der Fehlertoleranz findet durch iteratives Anwenden des Standardreglers H_0110 als Korrektor statt. Dieses Vorgehen ist inspiriert durch den Ablaufplan in /GUS 94/ und steht in Kontrast zur rein lokalen Sichtweise in /DEU 13, § 5/.

Bemerkung 3.69. Entsprechend der betrachteten Normen in Unterabschnitt 3.5.1.1 ist ε in Zeile 1 in Alg. 3.7 und in (3.179) zu $0 < \varepsilon \leqslant 1$ zu wählen. Nach /GUS 94/ stellt $\varepsilon = 0.8$ ein gutes Mittelmaß dar zwischen den Anforderungen

- führe möglichst wenig Schritte bei Einhaltung der Fehlertoleranzen aus (ε dicht an 1);
- halte die Anzahl der verworfenen Prädiktor-Schritte wegen Toleranzüberschreitung gering (ε deutlich im Abstand zu 1).

Für $\texttt{TOL}_{err}$ aus Zeile 4 in Alg. 3.7 ergibt sich damit $\texttt{TOL}_{err} \in [\varepsilon, 1]$. An dieser Stelle wird $\texttt{TOL}_{err} = 1$ gesetzt, der Philosophie in /DEU 13, § 5/ folgend, strenger zu regeln ($\varepsilon < 1$), aber mit der ursprünglichen Forderung $r_{cur} < 1$ zufrieden zu sein.

Wahl der Schrittweitenregler

Der Schrittweitenregler H_0110 ist bereits seit mehreren Jahrzehnten bekannt und in Gebrauch, /HAI 93/ verweist auf eine Quelle von 1961. Regelungstechnische Sichtweisen sind relativ jung, /GUS 94/,/SÖD 02/,/SÖD 03/. Die angesprochenen Defizite des Reglers H_0110 zeigen sich im Rahmen dieses Blickwinkels. Dennoch kann die Anwendung von H_0110 in der Praxis durchaus akzeptable Ergebnisse liefern. Da H_0110 die klassische

Wahl darstellt und wenig überraschend auch der ATHLET-Code hierauf basiert, steht dieser Regler im Testalgorithmus zur Verfügung.

Mit H_0220 steht ein weiterer dead-beat-Regler zur Wahl, der jedoch eine Adaptivitätsordnung von $p_A = 2$ aufweist. Der lokale Verlauf von $\hat{\varphi}$ wird über die lineare Extrapolation $\log \hat{\varphi}_{cur} + (\log \hat{\varphi}_{cur} - \log \hat{\varphi}_{prev})/1$ geschätzt. Da $\hat{\varphi}$ nicht als konstant modelliert wird, ermöglicht dieser Schrittweitenregler eine Prädiktion h_r, für die $h_r < h$ gilt, auch wenn *kein* lokaler Korrekturschritt vorgenommen wurde und auch für $\varepsilon = 1$. H_0110 hingegen erzeugt nur dann für $\varepsilon = 1$ ein h_r mit $h_r < h$, wenn $r_{cur} > 1$ gilt.

Als dritte Option ist der Regler $H211PI$ gegeben, welcher eine Adaptivitätsordnung von $p_A = 1$, aber auch eine Filter-Ordnung von $p_F = 1$ aufweist. Damit ist dieser Regler unempfindlicher gegenüber hochfrequenten Anteilen in $\hat{\varphi}$ als die obigen dead-beat-Regler und sollte somit zu glatteren Schrittweitenfolgen führen. Dies erkauft man sich damit, dass der Abgleich zwischen Toleranz und tatsächlichem Fehler langsamer vonstattengehen kann. Somit sind evtl. mehr Zeitschritte nötig.

Man beachte, dass die beschriebenen Prädiktoren nach Alg. 3.7 in min-max-Limiter eingebettet werden, um Ausreißer in der Vorhersage ob unzureichender lokaler Modellqualität abzuwenden. Ebenso wird eine etwaige Schrittweitenanpassungen mittels h_α berücksichtigt, um Kontraktionsanforderungen des involvierten Newtonprozesses zu genügen, s. hierzu auch die einleitende Diskussion in der Einleitung zum übergeordneten Abschnitt 3.5.1.

Funktionsweise von `calc_hpred`

Die Wahl des Prädiktors findet *vor* der numerischen Integration statt, eine adaptive Vorgehensweise ist bis dato nicht berücksichtigt (und auch schwerlich theoretisch zu untermauern, da zur Laufzeit auswertbare Kriterien zur Verfügung stehen müssten). Die Funktion `calc_hpred` berechnet den gewählten Prädiktor aus Tab. 3.18 genau dann, wenn maximal eine Schrittweitenkorrektur aufgrund einer überschrittenen Fehlertoleranz auftritt. Andernfalls wird stets die Schrittweite zur H_0110-Regelung zurückgegeben. Dieser Ablauf basiert auf /GUS 94/. Erstmaliges Überschreiten der Fehlertoleranz korrigiert das $\hat{\varphi}$-Maß auf aktuelle Werte. Zwei- oder mehrmaliges Überschreiten deutet auf eine schwerwiegendere Verletzung der Modellannahmen hin. H_0110 bietet jedoch auch hierbei im Sinne eines lokalen *Korrektors* eine adäquate Schrittweitenverkleinerung an ($\varepsilon < 1$). Die Regelschleife zur lokalen Prädiktion (global: Korrektur) muss jedoch neu gestartet werden, was eine Rückgabe einer Schrittweite, welche nur auf aktuellen Informationen basiert, rechtfertigt.

Wahl von `hfacmax` in Zeile 13 von Alg. 3.7

Für den Faktor `hfacmax` sind aktuell folgende auf Empirik und /HAI 93, S. 168 f./ basierende Werte gewählt. Die Reihenfolge der Werte spiegelt die Priorisierung wieder, wobei absteigend sortiert ist.

$$\texttt{hfacmax} = \begin{cases} 3 & J \text{ in diesem Schritt konstruiert} \\ 1.2 & \text{Fehler (zunächst) zu groß in diesem oder letztem Schritt} \\ 2 & \texttt{steprestr} \text{ gesetzt} \\ \texttt{fac}_{def} & \text{Standardwert vom Nutzer vorgegeben} \end{cases} \tag{3.181}$$

3.5.1.3 Strategien zum Jacobimatrix-Update

Kontraktionsbezogene Strategie

Im Rahmen einer voll impliziten Methode ist es recht offensichtlich, auf welchen Kriterien eine Update-Entscheidung zur Jacobimatrix fußen sollte: Hinreichend schnelle Konvergenz des Newtonprozesses sollte gewährleistet sein. Grundsätzlich gilt dies auch im $FiterRK$-Kontext, und in Alg. 3.7 sowie Alg. 3.8 wird hierfür die Schätzung α_N genutzt. Diese wiederum beruht auf (3.176), und eine sinnvolle Definition ergibt sich zu

$$\alpha_N := \max\Big[\max_l([\alpha_N]_{l,b}), \max_{l,i}([\alpha_N]_{l,i})\Big], \tag{3.182}$$

wobei $[\alpha_N]_{l,b}$ die Schätzung aus (3.176) für die in einem nichttrivialen Block gemeinsam iterierten Stufen bezeichnet und $[\alpha_N]_{l,i}$ sich auf Diagonalstufe i mit $i = \bar{m}+1, \ldots, s$ bezieht. Steht für den Block oder eine Diagonalstufe keine α_N-Schätzung zur Verfügung, weil nur ein Newtonschritt ausgeführt wurde, so ist der Block bzw. diese Diagonalstufe für die Auswertung von (3.182) auszuschließen. Es bestehen gute Chancen, dass bei einer Stufenordnung von $p > 1$ mindestens die Blockstufen mehr als einen Iterationsschritt erfahren, vgl. Korollar 3.37, und somit Informationen zur Bestimmung von α_N liefern können.

Liegt eine klassische W-Methode vor oder wird durch den Einsatz eines T2-Updates, s. Unterabschnitt 3.3.5.1, für einen nichttrivialen Block ebenfalls nur ein Newtonschritt ausgeführt, so ist α_N nicht bestimmbar. In der klassischen Literatur zu W-Methoden werden in diesem Fall verschiedene Ansätze verfolgt: In /SHA 97/ wird unabhängig von der Dynamik in jedem Zeitschritt eine neue Matrix konstruiert – es sei denn, der Nutzer hat das Problem von vornherein als linear gekennzeichnet. In /ZED 90/ wird nach

jeder Fehlerüberschreitung, i. e. $r_{cur} > 1$ in Alg. 3.7, eine neue Jacobimatriz angefordert. Deuflhard nutzt die Interpretation des bewusst abgebrochenen Newtonprozesses in /DEU 85/ und forciert einen zweiten Newtonschritt, so dass zumindest eine $[\alpha_N]$-Entität berechnet und somit eine Schätzung α_N bestimmt werden kann. Gekoppelt ist dies mit einer stringenten Anforderung an α_N, da $\alpha_N < 10^{-3}$ gefordert wird. Im Falle der linear impliziten Mittelpunktsregel ist die Berechnung hiervon ohne nennenswerte zusätzliche Kosten möglich, s. /HAI 96, § IV.9/, für den linear impliziten Euler hingegen steht die Lösung eines weiteren Gleichungssystems an.

Innerhalb dieser Arbeit wurde sich für das Deuflhardsche Prinzip entschieden: Wenn keiner der Newtonprozesse zur Berechnung der Stufenwerte einen zweiten Iterationsschritt ausführt, so wird für eine Stufe (oder den etwaigen Block) ein zweiter Iterationsschritt initiiert und dann die aus /GUS 97/ stammende Bedingung in Zeile 15 von Alg. 3.7 genutzt, um den Einfluss der Jacobimatrixapproximation abzuschätzen. Mit der gegebenen Infrastruktur, eine feste Anzahl von Newtonschritte ausführen zu können, ist ein solcher Zusatz leicht realisierbar. Es sollte aber frei stehen, ob die zusätzliche Newtonkorrektur auch für das Ergebnis Anwendung findet, s. hierzu die Diskussion in /HAI 96, § IV.9/ aber auch /HAI 93, Th. II.9.1/.

Spektrumsbezogene Strategie

Im Gegensatz zu einer voll impliziten Methode sind die in den Newtonprozessen (3.92) und (3.117b) berechneten Z-Entitäten abhängig von hJ. Diese Abhängigkeit ist merklich, wenn $\|hJ\|_{err}$ nicht klein ist. Ist dies der Fall oder ist $\|hf_0^{(1)}\|_{err}$ nicht klein, was einem nichttrivialen Einfluss von J für die Idealwahl $J = f_0^{(1)}$ entspricht, dann sieht die Strategie vor, gewisse Spektrumsinformationen von J und $f_0^{(1)}$ miteinander zu vergleichen, s. (3.183c). In Alg. 3.7 wird dieses Vorgehen als *Spektralschätzung* bezeichnet.

Die beiden Matrixnormen stehen nicht ohne Weiteres zur Verfügung und müssen von daher für ein effizientes Vorgehen geschätzt werden. Mit Hilfe eines noch zu bestimmenden $\xi \in \mathbb{R}^n$ sei die oben beschriebene Abfrage approximiert über

$$\text{Falls} \quad \|hJ\xi\|_{err} > \varepsilon_{norm} \quad \text{oder} \quad \|hf_0^{(1)}\xi\|_{err} > \varepsilon_{norm} \quad \text{untersuche Spektrum.} \tag{3.183a}$$

Das Produkt $f_0^{(1)}\xi$ steht z. B. approximativ über finite Differenzen zur Verfügung. Um „nicht klein“ im Rahmen eines Newtonprozesses zu quantifizieren, ist $\varepsilon_{norm} < \alpha_{ref}$ eine sinnvolle Bedingung. Konkret ist hier

$$\varepsilon_{norm} = 2^{-3} \cdot \alpha_{ref} \tag{3.183b}$$

gewählt. Die bereits durchgeführten Matrix-Vektor-Multiplikationen nutzend, ergibt sich der Kern des Spektrumtests zu

$$\frac{\|(f_0^{(1)} - J)\xi\|_2}{\|f_0^{(1)}\xi\|_2} > \varepsilon_{dist} \tag{3.183c}$$

mit der in dieser Arbeit getroffenen Wahl

$$\varepsilon_{dist} = 2^{-5}. \tag{3.183d}$$

Obiges Verhältnis ist genau dann klein, wenn $f_0^{(1)}\xi$ und $J\xi$ in Richtung *und* Länge (relativ zu $\|f_0^{(1)}\xi\|_2$) ungefähr gleich sind.

Es bleibt die Wahl von ξ zu diskutieren. Sei angenommen, dass $f_0^{(1)}$ und J je einen dominanten Eigenwert besitzen. Da die dominanten Eigenwerte das Stabilitätsverhalten der $FiterRK$-Methode bestimmen, wäre eine probate Wahl $\xi = (x_1 + x_2)/\|x_1 + x_2\|_{err}$, wobei x_1 normierter Eigenvektor zum dominanten Eigenwert von $f_0^{(1)}$ wäre und x_2 ein solcher Vektor zu J. Als Approximation kann

$$\xi = err_{est}^{loc}/\|err_{est}^{loc}\|_{err} \tag{3.183e}$$

dienen, da $err_{est}^{loc} \in \mathcal{O}(h^{p+1})$ für eine eingebettet Methode zum Ordnungspaar $p+1(p)$ gilt und sich err_{est}^{loc} allgemein über (3.136) bestimmt. Taylorentwicklung liefert dann eine Abhängigkeit von $\left[f_0^{(1)}\right]^p f_0$ und $J^p f_0$, s. in diesem Zusammenhang auch /HAI 96, Eq. (7.31;3)/. Diese Terme sind als p-fache Anwendung der Potenzmethode interpretierbar und die Operationen in (3.183a) und (3.183c) führen jeweils für eine der beiden Matrizen die Potenzmethode einen Schritt weiter aus. Es bestehen also gute Chancen, dass Informationen zu Eigenräumen von dominanten Eigenwerten in die Abfragen (3.183a) und (3.183c) eingehen.

T2-Update bezogene Strategie

Wird die Option eines T2-Updates wahrgenommen, so ist nach der Diskussion in Unterabschnitt 3.3.5.3 die Schranke (3.162) für $\|\widetilde{u}\|_2$ einzuhalten. Eine Verletzung dieser Schranke deutet auf eine unzureichende Qualität der Iterationsmatrix M hin. Aus $h \to 0$ folgt $\widetilde{u} \to 0$, und die Jacobimatrix verliert an Einfluss. Die Schrittweite kleiner zu wählen, hat eine Entsteifung des Systems zur Folge. Nicht weil eine solche Reduktion zur Einhaltung einer lokalen Fehlerschranke nötig ist, sondern ausschließlich deshalb, weil $J \neq f_0^{(1)}$ gilt. Um nicht den Vorteil linear impliziter Methoden zu verlieren, große Schritte bei potentiell steifen Differentialgleichungen gehen zu können, wird sich im Rahmen des Testalgorithmus für eine Verbesserung der aktuellen Jacobimatrixapproximation

J entschieden, statt der Verletzung der Schranke (3.162) mit einer Verkleinerung der Schrittweite h entgegenzuwirken.

Man beachte, dass in der Abfrage (3.162) statt der Euklidischen Norm auch die Fehlernorm $\|\cdot\|_{err}$ genutzt werden kann, wenn $\psi = 2$ in (3.178a) gilt. Alle Aussagen zum T2-Update in Abschnitt 3.3.5 bleiben dann erhalten, wenn von einem skalierten System (3.173) mit $D_{sys} = D_{err}$ ausgegangen wird. Ist $\psi = \infty$, wird also die Maximumnorm zur Fehlermessung genutzt, dann kann wegen der Normäquivalenz in $\mathbb{R}^n$ statt $\|D_{err}^{-1}\widetilde{u}\|_2$ auch $\sqrt{n} \cdot \|D_{err}^{-1}\widetilde{u}\|_\infty$ in (3.162) genutzt werden.

3.5.2 Numerischer Benchmark und Vergleich

Der in Abschnitt 3.5 vorgestellte Testalgorithmus wird genutzt, um die Qualität der entwickelten Verfahren aus Abschnitt 3.4 und der Schrittweiten- und Jacobimatrixkontrolle aus Abschnitt 3.5.1 im Kontext bekannter steifer Probleme auszutesten. Zum Vergleich wird auch die Extrapolation basierend auf dem linear impliziten Euler in diesem Kontext betrachtet. Ein solches Vorgehen ist ein wesentlicher Schritt zur Beurteilung der neuen Konzepte, da zum einen die gleiche Algorithmik genutzt wird, und zum anderen Probleme betrachtet werden, die sich als herausfordernd herausstellen und deren Lösung in der Literatur einsehbar ist und somit zum Abgleich zur Verfügung steht.

Auswahl von Testproblemen

Aus /HAI 96, § IV.10/ wurden die Probleme `VDPOL`, `ROBER`, `E5`, `PLATE` und `BEAM` ausgewählt. Motivation hierfür sind die teils sehr unterschiedlichen Charakteristika der Probleme, so dass ein breites Spektrum an Herausforderungen für die Methoden und den Testalgorithmus gegeben sind. Zur Erläuterung der genutzten Steifheitsbegriffe s. Bemerkung 3.1.

- `VDPOL`, $n = 2$, autonom, klassische Steifheit
  Dieses Problem beschreibt einen bestimmten elektrischen Stromkreis. Die Lösung weist lange Phasen glatten aber steifen Verhaltens auf, welche abrupt in schmale Phasen nichtsteifen Verhaltens wechseln. Die Lösung ist periodisch. Die Steifheit des Systems und damit die Glattheit der Übergänge steif/nicht steif wird über einen freien Parameter $\varepsilon_{\texttt{VDPOL}} > 0$ gesteuert. Je weiter sich $\varepsilon_{\texttt{VDPOL}}$ von Null entfernt, desto weniger steif zeigt sich das System. An dieser Stelle wird $\varepsilon_{\texttt{VDPOL}}$ zu $\varepsilon_{\texttt{VDPOL}} = 10^{-3}$ gesetzt ist. Damit ist das Problem merklich steif.

- `ROBER`, $n = 3$, autonom, klassische Steifheit
  Mit Hilfe der Gleichungen wird eine bestimmte Reaktionskinetik beschrieben. Die zweite Lösungskomponente $y_{(2)}$ darf nicht negativ werden, da sonst das System instabil

werden würde. Dies ist numerisch relevant, da $y_{(2)} \to 0$ für $t \to \infty$. Dementsprechend wird das Problem für ein sehr großes Zeitintervall betrachtet, s. Tab. 3.20.

- `E5`, $n = 4$, autonom, klassische Steifheit
  Auch hier liegt den Gleichungen eine bestimmte Reaktionskinetik zu Grunde. Das Problem ist schlecht skaliert, da keine der Lösungskomponenten größer als $2 \cdot 10^{-3}$ wird. Zusätzlich liegt eine starke Diskrepanz in den relativen Größenordnungen vor: Manche Lösungskomponenten unterscheiden sich um einen Faktor von bis zu 10^{10}. Dies legt es nahe, Systemskalierung mit einem sehr klein angesetzten TOL_{scl} in (3.171) zu nutzen. Ebenso ist die absolute Fehlertoleranz in (3.178) extrem klein zu wählen, s. a. Tab. 3.22. Die Reaktion verläuft sehr langsam, was wiederum zur Betrachtung eines sehr großen Zeitintervalles führt.

- `PLATE`, $n = 80$, nicht-autonom, klassische/oszillatorische Steifheit
  Das Problem basiert auf einer partiellen Differentialgleichung, welche herangezogen wird, um die Bewegung eines rollenden Fahrzeugs auf einer elastischen Platte zu simulieren. Hierbei wird auf den Plattenoperator $\Delta\Delta$ (doppelter Laplace-Operator) in $\mathbb{R}^2$ zurückgegriffen. Das System gewöhnlicher Differentialgleichungen entsteht aus einer Semidiskretisierung im Raum. Generell richtet sich die Dimensionierung des Problems nach der räumlichen Auflösung. Der hier gewählte Wert entspricht dem in /HAI 96, § IV.10/, wird aber auch für doppelte Größe ausgetestet, s. Tab. 3.24, um die Unabhängigkeit des Löserverhaltens von der räumlichen Auflösung aufzuzeigen. Das Problem ist linear in den Lösungsvariablen. Es sollte also nur eine Jacobimatrix konstruiert werden. Das Problem ist moderat steif mit Eigenwerten der Jacobimatrix auf der negativen reellen Achse sowie im α-Kegel aus Abb. 3.2 für $\alpha \approx 71\,^\circ$.

- `BEAM`, $n = 40$, nicht-autonom, oszillatorische Steifheit
  Für dieses Problem wird die Auswirkung einer Zugkraft, welche an der Spitze eines elastischen Stabes ansetzt, simuliert. Hierbei wird sich des Lagrange-Formalismus bedient. Das endlichdimensionale System gewöhnlicher Differentialgleichungen entsteht aus einer Diskretisierung des Stabes entlang seiner Länge. Somit ist auch dieses Problem per se beliebig dimensionierbar. Der Wert $n = 40$ entspricht der Vorgabe aus der Literatur. Die Eigenwerte der Jacobimatrix liegen dicht oder auf der imaginären Achse, was einen besonderen Fokus auf A-Stabilität der Verfahren legt. Das Standardproblem geht von einem Stab in Ruhelage zu Beginn der Simulation aus. Das Problem wird auch für leicht gestörte Eingangsdaten betrachtet, um den Effekt der Dämpfung des Schrittweitenreglers $H211PI$ aus Unterabschnitt 3.5.1.2 aufzuzeigen. Hierfür wird an der Spitze des Stabes ein kleiner Knick simuliert. Da das System

konservativ ist, entstehen hochfrequente Schwingungen mit geringer Amplitude in der Lösung. Der Vergleich der Schrittweitenregler ist in Abb. 3.4 zu finden.

Festlegung gewisser Kontrollparameter

Kriterien sind nur so gut, wie ihre freien Parameter sinnvoll gesetzt sind. Dies gilt insbesondere bei Kombination von verschiedenen Parametern. Unten stehende Wahl orientiert sich an der Literatur und/oder ergibt sich aus der gewonnenen Erfahrung zu einer Vielzahl an Testläufen. Ein Neusetzen der Parameter ist jederzeit möglich. In Hinblick auf eine ATHLET-bezogene Implementierung können die hier genutzten Festlegungen als Startiterierte genutzt werden.

- allgemeine Schrittweiten- und Jacobimatrixkontrolle, Abschnitt 3.5.1
  $\alpha_{ref} = 2 \cdot 10^{-1}$, $\alpha_{Jac} = 3 \cdot 10^{-2}$ zur Kontrolle der Kontraktion sowie $c_1 = c_2 = 1$ für die Toleranz zum Update einer Zerlegung
- Newton- und Fehlermodell, Unterabschnitt 3.5.1.1
  $\nu_{err} = 2^{-3} \cdot 10^{-1}$, $\nu_{res} = 2^{-1} \cdot 10^{-3}$, ν_{lin} wie in (3.177c) zur Abbruchbedingung (3.177) sowie $\psi = 2$ und max-Wahl zur Festlegung der Fehlernorm in (3.178)
- Schrittweitenprädiktion, Unterabschnitt 3.5.1.2
  Sicherheitsfaktor $\varepsilon = 0.8$ aus Bemerkung 3.69 zum Standardregler
- Jacobimatrix-Update, Unterabschnitt 3.5.1.3
  $\varepsilon_{norm} = 2^{-3} \cdot \alpha_{ref}$, $\varepsilon_{dist} = 2^{-5}$ zum Spektraltest (3.183) sowie $\delta = 2^{-3}$ aus (3.162) mit $\|\cdot\|_{err}$ statt $\|\cdot\|_2$ fürs T2-Update

3.5.2.1 Testläufe

Im Rahmen der Arbeiten zu diesem Projekt wurden für die obigen Probleme und die konstruierten Methoden sowie die Extrapolation im Zusammenspiel mit den verschiedenen Schrittweitenreglern und unter Berücksichtigung von Variationen obiger Kontrollparameter eine Vielzahl von Tests durchgeführt. Nachstehend werden tabellarisch für einige ausgewählte Fälle Ergebnisse präsentiert, um einen Eindruck der Qualität der Neuerungen zu erhalten. In der anschließenden Diskussion der Ergebnisse wird auch auf die Erfahrungen aus der Gesamtheit aller Tests eingegangen.

Die relative Fehlertoleranz ist für alle hier gegebenen Ergebnisse zu

$$rtol = 10^{-4}$$

in (3.178) gesetzt. Dies ist adäquat für die hier betrachteten Methoden relativ geringer Ordnung und geht ebenfalls konform mit den üblichen Toleranzwerten im ATHLET-Kontext.

Die absolute Fehlertoleranz ist entsprechend der in /HAI 96, § IV.10/ genutzten Werte gewählt. Als Standardschrittweitenregler ist $H211PI$ gesetzt. Ebenso ist standardmäßig Systemskalierung nach (3.171) für $TOL_{scl} = 10^{-10}$ eingeschaltet. Der Spektraltest (3.183c) wird jedoch stets ohne Systemskalierung betrachtet. Für Methoden, die nur einen Newtonschritt pro Stufe ausführen (z. B. $T_{3,3}$), wird insgesamt ein weiterer Schritt ausgeführt, um eine Schätzung für (3.182) zur Verfügung zu haben.

Die Bezeichnungen in den Tabellen sind wie folgt zu verstehen:[3]

- #steps: Anzahl der Zeitschritte, um von t_{start} zu t_{end} zu integrieren
- hfail: Anzahl der berechneten aber verworfenen Schrittweiten; in Klammern steht die relative Anzahl in Prozent auf zwei führende Stellen gerundet
- #Jcalc: Anzahl der Neuberechnungen der Jacobimatrix; entweder wird mit einer alten Jacobimatrix gearbeitet oder eine komplett neue berechnet, Teil-Updates existieren in diesem Kontext nicht
- #LS: Anzahl der gelösten linearen Gleichungssysteme
- Zeit[s]: Zeit in Sekunden für einen Löserdurchlauf, nur die ersten drei führenden Stellen werden berücksichtigt

Für die $FiterRK$-Methoden wird der Bezeichner ohne das führende Kürzel angegeben. $FiterRK32a$ wird somit z. B. zu $32a$. Der modifizierte Extrapolationsansatz, der auf (3.45) basiert, verhält sich für autonome Systeme exakt gleich zur Standardversion, da sich die Modifikation lediglich auf einen zeitexpliziten Anteil in f bezieht. Dieser ist bei autonomen Systemen per Definition nicht vorhanden. Es wird somit für autonome Systeme $modT_{3,3}$ nicht gesondert aufgeführt.

Tab. 3.19 `VDPOL` auf $[0, 2]$, $atol = rtol$, $h_0 = 10^{-3}$

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
32a	354	104(23)	137	3360	3.46
32b	252	85(25)	114	3204	3.21
43	144	60(29)	87	2944	3.03
32ex	440	127(22)	175	2292	2.72
$T_{3,3}$	1470	159(9.8)	287	11403	11.02

[3] Es werden wegen der kompakteren Schreibweise englischsprachige Bezeichner genutzt.

Tab. 3.20 ROBER auf $[0, 10^{11}]$, $atol = 10^{-6} \cdot rtol$, $h_0 = 10^{-3}$

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
32a	326	4(0.12)	148	2287	2.35
32b	242	3(0.12)	110	2163	2.2
43	154	1(0.06)	78	1931	1.99
32ex	Integration extrem langsam $\Rightarrow$ Abbruch				
$T_{3,3}$	1318	45(3.4)	123	9564	9.42

Tab. 3.21 ROBER auf $[0, 10^{11}]$, $atol = 10^{-6} \cdot rtol$, $h_0 = 10^{-3}$, verschiedene Schrittweitenregler

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
43(H_0110)	110	3(2.6)	61	1406	1.54
43(H_0220)	94	1(1)	73	1215	1.25
43($H211PI$)	154	1(0.06)	78	1931	1.99

H_0110 ist der Standardregler, s. a. Tab. 3.18

Tab. 3.22 E5 auf $[0, 10^{11}]$, $atol = 1.7 \cdot 10^{-24}$, $h_0 = 10^{-4}$, $TOL_{scl} = 10^{-15}$

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
32a	1200	13(1.1)	173	6919	7.06
32b	1048	12(1.1)	161	7917	7.55
43	255	11(4.1)	144	2881	2.95
32ex	6447	1739(21)	552	32832	39.89
$T_{3,3}$	2617	(3.3)	682	19026	18.65

Tab. 3.23 PLATE auf $[0, 7]$, $atol = 10^{-3} \cdot rtol$, $h_0 = 10^{-2}$

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
32a	236	33(12)	1	1799	2.70
32b	155	29(16)	1	1560	2.15
43	50	11(18)	1	565	0.75
32ex	283	33(10)	1	1264	2.34
$T_{3,3}$	1779	11(0.6)	†1	12530	26.08
$modT_{3,3}$	1445	28(1.9)	1	10311	20.70

†: Forcierung, nur eine Jacobimatrix zu bauen
(PLATE ist linear in y), s. a. unten stehende Diskussion

Tab. 3.24 PLATE auf $[0,7]$, $atol = 10^{-3} \cdot rtol$, $h_0 = 10^{-2}$, Dimensionsvergleich

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
32a $(n = 80)$	236	33(12)	1	1799	2.70
32a $(n = 160)$	229	36(13)	1	1767	3.26

Tab. 3.25 BEAM auf $[0,5]$, $atol = rtol$, $h_0 = 10^{-2}$

Methode	#steps	hfail	#Jcalc	#LS	Zeit [s]
32a	111	24(17)	54	1057	2.71
32b	78	14(15)	41	900	2.01
43	41	9(18)	27	667	1.33
32ex	138	20(13)	34	648	1.99
$T_{3,3}$	466	32(6.4)	131	3508	11.43
$modT_{3,3}$	420	30(6.6)	76	3192	9.33

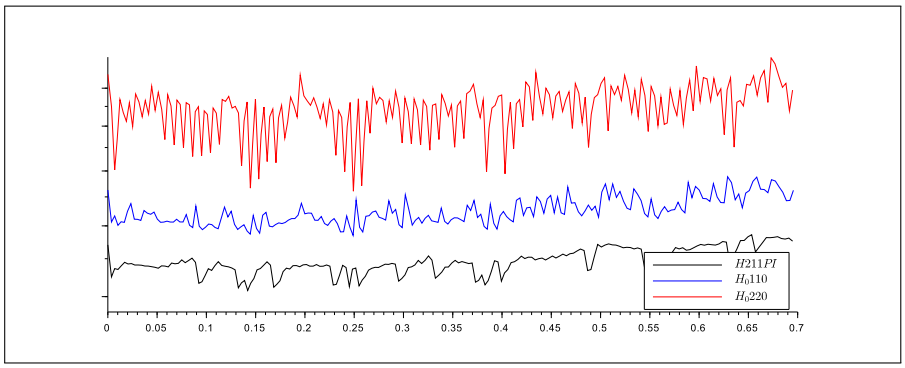


Abb. 3.4 Vergleich von Schrittweitenreglern für Problem BEAM und Methode $FiterRK32b$ auf [0,0.7] und mit Auslenkung von $3 \cdot 10^{-3}$ rad an der Spitze des Stabes; $n = 10$, $atol = rtol = 10^{-3}$. Zur besseren Übersicht sind die Schrittweitenfolgen übereinander gezeichnet. Alle Folgen sind von der Größenordnung $\mathcal{O}(10^{-3})$.

3.5.2.2 Diskussion der Ergebnisse

Bis auf eine Ausnahme haben alle Methoden für sämtliche hier betrachteten Probleme eine passende approximative Lösung errechnen können. Grundsätzlich funktionieren also die Methoden aus Abschnitt 3.4 sowie das Zusammenspiel mit der Schrittweiten- und Jacobimatrixsteuerung aus Abschnitt 3.5.1. Wie die konkreten Ergebnisse in den Tabellen 3.19-3.25 zeigen, liefern die neuen Methoden teils deutlich bessere Ergebnisse

als der Extrapolationsansatz. Anhand der nicht-autonomen Beispiele PLATE und BEAM erkennt man, dass sich im generischen Fall auch die Modifikation $modT_{3,3}$ basierend auf (3.45) bezahlt macht. Es besteht somit begründete Hoffnung, dass im Kontext steifer Differentialgleichungen zum ATHLET die neuen Konzepte gute Dienste leisten werden. Eine Anpassung an die dortigen Gegebenheiten findet sich in den Methoden, welche in Abschnitt 3.6.2 vorgestellt werden, wieder.

Im Folgenden wird auf einige Details zu den Methoden, Problemen und einigen Aspekten der Steuerung eingegangen.

$FiterRK43$

Generell wird $FiterRK43$ seiner erwarteten Rolle gerecht, mit den wenigstens Schritten eine Lösungsapproximation zu generieren, da es die einzige Methode zum Ordnungspaar $4(3)$ ist. Hierbei sind die Verhältnisse sogar derart, dass trotz Mehrarbeit pro Einzelschritt weniger Arbeit insgesamt verrichtet wird (ausgenommen VDPOL, s. weiter unten).

$FiterRK32a$/$FiterRK32b$

Beide Methoden weisen gute Stabilitätseigenschaften auf. Die Variante $32b$ muss mehr Arbeit pro Schritt leisten, wartet aber auch mit höherer Genauigkeit auf. Die Ergebnisse zeigen, dass sich dies meist auf lange Sicht auszahlt und somit $32b$ ein wenig performanter agiert als $32a$.

$FiterRK32ex$

Die Ausnahme zur erfolgreichen Integration aller Probleme wird durch $FiterRK32ex$ angewandt auf das Problem ROBER generiert. Auch nach mehreren Minuten war kein Erreichen von t_{end} in Sicht. Für weniger stringente Werte von ν_{lin} in (3.177c) wird das System sogar instabil, da $y_{(2)} < 0$ in der numerischen Approximation auftritt. Ein ähnliches Bild zeigt sich für das Problem E5. Im Vergleich zu den anderen Methoden muss übermäßig viel Arbeit geleistet werden. Im Intervall $[10^2, 10^6]$ findet ein wesentlicher Anteil der chemische Reaktion statt, die sehr glatt abläuft. Die Schrittweitenfolge oszilliert jedoch stark. Ein bewusstes Neuberechnen der Jacobimatrix in jedem Schritt erbrachte zwar eine merkliche Verbesserung, jedoch liegt die Schrittzahl von 2145 noch deutlich über denjenigen der anderen neuen Methoden. Diese Verhaltensmuster sind vermutlich den etwas schwächeren Stabilitätseigenschaften geschuldet. Zusätzlich findet keine Form von Newtoniteration statt, somit hat das Aussehen von J einen starken Einfluss auf das Verhalten der Methode. Dass bei exakter Jacobi-Information konkurrenzfähige Leistung auftreten kann, zeigt das Problem PLATE, s. Tab. 3.23.

$T_{3,3}/mod T_{3,3}$

Im gegebenen Testumfeld erweist sich der Extrapolationsansatz als weniger effizient. Dies mag der Gesamtheit aus geringer Stufenordnung, größerer Fehlerkonstante und schwächelnder A-Stabilität geschuldet sein. Besonders negativ fällt $T_{3,3}$ für das Problem `PLATE` auf. Hier zeigt sich die Schwäche, dass $\mathcal{C}_\alpha(1)$ aus (3.73) nicht erfüllt ist. Leider kann dies bei Standardeinstellungen des Lösers dazu führen, dass auf Basis von (3.182) eine neue Jacobimatrix angefordert wird, da ein zusätzlicher Newtonschritt durchgeführt wird, Konvergenz aber nicht bereits nach dem ersten Schritt auftritt und somit α_N einen nichttrivialen Wert annehmen kann. Es wurde somit forciert, keine neue Jacobimatrix zu berechnen.

Problem `VDPOL`

Bei diesem Problem treten lange steife Phasen auf, welche sich mit sehr kurzen nichtsteifen Phasen abwechseln. Alle Methoden durchlaufen die steifen Phasen sehr zügig, müssen jedoch sehr viele Schritte im nichtsteifen Teil verbringen, da hier starke Veränderungen im System ablaufen. Es ist im Grunde nicht angebracht, für diese Phasen einen (linear-)impliziten Löser einzusetzen. Auch die Steuerung zum Update der Jacobimatrix ist nicht auf nichtsteife Phasen mit viel Systemaktivität ausgelegt. Folglich stehen relativ viele Updates der Jacobimatrix an, wie in Tab. 3.19 zu sehen ist. Es bräuchte eine Beobachtung des Systems, wann Wechsel zwischen steifem und nichtsteifem verhalten anstehen. Solche Beobachtungsstrategien werden in /SHA 81/ oder auch /GUS 97/ diskutiert. Wenn Bedarf besteht, kann hieraus eine Adaption für die in diesem Abschnitt vorgestellte Steuerung erfolgen.

Schrittweitenregler

Die Schrittweitenregler aus Tab. 3.18 unterscheiden sich nicht frappierend in ihrem Verhalten, weswegen sich für die dargestellten Ergebnisse auf die Nutzung eines Reglers beschränkt wurde. Generell lässt sich aber sagen, dass H_0110 Standardregler und H_0220 (predicitv control) in der Regel leicht größere Schrittweiten ermöglichen. $H211PI$ ist etwas konservativer veranlagt, liefert aber aufgrund seiner Dämpfungseigenschaften teils deutlich glattere Schrittweitenfolgen, s. Abb. 3.4.

Fehlernorm

Auch die Art der Fehlernorm hat keinen allzu gravierenden Einfluss auf die Ergebnisse, weswegen hier nur eine Ausprägung betrachtet wurde. Tab. 3.24 zeigt sehr schön, wie sehr sich zwei Durchläufe des gleichen Problems nur mit verschiedener Dimension ähneln. Es wird feiner diskretisiert im Raum, was zu einer Erhöhung der Laufzeit wegen

größerer linearer System führt, jedoch nicht ein gänzlich anderes Löserverhalten zur Folge hat. Im Falle der Euklidischen Norm ist dies auf die Mittelung durch den Parameter ζ aus (3.178b) zurückzuführen. Die Maximumnorm ist per se nahezu unabhängig von solchen Dimensionsänderungen.

T2-Update

Für die Testläufe zu den Ergebnissen in den Tabellen 3.19-3.25 wurde auf eine Anwendung des T2-Updates verzichtet. Leider zeigt es ein recht unbeständiges Verhalten: Ab und zu können Jacobimatrixauswertungen gespart werden, manchmal erhöht sich die Anzahl. Oft werden mehr Schritte benötigt. Wenn die Evaluierung einer Jacobimatrix deutlich die Kosten weiterer Schritte überschreitet, kann das T2-Update evtl. als Option in Erwägung gezogen werden.

3.6 Das $FiterRK$-Konzept im ATHLET-Kontext

Im Laufe des Projektes wurden immer mehr Besonderheiten im ATHLET-Code erkannt, welche auf die Extrapolation mit dem linear-impliziten Euler zugeschnitten sind und die Anwendung einer generischen Methode erschweren:

NFKEY-Steuerung der f-Auswertungen und Kosten

Die Integration in FEBE ist auf Subschrittebene lose mit dem Wärmeleitungsmodul HECU gekoppelt. Der Einfluss der Kopplung zeigt sich in den Rückgabewerten der f-Evaluationen. Gesteuert wird das Verhalten der Kopplung über das Flag NFKEY. FEBE hat hierbei die Priorität, und die HECU-Daten werden auf $\hat{t} \in (t_0 + h]$ nachgezogen, sobald eine f-Auswertung für ein solches $\hat{t}$ ansteht. Für den Rückgabewert ist entscheidend, von welchem Zeitpunkt aus HECU-Daten nachgezogen werden. Mittels $\texttt{NFKEY} = 3$ wird sich der Daten am Anfang des lokalen Zeitschrittes bedient. Ein f-Aufruf mit $\texttt{NFKEY} = 4|7$ erwartet eine Historie von mindestens einer vorherigen f-Auswertung im offenen Intervall $(t_0, t_0 + h)$ und setzt die HECU-Kopplung vom letzten hierin involvierten Zeitpunkt fort. Diese Art von Kopplung impliziert, dass zwei f-Auswertungen direkt hintereinander mit den *gleichen* Eingabedaten seitens FEBE *nicht zwingend* zu dem gleichen Ergebnis führen müssen! Dies liegt daran, dass durch das Hinterherziehen der HECU-Daten beim ersten Aufruf, der zweite Aufruf von f eine andere Ausgangssituation vorfindet. Für einen Aufruf mit $\texttt{NFKEY} = 3$ gilt dies nicht. Tests haben gezeigt, dass eine durchgehende Wahl von $\texttt{NFKEY} = 3$ jedoch nicht zu befriedigenden Werten führt, wenn eine f-Auswertung eher in der Nähe von $t_0 + h$ statt t_0 berechnet wird. Die Kopplung auf Subschrittebene hat

also Relevanz. Auf der anderen Seite wird hierdurch natürlich jedweder Newton-Prozess erschwert. Jede Funktionsauswertung weiter rechts im lokalen Intervall bedarf im Grunde einer *Folge* von sukzessiven Auswertungen, um der Kopplung gerecht zu werden. Hierfür müssen neben den Zeitpunkten auch y-Daten zur Verfügung stehen. Wie sich aus den Tabellen in Abschnitt 3.4.1 erkennen lässt, bietet nur $FiterRK43$ eine Verteilung an Stufenwerte, die nicht allzu große Zeit-Lücken zwischen den einzelnen Stufenwerten lässt.

Die Art der Kopplung ist optimal an den Extrapolationsansatz angepasst, da dieser zur Berechnung der $T_{i,1}$ das Euler-Verfahren auf den Subschritten nutzt. Somit ergibt sich auf natürliche Weise pro Extrapolationsstufe ein sukzessives Voranschreiten im lokalen Integrationsintervall. Im Sinne der Rechen-Effizienz bietet das Extrapolationsverfahren den weiteren Vorteil, keine f-Auswertung an $t_0 + h$ zu benötigen, da jeder Eulerschritt lediglich linear-implizit gerechnet wird. Um sicher zu stellen, dass für den nächsten Zeitschritt keine Komplikationen in der f-Auswertung stattfindet, wird dennoch f an der finalen Approximation y_1 mit$\texttt{NFKEY} = 7$ ausgewertet. Aufgrund der Berechnung von $T_{3,1}$ befinden sich die `HECU`-Daten auf dem Stand von $t_0 + 2h/3$. Somit ist es wieder nur ein kleiner Subschritt, um die gekoppelten Daten nachzuziehen. Dieser finale Prüfschritt muss auch von jeder anderen Methode bereitgestellt werden können. Somit muss eine Situation geschaffen werden, dass die `HECU`-Daten nahe am vollen Zeitschritt $t_0 + h$ vorliegen, wenn y_1 berechnet wird. Man beachte, dass noch eine weitere Auswertung an y_1 durchgeführt wird, um sämtliche auf h-Schrittebene gekoppelten Werte nachzuziehen. Dies wird über $\texttt{NFKEY} = 100$ initiiert, und es wird davon ausgegangen, dass direkt davor der diskutierte Aufruf mit $\texttt{NFKEY} = 7$ stattgefunden hat.

FEBE führt inklusive der Aufrufe mit $\texttt{NFKEY} = 7|100$ in der Summe sieben f-Auswertungen pro Zeitschritt h aus. Diese Zählung berücksichtigt nicht die benötigten f-Evaluationen für ein etwaiges Update der Jacobimatrix und berechnet eine f-Auswertung für $\partial f_0/\partial t$ in (3.4). Eine solch niedrige Anzahl an f-Aufrufen kann generell nicht von $FiterRK$-Methoden erwartet werden, welche eine hohe Stufenordnung aufweisen und aus Gründen der Stabilität 0-explizit sind, s. hierzu auch Abschnitt 3.6.1.

Wie bereits in Unterabschnitt 2.3.1.5 beschrieben kapselt f die Modellierungsmächtigkeit von ATHLET im Kontext des DGL-Lösers. In der Summe geht der Aufwand zur Berechnung der f-Auswertungen nicht trivial in die Gesamtlaufzeit ein, s. auch Abschnitt 6.2. Es kann zwar über die in Bemerkung 2.12 beschriebenen Techniken eine Beschleunigung erzielt werden. Dennoch ist es freilich wünschenswert, die Anzahl an benötigten f-Evaluationen klein zu halten.

Bemerkung 3.70. Die NFKEY-Steuerung zur f-Auswertung verhindert im Grunde den sinnvollen Einsatz matrixfreier Verfahren zum Lösen der im Integrationsprozess involvierten Gleichungssysteme. Denn diesen müssten Produkte der Form $\partial f_0/\partial y \cdot d$ per finite Differenzen zur Verfügung gestellt werden. Hierfür existiert zwar die spezielle Wahl $\texttt{NFKEY} = 0$, weil solche Produkte auch bei der Erstellung der Jacobimatrix mit Hilfe des Seeds *am Anfang* des lokalen Zeitschrittes benötigt werden. Jedoch kann sich dies mit dem erarbeiteten Stand der HECU-Daten beißen. Z. B. muss zur Berechnung von $T_{3,1}$ ein lineares Gleichungssystem an $t_0 + 2h/3$ gelöst werden. Hierfür findet vorher eine Evaluierung von f an diesem Zeitpunkt stand, was ein Hinterherziehen der HECU-Daten hin zu diesem Zeitpunkt bewirkt. Es herrscht aus Sicht der HECU-Daten nicht die gleiche Situation wie am Anfang des lokalen Schrittes.

Bemerkung 3.71. Um mitunter kostspielige Funktionsauswertungen zu sparen, ist es im ATHLET-Kontext leider nicht ratsam, auf die modifizierte Variante des linear-impliziten Eulers, i. e. (3.45), zurückzugreifen, obwohl Steifakkuratesse hierdurch etabliert wird, s. Proposition 3.17. Auch für die Blockstufen der betrachteten $FiterRK$-Methoden wird aus gleichem Grund auf Auswertungen wie $f(t_0 + c_i h, y_0)$ verzichtet. Stattdessen wird in diesem Abschnitt für sämtliche betrachteten Verfahren für den ersten Newtonschritt von (3.133) oder (3.103) ausgegangen.

Kompensationsansatz zu Masseverlusten

Um Masseverlusten im Laufe der Gesamtrechnung entgegenzuwirken, wird parallel zur linear-impliziten Berechnung zu Vergleichszwecken das explizite Euler-Verfahren zur Integration der Massenanteile angewandt. Die nötigen Ableitungsinformationen kommen hierbei jedoch aus den f-Auswertungen zu den *implizit* bestimmten Größen. Anschließend wird auf gleiche Weise extrapoliert wie im linear-impliziten Fall.

Um eine solche Korrekturstrategie bedienen zu können, muss eine Verallgemeinerung des expliziten Vorgehens gefunden werden oder es müssen wahlweise f-Auswertungen an den benötigten Subschritten $h/2, h/3, 2h/3$ zur Verfügung stehen. Letzteres ist für die Methoden aus Abschnitt 3.4.1 nicht gegeben. Auf der anderen Seite wird ein linear-impliziter zu einem expliziten Schritt, wenn formal $J = 0$ gesetzt wird. Als mögliche Verallgemeinerung bietet es sich somit an, zu den letzten f-Auswertungen pro Stufenzeitwert c_i die entsprechenden Daten für die explizite Masseberechnung zu speichern und dann die Koeffizienten zu $y_1 = Y_i$, $i \in \{1, \ldots, s\}$, zur Linearkombination dieser gespeicherten Werte zu nutzen.

Dämpfung

In /DEU 85/ werden die Stabilitätseigenschaften des (linear-)impliziten Euler-Verfahrens mit dem Begriff *superstability* beschrieben. An jener Stelle ist dieser Begriff negativ konnotiert, da hiermit auf die extremen Dämpfungseigenschaften des Verfahrens hingewiesen wird. Diese Eigenschaft ist auf das äußerst kleine Stabilitätsgebiet zurückzuführen, welches sich zu $\{z \in \mathbb{C} \mid |1 - z| \leqslant 1\}$ ergibt. Im ATHLET-Kontext wird die Dämpfung positiv gesehen, da hiermit hochfrequente Anteile der Lösung herausgefiltert werden, welche eh nicht als allzu relevant oder sogar als hemmend im Fortschreiten der Lösung eingestuft werden. Besonders tritt diese Anforderung bei der Bestimmung der Massesträme zu Tage. Hier wird gänzlich auf die Extrapolation verzichtet und $T_{3,1}$ zum Fortführen der Lösung genutzt. Denn durch die Extrapolation wird ein Teil der Dämpfcharakteristik des Grundverfahrens herausgenommen. Auf der anderen Seite werden im ATHLET nur niedrige Extrapolationsordnungen bemüht, wodurch die Stabilitätsgebiete immer noch verhältnismäßig klein sind, s. /HAI 96, §Fig. IV.9.5/. Dies steht in direkter Relation zu den Polen der Stabilitätsfunktion. In Tab. 3.28 sind für $T_{3,3}$ sowie die 0-expliziten Methoden aus Abschnitt 3.4.1 und den in Abschnitt 3.6.2 vorgestellten Methoden die Pole der Stabilitätsfunktion angegeben. Die bisherigen Methoden reagieren sensitiver auf das analytische Verhalten der Lösung, was im klassischen Sinne positiv zu bewerten ist und durch die Ergebnisse in Unterabschnitt 3.5.2.1 untermauert wird. Jedoch ist dies kontraproduktiv, wenn bewusst auf hochfrequente Anteile der Lösung verzichtet wird. Hier treten dann die Methoden aus Abschnitt 3.6.2 in Erscheinung, welche deutlich kleinere Stabilitätsgebiete aufweisen und sich an denen der Extrapolation orientieren.

Tab. 3.26 Pol(e) von $R(z)$ für verschiedene Methoden – $FiterRK$-Methoden sind nur durch ihr Kürzel bezeichnet, s. Abschnitt 3.4.1 sowie Tab. 3.27 und 3.28

	$T_{3,3}$	32a	32b	43	32A1|A2
Pol(e), ggf. approximativ	{1,2,3}	{3.8}	{5.5}	{6.3}	{2}

3.6.1 Diskussion einer unteren Schranke zur Anzahl der f-Auswertungen

Sei in Anbetracht der Anforderungen eine 0-explizite $FiterRK$-Methode zum Ordnungspaar $3(2)$ betrachtet, welche die Ergebnisse aus Satz 3.42 nutzt, um ein Einpunktspektrum unter gewisser Kontrolle des Poles der Stabilitätsfunktion zuzusichern. Sei des Weiteren $y_1 = Y_i$ für ein $i \in \{1, \ldots, s\}$ und hiermit Steifakkuratesse gesichert. An dieser Stelle wird unabhängig von weiteren Stabilitätsforderungen diskutiert, wie weit die Anzahl $\#$ an

benötigten f-Auswertungen im Kontext der NFKEY-Steuerung reduziert werden kann.

Unter den obigen Bedingungen weist die Methode einen Block der Größe Drei auf. Um Funktionsauswertungen zu sparen, sei die Startapproximation im Block durch $Z_0 = 0$ gegeben und *ohne* Berücksichtigung der korrekten Zeitpunkte, i. .e. $\tau_i^0 = 0$ in (3.102). Es wird also auf $f(t_0, y_0)$ zugegriffen ($\# = 1$). Des Weiteren sei auf die Berechnung von $\partial f_0/\partial t$ verzichtet; nach Satz 3.26 hat dies keinen Einfluss auf die asymptotische Güte. Zwei Iterationsschritte reichen für den Block aus, wenn sich das Prinzip der Auslöschung nach Satz 3.36 zu Nutze gemacht wird. Dies resultiert in drei weiteren f-Auswertungen ($\# = 1 + 3 = 4$). Wählt man $c_1 < c_2 < c_3 < 1$, so kann mit den Blockapproximationen der Ordnung Zwei eine Approximation y_1^α gleicher Güte für den vollen Zeitschritt konstruiert werden. Diese kann als Startapproximation für eine Diagonalstufe der Ordnung Zwei sowie für eine weitere Diagonalstufe der Ordnung Drei mit $c_1 = 1$ genutzt werden. Ausschließlich eine Stufe der Ordnung Drei dem Block anschließen zu lassen funktioniert nicht, da dann nur drei Parameter zur Verfügung stehen, um den drei Ordnungsbedingungen und der zusätzlichen einzelnen Auslöschungsbedingung zu genügen. Im Sinne der benötigten f-Evaluationen erhöht die zusätzliche Diagonalstufe nicht den Aufwand, da mit einer f-Auswertung an y_1^α, ($\# = 4 + 1 = 5$), sofort hintereinander die Werte der beiden Diagonalstufen bestimmt werden können. Nach Satz 3.32 reicht ein Iterationsschritt aus. Es steht nun y_1 zur Verfügung, und die HECU-Daten stehen an $t_0 + h$. Um den f-Aufruf mit $\texttt{NFKEY} = 7$ sinnvoll ausführen zu können, muss noch einmal eine Folge von f-Auswertungen über das Intervall verteilt getätigt werden. Zum Einsetzen in f eignen sich die Blockstufenwerte, respektive zwei hiervon, wenn man sich an der Auswertungsfolge für $T_{3,1}$ orientiert. Somit kommen zwei weitere Evaluationen hinzu ($\# = 5 + 2 = 7$). Mit den abschließenden Aufrufen zu $\texttt{NFKEY} = 7$ und $\texttt{NFKEY} = 100$ ergibt sich in der Summe $\# = 7 + 2 = 9$.

Praktisch gesehen ist es sicherlich sinnvoll, $\partial f_0/\partial t$ zu berechnen, was zu einer weiteren f-Auswertung führt. Somit ergibt sich eine untere Schranke zu $\# = 9$ oder $\# = 10$.

3.6.2 Angepasste Methoden

Die in Abschnitt 3.4.1 tabellarisch dargestellten Methoden sind wie in Unterabschnitt 3.5.2.1 gezeigt von allgemeiner hoher Qualität und sind eine hervorragende Wahl für generische Aufgabenstellungen. Das NFKEY-Modell in ATHLET führt zu einem höheren Aufwand bei der Berechnung der Methodenwerte, schließt eine grundsätzliche Anwendbarkeit aber nicht aus. Ebenso kann wie diskutiert in gewisser Weise dem Kompensationsansatz zu den Masseverlusten entsprochen werden. Aber die Pole der Stabilitätsfunktion sind

festgelegt, s. Tab. 3.26, was wiederum das Dämpfungsverhalten charakterisiert. Das Problem BEAM aus Abschnitt 3.5.2 in der gestörten Variante erzeugt hohe Schwingungen in der analytischen Lösung, worauf die Methoden aus Abschnitt 3.4.1 deutlicher reagieren als der Extrapolationsansatz. Für den ATHLET-Kontext ist somit Skepsis angebracht, da Lösungskomponenten hoher Frequenz nicht aufgezeigt, sondern ausgedämpft werden sollen. Dies motivierte, die Ergebnisse aus Satz 3.42 zu nutzen, um Methoden zu konstruieren, welche deutlich kleinere Stabilitätsgebiete aufweisen, die denen der Extrapolation ähneln. Notwendig hierfür ist ein Pol z_{pol} zur Stabilitätsfunktion, welcher nicht zu weit rechts auf der reellen Achse liegt. nach einer eingehenden Analyse wurde sich für beide Verfahren für $z_{pol} = 2$ entschieden.

Bemerkung 3.72. Zur Konstruktion der angepassten Methoden wurde sich der gleichen technischen Hilfsmittel bedient, die auch zur Bestimmung der Methoden aus Abschnitt 3.4.1 genutzt wurden, s. Bemerkung 3.61 für Detailinformationen.

Eine Übersicht zu den adaptierten Methoden findet sich in Tab. 3.27 bzw. Tab. 3.28. Die Nomenklatur entspricht der aus Abschnitt 3.4.1. Hierbei wurde zu den üblichen Stufen der Mindestordnung Zwei auch je eine Stufe der Ordnung Eins, i. e. DO1, konstruiert, damit die Masseflüsse analog zum FEBE-Vorgehen gesondert behandelt werden können. Um mit der gegebenen Anzahl an Koeffizienten mehr Freiheitsgrade zur Verfügung zu haben, sich an den Stabilitätsfunktionen zur Extrapolation zu orientieren, wurde auf das Prinzip der Auslöschung verzichtet. Dennoch kann die Anzahl an Iterationsschritten für den jeweiligen Block auf Zwei reduziert werden, wenn $T_{2,1}$ zur Berechnung von Startapproximationen der Ordnung Eins genutzt wird. Der Einsatz hiervon ist möglich, da $z_{pol} = 2$ gilt und somit die Matrix in (3.97) genau der zur Berechnung von $T_{2,1}$ genutzten Matrix entspricht. Um nicht auf Blockstufenwerte verschiedener Approximationsgüten zur Berechnung der Diagonalstufen zugreifen zu müssen, wird nach dem ersten Iterationsschritt für den Block die gesamte Koeffizientenmatrix $\mathcal{A}$ als übergeordneter Block interpretiert. Der zweite Iterationsschritt wird somit zusammen mit den Diagonalstufen vorgenommen. Ausgenutzt wird hierbei wieder Satz 3.26. Notwendig hierfür ist die Bereitstellung an Transformationsdaten zu $\mathcal{A}^{-1}$ analog zu (3.93). Zur Herleitung sei $\mathcal{A}$ dargestellt über

$$\mathcal{A} = \begin{pmatrix} A_m & \\ B & L \end{pmatrix}$$

mit einer regulären unteren linken Dreiecksmatrix L. Aus (3.93) sind bereits die Transformationsgrößen T und Λ zu A_m bekannt. Eine kurze Rechnung liefert

$$\mathcal{A}^{-1} = \begin{pmatrix} A_m^{-1} & \\ -L^{-1}BA_m^{-1} & L^{-1} \end{pmatrix},$$

und mit

$$\mathcal{T} := \begin{pmatrix} T & \\ & I \end{pmatrix}$$

ergibt sich schließlich

$$\Lambda_{\mathcal{A}} := \mathcal{T}^{-1}\mathcal{A}\mathcal{T} = \begin{pmatrix} \Lambda & \\ -L^{-1}BA_m^{-1}T & L^{-1} \end{pmatrix}.$$

Da es sich hierbei um reine Methodenwerte handelt, können $\mathcal{T}$ und $\Lambda_{\mathcal{A}}$ im Vorfelde berechnet werden. Man beachte, dass durch diese spezielle Vorgehensweise die sonst für Diagonalstufen üblichen Rechengrößen wie `AlphAinvTI` und `nDinvGamAinvTI` aus (3.144) bzw. (3.145) nicht benötigt werden und ausschließlich Iterationsvorschriften der Form (3.96) oder (3.103) auftreten.

Bemerkung 3.73. Aufgrund der Orientierung an den Stabilitätsfunktion zu Extrapolation sollte es nicht erstaunen, dass im Vergleich zu den Methoden aus Abschnitt 3.4.1 die Fehlerkonstanten zu $FiterRK32A1|2$ von geringerer Güte sind. Das sollte nicht als Nachteil gewertet werden, da dies durchaus im Sinne der Dämpfung sein mag. Man vergesse hierbei auch nicht, dass gegenüber dem ATHLET-Extrapolationsansatz trotzdem die rechnerischen Vorteile eines Einpunktspektrums gegeben sind sowie die hohe Stufenordnung, welche die hier vorgestellten Methoden robust gegenüber dem Phänomen der Ordnungsreduktion macht.

Tab. 3.27 Stufeneigenschaften zur $FiterRK32A1$-Methode

i	$c_i \approx$	Stufe	A-stab	$R(\infty)$	StA	$R(x<0)>0$	Niter
1	$1/3$	B1	✓	0^+	✓	✓	3
2	$(3-\sqrt{3})/2$	B2	✓	0^+	✓	✓	3
3	1	B3|D1α	✓	0^-	✓	–	3
4	1	D1	✓	0^-	✓	–	1
5	1	DO1	✓	0^+	✓	✓	1

$\mathcal{C}(2)$ und $\mathcal{C}_\alpha(2)$ sind erfüllt (von DO1 abgesehen),

$$C_{err}^{B3} = 1/24, \qquad C_{err}^{D1} = 1/48, \qquad \frac{C_{err}^{D1}}{C_{err}^{B3}} = 1/2$$

f-Auswertungen

Unter der Prämisse, dass $T_{2,1}$ konstruiert wird, um Startnäherungen für den Block A_m zu erzeugen, bedarf es insgesamt 13 Auswertungen von f für $FiterRK32A1$ und 12 Aufrufe für $FiterRK32A2$. Letztere Methode birgt den Vorteil, die Evaluation an $t_0 + h/2$

Tab. 3.28 Stufeneigenschaften zur $FiterRK32A2$-Methode

i	$c_i \approx$	Stufe	A-stab	$R(\infty)$	StA	$R(x<0)>0$	Niter
1	0.209	B1	–	0^+	✓	✓	3
2	$1/2$	B2	✓	0^+	✓	✓	3
3	1	B3|D3α|D4α	✓	0^-	✓	–	3
4	$1/3$	D1α	✓	0^+	–	✓	
		D1	✓	0^+	✓	✓	1
5	$2/3$	D2α	✓	0^+	–	✓	
		D2	✓	0^+	✓	✓	1
6	1	D3	✓	0^+	✓	–	1
7	1	D4	✓	0^+	✓	–	1
8	1	DO1	✓	0^+	✓	✓	1

$\mathcal{C}(2)$ und $\mathcal{C}_\alpha(2)$ sind erfüllt (von DO1 abgesehen),
$C_{err}^{D3} = 1/32, \qquad C_{err}^{D4} = 1/120, \qquad \frac{C_{err}^{D4}}{C_{err}^{D3}} = 4/15$

während der Berechnung von $T_{2,1}$ wiederverwenden zu können. Im Verhältnis zu der in Abschnitt 3.6.1 diskutierten unteren Schranke von 10 Auswertungen bei Nutzung von $\partial f_0/\partial t$ ist also ein leichter Anstieg zu verzeichnen, der aber noch moderat ausfällt. Auf der anderen Seite sind merklich mehr Funktionsauswertungen vonnöten als beim FEBE-Ansatz, obwohl bereits durch die Mehrfachverwendung der f-Auswertung am Stufenwert zu B3 gespart wird. Dem gegenüber steht der Vorteil eines Einpunktspektrums, was sich wiederum kompensierend auf den Zeitbedarf auswirkt, s. hierzu auch Bemerkung 3.11.

Bemerkung 3.74. Um Funktionsauswertungen zu sparen, wird nicht wie bei den Methoden in Abschnitt 3.4.1 die Stufe zur Fehlerschätzung als Input für die abschließende Stufe genutzt. Folglich kann nicht auf die Darstellung (3.140) der Fehlerschätzung zuückgegriffen werden. Stattdessen muss zusätzlich (3.139) berechnet werden.

Einsatz im ATHLET-Kontext

Aufgrund der in Abschnitt 2.3 aufgezeigten Hemmnisse sowie der im letzten Abschnitt von Unterabschnitt 2.3.1.5 angesprochenen Wechselwirkungen der Kontrollstrukturen in FEBE zur Steuerung eines Integrationsschrittes verzögert sich noch der Einsatz der neuen Methoden über das Reverse-Communication-Interface im ATHLET/`NuT`-Kontext. Die Architektur zum Ausführen einer $FiterRK$-Methode nach Vorbild der Algorithmen 3.2-3.4 ist soweit vorhanden, allein die Berücksichtigung *aller* ATHLET-Spezifika steht noch aus, um stabile Simulationsläufe zu gewährleisten.

4 Kopplung mit anderen Codes — Erstellung einer stabilisierten Schnittstelle, Test iterativer Solver für Kopplungen

4.1 Anpassung der Inhalte

Dieser Arbeitspunkt war Teil des urspünglichen Angebotes und wurde im Laufe des Jahres 2016 durch den in Kapitel 5 präsentierten Themenkomplex zur Kopplung von Codes ersetzt. An dieser Stelle werden die bis zum Wechsel der Inhalte durchgeführten Arbeiten vorgestellt. Diese sind theoretischer Natur und können als Grundlage für weiterführende Überlegungen dienen.

4.2 Anforderungen und Ergebnisse

Im Mittelpunkt des urspünglichen Arbeitspaketes stand die Idee, einen Kopplungsmechanismus zu etablieren, der eine bessere Stabilität und eine höhere Ordnung als eine explizite lose Kopplung gewährleistet. Als Anwendungsfall war die Verknüpfung von ATHLET mit QUABOX/CUBBOX (Q/C) vorgesehen, welche einen solchen expliziten Ansatz über ein s. g. *staggered time synchronisation scheme*, /PÉR 11/, realisieren. Der ATHLET-Code vollzieht einen lokalen Zeitschritt von t_0 zu $t_0 + h$ und nutzt hierbei die Systeminformationen seitens Q/C an t_0. Nach Ermittlung der ATHLET-Werte an $t_0 + h$ werden diese an Q/C gesandt. Anschließend werden diese Daten genutzt, um im Q/C-Kontext eine Zeitintegration von t_0 zu $t_0 + h$ durchzuführen. Dieses Vorgehen wird wiederholt, bis ein vorgegebener Endzeitpunkt erreicht ist. Da ATHLET in einem Integrationsschritt nur mit konstanten Eingangsdaten seitens Q/C arbeitet, kann hinsichtlich des Gesamtsystems nicht erwartet werden, dass die Approximationsgüte im lokalen Schritt die Ordnung Eins überschreitet.

Geleistetes

Um eine höhere Ordnung zu erzielen, bedarf es einer engeren Kopplung. Ein möglicher Kandidat, um dies zu erreichen, ist der s. g. *Tangential Block Newton* (TBN)-Ansatz aus /MAC 99/. Dieses Verfahren ermöglicht eine implizite Kopplung der Systeme, was jedoch mit entsprechendem Mehraufwand in der Bestimmung der Systemgrößen einhergeht. Im Rahmen der getätigten Arbeiten zu diesem Arbeitspunkt wurden theoretische Untersuchungen zur Anwendbarkeit des TBN-Ansatzes gemacht. Der Fokus lag hierbei auf einer Formulierung des Kopplungsproblems auf Basis der durch die jeweiligen

Programme zur Verfügung gestellten numerischen Algorithmen in Verbindung mit einem Interpolationsansatz. Dies wurde in einem abstrakten Rahmen durchgeführt, um grundsätzliche Lösbarkeit und Ordnungseigenschaften aufzuzeigen. Man beachte, dass eine Beschränkung auf den TBN-Ansatz hierbei nicht notwendig ist, s. Bemerkung 4.1. ATHLET-Spezifika im Rahmen dieser Modellierung sind in Abschnitt 4.3.1 diskutiert.

Es sei angemerkt, dass im Rahmen anderer Arbeiten in der GRS die softwaretechnische Ausprägung der vorhandenen Kopplung von Q/C und ATHLET eine Grundüberholung erfuhr. Diese basiert auf der Nutzung des mithilfe der libadt zur Verfügung gestellten Plug-In-Konzepts zu ATHLET, s. Abschnitt 2.3 für den Einsatz der libadt in diesem Projekt. Aufgrund der Plug-In-Architektur wird die programmiertechnische Anbindung als auch das gegenseitige Zurverfügungstellen von Daten erheblich vereinfacht. Gerade Letzterem könnte eine Schlüsselrolle bei etwaigen zukünftigen Arbeiten zur stabileren Gestaltung der ATHLET-Q/C-Kopplung zukommen.

4.3 Approximationsmodell zum TBN-Ansatz

Um den TBN-Ansatz anwenden zu können, bedarf es einer adäquaten numerischen Modellierung der gekoppelten zeitvarianten Systeme. In den folgenden Erörterungen wird diese Modellierung zunächst recht allgemein gehalten. Eine Konkretisierung bzgl. ATHLET findet sich in Abschnitt 4.3.1. Auch wenn Interesse an dem Zusammenspiel der beiden Systeme über einen längeren Zeitraum besteht, so ist das hier vorgestellte Approximationsmodell lokal ausgerichtet, indem es einen Schritt $t_0 \to t_0 + h$ für $h > 0$ betrachtet. Dies ist dadurch motiviert, dass davon ausgegangen wird, dass beiden System jeweils ein Algorithmus zur Verfügung steht, welcher ausgehend vom zugehörigen Systemzustand an t_0 eine Approximation zum Zustand an $t_0 + h$ generiert. Aufgrund der Kopplung bedarf es Informationen des jeweils anderen Systems. Dieser Anforderung wird sich im Modell mittels eines Interpolationsansatzes angenommen. Die wesentliche Aussage in diesem Abschnitt ergibt sich darin, welche Approximationsgüte zu erwarten ist, wenn ein solcher Ansatz verfolgt wird. Um das Approximationsmodell adäquat diskutieren zu können, werden einige mathematische Konkretisierungen und Strukturen benötigt.

Modellherleitung

Zur Unterscheidung der beiden Systeme, sei das eine als x-System und das zweite als y-System bezeichnet. Entsprechend werden die zeitlichen Verläufe der Systeme, welche

bereits die Wechselwirkung berücksichtigen, über

$$x : [t_0, t_0 + h] \to \mathbb{R}^{n_x} \qquad \text{bzw.} \qquad y : [t_0, t_0 + h] \to \mathbb{R}^{n_y}$$

definiert. Gegebenenfalls ist hierin bereits eine örtliche Diskretisierung orts- und zeitabhängiger Systeme berücksichtigt. Die Genauigkeit einer solchen Diskretisierung spiegelt sich in der Größe der Dimensionszahlen n_x und n_y wieder. Diese seien im Rahmen der folgenden Betrachtungen als feste Werte vorgegeben. Des Weiteren seien $x_0 := x(t_0)$ und $y_0 := y(t_0)$ als bekannt sowie x und y als hinreichend glatt vorausgesetzt.

Hinsichtlich der numerischen Approximation von $x(t_0 + h)$ und $y(t_0 + h)$ wird davon ausgegangen, dass sich die Algorithmen wie folgt darstellen lassen: Ermittle für festes h Zwischengrößen Z_x und Z_y mittels

$$\begin{aligned} \phi_x(Z_x, u_y; h) &= 0 \\ \phi_y(Z_y, u_x; h) &= 0 \end{aligned} \tag{4.1}$$

und berechne hiermit Approximationen x_1 und y_1 zu $x(t_0 + h)$ bzw. $y(t_0 + h)$ über

$$\begin{aligned} x_1 &= x_0 + \delta_x(Z_x) \\ y_1 &= y_0 + \delta_y(Z_y). \end{aligned} \tag{4.2}$$

Die involvierten Funktionen

$$\begin{aligned} \phi_x &: \mathbb{R}^{m_x} \times \mathbb{R}^{k_y} \to \mathbb{R}^{m_x} \\ \phi_y &: \mathbb{R}^{m_y} \times \mathbb{R}^{k_x} \to \mathbb{R}^{m_y} \end{aligned} \qquad \text{und} \qquad \begin{aligned} \delta_x &: \mathbb{R}^{m_x} \to \mathbb{R}^{n_x} \\ \delta_y &: \mathbb{R}^{m_y} \to \mathbb{R}^{n_y} \end{aligned} \tag{4.3}$$

werden ebenfalls als glatt vorausgesetzt. Des Weiteren ist es sinnvoll, von folgender Beziehung auszugehen

$$h \to 0 \quad \Rightarrow \quad \begin{cases} Z_x \to 0 & \Rightarrow \quad \delta_x(Z_x) \to 0 \\ Z_y \to 0 & \Rightarrow \quad \delta_y(Z_y) \to 0. \end{cases} \tag{4.4}$$

Dies sichert Konsistenz in dem Sinne, dass das Systemverhalten korrekt wiedergegeben wird, wenn keine Zeitevolution stattfindet. Der Einfluss des jeweils anderen Systems wird durch die Variablen u_x und u_y dargestellt. Für zwei Sonderfälle können die numerischen Algorithmen dennoch als entkoppelt betrachtet werden:

- u_x und u_y können auf x und y zugreifen:
  Unter diesen Bedingungen kann in den Algorithmen mit exakten Informationen aus dem jeweils anderen System gearbeitet werden und die Genauigkeit der Approximationen x_1 und y_1 entspricht den Ordnungen der Verfahren, die (4.1) und (4.2) zugrunde liegen.

- u_x und u_y hängen ausschließlich von x_0 und y_0 ab:
  In diesem Fall wird die Dynamik des jeweils anderen Systems im Intervall $[t_0, t_0 + h]$ ignoriert. Da $x(t_0 + \xi_i h) - x_0 \in \mathcal{O}(h)\ \forall\, \xi_i \in [0, 1]$ gilt, kann nicht mehr als $x(t_0 + h) - x_1 \in \mathcal{O}(h^2)$ erwartet werden. Analoges ergibt sich für das y-System.

Beide Fälle bieten keine sinnvollen Handlungsoptionen. Im ersten Fall bedarf es der exakten Werte x und y, um die Approximationsmächtigkeit der numerischen Algorithmen voll auszuschöpfen. Im zweiten Fall ergibt sich eine sehr dürftige Approximationsgüte.

Selbst wenn exakte Informationen zur Verfügung stehen, benötigen die Größen u_x und u_y in der Regel nicht die kompletten (Funktions-)vektoren x und y. Seien die für die Kopplung (coupling) relevanten Komponenten mit x_c bzw. y_c benannt. Der hier vorgestellte Ansatz basiert auf der Interpolation dieser Daten mit Hilfe der Zwischengrößen Z_x und Z_y aus (4.1). Hierfür seien die Interpolationsabbildungen

$$\begin{aligned} p &: [t_0, t_0 + h] \times \mathbb{R}^{\mu_y} \to \mathbb{R}^{\ell_y} \qquad \text{mit} \quad p(t_0, \cdot) = y(t_0) \\ q &: [t_0, t_0 + h] \times \mathbb{R}^{\mu_x} \to \mathbb{R}^{\ell_x} \qquad \text{mit} \quad q(t_0, \cdot) = x(t_0) \end{aligned} \tag{4.5}$$

sowie passende Projektoren

$$\Pi \in \mathbb{R}^{\mu_x \times m_y} \qquad \Xi \in \mathbb{R}^{\mu_y \times m_x}$$

gegeben, so dass sich die Approximationen

$$p(t_0 + dh, \Pi Z_y) \approx y_c(t_0 + dh) \qquad \text{und} \qquad q(t_0 + ch, \Xi Z_x) \approx x_c(t_0 + ch)$$

für $c, d \in [0, 1]$ ergeben. Es wird davon ausgegangen, dass für die numerische Kopplung in (4.1) nur endlich viele Sample-Daten von x_c und y_c benötigt werden. Diese werden zusammengefasst zu

$$P(h, \Pi Z_y) := \begin{pmatrix} p(t_0 + d_1 h, \Pi Z_y) \\ \vdots \\ p(t_0 + d_r h, \Pi Z_y) \end{pmatrix} \qquad \text{und} \qquad Q(h, \Xi Z_x) := \begin{pmatrix} q(t_0 + c_1 h, \Xi Z_x) \\ \vdots \\ q(t_0 + c_s h, \Xi Z_x) \end{pmatrix},$$

wobei

$$\ell_x \cdot s = k_x \qquad \text{sowie} \qquad \ell_y \cdot r = k_y$$

gilt. Setzt man diese Größen für u_x und u_y in (4.1) ein, ergibt sich schließlich das System

$$\begin{aligned} \phi_x(Z_x, P(h, \Pi Z_y); h) &= 0 \\ \phi_y(Z_y, Q(h, \Xi Z_x); h) &= 0 \end{aligned} \tag{4.6}$$

in den Unbekannten Z_x und Z_y.

Bemerkung 4.1. Das System (4.6) weist $m_x + m_y$ Gleichungen in ebenso vielen Unbekannten auf. Somit kann dieses prinzipiell von jedem Verfahren angegangen werden, welches nichtlineare Gleichungssysteme löst. Eine Beschränkung auf den TBN-Ansatz ist dann sinnvoll, wenn nicht das zweiteilige System als ein Gesamtsystem behandelt werden kann und die gegenseitigen Abhängigkeiten moderat ausfallen.

Lösbarkeit des diskreten Systems (4.6)

Wie eingangs erwähnt, wird (4.6) für festes $h > 0$ betrachtet. In Hinblick auf die Frage der Lösbarkeit von (4.6) bietet es sich an, eine h-Varianz zuzulassen. Sei

$$\Psi : \mathbb{R}^{m_x+m_y} \times \mathbb{R} \to \mathbb{R}^{m_x+m_y}$$

definiert über

$$\Psi(Z,h) := \begin{pmatrix} \phi_x(Z_x, P(h, \Pi Z_y); h) \\ \phi_y(Z_y, Q(h, \Xi Z_x); h) \end{pmatrix} \qquad \text{mit} \qquad Z := \begin{pmatrix} Z_x \\ Z_y \end{pmatrix}.$$

Aus (4.4) folgt

$$\Psi(Z,h)_{|Z=\hat{Z},h=\hat{h}} = 0 \qquad \text{für} \qquad \hat{Z} = 0 \quad \text{und} \quad \hat{h} = 0.$$

Unter der zusätzlichen Annahme

$$\frac{\partial \Psi}{\partial Z}_{|Z=\hat{Z},h=\hat{h}} \quad \text{ist regulär} \tag{4.7}$$

ergibt sich aus dem Satz über implizite Funktionen, s. z. B. /VOSS 96, Satz 23.9/, die Aussage

$$\begin{aligned} &\exists \text{Umgebungen } \mathcal{U}(\hat{h}) \subset \mathbb{R}, \mathcal{U}(\hat{Z}) \subset \mathbb{R}^{m_x+m_y} : \\ &\qquad \forall h \in \mathcal{U}(\hat{h})\ \exists! Z \in \mathcal{U}(\hat{Z}), \text{ so dass (4.6) erfüllt ist.} \end{aligned} \tag{4.8}$$

Zu ähnlichen Resultaten kommt man, wenn man Standardresultate zum Newtonverfahren in diesem Kontext anwendet, s. z. B. /DEU 04/. Die Regularitätsannahme (4.7) sollte keine allzu große Hürde darstellen. Wie in Abschnitt 4.3.1 gezeigt wird, ist ein ATHLET-Beitrag hierzu leicht erkennbar, s. (4.21).

Approximationsgüte

Es stellt sich die Frage, welche Güte man von den Approximationen x_1 und y_1 in (4.2) erwarten kann, wenn Z_x und Z_y über (4.6) ermittelt werden. Die Schwierigkeit hierbei ist, die rekursiven Abhängigkeiten in den Griff zu bekommen. An dieser Stelle wird wie folgt vorgegangen:

Es ist realistisch anzunehmen, dass sich $x(t_0+h)$ und $y(t_0+h)$ für gegebenes $h>0$ darstellen lassen über

$$\begin{aligned} x(t_0+h) &= x_0 + \int_{t_0}^{t_0+h} g\Big(s, x(s), y_c(s)\Big) ds \\ y(t_0+h) &= y_0 + \int_{t_0}^{t_0+h} f\Big(s, y(s), x_c(s)\Big) ds \end{aligned} \tag{4.9}$$

mit geeigneten Funktionen g und f. Die Funktion g sei Lipschitz-stetig bzgl. x sowie bzgl. y_c. Analoges möge für f gelten. Es wird davon ausgegangen, dass zu $h>0$ die Größen Z_x und Z_y bereits über (4.6) bestimmt sind. Hiermit und mit den Interpolationsabbildungen p und q aus (4.5) seien folgende Funktionen wohldefiniert

$$\begin{aligned} \tilde{x}(t_0+h) &= x_0 + \int_{t_0}^{t_0+h} g\Big(s, \tilde{x}(s), p(s, \Pi Z_y)\Big) ds \\ \tilde{y}(t_0+h) &= y_0 + \int_{t_0}^{t_0+h} f\Big(s, \tilde{y}(s), q(s, \Xi Z_x)\Big) ds \end{aligned} \tag{4.10}$$

sowie die abgeleiteten Größen

$$\begin{aligned} \mathring{x}(t_0+h) &= x_0 + \int_{t_0}^{t_0+h} g\Big(s, x(s), p(s, \Pi Z_y)\Big) ds \\ \mathring{y}(t_0+h) &= y_0 + \int_{t_0}^{t_0+h} f\Big(s, y(s), q(s, \Xi Z_x)\Big) ds \end{aligned} \tag{4.11}$$

bestimmt. Die Idee hinter der Einführung von $\tilde{x}$ und $\tilde{y}$ ist, dass der Einfluss des jeweils anderen Systems über den Interpolationsansatz mit über (4.6) bestimmten Z_x und Z_y geschieht und somit je als zeitexpliziter Input zur Verfügung steht. Die Gleichungen in (4.6) zusammen mit den Berechnungen in (4.2) sind eine diskrete Analogie hierzu. Wenn für gegebenen y-Input der numerische Algorithmus zur Bestimmung von x_1 die lokale Ordnung η_x besitzt, dann gilt somit

$$\tilde{x}(t_0+h) - x_1 \in \mathcal{O}(h^{\eta_x}) \tag{4.12a}$$

und entsprechend

$$\tilde{y}(t_0+h) - y_1 \in \mathcal{O}(h^{\eta_y}), \tag{4.12b}$$

wobei η_y die lokale Ordnung des Algorithmus im y-System bezeichnet. Um die Approximationsgüte von x_1 und y_1 in Bezug auf x und y zu untersuchen, kann also stattdessen der Fehler von $\tilde{x}$ und $\tilde{y}$ zu x bzw. y betrachtet werden. Die Werte $\mathring{x}$ und $\mathring{y}$ aus (4.11) treten hierbei als Hilfsgrößen auf, um die verschiedenen Lipschitz-Bedingungen auszunutzen.

Es gilt

$$\begin{aligned}\left\|\begin{pmatrix}\mathring{x}(t_0+h)-\tilde{x}(t_0+h)\\ \mathring{y}(t_0+h)-\tilde{y}(t_0+h)\end{pmatrix}\right\|_2 &\leqslant \int_{t_0}^{t_0+h}\left\|\begin{pmatrix}g\Big(s,x(s),p(s,\Pi Z_y)\Big)\\ f\Big(s,y(s),q(s,\Xi Z_x)\Big)\end{pmatrix}\right. \\ &\qquad\left.-\begin{pmatrix}g\Big(s,\tilde{x}(s),p(s,\Pi Z_y)\Big)\\ f\Big(s,\tilde{y}(s),q(s,\Xi Z_x)\Big)\end{pmatrix}\right\|_2 ds \\ &\leqslant L_1\cdot\int_{t_0}^{t_0+h}\left\|\begin{pmatrix}x(s)-\tilde{x}(s)\\ y(s)-\tilde{y}(s)\end{pmatrix}\right\|_2 ds\end{aligned} \tag{4.13}$$

für angemessenes $L_1>0$. Auf der anderen Seite ergibt sich aus der Lipschitz-Stetigkeit hinsichtlich des Kopplungs-Inputs und für passendes $L_2>0$ die Abschätzung

$$\begin{aligned}\left\|\begin{pmatrix}x(t_0+h)-\mathring{x}(t_0+h)\\ y(t_0+h)-\mathring{y}(t_0+h)\end{pmatrix}\right\|_2 &\leqslant \int_{t_0}^{t_0+h}\left\|\begin{pmatrix}g\Big(s,x(s),y_c(s)\Big)\\ f\Big(s,y(s),x_c(s)\Big)\end{pmatrix}\right. \\ &\qquad\left.-\begin{pmatrix}g\Big(s,x(s),p(s,\Pi Z_y)\Big)\\ f\Big(s,y(s),q(s,\Xi Z_x)\Big)\end{pmatrix}\right\|_2 ds \\ &\leqslant L_2\cdot\int_{t_0}^{t_0+h}\left\|\begin{pmatrix}y_c(s)-p(s,\Pi Z_y)\\ x_c(s)-q(s,\Xi Z_x)\end{pmatrix}\right\|_2 ds.\end{aligned} \tag{4.14}$$

Ähnlich argumentierend wie zur Herleitung von (4.12) können die Interpolationsabbildungen derart gewählt werden, dass

$$\forall s\in[t_0,t_0+h]:\begin{cases}\tilde{x}_c(s)-q(s,\Pi Z_x)\in\mathcal{O}(h^{\nu_x}), & \nu_x\leqslant\eta_x,\\ \tilde{y}_c(s)-p(s,\Xi Z_y)\in\mathcal{O}(h^{\nu_y}), & \nu_y\leqslant\eta_y,\end{cases} \tag{4.15}$$

gilt. Hierbei sind $\tilde{x}_c$ und $\tilde{y}_c$ die für die Kopplung relevanten Komponenten von $\tilde{x}$ bzw. $\tilde{y}$. Mit Hilfe von (4.15) lässt sich der letzte Integrand in (4.14) abschätzen über

$$\begin{aligned}\left\|\begin{pmatrix}y_c(s)-p(s,\Pi Z_y)\\ x_c(s)-q(s,\Xi Z_x)\end{pmatrix}\right\|_2 &\leqslant \left\|\begin{pmatrix}y_c(s)-\tilde{y}_c(s)\\ x_c(s)-\tilde{x}_c(s)\end{pmatrix}\right\|_2+\mathcal{O}(h^{\min(\nu_x,\nu_y)}) \\ &\leqslant \alpha\left\|\begin{pmatrix}y(s)-\tilde{y}(s)\\ x(s)-\tilde{x}(s)\end{pmatrix}\right\|_2+\mathcal{O}(h^{\min(\nu_x,\nu_y)})\end{aligned} \tag{4.16}$$

mit einem $\alpha\in(0,1]$. Fasst man (4.13) bis (4.16) zusammen ergibt sich

$$\begin{aligned}\left\|\begin{pmatrix}x(t_0+h)-\tilde{x}(t_0+h)\\ y(t_0+h)-\tilde{y}(t_0+h)\end{pmatrix}\right\|_2 &\leqslant 2\max(L_1,\alpha\cdot L_2)\int_{t_0}^{t_0+h}\left\|\begin{pmatrix}y(s)-\tilde{y}(s)\\ x(s)-\tilde{x}(s)\end{pmatrix}\right\|_2 ds \\ &\quad+L_2\cdot\mathcal{O}(h^{\min(\nu_x,\nu_y)+1}),\end{aligned} \tag{4.17}$$

und aus dem Lemma von Gronwall, s. z. B. /VOSS 96, Satz 27.12/, folgt

$$\left\| \begin{pmatrix} x(t_0+h) - \tilde{x}(t_0+h) \\ y(t_0+h) - \tilde{y}(t_0+h) \end{pmatrix} \right\|_2 \leqslant L_2 \cdot \mathcal{O}(h^{\min(\nu_x,\nu_y)+1}) \cdot \exp\Big(2\max(L_1, \alpha \cdot L_2)h\Big). \quad (4.18)$$

Hiernach und nach (4.12) erhält man schließlich für $h \to 0$

$$\left.\begin{matrix} x(t_0+h) - x_1 \\ y(t_0+h) - y_1 \end{matrix}\right\} \in \mathcal{O}(h^{\varphi}) \quad \text{mit} \quad \varphi = \min\Big[\min(\nu_x, \nu_y) + 1, \min(\eta_x, \eta_y)\Big]. \quad (4.19)$$

Dieses Ergebnis liefert zwei nützliche Informationen:

- Die lokalen Ordnungen η_x und η_y der beiden involvierten Algorithmen sollte nicht zu weit auseinander liegen, da die geringere Ordnung das Gesamtverhalten dominiert.
- Die Interpolation kann es sich gegenüber den eigentlichen numerischen Algorithmen zur Bestimmung von x_1 und y_1 leisten, eine um Eins reduzierte lokale Ordnung zu etablieren, ohne die Güte für das Gesamtsystem zu schmälern. Nutzt man dies aus, kann es zu einer Verringerung der Anzahl an Z_x- und Z_y-Komponenten zur Bestimmung von q bzw. p führen. Es werden also weniger Daten des jeweils anderen Systems benötigt.

4.3.1 Das ATHLET-System im Approximationsmodell

Unabhängig davon, ob der bisher im ATHLET verwandte und in Abschnitt 3.2 diskutierte Extrapolationsansatz oder die in Abschnitt 3.6.2 eingeführten Methoden betrachtet werden, handelt es sich um einen $FiterRK$-Ansatz: Es werden eine a priori festgelegte Anzahl an Newtonschritten der Form (3.87) ausgeführt, um eine Approximation zu Z aus (3.86) zu erhalten. Anschließend wird (ggf. nur formal) über (3.135) eine Approximation für $y(t_0+h)$ generiert. In diesem Abschnitt sei diese mit $\hat{y}_1$ bezeichnet, wohingegen y_1 das Ergebnis von (3.135) für Z aus (3.86) darstellt. Entsprechend dieser Notation sei im Rahmen einer Kopplung das ATHLET-System als das im vorherigen Abschnitt deklarierte y-System gegeben.

Obschon ein $FiterRK$-Ansatz verfolgt wird, ist es nicht sinnvoll, ϕ_y in (4.1) mit dem hiermit verbundenen Newtonprozess zu identifizieren. Vielmehr bietet es sich an, für ϕ_y das voll implizite System (3.86) zu Grunde zu legen und das $FiterRK$-Konzept schlicht als endlich-iterativen Prozess zur Approximation von (3.86) zu sehen. Weist die voll implizite Variante die gleiche Ordnung auf wie die $FiterRK$-Adaption – was für alle hier betrachteten Methoden der Fall ist –, entsteht kein Güteverlust. Denn dann gilt auch

$$\tilde{y}(t_0+h) - \hat{y}_1 \in \mathcal{O}(h^{\eta_y})$$

analog zu (4.12b), was sich entweder direkt aus den Ordnungseigenschaften zur $FiterRK$-Methode zeigt oder sich aus $y_1 - \hat{y}_1 \in \mathcal{O}(h^{\eta_y})$ ergibt. Im Sinne der Kopplung wird somit basierend auf (3.86) und (3.135)

$$\begin{aligned} \phi_y(Z_y, u_x; h) &= Z_y - h(\mathcal{A} \otimes I)\mathcal{F}(Z_y; u_x) \\ \delta_y(Z_y) &= (b^T \otimes I)(\mathcal{A}^{-1} \otimes I)Z_y \end{aligned} \tag{4.20}$$

gesetzt. Dies schließt eine formale Erweiterung der Funktion $\mathcal{F}$ aus (3.84) um den Input u_x des zweiten Systems ein. Die Verknüpfung von ϕ_y zum voll impliziten System hat den Vorteil, dass sich die Gültigkeit von (4.4) für das y-System sehr leicht nachvollziehen lässt. Ähnlich verhält es sich bzgl. des Lösbarkeitskriteriums (4.7). Denn aus (4.20) folgt sofort

$$\frac{\partial \Psi}{\partial Z}_{|Z=\hat{Z},h=\hat{h}} = \begin{pmatrix} * & * \\ 0 & I \end{pmatrix} \tag{4.21}$$

Der ATHLET-Beitrag liefert also die Vereinfachung

$$\frac{\partial \Psi}{\partial Z}_{|Z=\hat{Z},h=\hat{h}} \text{ ist regulär} \quad \Leftrightarrow \quad \frac{\partial \Psi}{\partial Z_x}_{|Z=\hat{Z},h=\hat{h}} \text{ ist regulär.}$$

Man beachte, dass eine direkte Verknüpfung von ϕ_y mit dem Newtonprozess einer $FiterRK$-Methode ein Ableiten dieses gesamten Prozesses nötig machen würde, um eine Information wie (4.21) zu erhalten. Aufgrund von $y_1 - \hat{y}_1 \in \mathcal{O}(h^{\eta_y})$ besteht hieran jedoch kein Bedarf.

Für den Interpolationsansatz kann das ATHLET-System auf alle Fälle die Daten y_0 sowie $f(t_0, y_0, x_0)$ zur Verfügung stellen. Dies erlaubt eine lineare Interpolation. Reicht dies im Rahmen der globalen Ordnungsanforderungen bereits aus, wäre das x-System nicht implizit vom ATHLET-System abhängig. Auf der anderen Seite sollte sich ein deutlich stabileres Verhalten zeigen, wenn für die Interpolation mindestens auch auf Informationen an $t_0 + h$ zugegriffen werden kann. Dies geht verhältnismäßig effizient für Verfahren, welche (3.137) erfüllen, da dann die Projektionsmatrix Π sehr viel mehr Komponenten herausfiltern kann und sich entsprechend die Dimension k_y von u_y verkleinert.

5 Berücksichtigung paralleler Datenstrukturen und Operationen sowie Anbindung der numerischen Strukturen mittels MPI

5.1 Einbringung neuer Inhalte

Dieser Arbeitspunkt wurde im Laufe des Jahres 2016 dem Aufgabencanon hinzugefügt. Im Gegenzug wurden die Arbeiten zu Kapitel 4 ausgesetzt. Die Motivation, sich für eine Änderung des Arbeitsplanes auszusprechen, lag zum einen darin, für die in Abschnitt 2.3.1 beschriebene und für dieses Projekt etablierte `NuT`-Architektur das volle Potential der in Abschnitt 2.2 vorgestellten numerischen Bibliotheken hinsichtlich verteilter Speicherung und Rechnung ausschöpfen zu wollen. Zum anderen sollte bestehender Code möglichst getrennt von der Anbindung an die externen Bibliotheken gehalten werden, um Flexibilität und einfache Wartbarkeit zu wahren sowie bestehenden Code nicht unnötig zu verkomplizieren.

5.2 Anforderungen und Ergebnisse

Die in Abschnitt 5.1 beschriebene Motivation zur Änderung der Arbeitsinhalte, charakterisiert zeitgleich die Zielsetzung für diesen Arbeitspunkt.

Man beachte, dass dem Aspekt der Trennung von Inhalten eine besondere Bedeutung zufällt, da ein naives Ausnutzen der MPI-basierten Parallelität in PETSc mit einer n-fachen Ausführung des ATHLET-Codes einherginge. Dies wäre wenig effizient, da der ATHLET-Code im Wesentlichen serieller Art ist. Es würde nicht gemeinsam gerechnet werden, sondern n-mal das Gleiche. Um sich dieser nicht trivialen Redundanz zu entledigen, könnte man spezielle Schalter in den Bestandscode einbauen, welche neben einem stets laufenden Hauptprozess die anderen $n-1$ Prozesse erst dann zuschalten, wenn numerische Rechnungen anstünden und PETSc diese auch nutzen könnte. Dies wäre jedoch ein recht diffiziles Unterfangen und entspräche nicht dem in Abschnitt 2.3.1 beschriebenen Architektur-Paradigma des Minimal-Invasiven.

Geleistetes

Um den Anforderungen genüge zu tun, wurde der MPI-Standard neben der PETSc-basierten Beschleunigung von numerischen Berechnungen im Sinne der Möglichkeit zu einer *Inter-Process-Communication* genutzt. Diese Funktionalität wird nicht durch PETSc-Direktiven abgedeckt. Stattdessen war eigene MPI-Programmierung vonnöten. Eine

allgemeine Erläuterung hierzu im Rahmen der NuT-Architektur findet sich in Abschnitt 2.3. Eine ausführliche Beschreibung ist in Anhang A.2 gegeben. Nachfolgend werden einige MPI-spezifische Aspekte hiervon herausgestellt.

5.3 MPI-Spezifika der Kommunikation in der NuT-Architektur

Der MPI-Standard wird auf zwei verschiedene Arten genutzt. Zum einen dient er der Beschleunigung der numerischen Rechnungen, zum anderen der Anbindung der NuT-Worker-Prozesse an ATHLET. Dies geht einher mit einem bestimmten Communicator-Modell, s. Abb. 5.1.

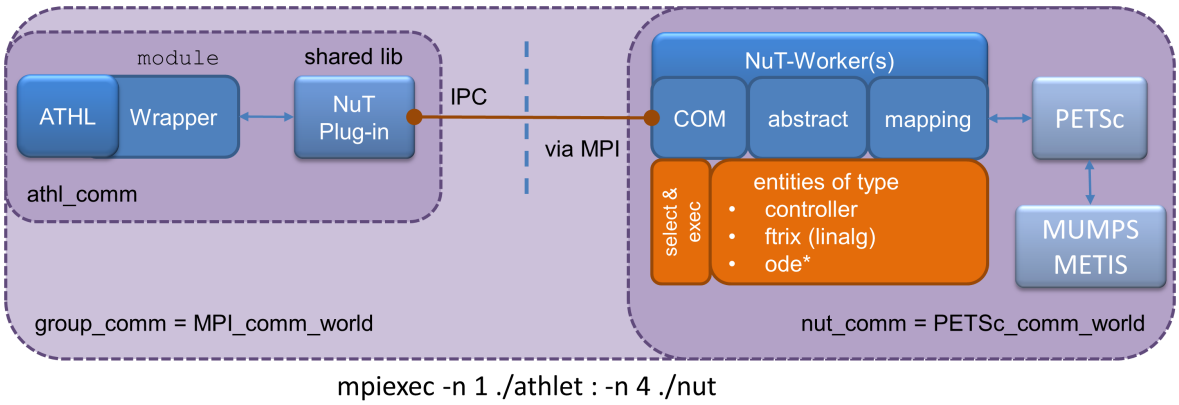


Abb. 5.1 Communicator-Modell zur NuT-Architektur. Entsprechend der Aufrufzeile wird genau ein ATHLET-Prozess gestartet, weswegen *group_comm* dem globalen *MPI_comm_world* entspricht. Jede Gruppe besitzt einen *nut_comm* der stets dem *PETSc_comm_world* für diese Gruppe entspricht.

5.3.1 Intraprozess-Kommunikation

Die für die numerischen Berechnungen und Speicherung der Jacobimatrix nötige MPI-Kommunikation zwischen den NuT-Worker-Prozessen wird vollständig von der PETSc-Bibliothek abgedeckt und gegenüber dem Nutzer weitestgehend transparent gehalten.

Mögliche zukünftige Funktionalitäten der NuT-Architektur, die keine Abdeckung durch PETSc erführen, können dennoch ohne Probleme auf Basis des Communicators *PETSc_comm_world* implementiert werden.

5.3.2 Interprozess-Kommunikation

Damit unterschiedliche Codes mit möglichst wenig Aufwand mit dem Numerical Toolkit gekoppelt werden können, ist es im Allgemeinen sinnvoll, dass die Schnittstelle hierzu

möglichst einfach ist. Aus dem Grund wird die Aufteilung der Daten für die Parallelität ausschließlich auf Seiten der NuT-Worker-Prozesse durchgeführt. Somit legt die Schnittstelle zur Steuerung der NuT-Worker-Prozesse fest, dass jeder dieser Prozesse exakt die gleichen Daten kommuniziert bekommt und selber entscheidet, welche Daten er verwendet und welche verworfen werden. Daher findet die komplette Kommunikation von ATHLET zum Numerical Toolkit per MPI-Broadcast statt – der ATHLET-Prozess sendet alle Eingabedaten an alle NuT-Worker-Prozesse zum Communicator *nut_comm*, s. auch Abb. 5.1.

Für die Rückgabe der verteilten Daten der NuT-Worker-Prozesse an ATHLET wäre es naheliegend, auf die Funktion MPI-Gather zurückzugreifen. Damit würde jeder NuT-Worker-Prozess seinen Teil der Rückgabedaten an den ATHLET-Prozess schicken, der sie sammelt und vervollständigt. Analog zur Kommunikation zwischen ATHLET und dem Numerical Toolkit, müsste ATHLET dazu allerdings wissen, nach welchem Schema die Daten zwischen den NuT-Worker-Prozessen verteilt sind. Dies ist nicht im Sinne der Architektur-Paradigmen in Abschnitt 2.3.1. Darum werden im Numerical Toolkit die verteilten Daten immer vom ersten Prozess innerhalb des Communicators *nut_comm* mithilfe der PETSc-Bibliothek per MPI-Gather gesammelt. Nachdem die Daten dort vollständig vorliegen, werden sie per Punkt-zu-Punkt-Kommunikation mittels MPI-Send und MPI-Receive an ATHELT übermittelt. Dieser Ablauf ist ausführlich in /JAC 17a, §4.3/ beschrieben.

6 Validierung der Modifikationen

6.1 Auswahl von Simulationsläufen

Um die in Kapitel 2 und 5 beschriebene NuT-Architektur auszutesten, war ursprünglich die Simulation eines Versuches an der PKL-Anlage oder einer vergleichbaren Integralanlage geplant. Zusätzlich sollte im ATHLET/CD-Kontext ein geeignetes Experiment an der QUENCH-Versuchsanlage gerechnet werden. Aufgrund von parallelen Entwicklungsarbeiten in der GRS hat sich das Testfeld leicht verschoben, ohne jedoch an Aussagekraft zu verlieren.

Es sei an dieser Stelle noch einmal ausdrücklich darauf hingewiesen, dass keine umfassende Validierung durchgeführt wird. Hauptsächlich geht es darum zu beurteilen, ob die entwickelten Modifikationen grundsätzlich funktionieren. In diesem Sinne ist nachfolgend der Begriff *Validierung* zu verstehen. Zusätzlich sind Performancemessungen an verschiedenen Modellen durchgeführt worden.

Auswahl – ATHLET

Im Juli des Jahres 2015 wurde in der GRS ein Jenkins-System, /KAW 17/, etabliert. Der Zweck eines solchen Systemes ist es, ein effektives Werkzeug zur kontinuierlichen Integration zur Verfügung zu haben. Eine zu prüfende Revision von ATHLET wird automatisch kompiliert und anschließend eingesetzt, um Testrechnungen mit ausgewählten Referenzdatensätzen durchzuführen. Jeder Referenzdatensatz beinhaltet zum einen Messdaten, und zum anderen einen Satz ATHLET-historischer Rechenergebnisse der relevanten Kenngrößen. Mittels der Gesamtheit dieser Ergebnisse wird ein Gültigkeitsbereich für die Validierung definiert. Verlassen die aktuellen Rechenresultate einen 2 %-Schlauch um besagten Gültigkeitsbereich wird der Test als fehlgeschlagen gewertet und hervorgehoben. Diese Validierungsmethode ist primär dafür geeignet, signifikante Abweichungen der Ergebnisse zwischen verschiedenen Revisionen von ATHLET aufzuzeigen. Ob das Verlassen des vorgegebenen Bereichs von qualitativer Bedeutung ist, muss von Fall zu Fall entschieden werden.

Das Jenkins-System eignet sich aufgrund der Fülle an vorhandenen Vergleichsdaten und der graphischen Aufbereitung der Ergebnisse hervorragend, um die NuT-Architektur im ATHLET-Kontext einer Überprüfung zu unterziehen. Statt der Betrachtung eines Versuches zu der PKL-Anlage wurden die Datensätze ROSA, ISB2 und LSTF aus dem Jenkins-Fundus herangezogen. Diese beziehen sich auf Versuchsanlagen zur Abbildung verschiedener Reaktortypen.

- ROSA – großes Leck in einem SWR
- ISB2 – mittleres Leck in einem DWR im WWER-Design
- LSTF – kleines Leck in einem DWR

Somit wird ein gewisses Spektrum an Störfällen abgedeckt. Nähere Beschreibungen hierzu finden sich im Validierunsgband zu ATHLET, s. /LER 16b/.

Hinsichtlich der Performancemessungen wurde auf die bereits in Tab. 2.5 genannten Beispiele PWR-3D-Sym und K3-NK-54H41N zurückgegriffen, welche hier unter den Bezeichnungen PWR-3D bzw. K3-2 diskutiert werden. Um auch ein sehr hochdimensionales Beispiel zur Verfügung zu haben, wird zusätzlich aus der K3-Reihe das Beispiel K3-19 betrachtet. Weitere Erläuterungen zu der getätigten Auswahl finden sich in Abschnitt 6.2.

Auswahl – ATHLET-CD

Die Kopplung von ATHLET und ATHLET-CD wird aktuell über das gleiche Plug-In-Konzept realisiert, welches auch für die in diesem Projekt aufgebaute NuT-Architektur genutzt wird, s. hierzu Abschnitt 2.3. Diese Art der Kopplung war zu Beginn des Projektes noch nicht vorhanden. Aufgrund des Plug-In-Konzeptes wird ATHLET als ausführendes Programm aufgerufen; es muss lediglich das ATHLET-CD-Plug-In sowie ein passender Datensatz zur Verfügung stehen. Damit ändert sich im Rahmen der NuT-Architektur nichts an der grundsätzlichen Aufrufweise, wie sie in Unterabschnitt 2.3.1.2 beschrieben ist. Es wird der *gleiche* Differentialgleichungs-Code ausgeführt, lediglich die rechte Seite in (1.1) ändert sich. Dies hat aber keinen Einfluss auf die etablierte NuT-Architektur, da f-Auswertung stets im ATHLET/CD-Kontext verbleiben, wie es in Unterabschnitt 2.3.1.5 diskutiert ist.

Dennoch wird der Anforderung, die NuT-Architektur im Rahmen von ATHLET/CD auszutesten, genüge getan, indem ein Station-Blackout-Störfall in einem generischen DWR gerechnet wird. Der Datensatz bedarf der aktuellen Entwicklerversion von ATHLET-CD, da diese eine verbesserte Nodalisierung aufweist, welche asymmetrische Effekte wie z. B. unterschiedliche Nachzerfallsleistungen, lokale Oxidationsspitzen und Blockadenbildungen berücksichtigt. Die Diskussion des Beispiels ist dem Abschnitt 6.2 zugeordnet, da durch die neue Nodalisierung benutzer- und inputabhängig 10-20 mal so viele Elemente im Kern entstehen, was die Rechnung entsprechend verlangsamt. Es geht also primär um die Fragestellung, wie viel Performancegewinn durch den Einsatz der NuT-Architektur zu erwarten sei.

Bemerkung 6.1. Man beachte, dass für die diskutierten Simulationsläufe in diesem Kapitel die NuT-Architektur noch nicht von einer erweiterten Differentialgleichungsnumerik Gebrauch macht. Zur Erläuterungen siehe Abschnitt 3.1. Es stehen aber effiziente –

und optional verteilt nutzbare – Datenstrukturen und Löser zu den involvierten linearen Gleichungssystemen zur vollen Verfügung. Diese werden hier in den drei in Unterabschnitt 2.3.1.2 vorgestellten Konfigurationen `lu`, `mumps` und `hybrid` im Tandem mit dem ATHLET-Löser FEBE zum Einsatz gebracht.

Bemerkung 6.2. Sämtliche hier vorgestellten Ergebnisdaten wurden durch die Linux-Version der `NuT`-Architektur berechnet. Stichprobenartig durchgeführte Testläufe unter Windows erzielten exakt die gleichen Ergebnisse wie unter Linux. Es ist darauf zu achten, dass auf beiden Plattformen identische Compilereinstellungen für die Erstellung von ATHLET und der PETSc-Bibliothek verwendet werden.

6.2 Performance

Zur Bewertung der Performance wird die kumulierte Rechenzeit der wichtigsten/zeitaufwändigsten Komponenten gemessen. Dazu gehört das Lösen der linearen Gleichungssysteme (GLS) und die Berechnung der f-Auswertungen für das Zeitschrittverfahren bzw. zur Erstellung der Jacobimatrizen (Jacobi/f-Auswertung). Der Zeitaufwand für alle restlichen Komponenten kann aus der Gesamtzeit berechnet werden, nachdem die Zeit für GLS und f-Auswertungen abgezogen wurde. Zur Implementierung wird die von PETSc angebotene Profiling-Komponente verwendet. Damit können die Zeitdaten einzelner Code-Bereiche wie z.B. dem GLS-Löser in Code 2.8 einfach aufgezeichnet werden. Die Zeitmessung pro Code-Abschnitt wird jeweils davor mit `PetscLogStagePush` gestartet und anschließend mit `PetscLogStagePop` wieder beendet. Neben den reinen Zeitdaten werden auch weitere Information wie Anzahl der Aufrufe, Speicherverbrauch und Kommunikationsdaten pro PETSc-Befehl innerhalb des ausgewählten Bereichs gespeichert, die eine gute Grundlage zur ausführlichen Analyse bieten.
Im Folgenden sei die Nutzung von ATHLET im Rahmen der erarbeiteten `NuT`-Architektur abkürzend mit *ATHLET-NuT* bezeichnet. Hierbei kommen entsprechend Bemerkung 6.1 die alternativen Lösungsverfahren für die linearen Gleichungssysteme sowie effiziente Datenstrukturen zum Einsatz. Man beachte, dass je nach verwandter Methode oder Parametereinstellung leicht unterschiedliche Ergebnisse vorliegen (können). Der Zeitbedarf zur Auswertung von f ist also nicht gänzlich unabhängig von der Konfiguration.

Das Ziel des Einsatzes weiterer Gleichungssystemlösers ist es, die Gesamtlaufzeit der Simulation so stark wie möglich zu reduzieren, ohne dabei an Stabilität zu verlieren. Ob diese Verbesserung signifikant ist, hängt davon ab, wie sehr der Simulationslauf von der linearen Algebra dominiert wird. Letztendlich ist es daher stets sinnvoll, in Vergleichen den

Gesamtzeitverbrauch sowie die benötigte Zeit für die lineare Algebra zu berücksichtigen.

Bemerkung 6.3. ATHLET berechnet vor Beginn der Transientenphase die Färbung der Jacobimatrix und führt den Algorithmus zur Fill-In-Reduktion aus. In der Regel ist der Rechenaufwand hierfür in absoluten Zahlen nicht signifikant, weswegen in den Balkendiagrammen zur Performancemessung kein explizites Herausstellen hiervon vorgenommen wird. Bei den Modellen zur `K-3`-Reihe ist der Zeitaufwand jedoch nicht trivial, worauf im Begleittext zu den Performancediagrammen eingegangen wird. Eine ausführliche Auflistung der Messungen zu Färbung und Fill-In-Reduzierung findet sich im Anhang, /JAC 17a/.

Das `i2mftrx`-Flag

Für alle Modelle kann mithilfe des Parameters `i2mftrx` eine erweiterte Nodalisierung aktiviert werden. Somit werden Querverbindungen auch in der Jacobimatrix berücksichtigt und nicht nur als Quellterme in f betrachtet. Folglich erhöht sich die Besetztheitsdichte. Im Standard-ATHLET führt diese Option in vielen Fällen zu einem unverhältnismäßig großen Mehraufwand zum Lösen der linearen Gleichungssysteme. Dies lässt sich gut anhand der in Abb. 6.2 und Abb. 6.3 dargestellten Laufzeiten des Standard-ATHLETs aufzeigen. Für das dort betrachtete Beispiel werden durch die Aktivierung des `i2mftrx`-Flags lediglich 6 % mehr Einträge in der Jacobimatrix erzeugt, vgl. Abb. 6.1. Dies hat jedoch eine Steigerung der Rechenzeit des Gleichungssystemlösers um den Faktor 14 zur Folge!

Im weiteren Text wird dem jeweiligen Modellnamen ein + angefügt, um darauf hinzuweisen, dass das `i2mftrx`-Flag aktiv ist, e. g. `PWR-3D+`.

Eckdaten zu den Modellen

Nachfolgend werden zu den verschiedenen Beispielen auch die Besetztheitsstrukturen der Jacobimatrizen angegeben. Neben der Verteilung der Einträge sind wesentliche Kenngrößen für die Komplexität des Problems zum einen die Dimension `dim` sowie die Anzahl an potentiellen Nicht-Null-Elementen `nnz`. Man beachte außerdem, dass im Rahmen der ATHLET-Modellierung die Struktur der Jacobimatrix immer symmetrisch ist, s. die Herleitung zu (2.4d) in Kapitel 2. Dies gilt jedoch *nicht* für die numerischen Einträge!

6.2.1 `PWR-3D`

Wie bereits in Tab. 2.5 erläutert ist der `PWR-3D`-Datensatz Teil der ATHLET-Installation. Simuliert wird ein 2F-Bruch in der Hauptkühlmittelleitung in einem DWR. Das Problem entspricht von der Größenordnung und Komplexität den üblichen Modellen, die mit

ATHLET bearbeitet werden. Die Besetztheitsstruktur der Jacobimatrix ist in Abb. 6.1 zu sehen.

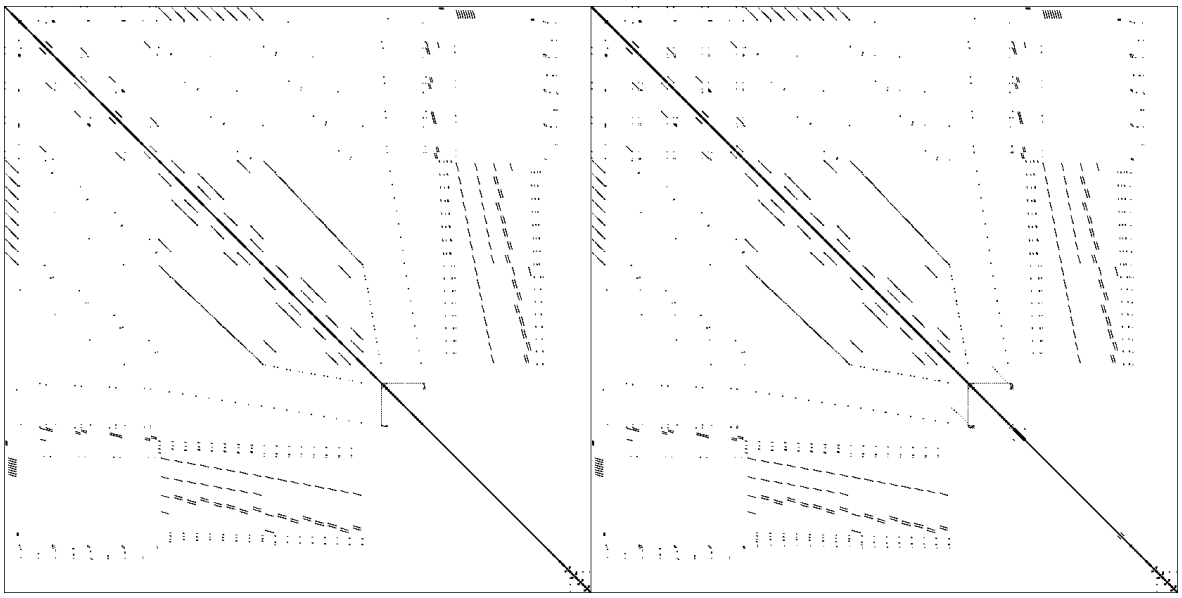


Abb. 6.1 PWR-3D: nnz $= 143.469$, dim $= 6.009$ (links)
PWR-3D+: nnz $= 151.961$, dim $= 6.009$ (rechts)

Abbildung 6.2 zeigt die Zeitanteile des GLS-Lösers (34 %) und für die f-Auswertungen (64 %) im Standard-ATHLET. Durch den Einsatz von MUMPS wird der GLS-Zeitbedarf halbiert, was den Anteil an der Gesamtzeitlaufzeit auf 23 % reduziert. Insgesamt ergibt sich eine Verbesserung der Laufzeit um 14 %.
Wird das i2mftrx-Flag eingeschaltet, zeigen sich kaum erkennbare Veränderungen an der Struktur der Jacobimatrix. Wie im obigen Paragraph *Das* i2mftrx*-Flag* bereits erwähnt führt dies jedoch zu einem erheblichen Anstieg an Rechenzeit. Statt drei Stunden sind es nunmehr über 16 Stunden, die benötigt werden.

Im Vergleich dazu reagiert die mumps-Konfiguration erwartungsgemäß auf den leichten Komplexitätsanstieg. Die neue Gesamtrechenzeit beträgt immer noch weniger als drei Stunden.

6.2.2 ATHLET-CD

Zur inhaltlichen Beschreibung des betrachteten Testfalles s. Abschnitt 6.1. Die verfeinerte Art der Nodalisierung schlägt sich direkt in der verdichteten Besetztheitsstruktur nieder, s. Abb. 6.4. Im Vergleich zu PWR-3D sind bei leicht reduzierter Dimension fast zehnmal so viel Einträge in der Jacobimatrix vorhanden.

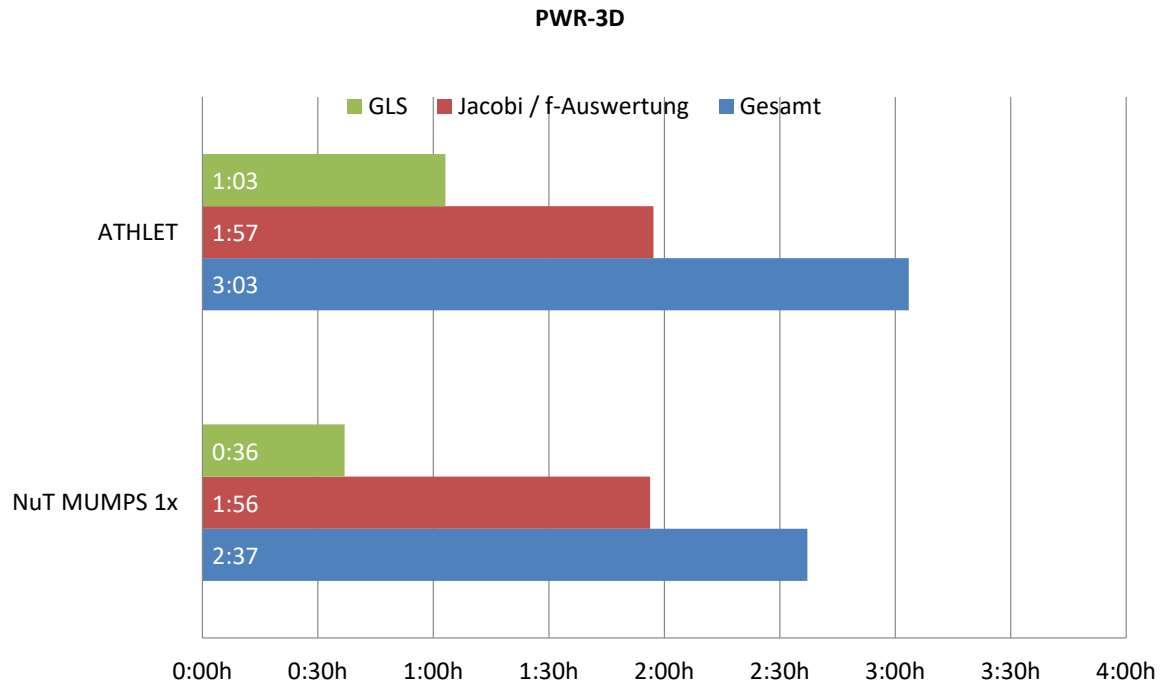


Abb. 6.2 Performancevergleich für PWR-3D, die NuT-Konfiguration läuft mit einem NuT-Worker-Prozess

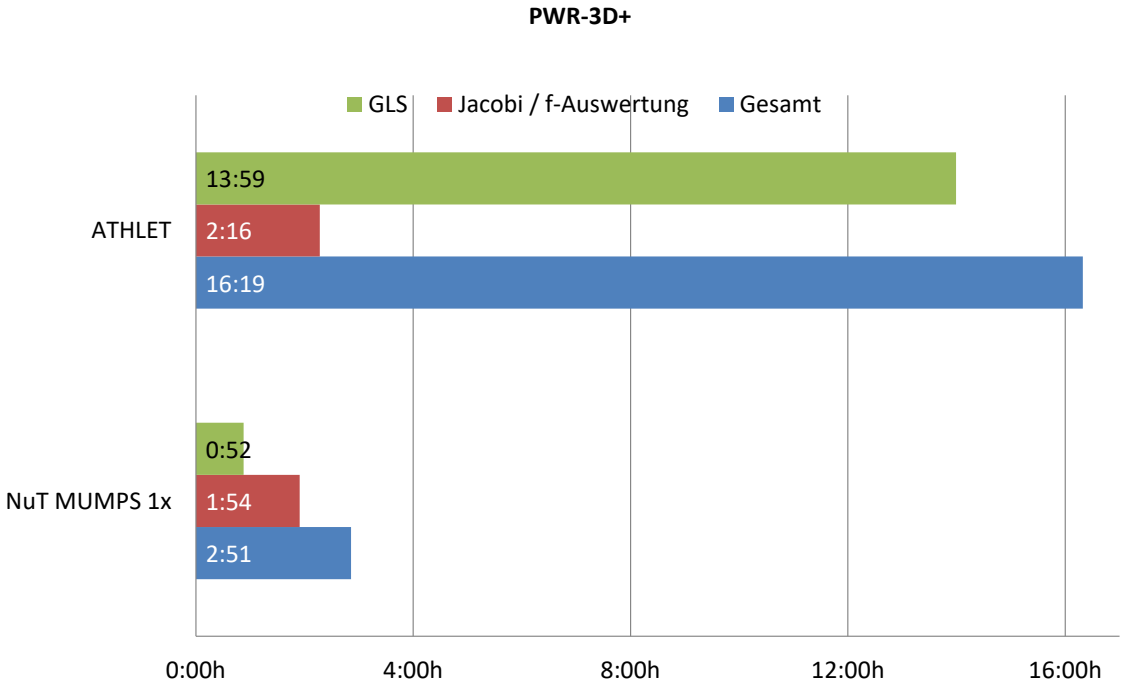


Abb. 6.3 Performancevergleich für PWR-3D+, die NuT-Konfiguration läuft mit einem NuT-Worker-Prozess – zusätzliche Berücksichtigung von Querverbindungen in der Jacobimatrix

Abbildung 6.5 zeigt eine deutliche Verbesserung des GLS-Zeitbedarfs, wenn der LU-Löser von PETSc Anwendung findet. Der Einsatz der HYBRID-Konfiguration in der Kombination

von GMRES und MUMPS als Präkonditionierer bringt keinen signifikanten Unterschied zu ATHLET mit sich. Ähnlich verhält es sich bei einer reinen Nutzung von MUMPS. Dies mag zum einen an der Problemgröße liegen – MUMPS ist nicht für solche kleinen, sondern für höherdimensionale Probleme ausgelegt. Zum anderen trägt sicherlich auch die verhältnismäßig hohe Anzahl an Nicht-Null-Elementen in Relation zur Dimension hierzu bei. Eine serielle Standard-LU-Zerlegung stört sich wenig an solchen Rahmenbedingungen. Da auch keine weitere Bibliothek (i. e. MUMPS) angesprochen wird, entfallen zusätzlich ein gewisser Kommunikations-Overhead und der Aufbau weiterer Speicherstrukturen. Dieses Beispiel zeigt gut, wie unterschiedlich die einzelnen Löserstrukturen agieren und wie sinnvoll es ist, verschiedene hiervon zur Verfügung zu stellen, um die jeweils passende Konfiguration ermitteln zu können.

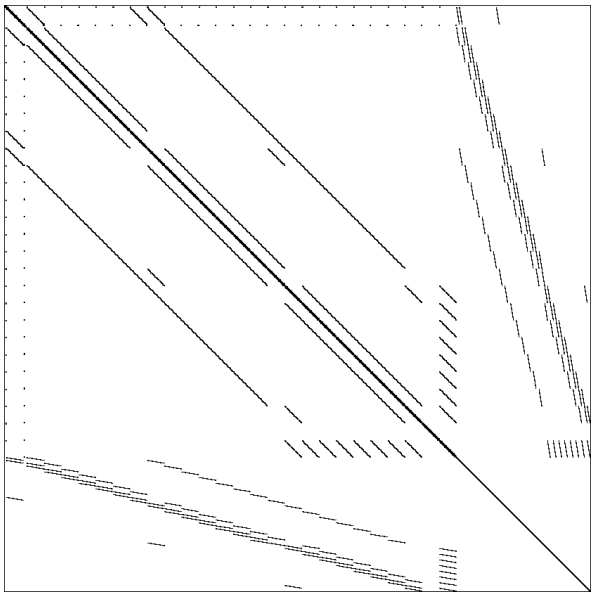


Abb. 6.4 `ATHLET-CD`: $nnz = 1.389.735$, $dim = 4.121$

6.2.3 `K3-n`

Die Modelle aus der `K3`-Reihe basieren auf den Daten aus /TER 08/ mit einigen Anpassungen im Detail. Simuliert wird das Abschalten einer der vier Hauptkühlmittelpumpen in einem VVER-1000 Kern. Die Simulation führt zunächst eine Start-up-Phase durch, um anschließend für 20-$30\,s$ Nominalleistung im Reaktor zu halten. Die Pumpenabschaltung erfolgt nach $500\,s$ Simulationszeit. Besonders an diesen Modellen ist die stabweise Nodalisierung eines heißen Kanals sowie einer gewissen Anzahl k an Nachbarelementen hierzu. Die Variable n ergibt sich somit zu $n = k + 1$.

Für diese Arbeit werden die Modelle `K3-2`, `K3-2+` und `K3-19` betrachtet. Dadurch lässt sich sehr schnell eine Skalierung der Dimensionierung der Gleichungssysteme vorneh-

men, was zum Performancevergleich verschiedener Konfigurationen besonders von Vorteil ist. Die Aktivierung des `i2mftrx`-Flags hat einen erheblichen Einfluss auf die Struktur und die Anzahl an Nicht-Null-Elementen der Jacobimatrix, s. Abb. 6.6.

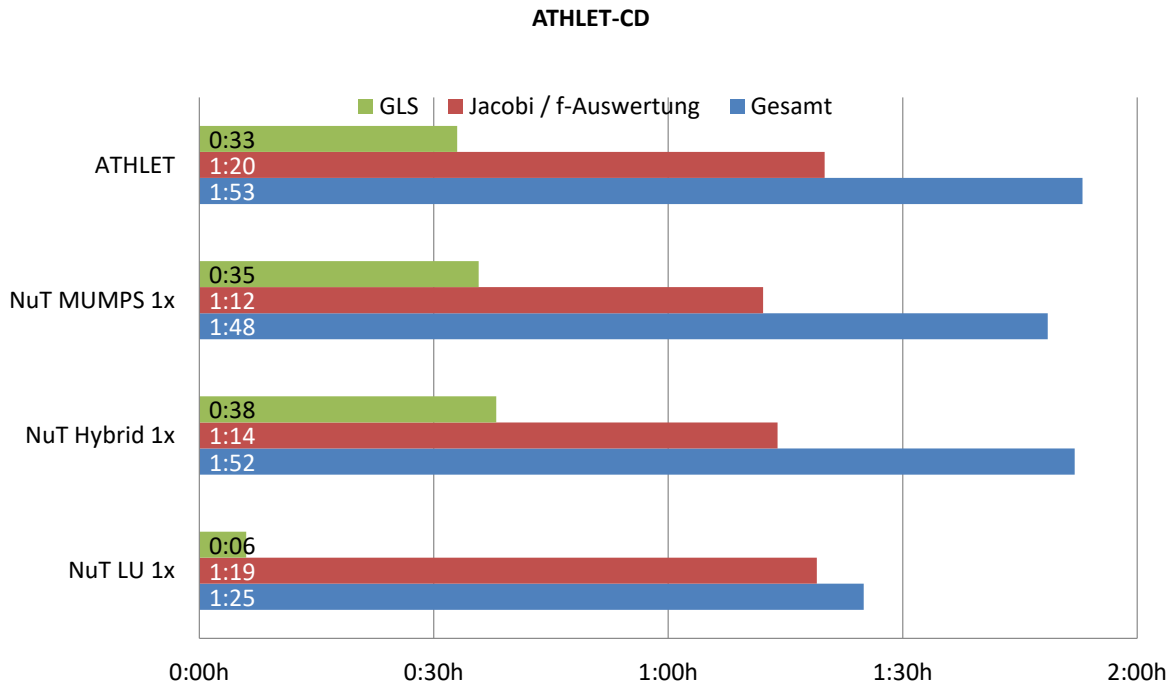


Abb. 6.5 Performancevergleich für `ATHLET-CD`, die `NuT`-Konfigurationen laufen mit einem `NuT`-Worker-Prozess

`K3-2` und `K3-2+`

Die Standard-ATHLET-Konfiguration benötigt für das kleinste Problem `K3-2` erheblich mehr Zeit als für herkömmliche Modelle. Der einmalig zu Beginn durchgeführte Algorithmus zur Fill-In-Reduktion läuft über zehn Stunden, und die Färbung der Jacobimatrix ist mit ca. 26 Minuten Zeitbedarf auch weit über dem Durchschnitt. ATHLET-NuT benötigt für diese Aufgaben *insgesamt* nur wenige Sekunden, obwohl sie deutlich öfter in Abhängigkeit von Strukturänderungen durchgeführt werden!

Die Simulation wurde mit ATHLET nach 45 Stunden abgebrochen. Bis dahin wurde eine Simulationszeit von $t = 0.1\,s$ erreicht. Das in Abb. 6.7 gezeigte Balkendiagramm ergibt sich durch Extrapolation. Hochgerechnet ist somit eine geschätzte Simulationszeit von drei Monaten zu erwarten. Der Zeitanteil von GLS beträgt über 99 % und deutet darauf hin, dass ATHLET Gleichungssysteme dieser Größenordnung nicht mehr effizient lösen kann.

Mit ATHLET-NuT sinkt die Startzeit von über zehn Stunden auf weniger als drei Minuten. Unter Einsatz der `mumps`-Konfiguration mit einem `NuT`-Worker-Prozess fällt die gesamte

Simulationszeit auf knapp unter 24 Stunden und mit der auf vier NuT-Prozesse erweiterten Konfiguration werden nur noch 18 Stunden für die komplette Rechnung benötigt.

Für K3-2+ ist der Unterschied zwischen der ATHLET-Standard-Konfiguration und den neuen Strukturen nochmals deutlich größer. Der Simulationslauf wurde mit ATHLET nach acht Tagen abgebrochen, da die Berechnung der Fill-In-Reduktion nach dieser Zeit noch nicht abgeschlossen wurde. Im ATHLET-NuT-Kontext ergibt sich eine verständliche Erhöhung der Laufzeit, diese fällt aber erwartungsgemäß gering aus. Man beachte, dass auch in diesem Fall die kumulierte Zeit für alle benötigten Fill-In-Reduzierungen und Färbungen während der gesamten Simulation nicht signifikant in die Gesamtlaufzeit eingeht, da diese weniger als eine Minute beträgt.

Beide Modelle profitieren von einer Erhöhung der Anzahl an NuT-Worker-Prozessen, wobei der Gewinn fürs Modell K3-2+ etwas besser ausfällt als für K3-2, vgl. Abb. 6.7 mit Abb. 6.8. Man beachte, dass es durchaus normal ist, dass die naive Erwartung der Viertelung des Zeitbedarfs für GLS nicht erfüllt ist, nur weil statt eines Prozesses vier zur Verfügung stehen. Verteiltes Rechnen geht immer mit einem gewissen zusätzlichen Kommunikationsbedarf einher, und nicht alle Komponenten der Gleichungssystemlöser sind parallelisierbar.

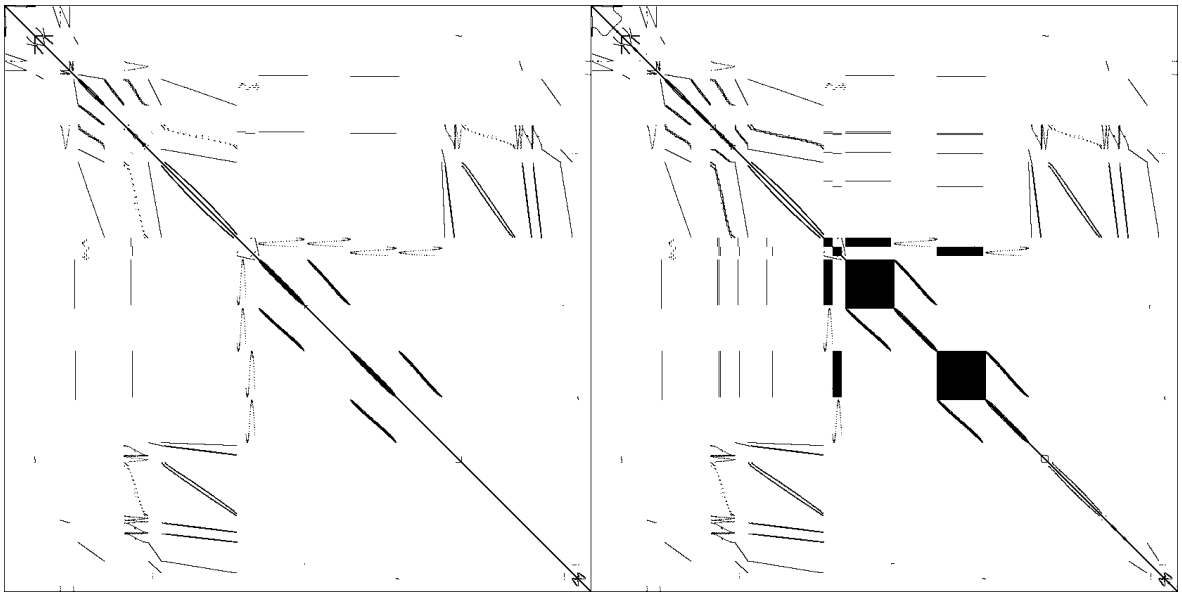


Abb. 6.6 K3-2: $nnz = 2.852.179$, $dim = 128.673$ (links)
K3-2+: $nnz = 5.240.891$, $dim = 128.673$ (rechts)

K3-19

Für K3-19 wird aufgrund der enormen Größe im Gegensatz zu den anderen Testfällen ATHLET mit zehn OpenMP-Threads gestartet, um die Rechenzeit der f-Auswertungen

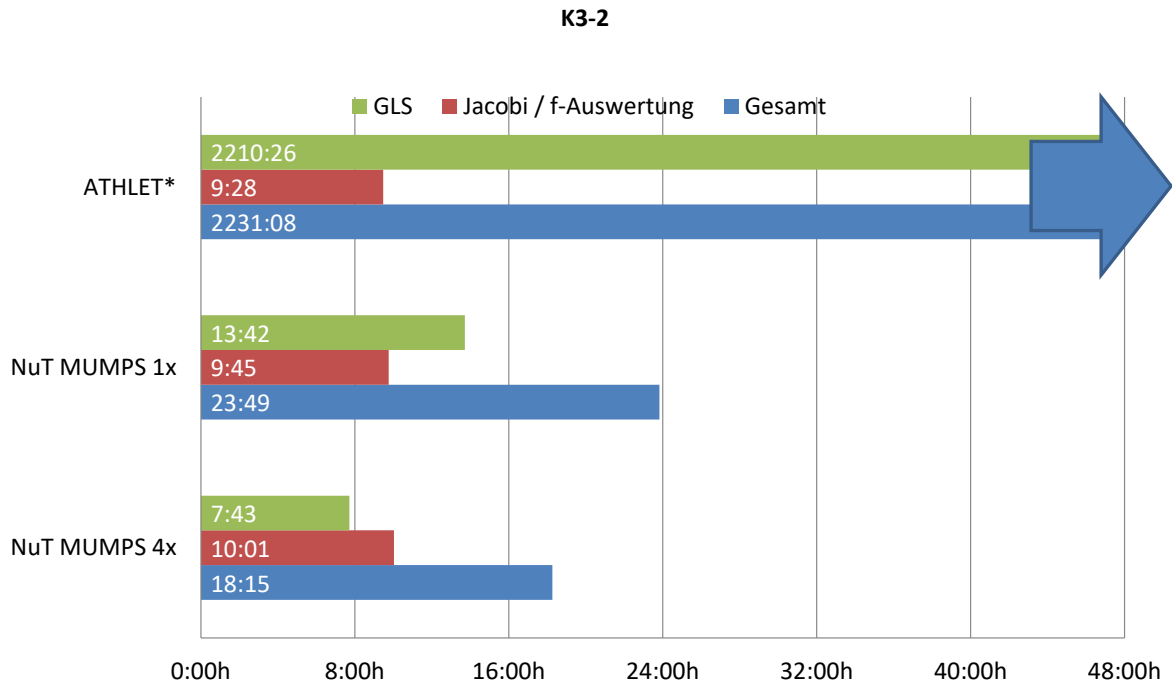


Abb. 6.7 Performancevergleich für K3−2, die NuT-Konfigurationenen laufen mit einem bzw. vier NuT-Worker-Prozess(en)

nicht zu groß werden zu lassen. Die ATHLET-interne lineare Algebra ist nicht mehr sinnvoll anwendbar. Insofern werden hier nur Ergebnisse der NuT-Architektur betrachtet.

In der mumps-Konfiguration mit einem NuT-Worker-Prozess zeigen sich bereits brauchbare Ergebnisse. Die Gesamtrechenzeit beträgt 213 Stunden. Eine Ausführung auf mehreren Prozessoren wird durch die Standard-Wahl des Threshold-Parameters u fürs partial pivoting vereitelt. Denn diese lässt den Fill-In pro Prozess signifikant ansteigen. Der Gesamtspeicherverbrauch im Vergleich zur Ein-Prozess-Version erhöht sich um mehrere Größenordnungen, bis MUMPS die Rechnung schließlich abbricht. Hier zeigt sich ein Nachteil der verteilten Speicherung der Jacobimatrix, denn Fill-In-Minimierung funktioniert am besten, wenn sämtliche Besetzungs-Informationen zur Matrix zur Hand sind. Jeder Prozess sieht aber nur einen Teil der Matrix. Mit der Wahl $u = 10^{-4}$ verschwindet die Speicherproblematik, jedoch geht dies zu Lasten der Genauigkeit. Dies motivierte den hybrid-Ansatz, in dem MUMPS mit $u = 10^{-4}$ als Präkonditionierer in Kombination mit der Krylov-Unterraum-Methode GMRES eingesetzt wird. Wie in Abb. 6.10 dargestellt, braucht die hybrid-Konfiguration mit einem NuT-Worker-Prozess im Vergleich zur mumps-Konfiguration gleicher Prozesswahl 40 % weniger Rechenzeit für den GLS-Anteil und kann in seiner parallelen Variante mit vier NuT-Prozessen weitere 32 % zum Lösen der linearen Gleichungssysteme einsparen. Dies führt zu einer Reduktion der Gesamtlaufzeit von knapp 9 Tagen zu ca. 6 Tagen.

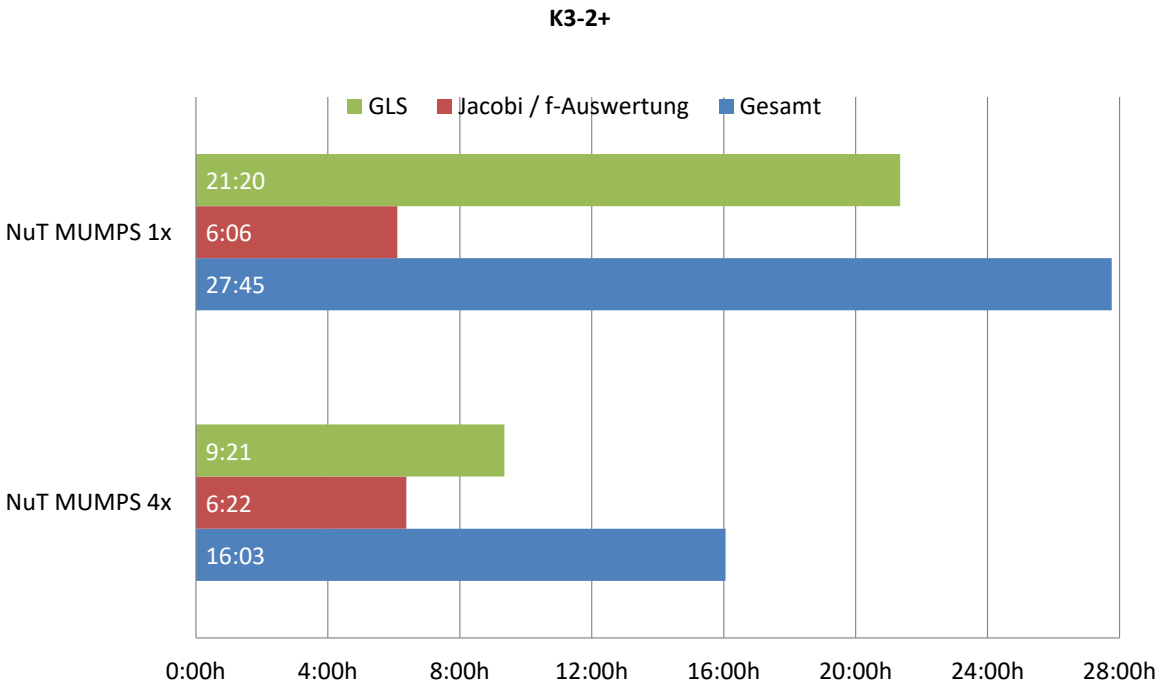


Abb. 6.8 Performancevergleich für K3−2+, die NuT-Konfigurationen laufen mit einem bzw. vier NuT-Worker-Prozess(en) – zusätzliche Berücksichtigung von Querverbindungen in der Jacobimatrix

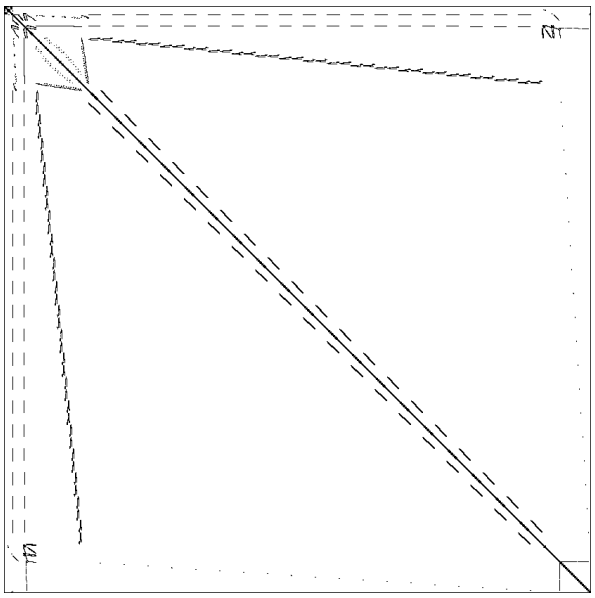


Abb. 6.9 K3−19: $nnz = 7.204.723$, $dim = 1.155.955$

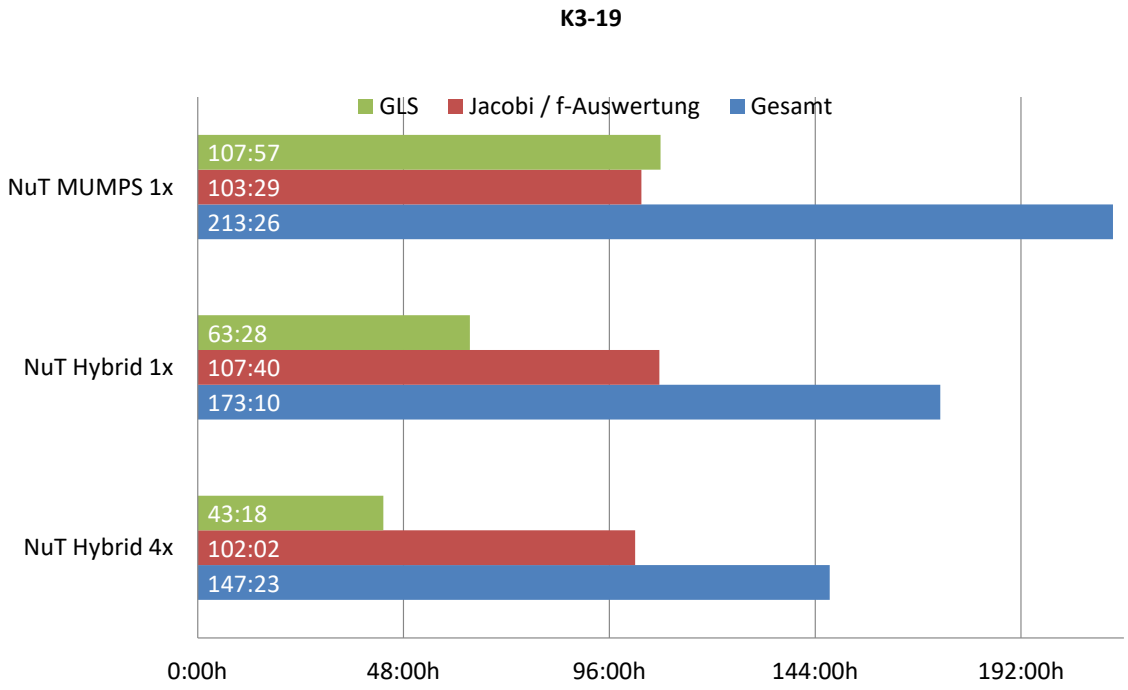


Abb. 6.10 Performancevergleich für `K3-19`, die `NuT`-Konfigurationen laufen mit einem bzw. vier `NuT`-Worker-Prozess(en)

6.3 Validierung

Die Basis für die Validierung bildet der Entwicklungsstand von ATHLET ohne und mit `NuT`-Erweiterung am 24. Juli 2017. ATHLET-NuT verwendet in dem Fall die `mumps`-Konfiguration mit einem `NuT`-Worker-Prozess, wobei zusätzliche Maßnahmen zur numerischen Stabilisierung ausgeführt werden. Diese bestehen in der Anwendung von Skalierungsmatrizen und ggf. einer zusätzlichen Permutationsmatrix vor dem Permutationsschritt zur Fill-In-Minimierung, s. auch die Arbeitsschrittfolge zum direkten Lösen auf Seite 17. Motivation hierzu ist, die Diagonalelemente von den restlichen Elementen der Matrix betragsmäßig abzuheben, s. /AME 17, §3.2/ für weitere Details hierzu. Diese Option besteht ausschließlich für die Ein-Prozess-Variante von MUMPS. Standardmäßig werden Skalierungen im Zerlegungsschritt eingebracht.

Die Testergebnisse der drei Referenzdatensätze `ROSA`, `ISB2` und `LSTF` sind in Tab. 6.1 abgebildet. Je nach Modell befinden sich für den Lauf mit dem Standard-ATHLET 88 % bis 99 % der Testgrößen vollständig innerhalb des Toleranzbereichs. Bezüglich ATHLET-NuT ergibt sich 90 % bis 99 %. In den Fällen `ISB2` und `LSTF` besteht ATHLET-NuT die gleiche Anzahl an Tests und bei `ROSA` schneidet ATHLET-NuT drei Prozentpunkte besser ab.

Abbildung 6.11 zeigt im Vergleich den einzigen Test, den ATHLET-NuT für `ISB2` nicht

besteht. Der fehlgeschlagene Test mit dem Standard-ATHLET ist in Abb. 6.12 dargestellt. Weitere repräsentative Testgraphen zu den Modellen LSTF und ROSA sind in Abb. 6.13-6.15 gegeben. Sowohl für den Standard-ATHLET als auch im Rahmen von ATHLET-NuT können alle fehlgeschlagenen Tests in zwei Kategorien eingeteilt werden: Der Toleranzbereich wird für längere Zeit, dafür aber nur sehr knapp, überschritten, oder es erfolgen Ausreißer in Form von kurzen Spitzen verschiedenen Betrages. Die Position der Überschreitungen kann hierbei durchaus konfigurationsspezifisch sein. Generell treten aber Abweichungen für die ATHLET-NuT-Konfiguration seltener auf.

Tab. 6.1 Validierungsergebnisse im Überblick. Absolute Anzahl und relativer Anteil erfolgreich durchgeführter Tests mit ATHLET und ATHLET-NuT im Vergleich.

	ROSA		ISB2		LSTF	
Tests gesamt	253		71		144	
ATHLET	223	88 %	70	99 %	130	90 %
ATHLET-NuT	230	91 %	70	99 %	130	90 %

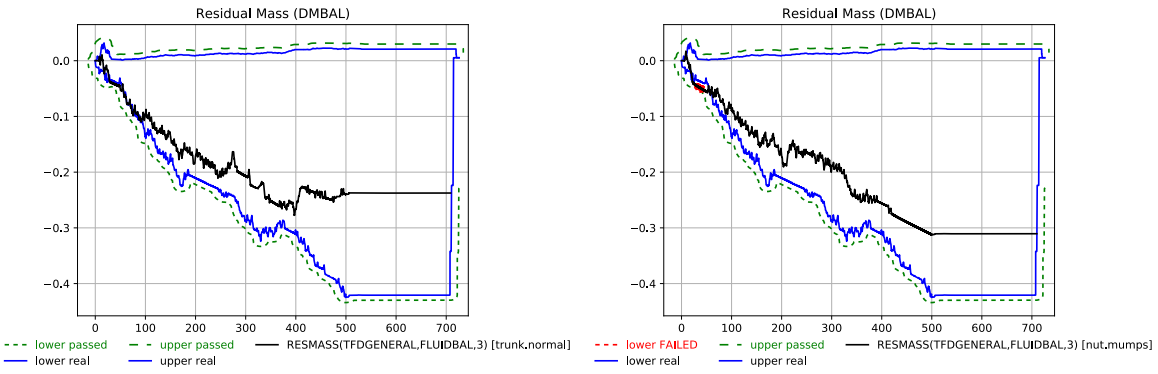


Abb. 6.11 ISB2: Massenresiduum. Obere/untere Schranke: blau, simulierte Größe: schwarz, 2 %-Toleranz: grün gestrichelt. ATHLET (links) zuerst leichte Überschreitung der unteren Schranke, aber noch innerhalb des 2 %-Bereichs. ATHLET-NuT (rechts) ähnlicher Verlauf wie ATHLET, nur knappe Überschreitung der 2 %-Toleranz.

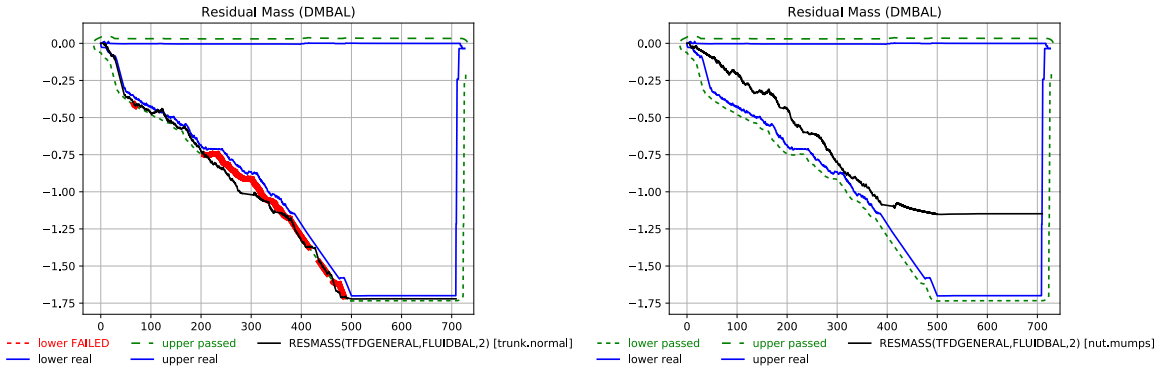


Abb. 6.12 ISB2: Massenresiduum. ATHLET (links) unterschreitet einen Großteil der Zeit knapp den 2 %-Bereich. ATHLET-NuT (rechts) vollständig im gültigen Bereich und betragsmäßig niedrigeres/besseres Endresiduum.

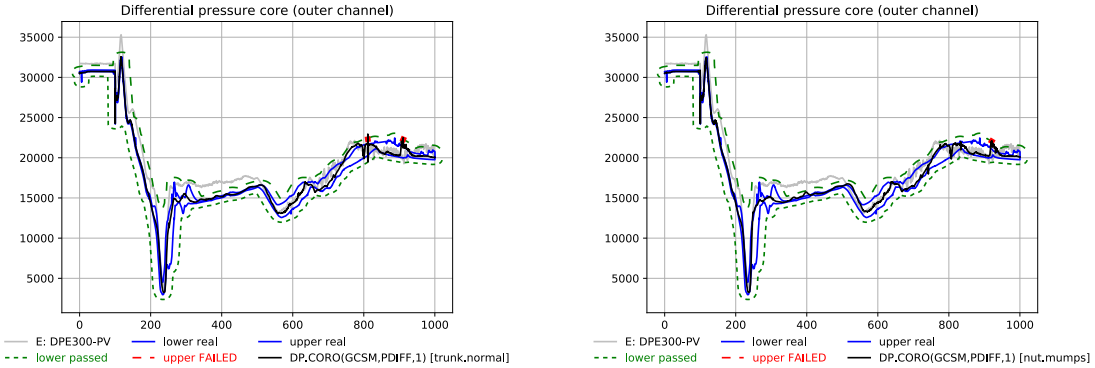


Abb. 6.13 LSTF: Druckdifferenz. ATHLET (links) zwei kurze Spitzen überschreiten den Toleranzbereich gegen Ende. ATHLET-NuT (rechts) gleiche Spitzen wie ATHLET, allerdings mit geringerem Ausschlag.

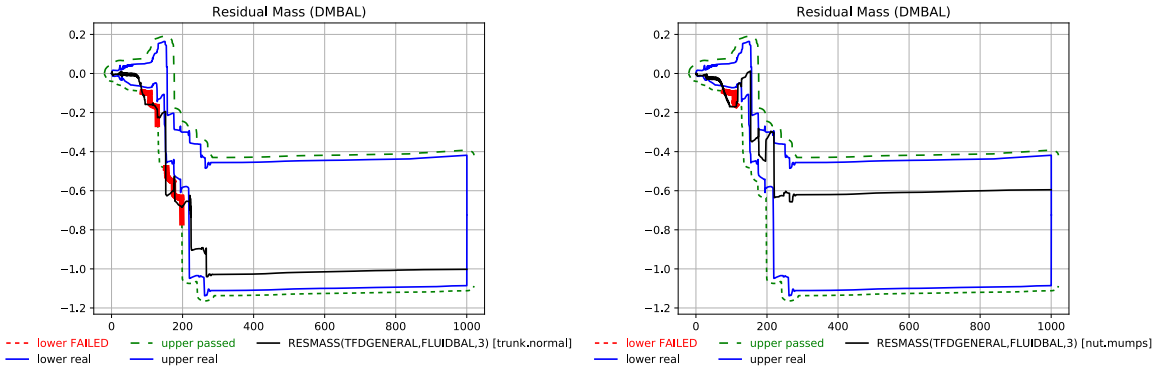


Abb. 6.14 LSTF: Massenresiduum. ATHLET (links) unterschreitet zu Beginn in zwei Abschnitten die unterere Schranke um etwas mehr als 2 %. ATHLET-NuT (rechts) vergleichsweise geringere Unterschreitung in nur einem Bereich. Betragsmäßig kleinerer/besserer stationärer Endstand.

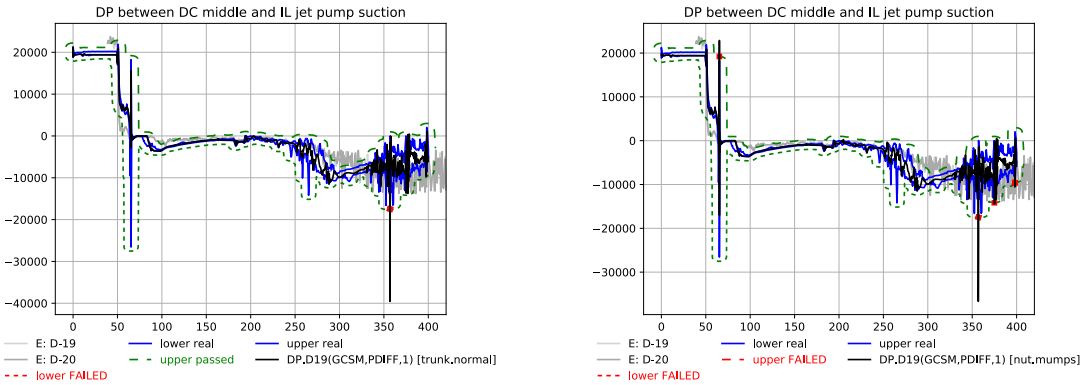


Abb. 6.15 ROSA: Fluidtemperatur. ATHLET (links) starke, aber kurze Unterschreitung der Toleranzgrenze. ATHLET-NuT (rechts): Mehrere Unterschreitungen durch Spitzen, dafür teilweise kleinerer Ausschlag.

7 Fazit und Ausblick

Das für dieses Projekt etablierte Numerical Toolkit fasst sämtliche ausgelagerten Funktionalitäten zur linearen Algebra und DGL-Numerik im ATHLET-Kontext zusammen und ist flexibel an den ATHLET-Code über das Konzept einer Inter-Process-Communication mittels MPI angebunden, vgl. Abb. 2.2. Damit der allgemeine ATHLET-Entwicklungsprozess weitestgehend ungestört hiervon bleibt und die notwendigen MPI-Direktiven gekapselt werden können, wird das Plug-In-Konzept zu ATHLET genutzt, welches durch die Hilfsbibliothek *libadt* möglich gemacht wird. Auf diesem Weg wird ebenso ATHLET-CD an ATHLET gebunden, womit auch in diesem erweiterten Kontext auf die Funktionsmächtigkeit des Numerical Toolkits zugegriffen werden kann.

Das Toolkit kapselt den Zugriff auf leistungsstarke externe Numerik-Bibliotheken wie PETSc und MUMPS. Durch die interne Aufteilung im Toolkit nach Kommunikation, abstrakter mathematischer Funktion und implementierungsnaher Abbildung wird die Verständlichkeit des Codes, die Wartung und ggf. Erweiterung des Funktionencanons grundsätzlich sehr vereinfacht. Ebenso wird das volle Potential an verteilter Rechnung und Speicherung von Daten, welches durch PETSc angeboten wird, ausgenutzt.

Die positiven Ergebnisse zu den Validierungsbeispielen in Kapitel 6 deuten darauf hin, dass durch das Einbringen der `NuT`-Architektur zur Berechnung der involvierten linearen Algebra generell keine qualitative Verschlechterung der Rechenergebnisse zu erwarten ist. Darüber hinaus zeigen die Rechnungen zu den Performancebeispielen, dass abhängig von der Problemgröße zum Teil signifikante Verbesserungen in der Laufzeit erzielt werden können. Dem Nutzer werden verschiedene Konfigurationen zum Lösen linearer Gleichungssysteme angeboten, welche bequem über die Kommandozeile selektiert werden können, s. Unterabschnitt 2.3.1.2. Auch Probleme großer Dimension sind nun dank der Option zum verteilten Rechnen effizient handhabbar.

Aufgrund der in Unterabschnitt 2.3.1.5 und Abschnitt 3.1 diskutierten Umstände sind bis zum Test der in Kapitel 3 entwickelten neuen Finite iteration Runge-Kutta-Methoden zum Lösen des Anfangswertproblemes (1.1) noch weitere Arbeiten notwendig, welche sich auf das Einbinden der ATHLET-spezifischen Kontrollstrukturen zur Zeitintegration beziehen. Jedoch zeigen die Ergebnisse in Abschnitt 3.5.2 zu allgemeinen steifen Systemen bereits, dass das $FiterRK$-Konzept ein wesentliches Potential zur Verbesserung der DGL-Numerik in sich trägt. Mit den angepassten Methoden aus Abschnitt 3.6 sollte dies auch für den speziellen Rahmen der ATHLET-Anforderungen gelten.

Im Sinne eines Ausblickes sollte genau hier als nächstes angesetzt werden: das stabile Ausführen einer $FiterRK$-Methode im ATHLET-Kontext sowie die anschließende Erweiterung der Schrittweiten- und Jacobimatrixsteuerung orientiert an Alg. 3.7.

Bereits von Anfang an wurde darauf Wert gelegt, dass die NuT-Architektur mit einem hohen Maß an Flexibilität ausgestattet ist und somit über das Konzept der Inter-Process-Communcation grundsätzlich auch mit anderen Programmen zusammenarbeiten kann. Es bietet sich somit an, das Toolkit derart anzupassen, dass es für die gesamte AC^2-Rechenkette zur Verfügung steht, sprich, auch zukünftig COCOSYS numerische Dienste anbietet. Somit stünde ein AC^2-einheitlicher Zugriff auf die involvierte Numerik zur Verfügung.

Literaturverzeichnis

/ABR 72/ M. Abramowitz und I. A. Stegun, Hrsg. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*. 10. Aufl. Bd. 55. Applied Mathematics Series. `http://people.math.sfu.ca/~cbm/aands/`; Zugriff 18. Mai, 2016. 1972 (siehe S. 93).

/AME 00/ P. R. Amestoy, I.S. Duff und J.-Y. L'Excellent. "Multifrontal parallel distributed symmetric and unsymmetric solvers". In: *Comput. Methods Appl. Mech. Eng.* 184.2-4 (2000), S. 501–520 (siehe S. 16).

/AME 17/ P. Amestoy u. a. *MUltifrontal Massively Parallel Solver (MUMPS 5.1.1) Users' guide*. `http://mumps.enseeiht.fr/doc/userguide_5.1.1.pdf`; Zugriff 17. Mai, 2017. 2017 (siehe S. 16 ff., 188).

/AME 16/ P. Amestoy u. a. *MUMPS Web page*. `http://mumps.enseeiht.fr`. 2016 (siehe S. 7, 12, 16).

/AUS 16a/ H. Austregesilo u. a. *ATHLET Mod 3.1A – Models and Methods*. Teil der Dokumentation zur ATHLET Software. März 2016 (siehe S. 3, 9, 23, 53, 56, 124).

/AUS 16b/ H. Austregesilo u. a. *ATHLET Mod 3.1A – Programmer's Manual*. Teil der Dokumentation zur ATHLET Software. März 2016 (siehe S. 23).

/BAL 16/ S. Balay u. a. *PETSc Web page*. `http://www.mcs.anl.gov/petsc`. 2016 (siehe S. 7 f., 12 f., 21).

/BRO 11/ M. Brookes. *The Matrix Reference Manual*. online veröffentlicht. `http://www.ee.imperial.ac.uk/hp/staff/dmb/matrix/intro.html`; Zugriff 03. November, 2016. 2011 (siehe S. 118, 122).

/BUT 64/ J. C. Butcher. "Implicit Runge-Kutta Processes". In: *Math. Comp.* 18.85 (Jan. 1964), S. 50–64 (siehe S. 70, 85).

/BUT 16/ J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. 3. Aufl. John Wiley & Sons, Ltd., 2016 (siehe S. 3, 64, 92).

/BUT 90/ J. C. Butcher und J. R. Cash. “Towards Efficient Runge-Kutta Methods for Stiff Systems”. In: *SIAM J. Numer. Anal.* 27.3 (Juni 1990), S. 753–761 (siehe S. 86, 127).

/BUT 98/ J. C. Butcher und R. P. K. Chan. “Efficient Runge-Kutta Integrators for Index-2 Differential Algebraic Equations”. In: *Math. Comp.* 67.223 (Juli 1998), S. 1001–1021 (siehe S. 86, 99).

/CEA 06/ CEA, CNRS und INRIA. *CeCILL-C FREE SOFTWARE LICENSE AGREEMENT*. `http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.html`. 2006 (siehe S. 21).

/COL 83/ T.F. Coleman und J.J. More. “Estimation of Sparse Jacobian Matrices and Graph Coloring Problems”. In: *SIAM J. Numer. Anal.* 20 (1983), S. 187–209 (siehe S. 43, 47).

/DAV 04/ T. A. Davis. “Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method”. In: *ACM Trans. Math. Softw.* 30.2 (2004), S. 196–199 (siehe S. 15).

/DAV 06/ T. A. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. SIAM, 2006 (siehe S. 16).

/DEM 99/ J. W. Demmel u. a. “A supernodal approach to sparse partial pivoting”. In: *SIAM J. Matrix Anal. Appl.* 20.3 (1999), S. 720–755 (siehe S. 16).

/DEU 04/ P. Deuflhard. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*. 1. Aufl. Bd. 35. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2004 (siehe S. 119 f., 135 f., 167).

/DEU 85/ P. Deuflhard. “Recent Progress in Extrapolation Methods for Ordinary Differential Equations”. In: *SIAM Review* 27.4 (Dez. 1985), S. 505–535 (siehe S. 78, 135, 144, 157).

/DEU 08/ P. Deuflhard und F. Bornemann. *Numerische Mathematik [Band] 1: Eine algorithmisch orientierte Einführung*. 4. Aufl. De Gruyter Berlin Boston, 2008 (siehe S. 3).

/DEU 13/ P. Deuflhard und F. Bornemann. *Numerische Mathematik [Band] 2: Gewöhnliche Differentialgleichungen*. 4. Aufl. De Gruyter Berlin Boston, 2013 (siehe S. 3, 64, 81, 141).

/DIB 13/ V. Diba. “Entwicklung und Analyse von Zeitadaptivitätsstrategien mit Methoden der Regelungstechnik”. Magisterarb. Universität Kassel FB 10 – Arbeitsgruppe Analysis und Angewandte Mathematik, Feb. 2013 (siehe S. 140).

/DIJ 68/ E. W. Dijkstra. “Go To Statement Considered Harmful”. In: *Communication of the ACM* 11.3 (1968), S. 147–148 (siehe S. 24).

/ENT 16/ Scilab Enterprises. *Scilab. Version 5.5.2*. `http://www.scilab.org`. 2016 (siehe S. 15, 52, 125, 130).

/FOR 17/ Forcheck b.v. *Forcheck Web page*. `http://www.forcheck.nl`. 2017 (siehe S. 4).

/FRE 07a/ Inc. Free Software Foundation. *GNU Lesser General Public License*. `http://www.gnu.org/licenses/lgpl-3.0.en.html`. 2007 (siehe S. 20).

/FRE 07b/ R. W. Freund und H. W. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. 10. Aufl. Springer Berlin Heidelberg, 2007 (siehe S. 3).

/GEB 05/ A. H. Gebremedhin, F. Manne und A. Pothen. “What Color Is Your Jacobian? Graph Coloring for Computing Derivatives”. In: *SIAM Rev.* 47.4 (2005), S. 629–705 (siehe S. 43 f.).

/GEB 13/ A. H. Gebremedhin, D. Nguyen, Md. M. A. Patwary und A. Pothen. “ColPack: Software for graph coloring and related problems in scientific computing”. In: *ACM Trans. Math. Softw.* 40.1 (2013) (siehe S. 14, 44).

/GRI 08/ A. Griewank und A. Walther. *Evaluating Derivatives – Principles and Techniques of Algorithmic Differentiation*. 2. Aufl. SIAM, 2008 (siehe S. 14).

/GUS 94/ K. Gustafsson. “Control theoretic techniques for stepsize selection in implicit Runge-Kutta methods”. In: *ACM Trans. Math. Softw.* 4 (1994), S. 496–517 (siehe S. 134, 138, 141 f.).

/GUS 97/ K. Gustafsson und G. Söderlind. “Control Strategies for the Iterative Solution of Nonlinear Equations in ODE Solvers”. In: *SIAM J. Sci. Comput.* 18.1 (Jan. 1997), S. 23–40 (siehe S. 132, 134 f., 137, 144, 153).

/HAI 93/ E. Hairer, S. P. Nørsett und G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2. Aufl. Bd. 8. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 1993 (siehe S. 3, 54 ff., 64, 75, 83, 87, 92, 138, 141, 143 f.).

/HAI 96/ E. Hairer und G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 2. Aufl. Bd. 14. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 1996 (siehe S. 3, 23, 58, 64, 66, 68, 70, 78 f., 81, 86, 92 f., 97, 120, 126, 135–138, 144–147, 149, 157).

/HEE 17/ D. van Heesch. *Doxygen Web page*. `http://www.stack.nl/~dimitri/doxygen`. 2017 (siehe S. 4).

/HER 03/ Michael Heroux u. a. *An Overview of Trilinos*. Techn. Ber. SAND2003-2927. Sandia National Laboratories, 2003 (siehe S. 15).

/HOF 81/ E. Hofer. “An A(α)-Stable Variable Order ODE-Solver and its Application as Advancement Procedure for Simulations in Thermo- and Fluid-Dynamics”. In: *Proceedings of the International Topical Meeting on Advances in Mathematical Methods for the Solution of Nuclear Engineering Problems*. München, Apr. 1981 (siehe S. 56).

/HOU 85/ N. Houbak, S. P. Nørsett und P. G. Thomsen. “Displacement or Residual Test in the Application of Implicit Methods for Stiff Problems”. In: *IMA J. Numer. Anal.* 5 (1985), S. 297–305 (siehe S. 136 f.).

/HUN 03/ W. Hundsdorfer und J. G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Bd. 33. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2003 (siehe S. 51, 73, 75 f.).

/JAC 13/ V. Jacht. *Implementation and Benchmarking of Sparse-Matrix Solvers for Two-Phase Flow Applications*. Bachelorarb. Technische Universität München – Fakultät für Informatik. Juni 2013 (siehe S. 15).

/JAC 17a/ V. Jacht. “Verteilte Parallelisierung thermohydraulischer Simulationen in ATHLET mithilfe von PETSc”. Magisterarb. Technische Universität München – Fakulät für Informatik, März 2017 (siehe S. 175, 180, 217).

/JAC 17b/ V. Jacht, J. Scheuer und T. Steinhoff. *An MPI-based software architecture for coupling thermal-hydraulic codes with PETSc*. Vortrag beim PETSc User Meeting 2017: UC Boulder, CO, USA; Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH. Juni 2017 (siehe S. 14).

/JAC 96/ K. R. Jackson, A. Kværnø und S. P. Nørsett. “An Analysis of the Order of Runge-Kutta Methods that Use an Iterative Scheme to Compute Their Internal Stage Values”. In: *BIT* 36.4 (1996), S. 713–765 (siehe S. 82, 85).

/KAR 99/ G. Karypis und V. Kumar. “A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM J. Sci. Comput.* 20 (1999), S. 359–392 (siehe S. 18).

/KAR 16/ G. Karypis und V. Kumar. *Metis Web page*. `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`. 2016 (siehe S. 7, 12, 18).

/KAW 17/ K. Kawaguchi. *The Jenkins project*. `https://jenkins.io`. 2017 (siehe S. 177).

/KNE 16/ M. Knepley. *The Portable Extensible Toolkit for Scientific Computing*. http://www.mcs.anl.gov/petsc/documentation/tutorials/TutorialCEMRACS2016.pdf; Zugriff 15. August, 2016. 2016 (siehe S. 12 f.).

/KVÆ 04/ A. Kværnø. *Singly Diagonally implicit Runge-Kutta Methods with an Explicit First Stage*. Techn. Ber. 1. preprint. Norges Teknisk-Naturvitenskapelige Universitet, Institutt for matematiske fag, 2004 (siehe S. 127).

/LAB 16/ Lawrence Livermore National Laboratory. *HYPRE high performance preconditioners – User's Manual*. http://computation.llnl.gov/sites/default/files/public/hypre-2.11.1_usr_manual.pdf; Zugriff 04. Oktober, 2016. 2016 (siehe S. 20).

/LER 16a/ G. Lerchl u. a. *ATHLET Mod 3.1A – User's Manual*. Teil der Dokumentation zur ATHLET Software. März 2016 (siehe S. 23, 41).

/LER 16b/ G. Lerchl u. a. *ATHLET Mod 3.1A – Validation*. Teil der Dokumentation zur ATHLET Software. März 2016 (siehe S. 178).

/LI 13/ X. Li. *Direct Solvers for Sparse Matrices*. online veröffentlicht. http://crd-legacy.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf; Zugriff 10. November, 2016. Juli 2013 (siehe S. 16 f.).

/MAC 99/ W. Mackens, J. Menck und H. Voß. *Coupling Iterative Subsystem Solvers*. Techn. Ber. 26. TU Hamburg-Harburg, AB Mathematik, Feb. 1999 (siehe S. 163).

/MAX 15/ Maxima. *Maxima, a Computer Algebra System. Version 5.37.5*. http://maxima.sourceforge.net/. 2015 (siehe S. 125).

/MEI 11/ A. Meister. *Numerik linearer Gleichungssysteme*. 4. Aufl. Vieweg+Teubner, 2011 (siehe S. 3, 8, 13, 99).

/MET 11/ M. Metcalf, J. Reid und M. Cohen. *Modern Fortran Explained*. 4. Aufl. Oxford University Press, 2011 (siehe S. 4).

/PÉR 11/ Y. Périn u. a. “Multi-scale coupled code systems: from coarse-mesh to high-fidelity LWR core calculations”. In: *Kerntechnik* 76.3 (2011), S. 154–159 (siehe S. 163).

/PRO 74/ A. Prothero und A. Robinson. “On the Stability and Accuracy of One-Step Methods for Solving Stiff Systems of Ordinary Differential Equations”. In: *Math. Comp.* 28.125 (Jan. 1974), S. 145–162 (siehe S. 66).

/ROM 16/ J. E. Roman, C. Campos, E. Romero und A. Tomás. *SLEPc Users Manual Scalable Library for Eigenvalue Problem Computations*. `http://slepc.upv.es/documentation/slepc.pdf`; Zugriff 20. Juni, 2016. 2016 (siehe S. 16).

/ROO 99/ H.-G. Roos und H. Schwetlick. *Numerische Mathematik: Das Grundwissen für jedermann*. B. G. Teubner Stuttgart Leipzig, 1999 (siehe S. 3).

/SAA 03/ Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2. Aufl. SIAM, 2003 (siehe S. 3, 41).

/SAA 11/ Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. 2. Aufl. SIAM, 2011 (siehe S. 3).

/SAN 16/ Sandia National Laboratories. *Trilinos Web page*. `https://trilinos.org`. 2016 (siehe S. 15).

/SCH 13/ A. Schaffrath u. a. *Vorgehensweise bei der Weitergabe der im Rahmen der Reaktorsicherheitsforschung des BMWi entwickelten GRS-Codes*. Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH. Nov. 2013 (siehe S. 19).

/SCH 02/ O. Schenk. *Lineare Gleichungssysteme I*. online veröffentlicht. `http://fgb.informatik.unibas.ch/lectures/archive/SS2002/algwiss/Lekt9.pdf`; Zugriff 20. April 20, 2015. 2002 (siehe S. 17 f., 45).

/SHA 93/ L. F. Shampine. “Ill-Conditioned Matrices and the Integration of Stiff ODEs”. In: *J. Comp. Appl. Math.* 48 (1993), S. 279–292 (siehe S. 96, 136).

/SHA 94/ L. F. Shampine. *Numerical Solution of Ordinary Differential Equations*. Chapman und Hall/CRC, 1994 (siehe S. 3, 64, 138).

/SHA 97/ L. F. Shampine. “The MATLAB ODE Suite”. In: *SIAM J. Sci. Comput.* 18.1 (Jan. 1997), S. 1–22 (siehe S. 143).

/SHA 81/ L. F. Shampine. “Type-Insensitive ODE Codes Based on Implicit *A*-Stable Formulas”. In: *Math. Comp.* 36.154 (Apr. 1981), S. 499–510 (siehe S. 153).

/SÖD 02/ G. Söderlind. “Automatic control and adaptive time-stepping”. In: *Num. Alg.* 31 (2002), S. 281–310 (siehe S. 134, 141).

/SÖD 03/ G. Söderlind. “Digital filters in adaptive time-stepping”. In: *ACM Trans. Math. Softw.* 29 (2003), S. 1–26 (siehe S. 134, 140 f.).

/SOL 16/ Cygwin Solutions, Red Hat u. a. *Cygwin Web page*. `https://www.cygwin.com/`. 2016 (siehe S. 12).

/SON 08/ P. Sonneveld und M. B. Gijzem. “IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Sys-tems of Linear Equations”. In: *SIAM J. Sci.* 31.2 (2008), S. 1035–1062 (siehe S. 8).

/STA 15/ M. Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. 3. Aufl. Free Software Foundation, Inc., 2015 (siehe S. 20).

/STE 16/ T. Steinhoff. *Providing a Single Point Spectrum for Runge-Kutta Schemes of High Stage Order Based on Perturbed Collocation*. Techn. Ber. preprint. GRS, 2016 (siehe S. 92, 95).

/STE 17/ T. Steinhoff. *Reisebericht zum PETSc User Meeting 2017*. Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH. Juni 2017 (siehe S. 14).

/STO 05/ J. Stoer und R. Bulirsch. *Numerische Mathematik 2*. 5. Aufl. Springer Berlin Heidelberg, 2005 (siehe S. 99).

/TEN 17/ Univ. of Tennessee, Berkeley Univ. of California, Univ. of Colorado Denver und NAG Ltd. *ScaLAPACK*. http://www.netlib.org/scalapack/index.html. 2017 (siehe S. 21).

/TER 08/ V. A. Tereshonok u. a. *Kalinin -3 Coolant Transient Benchmark – Switching-off of One of the Four Operating Main Circulation Pumps at Nominal Reactor Power*. OECD/NEA-DEC. 2008 (siehe S. 49, 183).

/THE 04/ The Apache Software Foundation. *Apache License Version 2.0*. http://www.apache.org/licenses/LICENSE-2.0. 2004 (siehe S. 21).

/TIN 67/ W. F. Tinney und J. W. Walker. “Direct solutions of sparse network equations by optimally ordered triangular factorization”. In: *Proceedings of the IEEE* 55.11 (1967), S. 1801–1809 (siehe S. 45).

/TIR 17/ P. Tirumalai u. a. *OpenMP – The OpenMP API specification for parallel programming*. http://www.openmp.org. 2017 (siehe S. 19).

/VÖL 10/ C. Völcker, J. B. Jørgensen, P. G. Thomsen und E. H. Stenby. “Adaptive Stepsize Control in Implicit Runge-Kutta Methods for Reservoir Simulation”. In: *Proceedings of the 9th International Symposium on Dynamics and Control of Process Systems*. Leuven, Belgium, Juli 2010 (siehe S. 132 ff., 136 ff.).

/VOSS 07a/ H. Voß. *Grundlagen der Numerischen Mathematik*. online veröffentlicht. https://www.mat.tuhh.de/lehre/material/grnummath.pdf; Zugriff 27. April, 2016. 2007 (siehe S. 3).

/VOSS 96/ H. Voß. *Mathematik für Studierende der Ingenieurwissenschaften III*. online veröffentlicht. https://www.mat.tuhh.de/lehre/material/IngMath_3.pdf; Zugriff 11. Mai, 2015. 1996 (siehe S. 167, 170).

/VOSS 07b/ H. Voß. *Numerische Simulation*. online veröffentlicht. https://www.mat.tuhh.de/lehre/material/numersim.pdf; Zugriff 27. April, 2016. 2007 (siehe S. 3).

/WIK 16a/ Wikipedia. *BSD licenses*. https://en.wikipedia.org/wiki/BSD_licenses; Zugriff 11. April, 2016. 2016 (siehe S. 21).

/WIK 16b/ Wikipedia. *GPL licenses*. https://en.wikipedia.org/wiki/GNU_General_Public_License; Zugriff 12. April, 2016. 2016 (siehe S. 20).

/ZED 90/ H. Zedan. “Avoiding the Exactness of the Jacobian Matrix in Rosenbrock Formulae”. In: *Computers Math. Applic.* 19.2 (1990), S. 83–89 (siehe S. 143).

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Verzeichnis von Codeabschnitten

A Anhang

A.1 Signaturen zum Funktionenkatalog im NuT-Plug-In

Nachstehend die Signaturen nebst einigen Erläuterungen zu den Funktionen, die über das NuT-Plug-In zur Verfügung gestellt werden. Der Katalog kann leicht nach Bedarf erweitert werden. Grundsätzlich bedienen sich alle Funktionen des gleichen MPI-Kommunikationsmodelles, wie es von der Prinzipweise im Beispiel in Unterabschnitt 2.3.1.3 dargelegt wird – s. auch Anhang A.2.

Code A.1 Funktionenkatalog im NuT-Plug-In

```
      subroutine execute(ent, func)
      ! internal function, execute specific entity [ent] with
         stated function [func] by means of NuT-worker(s)
      ! ...
         integer, intent(in) :: ent
         integer, intent(in) :: func
      ! ...
      end subroutine

! -----------------------------------------------------
! controller
! -----------------------------------------------------
!_PROC_EXPORT(initialize)
      subroutine initialize()
      ! initialize communication with NuT-worker(s) by setting up
         MPI-communicators
      ! ...
      end subroutine

!_PROC_EXPORT(finalize)
      subroutine finalize()
      ! finilize communication with NuT-worker(s), free all
         communicators
      ! ...
      end subroutine

!_PROC_EXPORT(barrier)
      subroutine barrier()
      ! initiate a MPI-barrier
      ! ...
      end subroutine
```

```
                                                                    29
! ------------------------------------------------------          30
! ode                                                               31
! ------------------------------------------------------          32
!_PROC_EXPORT(ode_initialize)                                       33
      subroutine ode_initialize(n)                                  34
      ! initialize ode-context                                      35
      ! [n] reflects the dimension of the problem                   36
         nutInteger, intent(in) :: n                                37
      ! ...                                                         38
      end subroutine                                                39
                                                                    40
!_PROC_EXPORT(ode_computeStages)                                    41
      subroutine ode_computeStages(code)                            42
      ! compute the stages of the current FiterRK-method,           43
      ! by [code] reverse communication is established              44
      ! ...                                                         45
         nutInteger, intent(inout) :: code                          46
      ! ...                                                         47
      end subroutine                                                48
                                                                    49
!_PROC_EXPORT(ode_test)                                             50
      subroutine ode_test(code)                                     51
      ! invoke simple test run of the current FiterRK-method,       52
      ! by [code] reverse communication is established              53
      ! ...                                                         54
         integer, intent(inout) :: code                             55
      ! ...                                                         56
      end subroutine                                                57
                                                                    58
!_PROC_EXPORT(ode_getFInput)                                        59
      subroutine ode_getFInput(i, vector, scalar)                   60
      ! aks for t [scalar], and y [vector] to evaluate f(t,y) for   61
         index [i], see ode_getFIndices
      ! ...                                                         62
         nutInteger, intent(in) :: i                                63
         nutReal, dimension(:), allocatable, intent(inout) ::vector 64
         nutReal, intent(out) :: scalar                             65
      ! ...                                                         66
      end subroutine                                                67
                                                                    68
                                                                    69
!_PROC_EXPORT(ode_getFIndices)                                      70
```

```
      subroutine ode_getFIndices(index, number)
      ! ask for start [index] and [number] of f-evaluations that
         shall be executed in a row by ATHLET
      ! ...
         nutInteger, intent(out) :: index
         nutInteger, intent(out) :: number
      ! ...
      end subroutine

!_PROC_EXPORT(ode_getSolution)
      subroutine ode_getSolution(yfinal, yo1, error)
      ! receive final local approximation [yfinal], special order 1
         approximation [yo1] and the [error] estimate from the NuT
         -worker(s)
      ! ...
         nutReal, dimension(:), allocatable, intent(inout) ::
            yfinal
         nutReal, dimension(:), allocatable, intent(inout) :: yo1
         nutReal, dimension(:), allocatable, intent(inout) :: error
      ! ...
      end subroutine

!_PROC_EXPORT(ode_reset)
      subroutine ode_reset(tag)
      ! ...
      ! hard or soft reset of FiterRK data for the current local
         time step, type of reset depends on [tag]
         nutInteger, intent(in) :: tag
      ! ...
      end subroutine

!_PROC_EXPORT(ode_setFOutput)
      subroutine ode_setFOutput(i, vector)
      ! send f-evaluation to the NuT-worker(s) for index [i], see
         ode_getFIndices
      ! ...
         nutInteger, intent(in) :: i
         nutReal, dimension(:), allocatable, intent(in) :: vector
      ! ...
      end subroutine

!_PROC_EXPORT(ode_setTimeStepSize)
      subroutine ode_setTimeStepSize(h)
```

```
      ! send the current local time step [h] to the NuT-worker(s)
      ! ...
         nutReal, intent(in) :: h
      ! ...
      end subroutine

! ------------------------------------------------------
! ftrix (linalg)
! ------------------------------------------------------
!_PROC_EXPORT(ftrix_initialize)
      subroutine ftrix_initialize(block_ePtr, block_rowPtr,
         block_colInd)
      ! initialize linear algebra entity with data by sending CSR-
         like pattern information of the Jacobian by means of [
         block_ePtr], [block_rowPtr] and [block_colInd]
      ! ...
         nutInteger, dimension(:), allocatable, intent(in) ::
            block_ePtr
         nutInteger, dimension(:), allocatable, intent(in) ::
            block_rowPtr
         nutInteger, dimension(:), allocatable, intent(in) ::
            block_colInd
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_finalize)
      subroutine ftrix_finalize()
      ! finalize ftrix-communication
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_buildMatStructure)
      subroutine ftrix_buildMatStructure(mat, activeRows,
         activeColumns, reuseMat)
      ! select matrix by [mat] (e.g. coloring Jacobian or real
         Jacobian) and build up the matrix strucure according to [
         activeRows] and [acticeColumns], exploit current structure
          if demanded by [reuseMat]
      ! ...
         nutInteger, intent(in) :: mat
         nutBool, dimension(:), allocatable, intent(in) ::
            activeRows
         nutBool, dimension(:), allocatable, intent(in) ::
```

```
            activeColumns
         nutBool, intent(in) :: reuseMat
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_clearFullJac)
      subroutine ftrix_clearFullJac()
      ! clear all elements of the Jacobian
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_clearPartialJac)
      subroutine ftrix_clearPartialJac(activeElements)
      ! clear particular elements of the Jacobian determined by [
         activeElements]
      ! ...
         nutBool, dimension(:), allocatable, intent(in) ::
            activeElements
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_clearJacOffset)
      subroutine ftrix_clearJacOffset()
      ! clear the shift parameter in (-shift*I+J)
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_solve)
      subroutine ftrix_solve(rhs_array, diagonalOffset, rhs_size)
      ! let the NuT-worker(s) solve a linear system for the rhs [
         rhs_array] of size [rhs_size] and for the shift parameter
         [diagonalOffset] in (-shift*I+J)
      ! ...
         nutReal, dimension(:), allocatable, intent(inout) ::
            rhs_array
         nutReal, intent(inout) :: diagonalOffset
         nutInteger, intent(in) :: rhs_size
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_computeColoring)
      subroutine ftrix_computeColoring(colInd_size, rowPtr_size)
      ! let the NuT-worker(s) compute a coloring for the Jacobian
```

```
         and receive size information [colInd_size] and [
         rowPts_size] regarding the coloring
      ! ...
         nutInteger, intent(out) :: colInd_size, rowPtr_size
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_getSeed)
      subroutine ftrix_getSeed(colInd, rowPtr)
      ! aks for the seed matrix S, received in CSR-format [colInd]
         and [rowPtr] (i. e. S^T)
      ! ...
         nutInteger, dimension(:), allocatable, intent(inout) ::
            colInd, rowPtr
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_addPerturbation)
      subroutine ftrix_addPerturbation(colInd, rowInd, val, size)
      ! send data ([val]) of size [size] from finite differences to
          the NuT-worker(s) to store it in the Jacobian, the
         arrangement where to put it is given by [colInd] and [
         rowInd]
      ! ...
         nutInteger, dimension(:), allocatable, intent(in) ::
            colInd
         nutInteger, dimension(:), allocatable, intent(in) ::
            rowInd
         nutReal, dimension(:), allocatable, intent(in) :: val
         nutInteger, intent(in) :: size
      ! ...
      end subroutine

!_PROC_EXPORT(ftrix_view)
      subroutine ftrix_view()
      ! view the current Jacobian
      ! ...
      end subroutine
```

A.2 Masterarbeit Volker Jacht

Beiliegend findet sich die Abschlussarbeit von Herrn Volker Jacht zur Erlangung des akademischen Grades Master of Science in Informatik durch die TU München, /JAC 17a/. Die Arbeit wurde im Zeitraum von Oktober 2016 bis März 2017 angefertigt. Der Titel lautet

Verteilte Parallelisierung thermohydraulischer Simulationen
in ATHLET mithilfe von PETSc

Im Rahmen jener Abschlussarbeit wird das informationstechnologische Konzept der MPI-basierten NuT-Architektur entwickelt und erläutert. Einige dieser Aspekte sind auch bereits in Kapitel 2 in der allgemeinen Vorstellung der NuT-Architektur aufgegriffen. Hauptsächlich dient die Anlage jedoch als Referenz zu Kapitel 5.

Aufgrund des Kooperationscharakters zwischen der TU München und der GRS war die Betreuung zweigeteilt. Die akademische Seite wurde durch Prof. Dr. Hans-Joachim Bungartz sowie Dr. Tobias Neckel vom Institut für Informatik V repräsentiert. Seitens der GRS war Dr. Tim Steinhoff betreuend tätig.

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Masterarbeit in Informatik

Verteilte Parallelisierung thermohydraulischer Simulationen in ATHLET mithilfe von PETSc

Volker Jacht

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Masterarbeit in Informatik

Verteilte Parallelisierung thermohydraulischer Simulationen in ATHLET mithilfe von PETSc

Distributed parallelization of thermohydraulic simulations in ATHLET using PETSc

Autor:	Volker Jacht
Aufgabensteller:	Prof. Dr. Hans-Joachim Bungartz
Betreuer:	Dr. Tim Steinhoff, Dr. Tobias Neckel
Abgabedatum:	15. März 2017

Ich versichere, dass ich diese Masterarbeit in Informatik selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15. März 2017 Volker Jacht

Inhaltsverzeichnis

Abkürzungen und Begriffe

ATHLET	Analysis of THermal-hydraulics of LEaks and Transients
AL	Abstrakter Löser
FEBE	Modul zur Zeitintegration (Forward-Euler, Backward-Euler)
FTRIX	Modul für dünnbesetzte Systeme (Sparse Matrix Package)
GLS	Gleichungssystem
ITAB-Arrays	Blockstruktur der Jacobi-Matrix
K3	Kalinin-3
MPI	Message Passing Interface
MUMPS	MUltifrontal Massively Parallel Solver
NuT	Numerical Toolkit
OpenMP	Open Multi-Processing
PETSc	Portable, Extensible Toolkit for Scientific Computation
PWR	Pressurized Water Reactor
Seed-Matrix	Farbinformationen der Jacobi-Matrix
TOP-Array	Zustand, welche Gleichungen aktiv/inaktiv sind

1 Einleitung

Die Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH führt Sicherheitsanalysen für nukleartechnische Anlagen durch. Ihr Schwerpunkt liegt auf der Entwicklung von Simulationsprogrammen, die hypothetische Zwischenfälle im Reaktor vorhersagen sollen. Zur Berechnung der Zweiphasenströmung im Kühlsystem eines Reaktors wird das thermohydraulische Simulationsprogramm ATHLET (Analysis of THermal-hydraulics of LEaks and Transients) entwickelt. Es simuliert mithilfe einer speziellen Finite-Volumen-Diskretisierung die Dynamik im Kühlkreislauf. Dabei entstehen üblicherweise Gleichungssysteme mit etwa 6.000 Unbekannten. Ein neuer Modellierungsansatz kann einen einstellbaren Anteil des Reaktorkerns detaillierter berechnen. Dies führt zu einer Vergrößerung der Gleichungssysteme auf bis zu 1.000.000 Unbekannten. Die Berechnung solcher Modelle kann in ATHLET bereits mehrere Monate dauern.
Eine besonders große Rolle im Bereich der Computersimulation spielt das wissenschaftliche Rechnen. Es beschäftigt sich mit der Entwicklung von effektiven Modellen und Algorithmen. Diese sind zur Durchführung großer Simulationen unerlässlich, da ungeeignete Verfahren schnell zu einem großen Ansteig an Rechenzeit führen.
Der im Argonne National Laboratory entwickelte numerische Toolkit PETSc bietet eine umfangreiche Sammlung an Lösungsmethoden für das wissenschaftliche Rechnen und kann gut in bestehnde Codes integriert werden.
Im Rahmen dieser Arbeit wird der ATHLET-Code auf sein mögliches Verbesserungspotential bezüglich seiner Rechengeschwindigkeit untersucht. Anhand der Ergebnisse wird ein Konzept entwickelt, wie diese Programmteile mithilfe von PETSc beschleunigt werden können. Neben den neuen Lösungsmethoden soll auch eine verteilte Parallelisierung umgesetzt werden, damit zukünftige Simulationen über einen Computer hinaus berechnet werden können.
Diese Masterarbeit ist aus einer Kooperation zwischen GRS und dem Lehrstuhl für Wissenschaftliches Rechnen (SCCS) an der Technischen Universität München enstanden.

2 ATHLET

Das Simulationsprogramm ATHLET wird bereits seit mehreren Jahrzehnten in der GRS entwickelt. Dieses Kapitel beschreibt zuerst die wichtigsten Aspekte der Numerik und ihrer Implementierung. Danach werden zwei wichtige Modelle vorgestellt, die im Anschluss als Berwertungsgrundlage dienen.

2.1 Numerik

Als physikalisches Grundmodell für den Kühlkreislauf dient der Masse- und Impulserhaltungssatz von Flüssigkeiten und Dampf. Der Kühlkreislauf besteht aus einem Netzwerk aus Kontrollvolumina und deren Verbindungen. Diese Netzwerkelemente werden mithilfe einer speziellen Finite-Volumen-Methode diskretisiert und ergeben ein steifes, nichtlineares, gewöhnliches Differentialgleichungssystem 1. Ordnung:

$$y'(t) = f(t, y(t)) \tag{2.1}$$

$$f : \mathbb{R}^{n+1} \to \mathbb{R}^n$$
$$y : \mathbb{R} \to \mathbb{R}^n$$

mit dem Anfangswert

$$y(t_0) = y_0$$

Die Dimension n des Lösungsvektors y ist von der Zahl der Netzwerkelemente und der Zahl der Lösungsvariablen pro Element abhängig.

2.1.1 Zeitintegration

Um das Anfangswertproblem der Differentialgleichung zu lösen, muss sie über die Zeit integriert werden [HS06, Kap. 26]. Da explizite Verfahren nur unzureichend

stabil für steife Differentialgleichungen sind, wird ausgehend vom impliziten Euler-Verfahren [Bun+13, Kap. 2.4.5]

$$\frac{y_{n+1} - y_n}{\Delta t_n} = f(t_{n+1}, y_{n+1}),\ n = 0, 1, ... \tag{2.2}$$

$$t_{n+1} = t_n + \Delta t_n$$

das linear-implizite Euler-Verfahren [HNW93, Kap. 4.9] benutzt,

$$\underbrace{\left[\Delta t^{-1} \cdot I - \overbrace{\frac{\partial f(y_n, t_n)}{\partial y}}^{\substack{\text{Jacobi-Matrix} \\ =J}}\right]}_{\substack{=A \\ \text{Systemmatrix}}} \cdot \Delta y = f(y_n, t_n) + \frac{\partial f(y_n, t_n)}{\partial t} \cdot \Delta t \tag{2.3}$$

welches sich aus einer Taylorentwicklung 1. Ordnung von Gleichung 2.2 ergibt:

$$\begin{aligned} \frac{y_{n+1} - y_n}{\Delta t} = & f(t_n, y_n) \\ & + \frac{\partial f(t_n, y_n)}{\partial y} \cdot (y_{n+1} - y_n) + \frac{\partial f(t_n, y_n)}{\partial t} \cdot \Delta t + \mathcal{O}(\Delta t) \end{aligned} \tag{2.4}$$

Aufgrund von Gleichung 2.3 muss ein Gleichungssystem der Form $Ax = b$ gelöst werden [Aus+16, Kap. 6.2]. Das linear-implizite Euler-Verfahren gehört zu den W-Methoden. Daher ist die Ordnung unbahänig von der verwendeten Jacobi-Matrix. Im Sinne der Stabilität sollte allerdings für die Jacobi-Matrix eine möglichst gute Annäherung für $\frac{\partial f(y_n, t_n)}{\partial y}$ verwendet werden. [HNW93, Kap. 4.9]. Die Approximation der Jacobi-Matrix wird in Unterabschnitt 2.1.2 beschrieben. Zusammen mit der gewichteten Einheitsmatrix I ergibt sich die Systemmatrix A.
Da das linear-implizite Euler-Verfahren von Konsistenzordnung 1 ist, gilt für den lokalen Diskretisierungsfehler, dass dieser in $\mathcal{O}(\Delta t^2)$ liegt. Um höhere Konsistenzordnung zu errreichen, wird in ATHLET ein Extrapolationsverfahren bis zur Ordnung 3 verwendet [Aus+16, Kap. 6.2.1].

Die zu berechnende Lösung y_{n+1} an $t_n + \Delta t_n$ wird im Folgenden als gesamter Zeitschritt bezeichnet. Anstatt sie durch einen einfachen Zeitschritt zu berechnen, werden nacheinander mehrere Lösungen mit steigender Genauigkeit erstellt. Dafür sei $T_{m,1}$ die von y_n aus approximierte Lösung an der Stelle $t_n + \Delta t_n$ durch m-faches Anwenden des linear-impliziten Euler-Verfahrens mit einer Teilzeitschrittweite von $\frac{\Delta t_n}{m}$. Für die kanonische Folge von Teilungsfaktoren $\{m_i\} = \{1, 2, 3\}$ werden Approximationen höherer Ordnung mittels Extrapolation über das Aitken-Neville Schema berechnet. Siehe auch Abbildung 2.1.

$$T_{j,k+1} = T_{j,k} + \frac{T_{j,k} - T_{j-1,k}}{(m_j / m_{j-k}) - 1} \tag{2.5}$$

Allerdings führt die Steigerung der Ordnung durch das Extrapolationsverfahren zu einer Verringerung der Stabilität des Verfahrens. Für $p = 2$ ist es A-Stabil und für die hauptsächlich verwendete Ordnung $p = 3$ immer noch $A(\alpha)$-stabil mit $\alpha \approx 90°$. In der Quelle [GR03] wird davon ausgegangen, dass sich die Stabilität für die Ordnung $p \leq 3$ nicht negativ auf ATHLET auswirkt.

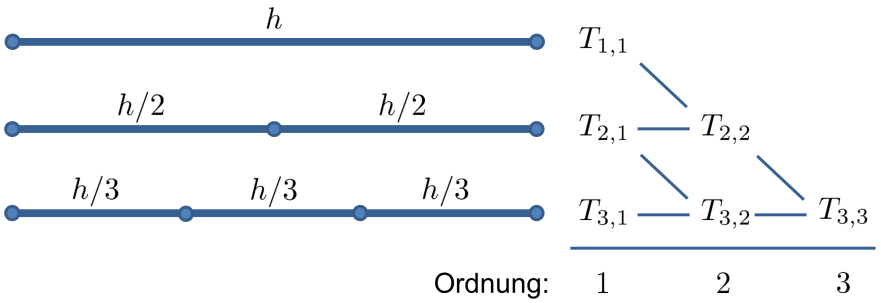


Abbildung 2.1: Aitken-Neville Schema zur Extrapolation. $h = \Delta t$ [Ste17, Abb. 3.1]

2.1.2 Jacobi-Matrix

Die Jacobi-Matrix J beinhaltet sämtliche partiellen Ableitungen erster Ordnung der kontinuierlich differenzierbaren Funktion f an der Stelle y. Es gilt $f'(y)_{ij} = \frac{\partial f_i}{\partial y_j}(y)$. In der Regel sind Netzwerkelement nur von wenigen adjazenten Elementen abhängig. Ist das Element i nicht abhängig vom Element j, so gilt für den Eintrg der Jacobi-Matrix $J_{ij} = 0$. In ATHLET ist die Anzahl der Nichtnullelemente der Jacobi-Matrix, im Gegensatz zu einer dichtbesetzten Matrix mit $\mathcal{O}(n^2)$, durch $\mathcal{O}(n)$ beschränkt. Somit lohnt es sich, die dünnbesetzte Struktur für die Speicherung und die darauf arbeitenden Algorithmen zu berücksichtigen. Bei einer dünnbesetzten Matrix werden Nichtnulleinträge explizit gespeichert und Nulleinträge durch Nichtspeicherung implizit als Null interpretiert. Dies erfolgt in ATHLET durch die Verwendung einer Blockmatrix, die die Positionen und Größe aller Blöcke speichert. Jeder Block wird auf Elementebene als dichtbesetzt angesehen und repräsentiert jeweils ein einzelnes Netzwerkelement [Aus+16, Kap. 6.1].

Die Jacobi-Matrix J wird durch finite Differenzen mittels gestörter Funktionsauswertung der rechten Seite der Differentialgleichung approximiert [Aus+16, Kap. 6.3].

$$J(y)d \approx \frac{f(y + \varepsilon d) - f(y)}{\varepsilon} \tag{2.6}$$

mit

$$d \in \mathbb{R}^n$$
$$\text{Störfaktor: } \varepsilon$$

Ist mit e_k der k-te Einheitsvektor bezeichnet, so selektiert die Störung $d = e_k$ die k-te Spalte der Jacobi-Matrix. Um J vollständig durch Gleichung 2.6 für $d = e_k$ mit $k = 1...n$ zu berechnen, sind insgesamt $n + 1$ Funktionsauswertungen von f nötig. Durch Färben der Jacobi-Matrix unter Ausnutzung der Dünnbesetztheit kann die Anzahl der rechenaufwändigen f-Auswertung reduziert werden [GMP05, Kap. 1.2]. Hierbei werden die y-Komponenten derart gruppiert, dass die zugehörigen Spalten in der Jacobi-Matrix innerhalb eine Gruppe orthogonal sind. Ziel ist es, möglichst wenig Gruppen zu benötigen, da die Anzahl der Gruppen direkt mit der Anzahl der f-Auswertungen korreliert. Da das Färbeproblem aber NP-schwer ist [GMP05, Kap. 1.2], muss auf Heuristiken zurückgegriffen werden. Als Ergebnis ergibt sich eine sogenannte Seedmatrix $S \in \{0,1\}^{n \times c}$, wobei c der Anzahl der Gruppen (= Farben) entspricht. Formal lassen sich sämtliche Elemente der Jacobi-Matrix durch das Produkt $J \cdot S$ bestimmen, siehe auch Abbildung 2.2. Praktisch wird Gleichung 2.6 pro Spalte von S genutzt. Anschließend wird die Kompaktform der Jacobi-Matrix wieder in ihre ursprüngliche Form gebracht.

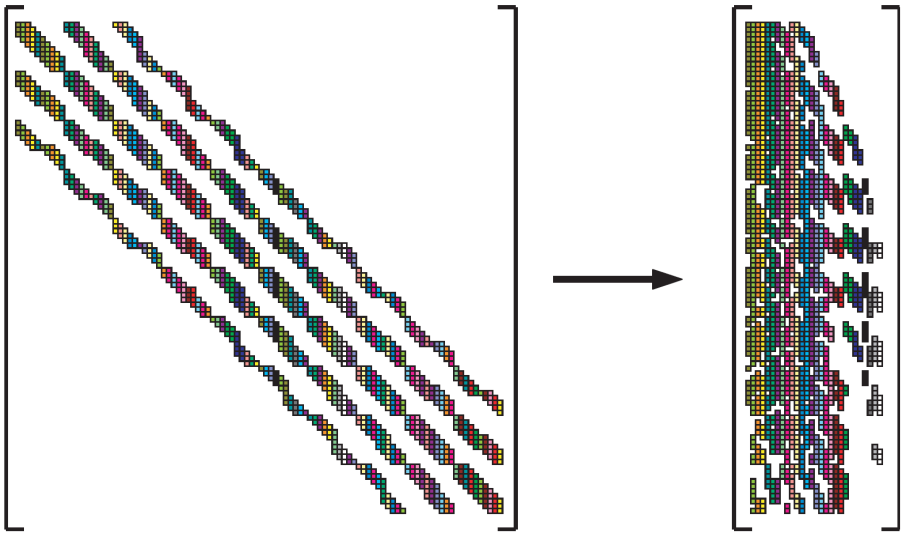


Abbildung 2.2: Jacobi-Matrix und ihre komprimierte Form $J \cdot S$. [GMP05, Fig 1.2]

Um Rechenzeit zu sparen, wird die Färbung der Jacobi-Matrix in ATHLET nicht wie beschrieben auf Elementebene, sondern auf Blockebene durchgeführt. Die Blockmatrix ist in der Regel um einen Faktor 5 kleiner als die Elementmatrix und produziert bei tatsächlich dichtbesetzten Blöcken ein Ergebnis von gleicher Qualität. Die Information, welche Blockspalten gemeinsam mit jeweils einer Störung erzeugt werden

können, muss nun für die f-Auswertung der Differentialgleichung auf Elementspalten übertragen werden. Wenn alle Blockspalten einer Farbe die gleiche Anzahl an Elementspalten besitzen, werden jeweils ihre k-ten Elementspalten zu einen gemeinsamen Störungsverktor zusammengelegt. Wenn Blockspalten von unterschiedlicher Größe der gleichen Farbe zugeordnet werden, so gibt die Elementanzahl der größten Blockspalte die Zahl der Störungen vor [GMP05].

2.1.3 Lineares Gleichungssystem

Neben der Generierung der Jacobi-Matrix ist das Lösen des linearen Gleichungssystems eine der teuersten Operationen im Simulationsverlauf. ATHLET verwendet dazu die Gaußelimination auf Blockebene und zerlegt damit die Systemmatrix in eine untere linke und eine obere rechte Dreiecksmatrix (LR-Zerlegung). Sie gehört zu den direkten Lösungsverfahren und ist für kleine bis mittelgroße Systemen die Standardvorgehensweise.

Eine Eigenschaft direkter Lösungsverfahren für dünnbesetzte Gleichungssysteme ist das Erzeugen von Fill-In [Aus+16, Kap. 6.3.3][Hof08]. Das sind implizite Nulleinträge in der Systemmatrix, die aufgrund der Zeilenelimination des Gaußverfahrens zu expliziten Nichtnulleinträgen werden. Je nach Matrixstruktur kann dadurch eine dünnbesetzte Matrix zu einer vollbesetzten Matrix werden. Um dies zu vermeiden, wird vor Anwendung des direkten Lösungsverfahrens ein Algorithmus eingesetzt, der den auftretenden Fill-In durch geschickte Zeilen- und Spaltenpermutation reduziert.

Der Lösungsprozess für dünnbesetzte Gleichungssysteme wird üblicherweise in mehrere Phasen eingeteilt. Im ersten Schritt erfolgt die Analysephase und symbolische Faktorisierung, die anhand der Matrixstruktur den später auftretenden Fill-In abschätzt und minimiert. Die symbolische Faktorisierung erstellt die Speicherstrukturen für Fill-In und LR-Zerlegung. Im Anschluss folgt die numerische Faktorisierung, welche die LR-Zerlegung mit den tatsächlichen Werten durchführt. Im letzten Schritt wird die Lösung x des Gleichungssystems durch Vorwärts- und Rückwärtssubstitution der rechten Seite b berechnet.

In Abhängigkeit der aufeinanderfolgenden Gleichungssysteme können die Ergebnisse der Zwischenschritte, wie in Abbildung 2.3 abgebildet, wiederverwendet werden [Bal+16a, Kap. 4.2].
Die Analysephase und die symbolische Faktorisierung müssen nur dann neu berechnet werden, wenn das neue Gleichungssystem eine neue Nichtnullstruktur aufweist. Das wird in ATHLET regelmäßig genutzt, da sich die Matrixstruktur nur selten bei einem Neustart ändert.
Bleibt die Matrix bei zwei aufeinanderfolgenden Gleichungssystemen komplett wertekonstant, muss für die Lösung nur der b-Vektor vorwärts und rückwärts substitutiert werden. Dies wird vorallem während des Extrapolationsverfahren ausgenutzt.

Innerhalb einer Reihe von Teilschritten ist dort die Schrittweite Δt konstant, wodurch auch die Systemmatrix zwischen den Gleichungssystemen gleich bleibt.

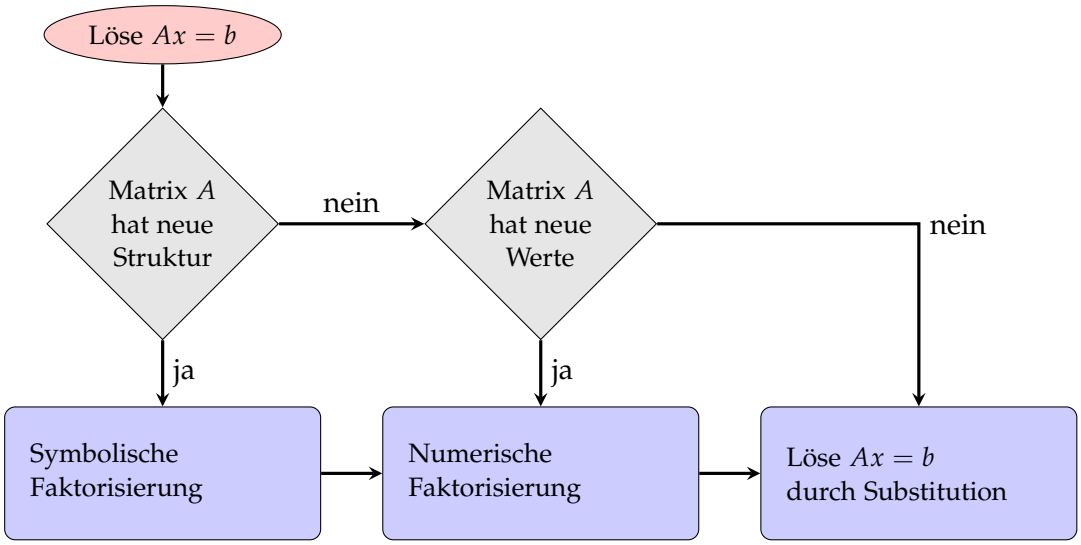


Abbildung 2.3: Lösen des linearen Gleichungssystems durch LR-Zerlegung

2.2 Lösung und Implementierung

Die Zeitintegration wird durch das FEBE-Modul (Forward-Euler, Backward-Euler) durchgeführt. Es berechnet die Zeitschritte durch Anwendung des Extrapolationsverfahrens und verwendet dessen Ergebnisse zur aktuellen Fehlerabzuschätzung. Abhängig davon, ob dieser den Fehlertoleranzen des Benutzers genügt, wird die Berechnung mit einem kleineren Zeitschritt wiederholt oder mit einer ggf. angepassten Zeitschrittweite fortgeführt. Zusätzlich prüft das FEBE-Modul die Aktualität der Jacobi-Matrix und veranlasst bei Bedarf eine volle oder partielle Neuberechnung. Zum Lösen eines Gleichungssystems oder zum Erstellen der Jacobi-Matrix wird das FTRIX-Modul verwendet, dessen Funktionsumfang in Abbildung 2.4 dargestellt ist [Aus+16, Kap. 6.1].

In der Startphase erstellt ATHLET anhand der Netzwerktopologie des Modells die Blockstruktur der Jacobi-Matrix. Sie ist aufgrund der symmetrischen Abhängigkeiten der Netzwerkelemente auch selbst strukturymmetrisch und wird in den Arrays ITAB2, ITAB5 und ITAB7 neben anderen abgeleiteten Informationen abgespeichert. Mithilfe des Blockzeilenindex kann über ITAB7 der Blockindex und über ITAB2 die Zahl der Elemente der jeweiligen Blockzeile ermittelt werden. ITAB5 ordnet jedem Blockindex einen Blockspaltenindex zu. Die Datenstruktur von ITAB5 und ITAB7

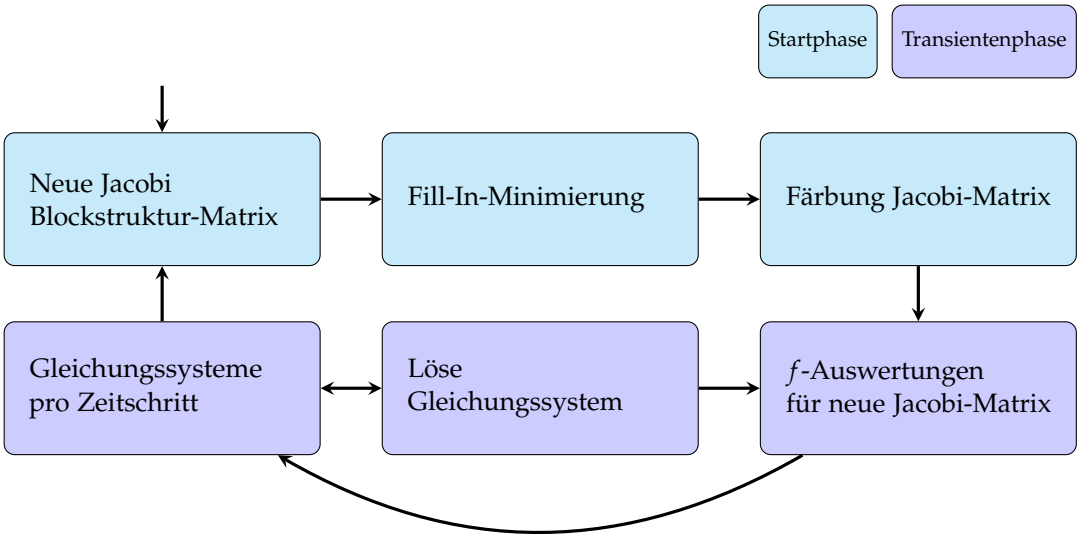


Abbildung 2.4: Funktionsumfang und Ablauf im FTRIX-Modul

entsprechen dem für dünnbesetzte Matrizen gängigem Compressed Row Storage (CSR) Format [Bar+94, Kap 4.3.1] auf Blockebene.
Im nächsten Schritt wird im FTRIX-Modul eine symmetrische Zeilen- und Spaltenpermutation der Jacobi-Matrix mittels Minimum Deficiency Algorithmus [TW67][Hof08] berechnet, sodass bei der späteren LR-Zerlegung möglichst wenig Fill-In entsteht. Zudem wird der für den Fill-In benötigte Speicher angelegt. Nun wird die Jacobi-Matrix nach dem SL (Smallest Last) Algorithmus [GMP05, Kap 3.5.1] für die komprimierte f-Auswertung gefärbt. Da alle bisherigen Schritte in FTRIX nur auf der Blockmatrixstruktur beruhen, können die Ergebnisse in der Regel während der gesamten Simulation beibehalten werden.
Phänomene, wie beispielsweise die 2-Phasenströmung und dadurch wandernde Pegelstände über verschiedene Kontrollvolumina, sorgen dafür, dass einzelne Gleichungen im Verlauf der Simulation an und ausgeschaltet werden können. Eine implizite Berücksichtigung davon würde zu einer ständigen Veränderung der Matrixstruktur und Wiederholung der Startphase führen. Deshalb werden bei der Erstellung der Blockmatrix alle Gleichungen als aktiv betrachtet. Das sog. TOP-Array enthält die Informationen darüber, welche Gleichungen tatsächlich eingeschaltet sind und in der Berechnung berücksichtigt werden müssen. Wird eine Gleichung mit dem Index i ausgeschaltet, werden alle expliziten Stellen der Spalte und Zeile i der Jacobi-Matrix mit expliziten Nullen gefüllt, wodurch die Struktur an sich unberührt bleibt. Nur in seltenen Fällen, wenn sich während der Simulation die Größe eines Blocks und damit auch die Blockstruktur ändert, wird die Simulation neugestartet und die vorbereitenden Schritte 1-3 aus der Startphase müssen erneut durchgeführt werden.

Nach Abschluss der Startphase beginnt die Transientenphase der Simulation, die durch das FEBE-Modul gesteuert wird. Zuerst werden anhand der Seedmatrix alle nötigen Funktionsauswertungen der Differentialgleichung durchgeführt und damit die Einträge der aktuellen Jacobi-Matrix generiert. Zusammen mit dem von der Zeitschrittsteuerung vorgegeben aktuellen Zeitschrittweite und dem Extrapolationsverfahren ergeben sich daraus mehrere Gleichungssysteme, die sukzessiv in FTRIX gelöst werden. Erfüllt das Ergebnis des Zeitschritts die Gütekriterien der Fehlersteuerung, wird mit der Berechnung des nächsten Zeitschritts fortgefahren. Andernfalls verwirft das FEBE-Modul den berechneten Zeitschritt und wiederholt diesen mit einer kleineren Schrittweite und oder veranlasst die Aktualisierung der Jacobi-Matrix. Abhängig von den Änderungen im System kann die Jacobi-Matrix vollständig oder partiell erneuert werden. Im partiellen Fall werden nur ausgewählte Spalten und Zeilen aktualisiert. Dies wird beispielsweise dann genutzt, wenn die Jacobi-Matrix vom FEBE-Modul zwar als aktuell bewertet wird, aber zeitgleich ein paar Gleichungen per TOP-Array ein- oder ausgeschaltet werden. In dem Fall müssen nur die betroffenen Gleichungen aktualisiert werden. Die restlichen Matrixelemente können unverändert bestehen bleiben.

2.3 Modelle und Parameter

Im folgenden Abschnitt werden die zur späteren Analyse und Bewertung genutzten Modelle und Optionen vorgestellt.

Kalinin-3

In einem neuem Ansatz soll der Reaktorkessel des WWE-Reaktors Kalinin 3 (K3) mithilfe einer detaillierten adaptiven Diskretisierung von Brennelementen simuliert werden. Während der Generierung des Datensatzes wird festgelegt, wie viele der maximal 331 Brennelemente mit dieser komplexen Methode später berechnet werden sollen. Zur Übersicht werden die Kalinin-3 Modelle im Folgenden als K3-n bezeichnet, wobei n für die Anzahl der feinaufgelösten Brennelemente steht. Die Berechnung des kleinsten Datensatzes mit 2 Brennelementen K3-2, verursacht in ATHLET bereits eine verhältnismäßig lange Rechenzeit, dessen mögliche Gründe im Anschluss analysiert werden. [NVP11] [Jac+17]

PWR-3D

Das PWR-3D Modell simuliert den Reaktorkessel eines deutschen Druckwasserreaktors. Dabei werden Thermofluid- und Wärmeleitobjekte samt Heizstäbe mit einer feinauflösenden 3-dimensionalen Topologie verknüpft. Dieses Modell entspricht

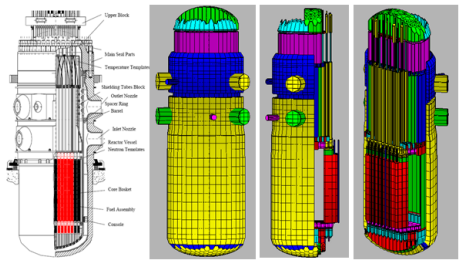


Abbildung 2.5: Modell des Kalinin-3 Reaktorkessels [Jac+17, Fig. 1]

von der Größenordnung und der Komplexität den Modellen, die typischerweise in ATHLET simuliert werden. Damit dient es als repräsentatives Referenzmodell und als Maßstab zum Vergleich mit dem größeren K3 Modell. [Ler+16, Kap. 9.10]

Netzwerk

Es gibt in ATHLET die Möglichkeit durch zusätzliche Verbindungen zwischen den Netzwerkelementen den gemischten Massestrom besser simulieren zu können. Durch die weiteren Abhängigkeiten nimmt die Anzahl der Einträge in der dünnbesetzten Systemmatrix zu. Diese Option wird durch das Setzen des Parameters $i2mftrx = 1$ im Datensatz nur bei Bedarf aktiviert, da die Verbesserung der Ergebnisse nicht immer den zusätzlichen und teilweise erheblichen Rechenaufwand rechtfertigen [Ler+16, S. 3-49]. Um die Eingabedatensätze mit eingeschaltetem $i2mftrx$ Parameter von den anderen unterscheiden zu können, enden diese Modellbezeichnungen mit dem Symbol „+“.

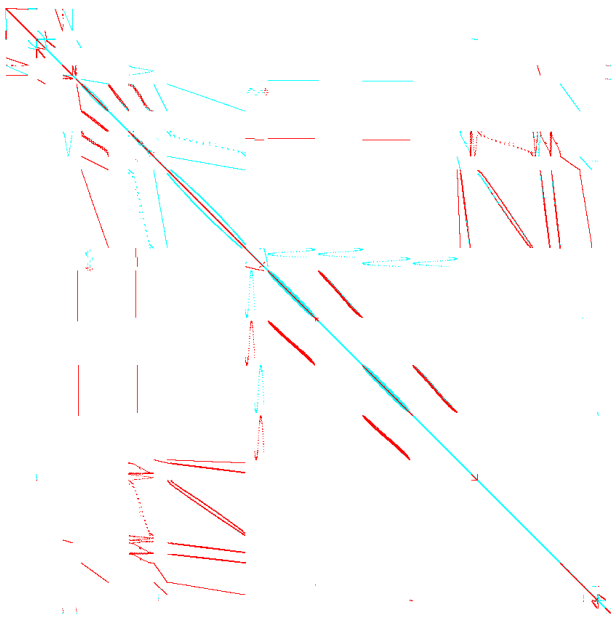


Abbildung 2.6: K3-2 Matrixstruktur mit $i2mftrx = 0$

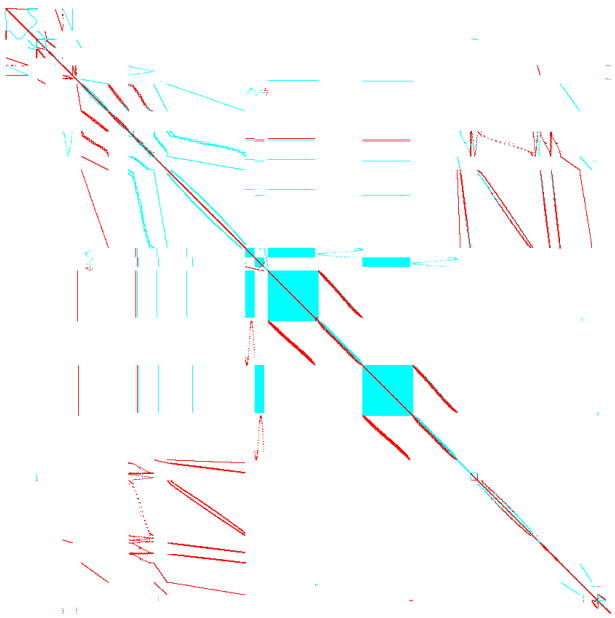


Abbildung 2.7: K3-2+ Matrixstruktur mit $i2mftrx = 1$

3 Anforderungen und Lösungskonzept

In dieses Kapitel sollen zuerst die rechenaufwändigsten Programmteile von ATHLET identifiziert werden. Dies erfolgt anhand einer Tabelle mit aufgeschlüsselten Berechnungszeiten der vorgestellten Modelle. Zusätzlich werden Aspekte des ATHLET-Codes bezüglich seiner Erweiterbarkeit untersucht und bewertet. Anschließend werden daraus Ziele für Verbesserungen und eine Anforderungsliste aus Rahmenbedingungen zusammengefasst, die es für die Umsetzung der Verbesserungen einzuhalten gilt. Schließlich werden auf dieser Basis das allgemeine Lösungskonzept und theoretische Aspekte für dessen Umsetzung vorgestellt.

3.1 Analyse

Um einen Überblick über die Leistung von ATHLET zu bekommen, werden PWR-3D und K3 Modell mit unterschiedlichen Parametern auf dem Linux-Cluster der GRS berechnet. Er verfügt über 41 Rechenknoten mit jeweils 2 Sockel zu je 10 Prozessorkernen, sowohl zwischen 128GB und 256GB Arbeitsspeicher und wird als Testsystem für alle Messungen dieser Arbeit verwendet.

In Tabelle 3.1 sind die verschiedenen Konfigurationen von PWR-3D und K3, sowie deren Simulationszeiten aufgelistet. Dazu gehört die Anzahl der Dimensionen und Nichtnulleinträge der Systemmatrix, sowie der Speicherverbrauch zum Zeitpunkt des ersten berechneten Zeitschritts. Im darauf folgenden Bereich ist jeweils die Anzahl der Zeitschritte, die zu lösenden Gleichungssysteme und die partiellen sowie vollen Berechnungen der Jacobi-Matrix angegeben. Darunter befindet sich die Berechnungsdauer zur Färbung der Jacobi-Matrix und die Gesamtzeit des Algorithmus zur Fill-In-Reduktion samt Allokierung der Fill-In-Datenstrukturen. Die Startphase fasst diese und alle anderen Berechnungen bis zu Beginn der Transientenphase zusammen, in der alle Zeitschritte bis zum Erreichen der Zielzeit durchgeführt werden. Ihr Hauptanteil besteht aus dem Lösen der Gleichungssysteme („Zeit GLS“) und der Generierung der Jacobi-Matrix inklusive Auswertungen der f-Funktion („Zeit Jacobi“). Zuletzt ist deren relativer Anteil aus der Transientenphase angegeben.

Tabelle 3.1: Rechenergebnisse von ATHLET

	PWR-3D	PWR-3D+	K3-2	K3-2+	K3-7
Dimension	6.009	6.009	128.673	128.673	258.371
Anzahl Einträge	143.469	151.961	2.852.179	5.240.891	5.865.471
Speicher in *GB*	0,21	0,22	4,0	-	10,0
Anzahl Zeitschritte	14.196	14.656	-	-	-
Anzahl GLS	145.472	145.926	-	-	-
Anzahl Jacobi Partiell	7.381	7.660	-	-	-
Anzahl Jacobi Voll	5.702	7.372	-	-	-
Fill-In in *s*	<1	8	37.080	>691.200	99.580
Färbung in *s*	<1	1	1.674	-	6.589
Startphase in *s*	5	14	39.121	-	106.945
GLS in *s*	3.797	50.372	122.424 · 65 *	-	197.679 · 165 *
Jacobi in *s*	7.031	8.213	525 · 65 *	-	1.745 · 165 *
Anteil GLS	34%	86%	99,6%	-	99,1%
Anteil Jacobi	64%	14%	0,4%	-	0,9%
Tansientenphase in *s*	11.005	58.753	122.969 · 65 *	-	199.472 · 165 *
Gesamt in *s*	11.010	58.767	8.032.106 *	-	33.019.825 *

* extrapolierte Werte

Geschwindigkeit

Die Rechenzeiten steigen zunehmend mit der Dimension und der Anzahl an Nichtnulleinträgen der Gleichungssysteme an. In der Gegenüberstellung des PWR-3D und des PWR-3D+ Modells, lässt sich erkennen, dass bei gleicher Anzahl an Unbekannten und nur 6% mehr Nichtnulleinträgen, sich die Gesamtrechenzeit um den Faktor 5 vervielfacht. Grund dafür ist ein massiver Anstieg in der Rechenzeit zur Lösung der Gleichungssysteme, dessen relativer Anteil aus der Tansientenphase sich von 34% auf 86% verschiebt. Da die beiden Gleichungssysteme nahezu die selbe Größe haben, ist die Ursache dafür wahrscheinlich eine ungünstigere Matrixstruktur. Diese verursacht deutlich mehr Fill-In, womit ATHLETs Löser nicht mehr optimal umgehen kann.
Noch deutlicher wird dies im Vergleich zwischen dem PWR-3D und dem K3-2 Datensatz. Die Systemmatrix im K3-2 Beispiel ist bezüglich ihrer Größe und Anzahl an Nichtnullelementen um den Faktor 20-mal größer. Dies sorgt schon in der Startphase für einen unverhältnismäßig größeren Rechenbedarf zur Berechnung der Fill-In-Reduktion und der Jacobi-Färbung. In der Transientenphase beträgt der Anteil zum Lösen der Gleichungssysteme nun mehr als 99%. Die Angaben über die Dauer der Transientenphase sind im K3-2 und im K3-7 Datensatz extrapoliert und daher nur als grobe Abschätzung zu verstehen. Daraus ergibt sich eine geschätze Gesamtrechenzeit von jeweils 3 bzw. 12 Monaten, wodurch es praktisch kaum noch möglich ist, mit diesen Modellen zu arbeiten. Werden die Daten und der Mehraufwand mit dem K3-2+ Modell verglichen, so wird klar, dass dieses überhaupt nicht mehr mit ATHLET berechenbar ist. Alleine die Reduktion des Fill-Ins konnte für das K3-2+ Modell nicht innerhalb von 8 Tagen berechnet werden.

Speicher

Neben der Rechenzeit ist im K3 Modell auch ein hoher Gesamtspeicherverbrauch, der auch mit der Anzahl der Brennelementen skaliert, festzustellen. Langfristiges Ziel ist es, das K3 Modell mit allen Brennelementen simulieren zu können. Da der Speicherverbrauch bereits mit 18 Brennelementen bei über 32GB liegt, wird es für K3-331 in jedem Fall nötig sein, den Speicher über einen Clusterknoten hinaus verteilen zu können. Andernfalls wird das K3-331 Modell voraussichtlich nicht komplett innerhalb des Arbeitsspeichers eines Systems berechnet werden können.

Programmstruktur

ATHLET besitzt historisch gewachsen teilweise unübersichtliche und nicht lückenlos dokumentierte Programmstrukturen. Es werden zwar FORTRAN-Module eingesetzt,

diesen fehlt es aber an klar definierten Schnittstellen darüber, welche Informationen eingehend und ausgehend nach dem Blackbox Prinzip gebraucht werden. Stattdessen werden Informationen teilweise über parametrisierte Funktionsaufrufe und teilweise über die Verwendung von globalen Variablen anderer Module ausgetauscht. Weiterhin kommt der häufige Einsatz von goto-Spunganweisungen zur Steuerung des Programmablaufs erschwerend hinzu und sorgt für unübersichtlichen Spaghetti-Code [Bro+98, Kap. 5]. Zudem sind Lösungsmethoden auf unterer Ebene teilweise direkt ohne Abstraktion mit Kontext aus höheren Implementierungsebenen vermischt. Die fehlende Abstraktion bei einem Projekt von dieser Größe und Komplexität führt dazu, dass sämtliche Änderungen im Code nur extrem schwer durchführbar sind [IH04, Kap 3.1]. So kann zwischen hoher mathematischer Semantik und unterer tatsächlicher Implementierung nicht ohne Weiteres differenziert werden. Beispielsweise ist es damit nicht möglich, nur die Mathematik zu verändern oder andere etablierte Verfahren für die zugrunde liegende Implementierung zu verwenden. Neue Entwicklungen und Implementierungen moderner Lösungsverfahren können aufgrund der fehlenden Schnittstellen und semantischen Trennung nur langsam und mit großem Aufwand in ATHLET eingebettet werden.
Ein weiterer Nachteil ist zudem, dass die Implementierungen einer eigentlich allgemeinen Methoden nicht für andere Programmteile wiederverwendbar ist, da keine abstrakten Datentypen verwendet werden. Beispielsweise werden in ATHLET während der Startphase mehrere dünnbesetzte Gleichungssysteme erstellt und gelöst. Dies geschieht allerdings nicht durch den umfangreichen Löser aus dem FTRIX-Modul für dünnbesetzte Systeme, sondern durch eine weitere nicht optimierte Implementierung, die die Gleichungssysteme auf dichtbesetzter Basis löst und je nach Modell unnötig Rechenzeit verbraucht. Ein weiteres Beispiel ist das Färben der Jacobi-Matrix. Dies geschieht, wie beschrieben, während der Startphase im FTRIX-Modul. Aufgrund der fehlenden Abstraktion kann diese Methode als Eingabe nur die aktuelle, volle Jacobi-Matrix verarbeiten. So ist es nicht möglich, alternative Färbungen, die für die partielle Neuberechnung der Jacobi-Matrix hilfreich wären, ebenfalls mit dieser Funktion durchzuführen. Als Ergebnis führt ATHLET die partielle Aktualisierung der Jacobi-Matrix ohne komprimierte f-Auswertung durch. Dies ist ineffizient und könnte eigentlich durch die Wiederverwendung des vorhandenen Färbealgorithmus behoben werden..

3.2 Anforderungen

Ziel ist es, die Rechenzeit der großen Modelle so zu reduzieren, dass sie innerhalb eines vernünftigen Zeitrahmens berechenbar sind. Dazu sollen die gefundenen kritischen Aspekte bezüglich Geschwindigkeit, Speicher und Programmstruktur verbessert werden.

Das größte Potential zur Geschwindigkeitssteigerung liegt in den Programmteilen, die die größten relativen Anteile an der Gesamtlaufzeit haben. Aufgrund der unübersichtlichen Programmstruktur, ist es schwer zu analysieren, in wie weit diese Programmteile noch optimiert werden können. Abschnitt 3.1 lässt vermuten, dass die eingesetzten Algorithmen nicht optimal arbeiten. Wie groß der Leistungsunterschied zu einem optimalen Vefahren ist, kann allerdings im Vorfeld nicht exakt abgeschätzt werden. Abgesehen von einer möglichen Leistungssteigerung profitiert ATHLET durch eine verbesserte Programmstruktur und erleichtert zukünftige Entwicklungen.

Mithilfe der folgenden Arbeitspunkte soll die Umsetzung geschehen.

Etablierte Algorithmen

Die benötigte Rechenzeit für das Lösen der GLS, das Färben der Jacobi-Matrix und der Fill-In-Reduktion machen den Großteil der Gesamtrechenzeit aus. Mit Ausnahmen des Erstellens der Jacobi-Matrix, nimmt der relative Anteil an der benötigten Rechenzeit bei diesen Programmteilen stark mit der Größe der gezeigten Modelle zu. Dies liegt vermutlich an einer nicht optimalen Implementierung der entsprechenden Algorithmen, was aber mit anderen kleinen bis mittelgroßen Modellen nicht so stark ins Gewicht fällt. Diese Aufgaben sind alle gut erforschte Problemstellungen, für die es bereits umfangreiche und sehr effektive Lösungsmethoden gibt. Daher ist es sinnvoll für die genannten Probleme weitgehend auf bestehende etablierte Lösungsverfahren und Datenstrukturen über externe Bibliotheken zurückzugreifen. Durch diese Anforderung soll gewährleistet werden, dass die Rechenzeit für größere Modelle nur so stark ansteigt, wie es mit dem aktuellen Stand der Technik auch nötig ist.

Verteilte Parallelisierung

Neben der Optimierung des sequentiellen Codes kann eine Parallelisierung der rechenaufwändigen Codesegmente für einen weiteren Geschwindigkeitszuwachs sorgen. Bei der verteilten Parallelität werden große rechenintensive Aufgaben, soweit möglich, in einzelne Teilaufgaben zerlegt und an mehrere eigenständige Prozesse verteilt. Diese können dann auf verschiedenen Prozessoren parallel ausgeführt werden. Da ein Teilproblem üblicherweise nur aus einem Bruchteil des Speichers und des Arbeitsumfangs des Gesamtproblems besteht, kann durch eine parallele Abarbeitung eine Beschleunigung der Gesamtrechnung und zeitgleich eine Reduktion des Speicherverbrauchs pro Prozess erreicht werden. Im Idealfall nimmt die Geschwindigkeit mit dem Faktor der verwendeten Prozessoren linear zu. Dies ist für Probleme aus der Praxis allerdings kaum realisierbar, da der Geschwindigkeitszuwachs in parallelen Programmen stark durch einen nicht parallelisierbaren Anteil

beschränkt ist (Amdahlsches Gesetz [HW10, Kap. 5.3.3]). Zusätzlich haben weitere Faktoren wie Fixkosten der Kommunikation zwischen den Prozessen und eine nicht gleichmäßige Lastenverteilung einen weiteren negativen Einfluss auf den erreichbaren Leistungszuwachs. Da neben der Rechenzeit auch der Speicherbedarf der großen Modelle zunimmt, beschränkt sich diese Arbeit auf die verteilte Parallelisierung, damit der Speicherverbrauch pro Rechenknoten reduzieren werden kann. Dies ist mit alternativen Parallelisierungskonzepten wie dem SharedMemory OpenMP nicht möglich und wird deshalb zurückgestellt [HW10, Kap. 6].

Abstrakter Löser

Wie beschrieben ist es für eine saubere Implementierung großer Projekte wichtig, dass sie in unterschiedliche Abstraktionsebenen ihrer Funktionalität und Daten aufgeteilt werden. Diese Abstraktion soll auch bei der Einführung der geforderten Lösungsverfahren umgesetzt werden. Daher muss bei der Implementierung darauf geachtet werden, dass sämtlicher Code, in dem mathematische Probleme von ATHLET gelöst werden, von Funktionen und Datenobjekten externer Bibliotheken komplett abstrahiert sind. Sie werden somit in einer unabhängigen Abstraktionsschicht implementiert, die im Folgenden als abstrakter Löser (AL) bezeichnet wird. Dadurch ist eine direkte Abhängigkeit zu externen Bibliotheken unterbunden und es kann immer klar zwischen ATHLET- und Bibliotheks-spezifischem Code unterschieden werden.

Für die Erweiterung müssen aufgrund des bestehenden ATHLET-Codes folgende Rahmenbedingungen besonders berücksichtigt werden:

- Wenig Änderung im ATHLETs Code und Build Management
- Multiplattformkompatibilität: Windows und Linux
- Lizenzbedingungen externer Bibliotheken

Alle Änderungen in ATHLET haben eine große Auswirkung auf den Code. Wie bei allen großen Projekten ist es wünschenswert, dass die Umsetzung, auch von umfangreichen Änderungen, mit möglichst wenigen Anpassungen im bestehenden Code realisiert wird. Damit können die Veränderungen im Code, die zuerst in einem parallelen Projektzweig implementiert und getestet werden, nach erfolgreichem Abschluss der Entwicklung leichter in den Hauptcode portiert werden.

Zusätzlich wird damit die Zahl der Schnittstellen minimiert, was der Übersicht und Flexibilität, sowie einer möglichen Fehlersuche zu Gute kommt.

Des Weiteren ist es erforderlich, dass möglichst keine Änderungen in ATHLETs Build Management nötig sind. ATHLET benutzt automatisierte Integrationstests, die kontinuierlich überprüfen, ob die aktuelle Version kompilierbar ist. Anschließend werden mit der kompilierten Version Testsimulationen durchführt und deren Ergebnisse verifiziert. Somit muss jede Änderung, die die Kompilierung betrifft, sorgfältig überprüft werden und auch die bestehenden Integrationstests dafür angepasst werden. Daher soll versucht werden, alle nötigen Bibliotheken ohne Änderungen in ATHLETs Build Management einzubinden.

Der neue Code sollte auf Windows Laptops mit nur zwei Prozessorkernen bis hin zum Linux Rechencluster mit mehreren Knoten lauffähig sein. Dem entsprechend müssen alle Änderungen im Code und die verwendeten externen Bibliotheken für Windows und Linux portabel sein.

Außerdem müssen auch etwaige Einschränkungen durch Lizenzbestimmungen der Bibliotheken berücksichtigt werden. Die Lizenzen einiger Bibliotheken erfordern es, dass alle damit verknüpften Programme ebenfalls unter die selbe Lizenz fallen. So müsste zum Beispiel bei der Verwendung einer Bibliothek, die unter GPL (General Public License) steht, der Quellcode von ATHLET ebenfalls unter GPL gestellt werden und damit der komplette Programmcode offengelegt werden. Die ist zurzeit nicht vorgesehen und schießt deshalb die Verwendung einer unter GPL stehenden Bibliothek aus.

3.3 Lösungskonzept

Als Lösungskonzept wird ATHLET mit einem dediziertem Prozeses, dem NuT-Prozess (Numerical Toolkit), gekoppelt. Die Kopplung erfolgt komplett über eine Art Server-Client Ansatz. Dabei schickt ATHLET als Server alle relevanten Informationen, die zum Lösen eines Problems erforderlich sind, an einen, oder im parallelen Fall an mehrere NuT-Prozesse, siehe Abbildung 3.1. Diese berechnen die geforderten Teilaufgaben von ATHLET und schicken anschließend die Ergebnisse zur Weiterverwendung zurück.

Der NuT-Prozess bietet die Plattform zur komfortablen parallelen Programmierung auf abstrakter mathematischer Ebene und stellt zugleich die Schnittstelle zu externen Bibliotheken dar.

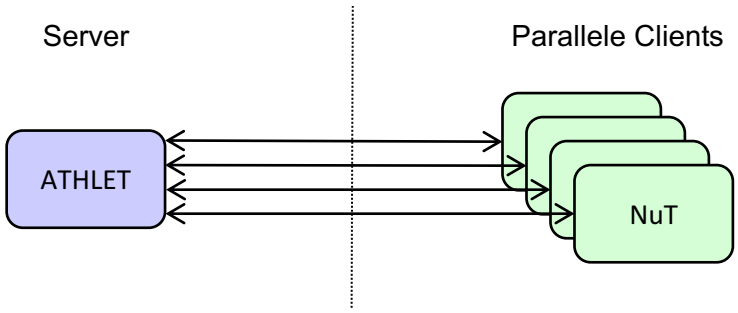


Abbildung 3.1: Ausgelagerte Parallelisierung per Server-Client Ansatz

Modularität

Der NuT-Prozess ist damit ein komplett abgeschlossenes und unabhängiges System, das nur über die vorher festgelegten Schnittstellen gesteuert werden kann. Dieser Ansatz bündelt die Kommunikation zwischen NuT-Prozessen und ATHLET auf wenige klare Schnittstellen. Das kommt der Forderung zugute, dass möglichst wenig Änderungen im ATHLET-Code durchgeführt werden.
Dieser modulare Ansatz ist sehr übersichtlich und ermöglicht es den Entwicklern, schneller alle Abhängigkeiten zu erfassen und bei Bedarf leicht Funktionen auszutauschen oder zu verändern [IH04, Kap 3.1].

Verteilte Parallelität

Neben der modularen Abgrenzung wird dadurch auch die verteilte Parallelität umgesetzt. Statt die Arbeit nur an einen Client weiterzuleiten, wird sie an mehrere NuT-Prozesse gleichzeitig verteilt. Davon berechnet jeder nur einen kleinen Anteil des Gesamtproblems, wodurch die Rechenzeit reduziert werden kann. Da in diesem Ansatz die Daten über das Netzwerk ausgetauscht werden, können sich die Clients auch auf unterschiedlichen Systemen befinden und erreichen somit die verteilte Speicherung der Daten. [HW10, Kap. 5.2.2]

Struktur und Abstraktion

Der NuT-Prozess ist in Kommunikations-, abstrakte Löser (AL)- und NuT-Schicht eingeteilt, siehe auch Abbildung 3.2.

Die Kommunikationsschicht ist die oberste Schicht. Sie enthält alle Schnittstelleninformationen zur Kommunikation mit ATHLET. Hier werden die Daten gesammelt und zur nächsten Ebene, der AL-Schicht weitergeleitet. Umgekehrt kümmert sie sich auch um die Rückgabe der in der AL-Schicht berechneten Ergebnisse an ATHLET.

Die darunterliegende AL-Schicht enthält an sich voneinander getrennte Module zum Bearbeiten unterschiedlicher Aufgabendomäne. Ein Aufgabendomän ist eine Zusammenfassung aller Aufgaben, die mithilfe einer gemeinsamen Datenbasis bearbeitet werden können. Neben dieser Datenbasis beinhaltet das Modul auch alle Funktionen, die zur Bearbeitung des Aufgabendomäns nötig sind. Die Implementierung erfolgt auf hoher Abstraktionsebene mithilfe der abstrakten Datentypen und Methoden, die durch die darunterliegende NuT-Schicht zu Verfügung gestellt werden.

Die untere Ebene ist die NuT-Schicht. Sie implementiert die abstrakten mathematischen Datentypen und darauf arbeitenden Funktionen für die AL-Schicht durch Verwendung einer etablierten Bibliothek. Die komplette Parallelität und die bibliotheksspezifischen Datentypen sind von der AL-Schicht abgekapselt.

In der NuT-Schicht werden alle abstrakten Datentypen für die AL-Schicht zur Verfügung gestellt. Für die richtige Abstraktion ist es wichtig, dass diese Datentypen außerhalb der NuT-Schicht nur als abstrakte Datentypen gesehen werden, ohne dass dessen Inhalte „sichtbar“sind. Daher werden alle relevanten Daten wie zum Beispiel das tatsächlich verwendete Objekt innerhalb der NuT-Schicht versteckt und auch nur dort abrufbar gemacht. Die NuT-Ebnene beinhaltet unterschiedliche Klassen von Methoden. Eine Klasse ist die Schnittstellen zum Umwandeln der abstrakten NuT-Objekte zu einfachen Datentypen und umgekehrt. Die andere Klasse stellt Rechenoperationen auf Basis der NuT-Objekte zu Verfügung.

Diese Aufteilung erfüllt alle drei Hauptziele. Durch die NuT-Schicht wird eine übersichtliche Plattform für die AL-Schicht geboten, mit der mathematische Probleme mithilfe von höheren abstrakten Methoden gelöst werden. Die tatsächlich verwendete Bibliothek und deren Lösungsmethoden, sowie die Umsetzung und Verteilung der Datenobjekte ist dem Entwickler komplett verborgen, sodass sich dieser auf die reine Problemlösung auf mathematischer Ebene konzentrieren kann. Mathematik und Implementierung lassen sich komplett unabhängig voneinander testen und anpassen. Ebenso wird die Abhängigkeit zur verwendeten Bibliothek stark reduziert, weil nur die untere NuT-Schicht auf ihr aufgebaut ist.

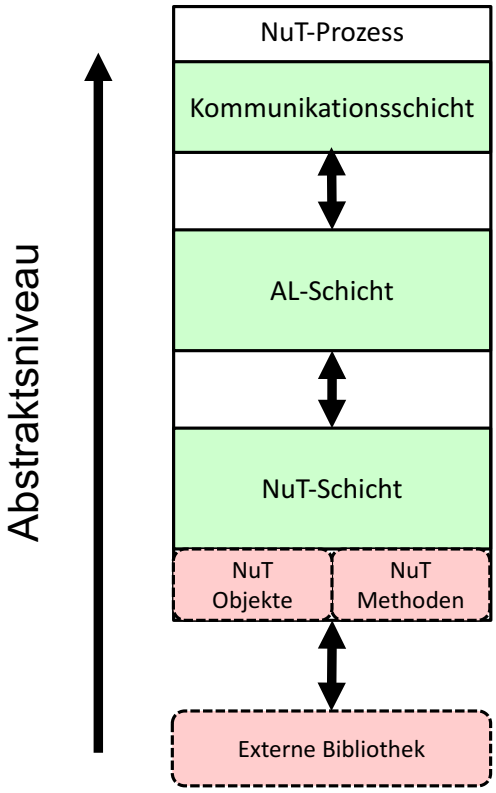


Abbildung 3.2: Abstraktionsebenen im NuT

4 Implementierung

Nachdem das theoretische Lösungskonzept erarbeitet ist, wird im folgenden Kapitel die konkrete Umsetzung beschrieben.

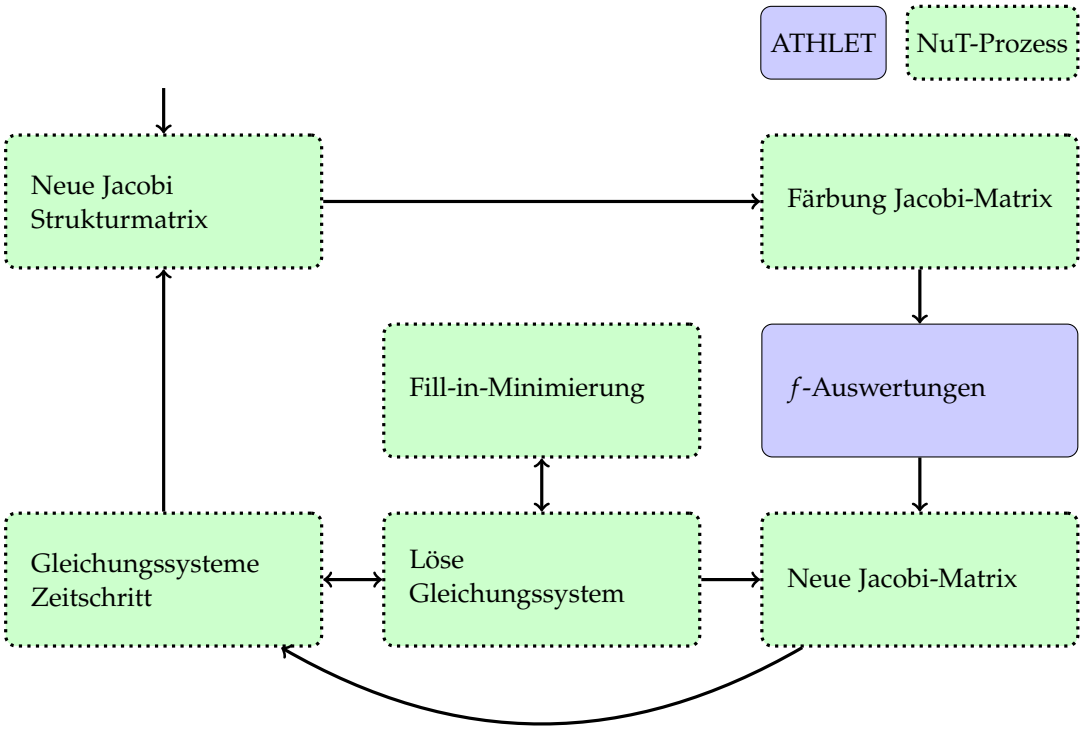


Abbildung 4.1: Aufteilung der Funktionen zwischen FTRIX und NuT

4.1 Funktionale Struktur und Ablauf

Es wird zuerst untersucht, welche Komponenten konkret von ATHLET in den NuT-Prozess ausgelagert werden. Im Abschnitt 3.1 wurden bereits die Bereiche ermittelt, die den größten Teil der Berechnungszeit ausmachen. Dazu gehört aus der Startphase die Fill-In-Minimierung und die Färbung der Jacobi-Matrix, die Auswertung

der Differentialgleichungsfunktion und das Lösen der Gleichungssysteme. Praktischerweise gehören all diese Komponenten mit Ausnahme der f-Auswertung zum FTRIX-Modul. Die f-Auswertung selber kann aufgrund ihrer hohen Komplexität und der Menge an benötigten Daten, die an den NuT Prozess übertragen werden müssten, nicht ausgelagert werden. Die neuen Zugehörigkeiten der Programmabschnitte sind in Abbildung 4.1 dargestellt. Abbildung 4.2 zeigt die zeitliche Abfolge durch die Steuerungsaufrufe des FEBE-Moduls und die vereinfachte Kommunikation mit den NuT-Prozessen.

4.1.1 Struktur der Jacobi-Matrix

Einstiegspunkt ist die Berechnung der Jacobi-Matrix. Soll das erste Mal eine Jacobi-Matrix berechnet werden, wird die Jacobi-Blockstruktur aus den ITAB-Arrays und das aktuelle TOP-Array mit Zustandsinformationen aller Gleichungen, an den NuT-Prozess übertragen und im entsprechenden Modul in der AL-Schicht gespeichert. Alle Berechnungen im NuT-Prozess finden auf Elementebene und nicht auf Blockebene statt. Solange die ITAB-Arrays in ATHLET unverändert bleiben, dienen sie im NuT-Prozess als komprimierte Basis zum Erstellen Datenstrukturen für die Elementmatrix. Nur wenn ATHLET einen Neustart aufgrund der Änderung der Blockstruktur durchführt, werden die ITAB-Arrays erneut aktualisiert und an den NuT-Prozess übertragen.

4.1.2 Färbung der Jacobi-Matrix

Nun wird daraus die Jacobi-Strukturmatrix zur anschließenden Färbung gebaut. Im Gegensatz zu ATHLET erfolgt die Färbung auf Elementebene und unter impliziter Berücksichtigung der Gleichungszustände. Das bedeutet, dass eine per TOP-Array abgeschaltete Gleichung nicht durch explizite Nullen in der Matrixstrukur berücksichtigt wird. Die daraus entstehende Seedmatrix beinhaltet nur die Elementspaltenindizes, die tatsächlich gestört werden müssen. Im Vergleich zu ATHLET, kann daher die Zahl der f-Auswertungen noch weiter reduziert werden. Das liegt daran, dass in ATHLET weder die Blockgröße, noch die abgeschalteten Gleichungen in der Färbung berücksichtigt werden und sich daher die Auslastung der f-Auswertung verringern kann. Um den Geschwindigkeitsvorteil durch die Reduktion der f-Auswertungen nicht wieder durch die Mehraufrufe des Färbealgorithmus zu verlieren, muss dieser deutlich schneller als die Implementierung von ATHLET sein.

4.1.3 Generierung der Jacobi-Matrix

Anschließend wird die Färbeinformation zurück an ATHLET übertragen, das damit die gestörten f-Auswertungen durchführt. Deren Ergebnisvektoren werden direkt nach jedem Funktionsaufruf spaltenweise an den NuT-Prozess zurückübertragen und in Form der Jacobi-Matrix verteilt gespeichert. An dieser Stelle ergeben sich zwei unterschiedliche Möglichkeiten. Die per TOP-Array abgeschalteten Gleichungen können in der Jacobi-Matrix explizit oder implizit berücksichtigt werden. Im expliziten Fall ergibt sich der Vorteil, dass sich die Nichtnullstruktur der Jacobi-Matrix durch Zuschalten oder Ausschalten von Gleichung nicht ändert. Dadurch kann die Speicherstruktur der Matrix beibehalten werden. Alle folgenden Gleichungssysteme können somit die einmalig durchgeführte Analysephase und die symbolische Faktorisierung wiederverwenden. Auf der anderen Seite ist der Aufwand für die zu berechnende LR-Zerlegung größer, da dieser von der Anzahl der explizit gespeicherten Einträge abhängig ist.
Mit der impliziten Berücksichtigung von abgeschalteten Gleichungen werden alle Phasen der LR-Zerlegung zunehmend beschleunigt. Dafür muss bei jeder Zustandsänderung einer Gleichung die komplette Datenstruktur der Jacobi-Matrix neu aufgebaut werden, weil Änderungen der dünnbesetzten CSR-Matrixstruktur im Nachhinein nicht effektiv möglich sind. Der Nachteil ist, dass Analysephase und symbolische Faktorisierung vergleichsweise öfters durchgeführt werden müssen. Ob sich die implizite oder explizite Berücksichtigung der Gleichungszustände lohnt, hängt davon ab, wie viele Gleichungen dynamisch abgeschaltet werden und wie oft sich die Zustände während der Simulation ändern.
Neben der vollständigen Erstellung der Jacobi-Matrix kann vom FEBE-Modul auch die partielle Aktualisierung angefordert werden. In dem Fall werden alle Werte der zuletzt berechneten Jacobi-Matrix beibehalten und eine Seedmatrix nur auf Basis der zur Aktualisierung angeforderten Spalten und Zeilen erstellt. Da, wie beschrieben, ATHLET zur partielle Berechnung der Jacobi-Matrix keine Färbung durchführt, können damit weitere f-Auswertungen eingespart werden.

4.1.4 Lösen der Gleichungssysteme

Nachdem die Jacobi-Matrix erstellt und auf allen NuT-Prozessen verteilt gespeichert wurde, übermittelt ATHLET die aktuelle Zeitschrittweite und den b-Vektor zur Lösung des Gleichungssystems. Anschließend wird die mit dem aktuellen Zeitschritt gewichtete Einheitsmatrix auf die Jacobi-Matrix aufaddiert und damit zur Systemmatrix überführt. Danach wird mit ihr und dem b-Vektor das resultierende Gleichungssystem gelöst.

4.1.5 Fill-In-Reduktion

Die ursprünglich in ATHLET immer in der Startphase berechnete Fill-In-Reduktion wird erst jetzt optional durch das angewandte Lösungsverfahren berechnet. Das ist sinnvoll, da eine Permutation zur Fill-In-Reduktion nur für direkte, aber nicht für iterative Lösungsverfahren Vorteile bringt. Analog zur Färbung kann es dadurch nötig sein, dass die Fill-In-Minimierung während der Simulation bei jeder Strukturänderung in der Jacobi-Matrix berechnet werden muss. Im letzten Schritt wird die parallel vorliegende Lösung des Gleichungssystems auf einem Prozess gesammelt und an ATHLET zurückgegeben.

4.1.6 Zusammenfassung

Mit dieser neuen Löserstruktur werden die zeitkritischen Abschnitte über wenig Schnittstellen aus dem FTRIX-Modul in den NuT-Prozess verschoben und dort verteilt berechnet. Die ebenfalls rechenintensive f-Auswertung der Differentialgleichung kann zwar nicht auslagert werden, jedoch besteht aufgrund einer größeren Auswahl an Färbealgorithmen und der konsequenten Färbung, auch im partiellen Fall, die Chance, die Zahl der nötigen Funktionsauswertungen und damit die Rechenzeit zu reduzieren.

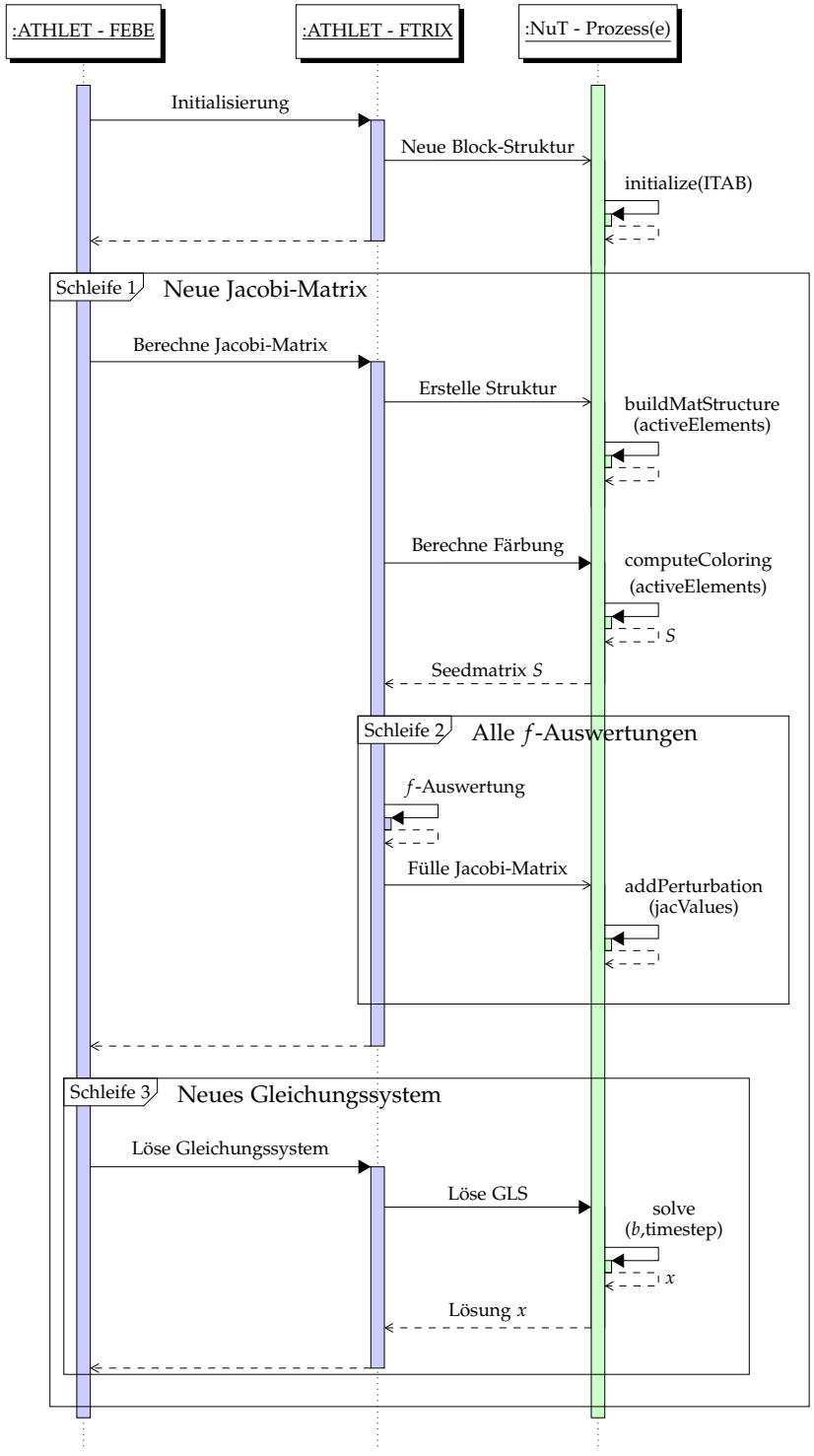


Abbildung 4.2: Kommunikation zwischen ATHLET und den NuT-Prozesse(n)

4.2 Bibliotheken

Im folgenden Abschnitt werden die zur Implementierung nötigen Bibliotheken MPI, PETSc und NuT-Plugin vorgestellt. Sie stellen zum einen die Basisschicht zur Implementierung des NuT und zum anderen ermöglichen sie die Kommunikation zwischen NuT- und ATHLET-Prozess.

4.2.1 PETSc

Als Grundlage für die Implementierung des NuT wird PETSc (Portable, Extensible Toolkit for Scientific Computation) verwendet. PETSc ist eine mächtige Bibliothek [Bal+97] auf Basis von LAPACK (Linear Algebra PACKage) [DG17] und kann Probleme aus dem Bereich des wissenschaftlichen Rechnens effektiv lösen. Es bietet viele Objekte und Funktionen auf unterschiedlichen mathematischen Ebenen. Der Umfang reicht von niedrigen Datentypen wie Indexmengen, Vektoren und Matrizen und der darauf angewandten linearen Algebra, über Krylov Unterraumverfahren und deren Vorkonditionierer zum Lösen von linearen Gleichungssystemen, bis hin zur obersten Schicht, die etablierte Methoden zum Lösen von nichtlinearen Gleichungssystemen und Zeitschrittverfahren anbietet [Bal+16a, Kap. 1]. Für fast alle Methoden und Datentypen existiert neben den sequentiellen auch parallele Impelmentierungen. Diese sind vom Entwickler weitgehend abgekapselt, sodass eine Umstellung vom sequentiellen Code zum parallelen Code nur wenige Änderungen erfordert. PETSc kümmert sich weitgehend selber um die verteilte Speicherung von parallelen Objekten, gibt dem Entwickler aber auch die Möglichkeit, die Parallelisierung manuell für seine Ansprüche zu optimieren.

Diese Arbeit beschränkt sich auf die Erneuerung von ATHLETs linearen Algebra Methoden und führt keine Veränderungen in der Numerik selbst durch. Dafür werden nur die unteren Schichten von PETSc bis hin zu den Krylov-Unterraumverfahren eingesetzt. Für die modulare Programmstruktur ist es wichtig, dass alle verwendeten Funktionalitäten und Objekte von PETSc nur für die Implementierung der NuT-Schicht eingesetzt werden und von der AL-Schicht und allen anderen Programmteile vollständig abgekapselt sind. Somit ist die Abhängigkeit zur PETSc Bibliothek auf die NuT-Schicht beschränkt.

Die aktuelle PETSc Version 3.7.5. steht unter der BSD-Lizenz. Sie enthält, im Gegensatz zur GPL, kein Copyleft und darf daher unter Beibehalt der Copyright-Vermerke ohne Offenlegung des ATHLET-Codes verwendet werden.

4.2.2 MPI

MPI (Message Passing Interface) ist ein Standard für den Nachrichtenaustausch zwischen mehreren Prozessen [For15a], die verteilt auf einem oder vielen Computern (Rechencluster) ausgeführt werden und kommt hauptsächlich für parallele Applikationen auf Supercomputern zum Einsatz. Es findet häufig Anwendung, um die Zusammenarbeit homogener Prozessgruppen zu koordinieren, kann aber auch für heterogene Prozessgruppen verwendet werden. Im Kontext von ATHLET wird MPI einerseits von PETSc als Grundlage zum verteilten Rechnen und Speichern genutzt und zum anderen für die Kommunikation zwischen ATHLET und den zugehörigen NuT-Prozessen.
Da es sich bei MPI nur um einen Standard handelt, kann abhängig von der Plattform und Präferenzen des Benutzers, die passende Impelmentierung aus MPICH, OpenMPI, IntelMPI oder Microsoft MPI [Msm] verwendet werden. Zu jeder MPI-Bibliothek gehört auch eine passende Laufzeitumgebung. Sie bietet die Tools, die zum Ausführen der parallelen Programme nötig sind. Dabei spezifiziert der Nutzer im einfachsten Fall die Anzahl der zu startenden Instanzen pro Prozess. Diese werden dann wie vorgegeben gestartet, wobei für jeden Prozess globale und private Umgebungsvariablen gesetzt werden. Damit sind die Prozesse voneinander unterscheidbar und es ermöglicht den Aufbau der Kommunikation zwischen ihnen.

Der Programmaufruf erfolgt mit dem Programm mpiexec. Ein exemplarischer Aufruf bestehend aus einem Verbund aus einem ATHLET-Prozess und vier NuT-Prozessen:

```
mpiexec -n 1 ./athlet : -n 4 ./nut-client
```

MPI-Prozesse können in Gruppen, sogenannte Kommunikatoren, eingeteilt werden. Jeder Prozess hat innerhalb dieser Gruppe seinen individuellen Rang. Die eindeutige Adressierung eines Prozesses innerhalb der gestarteten MPI-Prozesse erfolgt durch Angabe seines Kommunikators und dem zugehörigen Rang.

4.2.3 NuT-Plugin

ATHLET bietet die Möglichkeit zur Laufzeit externe Programmbibliotheken (.dll oder .so) als Plugins zu laden und zu verwenden. Da ATHLET zur Kompilierzeit frei von Abhängigkeiten externer Bibliotheken bleiben soll, wird die MPI Bibliothek nicht direkt in ATHLET verwendet, sondern komplett in das NuT-Plugin ausgelagert. Es wird nur bei Bedarf zur Laufzeit geladen und verhindert so direkte Linker-Abhängigkeiten. Das NuT-Plugin bildet das Gegenstück zur Kommunikationsschicht des NuT-Prozesses und bietet ebenso die Kommunikationsschnittstelle zum Datenaustausch zwischen ATHLET und NuT-Prozess. Das NuT-Plugin und

dessen Kommunikationsmethoden werden durch ein weiteres Schnittstellenmodul innerhalb von ATHLET bereitgestellt.

4.3 NuT-Prozess

Dieser Abschnitt beschreibt die Implementierung der drei Abstraktionsschichten im NuT.

4.3.1 Kommunikation

Die Kommunikationsschicht ist für die Steuerung des NuT-Prozesses zuständig.

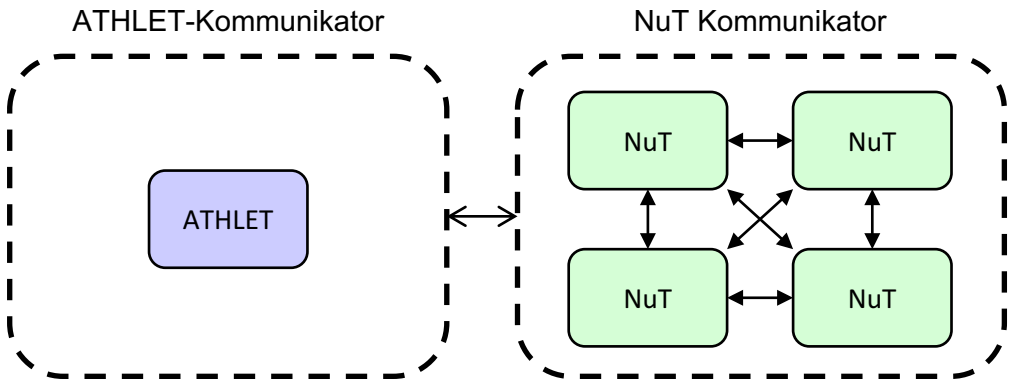


Abbildung 4.3: MPI Kommunikatoren zwischen ATHLET und NuT

Zuerst werden MPI-Bibliothek und Kommunikatoren initialisiert, die zum Datenaustausch zwischen den unterschiedlichen Prozessen benötigte werden. Zweck der MPI-Kommunikatoren ist es, die globale Kommunikation innerhalb der NuT-Prozesse, sowie zwischen ATHLET und den NuT-Prozessen, kollisionsfrei zu ermöglichen [For15b, Kap. 6.1.1]. Für jeden ATHLET Prozess und für alle zugehörigen NuT-Prozesse wird jeweils ein Kommunikator erstellt. Der letztere wird an PETSc als Basis weitergegeben und definiert damit einen abgeschlossenen Rechenverbund über den parallelisiert wird. ATHLET unterstützt bisher selbst keine Parallelität und darf deshalb auch nur als einfacher Prozess gestartet werden. Es besteht die Möglichkeit, dass auch der ATHLET-Prozess in Zukunft über MPI parallelisiert wird. Deshalb werden in der aktuellen Implementierung bereits alle NuT-Prozesse nach dem Start auf alle vorhandenen ATHLET-Prozesse zugeordnet. Jeder ATHLET-Prozess erhält dadurch den gleichen Anteil an ihm exklusiv zugeordneten NuT-Prozessen, um seine eigenen Berechnungen auszulagern. Wird nur ein einziger ATHLET-Prozess

gestartet, werden alle verfügbaren NuT-Prozesse ihm zugeordnet (Abbildung 4.3).

Die Kommunikationsschicht verbindet als Schnittstelle die Aufrufe von ATHLET mit den Funktionen aus der AL-Schicht. Daten müssen in beide Richtungen - von ATHLET zur AL-Schicht und umgekehrt - ausgetauscht werden. Da die weiterzuleitenden Daten teilweise sehr groß sind und die tatsächliche Größe erst unmittelbar vor dem Transfer bekannt ist, wird eine dynamische Speicherverwaltung eingesetzt. Damit dies so übersichtlich wie möglich geschieht und um mögliche Speicherlecks im Vorfeld auszuschließen, wird der für die Übertragung benötigte Speicher ausschließlich zentral in der Kommunikationsschicht allokiert und auch dort direkt im Anschluss wieder freigegeben. Dazu muss die Größe des zu übertragenden Speichers in der Kommunikationsschicht bereits vor der Übertragung bekannt sein. Soll beispielsweise ein Array von ATHLET zur AL-Schicht übertragen werden, so kommuniziert ATHLET davor dessen Größe, damit der Speicher vor dem Empfang allokiert werden kann. Im Anschluss wird der Zeiger an die Zielfunktion der AL-Schicht weitergegeben und dort eingelesen bzw. weiterverarbeitet. Nach erfolgreichem Aufruf wird dieser wieder in der Kommunikationsschicht wieder freigegeben. Werden von ATHLET aus Daten mit vorher nicht bekannter Größe aus der AL-Schicht angefordert, ist der Ablauf etwas umständlicher. Das liegt daran, dass die Daten in der AL-Schicht für gewöhnlich nur in abstrakter Form vorliegen. Wenn diese dann nach außen übertragen werden sollen, so müssen sie vorher in einen einfachen Datentyp umgewandelt werden. Die Umwandlung von abstrakten ggf. parallel verteilten Datentypen zu einen einfachen lokalen Datentypen kann nur in der NuT-Schicht erfolgen. Da auch nur dort die lokale Größe bekannt ist, muss sie von der Kommunikationsschicht vorher angefragt werden. Danach kann sie den Speicher allokieren und zur Befüllung an die NuT-Schicht weitergeben. Abschließend erfolgt die Übertragung an ATHLET und die Deallokierung.

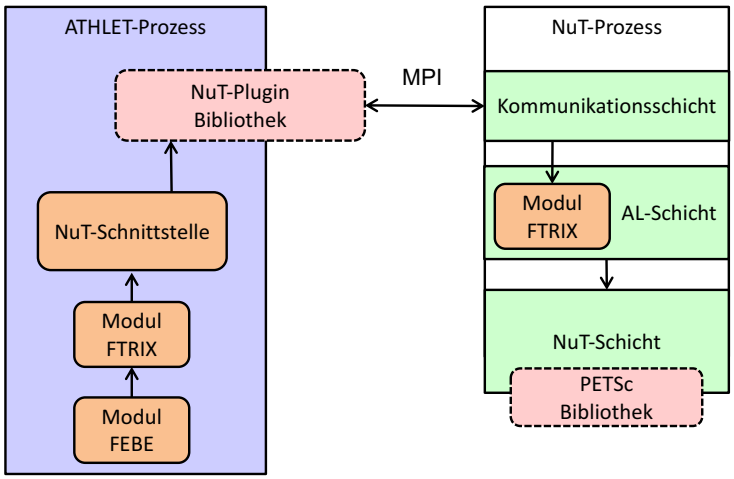


Abbildung 4.4: Kopplung zwischen ATHLET und NuT

4.3.2 Abstrakter Löser

In dieser Schicht befinden sich die Module, die in Form eines ALs ihre domänspezifischen Aufgaben berechnen. Im folgenden werden die verwendeten Daten und Lösungsfunktionen für den FTRIX-Aufgabendomän vorgestellt. Die Hauptaufgaben sind durch folgende Funktionen implementiert:

initialize Empfängt die Blockstruktur aus den ITAB-Arrays und speichert diese.

buildMatStructure Erhält die TOP-Arrays mit den Informationen darüber, welche Zeilen und Spalten aktiv sind und erstellt daraus in Kombination mit der gespeicherten Blockstruktur, die Jacobi-Matrix und die Jacobi-Strukturmatrix. Die letzte wird ausschließlich zum Färben verwendet.

computeColoring Führt die Färbung auf Basis der aktuellen Jacobi-Strukturmatrix durch.

getSeed Gibt die Seedmatrix im CSR-Format zurück.

addPerturbation Befüllt die Jacobi-Matrix vektorweise mit den tatsächlichen Werten aus der f-Auswertung.

solve Löst unter Angabe des Zeitschrittfaktors und des b-Vektors das Gleichungssystem.

clearJacOffset Entfernt den zuletzt aufaddierten Zeitschrittfaktor von der Diagonalen.

clearFullJac	Setzt alle Einträge der Jacobi-Matrix auf Null ohne dabei ihre Struktur zu modifizieren.
clearPartialJac	Setzt aller Werte der angegebenen Spalten/Zeilen der Jacobi-Matrix auf Null.
finalize	Deallokiert alle vorhandenen abstrakten Datenobjekte des Moduls.

Als Basis werden abstrakte Matrizen, Vektoren und Index-Listen verwendet:

Listing 4.1: Internes NuT PETSc Modul

```
! Jacobi-Matrix
nutMat, private :: Jac
! Jacobi-Matrixstruktur, auf der die Färbung durchgeführt wird
nutMat, private :: JacC
! Seedmatrix, für komprimierte f-Auswertung
nutMat, private :: JacS
! Vector mit der Lösung des Gleichungssystems
nutVec, private :: x
! ITAB-Arrays mit gespeicherter Blockstruktur der Jacobi-Matrix
nutIS, private :: JacBlock_ePtr
nutIS, private :: JacBlock_rowPtr
nutIS, private :: JacBlock_colInd
! Größe des Gleichungssystems
nutInteger n
```

Die Implementierung in Listing 4.2 basiert auf den Funktionen, die durch die NuT-Schicht zu Verfügung gestellt werden. Dieser Code-Abschnitt zeigt exemplarisch die Aufrufe im AL zum Lösen des Gleichungssystems. Der Aufruf von ftrix_solve() und die Übergabe der Parameter werden von ATHLET über die Kommunikationsebene durchgeführt. Das rhs-Array wird zur Eingabe des b-Vektors und anschließend zur Rückgabe des Lösungsvektors x verwendet. Der Gewichtungsfaktor diagonalOffset beinhaltet den Zeitschrittfaktor, der die aktuelle Jacobi-Matrix zur Systemmatrix überführt. Die Funktionen nut_arrayToVec() und nut_vecToArray() kümmern sich jeweils um die Konvertierung von Vektor-Array zum abstrakten Datenobjekt vom Typ nutVec und zurück. In der Funktion nut_matShift() wird der Faktor diagonalOffset auf die Diagonale der Jacobi-Matrix aufaddiert. Anschließend wird das entstandene Gleichungssystem mithilfe von nut_solve() gelöst.

Listing 4.2: Implementierung in der NuT-Schicht

```
...
subroutine ftrix_solve(rhs, diagonalOffset)
   implicit none
   nutReal, allocatable, intent(inout) :: rhs_array(:)
   nutReal, intent(in) :: diagonalOffset

   call nut_arrayToVec(rhs_array, x)

   call nut_matShift(Jac, diagonalOffset)
   call nut_solve(Jac, x)

   call nut_vecToArray(x, rhs_array)
end subroutine
...
```

4.3.3 NuT-Schicht

Die NuT-Schicht implementiert die abstrakten Funktionen für die AL-Schicht unter Verwendung von PETSc. In Listing 4.3 sind die internen Datentypen von Matrizen und Vektoren deklariert. Sie beinhalten neben dem eigentlichen PETSc-Datenobjekt noch weitere Hilfsvariablen. Diese ermöglichen beispielsweise eine sichere Speicherverwaltung durch die Verwendung von Handles. Sie zählen die verwendeten Referenzen und sorgen dafür, dass das Objekt nur deallokiert oder überschrieben werden kann, sofern es keine weiteren Referenzen mehr darauf gibt. Um zu verhindern, dass externe Module aus der AL-Schicht auf diese Daten zugreifen, wird ihnen der in Listing 4.4 angegebene, separate Datentyp zu Verfügung gestellt. Da in Fortran keine kontextfreien Zeiger existieren, werden C-Zeiger benutzt. Diese müssen bei

jedem Funktionsaufruf in der NuT-Schicht vom abstrakten NuT-Datentyp in den internen NuT-Datentyp umgewandelt werden.

Listing 4.3: Internes NuT PETSc Modul

```
...
   type :: nutMatP
      Mat :: a = 0
      KSP :: solver = 0
      nutBool, dimension(:), allocatable :: activeColumns
      nutInteger :: assembled = TAG_UNALLOCATED
      nutInteger :: cachedValues = 0
      nutReal :: shiftOffset = 0.0d0
      nutInteger :: handles = 1
   end type

   type :: nutVecP
      Vec a
      nutInteger :: assembled = TAG_ASSEMBLED
      nutInteger :: cachedValues = 0
      nutInteger :: handles = 1
   end type
...
```

Listing 4.4: NuT PETSc Modul

```
...
   type, bind(C) :: nutMat
      type(c_ptr) :: ptr
   end type

   type, bind(C) :: nutVec
      type(c_ptr) :: ptr
   end type
```

Listing 4.5 zeigt exemplarisch die Implemtierung einer Addition zweier Vektoren $v = \text{scalar} \cdot u + v$ mit einem skalaren Faktor. Als Eingabe wird jeweils der abstrakte Datentyp gefordert. Dieser wird dann mittels c_f_pointer() in den internen Datentyp umgewandelt. Die Funktion assembleVec_intern() prüft, ob sich die PETSc-Vektoren bereits im verwendbaren Zustand befinden und bereitet sie sonst mittels interner PETSc-Funktionen für die Verwendung vor. Zuletzt wird die Vektoraddition mit VecAXPY() und der Übergabe der tatsächlichen PETSc-Datentypen durchgeführt. Auf diese Art und Weise werden alle Funktionen zu Verfügung gestellt, die in der

NuT-Schicht benötigt werden.

Listing 4.5: NuT PETSc Modul

```
subroutine nut_vecAddVec(nut_v, scalar, nut_u)
   use nut_petsc_intern
   implicit none
   type(nutVec), intent(inout) :: nut_v
   nutReal, intent(in) :: scalar
   type(nutVec), intent(in) :: nut_u

   type(nutVecP), pointer :: v
   type(nutVecP), pointer :: u
   call c_f_pointer(nut_v%ptr, v)
   call c_f_pointer(nut_u%ptr, u)

   call assembleVec_intern(v)
   call assembleVec_intern(u)

   call VecAXPY(v%a, scalar, u%a, ierr)
end subroutine
```

4.4 Plattformkompatibilität Windows

Innerhalb der GRS wird ATHLET hauptsächlich auf dem internen Linux-Cluster verwendet. Da externe Anwender jedoch hauptsächlich Windows als Betriebssystem einsetzen, wird ATHLET für beide Plattformen angeboten. Deshalb ist es wichtig, dass alle verwendeten Erweiterungen unter Linux und Windows einsetzbar sind. Die umgesetzten Änderungen innerhalb des ATHLET Fortran Codes sind komplett plattformunabhängig und daher ohne weitere Anpassungen verwendbar. Die erforderlichen Bibliotheken MPI und PETSc sind in erster Linie für Linux gedacht und benötigen ein paar besondere Anpassungen, um auch unter Windows lauffähig zu sein. Auf Linux kommt OpenMPI in der Version 2.0.1 zum Einsatz. Es unterstützt den MPI 3.1 Standard und steht auf dem Linux-Cluster direkt zu Verfügung.

MPI

Für Windows gestaltet sich die Auswahl einer passenden MPI-Bibliothek schwieriger als unter Linux.
Für das unter Linux verwendete OpenMPI wurde die native Windows Unterstützung

nach der Version 1.6.5 im Jahr 2013 eingestellt [Ope]. Die aktuelle Version benötigt für die Kompilierung der OpenMPI Bibliothek und der Laufzeitumgebung unter Windows zwingend Cygwin. Cygwin beinhaltet eine Unix ähnliche Umgebung, mit der Linuxprogramme auf Windows portiert werden können. Die dort erstelle OpenMPI Bibliothek kann zwar verlinkt werden, allerdings benötigt die Laufzeitbibliothek zur Ausführung nach wie vor die Cygwin Umgebung. Es ist jedoch wichtig, dass ATHLET als ein fertiges Komplettpaket für Windowsanwender ausgeliefert werden kann und keine komplizierten Konfigurationen oder Zusatzinstallationen nötig sind. Da Cygwin für OpenMPI auch zur Laufzeit nötig ist, muss entsprechend eine andere MPI Bibliothek verwendetet werden. Neben der OpenMPI Bibliothek gibt es auch noch eine Implementierung von Intel und eine von Microsoft. Die Erstere kann momentan aufgrund von fehlenden Lizenzen nicht unter Windows verwendet werden. Somit fällt die Wahl auf Microsoft MPI. Diese kostenlose Bibliothek wird zurzeit aktiv entwickelt und befindet sich in der Version v8. Die Entwicklungs- und die Laufzeitbibliothek können jeweils separat ohne Registrierung heruntergeladen und installiert werden. Der Installationsvorgang der Laufzeitbibliothek geht reibungslos von statten und benötigt keine weitere Konfiguration durch den Benutzer. Es ist daher mit nur geringem Zusatzaufwand für den Endnutzer durch die MPI Installation zu rechnen. Leider unterstützt Microsoft MPI in der aktuellen Version nicht den, in OpenMPI bereits vorhandenen, MPI 3.1 Standard. Betreffend der aktuellen Implementierung des NuT-Prozesses ergeben sich hauptsächlich Unterschiede in der Einbindung der MPI-Module und bei den verwendeten Datentypen. Die Kompatibilität zu beiden Standards wird durch die Verwendung der passenden Datentypen mithilfe von Preprozessor Direktiven umgesetzt.

PETSc

Neben den integrierten Lösungsverfahren bietet PETSc auch vorbereitete Schnittstellen zur Erweiterung mit externen Paketen. Sie können bei der Konfiguration von PETSc bequem per Kommando angegeben werden. Anschließend lädt das PETSc-Konfigurationsskript diese selbstständig herunter und kompiliert sie. Um bei der ersten lauffähigen Kopplung von ATHLET und NuT-Prozess leichter beurteilen zu können, dass sie korrekt arbeitet, sollen im NuT-Prozess zuerst möglichst ähnliche Lösungsverfahren wie in ATHLET zum Einsatz kommen. Andernfalls würde sich die Frage stellen, ob mögliche auftretende Abweichung aufgrund von Fehlern in der Implementierung oder durch die Verwendung eines anderen Lösungsverfahrens hervorgerufen werden. Gerade die Verwendung iterativer Verfahren [Bar+94] können aufgrund eines unpassend eingestellten Konvergenzkriteriums das Ergebnis empfindlicher Systeme stark beeinflussen [Jac13, Kap 5.2]. Somit werden in dieser Arbeit nur direkte Lösungsverfahren eingesetzt. Zwar hat PETSc zum aktuellen Zeitpunkt über 17 parallele iterative Lösungsmethoden wie zum Beispiel

GMRES und BiCG-Stab, bietet aber von Haus aus nur eine einzige einfache Implementierung der LR-Zerlegung, die nicht parallelisiert ist [Bal+16b]. Die verwendete Konfiguration ist in Listing 4.6 aufgelistet und benutzt Netlibs LAPACK und ScaLAPACK, sowie MUMPS als parallelen Löser und METIS zur Fill-In-Reduktion.

Listing 4.6: Konfiguration von PETc

```
./configure \
--with-shared-libraries=1 \
--download-fblaslapack \
--download-scalapack \
--download-mumps \
--download-metis --download-metis-shared=0 \
--with-debugging=0 \
COPTFLAGS='-O3 -mtune=native' \
CXXOPTFLAGS='-O3 -mtune=native' \
FOPTFLAGS='-O3 -mtune=native'
```

5 Bewertung

In diesem Kapitel werden die sequentiellen und parallelen Ergebnisse des neuen Lösungsverfahrens vorgestellt. Neben den Rechenzeiten werden auch die Ergebnisse mithilfe von Integrationstests von ATHLET verifiziert und ausgewertet. Damit soll sichergestellt werden, dass die Implementierung weitgehend fehlerfrei und einsetzbar ist.

5.1 Löserauswahl

Von allen direkten Lösungsverfahren für lineare Gleichungssysteme, die von PETSc direkt oder per externen Schnittstelle angeboten werden, kommen nur die parallelen Verfahren wie MUMPS [ADL00] [Dav06] und SuperLU Dist [Li+16] in Frage. Aufgrund seiner umfangreichen Einstellmöglichkeiten und den besseren Ergebnissen in vorläufigen Vergleichtests, werden alle folgenden Rechnungen mit MUMPS durchgeführt.

Für die in MUMPS durchgeführte Fill-In-Reduktion, kann aus einer Vielzahl von Verfahren ausgewählt werden. Auf Empfehlung der PETSc-Entwickler wird METIS [KK99] eingesetzt. Metis ist ein serieller Algorithmus, der ursprünglich für die Partitionierung von Graphen entwickelt wurde und sehr gut für den Einsatz als Fill-In reduzierendes Verfahren geeignet ist.

Für das Färben der Jacobi-Matrix stehen die Algorithmen smallest-last (SL), largest-first (LF) und incidence-degree (ID) aus MINPACK [JJMH80] und die zwei PETSc Implementierungen greedy und Jones-Plassmann [Bal+16a, Kap. 5.6] zu Verfügung, die je nach Jacobi-Matrix unterschiedlich gute Ergebnisse liefern. Als Entscheidungskriterium für die Wahl des besten Algorithmus dient nicht die Anzahl der resultierenden Farben, sondern die kürzeste Rechenzeit für die vollständige Generierung der Jacobi-Matrix. Der Grund dafür ist, dass nicht nur die Anzahl der f-Auswertungen, sondern auch die Kombination der gemeinsam gestörten Elemente, Einfluss auf die Rechenzeit der f-Auswertung haben können. So kann sie vergleichsweise mit etwas mehr f-Auswertungen, aber anders kombinierten Störungen, im Gesamten geringer sein. Empirisch hat der Algorithmus incidence-degree (ID) die beste Leistung erzielt. Im Vergleich zum in ATHLET verwendeten smallest-last (SL)

Algorithmus beschleunigt dieser die Generierung der Jacobi-Matrix im Schnitt um etwa 15%.

5.2 Ergebnisse

Die Ergebnisse der sequentiellen Ausführung von ATHLET-NuT sind in Tabelle 5.1 dargestellt.

Sequentielle Ausführung

Die Rechenzeiten für Färbung und Fill-In-Reduktion werden im Gegensatz zu den Messungen aus ATHLET in Tabelle 3.1 nicht mehr der Startphase, sondern der Transientenphase zugeordnet. In ATHLET-NuT fallen sie auch bei den großen Modellen verhältnismäßig klein aus, obwohl sie ständig neuberechnet werden müssen. Beispielsweise benötigt ATHLET mit dem K3-7 Modell knapp 2 Stunden zur Berechnung der Färbung und 28 Stunden zur Fill-In-Reduktion. Die insgesamt von der Startphase benötigten 32 Stunden werden in ATHLET-NuT auf insgesamt 9 Minuten gesenkt. Die Zeitdauer zur Durchführungen der insgesamt 207 Färbungen während der Simulation beträgt zusammen nur 13s und die 41 Fill-In-Reduktionen sind innerhalb einer Minute berechnet. Dass hier bereits ohne Parallelisierung deutlich bessere Ergebnisse mit PETSc erzielt werden, zeigt wie ineffektiv die entsprechenden Algorithmen in ATHLET umgesetzt sind. In der Transientenphase machen sich weitere interessante Aspekte bemerkbar. Es gibt deutliche Schwankungen in der Anzahl der berechneten Zeitschritte und der Jacobi-Matrizen im Vergleich zu ATHLET. Das liegt wahrscheinlich wie in Abschnitt 5.3 angemerkt, an den sehr empfindlichen Systemen, die ATHLET produziert. Da davon abhängig der gesamte Rechenaufwand variiert, fließt es als Störfaktor in die Ergebnisse mit ein und macht sich gerade bei der Verwendung des $i2mftrx$ Parameters bemerkbar. So benötigt ATHLET-NuT im PWR-3D+ Modell nur 5.419 volle Berechnungen der Jacobi-Matrix statt ATHLET mit 7.372. Abgesehen davon, führt dieser in ATHLET im PWR-3D Beispiel zu einer Faktor 5-mal längeren Transientenphase und die K3 Modelle lassen sich praktisch gar nicht mehr berechnen. In ATHLET-NuT steigt die Transientenszeit im K3-2 Beispiel um 17% an. Im PWR-3D Beispiel bleibt sie ungeachtet von $i2mftrx$ in etwa gleich und im K3-7-Beispiel verursacht sie sogar eine Beschleunigung der Gesamtrechenzeit um 21%.

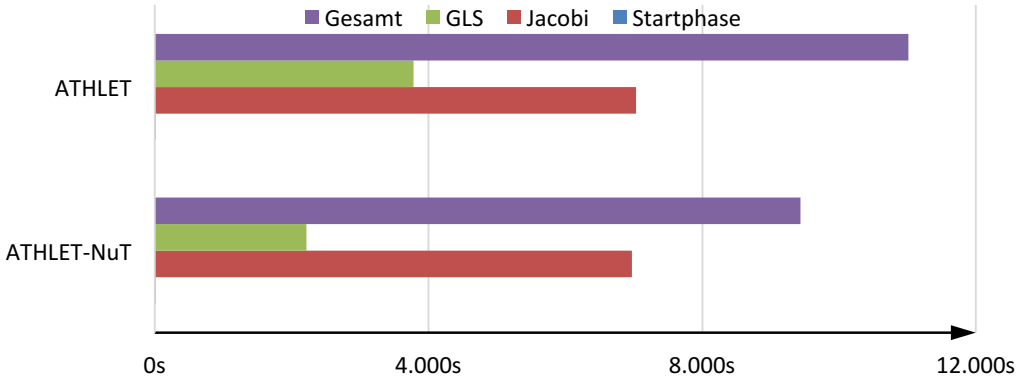


Abbildung 5.1: Sequentielle Simulationszeit PWR-3D

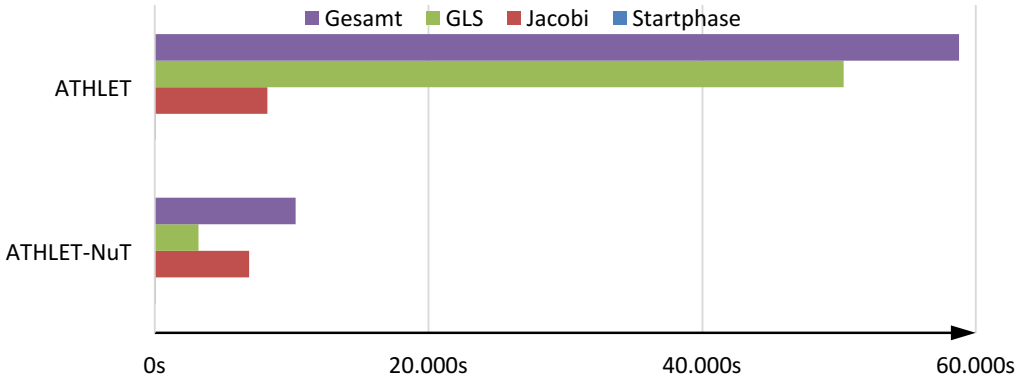
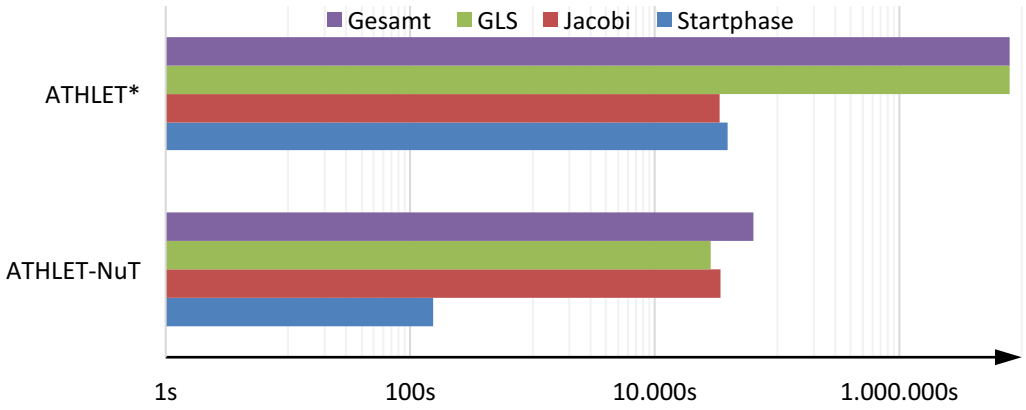


Abbildung 5.2: Sequentielle Simulationszeit PWR-3D+

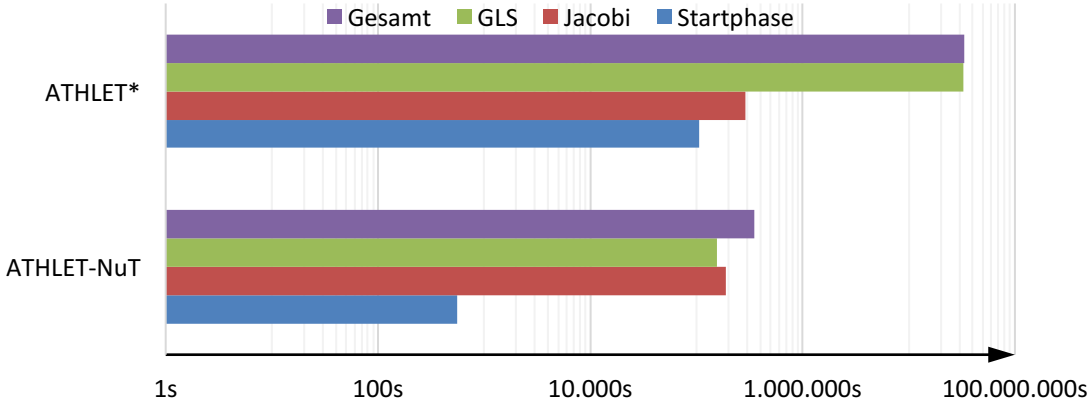
Tabelle 5.1: Sequentielle Rechenergebnisse von ATHLET-NuT

	PWR-3D	PWR-3D+	K3-2	K3-2+	K3-7	K3-7+
Dimension	6.009	6.009	128.673	128.673	258.371	258.371
Anzahl Einträge	143.469	151.961	2.852.179	5.240.891	5.865.471	13.693.047
Startphase in *s*	3	4	155	311	555	1.879
Anzahl Zeitschritte	15.292	14.738	2.164	1.868	2.237	1.890
Anzahl GLS	154.199	146.846	17.226	13.999	17.816	14.267
Anzahl Jacobi Partiell	7.755	7.223	264	185	185	156
Anzahl Jacobi Voll	5.603	5.419	907	569	1111	677
Färbung in *s*	6	6	6	20	13	72
Fill-In in *s*	11	11	27	30	62	80
GLS in *s*	2.215	3.168	28.585	49.379	155.640	152.080
Jacobi in *s*	6.978	6.882	34.439	24.615	190.912	118.316
Anteil GLS	23%	31%	45%	66%	44%	55%
Anteil Jacobi	74%	67%	54%	33%	54%	43%
Transientenphase in *s*	9.432	10.277	64.149	75.133	351.050	274.589
Gesamt in *s*	9.435	10.281	64.304	75.444	351.605	276.468



* extrapolierte Werte

Abbildung 5.3: Sequentielle Simulationszeit K3-2



* extrapolierte Werte

Abbildung 5.4: Sequentielle Simulationszeit K3-7

Parallele Ausführung

Um über weitere mögliche potentielle Leistungssteigerungen durch Parallelisierung eine Aussage zu treffen, ist es sinnvoll, die größten relativen Anteile der Gesamtrechenzeit zu betrachten. Die Berechnung der Jacobi-Matrix und das Lösen der Gleichungssysteme machen bei allen Modellen jeweils einen Anteil von $\geq 97\%$ der Gesamtzeit aus. Dabei variiert der Anteil des GLS-Lösers zwischen 23% und 66%. Somit kann durch eine Parallelisierung oder sequentieller Optimierung höchstens ein theoretischer Geschwindigkeitszuwachs des Faktors 3 erzielt werden. Zum Analysieren der Skalierbarkeit wird das K3-7+ Modell mit 1,2, 8, 16 und 32 NuT-Prozessen berechnet. Da ein Cluster-Knoten insgesamt nur 20 Prozessorkerne besitzt, werden die 32 NuT-Prozesse auf zwei Cluster-Knoten verteilt. Alle anderen werden innerhalb eines Cluster-Knotens ausgeführt. Da die zu Verfügung stehenden Cluster-Resourcen knapp sind, werden nur die ersten 100 von insgesamt 1.890 Zeitschritte berechnet. Da Startphase und das Berechnen der Jacobi-Matrix komplett sequentiell ablaufen und die benötigte Zeit zum Färben und zur Fill-In-Reduktion vernachlässigbar klein sind, besteht der parallele Anteil nur noch aus dem Lösen des Gleichungssystems. Sein relativer Anteil an der gesamten Simulationszeit beträgt im K3-7+ Modell für die ersten 100 Zeitschritte insgesamt 44%. Nach Amdahl kann damit der maximale Speedup (relative Beschleunigung der Rechenzeit) den Wert $\frac{1}{1-0,44} = 1,79$ nicht überschreiten. In Abbildung 5.5 ist der erreichte Speedup der Gesamtzeit, des Gleichungssystemlösers und der sequentiellen Startphase in Abhängigkeit der verwendeten NuT-Prozessen angegeben. Zusätzlich kann der ideale Speedup der Gesamtzeit nach Amdahl als Orientierungsmaßstab verwendet werden. Der Speedup der sequentiellen Startphase ist deshalb angegeben, weil die Ausführung jedes Prozesses zunehmend mit der Gesamtprozessorauslastung gebremst wird. Dieser Faktor lässt sich anhand der Startphase einschätzen, da sie ansonsten komplett unabhängig von der Parallelisierung ist und in erster Linie durch die Hintergrundsauslastung beeinflusst wird. Für einen sequentiellen Prozess ist eine Verschlechterung der Laufzeit bei voller Cluster-Auslastung im Vergleich zur alleinigen Ausführung von insgesamt 15% festzustellen.
Die gesamte Simulation erzielt ihren maximalen Speedup von 1,31 mit 4 Prozessoren. Darüber hinaus fällt der Speedup aufgrund von zunehmender Kommunikation und der beschriebenen höheren Gesamtauslastung wieder ab. Der Rechenanteil zur Lösung der Gleichungssysteme verbessert sich mit steigender Prozessoranzahl, wobei die Beschleunigung von 16 auf 32 Prozessoren etwas stagniert. Dies liegt daran, dass ab der Verwendung von 32 Prozessoren über einen Cluster-Knoten hinaus kommuniziert werden muss und sich folglich die Kommunikationskosten noch weiter erhöhen. Hinzu kommt, dass die verwendeten Systeme mit einer Dimension von zirka 250.000 für eine bessere Skalierung in MUMPS noch zu klein sind. Der Speicherverbrauch in ATHLET, der aufgrund der verteilten Auslagerung der Gleichungssysteme reduziert wird, fällt relativ gering aus. Im Datensatz K3-7 verbraucht

ATHLET in seiner ursprünglichen Form insgesamt 10GB an Arbeitsspeicher. Nach der Kopplung mit dem NuT werden davon etwa 400MB ausgelagert. Somit veranschlagt der ATHLET-Prozess anschließend immer noch insgesamt 9,6GB. Alleine mit der verteilten Speicherung der Gleichungssysteme ist es noch nicht möglich, die Prozessgröße von ATHLET signifikant zu verringer, um so größere Modelle berechnen zu können.

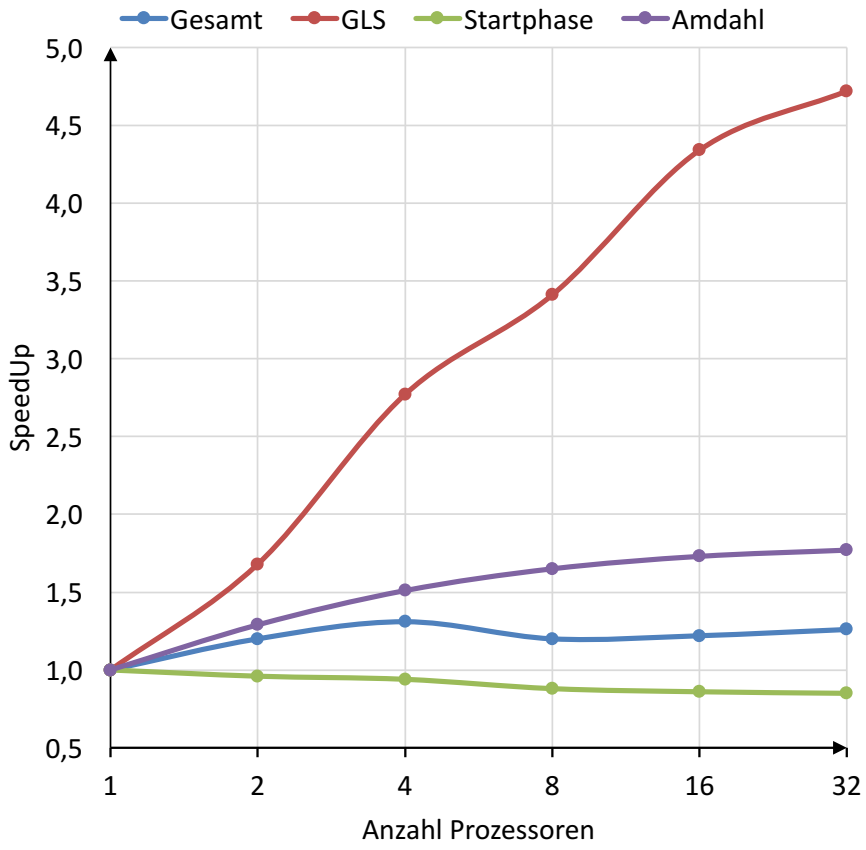


Abbildung 5.5: Parallele Rechenergbnisse von ATHLET-NuT für K3-7+

5.3 Verifikation

Zur Verifikation des Codes werden im Rahmen eines Integrationstests mehrere Referenzdatensätze mit ATHLET berechnet und die Ergebnisse ausgewählter Größen mit vorhandenen Referenzkurven verglichen. Diese Referenzkurven enthalten zum einen Messwerte einiger Größen aus echten Anlagen und zum anderen einen Bereich, der die Rechenergebnisse vergangener ATHLET Versionen über die Entwicklungszeit zusammenfasst. Nach der Berechnung der Referenzmodelle werden die berechneten Größen per Skript mit den Referenzkurven verglichen und überprüft, ob diese mit zuzüglich einer Toleranz von 2% innerhalb des vorgegebenen Bereichs liegen. Anschließend werden alle Abweichungen zwischen tatsächlichem Ergebnis und Referenzbereich sowohl graphisch als auch in Zahlen dargestellt. Damit kann schnell überprüft werden, welche und wie große Abweichungen durch eine Änderung im Code verursacht wird. Schwerpunkt bei der Bewertung der Ergebnisse liegt nicht darauf, dass der Toleranzbereich immer eingehalten wird. Viel wichtiger ist ein ähnlicher Kurvenverlauf im Vergleich zur Referenzkurve und falls Ausreißer vorhanden sind, wie lange sie zeitlich und wie stark sie den Toleranzbereich überschreiten. Neben physikalischer Größen werden auch statistische Daten der Numerik verglichen. Beispielsweise die Zeitschrittweiten oder die Anzahl der vollen und partiellen Aktualisierungen der Jacobi-Matrix über die Zeit.

Die Ergebnisse des Integrationstests auf Basis aller vier Referenzdatensätze sind in der Tabelle 5.2 zusammengetragen. Für jedes Modell werden unterschiedlich viele Teiltests durchgeführt, dessen Anzahl unter „Tests Gesamt“angegeben ist. Getestet wurde ATHLET im unmodifizierten Zustand und ATHLET mit NuT-Erweiterung, beide jeweils zum selben Entwicklungsstand (03. März 2017). Das ATHLET-NuT Programm verwendete die sequentielle Version von MUMPS als direktes Lösungsverfahren mit aktivierter numerischer Stabilisierung. Die Zahlen in der Tabelle geben an, wie viele Teiltests die entsprechende Implementierung positiv bestanden hat. Im aktuellen Entwicklungsstand besteht keines der beiden Programme alle Tests. Das liegt daran, dass die Referenzkurven auch auf empirischen Simulationsdaten basieren und in Abhängigkeit des aktuellen Entwicklungsprozesses von ATHLET sich Abweichungen der Ergebnisse ergeben. Die zu berechnenden System sind ungünstiger Weise zum Teil sehr empfindlich, wodurch kleine Veränderungen teilweise große Auswirkungen auf Spitzen und Schwingungen im Kurvenverlauf haben können. Andere Abweichungen sind auf Verbesserungen oder Verschlechterungen des Codes zurückzuführen, dessen Differenzierung ein manuelles Betrachten und Analysieren der Ergebnisse erforderlich macht.

Insgesamt schneidet ATHLET-NuT gleich gut oder im schlechtesten Fall mit 10 Prozentpunkten schlechter ab als ATHLET. Es ist anzunehmen, dass ATHLET-NuT in seiner Grundfunktion korrekt ist, da der Großteil der Tests positiv ausfällt. In Abbildung 5.6 sind jeweils zwei Ergebnisse von ATHLET und ATHLET-NuT gegen-

Tabelle 5.2: Ergebnisse des Integrationstests im Vergleich

	ISB2		LSTF		ROSA		LOFT	
Tests Gesamt	66		144		253		774	
ATHLET	65	98%	139	97%	242	96%	745	96%
ATHLET-NuT	61	92%	125	87%	242	96%	705	91%

übergestellt, bei dem ATHLET den Test bestanden hat und ATHLET-NuT nicht. Die beiden oberen Graphen zeigen den Massefluss im Reaktorkern von ATHLET (links) und ATHLET-NuT (rechts) und die zweit unteren Graphen zeigen den Verlauf des Masseresidiuums über die Zeit. Die blaue Linie steht für den empirisch bestimmten Referenzbereich, die graue Linie repräsentiert echte Messdaten und die schwarze Linie gibt die aktuell berechneten Werte an. Die roten Stellen markieren das Überschreiten des Rerefenzbereichs zuzügliche der 2% Toleranz. Die roten Abweichungen sind in beiden Fällen wahrscheinlich nicht als sehr gravierend einzuschätzen: Im oberen Beispiel verschiebt sich die Spitze einer Schwingung und im unteren Fall wird die Toleranzgrenze etwas mehr als im originalen ATHLET überschritten. Wichtig ist, dass der Kurvenverlauf von ATHLET-NuT grundsätzlich dem von ATHLET entspricht. Tests während der Entwicklung haben ergeben, dass die Häufigkeit der gezeigten Ausreißer innerhalb dieser Größenordnung stark von der Wahl des Gleichungssystemlösers und seinen Einstellungen abhängt. Zusätzlich werden in ATHLET bis jetzt keinerlei Verfahren verwendet, die gezielt durch Permutation und Skalierung die numerische Stabilität des Gleichungssystems optimieren. Insgesamt können mit der aktuellen Implementierung der ATHLET Erweiterung brauchbare Ergebnisse erzielt und die Kopplung als funktionierend betrachtet werden.

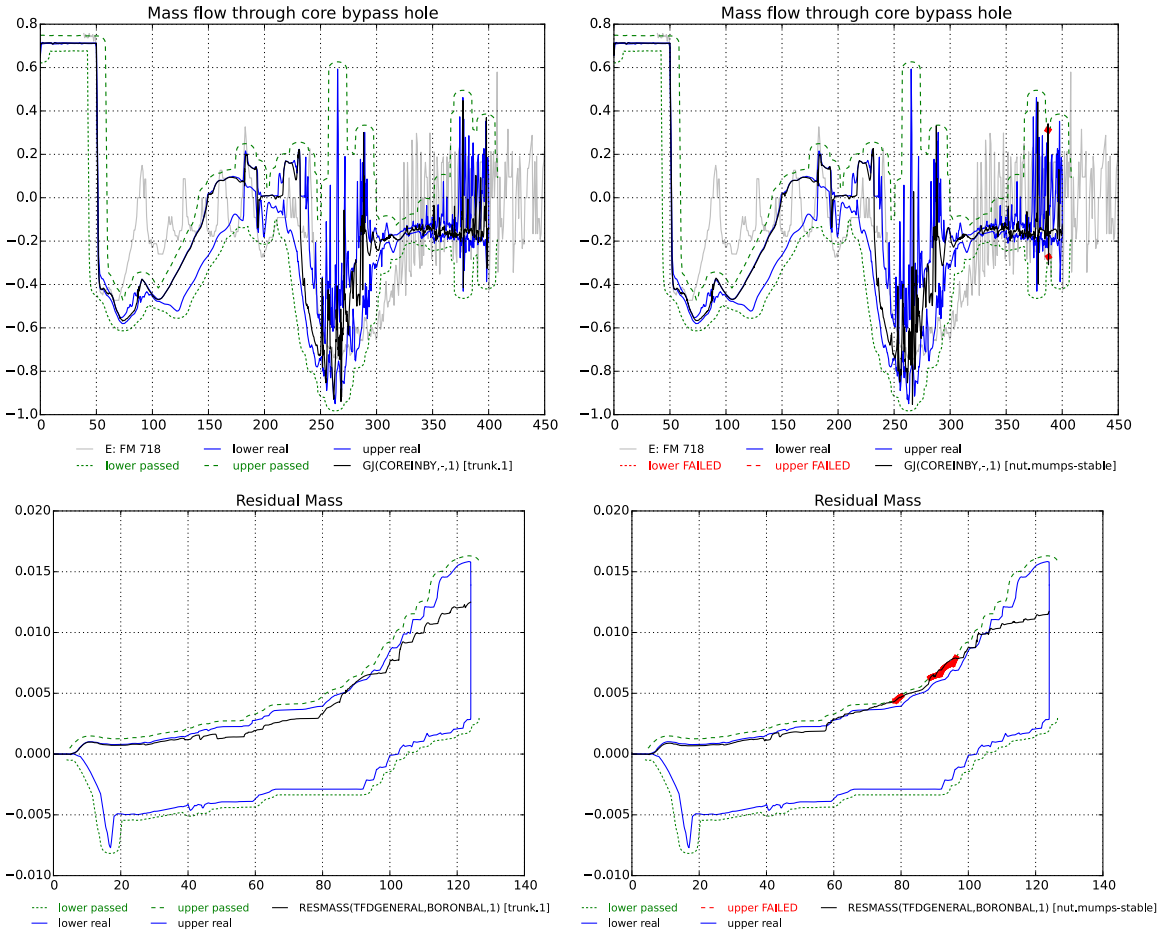


Abbildung 5.6: Referenzkurven mit ATHET (links) und ATHLET-NuT (rechts)

6 Zusammenfassung

Die Erweiterung von ATHLET durch den NuT bringt unterschiedliche Vorteile. Im Vergleich zu ATHLET können nun deutlich größere Modelle in vernünftiger Zeit berechnet werden. Zusätzlich führt die Verwendung des $i2mftrx$ Paramters nicht mehr zu einer unverhältnismäßigen Verschlechterung der Rechenzeit. Im K3-7 Beispiel kann sie damit sogar deutlich verkürzt werden. Diese Leistungsverbesserungen sind hauptsächlich den besseren sequentiellen Algorithmen zum Lösen der Gleichungssysteme, aber auch der Berechnung der Fill-In-Reduktion und der Färbung der Jacobi-Matrix zu verdanken. Letzterer reduziert zusätzlich, aufgrund seiner besseren Ergebnisse, die Zahl der f-Auswertung und damit die Berechnung der Jacobi-Matrix. Im kleinen Maßstab bis 4 Prozessen, kann die parallele Ausführung des mittelgroßen K3-7 Modells eine zusätzliche Beschleunigung um den Faktor 1,3 bewirken. Um eine deutlich besser Beschleunigung darüber hinaus zu erhalten, muss der parallele Rechenanteil erhöht werden. Diese geschieht durch die Verwendung von Modellen mit größeren Gleichungssystemen, der Parallelisierung von zusätzlichen Programmteilen und der Kommunikationsverbesserung. Abgesehen von der Rechenzeit kann ein Teil des verwendeten Speichers auf mehrere Cluster-Knoten verteilt werden. Allerdings ist der Speicherverbrauch durch das Gleichungssystem nur ein relativ kleiner Teil von ATHLET. Soll ATHLET aufgrund von begrenztem Speicher parallel ausgeführt werden, so ist auch nötig, dessen Speicherverbrauch zu optimieren. Schließlich ist durch die Kopplung von ATHLET und dem NuT eine neue abstrakte Löserplattform gegeben. Sie ermöglicht das komfortable Programmieren auf mathematischer Ebene unter Verwendung der mächtigen PETSc Bibliothek. Lösungsverfahren können damit viel leichter ausgetauscht werden und in Zukunft sollen auf dieser Basis alternative numerische Lösungsverfahren für die Zeitintegration umgesetzt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[Msm] 2017. URL: https://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx (siehe S. 31).

[Ope] 2017. URL: https://www.open-mpi.org/software/ompi/v1.6/ms-windows.php (siehe S. 39).

[ADL00] P. R. Amestoy, I.S. Duff und J.-Y. L'Excellent. „Multifrontal parallel distributed symmetric and unsymmetric solvers". In: *Comput. Methods Appl. Mech. Eng.* 184.2-4 (2000), S. 501–520 (siehe S. 41).

[Aus+16] H. Austregesilo u. a. *ATHLET Models and Methods*. 3.1A. Gesellschaft für Anlagen und Reaktorsicherheits (GRS) gGmbH. März 2016 (siehe S. 4 f., 8 f.).

[Bal+97] Satish Balay, William D. Gropp, Lois Curfman McInnes und Barry F. Smith. „Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Hrsg. von E. Arge, A. M. Bruaset und H. P. Langtangen. Birkhäuser Press, 1997, S. 163–202 (siehe S. 30).

[Bal+16a] Satish Balay u. a. *PETSc Users Manual*. Techn. Ber. ANL-95/11 - Revision 3.7. Argonne National Laboratory, 2016. URL: http://www.mcs.anl.gov/petsc (siehe S. 8, 30, 41).

[Bal+16b] Satish Balay u. a. *PETSc Web page*. http://www.mcs.anl.gov/petsc. 2016. URL: http://www.mcs.anl.gov/petsc (siehe S. 40).

[Bar+94] R. Barrett u. a. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994 (siehe S. 10, 39).

[Bro+98] William J. Brown, Raphael C. Malveau, Hays W. Skip McCormick und Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis: Refactoring Software, Architecture and Projects in Crisis*. 1. Auflage. John Wiley & Sons, 1998 (siehe S. 18).

[Bun+13] H.J. Bungartz, S. Zimmer, M. Buchholz und D. Pflüger. *Modellbildung und Simulation: Eine anwendungsorientierte Einführung*. eXamen.press. Springer Berlin Heidelberg, 2013. URL: https://books.google.de/books?id=0nUkBAAAQBAJ (siehe S. 4).

[Dav06] Timothy A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Philadelphia, PA, USA: Society for Industrial und Applied Mathematics, 2006 (siehe S. 41).

[DG17] J. Dongarra und E. Grosse. *Netlib Web page*. `http://www.netlib.org`. 2017 (siehe S. 30).

[For15a] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 3.1*. High Performance Computing Center Stuttgart (HLRS), 2015 (siehe S. 31).

[For15b] Message Passing Interface Forum. *MPI: a Message-passing Interface Standard: Version 3.1*. High-Performance Computing Center, 2015. URL: `https://books.google.de/books?id=uv1ajwEACAAJ` (siehe S. 32).

[GMP05] Assefaw Hadish Gebremedhin, Fredrik Manne und Alex Pothen. „What Color Is Your Jacobian? Graph Coloring for Computing Derivatives“. In: *SIAM Rev.* 47.4 (Apr. 2005), S. 629–705. DOI: `10.1137/S0036144504444711`. URL: `http://dx.doi.org/10.1137/S0036144504444711` (siehe S. 6 f., 10).

[GR03] Udo Graf und Peter Romstedt. *Efficient implicit time integration for two-phase flow simulation*. Techn. Ber. Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH, Germany, 2003 (siehe S. 5).

[HW10] Georg Hager und Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 2010 (siehe S. 20, 22).

[HNW93] E. Hairer, S. P. Nørsett und G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2. Aufl. Bd. 8. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 1993 (siehe S. 4).

[Hof08] E. Hofer. *Minimum Deficiency Algorithmus zur Fill-in Reduktion bei dünn besetzten Blockmatrizen*. Technische Notiz HOF-01/08. Gesellschaft für Anlagen und Reaktorsicherheits (GRS) gGmbH, 2008 (siehe S. 8, 10).

[HS06] T. Huckle und S. Schneider. *Numerische Methoden*. EXamen. press Series. Springer Berlin Heidelberg, 2006. URL: `https://books.google.de/books?id=0b-tK2TaX74C` (siehe S. 3).

[IH04] R. Isernhagen und H. Helmke. *Softwaretechnik in C und C++ - das Kompendium: modulare, objektorientierte und generische Programmierung ; ISO-C90, ISO-C99, ISO-C++98, MS-C++.NET ; [alle Programmbeispiele auf CD-ROM]*. Das Kompendium. Hanser, 2004. URL: `https://books.google.de/books?id=cWSfWOwSnHYC` (siehe S. 18, 22).

[JJMH80] B. S. Garbow J. J. Moré und K. E. Hillstorm. *User Guide for MINPACK-1*. Report ANL-80-74. Argonne National Laboratory, 1980 (siehe S. 41).

[Jac13] V. Jacht. *Implementation and benchmarking of sparse-matrix solver for two-phase flow applications*. 2013 (siehe S. 39).

[Jac+17] V. Jacht u. a. „Application of System Code ATHLET for Sub-channel Analysis“. 2017 (siehe S. 11 f.).

[KK99] G. Karypis und V. Kumar. „A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs“. In: *SIAM J. Sci. Comput.* 20 (1999), S. 359–392 (siehe S. 41).

[Ler+16] G. Lerchl u. a. *ATHLET User's Manual*. 3.1A. Gesellschaft für Anlagen und Reaktorsicherheits (GRS) gGmbH. März 2016 (siehe S. 12).

[Li+16] Xiaoye S Li u. a. *SuperLU Users' Guide*. 2016 (siehe S. 41).

[NVP11] S. P. Nikonov, K. Velkov und A. Pautz. *Detailed modeling of kalinin-3 npp vver- 1000 reactor pressure vessel by the coupled system code athlet/bipr-vver*. Techn. Ber. RRC Kurchatov Institute, Russia und Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH, Germany, 2011 (siehe S. 11).

[Ste17] Tim Steinhoff. *Zwischenbericht zum Vorhaben: Ausbau und Modernisierung der numerischen Verfahren in den Systemcodes ATHLET, ATHLET-CD, COCOSYS und ASTEC*. GRS-A 3883. Gesellschaft für Anlagen und Reaktorsicherheits (GRS) gGmbH, 2017 (siehe S. 5).

[TW67] W. F. Tinney und J. W. Walker. „Direct solutions of sparse network equations by optimally ordered triangular factorization“. In: *Proceedings of the IEEE* 55.11 (1967), S. 1801–1809 (siehe S. 10).

Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH

Schwertnergasse 1
50667 Köln
Telefon +49 221 2068-0
Telefax +49 221 2068-888

Boltzmannstraße 14
85748 Garching b. München
Telefon +49 89 32004-0
Telefax +49 89 32004-300

Kurfürstendamm 200
10719 Berlin
Telefon +49 30 88589-0
Telefax +49 30 88589-111

Theodor-Heuss-Straße 4
38122 Braunschweig
Telefon +49 531 8012-0
Telefax +49 531 8012-200

www.grs.de

ISBN 978-3-946607-52-6