TECHNISCHE UNIVERSITÄT BERLIN

Fachbereich: Physik

Diplomarbeit

(Diplom-Hauptprüfung)

bei Herrn/~~Frau~~ Prof.: E. Sedlmayr

Tag der Abgabe: 28. September 19 96

Johannes Kaiser — 150751

Name d. Diplomanden/~~Diplomandin~~ — Matrikel-Nr.

TUB-Vordruck I B - 451/90 6.5.4.3. (29.07)

Numerische Behandlung chemischer Ratennetzwerke für die Modellierung sauerstoffreicher stellarer Umgebungen

Diplomarbeit

von
Johannes Kaiser
Matr.-Nr. 150751

Betreuer:
Prof. Dr. E. Sedlmayr

angefertigt am
Institut für Astronomie und Astrophysik
der Technischen Universität Berlin
Sekretariat PN 8-1
Hardenbergstraße 36
10623 Berlin (Charlottenburg)

28. September 1996

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Massive, extended circumstellar envelopes (...) ejected by the red giants at the very end of their evolution are among the most interesting objects for cosmic chemistry.

[Beck *et al.* 1992]

1.1 Astrophysikalische Motivation

1.1.1 AGB-Sterne und ihre Hüllen

Massearme Sterne, also Sterne mit weniger als sechsfacher Masse der Sonne ($M < 6\mathrm{M}_\odot$), verlassen nach Erlöschen der Wasserstoffusion in ihrem Kern die Hauptreihe des Herzsprung-Russel-Diagramms (HRD), entwickeln sich zu Roten Riesen und fusionieren auf dem horizontalen Ast in ihrem Kern Helium (He) zu Kohlenstoff (C) und Sauerstoff (O). Sie entwickeln sich nach Erlöschen der He-Fusion auf dem Asymptotischen Riesen Ast (AGB), ohne daß in ihrem Kern die zum Einsetzen einer weiteren Kernfusion nötige Temperatur erreicht würde. Die Kerne dieser *AGB-Sterne* enthalten dann ca. 20% C und 80% O, und in Schalen um den Kern findet weiterhin He– und H-Fusion statt. Die Lage im HRD oberhalb der Hauptreihe resultiert aus relativ großen Leuchtkräften bei kleinen Effektivtemperaturen. Daraus kann auf relativ große Sternradien bei kleinen Effektivtemperaturen geschlossen werden.

Beobachtungen dieser Objekte, siehe z.B. Referenzen in [Loup *et al.* 1993], weisen einen Infrarotexzess auf, der durch staubhaltige, zirkumstellare Hüllen (ZSHn) erklärt wird. Außerdem kann auf eine Expansion der ZSHn mit Endgeschwindigkeiten von der Größenordnung von 10 km/s geschlossen werden.

Je nach beobachtetem Elementverhältnis ϵ_C/ϵ_O der C– zur O-Häufigkeiten in der Sternatmosphäre unterscheidet man M-Sterne ($\epsilon_C/\epsilon_O < 1$), S-Sterne ($\epsilon_C/\epsilon_O \approx 1$) und C-Sterne ($\epsilon_C/\epsilon_O > 1$). Ein AGB-Stern reichert seine Atmosphäre durch sog. *dredge-ups* bei der Entwicklung einer durchgehenden Konvektionszone von der He-fusionierenden Schale bis in die Atmosphäre sukzessive mit Kohlenstoff an und entwickelt sich so von einem M– über einen S– zu einem C-Stern.

Der Massenverlustrate $\dot{M}$ der Sterne durch den zur Entstehung der ZSHn nötigen Sternwind liegt in dem Bereich $10^{-8} \leq \dot{M} \leq 10^{-4}\mathrm{M}_\odot/\mathrm{yr}$ ([Knapp 1991]). Dadurch, daß der AGB-Stern so also seine äußeren Schichten verliert, erscheint er zunehmend blauer, bis seine Strahlung so energiereich ist, daß das zuvor abgestoßene Material ionisiert wird. Dann kann das

Rekombinationslicht als Leuchten der ZSH beobachtet werden, und das Objekt geht in das Stadium des Planetarischen Nebels über. Nachdem das gesamte Material des zentralen Objekts zu C und O fusioniert wurde, kühlt dieses durch seine Abstrahlung langsam zu einem Weißen Zwerg ab.

Das gesteigerte Interesse an ZSH gründet sich nun in ihrer Stellung als ein wesentliches „Verbindungsglied“ zwischen AGB-Sternen, Planetarischen Nebeln und der interstellaren Materie:

- Für ein detailliertes Verständnis des Übergangs von einem AGB-Stern zu einem Planetarischen Nebel ist offensichtlich eine physikalische Beschreibung des starken Windes von zentraler Bedeutung.

- Für die Beobachtung von AGB-Sternen ist die ZSH insofern von großem Interesse, als sie natürlich auf unserem Sehstrahl vor den zentralen Bereichen des Objekts liegt, so daß wir dieses nur durch den „Filter“ der ZSHn beobachten können und deren Einfluß für eine bedeutungsvolle Interpretation der beobachteten Spektren und Lichtkurven abschätzen müssen. An diesem Punkt tritt die — vielleicht in der Astrophysik besonders extreme — Verflechtung von experimenteller und theoretischer Physik zutage.

- Die Winde der ZSHn sind schließlich die Hauptquelle für die Injektion der schweren Elemente und insbesondere der kondensierten Phase in die interstellare Materie. Diese Phase wiederum spielt durch ihre Kühlfunktion bei der Entstehung von Protosternen eine Schlüsselrolle im kosmischen Materiekreislauf. Auch die Erde, auf der wir leben, besteht überwiegend aus vormals interstellarem Staub.

Bei der Kondensation der festen Phase aus dem Gas der ZSHn hat die molekulare Zusammensetzung dieses Gases einen essentiellen Einfluß auf die Menge und die Art des entstehenden Staubes. Deshalb ist die Beschreibung der ablaufenden chemischen Prozesse eine der wichtigsten Vorarbeiten zur Beschreibung der Staubentstehung insgesamt. Der detaillierteste Ansatz hierfür ist eine Modellierung als chemisches Ratennetzwerk der möglichen Spezies, d.h. Atome, Ionen und Moleküle.

1.1.2 Modellierung von zirkumstellaren Hüllen

Will man durch physikalische Modellierung zu einem besseren Verständnis der ZSHn gelangen, so muß man folgende Größen betrachten:

- Dichte und Geschwindigkeit (hydrodynamische Struktur)

- Temperatur, Strahlungsverteilung (thermodynamische Struktur)

- Druck (Zustandsgleichung)

- Ionen–, Elektronen–, Atom– und Molekülkonzentrationen (Chemie)

- Verteilungsfunktion der Staubpartikel (Staubnukleation und –wachstum)

Da alle diese Größen nichtlinear miteinander wechselwirken, ist eine analytische Lösung des Problems nicht möglich. Stattdessen wird mit numerischer Simulation gearbeitet, wobei letztendlich nur ein *selbstkonsistenter* Ansatz über die bloße Beschreibung des Phänomens hinaus die Möglichkeit zu physikalischem Verständnis bietet. Dies bedeutet, daß das Simulationsprogamm keine internen Parameter enthalten sollte, mit Hilfe derer das Ergebnis an die Beobachtungen gefittet werden kann; es sollte zu jedem Satz von stellaren Parametern (Masse M, Leuchtkraft L, effektive Temperatur T, elementare Häufigkeiten $\{\epsilon_i\}$) eindeutige Vorhersagen für die beobachtbaren Größen wie Massenverlust, Windgeschwindigkeit und Staubdichte liefern.

Durch ausreichend detaillierte Beschreibung der Staubbildung konnten [Gail und Sedlmayr 1987] erstmals mit Hilfe eines selbstkonsistenten Modells unter anderem die Endgeschwindigkeit von C-Stern-Winden reproduzieren. Diese Modelle wurden inzwischen noch deutlich verfeinert ([Fleischer *et al.* 1992], [Krüger und Sedlmayr 1996] u.a.), so daß man davon ausgehen kann, daß hauptsächlich der Staub durch Absorption von Strahlung und Reibung mit dem Gas den beobachteten starken Wind bei diesen Objekten antreibt.

Für M-Sterne dagegen konnte eine selbstkonsistente Modellierung noch nicht erfolgreich durchgeführt werden, denn während in C-Sternen praktisch das gesamte O in (sehr stabilen) CO-Molekülen gebunden ist und das übrige C zur Kondensation von Staub zur Verfügung steht, ist in M-Sternen praktisch alles C in CO-Molekülen enthalten, so daß keine organischen Moleküle zur Kondensation zur Verfügung stehen. Stattdessen wurde z.B. von [Gail und Sedlmayr 1986] vorgeschlagen, daß Staubkeime aus MgS, Fe, SiO kondensieren. Der daraus entstehende Staub scheint jedoch nicht stark genug von der Strahlung beschleunigt zu werden, um allein einen starken Wind zu treiben. Offensichtlich wurde also durch die in den bisherigen Modellen enthaltenen Näherungen ein bei M-Sternen wichtiger physikalischer Prozess vernachlässigt. Hierfür gibt es nun einige Kandidaten: Zusätzlich zur Strahlungsbeschleunigung könnte z.B. der Wind durch Dissipation von akustischen oder Alfén-Wellen getrieben werden ([Buchhammer 1994]).

Eine in allen Modellen enthaltene Annahme ist die des chemischen Gleichgewichts in den inneren Bereichen der Hüllen, obwohl schon [Goeres *et al.* 1988] zeigen konnte, daß dies keineswegs zutrifft. Detaillerte Studien von [Beck *et al.* 1992] zeigen, daß bei M-Sternen mit UV-Strahlungsfeld durch eine Corona oder Schocks der Zustand der Materie weit entfernt vom chemischen Gleichgewicht ist. Insbesondere die gefundene Möglichkeit, signifikante Mengen von CO zu photodissoziieren, könnte einen wesentlichen Einfluß auf die Staubbildung und somit auch auf die gesamte Struktur der Hülle haben.

Die Einbeziehung von Nichtgleichgewichtschemie in die vorhandenen selbstkonsistenten Modelle gestaltet sich jedoch relativ schwierig. Abgesehen davon, daß viele Ratenkoeffizienten in dem extremen Druck– und Temperaturbereich von ZSHn nicht genau bekannt sind, vervielfacht sich die Dimension des zu lösenden Differentialgleichungssystems (DGLSs), wenn man die algebraischen Gleichungen, die die Konzentrationen der Spezies im chemischen Gleichgewicht beschreiben, durch Differentialgleichungen (DGLen) zur Beschreibung im Nicht-Gleichgewicht ersetzt. Außerdem haben diese DGLen die Eigenschaft steifen Lösungsverhaltens, die ein implizites oder semi-impizites Lösungsverfahren für die DGLen nötig macht (siehe Abschnitt 2.2.1). Diese Verfahren sind aber nicht wünschenswert weil sie

eine grundlegende Umstrukturierung der bestehenden Simulationsprogramme erfordern würden (und dabei auf viel Erfahrung im Umgang mit expliziten Methoden verzichtet werden müßte), bei extrem ungünstigen Anfangsbedingungen und in der Nähe von Singularitäten (z.B. in der Windgleichung) weniger stabil sind als explizite und

bei großer Dimension des Problems langsamer als explizite sind.

Die klassische Lösung der Problems der Steifheit und der zuviel Rechenaufwand erfordernden großen Dimension des DGLSs würde in der Substitution des Ratennetzwerks durch ein vereinfachtes, nicht-steifes niedrigerer Dimension bestehen. Allerdings kann solch eine Näherung immer nur mit viel Erfahrung für spezielle äußere Bedingungen durchgeführt werden. Da aber in stellaren Winden Druck und Temperatur um viele Größenordnungen variieren, und das bei Auftreten von Stoßwellen auch noch in extrem kurzen Zeiträumen, kann kein einzelnes vereinfachtes Ratennetzwerk als gültige Näherung für den gesamten Bereich angenommen werden. Viele einzelne vereinfachte Ratennetzwerke für alle auftretenden Situationen zu bestimmen, ist mit den klassischen Methoden (siehe Abschnitte 2.3.4 und 2.3.5) jedoch zu aufwendig.

1.2 Zielsetzung und Aufbau dieser Arbeit

Es existiert jedoch ein Verfahren, genannt *Computational Singular Perturbation* (CSP), zur automatisierten „Entsteifung" von gewöhnlichen Differentialgeleichungssystem. Es wurde vor einigen Jahren zur Simulation von terrestischen Verbrennungsprozessen in Princeton entwickelt ([Lam und Goussis 1991]). Ich werde dieses Verfahren im Kapitel 3 genauer beschreiben. Zusätzlich ist sogar eine Implementierung des Verfahrens bei den Autoren auf Anfrage zugänglich.

Wenn CSP anwendbar ist und numerisch stabil genug funktioniert, sollte es möglich sein, mit seiner Hilfe eine detaillierte Behandlung chemischer Reaktionsraten als zusätzliches Modul in die bestehenden selbstkonsisten Modelle einzugliedern. Damit kann man dann genauer abschätzen, welchen Einfluß die Abweichung vom chemischen Gleichgewicht in M-Stern-Hüllen auf die Staubkondensation und den Wind hat.

Es ist nun das Ziel dieser Arbeit, CSP und die vorhandene Implementierung, CSP91, auf ihren Nutzen im Kontext der selbstkonsistenten Modellierung von AGB-Stern-Hüllen hin zu evaluieren und eventuell nötige Anpassungen zu identifizieren und durchzuführen.

Im nächsten Kapitel werde ich die Grundlagen der chemischen Reaktionskinetik zusammenfassen und anschließend im Kapitel 3 das theoretische Konzept von CSP einführen. Der Aufbau des Codes CSP91 aus Princeton wird im ersten Teil des Kapitels 4 dargestellt. Im zweiten Teil soll dieser Code getestet und sein Nutzen für die Simulation der Chemie von ZSHn diskutiert werden. Dabei hat sich die Notwendigkeit einer Neuimplementierung ergeben, die Kapitel 5 beschrieben wird. Schließlich sollen einige Tests der Implementierung im Kapitel 6 vorgestellt werden.

Kapitel 2

Modellierung der Chemie

2.1 Chemische Reaktionskinetik

Wir betrachten ein homogenes chemisches System mit n möglichen Spezies, zwischen denen R elementare [1] Reaktionen ablaufen. Jede dieser Reaktionen läßt sich in der Form

$$\sum_{i=1}^{n} s^E_{r,i} X_i \rightleftharpoons \sum_{i=1}^{n} s^P_{r,i} X_i \ , \quad r = 1 \ldots R \tag{2.1}$$

schreiben, wobei r die Reaktion und i die Spezies identifiziert. X_i ist das chemische Symbol der i-ten Spezies. $s^E_{r,i}$ und $s^P_{r,i}$ bezeichnen die stöchiometrischen Koeffizienten der Edukte und Produkte.

2.1.1 Kinetische und thermodynamische Kenngrößen

Stöchiometrische Koeffizienten

Aus der Differenz der stöchiometrischen Koeffizienten $s^E_{r,i}$ und $s^P_{r,i}$ der Edukte und Produkte ergeben sich die (generalisierten) stöchiometrischen Koeffizienten $s_{r,i}$ der r-ten Reaktion:

$$s_{r,i} = s^P_{r,i} - s^E_{r,i} \ . \tag{2.2}$$

Reaktionsraten

Da wir elementare Reaktionen betrachten, gehorchen die Reaktionsraten F^+_r und F^-_r von Hin- und Rückreaktion dem Massenwirkungsgesetz ([Laidler 1970]):

$$F^+_r = k^+_r \prod_{i=1}^{n} [X_i]^{s^E_{r,i}} \tag{2.3}$$

$$F^-_r = k^-_r \prod_{i=1}^{n} [X_i]^{s^P_{r,i}} \ . \tag{2.4}$$

$[X_i]$ bezeichnet dabei die Konzentration der i-ten Spezies, und die Proportionalitätsfaktoren k^+_r und k^-_r heißen Ratenkoeffizienten der Reaktionen.

[1] Elementare Reaktionen sind solche, bei denen die Edukte in einem einzelnen Schritt über einen einzigen aktivierten Komplex in die Produkte umgewandelt werden.

Die Temperaturabhängigkeit der Ratenkoeffizienten wird durch das empirisch gefunden *Arrheniusgesetz* beschrieben:

$$k^{\pm} = A \exp\left(-\frac{E_r^{\pm}}{RT}\right) , \tag{2.5}$$

wobei $E_r^{\pm}$ die Aktivierungsenergie und A den Frequenzfaktor bezeichnen. In einer mikroskopischen Beschreibung ist die Aktivierungsenergie die bei der betrachteten Reaktion zur Bildung des aktivierten Komplexes nötige Energie. Der Frequenzfaktor kann schwach von der Temperatur abhängen:

$$A = BT^{\alpha} \tag{2.6}$$

mit $B =$ konst., $\alpha =$ konst. und $-1 < \alpha \leq 2$.

Die (Netto-)Reaktionsrate F_r einer Reaktion ergibt sich analog den generalisierten stöchiometrischen Koeffizienten aus der Differenz der Raten von Rück- und Hinreaktion:

$$F_r = F_r^+ - F_r^- \; . \tag{2.7}$$

Chemisches Gleichgewicht

Im chemischen Gleichgewicht stellt sich *detailed balancing* ein, so daß die Nettoreaktionsrate verschwindet und man die Gleichgewichtskonstante K_r für die r-te Reaktion reaktionskinetisch als Quotient der Hin- und Rückreaktionsratenkoeffizeinten bestimmen kann:

$$K_r \equiv \frac{\prod_{i=1}^{n} [X_i]^{s_{r,i}^P}}{\prod_{i=1}^{n} [X_i]^{s_{r,i}^E}} = \frac{k_r^+}{k_r^-} = \mathrm{konst.}\,. \tag{2.8}$$

Andererseits läßt sich die Gleichgewichtskonstante durch eine thermodynamische Betrachtung mit Hilfe der Boltzmanngleichung aus der Änderung der freien Enthalpie ΔG_r der Reaktion, der universellen Gaskonstanten R und der Temperatur T bestimmen:

$$K_r = \exp\left(-\frac{\Delta G_r}{RT}\right) , \tag{2.9}$$

so daß jeder Ratenkoeffizient durch denjenigen seiner Rückreaktion ausgedrückt werden kann:

$$k_r^+ = k_r^- \exp\left(-\frac{\Delta G_r}{RT}\right) . \tag{2.10}$$

Dabei berechnet sich die Änderung der freien Enthalpie aus den Differenzen ΔH_r und ΔS_r der Enthalpien und Entropien der Produkte und Edukte der Reaktion: $\Delta G_r = \Delta H_r - T\Delta S_r$ (z.B. [Schöll 1993]).

2.1.2 Modellbildung

Darstellung im linearen Raum der Konzentrationen

In dem n-dimensionalen linearen Raum, der durch die Spezieskonzentrationen aufgespannt wird, sei der Zustand des chemischen Systems durch den Vektor $\mathbf{y}$ bestimmt:

$$\mathbf{y} \equiv \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \equiv \begin{pmatrix} [X_i] \\ \vdots \\ [X_N] \end{pmatrix} . \tag{2.11}$$

Dabei werde ich im folgenden immer y_i mit der Konzentration $[X_i]$ der Spezies X_i identifizieren.

Die Richtung, in die der Zustand durch eine chemische Reaktion geändert wird, wird dann von dem stöchiometrischen Vektor $\mathbf{s}_r$ angegeben:

$$\mathbf{s}_r \equiv \begin{pmatrix} s_{r,1} \\ \vdots \\ s_{r,N} \end{pmatrix} . \tag{2.12}$$

Die Stärke der Änderung ist durch die Nettoreaktionsrate F_r bestimmt. So ergibt sich für den Einfluß der r-ten Reaktion das DGLS

$$\dot{\mathbf{y}} \equiv \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{y} = F_r \mathbf{s}_r \tag{2.13}$$

und für das gesamte System mit R Reaktionen:

$$\dot{\mathbf{y}} = \mathbf{g} \tag{2.14}$$

$$\text{mit } \mathbf{g} \equiv \sum_{r=1}^{R} F_r \mathbf{s}_r \tag{2.15}$$

Sei außerdem zu einem Zeitpunkt t_{start} der Anfangszustand $\mathbf{y}_{\text{start}}$ gegeben, dann ist das Problem vollständig durch das Anfangswertproblem

$$\dot{\mathbf{y}} = \mathbf{g} \; , \quad \mathbf{y}(t_{\text{start}}) = \mathbf{y}_{\text{start}} \tag{2.16}$$

charakterisiert.

2.2 Numerische Probleme

Die bei der numerischen Lösung des Anfangswertproblems (2.16) entstehenden Anforderungen an den Lösungsalgorithmus und die verwendete Hardware gründen sich in erster Linie in den Größen der Dimension n des Problems und der Anzahl R der chemischen Reaktionen sowie der Eigenschaft der *Steifheit*, die bei praktisch allen chemischen Systemen in den DGLen auftritt.

2.2.1 Steifheit

Diese mathematische Eigenschaft von DGLen konnte bislang ([Hairer und Wanner 1991]) noch nicht direkter als von [Curtiss und Hirschfelder 1952] definiert werden:

> Stiff equations are equations where (...) implicit methods perform (...) tremendously better (...) than explicit ones.

Man kann jedoch feststellen, daß steife DGLs immer Zeitskalen sehr unterschiedlicher Größenordnung enthalten. Dies manifestiert sich in der Lösung im allgemeinen durch Auftreten von Grenzschichten, in denen die Lösung auf einer kleinen Zeitskala variiert, während sie sich außerhalb mit einer viel größeren Zeitskala entwickelt.

Implizite (und auch semi-implizite) Integratoren können ihre Schrittweite der Zeitskala der Lösung am jeweiligen Punkt anpassen, während explizite Integratoren sich immer an der kleinsten in der DGL auftretenden Zeitskala orientieren müssen. Bei steifen DGLs führt das dazu, daß sie über den gesamten Integrationsbereich die Schrittweite der Grenzschicht beibehalten, so daß oft viele Tausend oder Millionen Schritte noch nicht ausreichen, um das längerfristige Verhalten des Systems zu bestimmen.

In Abbildung 2.1 sind am Beispiel der zweiten Komponente von *Robertson's example*, eines in Abschnitt 4.2.1 beschriebenen steifen DGLSs, typische Resultate von expliziten Differentialgleichungslösern dargestellt: Man erkennt gut, daß die Schrittweite sich nicht der Zeitskala der Lösung anpaßt, sondern ungefähr konstant klein bleibt. Ihr Wert ist gut geeignet, den in der anfänglichen Grenzschicht stattfindenden schnellen Anstieg der Konzentration aufzulösen. Darüberhinaus ist als weiteres typisches Kennzeichen zu erkennen, daß die Schrittweite quasi unabhängig von der geforderten Genauigkeit der Lösung ist.

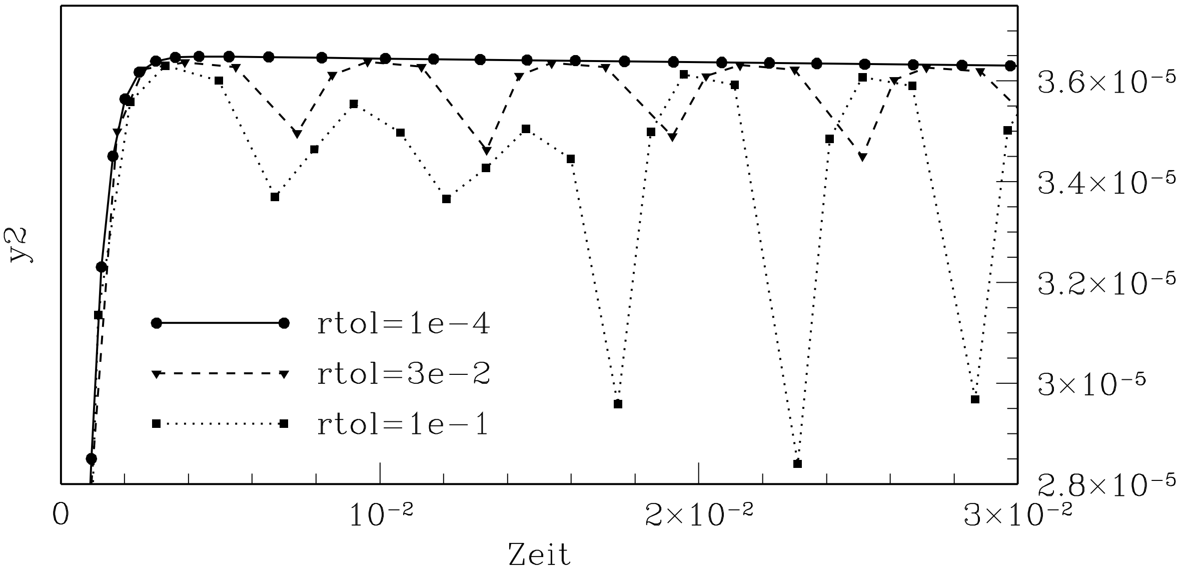


Abbildung 2.1: Mit `Dopri5` ([Hairer *et al.* 1987]) berechnete Lösungen unterschiedlicher relativer Genauigkeit `rtol` der zweiten Komponente von *Robertson's example* (absolute Genauigkeit $\mathtt{atol} = 10^{-12}$).

Per Definition kann das Problem „Steifheit" also durch Verwendung eines impliziten Integrators umgangen werden. Da dieses Integrationsverfahren auf einer Analyse der Jakobimatrix

$$\mathbf{J} \equiv \frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{y}} \tag{2.17}$$

beruht, gewinnt ein Programmpaket aber an Komplexität, und die Dimension des Problems kann das Verfahren möglicherweise zu sehr verlangsamen.

2.2.2 Anzahl der Spezies

Durch eine Berücksichtigung des chemischen Ratennetzwerkes kann die Dimension des DGLs zur Beschreibung einer ZSH um ein ganze Größenordnung angehoben werden: Beispielsweise hat das von [Krüger und Sedlmayr 1994] behandelte System die Dimension sechs, zu denen mindestens 50 weitere Dimensionen für die Konzentrationen der chemischen Spezies hinzukämen. Um die damit verbundene Steigerung der benötigten Rechenzeit zu minimieren, wäre die Bestimmung einer dimensionsreduzierten Näherung des Ratennetzwerkes erstrebenswert: Dabei würden nur die Konzentrationen weniger ausgewählter Spezies tatsächlich integriert und die übrigen algebraisch aus den integrierten berechnet.

Beim Einsatz eines impliziten Integrators ist die häufige Berechnung der Jakobimatrix erforderlich, bei einem semiimpliziten die Berechnung einer Näherung der Jakobimatrix ([Lüttke 1995]). Dabei geht die Dimension des Systems quadratisch ein. Bei einer großen Anzahl von zu berücksichtigenden Spezies, kann dies zu einem dominierenden Teil der Berechnung werden, so daß hier der Dimensionsanstieg noch kritischer ist als bei der Verwendung expliziter Löser.

Auch bei der Beschreibung des Staubwachstums durch ein System von Ratengleichungen entsteht eine vergleichbare „Explosion“ der Dimension des DGLs. Aufgrund der Analogie zu chemischen Ratennetzwerken können hier möglicherweise dieselben numerischen Methoden angewendet werden.

2.2.3 Anzahl der Reaktionen

Da viele der in (2.15) berücksichtigten Reaktionen verschwindend kleine Raten haben, könnte durch Vernachlässigung dieser Reaktionen die Berechnung der globalen Reaktionsrate deutlich verschnellert werden.

In ZSHs variieren die physikalischen Bedingungen wie Temperatur, Dichte und Strahlungsfeld jedoch so stark, daß in unterschiedlichen Bereichen sehr unterschiedliche Reaktionen dominieren und nicht *a priori* bestimmt werden kann, welche Reaktionen wo wichtig sind, so daß zunächst sehr viele (z.B. 556 in [Beck *et al.* 1992]) Reaktionen unterschiedlicher Wichtigkeit berücksichtigt werden müssen.

2.3 Konventionelle Näherungsmethoden

Es gibt vielfältige Methoden, um die globale Reaktionsrate (2.15) so zu modifizieren, daß das Anfangswertproblem (2.16) mit vorhandenen Integratoren gelöst werden kann. Dies geschieht durch eine „Entsteifung“ der DGL oder bzw. und durch eine Reduktion der Dimension mit Hilfe algebraischer Beziehungen zwischen einzelnen Komponenten y_i.

Die ersten drei Methoden (Abschitte 2.3.1—2.3.3) werden in [Goeres *et al.* 1988], die übrigen (Abschnitte 2.3.4 und 2.3.5) in [Williams 1985] eingehend dargestellt.

2.3.1 „Frozen Chemistry“

Die radikalste Methode der Näherung nimmt an, daß gar keine chemischen Reaktionen stattfinden: $F_r^{\pm} = 0$ für alle r. Dann verschwindet natürlich auch die globale Reaktionsrate: $\mathbf{g} = 0$ und die Lösung des Anfangswertproblems ist trivial beschreibbar: Das System behält seine anfängliche chemische Zusammensetzung $\mathbf{y}_{\mathrm{start}}$ bei.

Diese Näherung ist gerechtfertigt, wenn die Reaktionsraten so klein werden, daß der Effekt der Reaktionen in einem relevanten Zeitraum, z.B. der Zeitskala nicht-chemischer Prozesse die gewünschten Fehlertoleranzen unterschreitet. Die Reaktionsraten steigen nach dem Massenwirkungsgesetz (2.3) und (2.4) mit der Dichte des Systems. Deshalb kann die Näherung in einem Systems mit sehr kleiner Dichte gerechtfertigt sein.

Da die Ratenkoeffizienten von Reaktionen mit Aktivierungsenergie nach dem Arrheniusgesetz (2.5) gegen Null konvergieren, wenn die Temperatur sich dem absoluten Nullpunkt nähert, kann ein von diesen Raten dominiertes System unabhängig von seiner Dichte auch bei kleinen Temperaturen näherungsweise in den Zustand der *frozen chemistry* übergehen. Dies trifft ebenfalls auf viele Systeme mit wichtigen Reaktionen ohne Aktivierungsenergie zu, weil der Temperaturexponent α des Frequenzfaktors A, siehe (2.6), häufig positiv ist.

Die Näherung ist auch möglich, wenn nicht alle Reaktionsraten vernachlässigt werden können, sondern nur ein Teil: Dann „frieren“ diejenigen Spezies aus, die ausschließlich an den vernachlässigbaren Reaktionen beteiligt sind.

2.3.2 Gleichgewichtschemie

Auch wenn man annimmt, daß das System immer im chemischen Gleichgewicht ist, verschwindet der globale Ratenvektor, allerdings weil die Hin- und Rückraten sich im *detailed balancing* gegenseitig kompensieren: $F_r^+ = F_r^-$. In diesem Fall kann man die Gleichgewichtszusammensetzung des Systems analytisch aus den Gleichgewichtskonstanten K_r bestimmen, die wiederum mithilfe der Differenzen der freien Enthalpien berechnet werden können.

Insgesamt kann also das n-dimensionale DGL-System durch n algebraische Gleichungen für die Spezieskonzentrationen ersetzt werden.

Diese Näherung ist gerechtfertigt, wenn alle chemischen Reaktionen viel schneller ablaufen als der schnellste andere Prozeß. Insbesondere ist die Näherung nur gültig, wenn etwaige Photoreaktionen viel langsamer als die (bisher besprochenen) *kinetischen* Reaktionen sind.

2.3.3 Kinetisches Gleichgewicht

Finden Photoreaktionen und kinetische Reaktionen mit Zeitskalen ähnlicher Größe statt, während alle anderen Prozesse auf viel größeren Zeitskalen fortschreiten, so stellt sich ein Gleichgewicht zwischen Photo– und kinetischen Reaktionen ein. Es wird *kinetisches Gleichgewicht* genannt.

Die Gleichgewichtslage unterscheidet sich von der des rein chemischen Gleichgewichts und ist aufgrund der Photoraten nicht durch thermodynamische Größen bestimmbar. Stattdessen muß sie durch Relaxation des Ratennetzwerkes aller Reaktionen gefunden werden.

2.3.4 Partielles Gleichgewicht

Im Gegensatz zu den bisher dargestellten Näherungsmethoden wird das Ratennetzwerk der chemischen Reaktionen bei der *partial-equilibrium approximation* nicht durch algebraische Zustandsgleichungen ersetzt. Stattdessen wird es so modifiziert, daß es weniger steif wird und, falls gewünscht, eine niedrigere Dimension erhält.

Zur Durchführung der Näherung wird eine schnelle Reaktion q identifiziert, die sich annähernd im Gleichgewicht befindet. Dies bedeutet, daß sich die Hin– und Rückraten F_q^+ und F_q^- dieser Reaktion nahezu kompensieren. Die Näherung besteht in einer Vernachlässigung der Differenz zwischen F_q^+ und F_q^-, also der Nettorate F_q:

$$k_q^+ \prod_{i=1}^{n} (y_i)^{s_{q,i}^E} = F_q^+ = F_q + F_q^- \approx F_q^- = k_q^- \prod_{i=1}^{n} (y_i)^{s_{q,i}^P}. \tag{2.18}$$

Diese Näherungsgleichung ist eine algebraische Gleichung, die die Konzentrationen y_i der an der Reaktion q beteiligten Spezies miteinander verknüpft. Sie kann zu einer Zustandsgleichung für eine der an der Reaktion beteiligten Spezies umgeformt werden. Die Konzentration dieser Spezies, X_j, wird dann durch die Konzentrationen der anderen Spezies, $X_{i\neq j}$, ausgedrückt:

$$y_j \approx y_j(y_{i\neq j}). \tag{2.19}$$

Nimmt man diese Zustandsgleichung in die Beschreibung des Systems auf, dann kann die DGL für die j-te Spezies ignoriert werden, womit sich die Dimension des DGLSs um eins erniedrigt. Dieses DGLS ist allerdings noch genauso steif wie das ursprüngliche, weil es immer noch die Reaktionsrate $F_q = F_q^+ - F_q^-$ enthält.

Obwohl die Nettoreaktionsrate F_q vernachlässigbar gegenüber ihren Hin– und Rückraten ist, wird nichts über ihre Größe im Verhältnis zu den Nettoraten anderer Reaktionen angenommen oder impliziert; sie kann also durchaus einen entscheidenden Beitrag zur globalen Reaktionsrate g darstellen und ist in (2.15) nicht vernachlässigbar.

Zur Eliminierung von F_q aus den $n-1$ verbleibenden DGLn, wird durch Differenzieren von (2.19) eine Näherung $\tilde{g_j}$ der Konzentrationsänderung g_j bestimmt. Die Näherung enthält die vernachlässigte Reaktionsrate F_q nicht:

$$\tilde{g_j} = \frac{\mathrm{d}}{\mathrm{d}t} y_j(y_{i\neq j}). \tag{2.20}$$

Setzt man diese Näherung mit dem exakten g_j aus (2.15) gleich, so erhält man eine Näherung, die F_q nicht mehr aus F_q^+ und F_q^- bestimmt:

$$\tilde{g_j} \approx g_j \tag{2.21}$$

$$\Longleftrightarrow F_q \approx F_q(y_{i\neq j}, g_{i\neq j}). \tag{2.22}$$

Wenn man nun in den $n-1$ verbleibenden DGLn F_q durch diese Näherung substituiert, dann wird das DGLS tatsächlich weniger steif.

Da $F_q(y_{i\neq j}, g_{i\neq j})$ i.a. Terme g_i, also Ableitungen, enhält, muß das neue System durch Auflösen nach allen g_i bestimmt werden. Dabei entsteht ein System mit veränderten stöchimetrischen Vektoren $\tilde{\mathbf{s}_r}$ und Reaktionsraten $\tilde{F}_r$:

$$\dot{\mathbf{y}} = \sum_r^{R-1} \tilde{F}_r \tilde{\mathbf{s}}_r. \tag{2.23}$$

Aus diesem DGLS kann schließlich ein vereinfachter Reaktionsmechnismus konstruiert werden, der als entsteifte, vereinfachte Näherung anstelle des ursprünglichen Ratennetzwerkes verwendet werden kann.

Man kann an dieser Näherungsmethode erkennen, daß das Problem der Entsteifung von DGLSn sich — wie so vieles in der Numerik — als ein Problem der Substitution der kleinen Differenz zweier großer Zahlen durch eine alternative Beschreibung herausstellt.

Dieses Näherungsverfahren ist nach [Williams 1985, S. 568] zu subtil, um direkt automatisiert und von einem Computer durchgeführt zu werden.

2.3.5 Quasistationarität

Das Näherungsverfahren der *quasi-steady approximation*, auch *steady-state approximation* genannt, setzt wie die *quasi-equilibrium approximation* an einer einzelnen Stelle des Reaktionssystems an; es wird die (kleine) Konzentrationsänderung einer Spezies durch null genähert.

Genau genommen wird die Nettoänderung g_j sogenannter reaktionskinetischer Radikale (nicht zu verwechseln mit dem gebräuchlichen Begriff der reaktionsfreudigen „Radikale“) gegenüber ihren Produktions– und Vernichtungsraten g_j^+ und g_j^- vernachlässigt:

$$\sum_{r\in[1,R],\, s_{r,j}>0} F_r s_{r,j} \equiv g_j^+ = g_j + g_j^- \approx g_j^- \equiv \sum_{r\in[1,R],\, s_{r,j}<0} F_r s_{r,j}. \tag{2.24}$$

Intermediäre Spezies einer Kettenreaktion sind (reaktionskinetische) Radikale mit deren Konzentrationsänderung die *quasi-steady approximation* gut durchgeführt werden kann. Die Näherung ist umso besser, je schneller die Produktionsreaktionen des Radikals gegenüber seinen Vernichtungsreaktionen sind, denn dadurch wird seine Konzentration und damit auch ihre Änderung klein gegenüber den Konzentrationen der Edukte und Produkte der Kettenreaktion und deren Änderungen.

Durch Umformung von (2.24) kann die Rate einer schnellen Reaktion q durch diejenigen der übrigen Reaktionen ausgedrückt werden:

$$F_q \approx F_q(F_{r\neq q}). \tag{2.25}$$

Mit diesem Ausdruck wird die Rate F_q in (2.15) ersetzt und man erhält ein DGLS, in dem der Term $F_q = F_q^+ - F_q^-$ nicht mehr vorkommt. Wenn dieser Term die Steifheit des ursprünglichen Systems verursacht hat, dann ist das so genäherte also nicht mehr steif.

Durch Sortieren nach den auftretenden Reaktionsraten erhält die rechte Seite des genäherten DGLSs eine Form, die sich als Reaktionssystem mit veränderten stöchiometrischen Vektoren $\tilde{\mathbf{s}_r}$ interpretieren läßt:

$$\dot{\mathbf{y}} = \mathbf{g} \approx \sum_{r\in[1,R],\, r\neq q} F_r \mathbf{s}_r + F_q(F_{r\neq q})\mathbf{s}_q = \sum_{r\in[1,R],\, r\neq q} F_r \tilde{\mathbf{s}}_r. \tag{2.26}$$

Durch Auflösen von (2.24) nach der Konzentration einer der an der q-ten Reaktion beteiligten Spezies, kann eine algebraische Zustandsgleichung für diese Spezies gefunden und die Dimension des DGLSs entsprechend um eins verringert werden.

Kapitel 3

Numerische Analyse durch CSP

3.1 Grundidee

Das von [Lam und Goussis 1991] vorgeschlagene Näherungsverfahren CSP stellt eine Alternative zu den im letzten Kapitel dargestellten Näherungsverfahren dar. Es wurde hauptsächlich mit dem Ziel entwickelt, die Steifheit eines DGLSs während des numerischen Integrationsvorgangs automatisch zu vermindern. Zusätzliche chemisch-physikalische Erkenntnisse (z.B. die Identifizierung reaktionskinetischer Radikale) können aus den von dem Verfahren erzeugten Daten gewonnen werden. Im Gegensatz dazu gehen solche Erkenntnisse bei den konventionellen Näherungsverfahren als Voraussetzungen ein.

Das gesamte Verfahren basiert auf einer Analyse der globalen Reaktionsrate $\mathbf{g}$ und ihrer Transformation auf eine günstige Basis. Die Analyse und ihre Vorteile werden im Folgenden eingehend erläutert.

3.1.1 Zerlegung in Moden

Die noch zu bestimmende Basis bezeichne ich mit $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ oder $\{\mathbf{a}_i\}$ und die Matrix, die diese Basisvektoren als Spalten enthält mit $\mathbf{A}$. Ihre inverse Matrix bezeichne ich mit $\mathbf{B}$ und deren Zeilen mit $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ bzw. $\{\mathbf{b}_i\}$. Da eine Basis i.a. nicht orthogonal ist, ist ihre Inverse auch nicht gleich ihrer Transponierten: $\mathbf{B} \equiv \mathbf{A}^{-1} \neq \mathbf{A}^T$.

Bezeichne $\mathbf{f}$ die Transformierte des Vektors $\mathbf{g}$ dann lauten die Transformationsgleichungen:

$$\begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \equiv \mathbf{f} = \mathbf{B}\mathbf{g} \tag{3.1}$$

$$\mathbf{g} = \mathbf{A}\mathbf{f} \equiv \sum_{i=1}^{n} f_i\, \mathbf{a}_i \tag{3.2}$$

$$\mathbf{A} \equiv (\mathbf{a}_1, \ldots, \mathbf{a}_n) \tag{3.3}$$

$$\mathbf{B} \equiv \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} . \tag{3.4}$$

Diese Summendarstellung von $\mathbf{g}$ legt eine Interpretation in Analogie zur physikalischen Darstellung (2.15) nahe: Die Basisvektoren $\mathbf{a}_i$ entsprechen abstrakten stöchiometrischen Vektoren; sie werden *(Reaktions-)Moden* genannt. Die Komponenten f_i der Moden entsprechen den Reaktionsraten und werden *Amplituden* genannt. Es wird also der transformierte Vektor der globalen Reaktionsrate mit dem Vektor der Amplituden der Reaktionsmoden identifiziert.

An der Zeitableitung dieses Vektors entlang einer Lösung $\mathbf{y}(t)$ soll nun seine zeitliche Entwicklung diskutiert werden:

$$\begin{aligned}
\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{f} &= \left(\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{B}\right)\mathbf{g} + \mathbf{B}\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{g} \\
&= \left(\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{B}\right)\mathbf{g} + \mathbf{B}\mathbf{J}\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{y} \\
&= \left(\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{B}\right)\mathbf{g} + \mathbf{B}\mathbf{J}\mathbf{g} \\
&= \left(\left(\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{B}\right)\mathbf{A} + \mathbf{B}\mathbf{J}\mathbf{A}\right)\mathbf{f}\ .
\end{aligned} \tag{3.5}$$

Dabei bezeichnet $\mathbf{J}$ die Jakobimatrix $\mathrm{d}\mathbf{g}/\mathrm{d}\mathbf{y}$ des globalen Ratenvektors bezüglich des Zustandsvektors.

Man kann nun (3.5) mit der *Ratenmatrix*

$$\mathbf{\Lambda} \equiv \left(\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{B}\right)\mathbf{A} + \mathbf{B}\mathbf{J}\mathbf{A} \tag{3.6}$$

darstellen:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{f} = \mathbf{\Lambda}\mathbf{f}\ . \tag{3.7}$$

Um die Darstellung der grundlegenden Idee des Näherungsverfahrens übersichtlich zu gestalten, soll zunächst angenommen werden, daß die Jakobimatrix $\mathbf{J}$ zeitlich konstant sei und n linear unabhängige Eigenvektoren habe. Im Abschnitt 3.5 wird das Verfahren dann auf den allgemeinen Fall erweitert.

Aufgrund der zeitlichen Konstanz der Jakobimatrix sollen auch die Basisvektoren $\mathbf{a}_i$ und $\mathbf{b}_i$ konstant sein. Dann verschwindet der erste Summand in (3.6):

$$\mathbf{\Lambda} = \mathbf{B}\mathbf{J}\mathbf{A}\ . \tag{3.8}$$

$n \times n$–Matrizen mit n linear unabhängigen Eigenvektoren werden *normalisierbar* oder *diagonalähnlich* genannt. Sie sind durch eine Ähnlichkeitstransformation diagonalisierbar ([Zurmühl 1964]):

$$\mathbf{D} = \mathbf{B}\mathbf{J}\mathbf{A} = \mathrm{Diag}(d_1, d_2, \ldots, d_n)\ . \tag{3.9}$$

Die Matrizen $\mathbf{A}$ und $\mathbf{B}$ enthalten dabei entsprechend (3.3) und (3.4) die rechten und linken Eigenvektoren $\mathbf{a}_i$ und $\mathbf{b}_i$. Durch paarweise Normierung ($\mathbf{b}_i\mathbf{a}_i \stackrel{!}{=} 1$) wird das Matrizenpaar $\mathbf{B}$, $\mathbf{A}$ zu einem Biorthonormalsystem, d.h. $\mathbf{b}_i\mathbf{a}_j = \delta_{ij}$, bestehend aus linken und rechten Eigenvektoren von $\mathbf{J}$. Führt man die Ähnlichkeitstransformation (3.9) mit den normierten Eigenvektoren aus, dann enthält die Diagonalmatrix die (reellen oder komplexen) Eigenwerte λ_i von $\mathbf{J}$, weil Eigenwerte unter Ähnlichkeitstransformationen invariant sind ([Zurmühl 1964]):

$$\mathbf{\Lambda} = \mathbf{B}\mathbf{J}\mathbf{A} = \mathrm{Diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)\ . \tag{3.10}$$

Seien also die $\mathbf{A}$ und $\mathbf{B}$ als Biorthonormalsystem der Eigenvektoren gewählt, dann entkoppeln die DGLen für die einzelnen Komponenten von $\mathbf{f}$ und werden direkt analytisch lösbar:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} f_i &= \lambda_i f_i \qquad \text{für} \quad i = 1, \ldots, n & (3.11) \\ \Rightarrow f_i(t) &= f_i(t=0) \exp(\lambda_i t) \; . & (3.12) \end{aligned}$$

3.1.2 Klassifizierung der Moden

Anhand der Eigenwerte λ_i lassen sich qualitativ verschiedene Lösung von (3.12) unterscheiden. Zudem kann mit Hilfe der der Amplitude f_i festgestellt werden, welche Moden vernachlässigt werden können. Dadurch ergeben sich folgende Klassifizierungen der Moden:

Klassifizierung nach Eigenwerten

Die Strukturen der Lösungen der einzelnen Reaktionsmoden lassen sich durch die Real- und Imaginärteile der Eigenwerte charakterisieren:

- $\underline{\mathrm{Re}(\lambda_i) < 0}$: Die Amplituden der *relaxierenden* Moden fallen exponentiell ab.
- $\underline{\mathrm{Re}(\lambda_i) > 0}$: Die Amplituden der *explodierenden* Moden steigen exponentiell an.
- $\underline{\mathrm{Re}(\lambda_i) = 0}$: Die Amplituden der *konstanten* Moden bleiben konstant.
- $\underline{\mathrm{Im}(\lambda_i) \neq 0}$: Die Amplituden der *oszillierenden* Moden oszillieren.

Dabei kann jede Mode offensichtlich genau einer der drei ersten Kategorien und zusätzlich gegebenenfalls der vierten Kategorie zugeordnet werden.

All diese Änderungen finden auf einer Zeitskala statt, die durch den Kehrwert des Betrages des jeweiligen Eigenwertes gegeben ist:

$$\tau_i = \frac{1}{\|\lambda_i\|} \; . \tag{3.13}$$

Klassifizierung nach Amplituden

Durch die Größe der Amplitude entscheidet sich, ob der lokale Einfluß einer Mode im Rahmen der vorgegebenen Toleranzen $\mathbf{y}_{\mathrm{tol}}$ (oder auch der Maschinengenauigkeit) relevant oder vernachlässigbar ist; entsprechend werden die Moden in *aktive* und *inaktive* eingeteilt.

Die Toleranz $\mathbf{y}_{\mathrm{tol}}$ berechnet sich aus den Parametern der *relativen Toleranz* `rtol` und der *absoluten Toleranz* `atol`:

$$\mathbf{y}_{\mathrm{tol}} = \mathtt{rtol} \cdot \mathbf{y} + \begin{pmatrix} \mathtt{atol} \\ \vdots \\ \mathtt{atol} \end{pmatrix} \; . \tag{3.14}$$

Es wird dabei vorausgesetzt, daß alle Komponenten von $\mathbf{y}$ positiv sind. Diese Voraussetzung ist gerechtfertigt, weil die Komponenten von $\mathbf{y}_{\mathrm{tol}}$ die Konzentrationen der Spezies beschreiben und negative Konzentrationen unphysikalisch wären.

Das Produkt der Amplitude f_i mit dem Basisvektor $\mathbf{a}_i$ gibt die Rate der Konzentrationsänderungen durch die i-te Mode an. Wenn alle Komponenten dieses Vektors multipliziert mit der aktuellen Integrationsschrittweite Δt betragsmäßig kleiner als die entsprechenden Komponenten der vorgegebene Toleranz $\mathbf{y}_{\mathrm{tol}}$ sind, kann die Mode vernachlässigt werden:

$$\left(\bigwedge_{1 \le j \le n} |(\Delta t\, f_i \mathbf{a}_i)_j| < (\mathbf{y}_{\mathrm{tol}})_j \right) \quad \Rightarrow \quad i\text{-te Mode inaktiv/vernachlässigbar.} \tag{3.15}$$

Dabei bezeichnet $(.)_j$ die j-te Komponente eines Vektors.

Von besonderem Interesse sind die relaxierenden Moden, weil aktive relaxierende Moden nach einem Zeitintervall von der Größenordnung ihrer Zeitskala, $\Delta t > \mathcal{O}(\tau_i)$, inaktiv werden. Sie werden dann *relaxiert* genannt. Dies setzt allerdings voraus, daß sich die Toleranz langsamer als mit τ_i verändert. Da die vorgegebene absolute Toleranz `atol` eine von der Lösung unabhängige untere Schranke für die Toleranz definiert, ist die Voraussetzung jedoch immer erfüllt.

Da das Inaktivitätskriterium (3.15) die aus der Analyse der Moden **nicht** hervorgehende Größe der Integrationsschrittweite Δt enthält, wird in Abschnitt 3.2 ein modifiziertes Relaxationskriterium hergeleitet werden, in dem die Schrittweite durch die Zeitskala τ_i der Mode ersetzt wird.

3.1.3 Näherungsmöglichkeiten

Zeitentwicklung des gesamten Systems

Wenn nun die Eigenwerte (und –vektoren) in (3.7) nach der Größe der Realteile in absteigender Reihenfolge geordnet sind, also $\mathrm{Re}(\lambda_1) \ge \mathrm{Re}(\lambda_2) \ge \ldots \ge \mathrm{Re}(\lambda_n)$, ist zu erwarten, daß die M Moden mit negativen Eigenwerten sukzessive relaxieren:

$$\begin{array}{rcc} t > \mathcal{O}(\tau_n) & \Rightarrow & f_n \approx 0 \\ t > \mathcal{O}(\tau_{n-1}) & \Rightarrow & f_{n-1} \approx 0 \\ \vdots & \vdots & \vdots \\ t > \mathcal{O}(\tau_{n-M+1}) & \Rightarrow & f_{n-M+1} \approx 0\ . \end{array} \tag{3.16}$$

Dem entsprechend vereinfacht sich die DGL (3.7) bei fortschreitender Zeit immer weiter. Sie hat i.a. anfangs, bei $t = 0$, die volle Form

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} , \tag{3.17}$$

während sie sich dann bei $t > \mathcal{O}(\tau_m)$ (mit $m \le M$) vereinfacht zu:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{pmatrix} f_1 \\ \vdots \\ f_{n-m} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \approx \begin{pmatrix} \lambda_1 & & & & & \\ & \ddots & & & 0 & \\ & & \lambda_{n-m} & & & \\ & & & \lambda_{n-m+1} & & \\ & 0 & & & \ddots & \\ & & & & & \lambda_n \end{pmatrix} \begin{pmatrix} f_1 \\ \vdots \\ f_{n-m} \\ 0 \\ \vdots \\ 0 \end{pmatrix} . \tag{3.18}$$

Man erkennt, daß das n-dimensionale System sich effektiv auf ein $(n-m)$-dimensionales vereinfacht, in dem nur noch die Zeitskalen $\tau_1, \ldots, \tau_{n-m}$ auftauchen. Die Lösung ändert sich dann mit der sog. *aktiven Zeitskala* $\tau_{\mathrm{akt}} = \tau_{n-m}$.

Filtern der Steifheit

Im Eigenwertspektrum von DGLSen zur Beschreibung von chemischer Reaktionskinetik treten überwiegend Eigenwerte mit negativem Realteil auf ([Maas und Pope 1992]). Dies spiegelt die Eigenschaft chemischer Systeme wider, durch Reaktionen auf den Gleichgewichtszustand hin zu relaxieren. Außerdem sind die Beträge der eventuell auftretenden Eigenwerte mit positivem Realteil in der Regel vergleichsweise klein: $\|\lambda_i\| \ll \|\lambda_n\|$, also auch $\tau_n \ll \tau_i$, für $i \leq n-m$.

Große positive Eigenwerte äußern sich in explosionsartigen Reaktionen mit entsprechend kleinen Zeitskalen, so daß die Lösung solcher Probleme auch auf den kleinen Zeitskalen variiert und eine Vernachlässigung der schnellen (explodierenden) Moden nicht sinnvoll sein kann. Vor und nach einem explosionsartigen Prozess sind dagegen die betragsmäßig großen Eigenwerte negativ und das im Folgenden beschriebene Näherungsverfahren kann zur „Entsteifung“ des DGLSs angewendet werden.

Nähert man also (3.17) durch (3.18), so eliminiert man gerade diejenigen Relaxationsprozesse aus der Beschreibung, die auf kürzeren als den aktiven Zeitskalen geschehen. Diese sind jedoch genau die Ursache für die Steifheit des DGLSs, wie in Abschnitt 2.2 dargestellt wurde.

Das heißt also, daß das DGLS (2.14) „entsteift“ werden kann, indem man die kleinen Anteile der relaxierten Moden von g herausfiltert. g setzt sich zusammen aus seiner Projektion $\mathbf{g}_{\mathrm{fast}}$ in den Unterraum der relaxierten Moden und der Projektion $\mathbf{g}_{\mathrm{slow}}$ in den dazu orthogonalen Unterraum, in dem u.a. die aktiven Moden liegen. Die Näherung nach (3.18) besteht nun gerade in der Vernachlässigung von $\mathbf{g}_{\mathrm{fast}}$:

$$\begin{aligned} \mathbf{g} &= \underbrace{\sum_{i=1}^{n-m} f_i\,\mathbf{a}_i}_{} + \underbrace{\sum_{i=n-m+1}^{n} f_i\,\mathbf{a}_i}_{} \\ &= \mathbf{g}_{\mathrm{slow}} + \mathbf{g}_{\mathrm{fast}} \\ &\approx \mathbf{g}_{\mathrm{slow}} \, . \end{aligned} \tag{3.19}$$

Praktisch kann diese Näherung recht einfach durch Multiplikation mit der Projektionsmatrix $\mathbf{Q}_{slow}$ in den Unterraum der langsamen, nicht-relaxierten Moden durchgeführt werden. Dabei kann $\mathbf{Q}_{slow}$ entweder direkt oder, wegen der Orthogonalität der Unterräume, aus der Projektionsmatrix $\mathbf{Q}_{fast}$ in den Unterraum der schnellen relaxierten Moden bestimmt werden:

$$\mathbf{g}_{\mathrm{slow}} = \mathbf{Q}_{slow}\,\mathbf{g} \tag{3.20}$$

$$= (\mathbf{I} - \mathbf{Q}_{fast})\,\mathbf{g} \tag{3.21}$$

$$\mathbf{Q}_{slow} = \sum_{i=1}^{n-m} \mathbf{a}_i \odot \mathbf{b}_i \tag{3.22}$$

$$\mathbf{Q}_{fast} = \sum_{i=n-m+1}^{n} \mathbf{a}_i \odot \mathbf{b}_i \, . \tag{3.23}$$

$\odot$ bezeichnet hier das dyadische Produkt.

Dimensionsreduktion und physikalisch-chemische Interpretation der inaktiven Moden

Die Dimensionsreduktion der Näherung (3.18) kann auch direkt zur Ersetzung von DGLen durch algebraische Gleichungen genutzt werden.

Jede Mode i führt über die Transformationsgleichung (3.1) auf eine algebraische Gleichung, durch die die Konzentration einer Spezies in Abhängigkeit der Konzentrationen der anderen und der Amplitude f_i bestimmt werden kann. Dazu wird g gemäß (2.15) durch die Raten der chemischen Reaktionen ausgedrückt. Wenn die Mode inaktiv ist, kann f_i in dieser Gleichung vernachlässigt werden, und man erhält eine Zustandsgleichung für eine der Spezies. Die Dimension des DGLSs (2.14) läßt sich dann durch Substitution einer der n DGLen für die Spezieskonzentrationen durch diese algebraische Zustandsgleichung reduzieren.

Wenn aus einer relaxierten Mode eine Zustandsgleichung bestimmt wird, dann gilt diese nur näherungsweise. Sie reproduziert in einer entsprechenden Situation der mit der konventionellen *quasi-steady* oder *partial-equilibrium approximation* (siehe Abschnitt 2.3) hergeleiteten Näherungsgleichung.

Auch Moden, die nicht nicht relaxiert sind, können inaktiv sein; so drücken sich z.B. Erhaltungsgesetze wie die Elementerhaltung in inaktiven Moden mit verschwindenden Eigenwerten und verschwindenden Amplituden aus. Dies legt eine Interpretation als „Prozesse mit unendlicher Zeitskala“ nahe. Die Bestimmung von Zustandsgleichungen aus diesen Moden verläuft genau wie die konventionelle Bestimmung von Zustandsgleichungen aus Erhaltungssätzen.

Der Umkehrschluß von verschwindenden Eigenwerten auf globale Erhaltungsgrößen ist nur im Fall einer linearen Jakobimatrix möglich, denn solche Moden können im nichtlinearen Fall auch für einen begrenzten Zeitraum auftreten und sich davor und danach anders entwickeln (siehe Abschnitt 3.5). Dann kann man von "lokalen Erhaltungsgrößen" sprechen.

3.2 Erkennung der Relaxation

Je nach Wahl der Anzahl m der vernachlässigten Moden ergeben sich unterschiedlich starke Näherungen. Zur praktischen Anwendung benötigt man nun ein Gültigkeitskriterium für diese Näherungen.

Die geforderte lokale Genauigkeit der Lösung des Anfangswertproblems (2.16) sei durch den Toleranzvektor $\mathbf{y}_{\mathrm{tol}}$ gegeben. Die Komponenten der Toleranz $\mathbf{y}_{\mathrm{tol}}$ werden als nichtnegativ vorausgesetzt.

Wenn der Eigenwert λ_i der i-ten Mode negativ ist, kann ihr verbleibender *Gesamteinfluß* $\mathbf{y}_{\mathrm{ge},i}$ bestimmt werden, indem (3.12) über das Intervall (t, ∞) integriert wird:

$$\int_t^\infty f_i(t')\,\mathrm{d}t' = \int_0^\infty f_i(t)\exp(\lambda_i t')\,\mathrm{d}t' = \tau_i f_i(t)\ . \tag{3.24}$$

Durch Multiplikation mit dem zugehörigen Basisvektor $\mathbf{a}_i$ ergibt sich dann

$$\mathbf{y}_{\mathrm{ge},i} = \tau_i f_i \mathbf{a}_i\ . \tag{3.25}$$

Bei Vernachlässigung einer Mode geben die Komponenten des Gesamteinflusses die in den einzelnen Spezieskonzentrationen entstehenden Fehler an.

Einerseits kann man jede einzelne Mode durch Vergleich von $\mathbf{y}_{\mathrm{tol}}$ und $\mathbf{y}_{\mathrm{ge},i}$ klassifizieren, andererseits kann auch die Gültigkeit der gesamten Näherung bestimmt werden:

- Wenn die Beträge aller Komponenten des Gesamteinflusses <u>einer Mode</u> kleiner als die Komponenten der vorgegebenen Toleranz sind, ist die Mode als relaxiert anzusehen und sowohl global als auch lokal vernachlässigbar:

$$\left(\bigwedge_{1\le j\le n} |(\mathbf{y}_{\mathrm{ge},i})_j| < (\mathbf{y}_{\mathrm{tol}})_j \right) \quad \Rightarrow \quad i\text{-te Mode relaxiert/vernachlässigbar} \tag{3.26}$$

- Der aus der Vernachlässigung <u>mehrerer Moden</u> resultierende globale Fehler $\mathbf{y}_{\mathrm{err}}$ ergibt sich aus der Summe ihrer einzelnen Fehler $\mathbf{y}_{\mathrm{ge},i}$. Für seine Größe läßt sich auch mit Hilfe der Projektion $\mathbf{g}_{\mathrm{fast}}$ und der aktiven Zeitskala τ_{akt} eine obere Schranke angeben:

$$|(\mathbf{y}_{\mathrm{err}})_j| = \left|\sum_{i=1}^{m} (\mathbf{y}_{\mathrm{ge},i})_j\right| \le \left|\tau_{\mathrm{akt}} \sum_{i=1}^{m} f_i(\mathbf{a}_i)_j\right| = |\tau_{\mathrm{akt}}(\mathbf{g}_{\mathrm{fast}})_j| \ . \tag{3.27}$$

  Damit erhält man als Kriterium für die Vernachlässigbarkeit der ersten m Moden:

$$\left(\bigwedge_{1\le j\le n} |\tau_{\mathrm{akt}}(\mathbf{g}_{\mathrm{fast}})_j| < (\mathbf{y}_{\mathrm{tol}})_j \right) \quad \Rightarrow \quad m \text{ Moden relaxiert/vernachlässigbar} \ . \tag{3.28}$$

Beide Relaxationskriterien sind nur für Moden mit negativen Eigenwerten sinnvoll, da ansonsten die Integration (3.24) nicht möglich ist.

3.3 Korrektur des Zustandes

Der verbleibende Gesamteinfluß $\mathbf{y}_{\mathrm{ge},i}$ ist nicht nur eine Abschätzung der Größe des gemachten Fehlers, sondern gibt auch dessen Richtung an. Deshalb besteht die Möglichkeit, den durch Vernachlässigung einer Mode entstehenden Fehler zu minimieren, indem der Zustandsvektor $\mathbf{y}$ durch einen um die verbleibenden Gesamteinflüsse der vernachlässigten Moden korrigierten Vektor

$$\mathbf{y}_{\mathrm{k}} = \mathbf{y} + \sum_{i=1}^{m} \mathbf{y}_{\mathrm{ge},i} \tag{3.29}$$

ersetzt wird.

Zusätzlich zur allgemeinen Verbesserung der Lösung kann durch wiederholtes Durchführen dieser Korrektur verhindert werden, daß sich etwaige Rundungs- oder Näherungsfehler in den vernachlässigten Moden akkumulieren und relaxierte Moden aus numerischen Gründen (im Gegensatz zu realen, aus der Chemie resultierenden) wieder aktiv werden.

Da die vernachlässigten Moden das Relaxationskriterium (3.26), bzw. (3.28), erfüllen, ist die Korrektur immer kleiner als die vorgegebene Toleranz und es gilt im Rahmen der vorgegebenen Toleranz: $\mathbf{y}_{\mathrm{k}} = \mathbf{y}$. Somit kann die Qualität der Lösung durch diesen Schritt keinesfalls negativ beeinflußt werden.

3.4 Wahl einer „geeigneten“ Basis

Das Verfahren basiert auf der Vernachlässigung der Moden, mit denen die kleinsten Zeitskalen assoziiert sind. Die Vernächlässigung einer Mode i ist nur sinnvoll, wenn die folgenden drei Bedingungen erfüllt sind:

1. Ihre Amplitude f_i ist klein.
2. Ihr Eigenwert λ_i ist negativ.
3. Die Mode wird nicht durch Kopplung an andere Moden angeregt:
   - Die Mode koppelt an keine aktive Mode, d.h. die aktiven Moden spannen einen invarianten Unterraum auf.
   - Die Kopplung an andere inaktive Moden ist schwächer als ihre Dämpfung durch ihren Eigenwert.

Mit Hilfe der ersten Bedingung kann gemäß der Relaxationsbedingung (3.26) bestimmt werden, ob eine Mode innerhalb der gewünschten Toleranz zu einem bestimmten Zeitpunkt als inaktiv angesehen werden kann.

Ist die zweite Bedingung erfüllt, dann ist eine inaktive Mode relaxiert und es kann davon ausgegangen werden, daß sie „aus sich heraus“ relaxiert bleiben wird.

Die dritte Bedingung stellt sicher, daß eine relaxierte Mode nicht durch Wechselwirkung mit anderen Moden wieder aktiv werden kann. Insbesondere darf eine zu vernachlässigende Mode nicht an eine aktive Mode ankoppeln. Eine solche Kopplung würde sich in nicht verschwindenden Elementen in dem linken unteren $m\times(n-m)$–Block der Ratenmatrix manifestieren; es muß also gefordert werden, daß dieser Block nur Nullen enthält. Die Bedingung, daß die Kopplungen der relaxierten Moden untereinander schwächer als ihre Dämpfung sein sollen, führt entsprechend zu der Bedingung, daß der rechte untere $m \times m$–Block der Ratenmatrix diagonal dominant sein soll.

Insgesamt ist eine Basis $\{\mathbf{a}_i\}$ also geeignet, wenn die Jakobimatrix durch die von ihr vermittelte Ähnlichkeitstransformation in die folgende Gestalt überführt wird:

$$\mathbf{A}^{-1}\mathbf{J}\mathbf{A} = \Lambda = \left(\begin{array}{ccc|ccc} \lambda_1 & & X & X & \cdots & X \\ & \ddots & & \vdots & & \vdots \\ X & & \lambda_{n-m} & X & \cdots & X \\ \hline & & & \lambda_{n-m+1} & & x \\ & 0 & & & \ddots & \\ & & & x & & \lambda_n \end{array}\right), \tag{3.30}$$

wobei die X beliebige Werte und die x gegenüber den Diagonalelementen derselben Zeile betragsmäßig kleine Werte darstellen.

Eine vollständige Diagonalisierung der Jakobimatrix ist also nicht nötig.

3.4.1 Eigenvektorzerlegung

Die Eigenvektorzerlegung der Jakobimatrix mit ihrer vollständigen Entkopplung aller Moden erfüllt offensichtlich die Bedingung (3.30) für alle m mit $0 \leq m \leq n$.

Das Problem, daß nicht zu jeder allgemeinen quadratischen Matrix ein vollständiger Satz von Eigenvektoren existiert, wird in der Literatur meist zwar erwähnt, dann aber nicht weiter beachtet, denn:

> (...) the N non-orthonormal eigenvectors will "usually" span the N-dimensional vector space (...). [Press *et al.* 1992, S. 457]

Da man bei der aus der chemischen Reaktionskinetik stammenden Matrix $\mathbf{J}$ nicht gewährleisten kann, daß sie tatsächlich immer diagonalähnlich ist, muß diese Annahme nach dem Aufruf einer Eigenvektorzerlegungsroutine durch Überprüfen der Biorthogonalität der berechneten rechten und linken Eigenvektoren im Nachhinein verfiziert werden.

Die Eigenwerte einer Matrix sind entweder reell oder paarweise komplex konjugiert. Bei einer vollständigen Diagonalisierung der Matrix können auch die Eigenvektoren komplexwertig sein. Komplexe Arithmetik läßt sich jedoch vermeiden, indem die Matrix lediglich auf eine blockdiagonale Form gebracht wird, bei der die Diagonale aus 1×1– und 2×2–Blöcken besteht. Diese Blöcke sind reell und haben jeweils einen der reellen bzw. eines der komplex konjugierten Paare als Eigenwert(e). Auch die Ähnlichkeitstransformation ist dann reell ([Press *et al.* 1992]). Wenn man bei der Bestimmung von m darauf achtet, daß Moden mit gleicher Zeitskala immer nur gemeinsam vernachlässigt werden, dann erfüllt auch diese *reelle Eigenvektorzerlegung* immer die Bedingung (3.30).

Bei der Bildung der Projektionen in Eigenräume muß berücksichtigt werden, daß die rechten und linken Eigenvektoren i.a. nicht jeweils orthogonal zueinander stehen: $\mathbf{A}$ und $\mathbf{B}$ sind keine orthogonalen Matrizen. Sie bilden dagegen gemeinsam ein Biorthogonalsystem: Nur wenn man als $\{\mathbf{a}_i\}$ die rechten und als $\{\mathbf{b}_i\}$ die linken Eigenvektoren wählt (oder umgekehrt), ist die Biorthogonalitätsbedingung $\mathbf{BA} = \mathbf{I}$ erfüllt. Es müssen also immer zwei Sätze von Basen berücksichtigt werden. Zusätzlich müssen die Vektoren nicht einzeln sondern jeweils als Paare von rechten und linken Eigenvektoren normalisiert werden und die „doppelten Paare" zu komplex konjugierten Eigenwertpaaren müssen nachträglich biorthogonalisiert werden, siehe Abschnitt 5.3.3.

Das Sortieren der Basis nach dem Betrag der Eigenwerte ist bei dieser Zerlegung durch eine einfache Permutationsmatrix möglich.

Der Rechenaufwand für eine vollständige Eigenvektorzerlegung mit dem QR-Algorithmus steigt kubisch mit der Dimension des Problems: $\mathcal{O}(n^3)$. Details des Algorithmus sind in [Golub und vanLoan 1989] und [Press *et al.* 1992] beschrieben.

3.4.2 Schurvektorzerlegung

Auch in der reellen Schurform einer Matrix besteht die Diagonale aus 1×1– und 2×2–Blöcken mit den reellen bzw. komplex konjugierten Eigenwerten. Die Elemente unterhalb der Diagonalen verschwinden ebenfalls wie bei der reellen Eigenvektorzerlegung, aber oberhalb der Diagonalen können sie beliebige Werte annehmen ([Golub und vanLoan 1989]).

Auch diese Zerlegung erfüllt deshalb für alle m die Bedingung (3.30). Im Gegensatz zur Eigenvektorzerlegung handelt es sich sogar um eine orthogonale Transformation; die Schurvektoren sind also orthogonal zueinander, so daß bei der Bildung der Projektionsmatrix in den invarianten Unterraum mit einem einzigen Satz von Basisvektoren gearbeitet werden kann: Es werden dann die $\mathbf{b}_i$ paarweise mit den zugehörigen $\mathbf{a}_i$ identifiziert.

Dieser einfacheren Handhabung steht das Problem gegenüber, daß das Sortieren der Matrix nach den Beträgen der Eigenwerte zwar möglich ist, dabei aber die Schurvektoren angepaßt werden müssen.

Die benötigte Rechenzeit steigt zwar ebenfalls mit der dritten Potenz der Dimension, $\mathcal{O}(n^3)$, ist aber geringer als bei der Eigenvektorzerlegung, was aus der Tatsache ersichtlich wird, daß die Bildung der Schurvektorzerlegung den ersten Schritt üblicher Eigenvektorzerlegungsroutinen bildet ([Golub und vanLoan 1989]).

[Heroth 1995] zeigt exemplarisch, daß die Schurvektorzerlegung die Aufteilung der chemischen Reaktionskinetik weniger genau beschreibt als die Eigenvektorzerlegung. Dies liegt an der durch die Elemente oberhalb der Diagonalen vermittelten Kopplung der Moden. Deshalb habe ich mich gegen die Verwendung der Schurvektorzerlegung entschieden.

3.4.3 Zerlegung nach Lam und Goussis

[Lam und Goussis 1994] schlagen einen iterativen Algorithmus vor, der abhängig von der Wahl von m den linken unteren $m \times (n-m)$–Block einer Matrix auf null reduziert. Die Eigenwerte des rechten unteren $m \times m$–Blockes konvergieren gegen die betragsmäßig größten, die des linken oberen $(n-m) \times (n-m)$–Blockes gegen die betragsmäßig kleinsten Eigenwerte der Jakobimatrix:

$$\mathbf{B}^{i+1}_{\text{fast}} = \left(\left(\frac{\mathrm{d}}{\mathrm{d}t} \mathbf{B}^{i}_{\text{fast}} \right) \mathbf{A}^{i}_{\text{fast}} + \mathbf{B}^{i}_{\text{fast}} \mathbf{J} \mathbf{A}^{i}_{\text{fast}} \right)^{-1} \left(\left(\frac{\mathrm{d}}{\mathrm{d}t} \mathbf{B}^{i}_{\text{fast}} \right) + \mathbf{B}^{i}_{\text{fast}} \mathbf{J} \right) \tag{3.31}$$

$$\mathbf{A}^{i}_{\text{fast}} \equiv \left(\mathbf{a}^{i}_{1} \cdots \mathbf{a}^{i}_{m} \right) \tag{3.32}$$

$$\mathbf{B}^{i}_{\text{fast}} \equiv \begin{pmatrix} \mathbf{b}^{i}_{1} \\ \vdots \\ \mathbf{b}^{i}_{m} \end{pmatrix} . \tag{3.33}$$

Bei allen Größen bezeichnet der rechte obere Index die Anzahl der durchgeführten Iterationen. Nach jeder Iteration sollten die Basisvektoren neu normiert werden. Es werden nur die mit den m zu vernachlässigenden Moden (diejenigen mit den betragsmäßig größten Eigenwerten) assozierten Vektoren benötigt und berechnet: Die $\mathbf{A}^{i}_{\text{fast}}$ und $\mathbf{B}^{i}_{\text{fast}}$ sind $n \times m$– und $m \times n$–Matrizen. Dadurch wird die aufwendigste Rechenoperation, die Invertierung, an einer nur m– statt n–dimensionalen quadratischen Matrix durchgeführt. Diese Rechenzeitersparnis ist möglich, weil die m Basisvektoren des schnellen Unterraums vollständig ausreichen, um nach (3.21) die Projektionsmatrix in den langsamen Unterraum zu berechnen.

Die in (3.31) enthaltenen Zeitableitungen berücksichigen eine Zeitabhängigkeit der Jakobimatrix. Dadurch wird auch die Zerlegung zeitabhängig und die Zeitableitung in (3.6) verschwindet nicht. Diese Terme werden von den Autoren jedoch als klein im Vergleich zu den übrigen Termen von (3.31) beschrieben. Im Rahmen einer Integration lassen sich deshalb aus den Zerlegungen an vorangegangenen Lösungspunkten einfach und schnell ausreichend genaue Schätzungen für die Änderung der Basis berechnen. Wenn die Jakobimatrix zeitlich konstant ist, verschwinden die Zeitableitungen in (3.31).

Dieser Iterationsalgorithmus muß im Prinzip an jedem Integrationspunkt bis zur Konvergenz durchgeführt werden. Ein Abbruchkriterium wird nicht genannt, es wurden allerdings mit nur einem Iterationsschritt pro Integrationsschritt zur Entsteifung von einem DGLS ausreichend gute CSP-Näherungen erzielt [Goussis und Lam 1992]. Dabei wurde die

Basis des vorangegangenen Integrationspunktes jeweils als Anfangsschätzung für die Iteration benutzt.

Die ursprüngliche Herleitung dieses Algorithmus aus der singulären Störungstheorie führte zur Benennung des gesamten Verfahrens als *Computational Singular Perturbation*. Inzwischen wurde von [Goussis und Lam] gezeigt, daß es sich um eine Generalisierung des *v. Misesschen Iterationsverfahrens* ([Zurmühl 1964]) zur Berechnung des betragsmäßig größten Eigenwertes einer Matrix und des dazugehörigen Eigenvektors handelt.

3.5 Verallgemeinerung auf nicht-lineare Systeme

Chemisch reaktive Systeme sind i.a. nicht linear, weil die Reaktionsrate $\mathbf{g}$ in höherer als erster Ordnung vom Zustand $\mathbf{y}$ des Systems abhängt. Deswegen ist die Jakobimatrix $\mathbf{J}$ nicht konstant und ihre Eigenwerte λ_i und die geeignete Basis $\mathbf{A}$ zur Zerlegung von $\mathbf{J}$ werden ebenfalls zeitabhängig; das System ist im allgemeinen Fall nicht analytisch lösbar.

Da die Integration (3.24) im allgemeinen Fall nicht durchführbar ist, kann der verbleibende Gesamteinfluß einer Mode nicht bestimmt werden. Somit können die Relaxationskriterien aus Abschnitt 3.2 nicht angewendet werden. Deswegen kann die zu einem Zeitpunkt entsprechend der in den vorangehenden Abschnitten dargestellten Analyse und Klassifizierung der Moden zu einem späteren Zeitpunkt unzutreffend sein: Zunächst relaxierende Moden können später explodierend werden, zunächst inaktive Moden können wieder aktiv werden und umgekehrt.

3.5.1 Extrapolation der lokalen Näherung

Ein nicht-lineares System wird behandelt, indem es an jedem berechneten Lösungspunkt linearisiert wird. Dann kann auf die jeweilige Jakobimatrix das für den linearen Fall entwickelte Analyseverfahren angewendet werden.

Da das linearisierte System in einer Umgebung des Punktes eine gültige Näherung des nicht-linearen ist, sind auch die dort bestimmte Basiszerlegung und Modenklassifikation in der Umgebung gültige Näherungen. Sie können dort zur Näherung des DGLSs durch ein entsteiftes verwendet werden: Statt der sich änderndenden langsamen und schnellen Unterräume werden Tangentialräume an diese Unterräume am Punkt der Analyse verwendet.

Mit dieser Argumentation rechtfertigt sich die Berechnung eines neuen Lösungspunktes unter Durchführung der an dem alten Lösungspunkt gültigen Näherung, also der Projektion in den langsamen Unterraum des letzten Punktes.

Ob die am letzten Lösungpunkt berechneten vernachlässigbaren Moden eine im Rahmen der vorgegebenen Toleranz gültige Näherungen der am neuen Lösungspunkt vernachlässigbaren Moden darstellen, kann überprüft werden, indem am neuen Punkt das Relaxationskriterium (3.26) bzw. (3.28) der alten Basisvektoren $\mathbf{a}_i$ überprüft wird. Die Amplituden f_i müssen dabei durch die Projektionen des neuen Ratenvektors g auf die alte Basis berechnet werden. Wenn die Relaxationskriterien auch am neuen Punkt erfüllt sind, kann durch Interpolation darauf geschlossen werden, daß die alte Analyse der Moden in dem gesamten Intervall zwischen den beiden Integrationsschritten eine gültige Näherung darstellt.

Alternativ kann auch während der einzelnen Integrationsschritte vor jeder Projektion von g in den langsamen Unterraum der Tangentialraum an diesen Unterraum neu bestimmt werden, indem die jeweilige Jakobimatrix neu zerlegt wird. Dadurch vermeidet man zwar die Verwendung von nur genäherten Basisvektoren, eine Überprüfung der Relaxation der Moden an diesen Punkten kann jedoch zu falschen Ergebnissen führen, weil die Punkte zwar in der Nähe aber nicht immer auf der Lösung liegen.

3.5.2 Iterative Zerlegungsalgorithmen

Da die geeignete Basis sich innerhalb eines Reaktionsschrittes meist nur geringfügig ändert, eignet sich für einen iterativen Zerlegungsalgorithmus das Ergebnis der vorhergehenden Zerlegung sehr gut als Anfangsschätzung. Die dadurch erreichte schnelle Konvergenz vermindert den Rechenaufwand erheblich: Auf diese Weise erreichen [Lam und Goussis 1989] mit nur einem einzigen Iterationsschritt ihres Zerlegungsalgorithmus pro Integrationsschritt gute Resultate.

3.5.3 Tabellierung der lokalen Näherung

Da die fortlaufende Berechnung der Projektionsmatrix während des Ausführens der Integration sehr rechenzeitaufwendig ist, schlagen [Maas und Pope 1992] vor, die Projektion des Reaktionsvektors g in den langsamen Unterraum zu tabellieren und während der Integration nur sehr wenige Variable, z.B. drei oder auch nur eine, für die Entwicklung des gesamten chemischen Systems zu berücksichtigen.

Dieser Ansatz birgt zwei wesentliche Nachteile. Zum einen erscheint das Erstellen einer multidimensionalen Tabelle, die die gesamten im astrophysikalischen Kontext möglichen physikalischen Bedingungen (Dichte, Temperatur, Strahlung, Kondensationsgrad, ...) umfaßt, als nicht effizient durchführbar. Zum anderen ist durch die Beschränkung auf sehr wenige Reaktionsfortschrittsvariablen eine Modellierung der z.B. in Schocks auftretenden Grenzschichten nicht möglich (siehe nächster Abschnitt). An der Behebung der zweiten Beschränkung wird zur Zeit gearbeitet ([Heroth 1995]).

Aufgrund dieser Nachteile habe ich mich gegen eine weitere Verfolgung dieses Ansatzes entschieden.

3.6 Systeme mit Variablen Koeffizienten

Ändern sich die physikalischen Bedingungen, in denen sich das chemische System befindet, so werden dadurch auch die Ratenkoeffizienten $k^{\pm}$ der Reaktionen geändert. Dies führt, genauso wie die Nichtlinearität des chemischen Systems, zu einer Zeitabhängigkeit der Jakobimatrix und ihrer Basiszerlegung. Auch aus diesem Grund ist eine regelmäßige Neuberechnung der Basis nötig.

Darüberhinaus verschiebt sich die Gleichgewichtslage des chemischen Systems durch die geänderten Ratenkoeffizienten, und es ist möglich, daß die Gleichgewichtslage sich schneller vom aktuellen Zustand des Systems entfernt als dieser sich auf die Gleichgewichtslage zu bewegt; der Abstand von der Gleichgewichtslage kann also bei Systemen mit sich ändernden äußeren Bedingungen zunehmen. Dies führt dazu, daß die Anzahl m der relaxierten Moden im Laufe der Entwicklung zeitweise abnehmen kann: Das System entwickelt sich dann wieder auf einer kleineren Zeitskala.

Allgemein kann man sagen, daß Moden, die auf einer kleineren Zeitskala als die der Änderung der äußeren Bedingungen relaxieren, den relaxierten Zustand erreichen. Ihr Gesamteinfluß auf den Zustand läßt sich dann auch im nicht-relaxierten Zustand durch (3.26) abschätzen.

Treten wie in Schocks sehr schnelle Änderungen der äußeren Bedingungen auf, so werden viele, eventuell alle, zuvor relaxierten Moden wieder aktiv und dürfen nicht mehr vernachlässigt werden. In der Lösung des Systems treten dann die für Grenzschichten typischen starken Konzentrationsänderungen auf.

Kapitel 4

Das Programm CSP91

Für diese Arbeit stand das Programm CSP91 ([Lam 1993]) als Beispiel für die Implementierung des im Kapitel 3 beschriebenen Näherungsverfahrens zur Verfügung. In diesem Kapitel werde ich zunächst den Aufbau des Programms und dann einen Test seiner Leistungsfähigkeit durchführen und mit Blick auf meine Zielsetzung aus Abschnitt 1.2 diskutieren.

4.1 Programmaufbau

Das Programmpaket besteht aus drei Modulen: Modellierung der chemischen Reaktionskinetik, eine implizite Integrationsroutine zur Erstellung von Referenzlösungen und eine explizite Integrationsroutine mit eingebauter Näherung nach der in Kapitel 3 beschriebenen Methode.

Da sich die implizite Routine `Radau5` ([Hairer und Wanner 1991]) zur Erstellung von Referenzlösungen bereits bewährt hatte, habe ich das entsprechende Modul von CSP91 nicht benutzt. Um die Eigenschaften der expliziten Integrationsroutine mit eingebauter Näherung möglichst unverfälscht untersuchen zu können, habe ich außerdem das Modul zur Modellierung der chemischen Reaktionskinetik durch ein gebräuchliches steifes Testproblem (siehe Abschnitt 4.2.1) ersetzt.

Der Aufbau des expliziten Integrators mit CSP-Näherung ist in Abbildung 4.1 dargestellt; in der Initialisierung wird u.a. die Anzahl m der vernachlässigten Moden gleich null gesetzt. Dann wird nach dem von Lam und Goussis entwickelten Verfahren (siehe Abschnitt 3.4.3) eine Basis bestimmt. Wenn nun die schnellste noch nicht vernachlässigte Mode als relaxiert gemäß 3.26 erkannt wird, wird auch sie vernachlässigt; d.h. m wird um eins erhöht. Danach muß die Basis neu bestimmt werden, weil ihre Wahl von m abhängig ist. Die Abfrage der Relaxation wird nun solange wiederholt, bis alle relaxierten Moden vernachlässigt werden.

Daraufhin wird die Schrittweite als Bruchteil der aktiven Zeitskala bestimmt: $\Delta t = \alpha\tau_{\mathrm{akt}}$, wobei der Proportionalitätsfaktor α als Parameter vorgegeben werden muß. Er hat typischerweise Werte zwischen 0.1 und 1. Dabei wird allerdings ein maximaler Vergrößerungsfaktor zwischen dem letzten und dem aktuellen Zeitschritt nicht überschritten. Es wird dann ein expliziter Runge-Kutta Integrationsschritt 4. Ordnung ausgeführt. Dabei wird bei jeder

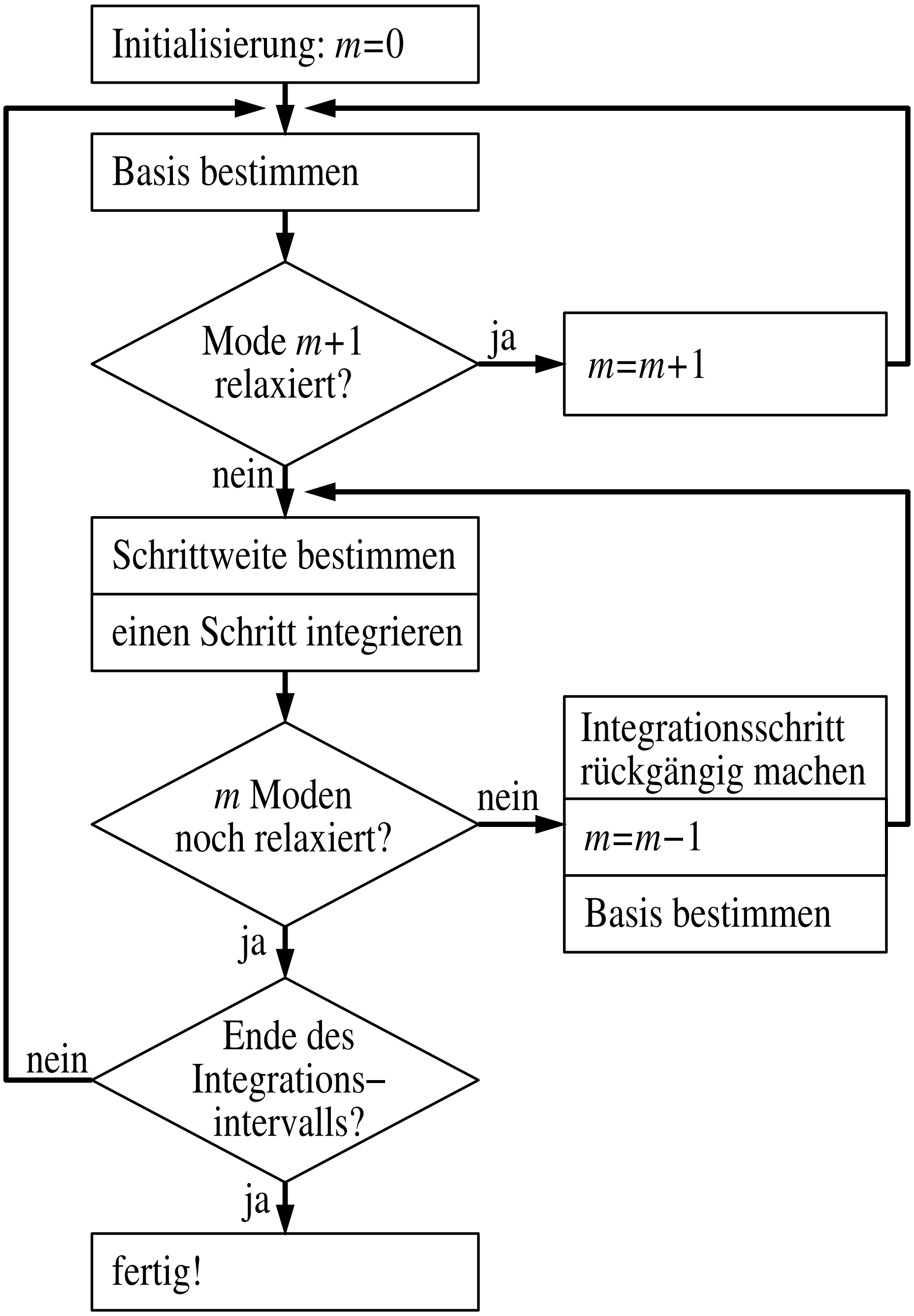


Abbildung 4.1: vereinfachtes CSP91 Flußdiagramm: m bezeichnet die Anzahl der vernachlässigten Moden. Bei der Bestimmung der Basis werden die Moden nach der Größe ihrer Eigenwerte sortiert (schnell relaxierende Moden zuerst). Die Korrektur entsprechend Abscnitt 3.3 wurde hier nicht brücksichtigt.

Berechnung der rechten Seite die Basis neu bestimmt, ebenso die Projektionsmatrix in den langsamen Unterraum. An die Integrationsroutine wird schließlich nur die Projektion der rechten Seite in den langsamen Unterraum übergeben.

Nach Ausführung des Integrationsschrittes wird nach (3.26) geprüft, ob alle vernachlässigten Moden noch relaxiert sind. Ist das nicht der Fall, dann ist die Näherung durch Projektion der rechten Seite in den langsamen Unterraum der Dimension $n - m$ während des Integrationsschrittes ungültig geworden, und der Schritt muß mit einer schwächeren Näherung wiederholt werden: Dazu wird der Schritt rückgängig gemacht, m wird um eins erniedrigt, die Basis wird neu bestimmt und die Schrittweitenbestimmung und der Integrationsschritt werden wiederholt. Wenn dann das Relaxationskriterium immer noch nicht erfüllt ist wird die Dimension des langsamen Unterraums weiter erhöht. Dies wird solange wiederholt, bis auch an dem neuen Punkt im Zustandsraum die Näherung gerechtfertigt ist und dieser als Lösungspunkt akzeptiert werden kann.

Schließlich wird geprüft, ob die Lösung das Ende des gewünschten Integrationsintervalls erreicht hat; wenn nicht, wird das gesamte Verfahren (bis auf die Initialisierung) wiederholt.

4.2 Anwendungstest

4.2.1 *Robertson's example*

Zur Diskussion der bei der Beschreibung chemischer Reaktionskinetik auftretenden steifen DGLSe wurde von [Robertson 1966] ein hypothetisches Reaktionssystem vorgeschlagen, das inzwischen unter dem Namen *Robertson's example* weite Verbreitung (z.B. [Hairer und Wanner 1991, Lüttke 1995]) als Testproblem für Löser steifer DGLen gefunden hat. Es soll auch hier verwendet werden.

Das Beispiel beschreibt die Reaktion eines Ausgangsstoffes A über eine intermediäre Spezies B zu dem Endprodukt C:

$$A \xrightarrow{0.04} B \tag{4.1}$$

$$B + B \xrightarrow{3\cdot 10^7} B + C \tag{4.2}$$

$$B + C \xrightarrow{10^4} A + C \tag{4.3}$$

Dadurch, daß die Ratenkoeffizienten (jeweils die Zahl über dem „$\longrightarrow$“) der Reaktionen sich um bis zu 9 Größenordnungen unterscheiden, können den drei Reaktionen entsprechend unterschiedliche Zeitskalen zugeordnet werden und das DGLS insgesamt wird steif. Es lautet:

$$\frac{\mathrm{d}}{\mathrm{d}t}[A] \equiv \frac{\mathrm{d}}{\mathrm{d}t} y_1 = -0.04 y_1 + 10^4 y_2 y_3 \tag{4.4}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}[B] \equiv \frac{\mathrm{d}}{\mathrm{d}t} y_2 = +0.04 y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2 \tag{4.5}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}[C] \equiv \frac{\mathrm{d}}{\mathrm{d}t} y_3 = +3 \cdot 10^7 y_2^2 \,. \tag{4.6}$$

Die von Robertson gewählten Anfangsbedingungen sind:

$$\mathbf{y}(0) \equiv \begin{pmatrix} y_1(0) \\ y_2(0) \\ y_3(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{4.7}$$

Die Lösung des Anfangswertproblems ist in den Abbildungen 4.2 und 4.3 mit linearer und logarithmischer Zeitskala aufgetragen. Sie wurde mit dem impliziten Löser /radau/ ([Hairer und Wanner 1991]) mit großer Genauigkeit (`rtol` = 10^{-12} `atol` = 10^{-15}) erzeugt. Dabei war aus Stabilitätsgründen eine Modifikation der Anfangsbedingungen zu $(1, 10^{-5}, 10^{-5})$ nötig. Dies offenbart eine der Schwächen bestehender impliziter Lösungsroutinen: Bei ungünstig gewählten Anfangsbedingungen können Konvergenzprobleme entstehen.

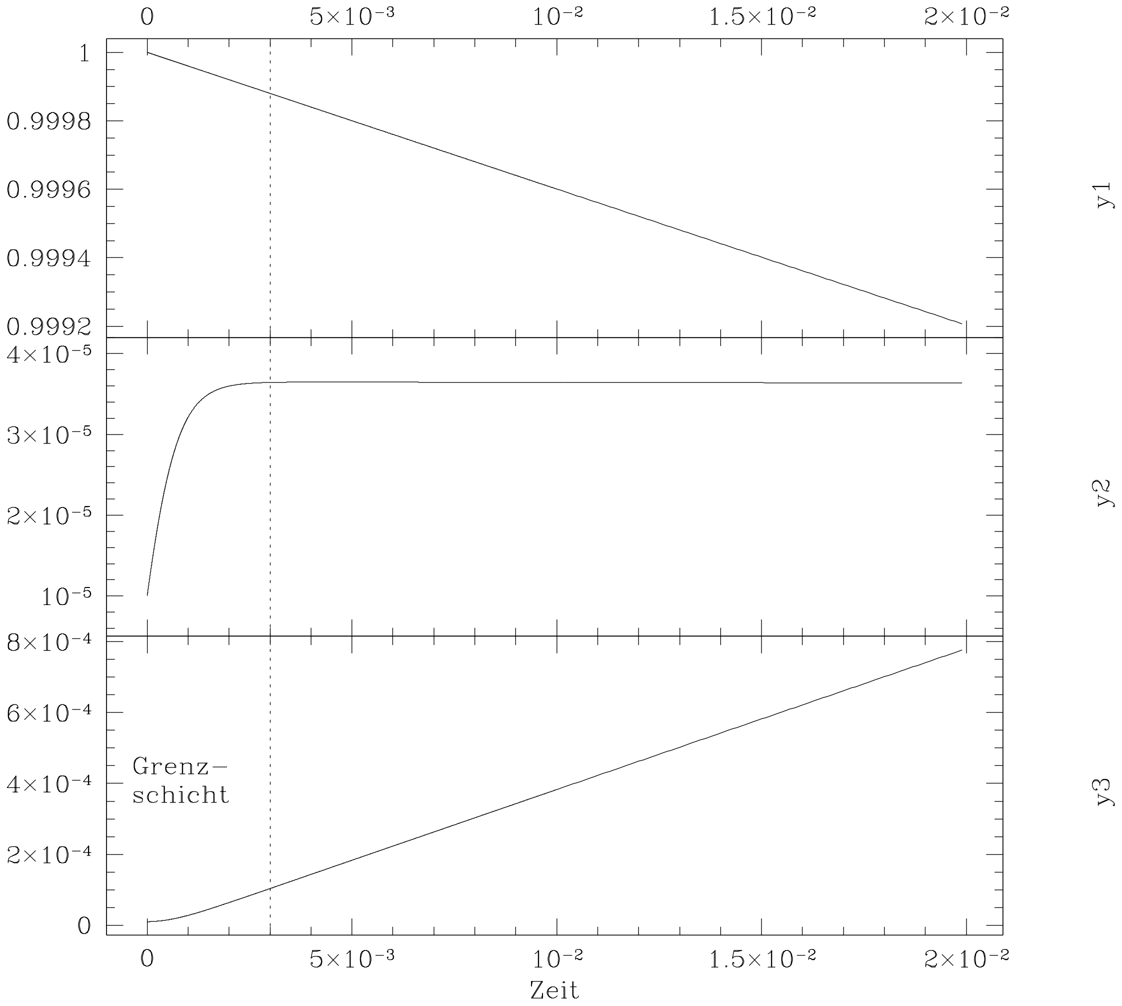


Abbildung 4.2: Lösung von *Robertson's example*, lineare Zeitachse

Die dargestellte Lösung weist in einer Grenzschicht bis ca. $t = 3 \cdot 10^{-3}$ zunächst einen sehr schnellen Anstieg der Konzentration der intermediären Spezies B auf. Darauf folgt bis ca. $t = 1$ eine lange Phase, in der die Konzentration von B nahezu konstant und vergleichsweise klein bleibt, während sich die Konzentrationen von A und C langsam ändern.

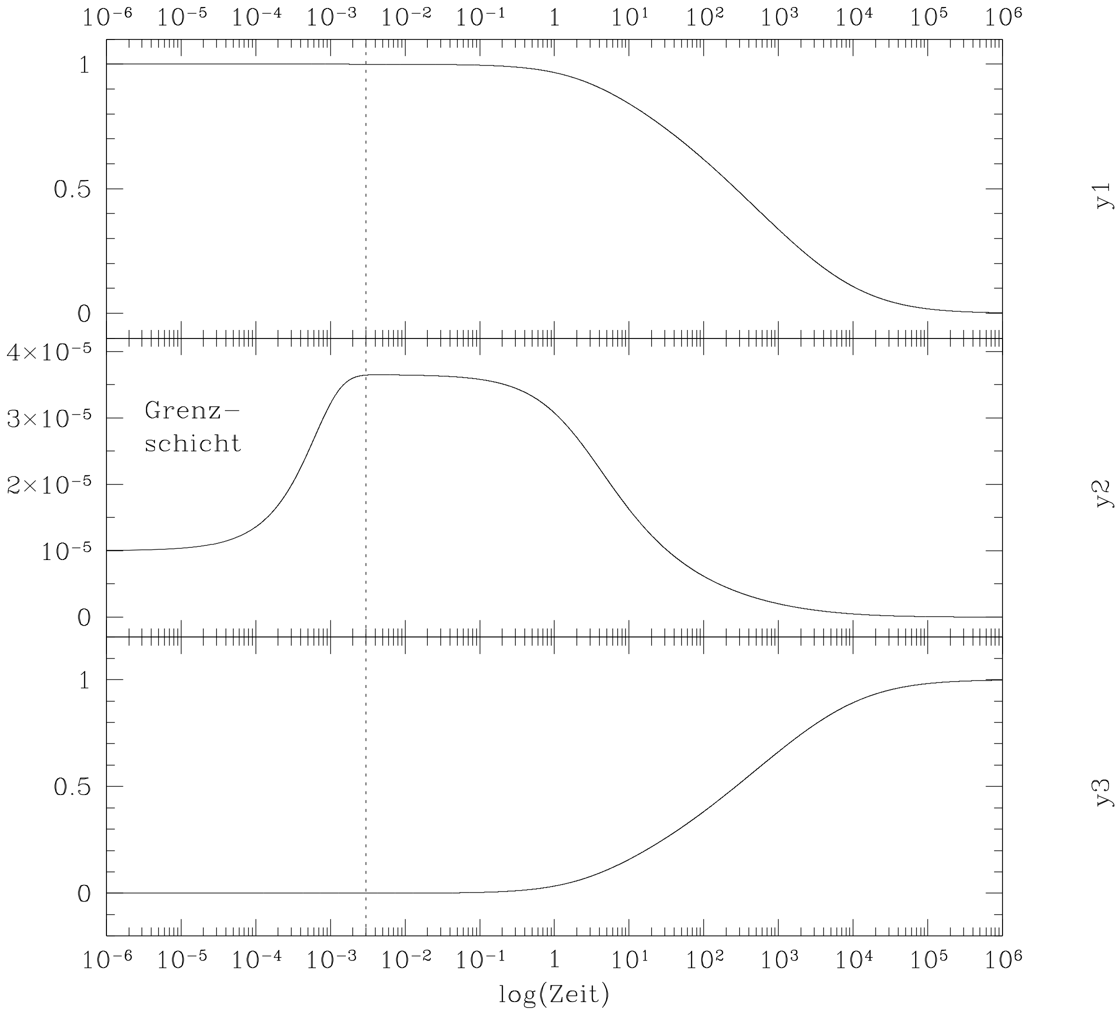


Abbildung 4.3: Lösung von *Robertson's example*, logarithmische Zeitachse

In dem Auftreten der Grenzschicht zeigt sich die Steifheit des Problems. B weist typische Kennzeichen eines (reaktionskinetischen) Radikals auf: außerhalb der Grenzschicht konstante, kleine Konzentration durch langsame Produktions– aber schnelle Abbaureaktionen. Dies läßt sich bereits an den Ratenkoeffizienten der Reaktionen (4.1) erkennen. Deshalb kann dieses Problem außerhalb der Grenzschicht konventionell durch eine Steady-State Approximation (Abschnitt 2.3.5) für B genähert und entsteift werden.

Vermutlich wegen der offensichtlichen Möglichkeit einer Steady-State Approximation wird das Problem gewöhnlich auf dem Intervall $[0, 1]$ behandelt. Im weiteren Verlauf relaxiert die Lösung auf einen Endzustand zu, in dem ausschließlich das Endprodukt C vorhanden ist.

4.2.2 Analyse der Moden von *Robertson's example*

Die Arbeitsweise des Programms CSP91 wird bei einer Anwendung zur Lösung von *Robertson's example* deutlich. Ich habe das Programm zunächst Zeitschritte von der Größe der jeweils aktuellen Zeitskala wählen lassen ($\alpha = 1$). Die so errechnete Lösung ist in Abbildung 4.4 dargestellt. Jeder berechnete Lösungspunkt ist als Dreieck eingezeichnet. Man erkennt, daß das Programm mit der Schrittweite $\Delta t = 10^{-3}$ beginnt, bis $t \approx 10^{-2}$, also in der Grenzschicht, eine relativ kleine Schrittweite beibehält und sie dann sukzessive erhöht. Die Schrittweite wird also entgegen dem Verhalten eines expliziten Lösers ohne CSP-Näherung tatsächlich der Zeitskala der Lösung angepaßt.

Zeitskalen und verbleibende Gesamteinflüsse $\mathbf{y}_{\mathrm{ge},i}$ der Moden sind in Abbildung 4.5 dargestellt. In dem Diagramm der Zeitskalen ist zusätzlich die gewählte Integrationschrittweite jeweils am Ende des Schrittes als Dreieck eingezeichnet.

Man erkennt deutlich die um einige Größenordnungen differierenden Zeitskalen der Moden. Sie bleiben vor dem Einsetzen der Relaxation des gesamten Systems auf seinen Endzustand, also bei $t \in [0, 1]$, konstant. Im Bereich der Grenzschicht fällt der Einfluß der ersten Mode schnell ab, bis er die Fehlertoleranzen bei $t = 5 \cdot 10^{-3}$ unterschreitet. Danach wird diese Mode als relaxiert erkannt und vernachlässigt. Deshalb wird die Schrittweite, die in der Grenzschicht gleich der Zeitskala der ersten Mode gewählt wurde, bis auf den Wert der Zeitskala der zweiten Mode vergrößert.

Durch die Relaxation der Mode ergibt sich eine objektive Definition für den bisher intuitiv benutzten Begriff *Grenzschicht*: Eine Grenzschicht ist derjenige Bereich des Integrationsintervalls, in dem eine relaxierenden Mode noch aktiv ist. Damit ist zu jeder relaxierenden Mode eine Genzschicht definiert. In der Grenzschicht ändert sich die Lösung nicht langsamer als mit der Zeitskala der jeweiligen Mode.

Die dritte Mode mit der größten Zeitskala läßt sich als Teilchenerhaltungsgesetz interpretieren. Ihr Eigenwert ist analytisch gleich null, die Zeitskala wäre nach (3.13) unendlich groß. Numerisch ergeben sich für den Eigenwert kleine Zahlen der Größenordnung der Maschinengenauigkeit ($\approx 10^{-16}$). Minimale Variationen des Eigenwertes werden dadurch bei der Berechnung der Zeitskala stark verkrößert. Dies äußert sich schließlich in der dargestellten großen, stark variierenden Zeitskala. Da die verbleibenden Gesamteinflüsse der Moden proportional zu ihren Zeitskalen sind, ergeben sich numerisch für die dritte Mode auch große Werte. Diese Zahlenwerte sind jedoch nicht als Gesamteinflüß der dritte Mode interpretierbar, weil die bei der Herleitung von (3.25) gemachte Vorraussetzung eines negativen Eigenwertes nicht erfüllt ist. Die Werte sind hier nur der Vollständigkeit halber eingezeichnet.

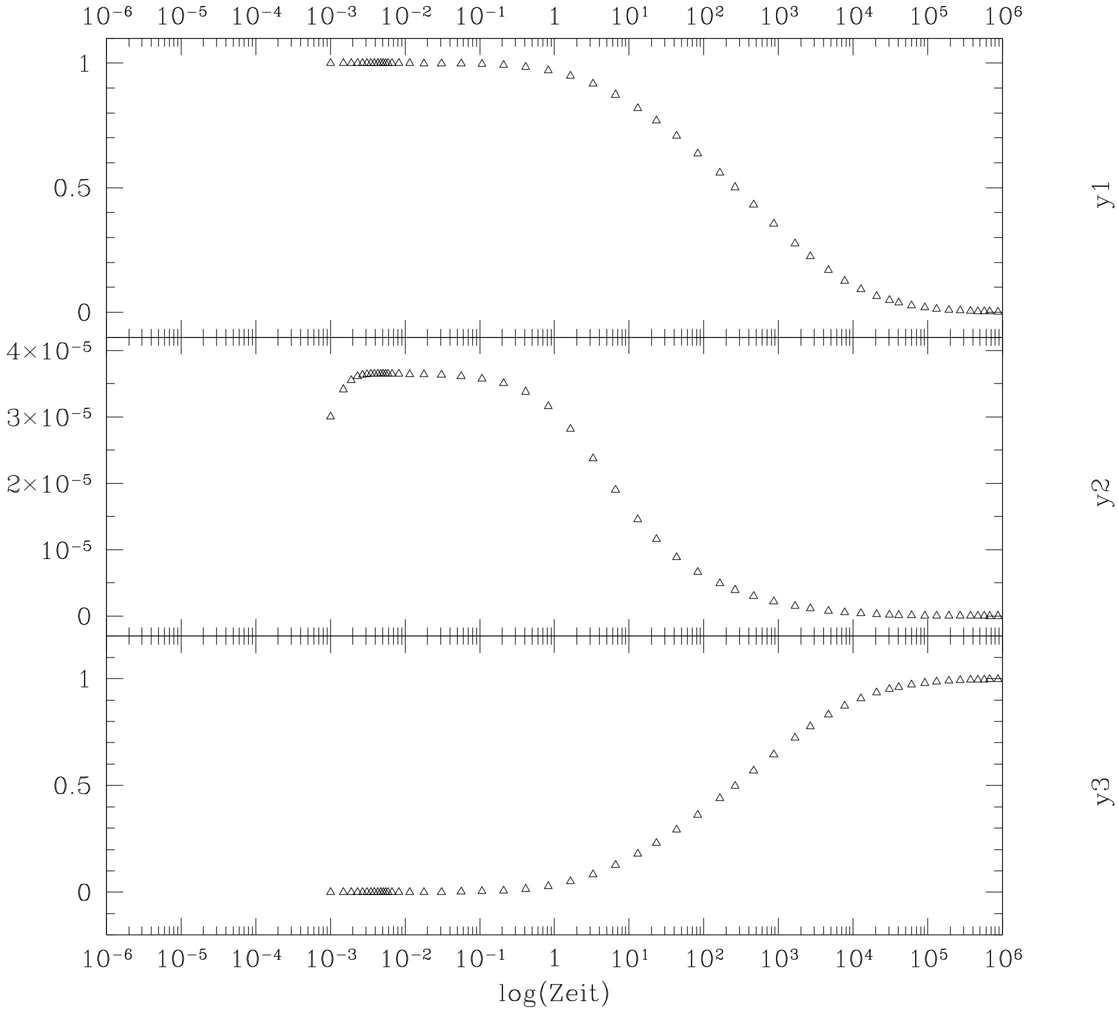


Abbildung 4.4: Lösung von *Robertson's example* mit CSP1

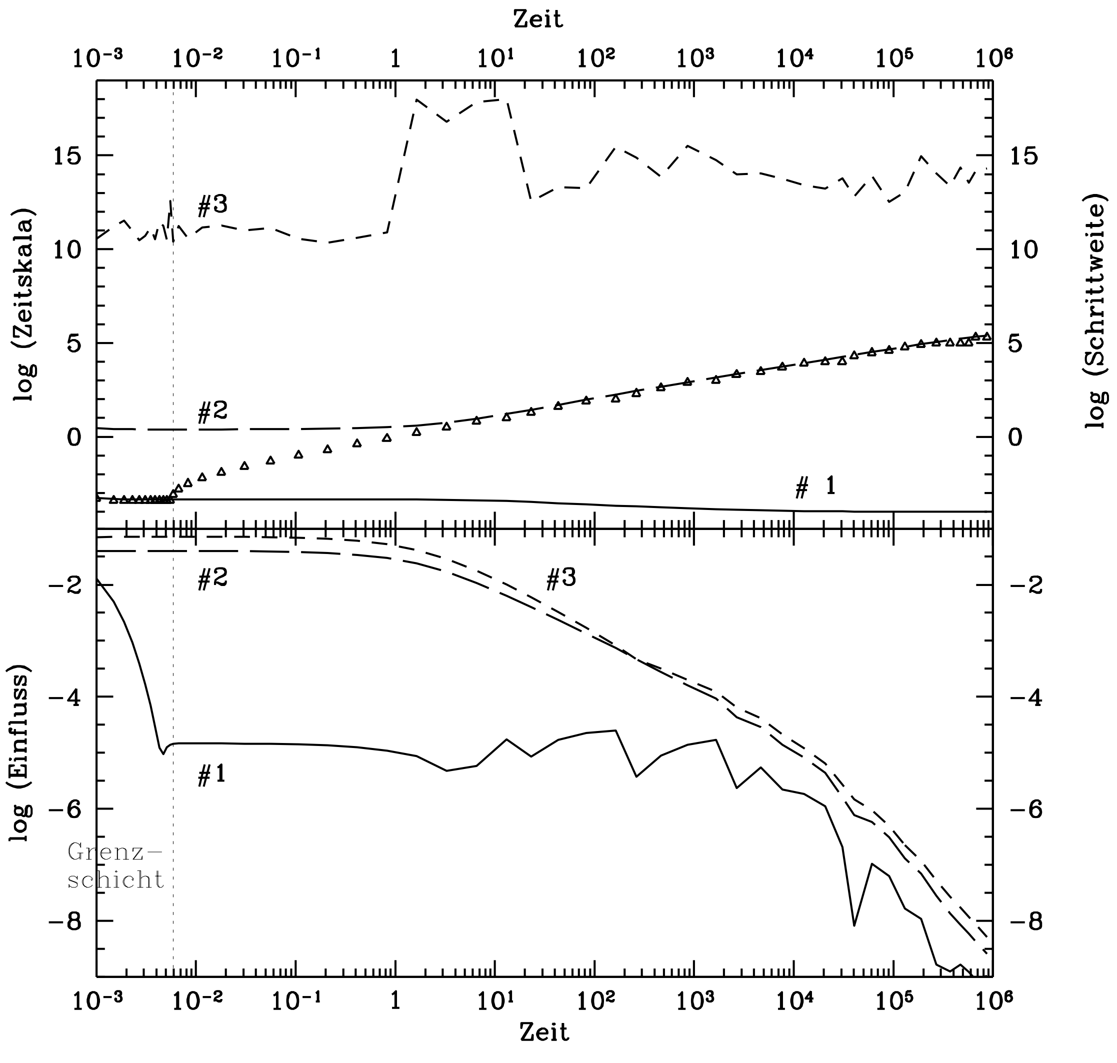


Abbildung 4.5: Moden von *Robertson's example* (CSP1). Die Dreiecke stellen die jeweilige Schrittweite des Integrators dar.

4.2.3 Vergleich von CSP91 und Radau5

Desweiteren habe ich *Robertson's example* zusätzlich mit der CSP91-Routine mit verändertem Parameter α ($= 0.1$) und mit der impliziten Lösungsroutine Radau5 (Beschreibung in [Hairer und Wanner 1991]) gelöst. Die Parameter der drei Lösungsroutinen sind in Tabelle 4.1 zusammengestellt; die beiden Läufe von CSP91 mit unterschiedlicher Parametereinstellung werde ich wie in dieser Tabelle „CSP1“ und „CSP2“ nennen. Radau5 berechnet den jeweils nächsten Lösungspunkt nach dem Runge-Kutta(RK)-Schema, allerdings nicht nach dem expliziten 4. Ordnung wie CSP91, sondern nach einem impliziten 5. Ordnung. Als Anfangsschrittweite für Radau5 wurde die eingebaute Standardgröße von 10^{-6} übernommen, während bei CSP91 dieser Parameter nicht benötigt wird, weil die Schrittweite einzig aus der aktiven Zeitskala bestimmt wird. Hierfür wurde der Parameter α auf 1 für CSP1 und 0.1 für CSP2 gesetzt. Dieser Parameter ist bei Radau5 offensichtlich nicht vorhanden. Die Toleranzen $\texttt{rtol} = 10^{-3}$ und $\texttt{atol} = 10^{-6}$ werden einheitlich vorgegeben, ebenso das Integrationsintervall. In den letzten beiden Zeilen der Tabelle sind die Anzahl der jeweils benötigten Schritte und die benötigte Rechenzeit aufgeführt.

	`Radau5`	CSP1	CSP2
Integrations-algorithmus	impliziter RK 5. Ordnung	expliziter RK 4. Ordnung	
Anfangsschrittweite	10^{-6}	—	
$\alpha = \Delta t/\tau_{\text{akt}}$	—	1	0.1
relative Toleranz `rtol`	10^{-3}		
absolute Toleranz `atol`	10^{-6}		
Integrationsintervall	$[0, 10^6]$		
Anzahl der Integrationsschritte	53	53	378
benötigte Rechenzeit	660 ms	140 ms	560 ms

Tabelle 4.1: Tabelle der Parameter der verschiedenen Integrationsmethoden

Die von Radau5 und CSP2 berechneten Lösungspunkte sind in den Abbildungen 4.6 und 4.7 analog zu Abbildung 4.4 dargestellt.

Trägt man die Schrittweiten über der Zeit auf (siehe Abbildung 4.8), so erkennt man die Grenzschicht an dem Verhalten von CSP91: Innerhalb der Grenzschicht bleibt die Schrittweite konstant, außerhalb steigt sie schnell auf Werte, die mit der von Radau5 gewählten Schrittweite vergleichbar sind. Dies ist genau auf den Einfluß der gemachten Näherung zurückzuführen, denn die von einem einfachen expliziten Integrator gewählte konstante Schrittweite würde in der Grafik als waagrechte Linie auf demselben niedrigen Niveau erscheinen, auf dem die Schrittweite von CSP1 innerhalb der Grenzschicht erscheint.

In Abbildung 4.9 können die gegenüber der im Abschnitt 4.2.1 vorgestellten Referenzlösung aufgetretenen absoluten Fehler der drei Verfahren mit den gemäß (3.14) berechneten Toleranzen $\mathbf{y}_{\text{tol}}$ verglichen werden. `Radau5` liegt erwartungsgemäß immer im Bereich der vorgegebenen Toleranzen, während CSP1 sie zwischenzeitlich leicht überschreitet. Dies kann jedoch, wie die Fehler von CSP2 zeigen, durch eine Schrittweitenwahl, die etwas kleiner als die jeweils aktuelle Zeitskala ausfällt, vermieden werden.

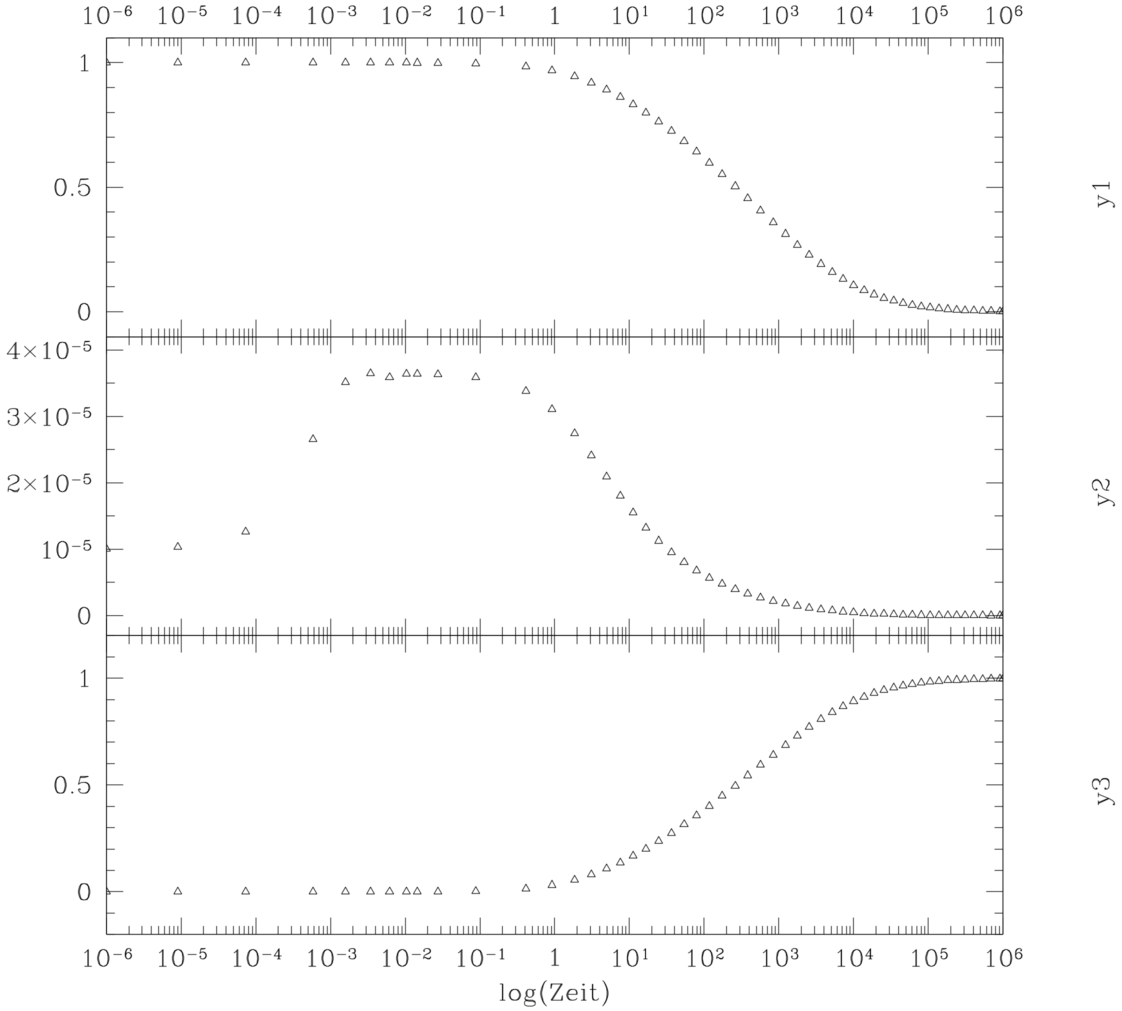


Abbildung 4.6: Lösung von *Robertson's example* mit `Radau5`

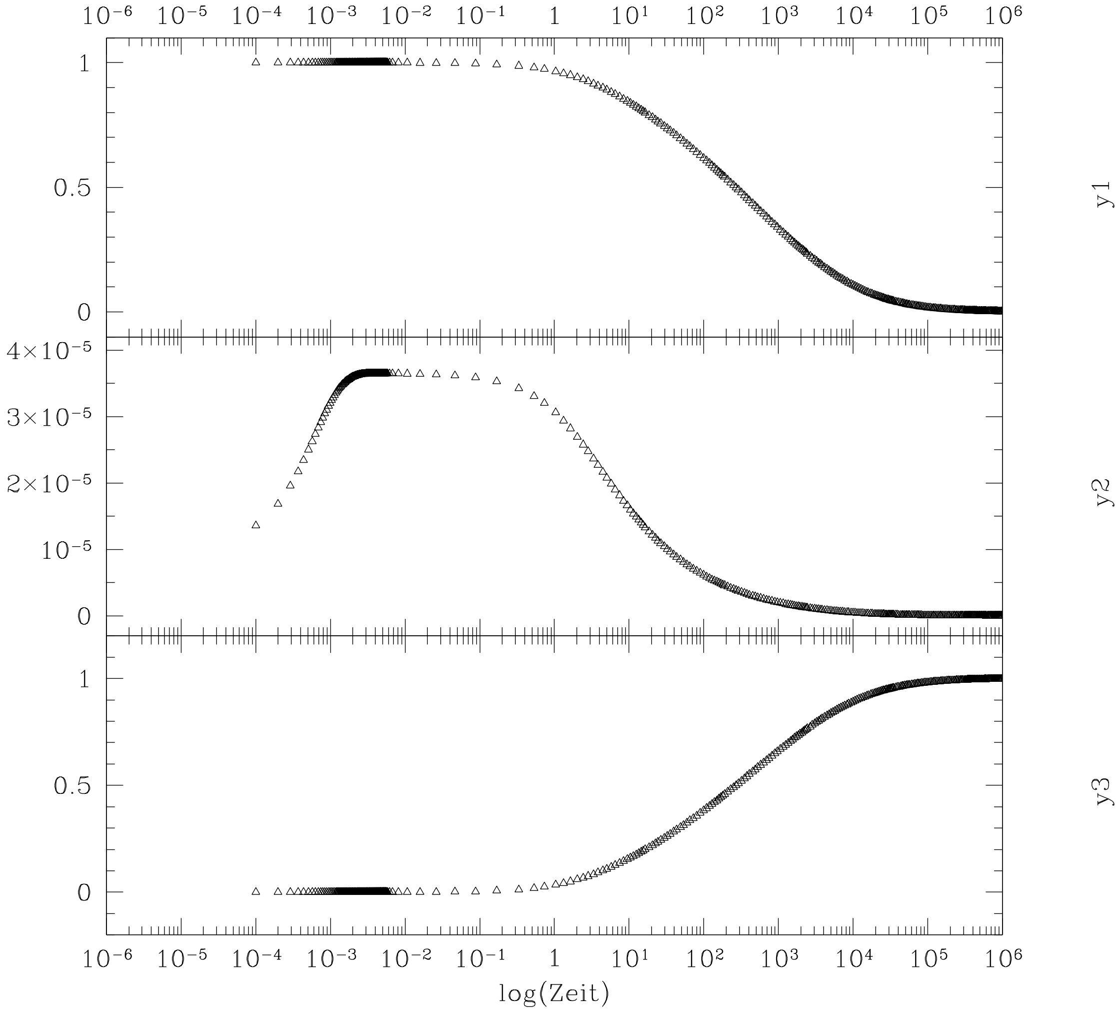


Abbildung 4.7: Lösung von *Robertson's example* mit CSP2

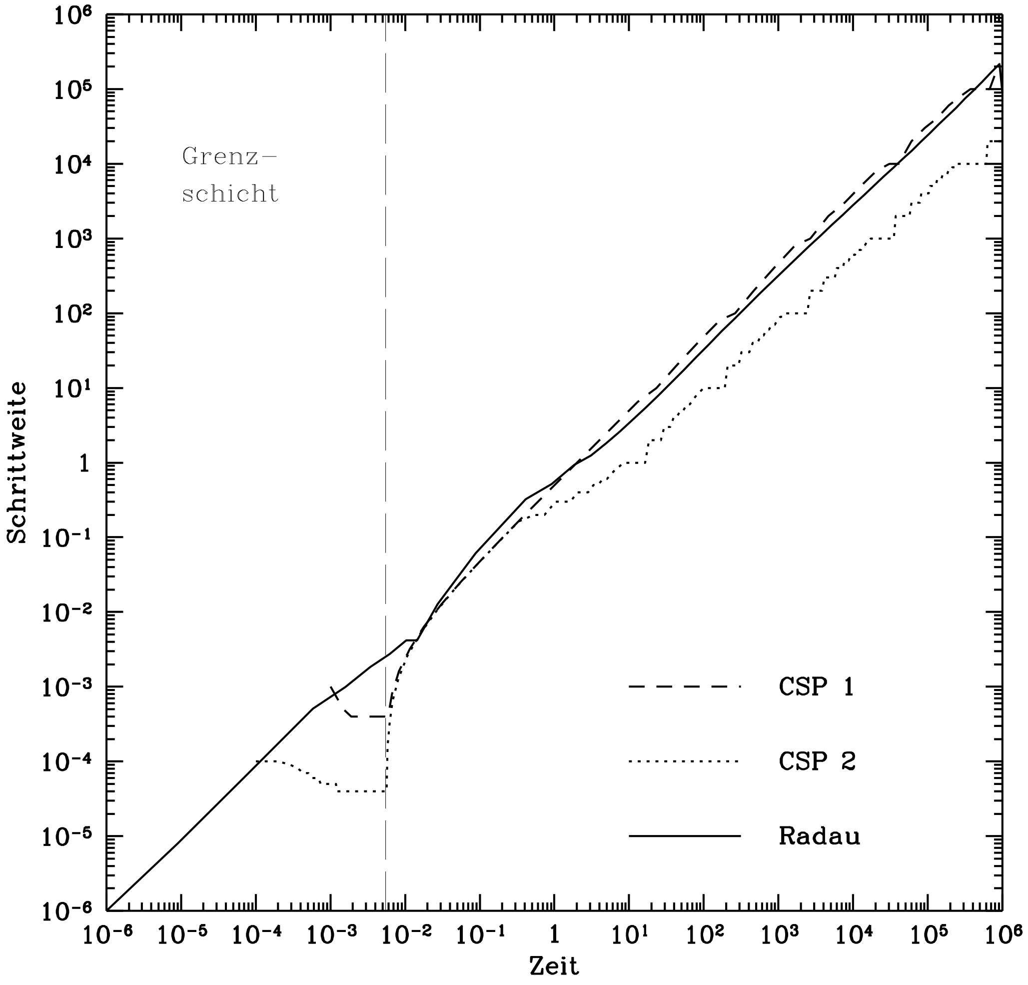


Abbildung 4.8: Schrittweiten der Integratoren

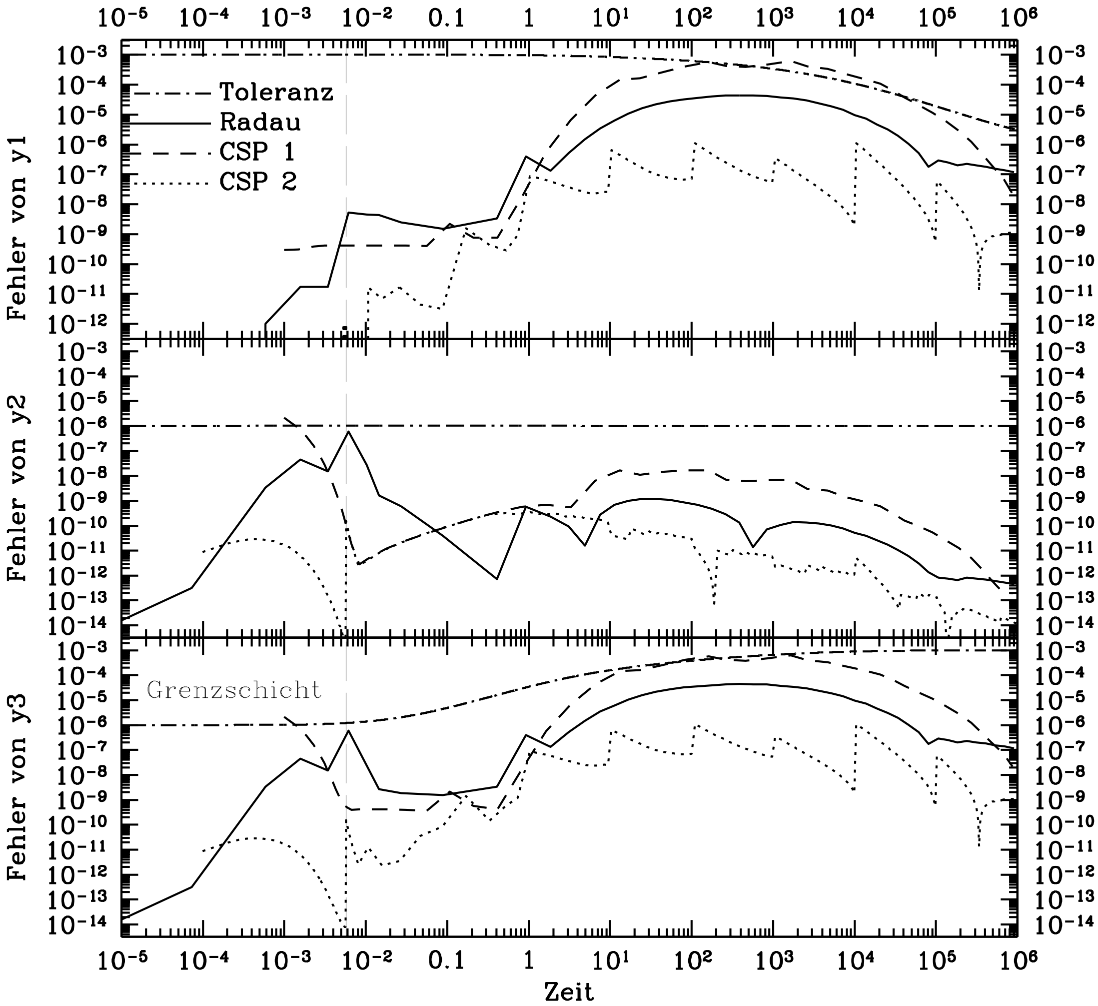


Abbildung 4.9: Fehler der verschiedenen Integratoren

4.3 Diskussion

Das Programm CSP91 kann, bei entsprechender Wahl des Parameters α, das Testproblem innerhalb der geforderten Toleranzen lösen. Da dies zwar mit etwas mehr Integrationsschritten als bei `Radau5`, aber mit einer geringeren, wenn auch größenordnungsmäßig vergleichbaren, Rechenzeit, geschieht, stellt das angewendete Näherungsverfahren in Kombination mit einem expliziten Löser tatsächlich eine Alternative zum Einsatz eines impliziten Lösers dar.

Dadurch, daß die Schrittweitensteuerung des Integrators mit Hilfe der aktuellen Zeitskala aus der Modenanalyse realisiert wird, sind diese beiden Programmteile derart eng miteinander verzahnt, daß diese Implementierung nicht als Modul in bestehende Programme zur Simulation von zirkumstellaren Hüllen importiert werden kann. Das ist jedoch das angestrebte Ziel dieser Arbeit. Da sich andererseits das Nährungsverfahren an sich als durchführbar herausgestellt hat, habe ich eine an die Forderung der Kompatibilität zu bereits bestehenden Programmen angepaßte Implementierung des Verfahrens erstellt. Diese wird im folgenden Kapitel vorgestellt.

Kapitel 5

Das Programm CSP++

5.1 Anforderungen

Die in der Einleitung angesprochenen Simulationsprogamme für ZSHn arbeiten mit Integrationsroutinen aus öffentlich zugänglichen Programmbibliotheken, für gewöhnliche DGLSe etwa `Dopri5`, [Hairer *et al.* 1987]. Diese Routinen rufen während ihrer Ausführung zwei lokal bereitgestellte Routinen auf:

`SOLOUT` wird nach jedem ausgeführten Integrationsschritt, hauptsächlich zur Ausgabe des berechneten Lösungspunktes, aufgerufen.

`RHS` wird zur Berechnung der jeweiligen rechten Seite $\mathbf{g}$ der DGL für jeden Integrationsschritt mehrmahls aufgerufen.

Die Routine `RHS` beinhaltet die gesamte physikalische Modellbildung, also in dieser Arbeit die chemischen Raten. Sie wurde für die Arbeit von [Beck *et al.* 1992] bereits implementiert.

Der grundsätzliche Aufbau beispielsweise eines auf `Dopri5` basierenden Programms ist in Abbildung 5.1 als Flußdiagramm dargestellt. Nach der Initialisierung wird die Routine `Dopri5` aufgerufen, die im Laufe der Integration mehrfach die Routinen `RHS` und `SOLOUT` aufruft. Nach Abschluß der Integration wird die Kontrolle zurück an das Hauptprogramm gegeben.

Ich werde einfachheitshalber den Namen `Dopri5` als Synonym für das gesamte auf dieser Routine basierende Programm benutzen.

Das neue Programm soll sich in diese Struktur einfügen lassen.

5.2 Gesamtstruktur der Implementierung

Da Kompatibiliät mit Programmpaketen, die die oben vorgestellte Struktur haben, gewährleistet werden soll, bieten sich für die Näherungen von CSP nur die beiden Routinen `RHS` und `SOLOUT` als Ansatzpunkt:

- Die zentrale Änderung gegenüber der bisherigen Implementierung ist die Substitution von `RHS` durch ein Routine `CSP_RHS`, die die bisherige Routine `RHS` zwar aufruft, aber dem Integrator nur eine Näherung $\mathbf{g}_{\mathrm{slow}}$, die Projektion in den langsamen Unterraum, des von `RHS` berechneten Vektors $\mathbf{g}$ zugänglich macht.

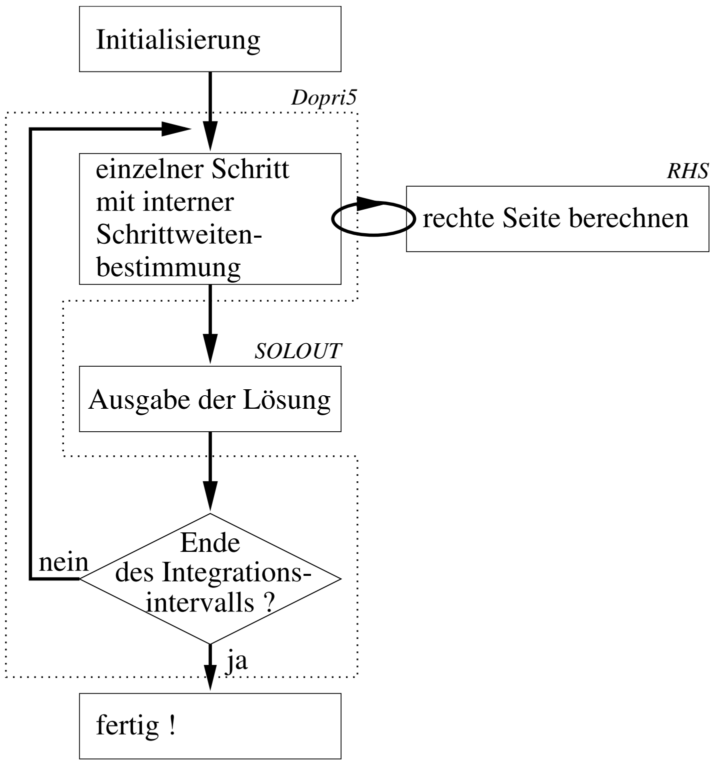


Abbildung 5.1: Flußdiagramm eines auf `Dopri5` basierenden Simulationsprogramms

- Zusätzlich können in `SOLOUT` weitere Aktionen unternommen werden. Insbesondere kann hier die Gültikeit der von CSP gemachten Näherung für den letzten Integrationsschritt überprüft werden. Die Aktionen werden in einer von `SOLOUT` aufzurufenden Routine `csp_go` durchgeführt.

Die Anpassung der Zerlegung in Moden an die sich ändernde Jakobimatrix **J**, die Bestimmungen der Anzahl der zu vernachlässigenden Moden und evtl. die Korrektur des Zustandes bezüglich der vernachlässigten Moden muß ebenfalls in einer oder beiden dieser Routinen geschehen. Die Implementierung soll hierbei alle Kombinationen ermöglichen.

Insgesamt ergibt sich die in Abbildung 5.2 dargestellte Struktur des Programms. Ich werde das gesamte Programm mit „CSP++“ bezeichnen. Durch Vergleich mit Abbildung 5.1 erkennt man, daß vier (als Routinen realisierte) Programmblöcke eingefügt wurden: `csp_ini` und `csp_end` werden zur Initialisierung und Speicherverwaltung vor und nach der Integration aufgerufen.

Während der Integration ruft `Dopri5` statt der ursprünglichen rechten Seite `RHS` die Routine `csp_rhs` auf. Diese kann je nach Konfiguration eine Basiszerlegung der Jakobimatrix durchführen, die Anzahl m der vernachlässigten Moden neu bestimmen und eine Korrektur des Zustandsvektors durchführen. Auf jeden Fall ruft sie `RHS` zur Berechnung von $\mathbf{g}$ auf und bestimmt daraus die Projektion $\mathbf{g}_{\mathrm{slow}}$ in den langsamen Unterraum. Dieser Vektor wird an `Dopri5` als rechte Seite des DGLSs zurückgegeben.

Nach jedem Integrationsschritt wird von `SOLOUT` die Routine `csp_go` aufgerufen. In dieser Routine kann wiederum je nach Konfiguration eine Basiszerlegung der Jakobimatrix durchgeführt, die Anzahl m der vernachlässigten Moden neu bestimmt und eine Korrektur

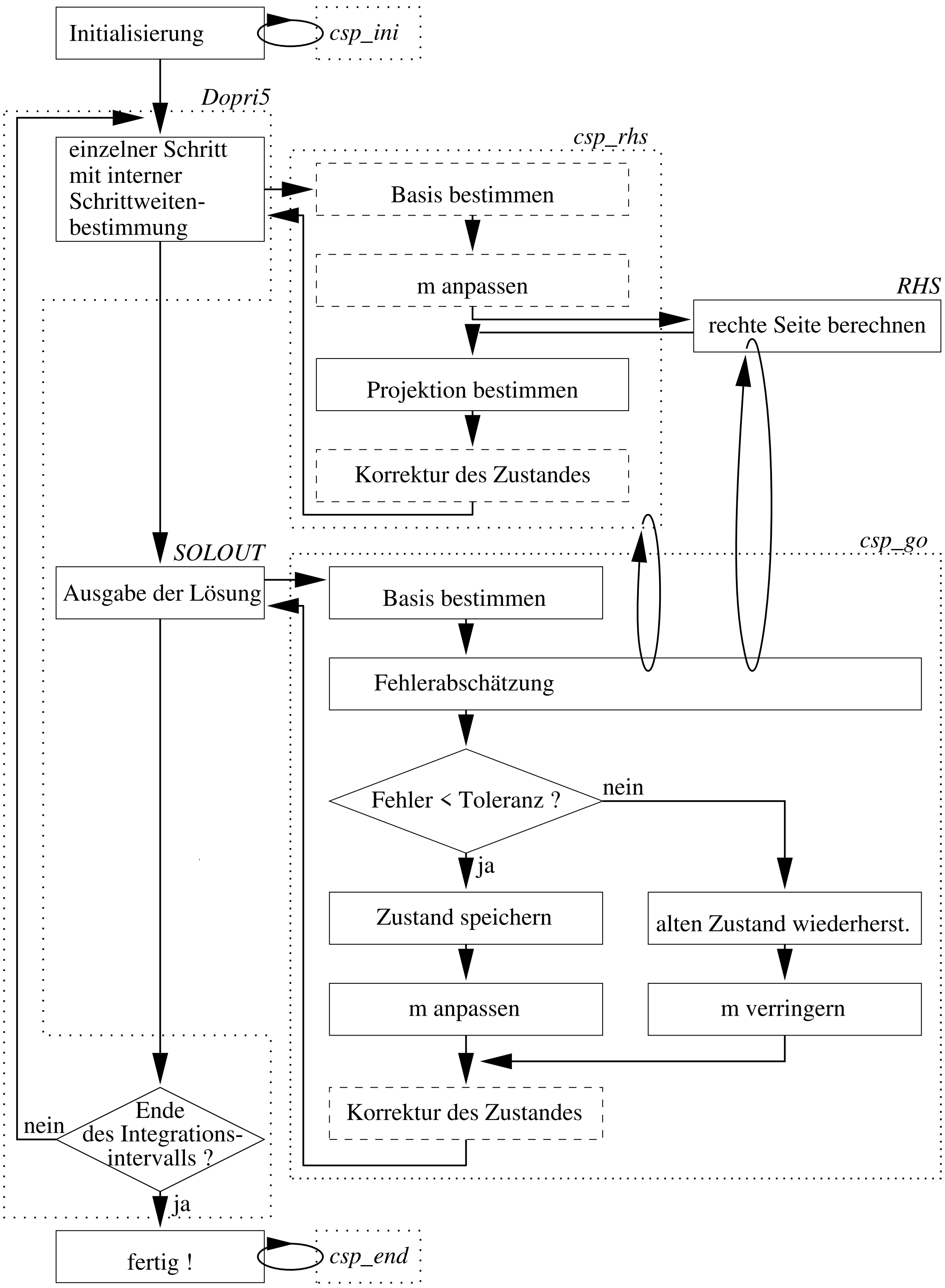


Abbildung 5.2: Flußdiagramm von CSP++. Die Anweisungen in gestrichelten Kästen sind optional. m bezeichnet die Anzahl der vernachlässigten Moden.

des Zustandsvektors durchgeführt werden. Da die Basis und m regelmäßig angepaßt werden müssen, kennzeichnet die Durchführung dieser Operationen in `csp_go` die absolute Minimalkonfiguration des Programms. Deshalb sind sie im Flußdiagramm nicht als optional gekennzeichnet.

Zusätzlich wird in `csp_go` der während des Integrationsschrittes durch die CSP-Näherung gemachte Fehler abgeschätzt. Dies kann durch Bildung der Differenz von $\mathbf{g}$ und $\mathbf{g}_{\text{slow}}$ geschehen, wozu die Routinen `RHS` und `csp_rhs` aufgerufen werden müssen: Die Multiplikation der Differenz mit der Schrittweite des letzten Integrationsschrittes ergibt eine Abschätzung für den gemachten Fehler. Wenn der Fehler größer als die vorgegebene Toleranz ist, wird durch kleiner Wahl von m die Näherung abgeschwächt und der Schritt rückgängig gemacht („der alte Zustand wiederhergestellt"), indem der Zustandsvektor und die Laufvariable mit den Werten des letzten korrekten Lösungspunktes überschrieben werden. Dazu müssen in dem Fall, daß der Integrationsschritt als gültig akzeptiert wird, diese Werte gespeichert werden. In dem letzten Fall kann auch die Stärke der Näherung, ausgedrückt durch m, an den neuen Zustand angepaßt werden.

Die Implementierung erfolgte in der Programmiersprache C++ ([Stroustrup 1991]), um das komplexe Problem möglichst übersichtlich programmieren zu können; insbesondere zur intuitiv verständlichen Darstellung der Vektor– und Matrixoperationen habe ich die Lineare Algebra–Bibliothek „jola" geschrieben und verwendet. Sie ist besonders zur Bearbeitung von Unterblöcken und einzelnen Zeilen und Spalten von Matrizen geeignet. Im Anhang A findet sich ein genauere Darstellung dieser Bibliothek.

Der (objektorientierte) C++–Code arbeitet problemlos mit bestehenden (prozeduralen) FORTRAN– und C-Programmbestandteilen zusammen. Für Aufrufe der CSP-Routinen aus C– und FORTRAN-Routinen wird im Abschnitt 5.3.5 ein Interface in der Syntax dieser Sprachen beschrieben. Interface besteht aus den oben erwähnten Routinen. Diese rufen dann die C++-Routinen auf.

5.3 Einzelne Bestandteile der Implementierung

Die gesamte Näherung wird durch ein Objekt der Klasse `Csp` repräsentiert, zu dessen Verständnis in den folgenden Abschnitten zunächst einige andere verwendete Klassen dargestellt werden. Die genauen Deklarationen der Klassen und Routinen können den im Anhang B abgedruckten Headerdateien entnommen werden.

5.3.1 Klasse `Problem`

Das numerische Anfangswertproblem (2.16) wird durch folgende Daten vollständig festgelegt:

- die Dimension n des DGLSs
- das DGLS, repräsentiert durch die Routine zur Berechnung ihrer rechten Seite $\mathbf{g}$, bzw. einen Pointer auf diese Routine
- das Integrationsintervall $[t_{\text{start}}, t_{end}]$
- die Anfangsbedingung $\mathbf{y}(t_{\text{start}})$

- die gewünschte Genauigkeit, üblicherweise gegeben durch die relative und die absolute Toleranz $rtol$ und $atol$: Komponentenweise soll der lokale Fehler der Lösung kleiner als $tol \equiv rtol \cdot y_i + atol$ sein. Allgemeiner können durch die Vektoren `rtol` und `atol` die Toleranzen für jede einzelne Komponente vorgegeben werden.

Diese Daten werden in einem Objekt der Klasse `Problem` zusammengefaßt, über das alle weiteren Klassen diese Daten lesen — aber nicht verändern — können. Bei der Initialisierung des Objektes müssen alle Daten angegeben werde, wobei die Toleranzen entweder als Skalare oder als Vektoren übergeben werden dürfen.

5.3.2 Klasse `State`

Die Lösung des Anfangswertproblems besteht in der eindeutigen Zuordnung eines Zustandsvektors $\mathbf{y}(t)$ zu jedem Zeitpunkt t, also in einer Auswahl von $(t, \mathbf{y})$–Paaren aus der Menge aller solcher Paare. Jedes Paar will ich in diesem Kapitel einen *Zustand* nennen. Es wird in dem Programm durch ein Objekt der Klasse `State` repräsentiert. Zu jedem Zustand lassen sich die rechte Seite der DGL, $\mathbf{g}$, der Toleranzvektor $\mathbf{tol}$, und die Jakobimatrix $\mathbf{J}$ eindeutig bestimmen. Auch diese Größen und eine Referenz auf das Problem, zu dem dieser Zustand gehört, sind in den Objekten der Klasse `State` enthalten. Durch die Bereitstellung dieser Daten in einem Objekt kann mehrmaliges Berechnen leicht vermieden werden.

Die Objekte der Klasse müssen für einen Zustand initialisiert werden, können später aber durch die Funktion `State.refresh(double t, Vector y)` mit einem anderen überschrieben werden. Dann wird der allozierte Speicherplatz mit den angebenen und neu berechneten Daten überschrieben. Zur Durchführung der Korrektur (Abschnitt 3.3) kann auch nur der Vektor $\mathbf{y}$ verändert werden, ohne daß alle anderen Daten in dem Objekt neu berechnet werden.

5.3.3 Klasse `Exhaustion`

Zu jedem Zustand kann eindeutig seine Zerlegung bestimmt werden, wenn man sich auf die Verwendung einer Zerlegungsroutine, z.B. Eigenvektorzerlegung, festgelegt hat. In der Klasse `Exhaustion` wird die Basis $\{\mathbf{b}_i\}$ und gegebenenfalls ihre Dualbasis $\{\mathbf{a}_i\}$ berechnet und gespeichert, ebenso die mit den Moden assoziierten Zeitskalen, Amplituden und geschätzten verbleibenden Einflüsse.

Außerdem wird in dieser Klasse die Anzahl der zu vernachlässigenden Moden und die damit bestimmte Projektionsmatrix in den langsamen Unterraum gespeichert.

Zerlegungsroutine

Zur Berechnung der Eigenvektorzerlegungen wurde die Routine `dgees` der LAPACK-Programmbibliothek ([Anderson *et al.* 1992]) verwendet. Sie berechnet die Zerlegung mit dem QR-Algorithmus ([Golub und vanLoan 1989]), ohne daß ihr eine Schätzung der Eigenvektoren als Startvektoren für die Iteration angegeben werden muß oder kann.

Die Verwendung einer Anfangsschätzung würde eine Modifikation der Routine und ihrer Unterroutinen erfordern, was zwar die Konvergenz erheblich beschleunigen, aber auch den Rahmen dieser Diplomarbeit sprengen würde.

5.3.4 Klasse `Csp`

Die Klasse `Csp` enthält zwei Objekte der Klasse `State`, eines für den aktuellen Zustand und eines als Sicherheitskopie des letzten, und ein Objekt der Klasse `Exhaustion`.

Die Memberfunktionen `Csp.rhs` und `Csp.work` sind die beiden in `RHS` und `SOLOUT` bzw. `csp_rhs` und `csp_go` aufzurufenden Routinen. In `Csp.rhs` wird eine aktuelle Näherung für die rechte Seite der DGL berechnet. In `Csp.work` wird geprüft, ob die bisherige Näherung, also die Projektionsmatrix, noch gültige Ergebnisse liefert. Wenn dies nicht der Fall ist, wird durch Überschreiben des aktuellen Zustandes mit der Sicherheitskopie der letzte Integrationsschritt rückgängig gemacht und die Näherung durch Verringern der Anzahl der vernachlässigten Moden genauer.

5.3.5 Interface zu C und FORTRAN

Vor Beginn der Integration muß je ein Objekt der Klassen `Problem` und `Csp` erzeugt und dabei initialisiert werden. Dies geschieht durch Aufruf der Funktion `csp_ini`. Während der Integration wird die rechte Seite des DGLSs durch die Funktion `csp_rhs` bereitgestellt. Diese Funktion ruft dazu lediglich `Csp.rhs` auf. In `SOLOUT` sollte die Funktion `csp_go`, die ein Interface zu `Csp.work` darstellt, zum Überprüfen der Gültigkeit der CSP-Näherung aufgerufen werden. Schließlich kann nach Abschluß der Integration durch Aufruf von `csp_end` in `csp_ini` allozierter Speicherplatz wieder freigegeben werden.

Kapitel 6

Anwendung von CSP++

6.1 *Robertson's example*

6.1.1 Konfiguration

Die Implementierung von CSP++ läßt offen, ob die Zerlegung der Jakobimatrix, die Anpassung der Anzahl m der vernachlässigten Moden und die Durchführung der Korrektur gemäß Abschnitt 3.3 nur in `csp_go`, nur in `csp_rhs` oder in beiden Routinen durchgeführt werden. Die verschiedenen Kombinationsmöglichkeiten habe ich an *Robertson's example* getestet.

Die Integration wurde mit den Toleranzen `rtol` $= 10^3$ und `atol` $= 10^6$ und verschiedenen der oben erwähnten Kombinationen durchgeführt. Die dabei aufgetretenen Fehler sind in den Abbildunge 6.1 und 6.2 dargestellt. Bei ihrer Berechnung wurde eine mit `Radau5` berechnete Lösung großer Genauigkeit (`rtol` $= 10^{12}$, `atol` $= 10^{15}$) als Referenz benutzt. Die ebenfalls dargestellte Toleranz wurde nach (3.14) aus der Referenzlösung bestimmt.

Die Ergebnisse aller Testläufe befinden sich in der Grenzschicht innerhalb der Toleranz. Da hier keine CSP-Näherung durchgeführt wird, beweist dies lediglich die korrekte Arbeitsweise der Integrationsroutine.

Da die Zerlegung in Moden und die Anpassung von m mindestens einmal pro Integrationsschritt gemacht werden müssen, habe ich sie bei allen Tests mindestens in `csp_go` durchgeführt. Findet keine weitere Aktion statt (w:ZA, r:–), so wird bei jedem Aufruf von `csp_rhs` durch die Integrationsroutine die Projektion von g in den beim letzten Aufruf von `csp_go` bzw. `SOLOUT` ermittelten langsam Unterraum bestimmt. In der Abbildung ist erkennbar, daß dabei nicht tolerierbare Fehler auftreten. Dies kann u.a. darauf zurückgeführt werden, daß sich aufgrund der Nichtlinerität des Systems die Basis und damit die Lage des langsamen Unterraums während eines Integrationsschrittes ändern.

Um die erkannte Fehlerquelle zu beseitigen, wurde zusätzlich bei jedem Aufruf von `csp_rhs` die Projektionsmatrix aus einer neu ermittelten Zerlegung bestimmt (w:ZA, r:Z). Damit lassen sich die Fehler zwar reduzieren — was die im letzten Absatz genannte Erklärung bestätigt — , aber für große Zeiten ($t > 10^4$) liegen die Fehler der Lösung über der geforderten Toleranz.

Durch zusätzliche Anpassung von m in `csp_rhs` (w:ZA, r:ZA) lassen sich die Fehler weiter reduzieren. Sie liegen aber auch bei dieser Integration am Ende des Integrationsintervalls, wenn das System auf seinen Endzustand relaxiert, nicht unterhalb der Toleranzschwelle.

Eine Korrektur nach Abschnitt 3.3 in `csp_go` — zunächst ohne weitere Aktion in

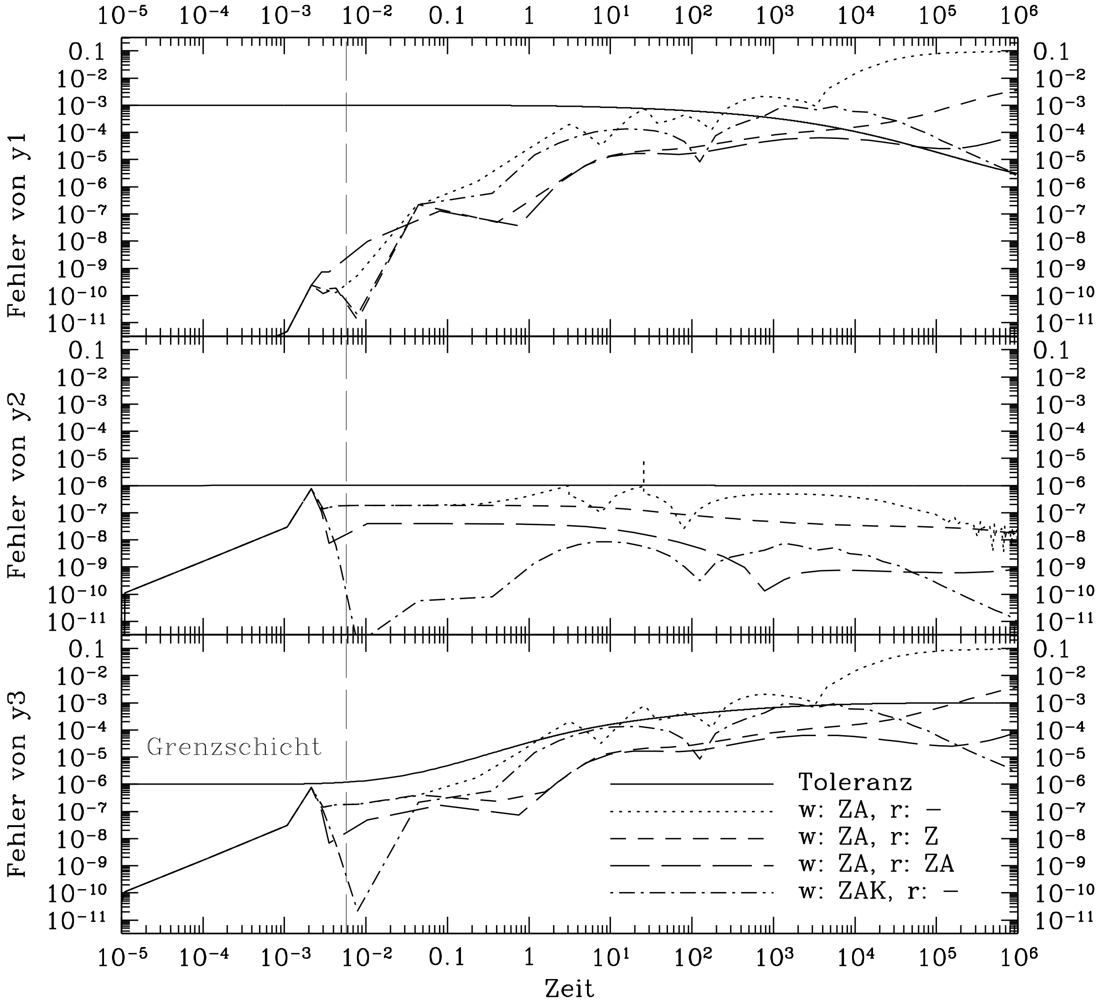


Abbildung 6.1: Fehler der mit CSP++ erstellten Lösungen von *Robertson's example*. Die in csp_go(w) und csp_rhs(r) verwendeten Näherungsmethoden sind: Zerlegung berechen, m Anpassen und Korrektur durchführen.

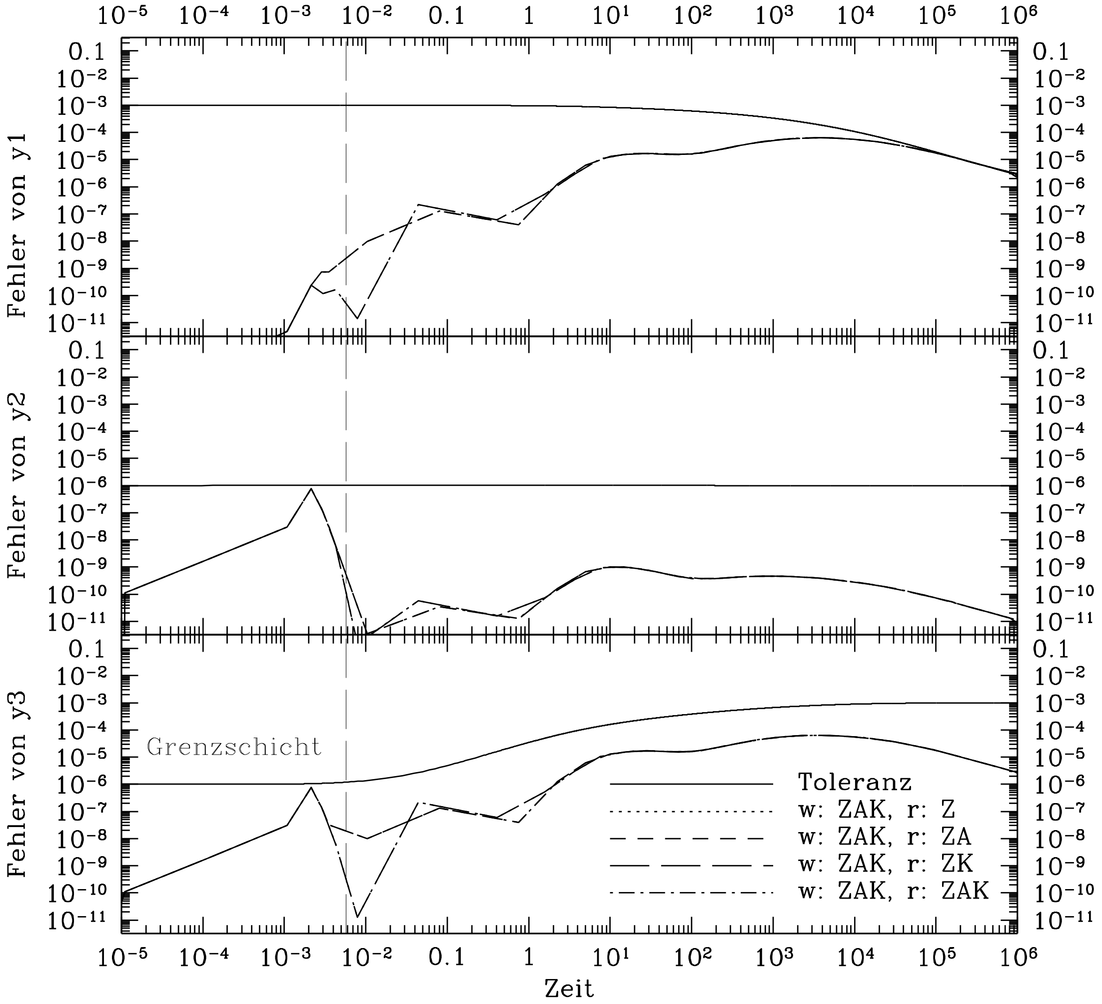


Abbildung 6.2: Fehler der mit CSP++ erstellten Lösungen von *Robertson's example*. Die in `csp_go`(w) und `csp_rhs`(r) verwendeten Näherungsmethoden sind: Zerlegung berechen, m Anpassen und Korrektur durchführen.

csp_rhs— (w:ZAK, r:–) verkleinert gerade in der letzten Relaxationsphase den Fehler auf eine tolerierbare Größe. Dies ist um so bemerkenswerter, als der Fehler zwischenzeitlich oberhalb der Toleranz liegt und danach wieder verringert wird: Die Korrektur scheint ein wichtiges Instrument bei der Annäherung an die Gleichgewichtslage zu sein.

Durch Kombination einer Korrektur in csp_go und einer Zerlegung in csp_rhs (w:ZAK, r:Z) ergibt sich im gesamten Integrationsbereich eine Lösung, die die geforderten Toleranzen einhält: Die Zerlegung in csp_rhs berechnet jeweils den aktuellsten Tangentialraum an den langsamen Unterraum und die Korrektur in csp_go gewährleistet eine weitere Relaxation auch der vernachlässigten Moden.

Zusätzliche Aktionen in csp_rhs (w:ZAK, r:Z(A)(K)) haben keinen nenneswerten Einfluß auf die Lösung. Dies bestätigt, daß das Verfahren in der Konfiguration w:ZAK, r:Z optimal arbeitet. Deshalb wird diese Konfiguration in allen weiteren Läufen von CSP++ verwendet.

6.1.2 Analyse der Moden

Die Inspektion der von CSP++ errechneten und benutzten Moden erlaubt einen Einblick in die Arbeitsweise der Näherungsmethode, und durch den Vergleich mit den Eigenschaften der von der ursprünglichen Implementierung CSP91 berechneten Moden kann getestet werden, ob die beiden Implementierungen die Idee des Verfahrens in gleicher Weise umsetzen.

Bei einer Integration von *Robertson's example* mit der durch rtol $= 10^6$ und atol $= 10^3$ gegebenen Toleranz wurden die Zeitskalen τ_i, die Amplituden f_i, die Norm $\|\mathbf{y}_{\mathrm{ge},i}\|$ der abgeschätzten verbleibenden Gesamteinflüsse und die Komponenten der Basisvektoren $\mathbf{a}_i$ und ihrer dualen Vektoren $\mathbf{b}_i$ für alle drei Moden ausgegeben.

In Abbildung 6.3 sind die Zeitskalen, Einflüsse und Amplituden dargestellt. Zusätzlich wurden die Schrittweiten des Intgrators aus dem Abstand der einzelnen Lösungspunkte bestimmt und in der Mitte des jeweiligen Schrittes als Dreieck eingezeichnet. Die gepunktete Linie zeigt die Ausdehnung der Grenzschicht an: links von dieser Linie ($t < 3 \cdot 10^{-3}$) wird keine Mode vernachlässigt ($m = 0$), bei $t = 3 \cdot 10^{-3}$ wird zum ersten Mal eine Mode als relaxiert eingestuft, und für größere Zeiten wird die Integration unter Vernachlässigung einer Mode fortgeführt ($m = 1$).

Die ersten zwei der drei Grafiken entsprechen der Darstellung in Abbildung 4.5 für CSP91. Es ist jedoch darauf zu achten, daß die Numerierung der Moden in entgegengesetzter Reihenfolge geschieht. Dies ist in der umgekehrten Sortierung der Moden in den zwei Implementierungen begründet. Die in CSP++ verwendete Sortierung folgt der Darstellung in Kapitel 3. Ihr Vorteil liegt darin, daß die üblichen Schurvektorzerlegungsalgorithmen (z.B. dgees aus der LAPACK-Bibliothek) zu einem späteren Zeitpunkt zur Bestimmung der Zerlegung eingesetzt werden können.

Die mit den Moden assoziierten Zeitskalen zeigen eine vollkommen analoge Entwicklung zu den von CSP91 berechneten. Bei den gemachten Zeitschritten zeigt sich die Unabhängigkeit der Integrationsroutine Dopri5 von der Interpretation des Problems als bestehend aus Moden mit Zeitskalen: Zunächst werden die Zeitschritte kleiner als nötig gewählt, weil der Integrator in Ermangelung weiterer Informationen mit der Standardschrittweite 10^{-6} beginnt. Die Schrittweite wird dann gesteigert und es zeigt sich, daß sie sogar innerhalb der Grenzschicht schon etwas über die aktive Zeitskala, die Zeitskala der dritten Mode, gehoben werden kann. Außerhalb der Grenzschicht erreicht die Schrittweite schnell die Größe der dann aktiven zweiten Mode und orientiert sich im weiteren Verlauf der Integration sehr

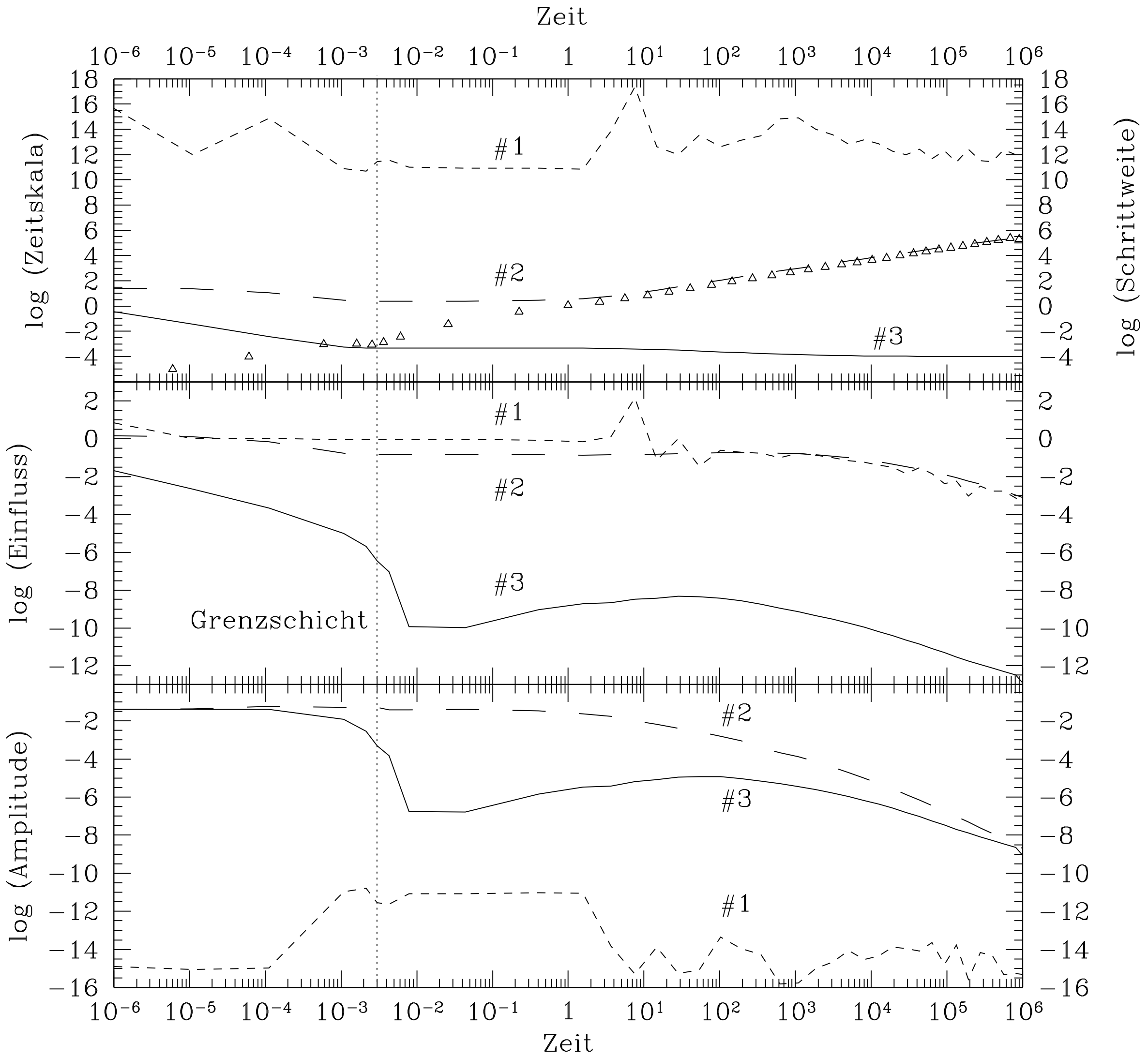


Abbildung 6.3: Mit CSP++ berechnete Moden von *Robertson's example*. Die Dreiecke stellen jeweils die Schrittweite des Integrators `Dopri5` dar.

genau an dieser. Da die Schrittweitensteuerung des Integrators keine direkten Informationen über die von CSP++ ermittelte aktive Zeitskala erhält, beschreibt die Projektion $\mathbf{g}_{\mathrm{slow}}$ tatsächlich nur noch die in dem Problem enthaltenen Prozesse, deren Zeitskalen gleich oder größer der aktiven Zeitskala sind.

Die Gesamtzahl der Integrationsschritte beträgt nun 37, während CSP91 mehr als 53 Schritte benötigte (siehe Tabelle 4.1), um das Problem innerhalb derselben Toleranzen zu lösen. Diese und die damit einhergehende Verminderung der Anzahl der Auswertungen der rechten Seite g zur Beschreibung der chemischen Raten um über 25% ist auf die an $\mathbf{g}_{\mathrm{slow}}$ statt an der aktiven Zeitskala orientierten Schrittweitensteuerung zurückzuführen.

Die verbleibenden Gesamteinflüsse $\mathbf{y}_{\mathrm{ge},i}$ der Moden entwickeln sich ebenfalls (nahezu) analog zu den von CSP91 berechneten. Obwohl die zweite Mode außerhalb der Grenzschicht nicht mehr integriert wird, fällt ihr Gesamteinfluß zunächst weiter ab. Der Abfall ist auf die angebrachte Korrektur zurückzuführen. Da auch CSP91 diese Korrektur durchführt, ist es jedoch auffällig, daß der verbleibende Gesamteinfluß der vernachlässigten Mode drei Größenordnungen unter dem Wert der Näherung durch CSP91 bleibt. Dies deutet darauf hin, daß die von CSP++ gemachte Näherung besser ist, denn der verbleibende Gesamteinfluß ist ja gerade das Maß für den durch die Vernachlässigung gemachten Fehler. Der gravierendste Unterschied zwischen den beiden Programme liegt in der Art der Zerlegung: Die von CSP++ benutzte Eigenvektorzerlegung bietet durch die vollständige Entkopplung aller Moden eine detailliertere Beschreibung des dynamischen Verhaltens des Systems als die in CSP91 verwendet Zerlegung nach [Lam und Goussis 1991] (siehe Abschnitt 3.4.3). Dadurch scheint sich eine bessere Projektion in den langsamen Unterraum bestimmen zu lassen.

Die Amplituden der Moden geben ihren momentanen Einfluß, d.h. Anteil an g, an. Hier zeigt sich deutlich, daß die erste Mode praktisch über das gesamte Integrationsintervall vernachlässigbar ist: Die von dieser Mode repräsentierte Größe bleibt erhalten. Welche Größe repräsentiert wird, soll im Folgenden anhand der Abbildung 6.5 diskutiert werden. Die dritte Mode hat innerhalb der Grenzschicht einen großen momentanen Einfluß. Er sinkt dann ab, bis die Mode am Ende der Grenzschicht vernachlässigt wird. Außerhalb der Grenzschicht fällt die Amplitude — wie der verbleibende Gesamteinfluß — zunächst durch Anbringen der Korrektur noch weiter. Sie bleibt dann im Verhältnis zur Amplitude der zweiten, aktiven Mode klein. Deren Amplitude dominiert den Bereich zwischen Grenzschicht und letztendlicher Relaxation des gesamten Systems. Im quasi relaxierten Zustand fallen schließlich alle Amplituden stark ab.

In Abbildung 6.4 sind die einzelnen Komponenten der rechten Eigenvektoren der Jakobimatrix dargestellt. Sie werden als Basisvektoren $\mathbf{a}_i$ für die Moden benutzt. Die Ausdehnung der Grenzschicht ist, wie in Abbildung 6.3, durch eine gepunktete Linie markiert.

Der Vektor g der Konzentrationsänderungen setzt sich gemäß (3.2) zu jedem Zeitpunkt aus den drei Basisvektoren gewichtet mit ihren Amplituden f_i zusammen. Wenn die Amplitude einer Mode vernachlässigbar ist, verschwindet auch ihr Anteil an g. Die Mode hat dann weder einen Einfluß auf die Änderung des Systems noch eine direkt physikalisch interpretierbare Bedeutung. Dies ist bei der ersten Mode über das gesamte Integrationsintervall hinweg der Fall.

Wenn die Amplitude einer Mode nicht vernachlässigbar klein ist, dann ist der Anteil dieser Mode an g auch nicht vernachlässigbar und die Komponenten dieser Mode lassen sich als stöchiometrische Koeffizienten einer abstrakten Reaktion deuten: die zweite Mode, die überall die größte Amplitude besitzt, repräsentiert die Nettoreaktion von einer Spezies A zu einer Spezies C: Ihre erste Komponente ist negativ, die dritte ist positiv, und die beiden

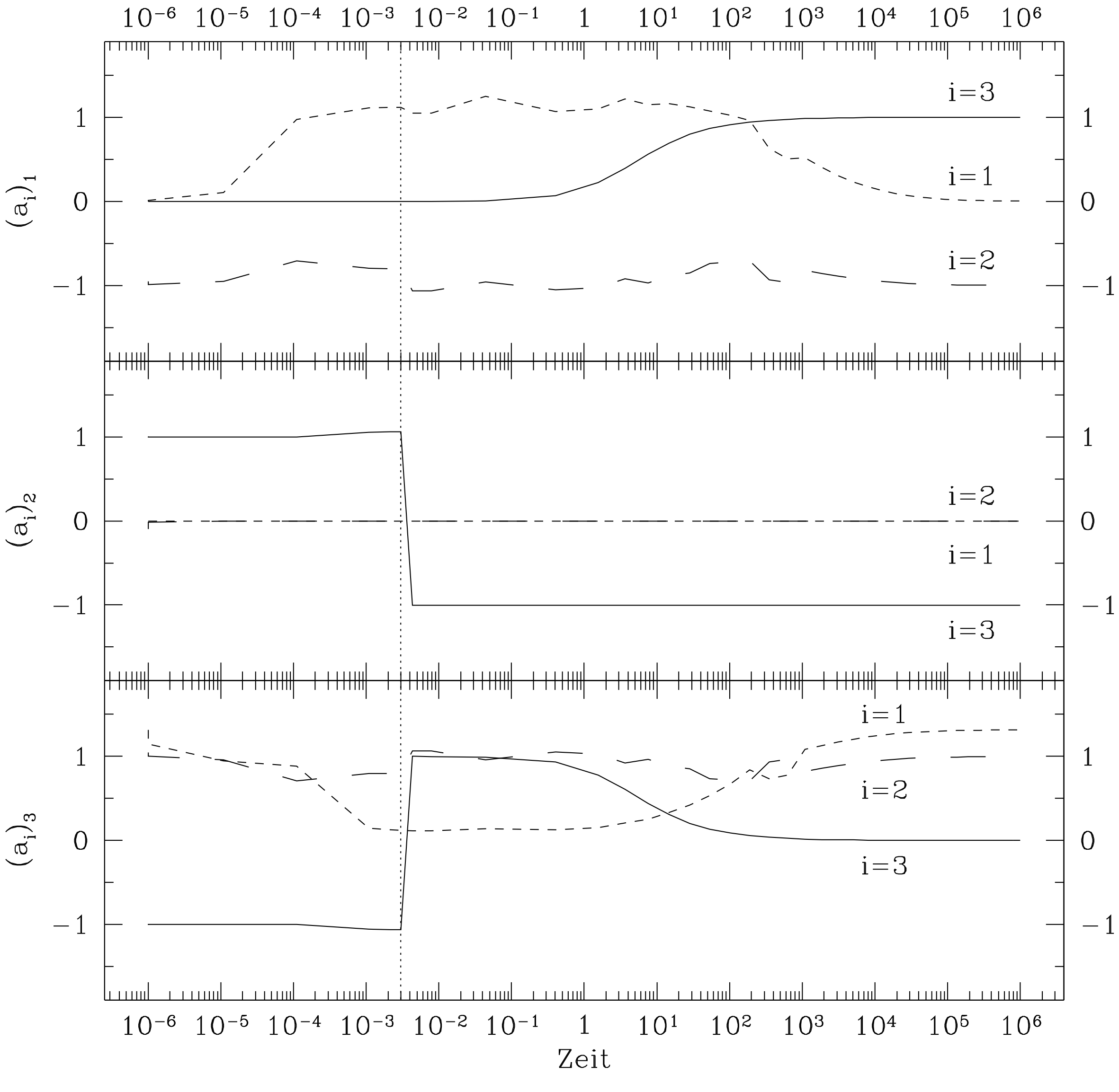


Abbildung 6.4: Komponenten der von CSP++ berechnete rechten Eigenvektoren $\mathbf{a}_i$ von *Robertson's example*. Die gepunktete Linie markiert den Rand der Grenzschicht.

Komponenten haben den gleichen Betrag. Da die zweite Komponente verschwindet, ist die Spezies B nicht an der Reaktion beteiligt. Die dritte Mode ist nur innerhalb der Grenzschicht mitentscheidend; sie repräsentiert dort eine Reaktion von der Spezies C zur Spezies B. Diese abstrakte Reaktion findet sich nicht in den physikalischen wieder[1]. Gleichwohl sind die Konzentrationen aller drei Spezies durch die abstrakte Reaktionen der zweiten und dritten Mode miteinander verknüpft. Außerhalb der Grenzschicht ist eine direkte Interpretation von $\mathbf{a}_3$ nicht sinnvoll. Aus dem auf die Vernachlässigung folgenden Vorzeichenwechsel ist jedoch darauf zu schließen, daß die erste Korrektur etwas zu stark war.

In Abbildung 6.5 sind die Komponenten der linken Eigenvektoren $\mathbf{b}_i$ in Richtung der Konzentrationsänderungen g_i der drei Spezies dargestellt. Die Lage Grenzschicht ist wiederum durch eine gepunktete Linie gekennzeichnet.

Die Amplituden der Moden berechnen sich nach (3.1) aus diesen $\mathbf{b}_i$:

$$f_i = \sum_{j=1}^{n} (\mathbf{b}_i)_j g_j \ . \tag{6.1}$$

Da g_j die Änderungen der j-ten Spezies ist, lassen sich diese Gleichungen als n Entwicklungsgleichungen deuten:

$$\frac{\mathrm{d}}{\mathrm{d}t} \sum_{j=1}^{n} (\mathbf{b}_i)_j \, y_j = f_i \ , \quad i = 1 \ldots n \ . \tag{6.2}$$

Bei verschwindendem Quellterm f_i bedeutet dies, daß die Größe $\sum_{j=1}^{n} (\mathbf{b}_i)_j y_j$ erhalten bleibt.

Die Amplitude der ersten Mode ist verschwindend gering. Da ihre Komponenten stets gleich groß sind, bleibt die Größe $\sum_{j=1}^{n} y_j$ über das gesamte Integrationsintervall erhalten. Dies entspricht dem Gesetz der Teilchenerhaltung.

Bei chemischen Systemen kann (6.2) für Moden mit Eigenwerten und Amplituden gleich Null (im Rahmen der Maschinengenauigkeit) als Erhaltungssatz jeweils eines Elementes gedeutet werden. Der Quotient $(\mathbf{b}_i)_j / (\mathbf{b}_i)_k$ gibt dann die in der j-ten Spezies enthaltene Anzahl von diesen Atomen relativ zur in der k-ten Spezies enthaltenen Anzahl an. Im Falle von *Robertson's example* enthalten alle drei Spezies dieselbe Anzahl des Elements.

Die zweite Mode hat nur eine Komponente in Richtung der Spezies B. Da außerhalb der Grenzschicht die Amplitude dieser Mode verschwindet, erhält man für $t > 3 \cdot 10^3$ aus (6.2) einen (näherungsweisen) Erhaltungssatz für diese Spezies:

$$g_2 \approx 0 \ . \tag{6.3}$$

Da dies äquivalent zu der Quasistationaritätsnäherung (2.24) ist, erweist sich B damit als ein reaktionskinetisches Radikal des Systems.

Offensichtlich wäre eine exakte Gleichsetzung von g_2 und null nicht richtig, denn die Konzentration von B ist nicht tatsächlich konstant sondern nur klein im Vergleich zu den Konzentrationsänderungen von A und C.

Insofern bleibt festzuhalten, daß sich physikalische, exakte Erhaltungssätze immer in Moden mit verschwindenden Eigenwerten und Amplituden manifestieren, während relaxierte Moden zu **lokal geltenden Näherungen** führen.

---

[1] In *Robertson's example* werden keine Rückreaktionen berücksichtigt

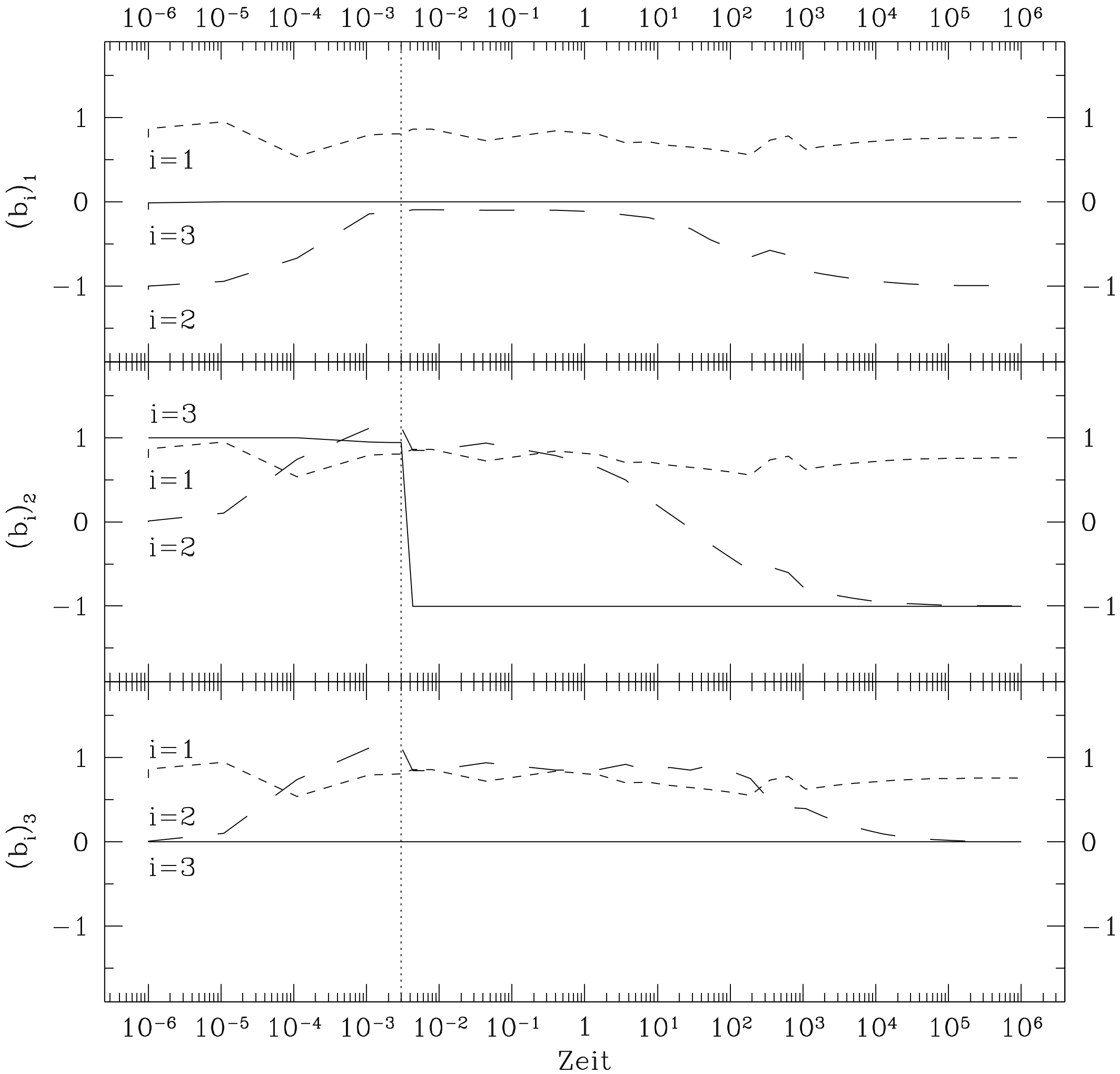


Abbildung 6.5: Komponenten der von CSP++ berechnete linken Eigenvektoren $\mathbf{b}_i$ von *Robertson's example*. Die gepunktete Linie markiert den Rand der Grenzschicht.

6.1.3 Benötigte Rechenzeit

Um die von CSP++ benötigte Rechenzeit einzuschätzen, wurde deren Abhängigkeit von der Dimension des zu lösenden Problems mit derjenigen von `Radau5` verglichen.

Als Testproblem diente eine Modifikation von *Robertson's example*, dessen Dimension sich in Dreierschritten wählen läßt. Ein n–dimensionales Problem (mit $n = 3L$) besteht dann aus L Problemen des Typs von *Robertson's example*. Die einzelnen 3–dimensionalen Probleme unterscheiden sich jedoch in den Ratenkoeffizienten:

$$\frac{\mathrm{d}}{\mathrm{d}t}\, y_{1+3l} = -\frac{0.04}{10(3l+1)}\, y_{2+3l} + \frac{10^4}{100(3l+1)}\, y_{2+3l}\, y_{3+3l} \tag{6.4}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\, y_{2+3l} = +\frac{0.04}{10(3l+1)}\, y_{1+3l} - \frac{10^4}{100(3l+1)}\, y_{2+3l}\, y_{3+3l} - \frac{3\cdot 10^7}{3l+1}\, y_{2+l}^2 \tag{6.5}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\, y_{3+3l} = +\frac{3\cdot 10^7}{3l+1}\, y_{2+3l}^2\,. \tag{6.6}$$

Dabei läuft der Index l von 0 bis $L-1$. Die DGLen der ersten drei Komponenten des Problems sind genau das ursprüngliche *Robertson's example*. Dadurch, daß der kleinste Ratenkoeffizient, 0.04, sich mit zunehmendem l stärker verkleinert als der größte, $3\cdot 10^7$, werden die 3–dimensionalen Teilprobleme mit zunehmendem l steifer. Deswegen wird das Gesamtproblem mit zunehmender Dimension steifer und deshalb schwerer numerisch zu lösen.

Als Integrationsintervall wurde, wie im letzten Abschnitt, $[0, 10^6]$ gewählt. Die Toleranz ist durch `rtol` $= 10^3$ und `atol` $= 10^6$ bestimmt. Ihre Einhaltung wurde an einzelnen Komponenten exemplarisch verifiziert.

Die Zeitmessung erfolgte mit dem UNIX-Befehl `timex`. Er mißt u.a. die vom Benutzer für die Ausführung eines Befehls verbrauchte Rechenzeit (`usr time`).

Die von CSP++ mit der Integrationsroutine `Dopri5` für die Lösung von Problemen mit Dimensionen zwischen 3 und 99 benötigte Rechenzeit wird in Abbildung 6.6 der von `Radau5` benötigten gegenübergestellt.

Man erkennt, daß `Radau5` um mindestens einen Faktor zehn schneller als CSP++ ist. Außerdem ist der Rechenzeitanstieg von CSP++ mit zunehmender Dimension offensichtlich schneller als linear. Genauere Informationen über diesen Anstieg kann man einer doppelt logarithmischen Auftragung derselben Daten entnehmen (siehe Abbildung 6.7). Zur Orientierung sind in der Abbildung drei gestrichelte Linien eingezeichnet, deren Steigungen einem linearen, einem quadratischen und einem kubischen Zusammenhang entsprechen.

Sowohl bei `Radau5` als auch bei CSP++ verläuft der Anstieg zunächst fast linear und wird dann steiler. Die Steigung des Graphen von `Radau5` hat sich am Ende des betrachteten Intervalls nahezu auf einen Wert erhöht, der einem quadratischen Anstieg entspricht, während die Rechenzeit von CSP++ oberhalb einer Dimension von ca. 30 schneller als quadratisch, aber langsamer als kubisch ansteigt.

Der bei niedriger Dimension relativ langsame Anstieg der Rechenzeiten ist darauf zurückzuführen, daß derjenige Rechenschritt, dessen Ausführungszeit in der höchsten Potenz von der Dimension abhängig ist, noch nicht die gesamte benötigte Zeit dominiert.

Durch den schnelleren Anstieg der Rechenzeit von CSP++ vergrößert sich der Faktor, um den CSP++ langsamer als `Radau5` ist, von 11 bei $n = 3$ auf 23 bei $n = 99$.

Bei dem direkten Vergleich der benötigten Zeiten von CSP++ und `Radau5` muß beachtet werden, daß es sich bei letzterem um eine bereits optimierte Implementierung handelt,

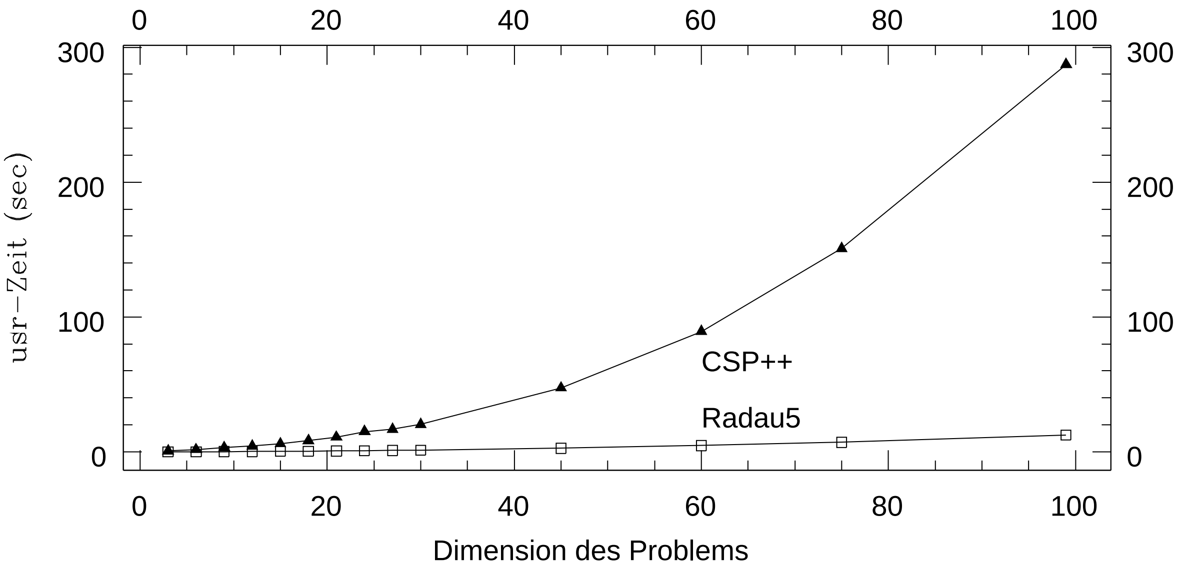


Abbildung 6.6: Von CSP++ und `Radau5` benötigte Rechenzeiten in Abhängigkeit der Dimension des Problems

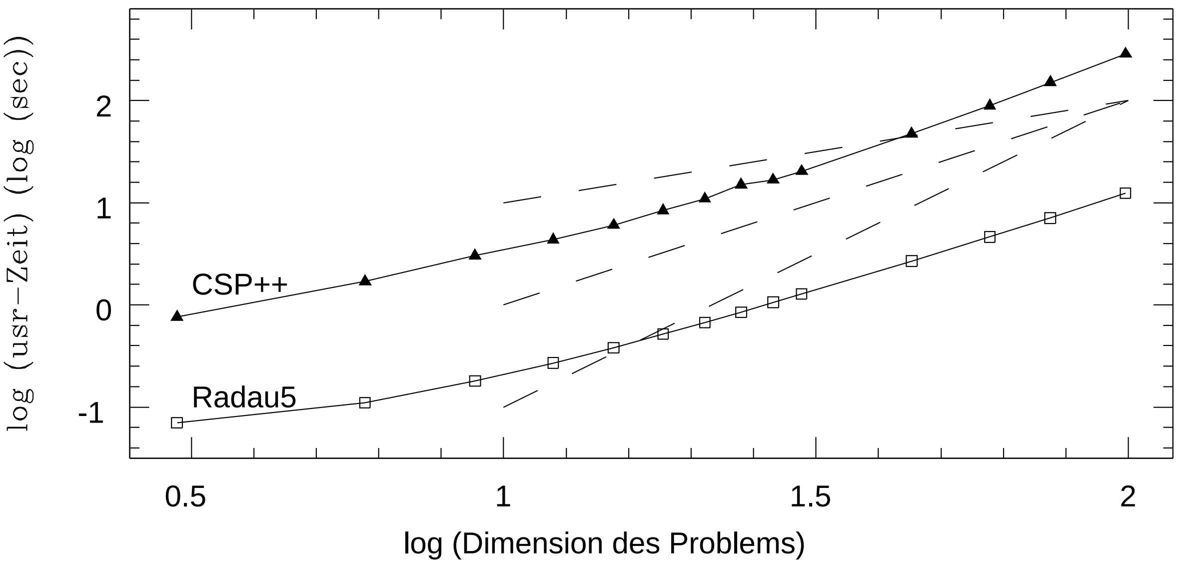


Abbildung 6.7: Von CSP++ und `Radau5` benötigte Rechenzeiten in Abhängigkeit der Dimension des Problems in doppelt logarithmischer Auftragung. Die drei gestrichelten Linien geben die Steigung bei linearem, quadratischem und kubischem Zusammenhang an.

während CSP++ noch viele Möglichkeiten zur Optimierung bietet, die im Rahmen dieser Diplomarbeit nicht ausgeschöpft werden konnten. Beispielsweise sei in diesem Zusammenhang daran erinnert, daß die Eigenvektorzerlegung (noch) keinen Gebrauch von der vorhandenen guten Schätzung der Eigenvektoren macht, siehe Abschnitt 3.5.2.

Selbst wenn eine Optimierung von CSP++ dieses Verfahren bei den betrachteten Dimensionen auf eine mit der von `Radau5` vergleichbaren Geschwindigkeit bringen kann, wird trotzdem aufgrund der intrinsisch kubischen Abhängigkeit von CSP++ von n, `Radau5` bei Problemen hinreichend großer Dimension immer schneller sein.

6.2 Sauerstoffreiche Sternhülle

Als Beispiel für chemische Ratennetzwerke von M-Sternhüllen habe ich das von [Beck *et al.* 1992] untersuchte System gewählt. Es beschreibt die Hülle eines M-Sterns mit dem Sternradius $R_* = 7 \cdot 10^{13}$cm, der Massenverlustrate $\dot{M} = 2 \cdot 10^{-6} \mathrm{M}_\odot/\mathrm{yr}$ und der Effektivtemperatur $T_* = 3000$K.

Für die Hüllenstruktur wird ein radialer Ausfluß mit konstanter Geschwindigkeit $v = 10$ km/s angenommen. Die Geschwindigkeit ist der einzige Unterschied zu dem Hüllenmodell von [Beck *et al.* 1992], in dem mit $v = 1$ km/s gearbeitet wird. Durch diese Geschwindigkeit wird ein linearer Zusammenhang zwischen der Zeit t und dem Radius R, bei dem sich ein Teilchen, daß sich beim Start der Integration ($t = 0$) bei $R = 2R_*$ befindet, hergestellt. Die Anzahldichte n des Gases bei einem Radius R kann nun berechnet werden:

$$n(R) = \frac{\dot{M}}{4\pi R^2 v \mu m_{\mathrm{H}}} \,. \tag{6.7}$$

Dabei bezeichnet m_{H} die Protonenmasse. Das mittlere Atomgewicht μ wird mit 1.4 angenommen.

Für die Temperaturstruktur der Hülle wird wie in [Beck *et al.* 1992] angenommen:

$$T(R) = \begin{cases} T_0 & (R < 5R_*) \\ T_0 \frac{5R_*}{R} & (5R_* < R < 15R_*) \\ \frac{1}{3} T_0 \sqrt{\frac{15R_*}{R}} & (R > 15R_*) \end{cases} \tag{6.8}$$

Dabei wird die Anfangstemperatur $T(2R_*)$ auf $T_0 = 1100$ K gesetzt. Die sich aus diesen Formel ergebende Hüllenstruktur ist in Abbildung 6.8 dargestellt.

Aufgrund der im letzten Abschnitt dargestellten geringen Rechengeschwindigkeit von CSP++ wurde das chemische Ratennetzwerk auf die von [Beck *et al.* 1992] identifizierte Sauerstoff-Wasserstoff-Gruppe reduziert. Sie beeinhaltet die in Tabelle 6.1 aufgeführten sieben Spezies. Dort sind auch ihre Anfangskonzentrationen bei $t = 0$ angegeben.

Es wurden die Raten der folgenden von [Beck *et al.* 1992] als effizient charakterisierten Reaktionen berücksichtigt:

$$H + OH \rightleftharpoons O + H_2 \tag{6.9}$$

$$O + OH \rightleftharpoons O_2 + H \tag{6.10}$$

$$H + H^- \rightleftharpoons H_2 + e^- \tag{6.11}$$

$$H_2 + OH \rightleftharpoons H + H_2O \tag{6.12}$$

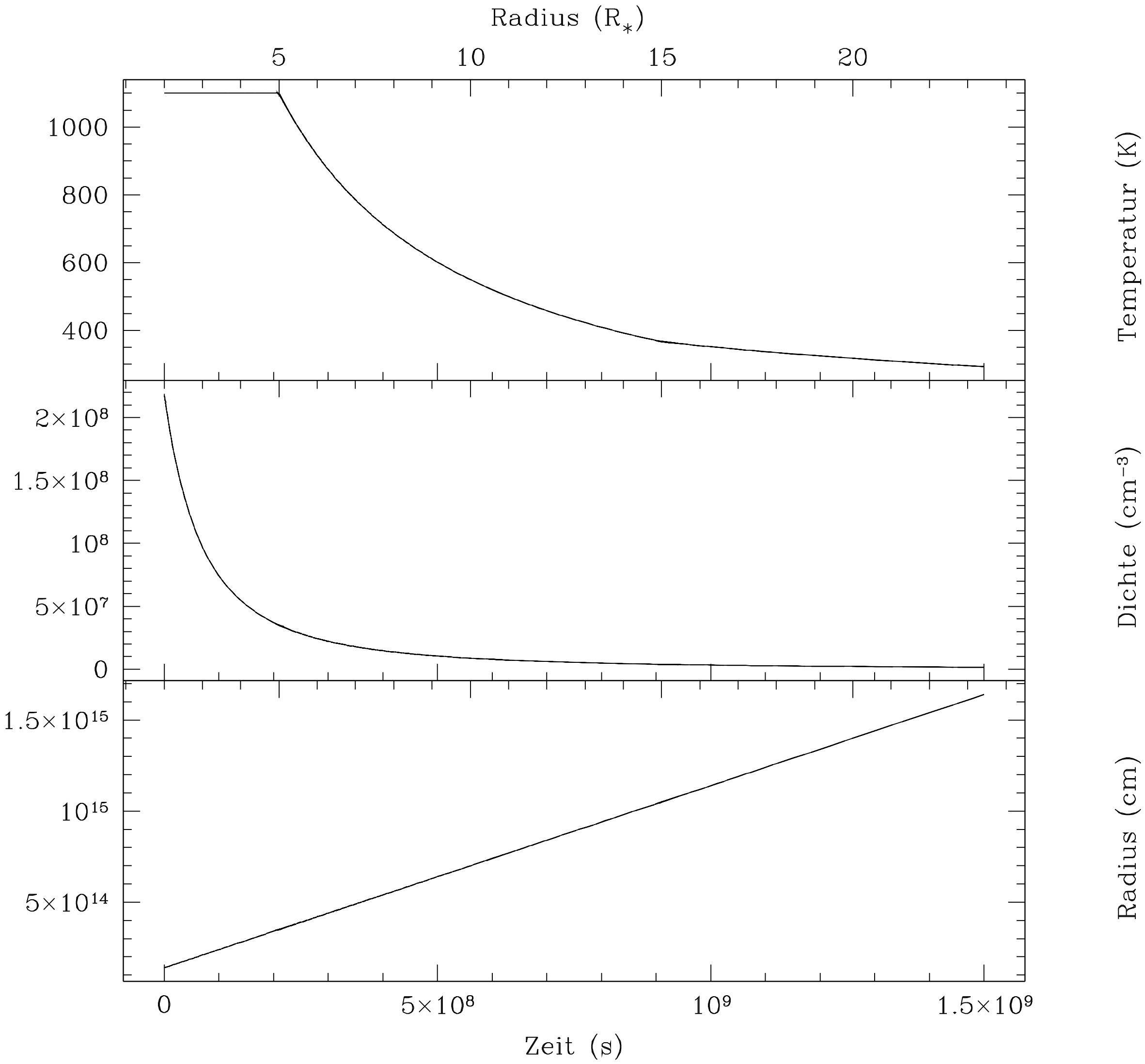


Abbildung 6.8: Temperatur- und Dichtestruktur der ZSH

chemisches Symbol	Anfangshäufigkeit
H	1
H_2	$1.3 \cdot 10^{-4}$
O	$5 \cdot 10^{-4}$
OH	$2 \cdot 10^{-7}$
O_2	$1.3 \cdot 10^{-8}$
H_2O	$2 \cdot 10^{-9}$
H^-	$2 \cdot 10^{-14}$

Tabelle 6.1: Spezies der OH-Gruppe und ihre Anfangshäufigkeiten (relativ zu Wasserstoff)

$$H + e^- \rightleftharpoons H^- + h\nu \quad (6.13)$$
$$O + H \rightleftharpoons OH + h\nu \quad (6.14)$$
$$OH + H \rightleftharpoons H_2O + h\nu \quad (6.15)$$
$$H_2 + O \rightleftharpoons H_2O + h\nu \quad (6.16)$$

Die chemischen Ratenkoeffizienten, Photoratenkoeffizienten und thermodynamischen Größen habe ich dem Modell B von [Beck *et al.* 1992] entnommen. Dieses Modell berücksichtigt keine chromosphärische UV-Strahlung.

Die relative Elektronenhäufigkeit wurde mit $2 \cdot 10^{-4}$ als konstant und das Strahlungsfeld als geometrisch verdünnt angenommen. Der geometrische Verdünnungsfaktor beträgt $(2R_*/R)^2$. Die relative Elektronenhäufigkeit und eine "Photonenhäufigkeit" werden als achte und neunte Spezies in das System aufgenommen, um die Programmierung der Ionisations– und Photoraten einfacher zu gestalten. Sie werden jedoch konstant gehalten, so daß in dem Modell zusätzlich zu den zwei Elementerhaltungssätzen für die Wasserstoff– und Sauerstoffatome zwei künstliche Erhaltungssätze für die Elektronen und Photonen impliziert sind.

Die Integration erfolgt beginnend bei $R = 2R_*$ über das Zeitintervall von 0 bis $1.5 \cdot 10^9$ Sekunden. Die Toleranz wird durch `rtol` $= 10^{-6}$ und `atol` $= 10^{-16}$ bestimmt.

Die mit CSP++ berechneten Spezieshäufigkeiten sind in Abbildung 6.9 dargestellt. Die Lösung weist dieselben Merkmale wie die von [Beck *et al.* 1992] veröffentlichte auf: Im Bereich konstanter Temperatur ($R < 5R_*$) bleiben die Häufigkeiten nahezu konstant, danach findet ein starker Anstieg der O_2-Häufigkeit statt, und die H_2O-Häufigkeit steigt kurz an, um danach wieder abzufallen. Diese Merkmale sind bei den vorliegenden Ergebnissen allerdings weniger ausgeprägt als bei dem Modell von [Beck *et al.* 1992], das sich aufgrund seiner geringern Windgeschwindigkeit näher an der kinetischen Gleichgewichtslage befindet.

Bei einer Berechnung desselben Problems mit `Radau5` wurde erneut die geringe Rechengeschwindikeit von CSP++ deutlich: `Radau5` benötigte 3.7 Sekunden `usr-Zeit`, während CSP++ 220 Sekunden `usr-Zeit` brauchte. Dieses Verhältnis läßt sich teilweise durch die Größe der gewählten Schrittweite erklären (siehe Abbildung 6.10): Die Schrittweite von `Radau5` ist erheblich größer. In der Abbildung ist zusätzlich die von CSP++ identifizierte aktive Zeitskala eingezeichnet: Sie liegt noch unter der von CSP++ (d.h. von der Schrittweitenbestimmung in `Dopri5`) gewählten Schrittweite.

Untersucht man die von CSP++ berechneten Moden, so stellt man zunächst fest, daß vier Moden verschwindende Eigenwerte haben. Diese Moden entsprechen den vier im Modell implizierten Erhaltungssätzen.

Die anderen fünf Moden des neundimensionalen Problems sind relaxierend. Ihre Zeitskalen τ_i, Gesamteinflüsse $\mathbf{y}_{\mathrm{ge},i}$ und Amplituden f_i sind in Abbildung 6.11 dargestellt. Zusätzlich sind im Diagramm der Zeitskalen die hydrodynamische Zeitskala R/v durch einen durchgezogenen schwarzen Graphen und die von CSP++ bestimmte aktive Zeitskala τ_{akt} durch einen gepunkteten Graphen markiert.

Im Bereich konstanter Temperatur ($R < 5R_*$) ändern sich die Moden so wie die Lösung nur geringfügig. Die vier Moden mit deutlich kleineren Zeitskalen als die hydrodynamische Zeitskala habe auch kleine Amplituden und verbleibende Gesamteinflüsse. Deswegen werden drei von ihnen als relaxiert erkannt und vernachlässigt: Die aktive Zeitskala wird auf den Wert der vierten (grünen) gesetzt. Daß relative viele Moden relaxiert sind, entspricht

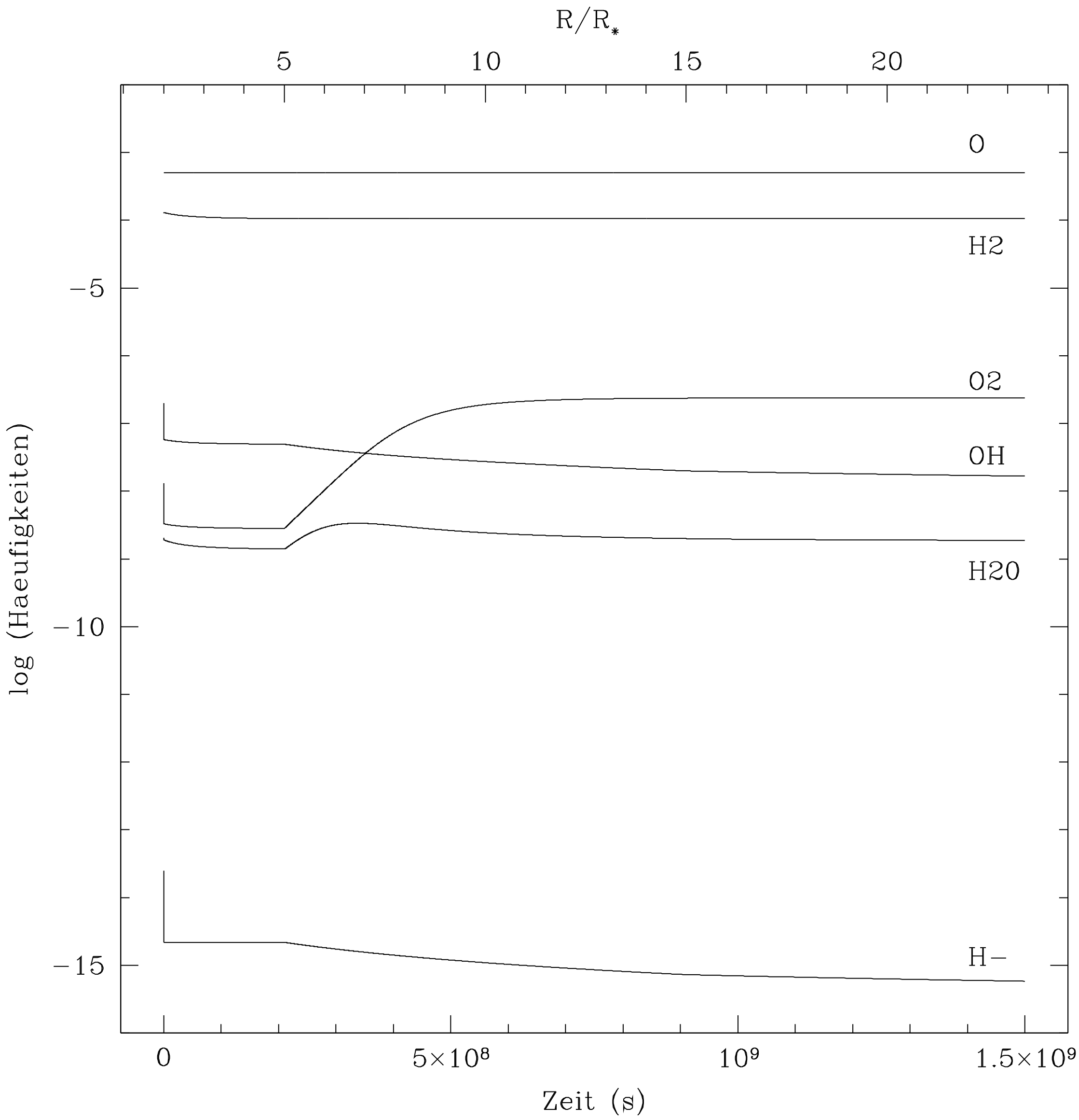


Abbildung 6.9: Mit CSP++ berechnete Spezieshäufigkeiten in der ZSH.

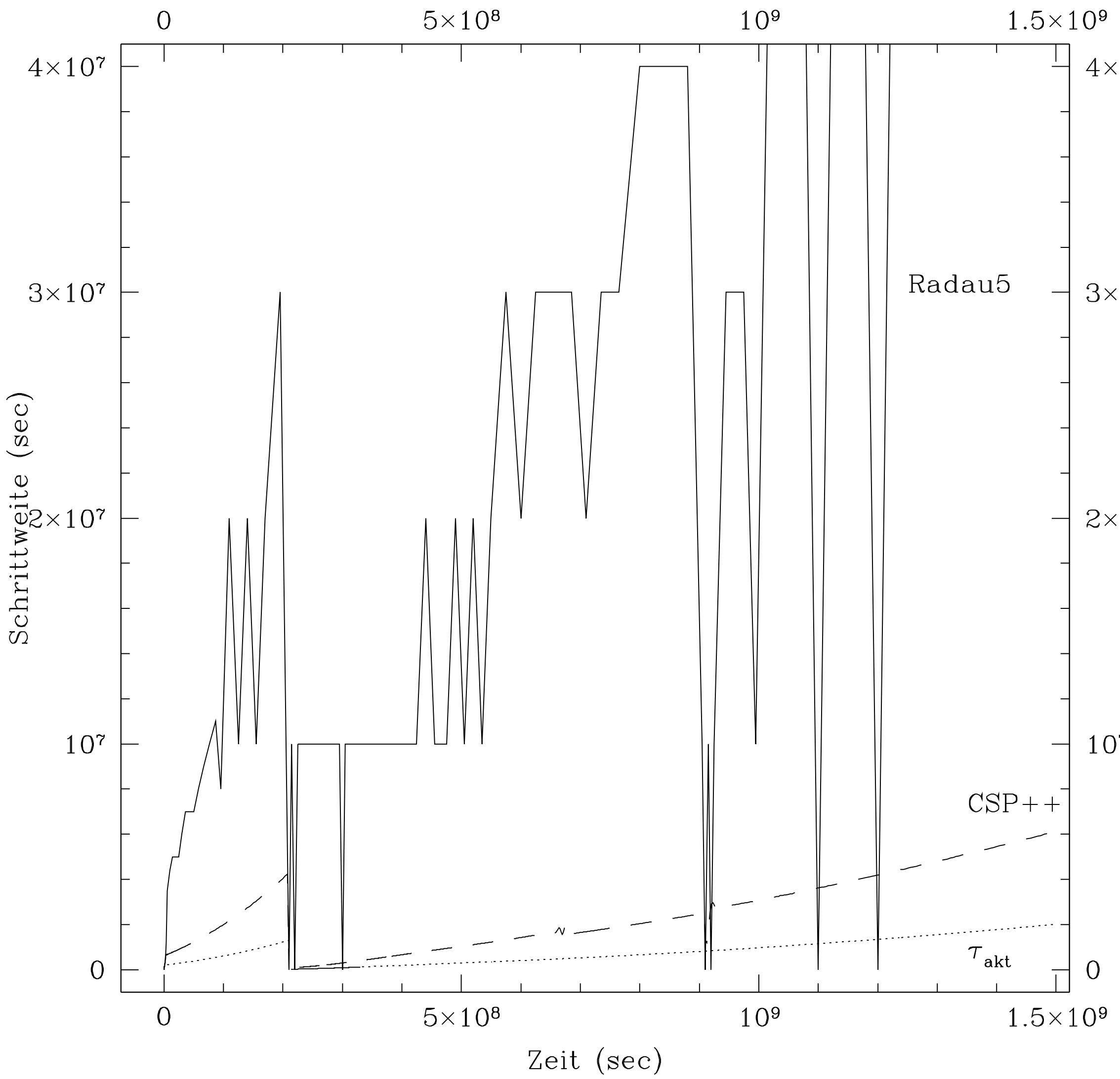


Abbildung 6.10: Schrittweiten der Integratoren und von CSP++ bestimmte aktive Zeitskala τ_{akt} des Systems. Die Außerhalb des Ausschnitts liegenden Punkte von `Radau5` haben eine Schrittweite von 10^8 Sekunden.

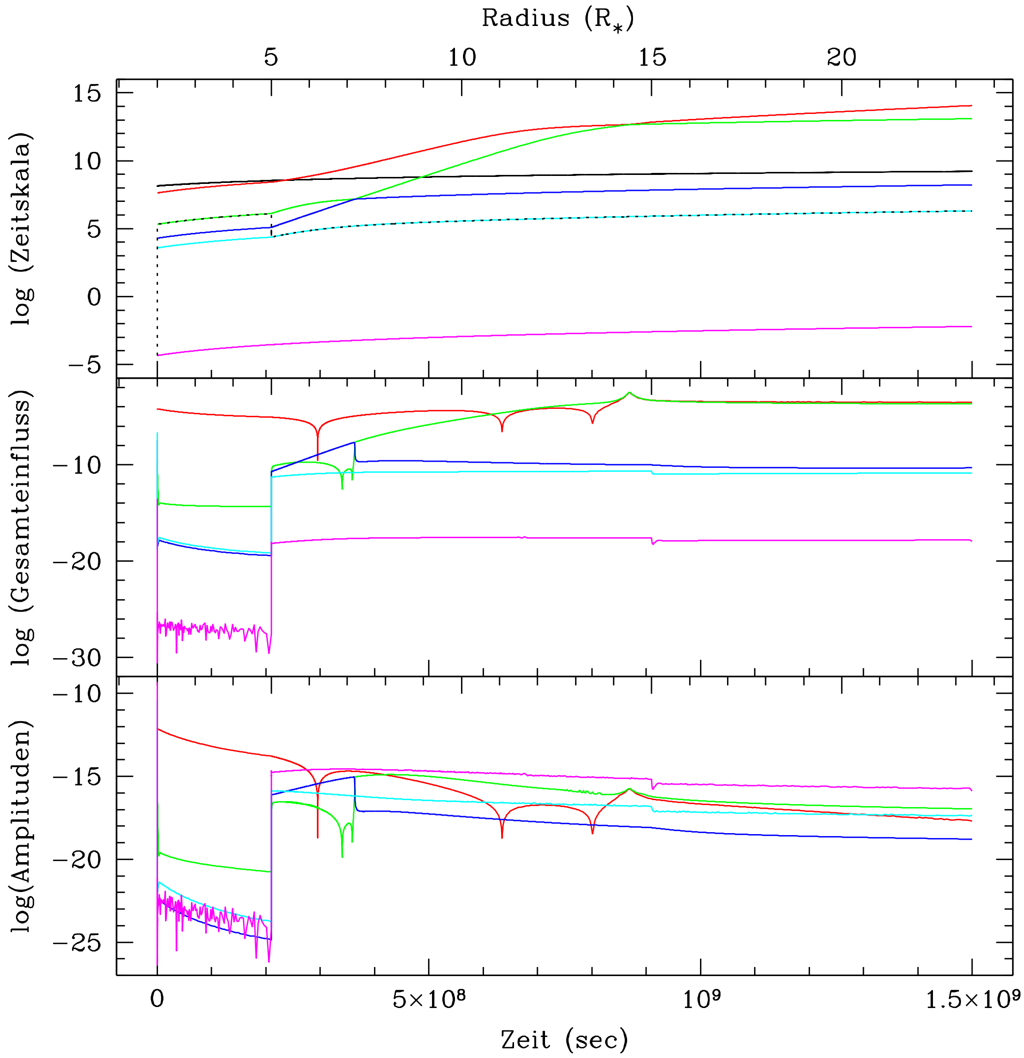


Abbildung 6.11: Zeitskalen τ_i, Gesamteinflüsse $\mathbf{y}_{\mathrm{ge},i}$ und Amplituden f_i der fünf relaxierenden Moden der M-Sternhüllenchemie. Die durchgezogene schwarze Linie ist die hydrodynamische Zeitskala R/v, die gepunktete Linie gibt die von CSP++ gefundene aktive Zeitskala τ_{akt} an.

der Aussage, daß das System sich relativ nah am kinetischen Gleichgewichtspunkt befindet.

In dem Bereich außerhalb $5R_*$ steigen die Zeitskalen zweier Moden stark an. Gleichzeitig steigt auch der verbleibende Gesamteinfluß einer Mode, der Gesamteinfluß der anderen bleibt unvermindert groß. Dies bedeutet, daß die Moden sich vom relaxierten Zustand entfernen. Sie sind repräsentieren die Reaktionen, die durch Ausfrieren aufgrund der verminderten Dichte und Temperatur weit vom Gleichgewichtszustand entfernt sind. Kennzeichnend für diesen Zustand ist die Tatsache, daß die hydrodynamische Zeitskala viel kleiner als diejenigen der Moden ist.

Auch die Gesamteinflüsse der anderen drei Moden steigen beim Einsetzen der Temperaturänderung bei $R = 5R_*$ an. Deswegen wird nur noch die Mode mit der kürzesten Zeitskala als relaxiert angesehen und entsprechend vernachlässigt: Die aktive Zeitskala wird auf den Zeitskalenwert der zweitschnellsten Mode, der türkisen, heruntergesetzt. Dies ist die Folge davon, daß sich das System von seiner (kinetischen) Gleichgewichtslage entfernt. Da die Zeitskala um ca. acht Größenordnungen kleiner als die aktive Zeitskala ist, wird das DGLS durch ihr Vernachlässigung so viel weniger steif, daß seine Lösung durch `Dopri5` möglich wird: Versucht man das ungenäherte System mit `Dopri5` zu lösen, dann wird die Integration aufgrund zu hoher Schrittzahl (10^5) bei $t = 15$ abgebrochen. Dies entspricht einer durchschnittlichen Schrittweite von $1.5 \cdot 10^{-4}$ und damit der Zeitskala der schnellsten Mode. Ein Vergleich mit der in Abbildung 6.10 dargestellten Schrittweite von CSP++ zeigt, daß die Vernachlässigung nur der schnellsten Mode eine Vergrößerung der Integrationsschrittweite um den Faktor 10^{10} ermöglicht.

Obwohl CSP diese enorme Steigerung der Schrittweite des expliziten Integrators `Dopri5` ermöglicht, kann an diesem Beispiel auch festgestellt werden, daß die Größe der von CSP++ gewählten Zeitschritte weit unter der Zeitskala der Änderung des Problems liegt. Insofern wird das Ziel, die Schrittweite des Integrators einzig an der Zeitskala der Lösung zu orientieren, nicht erreicht. Dies liegt darin begründet, daß CSP zwar die Zeitskalen der chemischen Reaktionen, nicht aber diejenigen der Änderung der äußeren physikalischen Bedingungen in der Analyse berücksichtigen kann. Deswegen sollte der Einsatz von CSP zur Simulation von Systemen, deren Zustand sich ständig in der Nähe eines sich langsam ändernden Gleichgewichtszustandes befindet, noch einmal prinzipiell geprüft werden.

Kapitel 7

Zusammenfassung und Ausblick

Ich habe das von [Lam und Goussis 1991] vorgeschlagene Verfahren *Computational Singular Perturbation* zur numerischen Integration steifer Systeme gewöhnlicher DGLen mit einem expliziten Integrationsalgorithmus vorgestellt. Es basiert auf der Zerlegung der rechten Seite des DGLSs in Moden, von denen die mit kurzen Zeitskalen assoziierten vernachlässigt werden.

Mit der Implementation CSP91 der Autoren konnte die praktische Durchführbarkeit des Verfahrens verifiziert werden. Die Analyse der Programmstruktur offenbarte jedoch eine so enge Kopplung zwischen den CSP-Näherungen einerseits und dem DGL-Löser andererseits, daß sich die Näherungsroutinen nicht als eigenständiges Modul aus dem Programm auskoppeln lassen.

Daraufhin habe ich das Verfahren in dem Modul CSP++ implementiert, das vollständig unabhängig von der Integrationsroutine und der Routine zur Berechnung der rechten Seite des DGLSs arbeitet. Es kann deswegen problemlos in bereits bestehende Programme zur Lösung von Systemen gewöhnlicher DGLen, wie denjenigen zur Simulation zirkumstellarer Hüllen, eingebunden werden. Obwohl das Modul in der Programmiersprache C++ realisiert wurde, konnte ich die Kompatibilität zu in C und FORTRAN geschriebenen Programmpaketen durch ein Interface in der Syntax dieser Sprachen sicherstellen.

Das von mir implementierte Modul ist in Verbindung mit der Integrationsroutine `Dopri5` dem Programm CSP91 insofern überlegen, als die durchschnittliche Integrationsschrittweite größer ist. Diese Überlegenheit ist einerseits auf die nicht direkt von CSP abhängige Schrittweitensteuerung durch `Dopri5` zurückzuführen, andererseits werden die Moden durch die verwendete Eigenvektorzerlegung mit Konvergenzkontrolle genauer bestimmt als durch die Blockdiagonalisierung in CSP91.

Eine Untersuchung der für Probleme unterschiedlicher Dimension benötigten Rechenzeit offenbart bei CSP++ einen Anstieg mit einer höheren Potenz (> 2) der Dimension als bei der impliziten Lösungsroutine `Radau5`. Deswegen sollten implizite oder semi-implizite Integratoren zur Lösung großer Probleme der gegenwärtigen Implementation des CSP-Verfahrens vorgezogen werden. Bei der selbstkonsistenten Simulation von ZSHn mit den bestehenden Programmen ist dies nicht möglich. Ein Einsatz von CSP++ erscheint aber auch wenig sinnvoll, bis die benötigte Rechenzeit deutlich gesenkt worden ist.

CSP++ wird durch die Eigenvektorzerlegung erheblich verlangsamt, weil die verwendete LAPACK-Routine keinen Gebrauch von der vorhandenen guten Näherung durch die Eigenvektoren am vorangegangenen Lösungspunkt macht. Eine entsprechend modifizierte

Zerlegungsroutine könnte die Ausführung des gesamten Moduls deutlich beschleunigen. Ein Einsatz des Moduls in den Simulationsprogrammen für ZSH könnte dann sinnvoll und möglich zu sein.

Zudem kann bei den chemischen Ratennetzwerken zur Beschreibung von ZSHn das Problem auftreten, daß die berechneten Eigenvektoren der Jakobimatrix nicht linear unabhängig sind, also keine Basis des Vektorraums bilden. Dieses Problem könnte z.B. durch eine Verwendung der Schurvektorzerlegung umgangen werden, denn die Schurvektoren jeder reellen quadratischen Matrix bilden eine Orthonormalsystem.

Am Beispiel der Berechnung eines stark vereinfachten chemischen Ratennetzwerkes auf einer vorgegebenen Hüllenstruktur, wurde deutlich, daß das Verfahren im derzeitgen Stadium zwar funktionstüchtig ist, aber noch nicht leisungsfähig genung, um bei einer selbstkonsistenten Simulation einer ZSH gewinnbringend eingesetzt werden zu können. In diesem Zusammenhang muß insbesondere die Wirkung sich langsam ändernder äußerer Bedingungen auf das Näherungsverfahren bedacht werden.

Selbst wenn eine weitere Optimierung des Verfahrens seine Leistungfähigkeit nicht ausreichend für einen Einsatz bei der selbstkonsistenten Simulation von ZSHn steigern können sollte, lassen die Möglichkeiten zur Gewinnung physikalisch-chemischer Einsichten durch Analyse der berechneten Moden und ihrer Eigenschaften das Verfahren als attraktives Werkzeug bei der Erkenntnissuche erscheinen.

Danksagung

Jedem einzelnen Mitglied des Institutes gebührt Dank für seinen/ihren jeweiligen Beitrag zur angenehmen Atmosphäre in der Arbeitsgruppe.

Besonders danken möchte ich Daniel Krüger für seine Betreuung dieser Arbeit und die vielen darüber hinausgehenden Diskussionen.

Im Gespräch mit Christian Chang habe ich tiefgreifende Einsichten gewonnen, und ebenso geholfen haben mir die spontane Unterstützung von Beate Patzer und die Erläuterungen von Holger Beck zu seinem Simulationsprogramm ([Beck *et al.* 1992]).

Literaturverzeichnis

[Anderson *et al.* 1992] Anderson, E., Z.Bai, Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostouchov, S., und Sorensen, D., 1992. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadephia. ftp://netlib.att.com/netlib/lapack.

[Beck *et al.* 1992] Beck, H., Gail, H.-P., Henkel, R., und Sedlmayr, E., 1992. Chemistry in circumstellar shells. I. Chromospheric radiation fields and dust formation in optically thin shells of m-giants. *Astronomy and Astrophysics*, **265**, 626–642.

[Brezinski 1989] Brezinski, C., Editor, 1989. *Numerical and Applied Mathematics*, Band 1.2 der Reihe *IMACS Transactions on Scientific Computing '88*. J.C. Baltzer AG Scientific Publishing Co., Basel, Switzerland. (*IMACS Annals on Computing and Applied Mathematics*).

[Buchhammer 1994] Buchhammer, J., 1994. Stationäre staubbildende Winde mit Wellendruck. Diplomarbeit, Fachbereich 04 (Physik) der Technischen Universität Berlin.

[com1992] University of Sydney, July 5–10 1992. *24th International Symposium on Combustion*, Sydney, Australia.

[Curtiss und Hirschfelder 1952] Curtiss, C. F. und Hirschfelder, J. O., 1952. Interation of stiff equations. *Proc. Nat. Acad. Sci.*, **38**, 235–243.

[Fleischer *et al.* 1992] Fleischer, A. J., Gauger, A., und Sedlmayr, E., 1992. Circumstellar dust shells around long-period variables. I. Dynamical models of c-stars including dust formation, growth and evaporation. *Astronomy and Astrophysics*, **266**, 321–339.

[Gail und Sedlmayr 1986] Gail, H. P. und Sedlmayr, E., 1986. The primary condensation process for dust around late M-type stars. *Astronomy and Astrophysics*, **166**, 225–236.

[Gail und Sedlmayr 1987] Gail, H. P. und Sedlmayr, E., 1987. Dust formation in stellar winds. III. Self-consistent models for dust-driven winds around c-stars. *Astronomy and Astrophysics*, **171**, 197–204.

[Goeres *et al.* 1988] Goeres, A., Henkel, R., Sedlmayr, E., und Gail, H.-P., 1988. Nonequilibrium chemistry in circumstellar shells. *Reviews in Modern Astronomy*, **1**, 231–248.

[Golub und vanLoan 1989] Golub, G. H. und van Loan, C. F., 1989. *Matrix Computations*. The Johns Hopkins University Press, 2. Auflage.

[Goussis und Lam] Goussis, D. A. und Lam, S. H., . *An Iterative Algorithm for Block-Diagonalization of a Matrix*. Department of Mechanial and Aerospace Engineering, Princeton University, D. A. Goussis, Mechanical Engineering Department, University of Patras, 26110 Rio-Patras, Greece. Datum unbekannt.

[Goussis und Lam 1992] Goussis, D. A. und Lam, S. H., July 5–10 1992. A study of homogeneous methanol oxidation kinetics using CSP. In combustion [com1992].

[Hairer *et al.* 1987] Hairer, E., Nørsett, S. P., und Wanner, G., 1987. *Solving Ordinary Differential Equations I*. Springer-Verlag, Berlin.

[Hairer und Wanner 1991] Hairer, E. und Wanner, G., 1991. *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin.

[Heroth 1995] Heroth, J., 1995. Adaptive Dimensionsreduktion chemischer Reaktionssysteme. Diplomarbeit, Freie Universität Berlin, Fachbereich Mathematik und Informatik.

[Knapp 1991] Knapp, G. R., 1991. Death and transfiguration — Mass loss from cool evolved giant stars and evolution to planetary nebulae. In *Frontiers of stellar evolution (A92-51676 22-90)*, Seiten 229–263. Astronomical Society of the Pacific, San Francisco.

[Krüger und Sedlmayr 1994] Krüger, D. und Sedlmayr, E., 1994. Two-fluid models for stationary dust driven winds. I. Momentum and energy balance. *Astronomy and Astrophysics*, **290**, 573–589.

[Krüger und Sedlmayr 1996] Krüger, D. und Sedlmayr, E., 1996. Two-fluid models for stationary dust driven winds. II. The grain size distribution in consideration of drift. *Astronomy and Astrophysics*, . submitted.

[Laidler 1970] Laidler, K. J., 1970. *Reaktionskinetik I, Homogene Gasreaktionen*. Nummer 290/290a* in B·I-Hochschultaschenbücher. Bibliographisches Institut, Mannheim/Wien/Zürich.

[Lam 1993] Lam, S. H., 1993. Using CSP to understand complex chemical-kinetics. *Combustion Science and Technology*, **89**(5–6), 375–404.

[Lam und Goussis 1989] Lam, S. H. und Goussis, D. A., 1989. Basic theory and demonstrations of computational singular perturbation for stiff equations. In Brezinski [Brezinski 1989], Seiten 487–492.

[Lam und Goussis 1991] Lam, S. H. und Goussis, D. A., 1991. Conventional asymptotics and computational singular perturbation for simplified kinetics modelling. In Smooke [Smooke 1991], Kapitel 10, Seiten 227–242.

[Lam und Goussis 1994] Lam, S. H. und Goussis, D. A., 1994. The CSP method for simplifying kinetics. *Inernational Journal of Chemical Kinetics*, **26**, 461–486.

[Loup *et al.* 1993] Loup, C., Forveille, T., Omont, A., und Paul, J. F., June 1993. CO and HCN observations of circumstellar envelopes. A catalogue — Mass loss rates and distributions. *Astronomy and Astrophysics Supplement Series*, **99**, 291–377.

[Lüttke 1995] Lüttke, M., 1995. Ein effizientes Rechenverfahren zur selbstkonsistenten Modellierung staubbildender Sterne. Diplomarbeit, Technische Universität Berlin, Institut für Astronomie und Astrophysik.

[Maas und Pope 1992] Maas, U. und Pope, S. B., 1992. Simplifying chemical kinetics: Intrinsic low-dimensional manifolds in composition space. *Combustion and Flame*, **88**, 239–264.

[Press *et al.* 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., und Flannery, B. P., 1992. *Numerical Recipes in C*. Cambridge University Press, 2. Auflage.

[Robertson 1966] Robertson, H. H., 1966. The solution of a set of reaction rate equations. In Walsh [Walsh 1966], Seiten 178–182.

[Schöll 1993] Schöll, E., 1993. Theoretische Physik IV: Thermodynamik und Statistik. Skript zur Vorlesung im WS 92/93 an der TU Berlin.

[Smooke 1991] Smooke, M. D., Editor, 1991. *Reduced Kinetic Mechanisms and Asymptotic Approximations for Methane-Air Flames*. Lecture Notes in Physics. Springer-Verlag, Berlin, Heidelberg.

[Stroustrup 1991] Stroustrup, B., 1991. *The C++ programming language*. Addison-Wesley Publishing Company, 2. Auflage.

[Walsh 1966] Walsh, J., Editor, 1966. *Numerical Analysis, An Introduction*. Academic Press.

[Williams 1985] Williams, F., 1985. *Combustion Theory, The Fundamental Theory of Chemically Reacting Systems*. Benjamin/Cummings Publ. Co., Menlo Park, Ca., 2nd Auflage.

[Zurmühl 1964] Zurmühl, R., 1964. *Matrizen*. Springer-Verlag, Berlin/Göttingen/Heidelberg, 4. Auflage.

Anhang A

Eine C++–Klassenbibliothek für Lineare Algebra

Um die in einem in C++ geschriebenen Modul eines hauptsächlich in FORTRAN geschriebenen Programmpakets enthaltenen Vektor- und Matrixoperationen effizient zu programmieren, habe ich die hier vorgestellte Bibliothek von C++-Klassen zur Repräsentation der Vektoren und Matrizen entworfen und benutzt.

A.1 Anforderungen

- Kompatibilität zu FORTRAN–Routinen
- einfache Bearbeitung von Unterblöcken einer Matrix
- „mathematische“ Notation bei der Programmierung von Matrixoperationen

A.2 Klassendesign

A.2.1 Repräsentation von Matrizen und Vektoren

Objekte der Klasse `Matrix` repräsentieren, wie der Name schon sagt, Matrizen. Die Elemente werden in einem Array spaltenweise gespeichert. Dadurch können sie beim Aufruf einer FORTRAN–Routine durch einen Pointer auf das erste Element zur weiteren Bearbeitung übergeben werden. Umgekehrt kann ohne ein Kopieren der Elemente aus einem gegebenen Array ein `Matrix`–Objekt konstruiert werden.

Die Vektoren repräsentierende Klasse `Vector` ist von der Klasse `Matrix` abgeleitet, wobei die Anzahl der Spalten auf eins gesetzt wird. Mit einem Vektor als $(n \times 1)$–Matrix können somit auch alle Matrixoperationen ausgeführt werden. Lediglich eine kleine Anzahl von Operatoren wird für Vektoren entsprechend ihrer mathematischen Interpretation neu definiert; so bedeutet z.B. der `operator *= (Vector& v, Matrix& m)` „anwenden (von links Multiplizieren) der Matrix `m`“, während der `operator *= (Matrix& m1, Matrix& m2)` bedeutet: „von rechts Multiplizieren der Matrix `m2`“.

A.2.2 Repräsentation von Unterblöcken von Matrizen

Um mit einem Unterblock eines `Matrix`–Objektes wie mit einer einzelnen Matrix arbeiten zu können, wurde die Klasse `MatrixBlock` entworfen. Sie alloziert niemals eigenen Speicher für Elemente, sondern enthält nur Referenzen auf die Spalten des Unterblocks innerhalb des Arrays der gesamten Matrix. Dadurch sind die Elemente eines `MatrixBlock`–Objektes spaltenweise gespeichert, ohne daß Aussagen über die Lage der einzelnen Spalten zueinander impliziert werden.

Die in `Matrix` verwendete Anordnung der Elemente im Speicher ist offensichtlich ein Spezialfall derjenigen in `MatrixBlock`: alle Spalten direkt hintereinander. Deshalb wird `MatrixBlock` als Basisklasse für `Matrix` verwendet. Durch Definition der meisten Matrixoperationen für die Klasse `MatrixBlock` und ihre Vererbung an `Matrix` kann nun sichergestellt werden, daß Objekte der beiden Klassen weitestgehend auf die gleiche Weise verwendet werden können. Einzig die Kompatibilität zu FORTRAN–Arrays ist bei `MatrixBlock`–Objekten nicht gegeben.

Auf Reihen und Spalten von Matrizen kann mit den von `MatrixBlock` abgeleiteten Klassen `MatrixRow` und `MatrixCol` zugegriffen werden: Die Dimensionen werden bei Objekten dieser Klassen automatisch entsprechend der ursprünglichen Matrix bzw. gleich eins eingestellt.

Bei Objekten der drei referierenden Klassen `MatrixBlock`, `MatrixRow` und `MatrixCol` muß darauf geachtet werden, daß sie bei jeder Verwendung auf die Existenz derjenigen Objekte, auf die sie referieren, angewiesen sind.

A.2.3 Binäre Operationen

Operationen mit zwei Argumenten, von denen eines eine Matrix ist, werden als Memberfunktionen der Klasse `MatrixBlock` definiert und möglichst durch Überladen von Operatoren intuitiv und übersichtlich verwendbar gemacht; z.B. wird das Hinzuaddieren einer `Matrix B` zu einer `Matrix A` als `A += B` programmiert.

A.2.4 Operationen mit drei Argumenten

Operationen, die drei Argumente benötigen, z.B. $A = B * C$, können nicht direkt durch Operatorüberladen definiert werden, weil in C++ nur einen einziger tenärer Operator zur Verfügung steht: `? :`. Um das Erzeugen und Kopieren von großen temporären Objekten, z.B. eines `Matrix`–Objektes mit den Elementen von $B*C$, zu vermeiden, wurde die abstrakte Klasse `LazyMatrix` definiert, die eine Matrix nicht durch Abspeichern aller Elemente repräsentiert, sondern durch Referenzen auf zwei Matrizen und eine virtuelle Anweisung, wie aus diesen beiden Matrizen die Elemente zu bestimmen sind. Diese Anweisung wird dann erst bei der Ausführung des zweiten Operators (in dem Beispiel der Zuweisung „=“), wenn auch das dritte Argument (`A`) bekannt ist, ausgeführt.

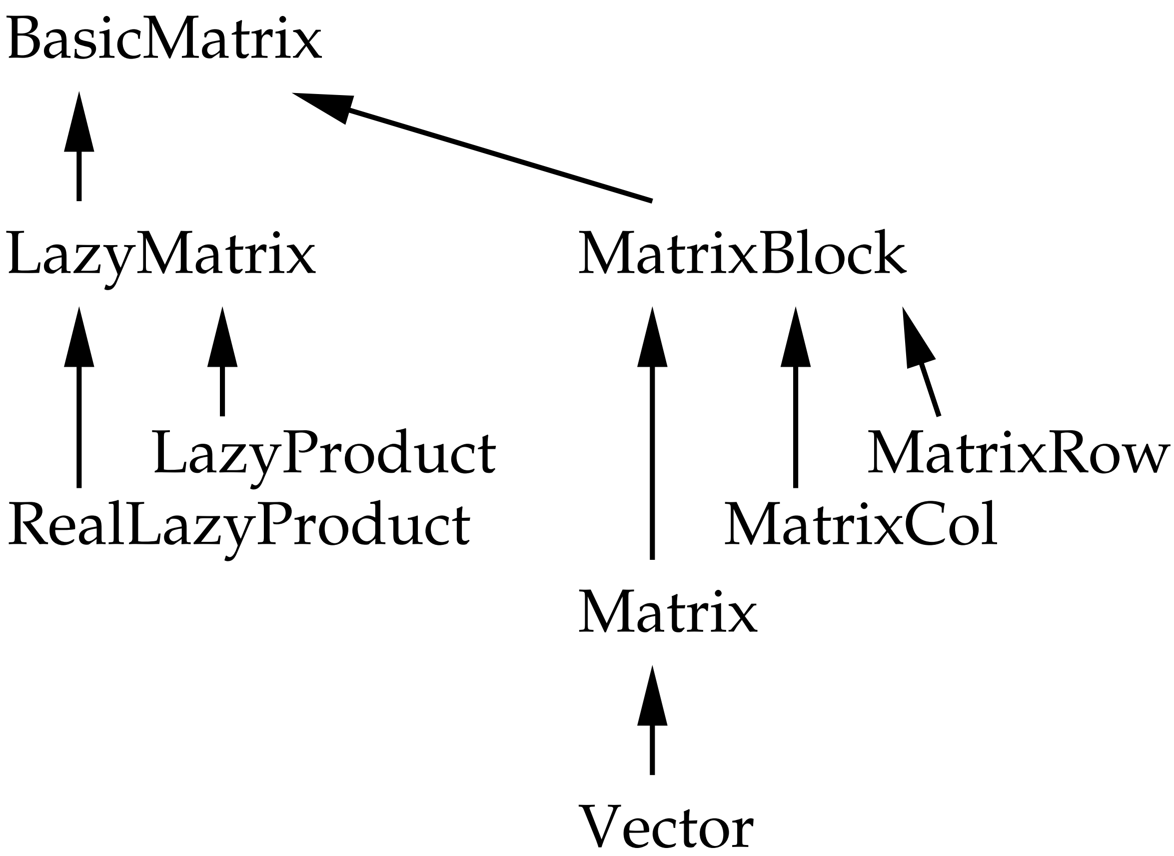
Je nach benötigter Operation werden von `LazyMatrix` Klassen mit der passenden Anweisung zur Bestimmung der Matrixelemente abgeleitet. Z.B. wurde für die Zuweisung eines Produktes zweier Matrizen die Klasse `LazyProduct` definiert. Ein Objekt dieser Klasse wird von dem `operator * (MatrixBlock&, MatrixBlock&)` zurückgegeben und die Anweisung zur Berechnung der Elemente erst von dem `operator = (MatrixBlock&,`

`LazyMatrix&)` ausgeführt. Die Klasse `RealLazyProduct` bewerkstelligt die Zuweisung eines Produktes von einem Skalar und einer Matrix analog.

A.2.5 Überblick

Während sich die Klassen `MatrixBlock` und `LazyMatrix` in der Art der internen Darstellung der Matrixelemente unterscheiden, repräsentieren beide doch Matrizen im mathematischen Sinn. Diese Gemeinsamkeit wird zur Ableitung der beiden Klassen von einer gemeinsamen, abstrakten Basisklasse `BasicMatrix` genutzt. Diese Klasse enthält im wesentlichen die Dimensionen der Matrizen, Ausgabefunktionen und deklariert virtuelle Funktionen zum Zugriff auf einzelne Elemente.

Insgesamt ergibt sich damit folgende Vererbunghierachie:



A.3 Komplexe Operationen

A.3.1 Kompatibilität mit anderen Bibliotheken

Aufgrund der Kompatibilität der Elementspeicherung zur Matrizendarstellung in FORTRAN können Routinen aus dem größten Teil der vorhandenen, bewährten und optimierten

Bibliotheken für die Lösung komplexer Probleme der numerischen linearen Algebra eingebunden werden.

Z.B. verwende ich LAPACK–Routinen zur Bestimmung der Eigen– und Schurvektorzerlegungen einer Matrix.

A.3.2 Neu implementierte Algorithmen

Einfachere Algorithmen, die evtl. nicht sofort in den üblichen Bibliotheken gefunden werden, können der Bibliothek natürlich problemlos selber hinzugefügt werden. So enthält sie z.B. bereits eine dem modifizierten Gram-Schmidt-Verfahren aus [Golub und vanLoan 1989] nachempfundene Routine zur Biorthonormalisierung zweier Matrizen.

A.4 Listing der Headerdatei mit den Klassendefinitionen

```
#ifndef _jola_h
#define _jola_h 100 * 1 + 0

// dies ist jola 1.0

#include <iostream.h>
#include "myenv.h"

typedef double REAL;

class BasicMatrix;
class MatrixBlock;
class MatrixCol;
class MatrixRow;
class Matrix;
class Vector;
class LazyMatrix;
class LazyProduct;
class RealLazyProduct;

////////////////////////////////////////////////////////////////////////////////
class BasicMatrix                        // abstract base class of all others //
////////////////////////////////////////////////////////////////////////////////
{
// data ////////////////////////////////////////////////////////////////////////
private:
  // Validation code:
  int valid_code;
  // Matrix validation code value:
  enum { MATRIX_val_code = 1000 * ('j'+'o'+'l'+'a') + _jola_h };

protected:
  short nrows; // No. of rows
  short ncols; // No. of columns

// member functions ////////////////////////////////////////////////////////////
public:
// Con-/Destructor: //
  inline BasicMatrix (const short nr, const short nc);
  virtual ~BasicMatrix ();

// Status inquires: //
  inline short q_nrows (void) const;    // returns no. of rows
  inline short q_ncols (void) const;    // returns no. of columns
  inline void is_valid (void) const;    // aborts program if not an object!
```

```
// Access to individual elements as rvalues (i.e. read-only): //
  virtual const REAL operator () (const short rown, const short coln) const = 0;

// Comparisions: //
  inline boolean same_dimensions (const BasicMatrix& other) const;

// Return element if 1x1 matrix: //
  REAL scalar (void) const;
};

// related functions ///////////////////////////////////////////////////////////
ostream& operator << (ostream& ostr, const BasicMatrix& ma);

////////////////////////////////////////////////////////////////////////////////
class MatrixBlock // Elements are stored as arrays containing one column each //
////////////////////////////////////////////////////////////////////////////////
  : public BasicMatrix
{
friend class LazyProduct;
friend class RealLazyProduct;

// data ////////////////////////////////////////////////////////////////////////
protected:
  const MatrixBlock& original;      // object containing actual elements
  REAL ** index;     // index[i] = &matrix(0,i) (col index)

// member functions ////////////////////////////////////////////////////////////
// constructors: //
  // Constructor to be used by Matrix::Matrix ()
  // doesn't initialize elements of index:
  MatrixBlock (const short nrows, const short ncols);

public:
  // Copy (e.g. for temporaries):
  MatrixBlock (const MatrixBlock& other);
  // Refer to block (rows: row_lwb--row_upb, cols: col_lwb--col_upb) of other:
  MatrixBlock (const MatrixBlock& other, const short row_lwb,
       const short row_upb, const short col_lwb,
       const short col_upb);

// Destructor: //
  ~MatrixBlock ();

// Access: //
  // Access to arrays of columns:
  inline REAL * q_column (const short col_index) const;

  // Access individual elements as lvalues:
  inline       REAL& operator () (const short rown, const short coln);
  // Access individual elements as rvalues:
  inline const REAL  operator () (const short rown, const short coln) const;

// Assignments: //
  inline  MatrixBlock& clear (void);    // Clear the matrix (fill out with 0)
          MatrixBlock& unity (void);       // it's a unity matrix afterwards
                                           // (matrix needn't be square)
  // (beware: operator = (...) is not inherited by derived classes!)
  // Assignment of REAL to all the elements:
          MatrixBlock& operator = (const REAL val);
  // Matrix assignements (beware: source MUST NOT depend on *this!!!):
          MatrixBlock& operator = (const BasicMatrix& source);
          MatrixBlock& operator = (const MatrixBlock& source);

// Modifications of the matrix as a mathematical object: //
  // transpose a SQUARE matrix:
         MatrixBlock& transpose (void);
```

```
  // Multiply all elements by a scalar value:
         MatrixBlock& operator *=  (const double val);
  // Return a matrix which is the product of *this and a scalar value:
  inline RealLazyProduct operator * (const double val);

  MatrixBlock& operator += (const MatrixBlock& source);
  MatrixBlock& operator -= (const MatrixBlock& source);
  // += scalar*source:
  MatrixBlock& add (const double scalar, const MatrixBlock& source);
  // Matrix multiplication:
  // (make SURE that the product is NEITHER assigned to *this NOR to right!)
  inline LazyProduct operator * (const MatrixBlock& right);
  // inplace multiplication of *this with source:
  MatrixBlock& operator *= (const MatrixBlock& source);

// Modifications of the elements (one by one): //
  MatrixBlock& abs (void); // Take the absolute value of each element
  MatrixBlock& sqr (void);      // Square each element
  MatrixBlock& sqrt(void);      // Take the square root of each element

  MatrixBlock& operator +=  (const double val); // Add to each element
  MatrixBlock& operator -=  (const double val); // Take from each element
  // Multiply by corresponding element:
  MatrixBlock& element_mult (const MatrixBlock& source);
  // Divide by corresponding element:
  MatrixBlock& element_div  (const MatrixBlock& source);

// Comparisions: //
  boolean operator ==  (const REAL val) const; // ? all elems == val
  boolean operator !=  (const REAL val) const; // ? all elems != val
  boolean operator <   (const REAL val) const; // ? all elems <  val
  boolean operator <=  (const REAL val) const; // ? all elems <= val
  boolean operator >   (const REAL val) const; // ? all elems >  val
  boolean operator >=  (const REAL val) const; // ? all elems >= val

  boolean operator == (const MatrixBlock& m) const; //compare element by element
  boolean operator <  (const MatrixBlock& m) const; //compare element by element
  boolean operator >  (const MatrixBlock& m) const; //compare element by element
  boolean operator <= (const MatrixBlock& m) const; //compare element by element
  boolean operator >= (const MatrixBlock& m) const; //compare element by element
};

// related functions /////////////////////////////////////////////////////////
// Return a matrix which is the product of mb and a scalar value:
inline RealLazyProduct operator * (const REAL val, MatrixBlock& mb);
// comparisions:
inline boolean operator == (const REAL val, const MatrixBlock& mb);
inline boolean operator != (const REAL val, const MatrixBlock& mb);
inline boolean operator <  (const REAL val, const MatrixBlock& mb);
inline boolean operator >  (const REAL val, const MatrixBlock& mb);
inline boolean operator <= (const REAL val, const MatrixBlock& mb);
inline boolean operator >= (const REAL val, const MatrixBlock& mb);

////////////////////////////////////////////////////////////////////////////////
class MatrixRow                             // a single row of a given matrix //
////////////////////////////////////////////////////////////////////////////////
  : public MatrixBlock
{
// no data ///////////////////////////////////////////////////////////////////

// member functions //////////////////////////////////////////////////////////
public:
// Constructor: //
  // Refer to row no. no_row of other matrix:
  inline MatrixRow (const MatrixBlock& other, const short no_row);
```

```
// Assignments: //
  // to each element:
  inline MatrixRow& operator = (const REAL val);
  // element by element (dimensions must be the same):
  inline MatrixRow& operator = (const BasicMatrix& source);
  inline MatrixRow& operator = (const MatrixBlock& source);
  inline MatrixRow& operator = (const MatrixCol& source);
  inline MatrixRow& operator = (const Vector& source);
};

////////////////////////////////////////////////////////////////////////////////
class MatrixCol                              // a single column of a given matrix
////////////////////////////////////////////////////////////////////////////////
  : public MatrixBlock
{
// no data ////////////////////////////////////////////////////////////////////

// member function ////////////////////////////////////////////////////////////
public:
// Constructor: //
  // Refer to column no. no_row of other matrix:
  inline MatrixCol (const MatrixBlock& other, const short no_col);

// Assignments: //
  inline MatrixCol& operator = (const REAL val);
  inline MatrixCol& operator = (const BasicMatrix& source);
  inline MatrixCol& operator = (const MatrixBlock& source);
  inline MatrixCol& operator = (const MatrixRow& source);
};

////////////////////////////////////////////////////////////////////////////////
class Matrix                                         // eine eigenstaendige Matrix
////////////////////////////////////////////////////////////////////////////////
  : public MatrixBlock
{
// data ///////////////////////////////////////////////////////////////////////
private:
  const boolean newarray;             // was the array of elements allocated by
                                      // this objects constuctor or before?
protected:
  const int nelems;                   // no of elements (nrows*ncols)
  REAL * elements;                    // points to first element

// member functions ///////////////////////////////////////////////////////////
private:
  inline void allocate (const boolean make_new_array);  // help constructors

public:
// Constructors: //
  // copy:
  inline Matrix (const Matrix& other);     // (e.g. for temporaries)
  inline Matrix (const BasicMatrix&  other);
  inline Matrix (const MatrixBlock&  other);
  // allocate new array:
  inline Matrix (const short nrows, const short ncols);
  // use array ep for element storage, preserving its contents:
  inline Matrix (const short nrows, const short ncols, REAL *ep);
  // Read the matrix from istr, dimensions must be provided:
  Matrix (const short nr, const short nc, istream& istr);

// Destructor: //
  inline ~Matrix (void);

// Status inquires: //
  inline boolean is_newarray (void) const;
  inline int q_no_elems() const;
```

```
  inline REAL * q_elements (void) const;

// Assignments: //
  inline Matrix& operator = (const REAL val);
  inline Matrix& operator = (const BasicMatrix& source);
  inline Matrix& operator = (const MatrixBlock& source);
};

////////////////////////////////////////////////////////////////////////////////
class Vector                             // ein Vektor als einspaltige Matrix //
////////////////////////////////////////////////////////////////////////////////
  : public Matrix
{
// no data /////////////////////////////////////////////////////////////////////

// member functions ////////////////////////////////////////////////////////////
public:
// Constructors: //
  // make new object of identical appearance:
  inline Vector (const MatrixBlock&  other);
  inline Vector (const MatrixRow&  other);
  // allocate new array:
  inline Vector (const short no_elements);
  // use array ep for element storage, preserving its contents:
  inline Vector (const short no_elements, REAL *ep);

// Assignments: //
  inline Vector& operator = (const REAL val);     // Assignment to all the elems
  inline Vector& operator = (const BasicMatrix& source);
  inline Vector& operator = (const MatrixBlock& source);
  inline Vector& operator = (const MatrixRow& source);

// Access to individual elements: //
  inline const REAL  operator () (const short ne) const;  // rvalue
  inline       REAL& operator () (const short ne);        // lvalue

// operators with special meaning for vectors: //
  // apply operator (ie matrix) source; *this = source * *this:
  inline Vector& operator *= (const MatrixBlock& source);
  // multiply vector with scalar value:
  inline Vector& operator *= (const REAL val);
};

// related functions ///////////////////////////////////////////////////////////
ostream& operator << (ostream& ostr, const Vector& v);

////////////////////////////////////////////////////////////////////////////////
class LazyMatrix     // Matrix mit impliziter Anweisung zur Elementbestimmung //
////////////////////////////////////////////////////////////////////////////////
  : public BasicMatrix
{
friend MatrixBlock;

// no data /////////////////////////////////////////////////////////////////////

// member functions ////////////////////////////////////////////////////////////
private:
  // return value of element without checking ranges of indices:
  const REAL element (const short row, const short col) const;
public:
  LazyMatrix (const short row, const short col);
  // check ranges of indices and call element(row,col)
  const REAL operator () (const short row, const short col) const = 0;
};

////////////////////////////////////////////////////////////////////////////////
```

```
class LazyProduct                // Represent the product of two MatrixBlocks //
////////////////////////////////////////////////////////////////////////////////
  : public LazyMatrix
{
// data ////////////////////////////////////////////////////////////////////////
private:
  const MatrixBlock& left;
  const MatrixBlock& right;

// member functions ////////////////////////////////////////////////////////////
  // return value of element without checking ranges of indices:
  const REAL element (const short row, const short col) const;

public:
// constructor: //
  LazyProduct (const MatrixBlock& left, const MatrixBlock& right);
// return value of element: //
  const REAL operator () (const short row, const short col) const;
};

////////////////////////////////////////////////////////////////////////////////
class RealLazyProduct   // Represent the product of a MatrixBlock with a REAL //
////////////////////////////////////////////////////////////////////////////////
  : public LazyMatrix
{
// data ////////////////////////////////////////////////////////////////////////
private:
  const MatrixBlock& ma;
  const REAL val;

// member functions ////////////////////////////////////////////////////////////
// return value of element without checking ranges of indices:
  const REAL element (const short row, const short col) const;

public:
// constructor: //
  RealLazyProduct (const MatrixBlock& matrix, const REAL value);
// return value of element: //
  const REAL operator () (const short row, const short col) const;
};

#include "inline/jola.inl"

#endif // ifndef _jola_h
```

Anhang B

Listing der Deklarationen der CSP++-Klassen

Im Folgenden sind die Headerdateien der von CSP++ benutzten C++–Klassen und der Interfacefunktioen (Abschnitt B.5) aufgelistet.

B.1 Problem.h

```
#ifndef _Problem_h
#define _Problem_h

class Problem
{
friend class State;

  short dim;            // Dimension
  double rtol;          // relative Toleranz
  double atol;          // absolute Toleranz
  void (*rhs) (int *d, double *x, double *y, double *g);
                        // Funktion zur Berechnung von g = dy/dx
public:

  Problem (short d, double rt, double at,
   void (*r) (int *d, double *x, double *y, double *g));

  short q_dim (void) const; // return dim
};

#endif
```

B.2 State.h

```
#ifndef _State_h
#define _State_h

#include "jola.h"
#include "Problem.h"

class State {

  friend class Exhaustion;
  friend ostream& operator << (ostream& ostr, const State& s);
  friend istream& operator >> (istream&, State&);
```

```
  const Problem& prb;

  double xstore;            // Dummy zur Initialisierung von x
  double& x;
  Vector y;
  Vector y_tol;             // Toleranzvektor
  Vector g;
  Matrix *jacobiptr;
  struct {
    boolean write_x;
    boolean write_y;
    boolean write_y_tol;
    boolean write_g;
    boolean write_Jacobian;
  } flags;         // fuer Ausgabefunktion operator << (ostream&, const State&)

public:

  State (const Problem& p);
  State (const short d, double& xx, double *yy, const Problem& p);
  ~State (void);
  State& operator = (const State&);
  Matrix jacobi (void);     // gibt Jakobimatrix an diesem Punkt
  short q_dim (void) const; // quote prb.dim
  double q_x (void) const;  // quote x

                            // Kopieren der Vektorelemente von y nach ep[d]:
  void copy_y (const short d, double *ep) const;
                            // Kopieren der Vektorelemente von g nach ep[d]:
  void copy_g (const short d, double *ep) const;
  const Vector& rhs (void) const; // die unghenaeherte RHS der ODEs

private:
                            // die UNgenaeherte RHS des ODEs:
  Vector rhs (const double& x, const Vector& y) const;
};

#endif
```

B.3 Exhaustion.h

```
#ifndef _Exhaustion_h
#define _Exhaustion_h

#include "jola.h"
#include "State.h"

class Exhaustion {

  friend istream& operator >> (istream&, Exhaustion&);
  friend ostream& operator << (ostream&, const Exhaustion&);

protected:
  short ex;                // Anzahl der vernachlaessigten (exhausted) Moden
  Matrix basis, dualbasis; // die gefundenen Moden, sortiert nach Grosze der
                             // der entsprechenden Eigenwerte: grosz zuerst
  Vector timescale;        // mit den Moden assoziierte Zeitskalen:
                             // -1/Eigenwert; negativ fuer explosive Mode
  Matrix slow_proj;        // Projektionsmatrix in der langsamen Unterraum
  // ab hier sind die Daten auch vom jeweiligen Zustand abhaengig
  Vector amplitude;    // Amplituden der Moden
  Matrix influence;    // Soviel sollte die Mode in unendlicher Zeit bewirken
  Vector exhausted;    // 1, wenn Mode exhausted; sonst 0
```

```
  struct {
    boolean write_ex;
    boolean write_orthonormality;
    boolean active_timescale;
    boolean basis;
    boolean dualbasis;
    boolean timescales;
    boolean amplitudes;
    boolean influence;
    boolean exhausted;
    boolean slow_proj;
  } flags;    // fuer Ausgabefunktion operator << (ostream&, const Exhaustion&)

public:

  Exhaustion (const short d);      // es wird nichts genaehert
  Exhaustion (State& s);           // Naehrung entsprechend s

  // return genaeherte rhs der ODE:
  Vector slow_rhs (const State&) const;

  // 1, wenn Naehrung bei s mit Zeitskala tau gueltig ist;
  // 0, wenn ungueltig:
  boolean valid (const State& s, const double tau) const;

  // 1, wenn Naehrung bei s mit Zeitskala der aktiven Moden gueltig
  // 0, wenn ungueltig:
  boolean valid (const State& s) const;

  void adjust (State& s);     // passe Naeherung an Zustand s an.

  // mache Naehrung schwaecher, ie weniger Moden vernachlaessigen:
  void weaker (State&);

  void updates (State&); // von slow_proj und evtl. aehnlichem
  void decompose (State& s);   // Zerlegung (dh basis, dualbasis, timescales
                                 // zu Zustand s bilden

  //quote ex/amplitude/timescale/exhausted:
  short q_ex (void) const;
  double q_amplitude (const short& i) const;
  double q_timescale (const short& i) const;
  double q_exhausted (const short& i) const;
  void correction (State& s);

  // Ausgabe der gespeicherten Daten:
  ostream& mode (ostream& ostr, const short no) const;

private:

  // mache Naehrung staerker, ie mehr Moden vernachlaessigen:
  void stronger (State&);
  void eigenvectors (State& s);  // Zerlegung in Eigenvektren
  void schurvectors (State& s);  // Zerlegung in Schurvektoren
};

#if _jola_h != 100 * 1 + 0
#error jola.h liegt nicht in der Version 1.0 vor!
#endif

#endif // ifndef _Exhaustion_h
```

B.4 Csp.h

```
#ifndef _Csp_h
#define _Csp_h

#include <fstream.h>
#include "State.h"
#include "Exhaustion.h"

class Csp                   // ein solches Objekt enthaelt alles, was CSP
                            // braucht und leistet
{
  friend ostream& operator << (ostream&, const Csp&);
  friend istream& operator >> (istream&, Csp&);

  const char *mode_file;  // log-Datei
  const Problem& prb;     // Gemeinsames aller Zustaende
  State alt;              // backup, falls der letzte Schritt rueckgaengig
                          // gemacht werden musz

  Exhaustion approx;      // das wesentliche
  boolean first_call_to_work;  // bedeutet das, was es heisst!!!
  struct {
    boolean decomp_in_work;
    boolean correction_in_work;
    boolean adjust_in_work;
    boolean decomp_in_rhs;
    boolean correction_in_rhs;
    boolean adjust_in_rhs;
    boolean write_g_minus_gslow;
    boolean state_log;
    boolean approx_log;
  } flags;              // fuer operator << (ostream&, const Csp&)

public:
  // Initialisierung:
  Csp (const Problem& prb);
  // CSP-Naeherung der RHS des ODEs:
  Vector rhs (State& neu);
  // Naeherungsmethode (dh Zerlegung...) anpassen:
  void correction (State& s);
  // nicht benutzt:
  void correction (double *y, const short no_mode);
  char work (State& neu, double tau = 0);
  // gib Anzahl der vernachlaessigten Moden zurueck:
  short q_ex (void) const;
  // Ist auch hier und auf dieser timescale die
  // Naeherung gueltig?
  boolean valid (const State& neu, double *timescale) const;
  // Schreibe Logdatei
  void write_log (void) const;
};

#endif
```

B.5 access.h

```
#ifndef _access_h
#define _access_h

/* Aufrufe der Csp-Memberfunktionen in C- bzw. FORTRAN-Syntax: */

/* Initialisierung */
void csp_ini (const int *dim, const double *rtol, const double *atol,
```

```
      const void (*rhs) (int *d, double *x, double *y, double *g));

/* Berechne die Projektion in den langsamen Unterraum! (Csp.rhs(.)) */
void csp_rhs (const int *d, double *x, double *y, double *g);

/* in SOLOUT aufzurufen (Csp.work(.)) */
void csp_go (int *nr, double *xold, double *x, double y[], int *n,
        int *irtrn);

/* Vergiss alles und gib den allozierten Speicher wieder frei! */
void csp_end (void);

/* Gib die Anzahl (m) der zZ vernachlaessigtne Moden an! */
int csp_q_ex (void);

/* Ist die Naeherung zur Zeit x fuer den Zustand y
   auf einer Zeitskala tau gueltig? */
boolean csp_valid (int *n, double *x, double *y, double *tau);

/* Schreibe die naechste Zeile der Logdatei! */
void csp_log (void);

#endif
```

Erklärung

Hiermit erkläre ich, diese Abeit eigenständig und ohne fremde Hilfe durchgeführt und geschrieben zu haben.

Berlin, 28. September 1996

Johannes Kaiser