Parallele Zweischritt-W-Methoden

Dissertation

zur Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)

vorgelegt der

Mathematisch–Naturwissenschaftlich–Technischen Fakultät

(mathematisch–naturwissenschaftlicher Bereich) der Martin–Luther–Universität

Halle–Wittenberg

von Herrn Dipl.-Math. Helmut Podhaisky

geb. am 30. Juni 1973 in Halberstadt

Gutachter:

1. Herr Prof. Dr. R. Weiner, Martin–Luther–Universität Halle–Wittenberg
2. Herr Prof. Dr. B. Schmitt, Philipps-Universität Marburg
3. Herr Prof. Dr. K. Strehmel, Martin–Luther–Universität Halle–Wittenberg

Datum der Verteidigung: 10. Januar 2002

urn:nbn:de:gbv:3-000002903
[http://nbn-resolving.de/urn/resolver.pl?urn=nbn%3Ade%3Agbv%3A3-000002903]

Parallele Zweischritt-W-Methoden

Inhaltsverzeichnis

Abkürzungen und Symbole **iii**

1 Einleitung **1**

2 Parallele Zweischritt-W-Methoden **3**
2.1 Parallele Lösung gewöhnlicher Differentialgleichungen . . . 3
2.2 Definition von parallelen Zweischritt-W-Methoden . . . 6
2.3 Vereinfachende Bedingungen und Konvergenz . . . 7
2.4 Lineare Stabilität . . . 12
2.5 Über den Zusammenhang von PTSW-Verfahren und DIMSIMs . . . 13

3 Konstruktion von Parametersätzen **15**
3.1 Einleitung . . . 15
3.2 Steifgenaue Verfahren mit nilpotenter Stabilitätsmatrix $M(\infty)$ . . . 16
3.3 Optimierte PTSW-Verfahren . . . 20
3.3.1 Die Bedingung B($s+1$) . . . 20
3.3.2 Ein Fortran-Programm für die numerische Suche . . . 21
3.3.3 Zweistufige L(α)-stabile Verfahren . . . 22
3.3.4 Dreistufige L(α)-stabile Verfahren . . . 24
3.3.5 Vierstufige L(α)-stabile Verfahren . . . 27
3.3.6 Zweistufige A(α)-stabile Verfahren . . . 28
3.3.7 Dreistufige A(α)-stabile Verfahren . . . 28
3.3.8 Vierstufige A(α)-stabile Verfahren . . . 29
3.3.9 Zusammenfassung der Optimierungen . . . 29
3.4 Ausflug: Explizite Verfahren hoher Ordnung . . . 32

4 Implementierung von PTSW-Verfahren **35**
4.1 Einleitung . . . 35
4.2 Auswahl der Verfahren . . . 35
4.3 Der Startschritt . . . 36
4.4 Lösung der Stufengleichungen . . . 37
4.4.1 Vorbetrachtungen . . . 37
4.4.2 Lösung mit LU-Zerlegung . . . 38
4.4.3 Lösung mit Krylovverfahren . . . 39
4.5 Schrittweitensteuerung . . . 41
4.6 Anwendung auf differential-algebraische Systeme . . . 42
4.7 Programmierung und Parallelisierung . . . 43
4.7.1 Einleitung . . . 43
4.7.2 Nutzerschnittstelle . . . 43
4.7.3 Speicherverwaltung . . . 44
4.7.4 Parallelisierung . . . 44

5 Numerische Tests **47**
5.1 Benchmarks für ODE-Integratoren 47
5.2 Die Testumgebung 49
5.3 Vergleich mit RODAS und RADAU 49
5.4 Vergleich mit ROWMAP und VODPK 52
5.5 Diskussion und Konsequenzen der numerischen Experimente 56

6 Zusammenfassung und weiterführende Bemerkungen **59**

A Maple-Skripte **65**
A.1 Einleitung 65
A.2 Explizite Zweischritt-Runge-Kutta-Verfahren 65
A.3 Plotten von Stabilitätsgebieten 66
A.4 Konstruktion einer vierstufigen steifgenauen PTSW-Methode nach Satz 3.1 67
A.5 Fortran-Koeffizientensätze 68

Abkürzungen und Symbole

ODE	gewöhnliche Differentialgleichung (*ordinary differential equation*)
PTSW-Methode	parallele Zweischritt-W-Methode (*parallel two-step W-method*)
$a_{ij}, \gamma_{ij}, \gamma, b_i, v_i, c_i$	Koeffizienten einer s-stufigen PTSW-Methode (2.4)
$A, \Gamma, \beta, b^T, v^T, c$	Koeffizienten in Matrixschreibweise, $\beta = A + \Gamma$
C(q),Γ(q),CΓ(q), B(p)	vereinfachende Konsistenzbedingungen (2.7)
$V_0, V_1, F, D, P_s, S, C$	Matrizen in den Konsistenzbedingungen
h_m, σ_m	Zeitschrittweite, Schrittweitenverhältnis $\sigma_m = h_m/h_{m-1}$
$M(z), \varrho(M(z))$	Stabilitätsmatrix (2.16), Spektralradius
$z = h\lambda$	Stabilitätsvariable für $y' = \lambda y$
$\mathcal{S} = \{z : \varrho(M(z)) < 1\}$	Stabilitätsgebiet
$w = z/(1 - \gamma z)$	Transformierte Stabilitätsvariable
$\varphi(x) = \sum_{i=0}^{s} \varphi_i x^i = \prod_{i=1}^{s} (x - c_i)$	Frobeniuspolynom zu Knoten c_i, $\varphi_s = 1$
φ_i	Elementarsymmetrische Polynome
α	Winkel der A(α)- bzw. L(α)-Stabilität
C_{p+1}, C_{p+2}	Hauptfehlerterme für $y' = \lambda y$
$\mathbb{1}^T = (1, \ldots, 1)$	Einsen-Vektor
$e_i^T = (\delta_{ij})_{j=1,\ldots,s}$	i-ter Einheitsvektor
$X \otimes Y$	Kroneckerprodukt $X \otimes Y = \begin{pmatrix} x_{11}Y & x_{12}Y & \ldots \\ x_{21}Y & x_{22}Y & \ldots \\ \ldots & \ldots & \ldots \end{pmatrix}$

1 Einleitung

Die Modellierung physikalischer oder technischer Systeme führt häufig auf große Differentialgleichungssysteme, die praktisch nur durch leistungsfähige numerische Methoden auf schnellen Computern gelöst werden können. Da hierfür der Bedarf an Rechenleistung beliebig hoch sein kann, entsteht die Frage, ob auch für die inzwischen weit verbreiteten Parallelrechner angepaßte Integrationsverfahren konstruiert werden können. Dabei sind nicht nur Hochleistungsrechner mit vielen hundert Prozessoren, sondern gerade auch preiswerte Mittelklasserechner (sogenannte „Workgroupserver“) mit zwei bis acht Prozessoren interessant. Um diese Hardware gut ausnutzen zu können, erscheinen Integrationsverfahren, die *parallel bezüglich der Methode* sind, ideal. Sie lassen sich gut für die kleine Prozessorenzahl auslegen und so programmieren, daß ein Nutzer sie ohne Zusatzaufwand für die Parallelisierung verwenden kann.

Es hat sich jedoch in der Vergangenheit in umfangreichen Untersuchungen gezeigt, daß klassische Verfahrensansätze, wie z.B. Runge-Kutta-Verfahren, für eine derartige Parallelisierung bezüglich der Methode kaum geeignet sind. Um sinnvolle parallele Strukturen zu erhalten, sind Verallgemeinerungen z.B. in Form von externen Stufen notwendig. Das bringt einige Schwierigkeiten mit sich: die Verfahrensstruktur, damit auch die Auswahl geeigneter Verfahrensparameter, die Analyse der numerischen Eigenschaften und schließlich auch die Implementierung werden komplizierter. So kann es leicht passieren, daß neu konstruierte Verfahren zwar gut parallelisierbar, aber trotzdem enttäuschend ineffizient sind. Deshalb sind die sorgfältige Wahl des Integrationsschemas und die dazugehörige numerische Analyse entscheidend. Sieht man es pragmatisch, so sind die neuen parallelen Verfahren nur dann von praktischem Wert, wenn sie für relevante Klassen von Differentialgleichungssystemen bereits auf einem Prozessor mit bewährten sequentiellen Verfahren einigermaßen konkurrenzfähig sind. Denn nur dann läßt sich durch parallele Rechnung bei gleichmäßiger Lastverteilung auf mehrere Prozessoren ein erheblicher Vorteil gegenüber den sequentiellen Standardmethoden erzielen. Der konkrete Speedup hängt allerdings stark vom Differentialgleichungssystem, der verwendeten Hardware und der Implementierung des Integrationsverfahrens ab. *A priori* Abschätzungen sind kaum möglich, so daß die gemessene Rechenzeit für ein konkretes Beispiel und eine gewünschte Integrationsgenauigkeit zum Hauptkriterium bei der Bewertung eines Integrationsverfahrens wird. Umfangreiche numerische Tests nehmen deshalb auch in dieser Arbeit großen Raum ein.

Gegenstand der vorliegenden Arbeit ist die Herleitung, Analyse, Implementierung und Bewertung von parallelen Zweischritt-W-Methoden (*parallel two-step W-methods, PTSW-Verfahren*). Das betrachtete Integrationsschema erhält seine parallele Struktur durch externe Stufen, ganz ähnlich wie bei parallelen expliziten Zweischritt-Runge-Kutta-Verfahren. Um gute lineare Stabilitätseigenschaften zu ermöglichen, orientieren wir uns beim Design der PTSW-Verfahren stark an den linear-impliziten Stufengleichungen sequentieller W-Methoden. Das hergeleitete Stufenschema der PTSW-Verfahren ist bereits in idealer Weise parallel, jede Stufe kann unabhänig von den übrigen auf einem Prozessor berechnet werden. Auch hohe Ordnungen und hohe Stufenordnungen, auf die wir uns in dieser Arbeit konzentrieren, können mit Hilfe von vereinfachenden Konsistenzbedingungen leicht abgesichert werden. Schwierig sind die Stabilitätsanalyse, die Wahl der Koeffizienten und das Austesten verschiedener Steuerheuristiken für eine robuste und effiziente Implementierung. Die Ergebnisse der theoretischen Untersuchungen sind teilweise bereits in Fachzeitschriften publiziert. Aus Platzgründen mußte in diesen Publikationen auf eine genaue Beschreibung der Herleitung der Verfahren und ihrer Implementierung verzichtet werden. Wir möchten daher in der vorliegenden Arbeit gerade auch diese technischen Details, die für die Effizienz der PTSW-Verfahren entscheidend sind, ausführlich vorstellen und diskutieren.

In Kapitel 2 geben wir einen kurzen Überblick über parallele Integrationsverfahren, motivieren und definieren das Verfahrensschema der PTSW-Methoden. Wir zeigen, daß aus den mittels Taylorentwicklung

gewonnenen vereinfachenden Konsistenzbedingungen bereits die Konvergenz der Verfahren folgt. Eine Übertragung der Konzepte der A-Stabilität und vor allem der L-Stabilität auf die lineare Matrizenrekursion der Zweischritt-Verfahren liefert ein leistungsfähiges Kriterium zur Konstruktion. Wie wir abschließend zeigen, lassen sich PTSW-Verfahren auch als verallgemeinerte lineare Mehrschrittverfahren, konkret sogenannte DIMSIMs, deuten, bei denen die nichtlinearen impliziten Abhängigkeiten durch einen Schritt mit dem Newton-Verfahren eliminiert werden.

In Kapitel 3 werden konkrete Koeffizientensätze für PTSW-Verfahren hergeleitet. Wir zeigen, daß L-stabile Verfahren hoher Ordnung $p = s$ mit bis zu $s = 12$ Stufen existieren. Dabei verwenden wir die etwas restriktive Forderung nach Nilpotenz der Stabilitätsmatrix und erhalten eindeutig bestimmte Koeffizienten. Lösen wir uns von der Nilpotenzforderung, so können wir die gewonnenen Freiheitsgrade zur Optimierung der Verfahren einsetzen. Wir diskutieren bis $s = 4$ die Konstruktion aussichtsreicher Parametersätze, die z.B. eine zusätzliche Bedingung für Ordnung $p = s + 1$ erfüllen. Für $s = 2$ und $s = 3$ Stufen finden wir robuste Verfahren mit kleinen Fehlerkonstanten.

In Kapitel 4 diskutieren wir die Implementierung von PTSW-Verfahren mit variabler Schrittweite. Wir beschreiben eine geeignete Startstrategie und eine approximative Lösung der Stufengleichungssysteme mit Hilfe des Arnoldiverfahrens. Weil die Konvergenzordnung des Verfahrens unabhängig von der verwendeten Krylovdimension ist, dient das Krylovverfahren weniger zur Reduktion lokaler Fehler, als vielmehr zur Stabilisierung des gesamten Integrationsprozesses. Durch die residuumkontrollierte Wahl der Krylovdimension ergibt sich eine automatische Steifheitserkennung, die sich als sehr vorteilhaft erwiesen hat.

Anhand von Testproblemen vergleichen wir in Kapitel 5 die PTSW-Verfahren auf einem Parallelrechner mit sequentiellen Standardcodes. Für kleine Beispiele vergleichen wir mit dem impliziten Runge-Kutta-Code RADAU und der Rosenbrockmethode RODAS. Bei semidiskretisierten Reaktions-Diffusions-Gleichungen vergleichen wir mit den Krylov-Integratoren ROWMAP und VODPK. Wir sehen, daß für gewisse Probleme die neu konstruierten Verfahren bereits sequentiell konkurrenzfähig mit den Standardintegratoren sind und durch die Parallelisierung deutlich effizienter sein können. Wir beobachten, daß bei Krylovapproximation die Robustheit der Verfahren besonders kritisch ist und PTSW-Verfahren mit besserer Stabilität effizienter als PTSW-Verfahren höherer Ordnung sind.

Abschließend werden die Ergebnisse zusammengefaßt, und es werden Ansatzpunkte zur Weiterentwicklung der Verfahren formuliert. Im Anhang der Arbeit demonstrieren wir, wie mit Hilfe des Computeralgebra-Programmes Maple Konstruktionsschritte bei der Entwicklung der PTSW-Verfahren automatisiert werden können.

2 Parallele Zweischritt-W-Methoden

2.1 Parallele Lösung gewöhnlicher Differentialgleichungen

Wir betrachten in dieser Arbeit Anfangswertprobleme für gewöhnliche Differentialgleichungssysteme (*ordinary differential equations, ODEs*) der Form

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \quad y, f \in \mathbb{R}^n, \quad t \in [t_0, t_e], \tag{2.1}$$

und setzen eine eindeutige Lösung $y(t)$ voraus, die für die jeweiligen Ordnungsaussagen hinreichend glatt sei. Damit muß dann auch die rechte Seite f entsprechend glatt sein. Weitere Forderungen knüpfen wir nicht an f, wir stellen uns aber vor, daß die Auswertung von f aufwendig ist, z.B. weil die Dimension n (z.B. mit $n \gg 1000$) sehr groß ist oder f komplizierte Objekte enthält, sonst dürfen wir keinen Vorteil durch die Parallelisierung erwarten. Etwas konkreter könnte das ODE-System (2.1) z.B. durch eine Ortsdiskretisierung von Reaktions-Diffusions-Gleichungen entstanden sein. Bei entsprechend feiner Ortsgitterwahl ist n sehr groß und das System steif.
Der Entwurf und die Analyse von numerischen Methoden zur Lösung solcher Anfangswertprobleme ist ein Kerngebiet der numerischen Mathematik mit einer langen Tradition, einer weit ausgebauten Theorie und umfangreichen Anwendungen. Durch die Entwicklung der Rechentechnik motiviert, betrachtet man in jüngerer Zeit auch verstärkt parallele Verfahren. Einen guten Überblick über unterschiedliche parallele Ansätze findet man in der Monographie von Burrage [11]. Wir geben hier eine kurze Einführung mit einigen neueren Ergebnissen.
Parallele Integrationsverfahren zur Lösung des Systems (2.1) unterteilt man (nach Gear, z.B. in [11]) oft in solche, die *parallel bezüglich des Systems* und solche, die *parallel bezüglich der Methode* sind. Vom ersten Fall spricht man, wenn die Berechnung der rechten Seite f auf mehrere Prozessoren aufgeteilt wird. Dieser Ansatz hat viele Vorteile:

- bewährte sequentielle Integrationsmethoden können zugrundegelegt werden,
- durch das Verteilen der Daten können sehr große Probleme mit vielen, z.B. 10…200, Prozessoren (überhaupt) gelöst werden,
- durch das (weitgehende) Vermeiden globaler Operationen und die Nutzung schneller lokaler Speicher (Caches) erhält man oft hohe Speedups.

Typischerweise werden dann bei impliziten Verfahren die bei der Integration auftretenden linearen Gleichungssysteme nur noch iterativ, etwa durch Krylovunterraumtechniken, gelöst. Damit müssen neben den f-Aufrufen noch Jacobimatrix-Vektor-Multiplikationen, eventuell mit geeigneten Vorkonditionierern, verteilt berechnet werden. Hauptnachteil bei einer Parallelisierung bezüglich des Systems ist der hohe Programmieraufwand durch den Nutzer. Für jedes Differentialgleichungssystem ist neu zu überlegen, wie Daten und zugehörige Berechnungen auf die Prozessoren verteilt werden und wie Mechanismen zur Kommunikation und Synchronisation implementiert werden. Setzt man sogenannte Message-Passing Bibliotheken (wie z.B. das *message passing interface*, kurz MPI) ein, kann die Integration auch auf mehrere, durch ein schnelles Netzwerk verbundene, Rechner (sogenannte Cluster) verteilt werden. Mit einfacher PC-Technik können so relativ preiswert Hochleistungsrechner mit mehreren hundert Knoten zusammengesetzt werden [47]. Ein typisches Beispiel für eine Parallelisierung bezüglich des Systems ist der ODE-Löser PVODE [32]. PVODE basiert auf dem bekannten sequentiellen BDF-Mehrschrittcode VODPK. Die Datenparallelisierung erfolgt mittels MPI und die linearen Gleichungssysteme des Newtonverfahrens werden durch GMRES(l) gelöst. PVODE erreicht gute Speedups und erweist sich, auch bei eigenen Messungen mit bis zu 16 Prozessoren,

als äußerst effizient. Häufig findet man auch die Parallelisierung bezüglich des Systems eng verknüpft mit der Ortsdiskretisierung partieller Differentialgleichungen z.B. durch Finite Elemente. Bei adaptiver Gitterverfeinerung ist dann dynamisches *Loadbalancing* notwendig. Die Entwicklung und Analyse solcher Algorithmen gilt als besonders schwierig und zeitaufwendig. Die zugehörige Software wird sehr komplex, wie man beispielsweise an dem parallelen Mehrgitterlöser `ug` [3] sehen kann.

Beim alternativen Zugang, den wir mit den Zweischritt-W-Methoden in dieser Arbeit verfolgen wollen, *bezüglich der Methode zu parallelisieren*, hat man andere Ziele. Das Integrationsverfahren soll selbst bereits parallel sein, unabhängig von der zu lösenden Differentialgleichung. Der Vorteil ist für einen Nutzer offensichtlich, er muß sich gar nicht um die Parallelisierung kümmern und kann seine Probleme einfach auf einem Parallelrechner beschleunigt ausführen lassen. Ein Nachteil ist die schlechte Skalierbarkeit, die Anzahl der maximal sinnvoll einsetzbaren Prozessoren ist klein. Setzt man Mehrprozessormaschinen ein, wie sie jüngst als sogenannte Workgroup-Server (z.B. Sun Enterprise 450) verstärkt angeboten werden, so hat man

- wenige (z.B. zwei bis zehn) leistungsstarke Prozessoren, die
- auf gemeinsamen, globalen Speicher (*(virtual) global shared memory*) zugreifen können und
- mittels Threadprogrammierung (z.B. durch Schleifendirektiven in OpenMP oder durch entsprechende Laufzeitbibliothek (Posix-Threads, *Pthreads*)) einfach und wirkungsvoll synchronisiert werden können.

Gravierend ist der Umstand, daß sich klassische Integrationsverfahren kaum bezüglich der Methode parallelisieren lassen. Für explizte Runge-Kutta-Verfahren beispielsweise bestimmt die Anzahl der sequentiellen Stufen bereits die maximale Ordnung [34]. Gute explizite Runge-Kutta-Verfahren sind daher stets sequentiell. Bei impliziten können die Stufen i.a. zwar nicht unabhängig voneinander berechnet werden, aber innerhalb eines Newtonschrittes kann auf der Ebene der linearen Algebra parallelisiert werden. In der Literatur finden sich ganz unterschiedliche Ideen hierfür, die dann oft exemplarisch auf die Radau-IIA-Verfahren angewendet werden. In der einfachsten Variante parallelisiert man einen sequentiellen Code, ohne den numerischen Algorithmus zu ändern. Die parallele Variante bleibt (bis auf Rundungsfehler) äquivalent und muß nicht neu analysiert werden. Für das dreistufige RADAU5 [30] hat Pulch [43] drei Unterroutinen parallelisiert: die numerische Berechnung der Jacobimatrix, die LU-Zerlegung der Newton-Matrix (durch einen entsprechenden ScaLAPACK-Aufruf) und die Funktionsauswertungen in den Stufen pro Newtoniteration. Pulch diskutiert ein akademisches Beispiel mit dichter Jacobimatrix, das so groß gewählt wird, daß 90 Prozent der Rechenzeit für die Generierung der Jacobimatrix benötigt werden. Letzteres ist hervorragend parallelisierbar und ein Speedup von 3.6 mit vier Prozessoren wird möglich. Für kleinere Beispiele oder Bandjacobimatrizen wäre die LU-Zerlegung ein Flaschenhals und die Parallelisierung uneffektiv.
Weitergehende Parallelisierungsmöglichkeiten, insbesondere für Radau-IIA-Verfahren, wurden in den zurückliegenden Jahren am CWI-Institut in Amsterdam vorgeschlagen, siehe z.B. [51]. Leitidee ist eine parallelisierbare Approximation an die Newton-Matrix. Hier geht die Äquivalenz zur sequentiellen Grundvariante verloren, insbesondere verschlechtert sich die Konvergenzgeschwindigkeit des nichtlinearen Lösers. Ein guter Kompromiß, der schließlich auch im parallelen Radau-7-Code `PSIDE` implementiert wurde, ist PILSRK (*parallel iterative linear system solver for RK methods*) [50]. Durch eine Block-Tridiagonal-Approximation sind beim vierstufigen `PSIDE` innerhalb eines Schrittes vier verschiedene LU-Zerlegungen zu berechnen, was hohe Speedups sichert. Der Vergleich anhand der Beispiele im CWI-Testset [23] mit dem sequentiellen RADAU-Code fällt etwas uneinheitlich aus, weil die Konvergenzverschlechterung im Newtonverfahren sehr vom Problem abhängt. In eigenen Rechnungen war das parallele `PSIDE` bei Rechnung mit vier Prozessoren

nicht effizienter als RADAU, siehe [38], weshalb wir in dieser Arbeit als Referenzverfahren den sequentiellen Code RADAU von Hairer und Wanner [31] verwenden.
Eine Klasse von impliziten Runge-Kutta-Verfahren, die sich gut für eine Parallelisierung eignen, sind die sogenannten MIRK-Verfahren (*multi implicit Runge-Kutta*). Sie haben eine Stabilitätsfunktion mit paarweise verschiedenen reellen Polstellen. Dadurch können im Newtonverfahren (in einer geeigneten Formulierung nach W-Transformation) LU-Zerlegungen und Rücksubstitutionen parallel berechnet werden [5]. Bei sequentieller Rechnung haben MIRK-Verfahren einen beinahe s-fach höheren Aufwand als SIRK-Verfahren (mit s-fachem reellen Pol) pro Schritt, so daß sie nur dann konkurrenzfähig sein könnten, wenn größere Schrittweiten möglich wären. Dafür ist jedoch kein theoretischer Grund erkennbar, da sowohl für SIRK als auch für MIRK die Ordnungsbarriere $p \leq s + 1$ für s-stufige Verfahren gilt. Eine Diskussion zu Bendtsens MIRK-Code ParSODES findet sich in Abschnitt 5.1.
Um parallele Verfahren mit entkoppelten Stufen erhalten zu können, verlassen wir die klassischen Runge-Kutta-Verfahren und betrachten allgemeinere Ansätze, die sich im Rahmen der Theorie der verallgemeinerten linearen Methoden (*general linear methods, GLM*) beschreiben lassen. Die GLM-Klasse beschreibt „mehrstufige Mehrschrittverfahren“. Als Extremfälle sind die Runge-Kutta-Verfahren (ein Schritt, mehrere Stufen) und die Mehrschrittverfahren (eine Stufe, mehrere Schritte) enthalten. Eine weit ausgebaute Theorie wurde wesentlich von Butcher [13] entwickelt. Obwohl GLM-Verfahren seit vielen Jahren bekannt sind, hat man bisher kaum effiziente Verfahren entwickeln können, die nicht einem der beiden obigen Extreme (Runge-Kutta-Verfahren oder Mehrschrittverfahren) angehören. Die vielen zusätzlichen Freiheitsgrade verkomplizieren die Analyse und Implementierung stark, und sich widersprechende Kriterien für gute Verfahren (Ordnung, Fehlerkonstanten, Stabilität, usw.) sind schwer gegeneinander abzuwägen. Im Zusammenhang mit der Frage nach parallel berechenbaren Stufen sind bestimmte Klassen von GLM-Verfahren in letzter Zeit intensiver untersucht worden. Burrage und Suhartanto [12] betrachten Mehrschritt-RADAU-Methoden und lösen das nichtlineare Gleichungssystem auf die gleiche Weise wie in PSIDE mit PILSRK. Die Stufen sind, wie bei PSIDE, nicht entkoppelt und der Mehrschrittcharakter dient nicht der Parallelisierung, sondern der Ordnungserhöhung. Enekel und Jackson [25] setzen bei ihren DIMSEMs (*diagnonally implicit single-eigenvalue methods*) parallele Stufen ein. Die von ihnen entwickelten Verfahren erweisen sich aber nicht als konkurrenzfähig. Eng mit den DIMSEMs und auch mit den PTSW-Verfahren dieser Arbeit verwandt sind DIMSIMs (*diagonally implicit multistage integration methods*) von Butcher u.a. [14, 15]. In den numerischen Tests in der Arbeit von Butcher [15] sind die dort konstruierten DIMSIMs allerdings nicht konkurrenzfähig zu Standardintegratoren. Dem Zusammenhang zwischen PTSW-Verfahren und DIMSIMs ist Abschnitt 2.5 gewidmet.
Bei expliziten Verfahren sind aus Sicht der Parallelisierbarkeit die von Cong [18] vorgeschlagenen Zweischritt-Runge-Kutta-Verfahren (*explicit pseudo two-step Runge-Kutta methods, EPTRK methods*) ideal, die auch GLMs sind. Pro Stufe ist nur eine sequentielle Funktionsauswertung nötig. Wir haben aus dieser Klasse Verfahren mit $s = 5$ und $s = 8$ Stufen mit Schrittweitensteuerung auf einem shared-memory Rechner implementiert und getestet [20] und dabei eine hervorragende Eignung für nichtsteife Probleme und hohe Genauigkeiten festgestellt. Die geringe Stabilität der EPTRK-Verfahren läßt sich auf Kosten der Genauigkeit etwas verbessern, [42]. Im Abschnitt 3.4 diskutieren wir die Herleitung von Koeffizientensätzen für EPTRK-Verfahren hoher Ordnung mit kleinen Stufenfehlern.
Erwähnenswert ist in diesem kurzen Überblick noch eine dritte Art der Parallelisierung. Werden mehrere Approximationen an die Lösung $y(t)$ zu unterschiedlichen Zeitpunkten gleichzeitig berechnet, so spricht man auch von einer *Parallelisierung bezüglich der Zeit*. Dieser Begriff ist allerdings kaum scharf von einer Parallelisierung bezüglich der Methode zu trennen. Man könnte die Block-Prädiktor-Nyström-Verfahren (BPIRKN) [22] oder die parallelen Randwertproblem-Methoden von Brugnano u.a. [10] als parallel bezüg-

lich der Zeit bezeichnen.

2.2 Definition von parallelen Zweischritt-W-Methoden

Eine bewährte Klasse von Integrationsverfahren für steife Differentialgleichungssysteme sind linear-implizite Runge-Kutta-Verfahren, z.B. [48, 49]. Bei diesen Verfahren wird eine numerische Lösung u_m für das System (2.1) durch ein s-stufiges Einschritt-Schema mit Schrittweite h_m der Gestalt

$$Y_{mi} = u_m + h_m \sum_{j=1}^{i-1} a_{ij} k_{mj}, \tag{2.2a}$$

$$(I - h_m \gamma T_m) k_{mi} = f(t_m + h_m c_i, Y_{mi}) + h_m T_m \sum_{j=1}^{i-1} \gamma_{ij} k_{mj}, \quad i = 1, \ldots, s, \tag{2.2b}$$

$$u_{m+1} = u_m + h_m \sum_{i=1}^{s} b_i k_{mi} \tag{2.2c}$$

berechnet. Die Parameter $a_{ij}, \gamma_{ij}, \gamma$, die Gewichte b_i und die Knoten c_i sind reell und konstant. Die Matrix $T_m \in \mathbb{R}^{n \times n}$ in den linearen Gleichungssystemen zur Berechnung der Stufenwerte k_{mi} stellt i.a. eine Approximation an die Jacobimatrix dar und bestimmt wesentlich die Stabilitätseigenschaften des Integrationsverfahrens. Ist im Schema (2.2) eine beliebige Matrix T_m zugelassen, so sprechen wir von einer W-Methode. Bekanntlich läßt sich das Schema (2.2) auch als diagonal-implizites Runge-Kutta-Verfahren mit einem Newtonschritt mit Matrix T_m als Approximation an die Jacobimatrix $f_y(t_m, u_m)$ interpretieren. Eine Parallelisierung ist i.a. nicht möglich, die Stufen hängen voneinander ab und müssen nacheinander berechnet werden. Wir können die Stufen jedoch dann unabhängig berechnen, wenn wir eine Zweischritt-Rekursion verwenden und auf Werte $k_{m-1,j}$ des zurückliegenden Schrittes zurückgreifen. Orientierung geben uns dabei die expliziten Zweischritt-Runge-Kutta-Verfahren (EPTRK) mit externen Stufen der Form

$$Y_{mi} = u_m + h_m \sum_{j=1}^{s} a_{ij} k_{m-1,j} \tag{2.3a}$$

$$k_{mi} = f(t_m + h_m c_i, Y_{mi}) \tag{2.3b}$$

$$u_{m+1} = u_m + h_m \sum_{i=1}^{s} b_i k_{mi}. \tag{2.3c}$$

Man beachte, daß wir bei der Berechnung der Stufen in (2.3a) s statt $i-1$ Summanden verwenden können, ohne den expliziten Charakter zu verlieren. Für den linear-impliziten Fall führt eine Übertragung direkt auf folgendes Schema für parallele Zweischritt-W-Methoden (**PTSW-Methoden**)

$$Y_{mi} = u_m + h_m \sum_{j=1}^{s} a_{ij} k_{m-1,j}, \tag{2.4a}$$

$$(I - h_m \gamma T_m) k_{mi} = f(t_m + h_m c_i, Y_{mi}) + h_m T_m \sum_{j=1}^{s} \gamma_{ij} k_{m-1,j}, \quad i = 1, \ldots, s, \tag{2.4b}$$

$$u_{m+1} = u_m + h_m \sum_{i=1}^{s} (b_i k_{mi} + v_i k_{m-1,i}), \tag{2.4c}$$

wobei wir für die Berechnung von u_{m+1} zusätzlich die alten Stufenwerte $k_{m-1,i}$ verwenden, was bei geeigneter Wahl der Gewichte (v_i) später L-stabile Verfahren ermöglichen wird. Die Parameter γ, b_i und die

paarweise verschiedenen Knoten c_i sind wieder konstant, für a_{ij}, γ_{ij} und v_i werden wir aus Konsistenzgründen eine Abhängigkeit vom Schrittweitenverhältnis $\sigma_m = h_m/h_{m-1}$ zulassen. Die Matrix T_m approximiert die Jacobimatrix $f_y(t_m, u_m)$. Wählen wir $T_m = 0$ und $v_i = 0$, so ergeben sich wieder die expliziten EPTRK-Methoden (2.3). Wir fordern, daß u_m die Lösung $y(t_m)$ und die Stufenwerte k_{mi} deren Ableitungen $y'(t_m + c_i h_m)$ approximieren.
Wir verzichten an dieser Stelle ausdrücklich auf weitere Verallgemeinerungen, obwohl zusätzliche Parameter in das Schema (2.4) leicht eingeführt werden können. Beispielsweise könnten wir stets u_m durch die Summe $\theta u_m + (1-\theta) u_{m-1}$ ersetzen (wie in [2]), oder in den Stufengleichungen $\sum_{i=1}^{s} v_{ij} f(t_m + h_m c_j, Y_{mj})$ einführen. Solche Verallgemeinerungen erschweren die Analyse, sind eventuell redundant (d.h. durch Variablentransformation auf die obige Form reduzierbar, z.B. durch einen Ansatz $\tilde{K}_m = (V^{-1} \otimes I) K_m$), oder schwer zu motivieren (denn warum sollte auf ältere/ungenauere Approximationen zurückgegriffen werden, wenn bessere bereits berechnet wurden?). Wir wollen erst das oben angegebene PTSW-Schema verstehen, bevor wir die anschauliche Orientierung an den bewährten sequentiellen W-Methoden aufgeben. Sind gute Vertreter in der obigen Klasse von Integrationsverfahren gefunden, sollten diese auch eine gute Orientierung für weitere Verallgemeinerungen geben. Interessante Varianten wären z.B. die Wahl unterschiedlicher Nenner γ_i in den einzelnen Stufen oder eine σ_m-abhängige Wahl $\gamma = \gamma(\sigma_m)$, andere werden am Ende der Arbeit im Abschnitt 6 diskutiert.
Indem wir die Parameter a_{ij}, γ_{ij}, v_i, b_i und c_i geeignet zu Matrizen bzw. Vektoren A, Γ, v, b und c zusammenfassen und für die Stufen die Abkürzungen $Y_m = (Y_{m1}^T, \ldots, Y_{ms}^T)^T$, $K_m = (k_{m1}^T, \ldots, k_{ms}^T)^T$ einführen, erhalten wir mit Verwendung des Kroneckerproduktes ‚$\otimes$‘ die kompaktere Schreibweise für die PTSW-Verfahren

$$Y_m = \mathbb{1} \otimes u_m + h_m (A \otimes I) K_{m-1}, \tag{2.5a}$$

$$(I \otimes (I - h_m \gamma T_m)) K_m = F(t_m \mathbb{1} + h_m c, Y_m) + h_m (\Gamma \otimes T_m) K_{m-1}, \tag{2.5b}$$

$$u_{m+1} = u_m + h_m \left[(b^T \otimes I) K_m + (v^T \otimes I) K_{m-1} \right]. \tag{2.5c}$$

Dabei ist $F(t_m + \mathbb{1} h_m c, Y_m) = (f(t_m + c_1 h_m, Y_{m1})^T, \ldots, f(t_m + c_s h_m, Y_{ms})^T)^T$, und $\mathbb{1}$ der Einsenvektor $[1, \ldots, 1]^T$.

2.3 Vereinfachende Bedingungen und Konvergenz

In diesem Abschnitt leiten wir die Ordnungsbedingungen für PTSW-Verfahren (2.4) mit beliebig gewähltem T_m her. Dabei können wir ohne Beschränkung der Allgemeinheit ein autonomes Differentialgleichungssystem zugrundelegen. Die gewonnenen Konvergenzaussagen gelten dann auch für ein nichtautonomes System (weil dieses durch Dimensionsvergrößerung „autonomisiert“ werden kann). Einzige Ausnahme ist Teil (b) in Satz 2.1, der nur für autonome Systeme gilt. Zur Analyse des lokalen Fehlerverhaltens der PTSW-Verfahren, also der Konsistenz, setzen wir analytische Lösungen $y(t)$ und $y'(t)$ für u_m und k_{mi} in das Integrationsschema ein, vgl. [30]. Das Schema ist nicht exakt erfüllt, und wir erhalten Residuen δ_{mi} und η_m mit

$$\begin{aligned}
\widetilde{Y}_{mi} &= y(t_m) + h_m \sum_{j=1}^{s} a_{ij} y'(t_{m-1} + c_j h_{m-1}) \\
(I - h_m \gamma T_m) y'(t_m + c_i h_m) &= f(\widetilde{Y}_{mi}) + h_m T_m \sum_{j=1}^{s} \gamma_{ij} y'(t_{m-1} + c_j h_{m-1}) + \delta_{mi} \qquad (2.6) \\
y(t_{m+1}) &= y(t_m) + h_m \sum_{i=1}^{s} b_i y'(t_m + c_i h_m) + h_m \sum_{i=1}^{s} v_i y'(t_{m-1} + c_i h_{m-1}) + \eta_m.
\end{aligned}$$

Um Einschränkungen an das Schrittweitenverhältnis zu umgehen, setzen wir $h = \max_m h_m$, entwickeln y und y' an der Stelle t_m in Taylorreihen und fordern $\delta_{mi} = h_m \mathcal{O}(h^q)$ und $\eta_m = h_m \mathcal{O}(h^p)$ und erhalten für die Stufen zwei Bedingungen (ohne und mit T_m) der Form

$$\sum_{l=1}^{q} \frac{(c_i h_m)^l}{l!} y^{(l+1)}(t_m) = h_m \sum_{l=1}^{q} \sum_{j=1}^{s} a_{ij} \frac{((c_j - 1)h_{m-1})^{l-1}}{(l-1)!} y^{(l+1)}(t_m) + h_m \mathcal{O}(h^q)$$

$$-\gamma h_m \sum_{l=0}^{q-1} \frac{(c_i h_m)^l}{l!} y^{(l+1)}(t_m) = h_m \sum_{l=0}^{q-1} \sum_{j=1}^{s} \gamma_{ij} \frac{((c_j - 1)h_{m-1})^{l}}{l!} y^{(l+1)}(t_m) + h_m \mathcal{O}(h^q)$$

und für die dritte Gleichung in (2.6)

$$\sum_{l=1}^{p} \frac{h_m^l}{l!} y^{(l)}(t_m) = h_m \sum_{l=1}^{p} \left[\sum_{j=1}^{s} b_j \frac{(c_j h_m)^{l-1}}{(l-1)!} y^{(l)}(t_m) + \sum_{j=1}^{s} v_j \frac{((c_j - 1)h_{m-1})^{l-1}}{(l-1)!} y^{(l)}(t_m) \right] + h_m \mathcal{O}(h^p).$$

Ein Koeffizientenvergleich nach h-Potenzen liefert schließlich die **vereinfachenden Konsistenzbedingungen**

$$\mathrm{C}(q): \qquad \sum_{j=1}^{s} a_{ij}(c_j - 1)^{l-1} = \sigma^{l-1} \frac{c_i^l}{l}, \quad l = 1, ..., q$$

$$\Gamma(q): \qquad \sum_{j=1}^{s} \gamma_{ij}(c_j - 1)^{l-1} = -\gamma \sigma^{l-1} c_i^{l-1}, \quad l = 1, ..., q,$$

$$\mathrm{B}(p): \quad \sum_{j=1}^{s} b_j \sigma^{l-1} c_j^{l-1} + \sum_{j=1}^{s} v_j (c_j - 1)^{l-1} = \frac{\sigma^{l-1}}{l}, \quad l = 1, ..., p.$$

Die Bedingungen $\mathrm{C}(q)$ und $\mathrm{B}(p)$ unterscheiden sich von den bekannten vereinfachenden Bedingungen für implizite Runge-Kutta-Verfahren durch die verschobenen Argumente $(c_j - 1)^l$ anstelle von $(c_j)^l$ und das Auftreten des Schrittweitenverhältnisses $\sigma = \sigma_m = h_m / h_{m-1}$ und der Parameter v_j.
Erfüllt ein PTSW-Verfahren $\mathrm{C}(q)$ und $\Gamma(q)$, so sagen wir, es hat **Stufenordnung** q.
Ist T_m eine Approximation an die Jacobimatrix mit $T_m = f_y(u_m) + \mathcal{O}(h)$, so gilt $\delta_{mi} = h_m \mathcal{O}(h^q)$ bereits mit Stufenordnung $q - 1$, wenn zusätzlich die Bedingung

$$C\Gamma(q): \quad \sum_{j=1}^{s} (a_{ij} + \gamma_{ij})(c_j - 1)^{l-1} = \sigma^{l-1} \left(\frac{c_i^l}{l} - \gamma c_i^{l-1} \right), \quad l = 1, ..., q, \tag{2.7}$$

erfüllt ist. Man sieht, daß $C\Gamma(q)$ sich durch Addition von $C(q)$ und $\Gamma(q)$ ergibt, also eine leichte Abschwächung darstellt.
Ziel der Konvergenzuntersuchungen ist eine Abschätzung der globalen Fehler

$$\varepsilon_m = \|y(t_m) - u_m\|, \quad \nu_{m+1} = \max_{i=1}^{s} \|y'(t_m + c_i h_m) - k_{mi}\|.$$

Wichtig ist im folgenden eine gleichmäßige Beschränktheit der schrittweitenabhängigen Parameter a_{ij}, γ_{ij} und v_i. Bei den Verfahren, die wir konstruieren, erhält man diese feste Schranke wenn Schrittweitenvergrößerungen beschränkt sind, $\sigma_m < \sigma_{\max}, \quad \forall m$. Um die Fehler ε_m und ν_m abzuschätzen, verwenden wir das folgende Lemma.

Lemma 2.1. *Die Folge (ε_m), (ν_m) erfülle*

$$\begin{pmatrix}\varepsilon_{m+1}\\ \nu_{m+1}\end{pmatrix} \le \begin{pmatrix}1+ah_m & bh_m\\ c & bch_m\end{pmatrix}\begin{pmatrix}\varepsilon_m\\ \nu_m\end{pmatrix} + h_m\begin{pmatrix}f_m\\ g_m\end{pmatrix}, \quad m=0,1,...$$

mit nichtnegativen Konstanten a,b,c und $f_m,g_m,h_m \ge 0$ und $\sum_{j=0}^{m} h_j \le (t_e-t_0)$. Setze $L := a+bc$. Dann gilt

$$\max\{\varepsilon_m,\nu_m\} \le Ce^{L(t_e-t_0)}\big(\varepsilon_0 + bh\nu_0 + (t_e-t_0)\max_j(f_j+hg_j)\big), \quad m=0,1,...,$$

wobei $h=\max_m h_m$ ist.

Beweis. In [38]. □

Satz 2.1 (Konvergenzsatz). *Gegeben sei eine PTSW-Methode* (2.4) *für das Differentialgleichungssystem* (2.1)*. Für die Anfangswerte gelte $\varepsilon_0 = \mathcal{O}(h^p)$ und $\nu_0 = \mathcal{O}(h^q)$. Die Koeffizienten des PTSW-Verfahrens seien gleichmäßig beschränkt. Es sei $h=\max_m h_m$.*
a) Sind die vereinfachenden Bedingungen $C(q)$, $\Gamma(q)$ und $B(p)$ erfüllt, so konvergiert das Verfahren mit Ordnung p^* für beliebige Matrizen T_m, d.h.

$$\varepsilon_m = \mathcal{O}(h^{p^*}), \quad \nu_m = \mathcal{O}(h^{p^*}), \quad p^* := \min\{q+1,p\}. \tag{2.8}$$

b) Gilt nur $C(q-1)$, $\Gamma(q-1)$, $C\Gamma(q)$ und $B(p)$ mit $p,q \ge 1$, so konvergiert die Methode ebenfalls mit Ordnung p^*, falls $T_m = f_y(u_m)+\mathcal{O}(h)$ ist und die rechte Seite f der Differentialgleichung (2.1) *nicht explizit von t abhängt.*

Beweis. a) Die vereinfachenden Bedingungen liefern zunächst Abschätzungen für die lokalen Fehler

$$\widetilde{Y}_{mi} = y(t_m+c_ih_m) + h_m\mathcal{O}(h^q), \quad \delta_{mi} = h_m\mathcal{O}(h^q), \quad \eta_m = h_m\mathcal{O}(h^p).$$

Subtrahieren wir (2.4) von (2.6) ergibt sich für die globalen Fehler

$$\begin{aligned}(I-h_m\gamma T_m)(y'(t_m+c_ih_m)-k_{mi}) &= f(y(t_m+c_ih_m)+h_m\mathcal{O}(h^q)) - f(Y_{mi})\\ &\quad + h_mT_m\sum_{j=1}^{s}\gamma_{ij}(y'(t_{m-1}+c_jh_{m-1})-k_{m-1,j}) + h_m\mathcal{O}(h^q)\\ y(t_{m+1})-u_{m+1} &= y(t_m)-u_m + h_m\sum_{i=1}^{s} b_i(y'(t_m+c_ih_m)-k_{mi})\\ &\quad + h_m\sum_{i=1}^{s} v_i(y'(t_{m-1}+c_ih_{m-1})-k_{m-1,i}) + h_m\mathcal{O}(h^p),\end{aligned}$$

mit der Lipschitz-Stetigkeit von f also schließlich die Rekursion

$$\begin{pmatrix}\varepsilon_{m+1}\\ \nu_{m+1}\end{pmatrix} = \begin{pmatrix}1+\mathcal{O}(h_m) & \mathcal{O}(h_m)\\ \mathcal{O}(1) & \mathcal{O}(h_m)\end{pmatrix}\begin{pmatrix}\varepsilon_m\\ \nu_m\end{pmatrix} + h_m\begin{pmatrix}\mathcal{O}(h^{p^*})\\ \mathcal{O}(h^{p^*-1})\end{pmatrix}, \tag{2.9}$$

wobei $p^*=\min(q+1,p)$. Wir können Lemma 2.1 anwenden und erhalten die Behauptung.
b) Mit dem vorangegangenen folgt zunächst Konvergenz mit Ordnung $\min\{p,q\} \ge 1$, also insbesondere ist wegen der Voraussetzung $T_m = f_y(u_m)+\mathcal{O}(h)$ eine Approximation an die Jacobimatrix der exakten Lösung,

$$T_m = f_y(y(t_m)) + \mathcal{O}(h). \tag{2.10}$$

Taylorentwicklung liefert jetzt

$$\widetilde{Y}_{mi} = y(t_m + c_i h_m) - h_m^q C_q + h_m \mathcal{O}(h^q)$$

mit

$$C_q = \frac{1}{(q-1)!}\left(\frac{c_i^q}{q} - \sum_{j=1}^{s} a_{ij}(c_j - 1)^{q-1}\frac{1}{\sigma_m^{q-1}}\right) y^{(q)}(t_m)$$

Für δ_{mi} erhalten wir

$$\begin{aligned}
\delta_{mi} &= (I - h_m\gamma T_m)y'(t_m + c_i h_m) - f\Big(y(t_m + c_i h_m) - h_m^q C_q + h_m\mathcal{O}(h^q)\Big) \\
&\quad - h_m T_m \sum_{j=1}^{s} \gamma_{ij} y'(t_m + (c_j - 1)\frac{h_m}{\sigma_m^{q-1}}) \\
&= (I - h_m\gamma T_m)y'(t_m + c_i h) - y'(t_m + c_i h) + h_m^q f_y(y(t_m))C_q \\
&\quad - h_m T_m \sum_{j=1}^{s}\gamma_{ij}\sum_{l=1}^{q} y^{(l)}(t_m)\frac{(c_j-1)^{l-1}h_m^{l-1}}{\sigma_m^{l-1}(l-1)!} + h_m\mathcal{O}(h^q)
\end{aligned}$$

Und mit $C\Gamma(q)$ schließlich

$$\delta_{mi} = h_m\mathcal{O}(h^q) = h_m\mathcal{O}(h^{p^*-1}),$$

und mit (2.9) liefert Lemma 2.1 die Behauptung. □

Später werden wir die Stufengleichungen iterativ mit Krylovtechniken lösen. Interpretieren wir diese approximativen Verfahren als exakte PTSW-Methode mit gestörten Matrizen $T_{mi} \approx T_m$, so werden diese in den einzelnen Stufen unterschiedlich sein. Auch dann bleibt die Konvergenzordnung erhalten, wie der folgende Satz zeigt.

Satz 2.2. *Gegeben sei ein PTSW-Verfahren mit modifizierten Stufengleichungen*

$$(I - h_m\gamma T_{mi})k_{mi} = f(Y_{mi}) + h_m T_{mi}\sum_{j=1}^{s}\gamma_{ij}k_{m-1,j}, \quad i = 1,\ldots,s, \tag{2.11}$$

mit beliebig gewählten Matrizen T_{mi}. Sei weiter $h = \max_m h_m$ und gelte $C(q)$, $\Gamma(q)$ und B(p), sowie $\varepsilon_0 = \mathcal{O}(h^p), \quad \nu_0 = \mathcal{O}(h^q)$. Dann konvergiert das Verfahren mit Ordnung $p^* = \min\{q+1, p\}$.

Beweis. Wir haben im ersten Teil des Beweises von Satz 2.8 kein Gebrauch davon gemacht, daß T_m in allen Stufen gleich war. Also gilt auch unter den oben abgeschwächten Voraussetzungen die Rekursion (2.9), woraus die Behauptung mit Lemma 2.1 unmittelbar folgt. □

Bemerkung 2.1. Man beachte, daß die Ordnung jedoch verloren geht, wenn man in der Stufengleichung (2.11) links und rechts des Gleichheitszeichens unterschiedliche Matrizen T_{mi} einsetzt. Durch die Umformung (4.1) bei der Implementierung wird die Multiplikation mit T_m auf der rechten Seite eliminiert und dieses Problem damit automatisch vermieden. ◊

Mit Hilfe der Vandermonde-Matrizen gebildet mit den Knoten c_i

$$V_0 := \left(c_i^{j-1}\right)_{i,j} \in \mathbb{R}^{s\times q}, \quad V_1 := \left((c_i - 1)^{j-1}\right)_{i,j} \in \mathbb{R}^{s\times q}, \tag{2.12}$$

können wir die vereinfachenden Bedingungen sehr kompakt in Matrixschreibweise zusammenfassen. Hierfür benötigen wir noch die Diagonalmatrizen

$$D := \operatorname{diag}(1, 2, \ldots, q), \quad S := \operatorname{diag}(1, \sigma, \ldots, \sigma^{q-1}), \quad C := \operatorname{diag}(c_1, \ldots, c_s),$$

sowie die Pascalsche obere Dreiecksmatrix $P_q = (p_{ij})_{i,j=1}^q$, $p_{ij} = \binom{j-1}{i-1}$ wobei $p_{ij} = 0$ für $j < i$. Damit gilt

$$V_1 = V_0 \begin{pmatrix} 1 & -1 & 1 & \ldots \\ & 1 & -2 & \\ & & 1 & \\ & & & \ddots \end{pmatrix} = V_0 P_q^{-1}, \quad V_0 = V_1 P_q.$$

Die vereinfachenden Bedingungen werden zu

$$\begin{array}{rrcl} C(q): & A V_0 P_q^{-1} & = & C V_0 S D^{-1} \\ \Gamma(q): & \Gamma V_0 P_q^{-1} & = & -\gamma V_0 S \\ C\Gamma(r): & (A+\Gamma) V_0 P_r^{-1} & = & (C V_0 D^{-1} - \gamma V_0) S \\ B(p): & b^T V_0 S + v^T V_0 P_p^{-1} & = & \mathbb{1}^T S D^{-1}. \end{array} \tag{2.13}$$

Für Methoden mit hoher Ordnung und Stufenordnung $q = p = s$ sind die Matrizen V_0 und P_s regulär und wir können (2.13) umstellen

$$\begin{aligned} A &= C V_0 S D^{-1} P_s V_0^{-1} \\ \Gamma &= -\gamma V_0 S P_s V_0^{-1} \qquad (2.14) \\ v^T &= [\mathbb{1}^T S D^{-1} - b^T V_0 S] P_s V_0^{-1} \\ \beta &= A + \Gamma = V_0 (F D^{-1} - \gamma I) S P_s V_0^{-1}. \qquad (2.15) \end{aligned}$$

Die ersten drei Gleichungen nutzen wir, um die Verfahrensparameter bei Schrittweitenwechsel neu zu berechnen. Die Matrix β in der letzten Gleichung wird bei den folgenden Stabilitätsuntersuchungen eine wichtige Rolle spielen. Für die Darstellung (2.15) haben wir das folgende Lemma angewendet.

Lemma 2.2. *Sei $V_0 = (c_i^{j-1})_{i,j} \in \mathbb{R}^{s \times s}$. Dann ist*

$$C V_0 = V_0 F, \quad F := \begin{pmatrix} 0 & \cdots & & -\varphi_0 \\ 1 & & & -\varphi_1 \\ & \ddots & & \vdots \\ & & 1 & -\varphi_{s-1} \end{pmatrix}$$

mit der Frobeniusmatrix F zum Knotenpolynom $\varphi(x) = \sum_{j=0}^s \varphi_j x^j := (x - c_1) \cdots (x - c_s)$.

Beweis. Durch die Subdiagonale in F werden die Spalten 2 bis s in V_0 in die Spalten 1 bis $s-1$ verschoben. Die Elemente in der letzten Spalte in CV_0 sind $c_i^s = -\sum_{j=0}^{s-1} \varphi_j c_i^j$.

□

2.4 Lineare Stabilität

Wenden wir ein PTSW-Verfahren (2.4) mit konstanter Schrittweite $h_m = h$ und exakter Jacobimatrix $T_m = \lambda$ auf die skalare Testgleichung $y' = \lambda y$ an, so ergibt sich die Rekursion

$$\begin{pmatrix} hK_m \\ u_{m+1} \end{pmatrix} = M(z) \begin{pmatrix} hK_{m-1} \\ u_m \end{pmatrix}, \quad M(z) := \begin{pmatrix} w\beta & w\mathbb{1} \\ b^T w\beta + v^T & 1 + b^T w\mathbb{1} \end{pmatrix}, \tag{2.16}$$

mit $z = h\lambda, \quad w = z/(1-\gamma z)$. Für $\operatorname{Re}\lambda < 0$ verschwindet die exakte Lösung $y(t) = Ce^{\lambda t}$ für $t \to \infty$. Die numerische Lösung konvergiert gegen null, wenn $\varrho(M(z)) < 1$ ist. Dies führt zur folgenden Übertragung der Begriffe der A- und der L-Stabilität.

Definition 2.1. Gegeben sei ein PTSW-Verfahren (2.4) mit zugehöriger Stabilitätsmatrix $M(z)$ (2.16). Wir nennen die Menge $\mathcal{S} = \{z \in \mathbb{C} : \varrho(M(z)) < 1\}$ Stabilitätsgebiet. Das Verfahren heißt

- A-stabil, wenn $\mathbb{C}^- \subseteq \bar{\mathcal{S}}$,
- A(α)-stabil, wenn $\{z \in \mathbb{C}^- : |\arg(z) - \pi| \le \alpha\} \subseteq \bar{\mathcal{S}}$ für $\alpha \in [0, \pi]$ und
- L-stabil bzw. L(α) stabil, wenn es A- bzw. A(α)-stabil ist und

$$e_{s+1}^T M(z) \to 0 \quad \text{für } \operatorname{Re} z \to -\infty \tag{2.17}$$

  gilt.

Motivation bei der Definition der L-Stabilität ist eine Entkopplung der numerischen Lösung $u_m \to 0$ von u_{m-1} und den Stufen K_{m-1} für $\operatorname{Re} z \to -\infty$. Durch die Struktur der Stabilitätsmatrix erhalten wir aus der Forderung (2.17) eine Bedingung für die Gewichte b_i und v_i.

Lemma 2.3. *Eine PTSW-Methode* (2.4) *erfüllt genau dann* (2.17)*, wenn*

$$b^T\mathbb{1} = \gamma \quad \textit{und} \quad v^T\gamma = b^T\beta \tag{2.18}$$

gilt.

Beweis. Wegen $\lim_{\operatorname{Re} z \to -\infty} w = -1/\gamma$ ist (2.17) äquivalent zu

$$-\frac{1}{\gamma} b^T\beta + v^T = 0, \quad 1 - \frac{1}{\gamma} b^T\mathbb{1} = 0.$$

□

Bei Methoden hoher Ordnung und Stufenordnung legt die L-Stabilitätsbedingung (2.17) die Parameter v_i und b_i vollständig fest.

Satz 2.3. *Sei $B(s)$ und $C\Gamma(s)$ durch eine PTSW-Methode* (2.4) *erfüllt. Dann sind äquivalent*

(i) Bedingung (2.17)

(ii) Es gibt ein i mit $c_i = 1$ und $b^T = \gamma e_i^T, \quad v^T = e_i^T\beta$.

Beweis. Aus B(s) folgt

$$v^T = \left[\mathbb{1}^T S D^{-1} - b^T V_0 S\right] V_1^{-1}. \tag{2.19}$$

Bedingung (2.17) ist äquivalent zu (2.18). Wir ersetzen in (2.18) v^T durch obige Beziehung und erhalten nach kurzem Umformen

$$b^T \mathbb{1} = \gamma \quad \text{und} \quad b^T V_0 F = \gamma \mathbb{1}^T. \tag{2.20}$$

Nun gruppieren wir diese $s+1$ Komponenten $[b^T\mathbb{1}, b^T V_0 F]$ um: aus den ersten s erhalten wir mit der Gestalt von F, $b^T\mathbb{1} = b^T V_0 e_1$ und der Regularität von V_0 den Vektor $b^T = \gamma \mathbb{1}^T V_0^{-1}$. Das setzen wir in die letzte Komponente ein und erhalten

$$b^T V_0 = \gamma \mathbb{1}^T \text{und} \quad \mathbb{1}^T F e_s = 1. \tag{2.21}$$

Damit haben wir die Äquivalenz von (i) und (2.21). Wegen $\mathbb{1}^T F e_s = -\sum_{i=0}^{s-1} \varphi_i = 1 - \varphi(1)$, also $\varphi(1) = \prod_{i=0}^{s}(1 - c_i) = 0$, ist das gerade die Behauptung. □

Ohne Beschränkung der Allgemeinheit können wir bei L-stabilen PTSW-Verfahren, die den Voraussetzungen von Satz 2.3 genügen, annehmen, daß

$$c_s = 1, \quad b^T = \gamma e_s^T, \quad v^T = e_s^T \beta. \tag{2.22}$$

gilt. In Analogie zu der Situation bei impliziten Runge-Kutta-Verfahren nennen wir ein PTSW-Verfahren, das Bedingung (2.22) erfüllt, **steifgenau** (*stiffly accurate*).

2.5 Über den Zusammenhang von PTSW-Verfahren und DIMSIMs

Klassische W-Methoden können aus diagonal-impliziten Runge-Kutta-Verfahren hergeleitet werden, wenn nur eine Newtoniteration ausgeführt wird. In ähnlicher Weise können wir auch die PTSW-Verfahren als linearisierte Version von parallelen, diagonal-impliziten Zweischritt-Runge-Kutta-Verfahren interpretieren, die durch das folgende Schema definiert sind:

$$k_{m,i} = f(t_m + c_i h_m, u_m + h_m \sum_{j=1}^{s} \beta_{ij} k_{m-1,j} + h_m \gamma k_{m,i}), \quad i = 1, \ldots, s \tag{2.23a}$$

$$u_{m+1} = u_m + h_m \sum_{j=1}^{s} \left(b_j k_{m,j} + v_j k_{m-1,j}\right). \tag{2.23b}$$

Die nichtlinearen Stufengleichungssysteme lösen wir mit dem vereinfachten Newtonverfahren mit Jacobimatrix $T_m \approx f_y(t_m, u_m)$. Dann erhalten wir eine Iteration für die Stufen $k_{m,i}^j, \quad j = 1, 2, \ldots$ in der Form

$$\begin{aligned}(I - h_m \gamma T_m)(k_{m,i}^{j+1} - k_{m,i}^j) = \\ f(t_m + c_i h_m, u_m + h \sum_{j=1}^{s} \beta_{ij} k_{m-1,j} + h\gamma k_{m,i}^j) - k_{m,i}^j.\end{aligned} \tag{2.24}$$

Führen wir nur genau einen Newtonschritt aus, so erhalten wir die Klasse der PTSW-Verfahren.

Satz 2.4. *Approximieren wir im impliziten Integrationsschema* (2.23) *die Stufen* k_{mi} *durch die Iterierten* $k_{m,i}^1$ *nach einem Newtonschritt* (2.24) *mit Jacobimatrix* $T_m \approx f_y(t_m, u_m)$*, so ergibt sich mit den Startwerten* $k_{m,i}^0 = -\sum \frac{\gamma_{ij}}{\gamma} k_{m-1,j}$ *das PTSW-Verfahren* (2.4)*.*

Beweis. Setzt man die Startnäherungen $k_{m,i}^0$ in die Newtoniteration (2.24) ein, so folgt die Behauptung unmittelbar. □

Bemerkung 2.2. Für lineare Probleme ist die implizite Methode (2.23) äquivalent zur PTSW-Methode (2.4) mit exakter Jacobimatrix, weil schon nach einem Newtonschritt die Stufeniterierten exakt sind. Folglich gelten sämtliche Stabilitätsüberlegungen für PTSW-Verfahren auch für die obigen Methoden (2.23). $\Diamond$

Bemerkung 2.3. Die Konvergenz des Verfahren (2.23) kann analog zu Satz (2.8) analysiert werden. Für die Behandlung der lokalen Fehler in den Stufen tritt anstelle von C(q) und $\Gamma(q)$ die Bedingung

$$C\Gamma(q): \quad \sum_{j=1}^{s} \beta_{ij}(c_j-1)^{l-1} + \gamma c_i^{l-1} = c_i^l/l, \quad l = 1,\ldots,q,$$

auf, deren Erfülltsein wir wieder als Stufenordnung q bezeichnen. $\Diamond$

Das implizite Verfahren (2.23) können wir als DIMSIM (*diagonally implicit multistage integration method*) interpretieren. Wir orientieren uns an den Bezeichnungen in der Arbeit von Butcher und Chartier [14] und beschränken uns auch auf konstante Schrittweiten $h_m = h$ und autonome Differentialgleichungssysteme. In Kroneckerschreibweise erhalten wir mit

$$Y_m = \mathbb{1} \otimes u_m + h(\beta \otimes I)K_{m-1} + h\gamma K_m$$

und $K_m = F(Y_m)$ eine Darstellung für das Verfahren (2.23)

$$\begin{pmatrix} Y_m \\ hK_m \\ u_{m+1} \end{pmatrix} = \left[\begin{pmatrix} \gamma I & \beta & \mathbb{1} \\ I & 0 & 0 \\ b^T & v^T & 1 \end{pmatrix} \otimes I \right] \begin{pmatrix} hF(Y_m) \\ hK_{m-1} \\ u_m \end{pmatrix},$$

als verallgemeinertes lineares Mehrschrittverfahren mit s inneren Stufen Y_m und $r = s+1$ externen Stufen $y^{[m+1]} = [hK_m^T, u_{m+1}]^T$. Mit $U = [\beta, \mathbb{1}]$, $B = [I, b]^T$ und $V = [e_{s+1}v^T, e_{s+1}]$ ergibt sich schließlich

$$\begin{pmatrix} Y_m \\ y^{[m+1]} \end{pmatrix} = \left[\begin{pmatrix} \gamma I & U \\ B & V \end{pmatrix} \otimes I \right] \begin{pmatrix} hF(Y_m) \\ y^{[m]} \end{pmatrix}. \tag{2.25}$$

Zusammenfassend stellen wir fest, daß die PTSW-Verfahren dieser Arbeit, zumindest für konstante Schrittweiten, als DIMSIMs (2.25) mit einem Newtonschritt mit Matrix T_m interpretiert werden können. Somit sind sie mit Butchers Verfahren ([14, 15]) verwandt. Aber auch bei exakter Lösung des Newtonsystems (2.23) unterscheiden sie sich wesentlich von Butchers Konstruktionen, schon allein deshalb, weil Butcher DIMSIMs mit $r = s$ externen Stufen betrachtet. In [15] wird eine parallele Implementierung von DIMSIMs in Nordsieckform beschrieben. Die numerischen Ergebnisse sind jedoch eher enttäuschend, so daß folgendes Fazit gezogen wird ([15], Seite 14):

> „Thus, there is a long way to go before we can be really sure which of these large classes of methods give the best performance. The development of efficient software for the solution of ODEs requires many years of experience. Our methods are new and the computational experience gained with them so far is very limited. It is hoped that further computational experience and understanding of these methods, along with a deeper understanding of the issues relating to parallel computation, will enable a more efficient implementation".

Im Gegensatz dazu sind die PTSW-Verfahren dieser Arbeit bereits sequentiell konkurrenzfähig zu Standardintegratoren. Dies zeigt deutlich, daß der verwendete PTSW-Ansatz und die Konstruktion der Verfahren inklusive ihrer Implementierung eine günstige Wahl darstellen. Mögliche weitere Verbesserungen diskutieren wir im Abschnitt 6.

3 Konstruktion von Parametersätzen

3.1 Einleitung

In diesem Abschnitt wollen wir konkrete Parametersätze für PTSW-Verfahren (2.4) herleiten. Dabei fordern wir für ein s-stufiges Verfahren stets die vereinfachenden Konsistenzbedingungen $C(s), \Gamma(s), B(s)$ und haben somit stets Stufenordnung s und Ordnung $p \geq s$ nach Satz 2.1. Die Stabilität untersuchen wir für konstante Schrittweiten mit Hilfe des Spektralradius der Stabilitätsmatrix $M(z)$, definiert in Gleichung (2.16). Wir betrachten zwei Fälle, nämlich steifgenaue Verfahren, die Gleichung (2.22) erfüllen und potentiell L-stabil sind, und Verfahren mit $v^T = 0$, die nur A-stabil sein können, dafür aber auch bei variabler Schrittweite mit Ordnung $s + 1$ konvergieren können, vgl. Bemerkung 3.5.

Bei einem s-stufigen steifgenauen Verfahren haben wir s freie Parameter, $c_1, \ldots, c_{s-1}$ und γ, die wir so wählen können, daß die Stabilitätsmatrix nilpotent ist, $\varrho(M(\infty)) = 0$. Dadurch werden sämtliche Freiheitsgrade festgelegt. Der Konstruktion und Analyse der resultierenden Verfahrensklasse mit nilpotenter Matrix $M(\infty)$ ist Abschnitt 3.2 gewidmet. Unter den bis $s = 12$ berechneten Verfahren finden wir für jede Stufenzahl mindestens ein L-stabiles Verfahren.

Ersetzen wir die strenge Nilpotenz-Forderung durch $\varrho(M(\infty)) < 1$, so können wir die Freiheitsgrade zur Optimierung der Verfahren einsetzen, was einigen Aufwand verlangt. Für eine Suche nach „guten Verfahren“ im Abschnitt 3.3 benötigen wir geeignete Konstruktionskriterien und ein leistungsfähiges Suchwerkzeug. Wir untersuchen neben dem Spektralradius $\varrho(M(\infty))$ die Hauptfehlerkonstanten C_{s+1} und C_{s+2} für $y' = \lambda y$ und den A(α)- bzw. L(α)-Winkel numerisch mit Hilfe des in Abschnitt 3.3.2 beschriebenen Fortran-Programms. Als Ergebnis erhalten wir zwei-, drei- und vierstufige A- bzw. L-stabile Verfahren.

Im letzten Abschnitt dieses Kapitel konstruieren wir explizite Verfahren hoher Ordnung mit minimalem Stufenfehler. Dieser Teil zeigt, daß die Techniken und der Formalismus, den wir für die PTSW-Verfahren entwickelt haben, auch auf verwandte Verfahrensklassen erfolgreich angewendet werden können.

Mit den drei Überlegungen in den folgenden Bemerkungen legen wir eine Basis für die Konstruktionen in diesem Kapitel.

Bemerkung 3.1 (Permutationsinvarianz der Verfahren). Durch die identische Stufenordnung s sind die Stufen der PTSW-Verfahren ‘gleichberechtigt’. Folglich dürfen wir erwarten, daß ein PTSW-Verfahren invariant gegenüber Permutation der Knoten c_i ist. Demnach müssen in den entscheidenden Konstruktionsgrößen (wie z.B. in dem charakteristischen Polynom $\det(\lambda I - M(z))$ und damit in den Hauptfehlerkonstanten, dem α-Winkel usw.) die Knoten c_i in symmetrischen Formen auftreten. Wir können daher anstelle der Knoten auch die elementarsymmetrischen Polynome φ_i zur Parametrisierung verwenden, was die (insbesondere Maple-gestützten) Konstruktionen teilweise wesentlich erleichtert. Die mathematische Grundlage bildet Lemma 2.2. ◊

Bemerkung 3.2 (Die Eigenwertentwicklung $\lambda(z)$ für $M(z)$). In $z = 0$ hat die Stabilitätsmatrix $M(z)$, Gl. (2.16), einen Eigenwert $\lambda = 1$ und einen s-fachen Null-Eigenwert. Damit ist $\lambda(z)$ in der Umgebung von $\lambda = 1, z = 0$ eine analytische Funktion. Eine direkte Herleitung der Entwicklung findet man im Report [37], eine computer-algebraische mit Hilfe impliziten Differenzierens im Anhang dieser Arbeit. Bis z^p stimmt die Entwicklung mit der Exponentialfunktion e^z überein, die Konstanten vor den z^{p+1} und z^{p+2} Termen legen die Hauptfehlerkonstanten fest. Wir definieren C_{p+1} und C_{p+2} mit $e^z - \lambda(z) = C_{p+1}z^{p+1} + C_{p+2}z^{p+2} + \mathcal{O}(z^{p+3})$.
Die Eigenwertentwicklung sichert die Nullstabilität der PTSW-Verfahren und liefert ein notwendiges Kriterium für die A-Stabilität, siehe [37]. ◊

Bemerkung 3.3 (Charakterisierung des Winkels α der A(α)-Stabilität). Nach Definition ist α der größte Winkel $\in [0,\pi]$ für den der Sektor $\{z \in \mathbb{C} : |\arg(z) - \pi| \leq \alpha\}$ im abgeschlossenen Stabilitätsgebiet $\bar{S}$, $S = \{z \in \mathbb{C} : \varrho(M(z)) < 1\}$ enthalten ist. Äquivalent können wir auch den Rand des Stabilitätsgebiets ∂S entlanglaufen und den Winkel berechnen durch $\alpha = \inf_{z \in \partial S} \{|\arg(z) - \pi|\}$. Mit der Darstellung

$$\partial S = \{z : \varrho(M(z)) = 1\} \subseteq \{z : \text{ein Eigenwert } \lambda \text{ von } M(z) \text{ hat Betrag 1, d.h. } \lambda = e^{i\psi}\} =: R$$

und der Zerlegung $M(z) = M_0 + wM_1$ mit $w = z/(1-\gamma z)$ mit konstanten Matrizen M_0 und M_1 können wir die Randpunktemenge R als Lösung eines verallgemeinerten Eigenwertproblems für w in der Form

$$R = \left\{ z = w/(1+\gamma w) : \quad wM_1\xi = \left[e^{i\psi} - M_0\right]\xi, \quad \text{mit } \psi \in [0,\ldots,2\pi) \right\} \tag{3.1}$$

charakterisieren und dadurch mit geeigneter Wahl von diskreten Punkten $\psi_j = 2\pi j/N_\psi, \quad j = 1,..,N_\psi$ den A(α)-Winkel approximativ berechnen. $\Diamond$

3.2 Steifgenaue Verfahren mit nilpotenter Stabilitätsmatrix $M(\infty)$

Wir starten mit $s = 1$. Fordern wir die Steifgenauigkeit und die vereinfachenden Konsistenzbedingungen B(1), C(1) und Γ(1), so bleibt in der 2×2-Stabilitätsmatrix nur γ als freier Parameter, es ist

$$M(z) = \begin{pmatrix} w(1-\gamma) & w \\ (\gamma w+1)(1-\gamma) & \gamma w+1 \end{pmatrix}, \quad \text{also} \quad M(\infty) = \begin{pmatrix} -\frac{1-\gamma}{\gamma} & -\frac{1}{\gamma} \\ 0 & 0 \end{pmatrix}, \quad w = \frac{z}{1-\gamma z}.$$

Nur für $\gamma = 1$ ist $\varrho(M(\infty)) = 0$. Der Nicht-Null-Eigenwert für $\gamma = 1$ von $M(z)$ ist gerade die Stabilitätsfunktion $1/(1-z)$ des impliziten Euler-Verfahrens, das Verfahren ist also L-stabil.
Die Existenz von L(α)-stabilen PTSW-Verfahren für $s \geq 2$ ist *a priori* nicht klar. Wir wollen hier zeigen, daß auch bei der relativ restriktiven Forderung nach Nilpotenz der Stabilitätsmatrix für $z \to \infty$, also $\varrho(M(\infty)) = 0$, für praktisch interessante Stufenzahlen solche Verfahren existieren. In den numerischen Tests werden wir sehen, daß diese Verfahren für $s \leq 3$, insbesondere in Kombination mit einer Krylovapproximation, eine gute Effizienz vorweisen können. Eine geeignete Konstruktion liefert der folgende Satz.

Satz 3.1. *Ein s-stufiges PTSW-Verfahren* (2.4) *erfülle die vereinfachenden Konsistenzbedingungen B(s), C(s), $\Gamma(s)$ sowie die Steifgenauigkeits-Bedingungen $c_s = 1$, $b^T = \gamma e_s^T$ und $v^T = e_s^T\beta$.*
Dann gibt es ein Polynom $\pi(x)$ vom Grad s und ein Polynom $\varphi^*(x,y)$ vom Grad s in x und Grad $s-1$ in y, so daß folgende Aussagen äquivalent sind:

(i) Die Stabilitätsmatrix $M(\infty)$ des PTSW-Verfahren ist nilpotent, d.h. $\varrho(M(\infty)) = 0$.

(ii) Der Parameter γ ist Nullstelle von $\pi(x)$ und die Knoten c_i sind Nullstellen vom Polynom $\varphi^*(c,\gamma)$.

Für eine vorgegebene Stufenzahl s gibt es maximal s verschiedene PTSW-Verfahren mit $\varrho(M(\infty)) = 0$.

Beweis. Durch die Steifgenauigkeit (2.22) verschwindet in $M(\infty)$ die letzte Zeile und somit ist $\varrho(M(\infty)) = \frac{1}{\gamma}\varrho(\beta)$. Mit der Darstellung (2.15) ist (i) äquivalent zur Nilpotenz von $\tilde{\beta} := (FD^{-1} - \gamma I)P_s$. Die Eigenwerte von $\tilde{\beta}$ sind genau dann null, wenn die Potenzen im charakteristischen Polynom $\det(xI - \tilde{\beta})$ bis auf den x^s-Term verschwinden. Diese insgesamt s polynomialen Bedingungen können wir geeignet zur Konstruktion der gesuchten Polynome $\pi(\gamma)$ und $\varphi^*(x,y)$ verwenden. Zur Illustration der Struktur geben wir die

Matrix $(xI - \tilde{\beta})$ für $s = 5$ an,

$$(xI - \tilde{\beta}) = \begin{bmatrix} \gamma + x & \gamma & \gamma & \gamma & \gamma + 1/5\,\varphi_0 \\ -1 & -1 + \gamma + x & -1 + 2\,\gamma & -1 + 3\,\gamma & -1 + 4\,\gamma + 1/5\,\varphi_1 \\ 0 & -1/2 & -1 + \gamma + x & -3/2 + 3\,\gamma & -2 + 6\,\gamma + 1/5\,\varphi_2 \\ 0 & 0 & -1/3 & -1 + \gamma + x & -2 + 4\,\gamma + 1/5\,\varphi_3 \\ 0 & 0 & 0 & -1/4 & -1 + 1/5\,\varphi_4 + \gamma + x \end{bmatrix}. \tag{3.2}$$

Entscheidend ist die Hessenbergform durch den FD^{-1}-Anteil. Wenn wir nämlich die Matrix in (3.2) nach der letzten Spalte entwickeln, so haben die Matrizen für die Subdeterminanten auch Hessenbergform. Dies erlaubt sukzessives Auflösen nach φ_{s-i} mit $i = 1, \ldots, s-1$. Wir setzen die Koeffizienten vor den x^{s-i}-Potenzen im charakteristischen Polynom $\det(xI - \tilde{\beta})$ null und stellen dann nach φ_{s-i} um. Dafür benutzen wir die Tatsache, daß der Koeffizient vor x^{s-i} nur die Parameter $\varphi_{s-1}, \ldots, \varphi_{s-i}$ enthält und der Vorfaktor vor φ_{s-i} mit $(s-i)!/s!$ nicht verschwindet. Somit erhalten wir für φ_{s-i} Polynome in γ bis Grad $s-1$, für $i = 1, \ldots, s-1$. Das fehlende φ_0 berechnen wir aus der Steifgenauigkeit $\varphi(1) = \sum_{i=0}^{s} \varphi_i = 0$, also mit $\varphi_0 = -\sum_{i=1}^{s} \varphi_i$. Damit sind alle Parameter φ_i durch Polynome in γ mit Maximalgrad $s-1$ gegeben. Ersetzen wir in $\pi(\gamma) := \det(\tilde{\beta})$ und $\varphi^*(x, \gamma) := x^s + \sum_{i=0}^{s-1} x^i \varphi_i$ die Parameter φ_i durch diese Polynome, so erhalten wir die Äquivalenz von (i) und (ii). Die Maximalzahl von s verschiedenen PTSW-Verfahren erhalten wir genau dann, wenn $\pi(\gamma)$ nur reelle, voneinander verschiedene Nullstellen besitzt. □

Mit Satz 3.1 haben wir ein geeignetes Konstruktionsmittel für die PTSW-Verfahren mit nilpotentem $M(\infty)$. Wir bestimmen die Polynome $\pi(\gamma)$ und $\varphi^*(c, \gamma)$ und erhalten mit den Nullstellen die möglichen Werte für γ und die Knoten c_i. Bis $s = 12$ berechnen wir nun diese Verfahren. Die Fehlerkonstante C_{s+1} und die numerisch berechneten L(α)-Winkel (durch Lösen des Eigenwertproblems (3.1) mit $N_\psi = 100$) geben wir in Tabelle 3.1 an. Wenn ein Verfahren für die numerischen Tests im folgenden Kapitel verwendet wird, so geben wir die Bezeichnung aus Tabelle 4.1 in Klammern an. Eine automatische Berechnung mittels Maple wird für das Beispiel $s = 4$ im Anhang der Arbeit im Abschnitt A.4 demonstriert.

s=2 Wir erhalten die Polynome

$$\begin{aligned} \pi(\gamma) &= -\gamma^2 + 3\,\gamma - 3/2 \\ C_3 &= 2\,\gamma^2 - 7/2\,\gamma + 7/6 \\ C_4 &= 6\,\gamma^3 - \frac{25}{2}\,\gamma^2 + 8\,\gamma - \frac{41}{24} \\ \varphi^*(c, \gamma) &= c^2 + (2 - 4\,\gamma)\,c - 3 + 4\,\gamma. \end{aligned}$$

Es gibt zwei reelle Lösungen für γ. Beide liefern L-stabile Verfahren. Die Werte sind

- $\gamma = 0.63397459$, $c_1 = -0.4641016$, $c_2 = 1$, (PTSW2B)
- $\gamma = 2.36602540$, $c_1 = 6.4641016$, $c_2 = 1$,

wobei das zweite Verfahren durch den Wert für c_1 und der großen Fehlerkonstante für eine Implementierung nicht in Frage kommt.

s=3 Die Polynome sind

$$\pi(\gamma) = \gamma^3 - 6\,\gamma^2 + 8\,\gamma - 8/3$$
$$C_4 = -3\,\gamma^3 + 11\,\gamma^2 - \frac{61}{6}\,\gamma + \frac{61}{24}$$
$$C_5 = -12\,\gamma^4 + \frac{97}{2}\,\gamma^3 - \frac{185}{3}\,\gamma^2 + \frac{385}{12}\,\gamma - \frac{709}{120}$$
$$\varphi^*(c,\gamma) = c^3 + (6 - 9\,\gamma)\,c^2 + \left(9 - 30\,\gamma + 18\,\gamma^2\right)c - 18\,\gamma^2 - 16 + 39\,\gamma,$$

und es gibt drei Lösungen:

- $\gamma = 0.51554560$, $c_1 = -2.0253928$, $c_2 = -0.33469671$, $c_3 = 1$, (PTSW3B)
- $\gamma = 1.21013831$, $c_1 = -0.9905341$, $c_2 = 4.88177899$, $c_3 = 1$,
- $\gamma = 4.27431608$, $c_1 = 7.4028936$, $c_2 = 24.06595112$, $c_3 = 1$.

Für die Rechnungen mit Krylovapproximation liefert das Verfahren PTSW3B im Kapitel 5 hervorragende Ergebnisse.

s=4 Mit den Polynomen

$$\pi(\gamma) = -\gamma^4 + 10\,\gamma^3 - 25\,\gamma^2 + \frac{125}{6}\,\gamma - \frac{125}{24}$$
$$C_5 = 4\,\gamma^4 - 25\,\gamma^3 + \frac{265}{6}\,\gamma^2 - \frac{671}{24}\,\gamma + \frac{671}{120}$$
$$C_6 = 20\,\gamma^5 - 133\,\gamma^4 + \frac{1705}{6}\,\gamma^3 - \frac{6379}{24}\,\gamma^2 + \frac{2729}{24}\,\gamma - \frac{6487}{360}$$
$$\varphi^*(c,\gamma) = c^4 + (12 - 16\,\gamma)\,c^3 + \left(48 - 132\,\gamma + 72\,\gamma^2\right)c^2 +$$
$$\left(64 + 336\,\gamma^2 - 96\,\gamma^3 - 288\,\gamma\right)c - 125 - 408\,\gamma^2 + 96\,\gamma^3 + 436\,\gamma$$

finden wir zwei Verfahren, die sich für eine Implementierung eignen:

- $\gamma = 0.45645866$, $c_1 = -3.3252678$, $c_2 = -2.10534506$, $c_3 = -0.26604845$, $c_4 = 1$, (PTSW4B)
- $\gamma = 0.87242087$, $c_1 = -2.7494421$, $c_2 = -0.70716296$, $c_3 = 4.41533918$, $c_4 = 1$, (PTSW4C)
- $\gamma = 1.94428835$, $c_1 = -1.5167359$, $c_2 = 5.2193893002$, $c_3 = 14.4059603$, $c_4 = 1$,
- $\gamma = 6.72683209$, $c_1 = 8.5440019$, $c_2 = 26.771976846$, $c_3 = 59.31333479$, $c_4 = 1$.

Wegen der wachsenden Komplexität geben wir für $s > 4$ nur die möglichen Werte für γ und die Eigenschaften der resultierenden Verfahren in Tabelle 3.1 an. Wir machen folgende Beobachtungen: Es gibt stets s verschiedene positive Werte für γ, also auch s verschiedene Verfahren. Mit wachsendem γ wächst die Fehlerkonstante C_{s+1}, so daß, gerade für größere Stufenzahlen, die meisten Koeffizientensätze keine vernünftigen Verfahren liefern. Für jede Stufenzahl gibt es mindestens ein L-stabiles Verfahren. Wir fassen diese Ergebnisse in folgendem Satz zusammen.

Satz 3.2. *Es gibt L-stabile s-stufige PTSW-Verfahren mit $\varrho(M(\infty)) = 0$ mit $C(s), \Gamma(s), B(s)$ für alle Stufenzahlen s bis mindestens $s = 12$.*

s	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7	
2	0.6339	2.3660						
	0.2483	4.0817						
	90.0	90.0						
3	0.5155	1.2101	4.2743			γ		
	0.1871	1.0308	74.218			$\lvert C_{s+1}\rvert$-Fehlerkonstante		
	84.8	90.0	89.4			L(α)-Winkel		
4	0.4564	0.8724	1.9442	6.7268				
	0.1718	0.5331	8.3924	2396.6				
	68.5	89.9	90.0	88.2				
5	0.4207	0.7143	1.3012	2.8396	9.7239			
	0.1779	0.3759	2.7826	112.80	120863			
	32.0	84.3	90.0	89.9	87.1			
6	0.3966	0.6230	1.0117	1.8056	3.8968	13.265		
	0.2000	0.3168	1.4463	22.978	2253.7	$8.77e6$		
	0	69.5	89.3	90.0	89.8	86.4		
7	0.3792	0.5636	0.8491	1.3521	2.3865	5.1164	17.352	
	0.2388	0.3002	0.9661	8.5635	273.42	62504.	$8.66e8$	
	0	36.2	81.9	90.0	90.0	89.6	85.2	
8	0.3660	0.5217	0.7456	1.1027	1.7368	3.0443	6.4983	...
	0.2986	0.3095	0.7585	4.4490	71.546	4409.9	$2.29e6$	...
	0	0	66.5	87.3	90.0	89.9	89.4	...
9	0.3556	0.4906	0.6740	0.9463	1.3846	2.1659	3.7792	...
	0.3874	0.3399	0.6670	2.8548	28.481	796.75	92347.	...
	0	0	30.1	78.3	89.7	90.0	89.9	...
10	0.3471	0.4665	0.6215	0.8395	1.1666	1.6956	2.6398	...
	0.5178	0.3924	0.6383	2.1211	14.773	240.05	11361.	...
	0	0	0	60.7	84.3	90.0	89.9	...
11	0.3401	0.4473	0.5815	0.7622	1.0194	1.4072	2.0356	...
	0.7097	0.4715	0.6522	1.7559	9.1963	99.716	2564.1	...
	0	0	0	0	73.3	87.6	90.0	...
12	...	0.7037	0.9137	1.2141	1.6682	2.4049	3.7224	...
	...	1.5796	6.5431	51.631	846.13	33738.	$4.33e6$	...
	...	0	51.2	80.2	89.4	90.0	89.9	...

Tabelle 3.1: Eigenschaften nilpotenter steifgenauer PTSW-Verfahren, konstruiert nach Satz 3.1.

Bemerkung 3.4 (Verfahren mit $v^T = 0$ und $\varrho(M(\infty)) = 0$). Untersuchen wir, wann die transformierte Stabilitätsmatrix mit

$$\operatorname{diag}(V_0, 1)^{-1} M(z) \operatorname{diag}(V_0, 1) = \begin{pmatrix} w\tilde{\beta} & we_1 \\ w\mathbb{1}^T D^{-1}\tilde{\beta} & 1+w \end{pmatrix} \tag{3.3}$$

von A(α)-stabilen PTSW-Verfahren mit $v^T = 0$, B(s), C(s) und $\Gamma(s)$ nilpotent ist, können wir analog zu Satz 3.1 vorgehen. Wir betrachten zunächst den Fall $s = 3$. Die Stabilitätsmatrix $M(\infty)$ ist nilpotent, wenn γ Nullstelle von $\pi(\gamma)$ ist und die Knoten c_i die Nullstellen von $\varphi^*(c, \gamma)$ sind, mit

$$\pi(\gamma) = -\gamma^4 + 6\,\gamma^3 - 8\,\gamma^2 + 8/3\,\gamma \tag{3.4}$$

$$\varphi^*(c, \gamma) = x^3 + (9 - 12\,\gamma)\,x^2 + \left(24 - 66\,\gamma + 36\,\gamma^2\right)x + 16 - 72\,\gamma + 84\,\gamma^2 - 24\,\gamma^3. \tag{3.5}$$

Die Lösungen $\gamma > 0$ sind

- $\gamma = 0.515545602$, $c_1 = -2.69693895$, $c_2 = -0.9387639341$, $c_3 = 0.822250116$ (PTSW3B-2),
- $\gamma = 1.210138313$, $c_1 = -1.410567916$, $c_2 = 1.145543795$, $c_3 = 5.786683881$,
- $\gamma = 4.274316085$, $c_1 = 1.972157174$, $c_2 = 10.86854396$, $c_3 = 29.45109188$.

Nicht nur die Werte für γ, sondern sogar das charakteristische Polynom $\det(\lambda I - M(z))$ (und damit α und C_{s+1}) sind mit den oben berechneten steifgenauen Verfahren gleicher Stufenzahl identisch. Damit haben z.B. PTSW3B und PTSW3B-2 das gleiche asymptotische Verhalten für autonome lineare Probleme (trotz der unterschiedlichen Knoten c_i). Bei praktischen Rechnungen (für nichtlineare Beispiele) ist PTSW3B-2 nicht konkurrenzfähig, was den Vorteil der stärkeren Fehlerdämpfung in einem Schritt durch die Steifgenauigkeit von PTSW3B deutlich zeigt.
Bis $s = 12$ haben wir verifiziert, daß die charakteristischen Polynome $\det(\lambda I - M(z))$ der steifgenauen Verfahren mit $M(\infty) = 0$ auch stets durch Verfahren mit $v^T = 0$ erhalten werden können, was vermutlich auch als Reduktionssatz für beliebige Stufenzahl formuliert und (aufwendig) bewiesen werden könnte, ganz ähnlich wie für die expliziten Verfahren in [42]. Wegen der geringen praktischen Bedeutung der Verfahren mit $v^T = 0$ und $\varrho(M(\infty)) = 0$ verzichten wir auf eine Ausführung. ♢

3.3 Optimierte PTSW-Verfahren

3.3.1 Die Bedingung B($s + 1$)

Im vorangegangenen Abschnitt haben wir L(α)-stabile Verfahren mit Ordnung und Stufenordnung $p = q = s$ konstruiert. Die Stabilität haben wir durch die strenge Forderung $\varrho(M(\infty)) = 0$ abgesichert. Es liegt nun nahe, diese Forderung abzuschwächen, um dadurch die zusätzliche Ordnungsbedingung B($s + 1$) erfüllen zu können, denn dann liefert der Konvergenzsatz 2.1 die Ordnung $p = s + 1$.
Wir nutzen dabei die einfache Interpretation von B(r) als Quadraturbedingung, formuliert in folgendem Lemma:

Lemma 3.1. *Gegeben sei ein s-stufiges PTSW-Verfahren* (2.4)*. Das Verfahren erfüllt B(r) genau dann, wenn es als Quadraturformel für Polynome bis zum Grad $r - 1$ exakt ist.*

Beweis. Sei zunächst die zugrundeliegende Quadraturformel

$$\int_0^\sigma f(t)\,dt \approx \sigma \left[\sum_{i=1}^{s} b_i f(\sigma c_i) + \sum_{i=1}^{s} v_i f(c_i - 1) \right] \tag{3.6}$$

für alle Polynome bis zum Grad $r-1$ exakt. Setzen wir die Monome $f(t) = t^l, \quad l = 0, \ldots, r-1$ ein, so erhalten wir gerade Bedingung B(r). Umgekehrt folgt aus B(r) die exakte Quadratur der Monome und mit der Linearität der Quadraturformel somit die Behauptung. □

Für steifgenaue Verfahren ergibt sich eine leicht zu prüfende Bedingung.

Korollar 3.1. *Ein steifgenaues PTSW-Verfahren mit s Stufen erfülle B(s), C(s) und $\Gamma(s)$. Die Bedingung B($s+1$) ist dann äquivalent zu*

$$\int_0^\sigma \varphi(t+1)\,dt = \sigma\gamma\varphi(\sigma+1), \tag{3.7}$$

was sich für $\sigma = 1$ zu

$$\int_0^1 \varphi(t+1)dt = \gamma\varphi(2), \tag{3.8}$$

vereinfacht.

Beweis. Unter der Voraussetzung B(s) erfüllt das Verfahren genau dann B($s+1$), wenn es für ein Polynom vom Grad s exakt ist. Wir wählen $f(t) = \varphi(t+1)$. Dann ist B($s+1$) äquivalent mit

$$\int_0^\sigma \varphi(t+1)\,dt = \sigma\left[\sum_{i=1}^{s} b_i\varphi(\sigma c_i + 1) + \sum_{i=1}^{s} v_i\varphi(c_i)\right],$$

woraus mit der Steifgenauigkeit (2.22) und wegen $\varphi(c_i) = 0$ die Behauptung folgt. □

Leider können wir die Superkonvergenz bei steifgenauen Verfahren nur für konstante Schrittweiten erfüllen, denn es gilt der folgende Satz.

Satz 3.3. *Ein steifgenaues, mehrstufiges PTSW-Verfahren mit B(s), C(s), $\Gamma(s)$ kann B($s+1$) für variable Schrittweiten nicht erfüllen.*

Beweis. Wir differenzieren Gleichung (3.7) nach σ und erhalten

$$\varphi(\sigma+1) = \gamma\varphi(\sigma+1) + \sigma\gamma\dot{\varphi}(\sigma+1).$$

Die einzige Lösung ist $\gamma = 1/(s+1)$ und $\varphi(x) = (x-1)^s$. Also sind alle c_i gleich, und wir haben kein zulässiges PTSW-Verfahren, wenn wir von $s = 1$ mit $\gamma = 1/2$ absehen. □

Obwohl die steifgenauen Verfahren für variable Schrittweiten nur Ordnung s erreichen können, hat sich bei den numerischen Tests herausgestellt, daß die Superkonvergenzbedingung (3.8) für konstante Schrittweiten eine sinnvolle Forderung ist, schließlich bestimmt das Residuum in B($s+1$) die Hauptfehlerkonstante C_{s+1}.

Bemerkung 3.5. Weil sich für Verfahren mit $v^T = 0$ der Schrittweitenfaktor σ in den Quadraturbedingungen B(r) heraushebt, tritt bei diesen Verfahren keine Ordnungsreduktion durch eine Schrittweitensteuerung auf. ◊

3.3.2 Ein Fortran-Programm für die numerische Suche

Für die numerische Suche nach stabilen PTSW-Verfahren mit guten Genauigkeitseigenschaften variieren wir ein oder zwei freie Parameter, z.B. γ und einen Knoten c_i, und zeichnen dann Bilder für $\varrho(M(\infty))$ die Hauptfehlerkonstanten $|C_{s+1}|, |C_{s+2}|$ und den Winkel α der $A(\alpha)$-Stabilität. Wie die in diesen Bildern dargestellten Konturflächen zu interpretieren sind und wie wir daraus interessante Parameter auswählen,

erläutern wir anhand eines Prototypen in Abbildung 3.1. Hier stellen wir nun das für die Erzeugung dieser Bilder entworfene und in den folgenden Abschnitten intensiv verwendete Fortran-Analyse-Programm `ana.f` vor. Wir arbeiten mit der transformierten Stabilitätsmatrix $\tilde{M}(y)$

$$\mathrm{diag}(V_0^{-1}, 1)M(z)\,\mathrm{diag}(V_0, 1) = \tilde{M}(z) = \tilde{M}_0 + w\tilde{M}_1, \quad w = \frac{z}{1-\gamma z}$$

und benötigen zunächst Unterroutinen für

1. die Generierung der Matrizen $\tilde{M}_0, \tilde{M}_1$ und $\tilde{M}(\infty) = \tilde{M}_0 - \frac{1}{\gamma}\tilde{M}_1$, die Fehlerkonstanten $|C_{s+1}|$ und $|C_{s+2}|$ in Abhängigkeit der Parameter $[\gamma, \varphi_0, \ldots, \varphi_{s-1}]$,
2. die gewünschten Parametrisierungen $[\gamma, \varphi_0, \ldots, \varphi_{s-1}] = \Phi(x, y, \texttt{par})$ der wählbaren Freiheitsgrade, z.B. unter Berücksichtigung von B($s+1$),
3. die Berechnung des Spektralradius $\varrho(\tilde{M}(\infty))$ und des $A(\alpha)$-Winkels über die Lösung des verallgemeinerten Eigenwertproblems (3.1), hier in der Form $w\tilde{M}_1\xi = \left[e^{i\psi} - \tilde{M}_0\right]$ mit $\psi = j2\pi/N_\psi$, $j = 1, \ldots, N_\psi$, $N_\psi = 100$.

Für die Eigenwertprobleme verwenden wir die LAPACK-Routinen ZGEEV und ZGEGV. Damit ist der mathematische Teil in `ana.f` vollständig beschrieben, die übrigen Programmzeilen dienen

- der Auswertung der Kommandozeilenoptionen und der Auswahl der zu verwendenden Subroutinen,
- dem Durchlaufen des Parameterraumes $[\gamma, \varphi_0, \ldots, \varphi_{s-1}] = \Phi(x, y, \texttt{par})$ zur Berechnung und Analyse der jeweiligen Stabilitätsmatrizen für die Erstellung eines $x \times y$–Bildes im PGM-Format,
- der Anzeige des Bildes mit der Möglichkeit, zu einzelnen, mit der Maus gewählten Bildpunkten, Statistiken abzurufen.

3.3.3 Zweistufige L(α)-stabile Verfahren

Wenn wir zweistufige L(α)-stabile Verfahren konstruieren wollen, so haben wir zwei freie Parameter, γ und c_1. Wir variieren diese Parameter im Bereich $[0, 2] \times [-3, 3]$ und berechnen den Spektralradius $\varrho(M(\infty))$, die Fehlerkonstanten C_3, C_4 und den α-Winkel und erhalten Abbildung 3.2. Im Punkt $\gamma = 0.63397459, c_1 = -0.4641016$ ist das Verfahren nilpotent (PTSW2B), erkennbar am schwarzen „Fleck“ im Teilbild (a). L(α)-stabile Verfahren gibt es für $\gamma \geq 0.5$. Man sieht, daß die Fehlerkonstanten C_3 und C_4 mit wachsendem γ i.a. zunehmen. In einem großen Parameterbereich sind die Verfahren L-stabil, Teilbild (d). Innerhalb des betrachteten Ausschnittes ist der Spektralradius nur in der Umgebung von PTSW2B für $c_1 \in [-1.2, 0.2]$ und $\gamma \in [0.55, 0.85]$ kleiner als 0.5. Es ist nicht möglich, Parameter c_1 und γ so zu wählen, daß Spektralradius und Fehlerkonstanten gleichzeitig klein sind.
Wir diskutieren nun zweistufige steifgenaue Verfahren, die für konstante Schrittweiten zusätzlich Bedingung B(3) erfüllen. Dann folgt mit (3.8) $c_1 = 1 + (1/3 - \gamma)/(1/2 - \gamma)$, d.h. wir bewegen uns auf dem oberen dunklen Ast in Abbildung 3.2 (b) entlang. Wir erhalten so $\varrho(M(\infty)), C_4$ und α in Abhängigkeit von γ, dargestellt in Abbildung 3.3.
Der Spektralradius wird für $\gamma \approx 1.3$ minimal mit $\varrho(M(\infty)) \approx 0.67$, Abbildung 3.3 (a). Für $\gamma > 0.77$ sind die Verfahren L-stabil, Teilbild (b). Wir wählen $\gamma = 4/5$ und erhalten das Verfahren PTSW2A mit

$$\gamma = 4/5, \quad c_1 = 23/9, \quad |C_4| \approx 0.1798, \quad \varrho = 0.8799 \quad \text{(PTSW2A)}.$$

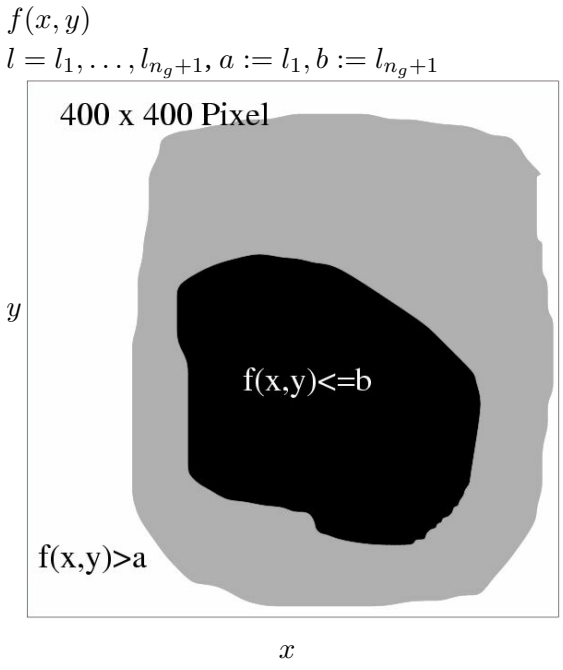

Abbildung 3.1: Konturbild für eine fiktive Funktion $f(x, y)$. Wir verwenden stets (auch für alle folgenden Abbildungen) 400×400 Pixel. Ursprung $(x_{\text{min}}, y_{\text{min}})$ ist die linke untere Ecke. Interessante Bereiche stellen wir dunkel dar: bei Spektralradius und Fehlerkonstanten das Minimum, bei dem α-Winkel das Maximum. Für die Festlegung der Grauwerte benutzen wir die Niveauhöhen $l = l_1, \ldots, l_{n_g+1}$.

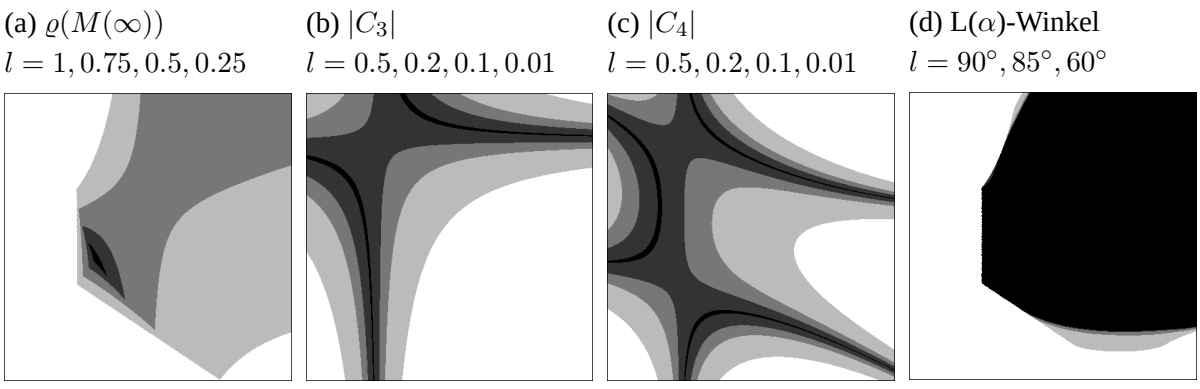

Abbildung 3.2: Ergebnis für zweistufige steifgenaue Verfahren mit $\gamma = x = 0, \ldots, 2$ und $c_1 = y = -3, \ldots, 3$. Dunkle Flächen kennzeichnen einen kleinen Spektralradius, kleine Fehlerkonstanten und große α-Winkel und sind somit für die Auswahl guter Verfahren interessant.

(a) $\varrho(M(\infty))$, $|C_4|$ (b)L(α)-Winkel

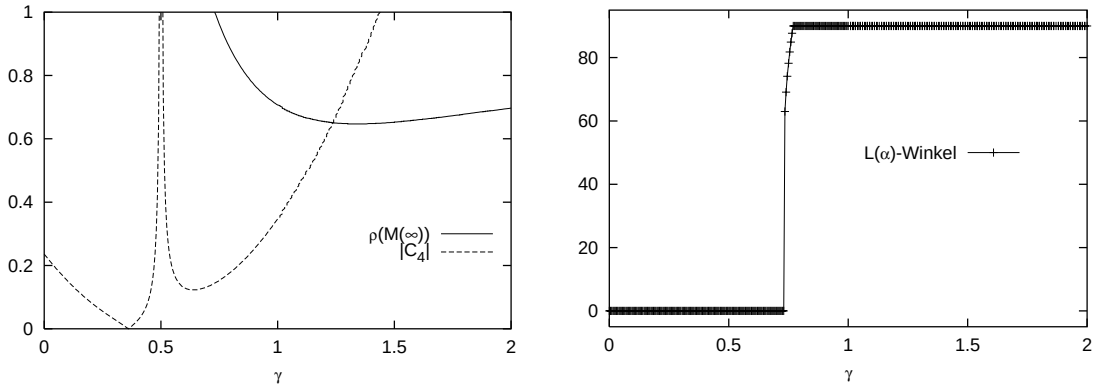

Abbildung 3.3: Ergebnis für zweistufige steifgenaue Verfahren mit B(3) für $\sigma = 1$. Für $\gamma = 0.8$ erhalten wir das L-stabile Verfahren PTSW2A.

Bemerkung 3.6. Wir haben auch die Wahl $\gamma = 1.2$ getestet, weil dann eine spezielle G-Norm-Abschätzung für $\|M(z)\|_G \leq 1, \mathrm{Re}\, z \leq 0$ konstruiert werden kann, [54]. Bei praktischen Rechnungen war das Verfahren mit $\gamma = 1.2$, außer bei einigen DAE-Testproblemen [41], weniger genau und deutlich weniger effizient als PTSW2A. $\Diamond$

3.3.4 Dreistufige L(α)-stabile Verfahren

Bei Berücksichtigung der Steifgenauigkeit bleiben uns drei freie Parameter zur Optimierung: c_1, c_2 und γ. Wir wählen c_1 und γ aus dem Bereich $[-3, 3] \times [0, 2]$ für die Abbildung 3.4. Der dritte Parameter c_2 durchläuft für jeden Bildpunkt das Intervall $[-3, 3]$ (mit Schrittweite $6/400$). Wir reduzieren diese dreidimensionale Information auf das 2D-Bild, indem wir bezüglich c_2 nur die Optimalwerte (minimaler Spektralradius und maximaler α-Winkel) darstellen.
Das Verfahren PTSW3B (mit $\varrho(M(\infty)) = 0$) finden wir in den Punkten $x = \gamma \approx 0.51, y = c_1 \approx -2.02$ bzw. $y = c_1 \approx -0.33$, also im dunklen Streifen im Teilbild (a) bei $\gamma \approx 0.5$. Im betrachteten Ausschnitt gibt es L(α)-stabile Verfahren nur mit $\gamma > 0.45$. Eine gute Dämpfung $\varrho(M(\infty)) < 0.4$ ist mit $\gamma < 0.65$ möglich. Andererseits existieren L-stabile Verfahren nur für $\gamma > 0.73$, erkennbar an der schwarzen Fläche im Teilbild (b), d.h. eine gute Dämpfung ist mit der L-Stabilität nicht vereinbar. Fixieren wir γ, so können wir $\varrho(M(\infty))$ und α in Abhängigkeit von c_1 und c_2 zeichnen, Abbildung 3.5. Wir sehen in der Serie von Bildern der unteren Tafel (b), daß $\alpha = 90°$ nur am Rand, also mit einem Knoten $c_i \approx 3$, möglich ist. Es ist nicht möglich, PTSW3B durch geringe Variation der Parameter zu verbessern: A-Stabilität oder Superkonvergenz sind nur mit $\gamma > 0.73$ möglich. Wählen wir ein kleineres γ, so können wir gegenüber PTSW3B die Fehlerkonstante geringfügig reduzieren (auf Kosten des Spektralradius), was bei numerischen Tests keine Verbesserung darstellte.
Betrachten wir nun dreistufige Verfahren, die B(4) für konstante Schrittweite erfüllen. Dann ergibt sich aus (3.8) die Beziehung

$$c_2 = 1/2\, \frac{17 - 48\gamma + 24\,\gamma c_1 - 10\, c_1}{-12\gamma + 5 - 3\, c_1 + 6\gamma c_1}, \tag{3.9}$$

d.h. uns verbleiben die Parameter c_1, γ für eine numerische Suche. Die Ergebnisse für den Spektralradius, die Hauptfehlerkonstante und den L(α)-Winkel sind in Abbildung 3.6 dargestellt. Wir machen folgende Beobachtungen: Die Nullstellen des Nenners $-12\gamma + 5 - 3\, c_1 + 6\gamma c_1$ in (3.9) liegen als „weißer Graben“ im oberen Bilddrittel, bei $c_1 \approx 2.1$. Die beiden Knoten c_1 und c_2 liegen entweder beide unterhalb oder bei-

(a) $\varrho(M(\infty))$, $l = 1, 0.8, 0.6, 0.4, 0.2$ (b) α, $l = 90°, 89°, 85°, 80°, 70°$

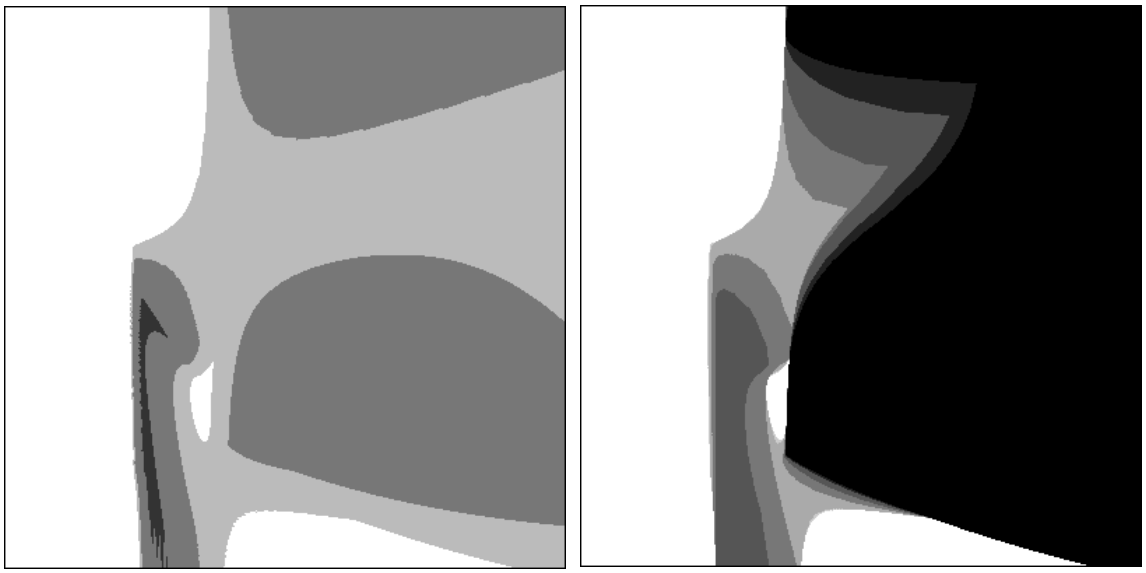

Abbildung 3.4: Ergebnis für dreistufige steifgenaue Verfahren mit $x = \gamma = 0, \ldots, 2$ und $y = c_1 = -3, \ldots, 3$. Dargestellt sind Minimum für Spektralradius und Maximum für α-Winkel unter Variation von $c_2 = (j/400)6 - 3, j = 0, \ldots 400$.

de oberhalb dieser Singularität. L-stabile Verfahren gibt es nur im letzteren Fall, wenn zusätzlich $\gamma > 1.3$ ist. Allerdings sind die Winkel im unteren Teilgebiet teilweise auch deutlich größer als $88°$. Eine starke Dämpfung durch einen kleinen Spektralradius findet man im gewählten Ausschnitt nirgends, stets ist $\varrho(M(\infty)) > 0.83$. Für $\gamma = 0.85$ und $\gamma = 1.45$ betrachten wir Querschnitte in Abbildung 3.7. Für $\gamma = 0.85$ haben wir zwei Möglichkeiten, den Knoten c_1 zu wählen: in $[-2, 0]$ oder in $[2.3, 2.4]$, Teilbild (a). Die erste Wahl erscheint zunächst wesentlich günstiger, liegt sie doch weit entfernt von der Singularität in (3.9). Tatsächlich haben wir gar keine Wahl: durch die Superkonvergenzbedingung (3.9) erhalten wir stets einen Knoten im ersten und einem im zweiten Intervall.
Wir wählen schließlich $\gamma = 0.85$ $c_1 = -1$ und erhalten

$$\gamma = 0.85, \quad c_1 = -1, \quad c_2 = 2.34246575, \quad \alpha = 88.4°, \quad |C_5| = 0.27428 \quad \text{(PTSW3A)}.$$

Für $\gamma = 1.45$ erhalten wir ein L-stabiles Verfahren mit

$$\gamma = 1.45, \quad c_1 = 2.32842724 \quad c_2 = 2.94999281, \quad \alpha = 90°, \quad |C_5| = 0.$$

Das zweite Verfahren haben wir so konstruiert, daß auch die Fehlerkonstante C_5 verschwindet (und alle Parameter allein durch die Wahl von γ festgelegt sind). Trotzdem war es bei praktischen Rechnungen weniger genau als PTSW3A.

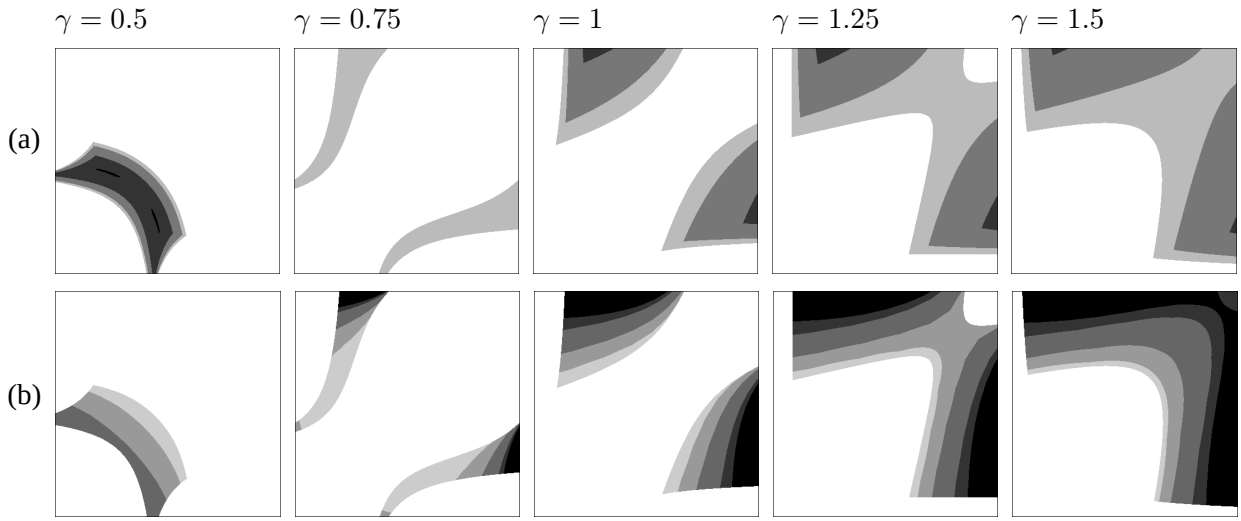

Abbildung 3.5: Spektralradius (a) und α-Winkel (b) für dreistufige steifgenaue Verfahren in Abhängigkeit von $c_1, c_2 \in [-3, 3]^2$ für fest gewähltes γ. Wir verwenden als Niveauhöhen $l = 1, 0.95, 0.8, 0.4$ für $\varrho(M(\infty))$ bzw. $l = 90°, 89°, 85°, 80°, 70°$ für α. Wir erhalten $\varrho(M(\infty)) < 0.5$ nur für $\gamma = 0.5$ und $\alpha = 90°$ ab $\gamma \geq 0.75$.

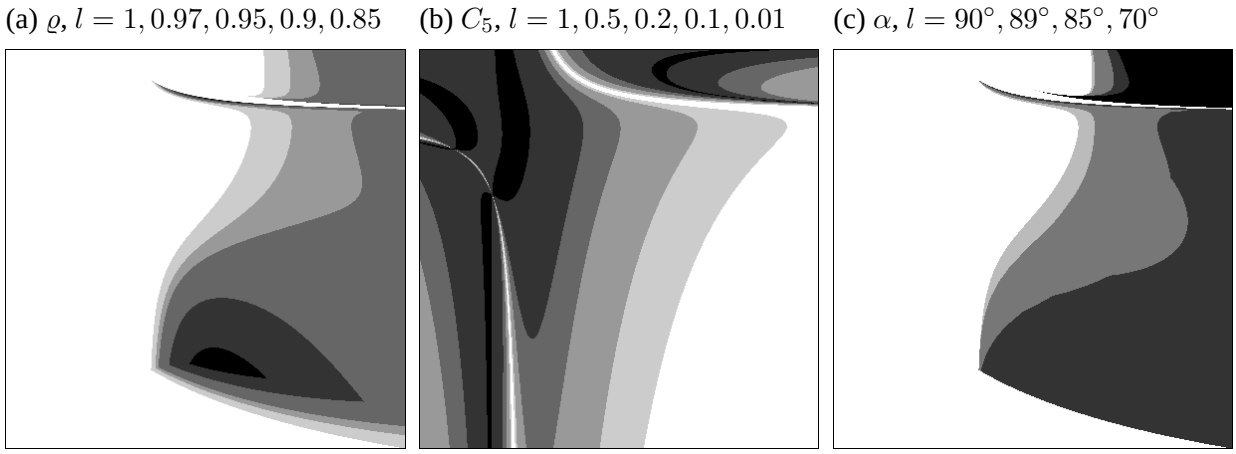

Abbildung 3.6: Ergebnis für dreistufige steifgenaue Verfahren mit B(4) für $\sigma = 1$ in Abhängigkeit von $x = \gamma \in [0, 2]$ und $y = c_1 \in [-3, 3]$.

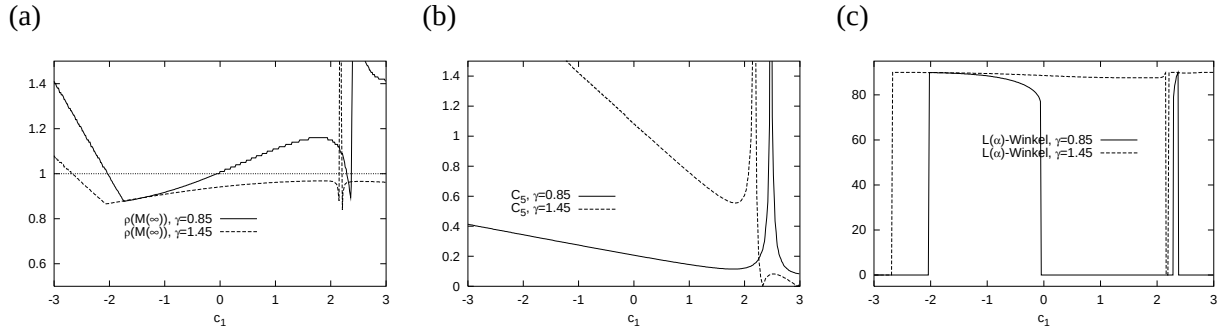

Abbildung 3.7: Spektralradius $\varrho(M(\infty)$, Fehlerkonstante C_5 und skalierter α-Winkel für dreistufige steifgenaue PTSW-Verfahren mit B(3) für $\sigma = 1$ für $\gamma = 0.85$ und $\gamma = 1.45$.

3.3.5 Vierstufige L(α)-stabile Verfahren

Wir fordern B(5) für $\sigma = 1$ und erhalten aus (3.8)

$$c_3 = 1/5\,\frac{-147 + 480\,\gamma - 50\,c_1c_2 + 85\,c_1 - 240\,\gamma c_2 - 240\,\gamma c_1 + 85\,c_2 + 120\,\gamma c_1 c_2}{10\,c_2 - 17 + 48\,\gamma - 6\,c_1c_2 - 24\,\gamma c_2 - 24\,\gamma c_1 + 10\,c_1 + 12\,\gamma c_1 c_2}. \tag{3.10}$$

Es bleiben drei freie Parameter: c_1, c_2 und γ. Wir variieren $x = \gamma \in [0, 2]$ und $y = c_1 \in [-3, 3]$ und zeichnen den minimalen Spektralradius für $c_2 = -3 + (6/2000)j$, $j = 0, \ldots 2000$ in Abbildung 3.8.

(a) $\varrho(M(\infty))$, $l = 1, 0.95, 0.9, 0.85$ (b) $\varrho(M(\infty))$, $l = 1, 0.95, 0.9, 0.85$

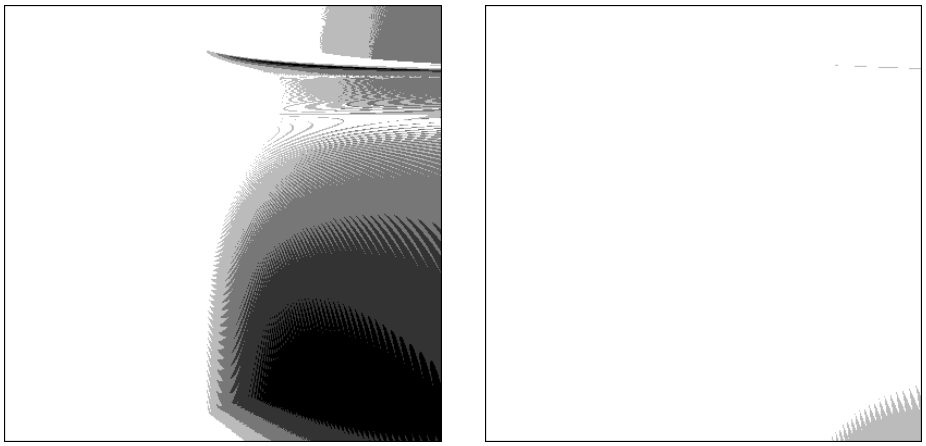

Abbildung 3.8: Ergebnis für steifgenaue vierstufige PTSW-Verfahren mit B(5). Wir wählen $x = \gamma \in [0, 2]$ und $y = c_1 \in [-3, 3]$, variieren $c_2 = -3 + (6/2000) * j$, $j = 1, \ldots, 2000$ und zeichnen das Minimum für $\varrho(M(\infty))$ bezüglich c_2. Im rechten Teilbild berücksichtigen wir bei dieser Minimierung ein Verfahren nur dann, wenn $|c_3| < 4$ gilt.

Es gibt, wenn wir zunächst das linke Bild betrachten, einen großen Parameterbereich, in dem das PTSW-Verfahren stabil ist. Allerdings müssen wir noch prüfen, wie die jeweils zugehörigen Werte für c_3 ausfallen. Und hier finden wir leider nur sehr große Beträge (≈ 10 und mehr). Daher beschränken wir uns bei der Minimierung bezüglich c_2 im rechten Teilbild 3.8 auf solche Werte, für die $|c_3| < 4$ gilt. Der stabile Bereich ist nun erheblich kleiner und für die Forderung $|c_3| < 3.5$ verschwindet er ganz. Die Hoffnung, ein für praktische Rechnungen geeignetes vierstufiges Verfahren mit B(5) (für $\sigma = 1$) zu finden, muß daher aufgegeben werden. Wenn wir zur Kontrolle z.B. $\gamma = 1.5$ fixieren und den Spektralradius in Abhängigkeit von c_1, c_2 zeichnen, so sind die L(α)-stabilen Parameterbereiche kaum erkennbare dünne Linien in unmittelbarer Nachbarschaft zur Menge der Polstellen in (3.9) (nicht dargestellt). Wählen wir ein Verfahren, wie z.B.

$$\gamma = 1.5, \quad c_1 = -3.765, \quad c_2 = 2.191758991, c_3 = 4.05, \varrho(M(\infty)) = 0.98, \alpha = 84.5^\circ.$$

so ist es sehr empfindlich und für praktische Rechnungen nicht geeignet.
Verzichten wir auf die Forderung B(5), so haben wir vier Freiheitsgrade. Mit den Techniken dieses Abschnitts können wir sie nicht mehr erschöpfend diskutieren. Es ist aber möglich, PTSW4C lokal zu verbessern. Die Fehlerkonstante C_5 können wir nicht zum Verschwinden bringen, aber nach einigem Suchen wenigstens gegenüber PTSW4C etwas verringern. Wir verwenden für die numerischen Tests das folgende Verfahren:

$$\gamma = 4/5, \quad c_1 = -1, c_2 = -1/2, c_3 = 4, c_4 = 1, \alpha = 87.9^\circ, |C_5| = 0.22 \text{ (PTSW4A)}.$$

3.3.6 Zweistufige A(α)-stabile Verfahren

Es gibt drei freie Parameter: c_1, c_2 und γ zur Optimierung. Mit der Bedingung B(3) eliminieren wir $c_2 = \frac{1}{3}(-2+3c_1)/(-1+2c_1)$ und erhalten Abbildung 3.9. A(α)-stabile Bereiche gibt es in der Nähe der Singularität $c_1 = 1/2$. Für $\gamma < 0.8$ ergeben sich Knoten $c_1 = 1/2 + \varepsilon$, $c_2 \approx -1$ und für $\gamma > 0.8$ Knoten $c_1 = 1/2 - \varepsilon$ und $c_2 \in [0.6, 3]$. A-stabile Verfahren gibt es nur für $\gamma > 0.8$, Teilbild (c). Im gesamten Ausschnitt ist $\varrho(M(\infty))$ stets größer als 0.6.
Für $\gamma = 0.48$ wählen wir das Verfahren

$$c_1 = 5/9, c_2 = -1, \gamma = 0.48, \varrho = 0.6796, \alpha = 81.1^\circ \quad \text{(PTSW2C)}.$$

Eine zweite Variante, bei der auch die Fehlerkonstante C_4 verschwindet, erhalten wir z.B. für $\gamma = 1.2$ mit der Wahl:

$$c_1, c_2 \text{ als Nullstellen von } \varphi(x) = 203/282 - 99/47x + x^2, \gamma = 1.2, \varrho(M(\infty)) = 0.9438, \alpha = 87.7^\circ.$$

Bei numerischen Tests war PTSW2C deutlich effizienter als das obige Verfahren mit $\gamma = 1.2$.

(a) $\varrho(M(\infty))$
$l = 1, 0.9, 0.75$

(b) $|C_4|$
$l = 1, 0.5, 0.1, 0.01$

(c) α
$l = 90^\circ, 89^\circ, 85^\circ, 80^\circ, 60^\circ$

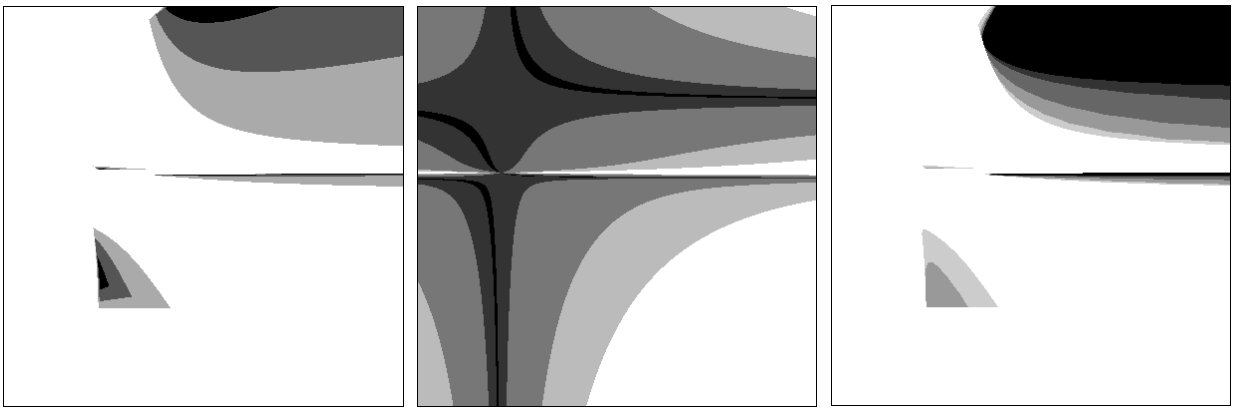

Abbildung 3.9: Ergebnis für zweistufige Verfahren mit $v^T = 0$ und B(3) mit $x = \gamma \in [0, 2]$ und $y = c_1 \in [-2, 2]$. Mit $c_1 \to 1/2$ folgt $c_2 \to \infty$, daher die weiße Linie für $y = c_1 = 1/2$. Wir wählen PTSW2C mit $c_1 = 5/9, \gamma = 0.48$.

3.3.7 Dreistufige A(α)-stabile Verfahren

Mit der Bedingung B(4) eliminieren wir den Knoten c_3 und erhalten

$$c_3 = 1/2\,\frac{3 - 4\,c_1 + 6\,c_1c_2 - 4\,c_2}{6\,c_1c_2 - 3\,c_1 + 2 - 3\,c_2}. \tag{3.11}$$

Es bleiben drei Parameter für die Optimierung: c_1, c_2 und γ. Wir stellen den Spektralradius $\varrho(M(\infty))$ in Abhängigkeit von $x = \gamma$ und $y = c_1$ dar, Abbildung 3.10. Wir zeichnen für jeden Bildpunkt das Minimum bezüglich c_2 mit $c_2 = -3 + (6/400)j, j = 0, \ldots, 400$ (Teilbild (a)), bzw. $c_2 = -3 + (6/5000)j, j = 0, \ldots, 5000$ (Teilbild (b)). Es gibt A(α)-stabile Verfahren mit $\gamma > 0.4$. Im gesamten Ausschnitt ist der Spektralradius stets größer 0.6. Der starke Unterschied zwischen beiden Teilbildern macht deutlich, wie empfindlich die Wahl der Knoten ist. Wir fixieren nun $\gamma = 1$. Dann können wir $\varrho(M(\infty))$ und α als

Funktion von den Knoten c_1 und c_2 berechnen, Abbildung 3.11. Die feine dunkle Linie befindet sich in unmittelbarer Umgebung der Singularität in (3.11). Zwei Knoten müssen wir aus diesem schmalen Band wählen. Der dritte Knoten hat dann zwangsläufig einen großen Betrag. Wir entscheiden uns für die folgende Wahl der freien Parameter:

$$\gamma = 1, c_1 = -0.96, c_2 \approx 0.52766, c_3 = 3.28, \alpha = 87.3^\circ, \varrho(M(\infty)) = 0.93, \text{ (PTSW3C) .}$$

(a) $\varrho(M(\infty))$ (b) $\varrho(M(\infty))$

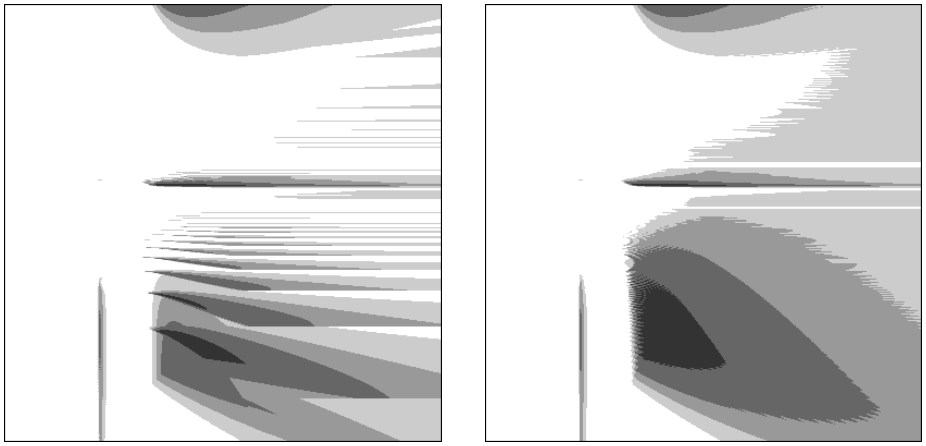

Abbildung 3.10: Ergebnis für dreistufige Verfahren mit $v^T = 0$ und B(4). Wir wählen $x = \gamma \in [0, 2]$ und $y = c_1 \in [-3, 3]$, variieren in Bild (a) $c_2 = -3 + (6/400)j, j = 0, \ldots, 400$ (a) bzw. $c_2 = -3 + (6/5000)j, j = 0, \ldots, 5000$ in Bild (b) und zeichnen das Minimum für den Spektralradius bezüglich c_2. Wir verwenden die Niveauhöhen $l = 1, 0.95, 0.9, 0.8, 0.6$.

3.3.8 Vierstufige A(α)-stabile Verfahren

Wir haben insgesamt fünf Freiheitsgrade. Wenn wir B(5) fordern, bleiben noch vier. Weil das systematische Durchsuchen des Parameterraumes nach stabilen Verfahren zu aufwendig wird, ändern wir die Konstruktionsstrategie: wir versuchen die notwendige Bedingung $\varrho(M(\infty)) < 1$ zu erfüllen, indem wir die Koeffizienten des charakteristischen Polynoms $\det(\lambda I - M(\infty))$ minimieren. Wir verwenden die Darstellung (3.3) und erhalten damit ein lineares Quadratmittelproblem in den elementarsymmetrischen Polynome φ_i. Dieses Problem ist leicht zu lösen und führt uns schließlich auf das folgende Verfahren

$$\gamma = 0.7535833817, c_1 = -3, 3397654896, c_2 - 1.4870830356, c_3 = 0.5362026478, c_4 = 3.5569894699,$$
$$\varrho(M(\infty)) = 0.9262701231, \alpha = 79.2^\circ.$$

Bei numerischen Tests mit diesen Parametern (vergleiche [37]) wurde die Konvergenz mit Ordnung fünf numerisch bestätigt. Das Verfahren erwies sich jedoch als sehr empfindlich und findet daher in dieser Arbeit keine weitere Berücksichtigung.

3.3.9 Zusammenfassung der Optimierungen

Ausgangspunkt für die Optimierungen war die Frage, ob wir die L(α)-stabilen Verfahren mit nilpotenter Stabilitätsmatrix $M(\infty)$ aus Satz 3.1 deutlich verbessern können. Für eine Antwort ist es zweckmäßig folgende drei Fälle zu unterscheiden:

(a) $\varrho(M(\infty))$ (b) α

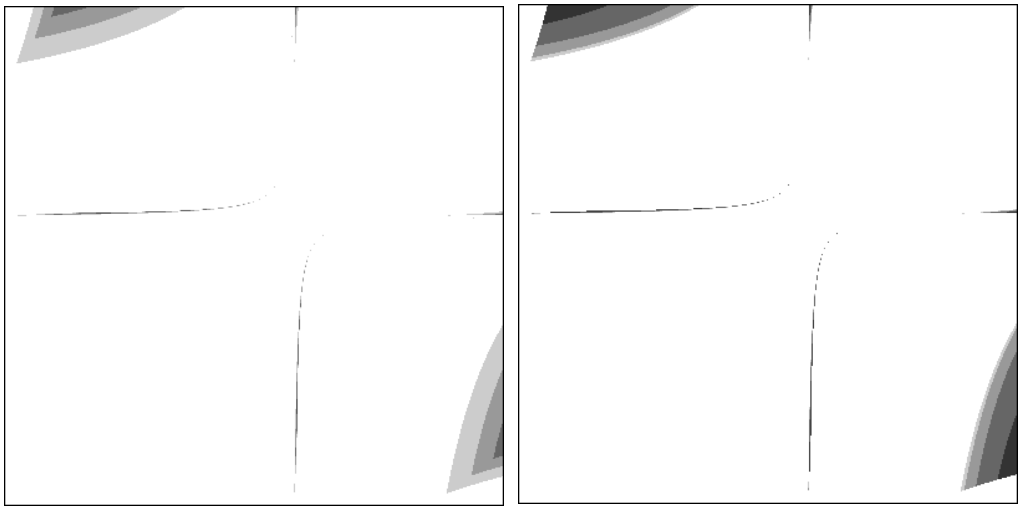

Abbildung 3.11: Ergebnis für dreistufige Verfahren mit $v^T = 0$ und $B(4)$ und $\gamma = 1$. Wir wählen als Niveauhöhen $l = 1, 0.95, 0.9, 0.8, 0.6$ für (a) bzw. $l = 90°, 89°, 85°, 80°, 60°$ für (b).

(a) L(α)-stabile Verfahren mit $\varrho(M(\infty)) < 0.5$,

(b) L(α)-stabile Verfahren mit $0.5 < \varrho(M(\infty)) < 1$,

(c) A(α)-stabile Verfahren mit B$(s+1)$.

Fall (a): Die Parameter möglicher Verfahren mit $\varrho(M(\infty)) < 0.5$ liegen in enger Nachbarschaft zu denen der nilpotenten Verfahren, wobei schon bei geringen Parameteränderungen der Spektralradius erheblich vergrößert wird. Qualitative Verbesserungen, wie z.B. das Erfüllen einer zusätzlichen Ordnungsbedingung B$(s+1)$ sind nicht möglich. Einziger Vorteil wäre eine (geringfügige) Verkleinerung der Hauptfehlerkonstanten. Die nilpotenten Verfahren aus Satz 3.1 sind also lokal kaum zu optimieren. Nur für $s = 4$ entwerfen wir ein entsprechendes Verfahren (PTSW4A).

Fall (b): Wenn wir unsere Forderung an den Spektralradius relaxieren und nur $\varrho(M(\infty)) < 1$ verlangen, können wir die Ordnungsbedingung B$(s+1)$ für konstante Schrittweiten erfüllen. Für $s = 2$ und $s = 3$ finden wir aussichtsreiche PTSW-Verfahren (PTSW2A, PTSW3A mit $\varrho(M(\infty)) \doteq 0.8799$ bzw. $\doteq 0.9178$). L-Stabilität ist nur für $\gamma > 0.73$ möglich.

Fall (c): Hauptvorteil der A(α)-stabilen Verfahren mit $v^T = 0$ ist, daß B$(s+1)$ auch für variable Schrittweiten gilt. Uns gelingt es, für $s = 2$ und $s = 3$ eine geeignete Parameterwahl zu identifizieren (PTSW2C, PTSW3C).

Generell gilt für diese Optimierungen, daß sie mit steigender Stufenzahl wesentlich schwieriger werden. Nur für $s \leq 3$ finden wir robuste PTSW-Verfahren, die B(s), C(s) und $\Gamma(s)$ erfüllen. Die Forderung B$(s+1)$ liefert eine rationale Funktion für einen Knoten c_i. Leider liegen die A(α)- bzw. L(α)-stabilen Verfahren, die B$(s+1)$ erfüllen, oft nur in unmittelbarer Nähe der zugehörigen Singularität. Wir erhalten zwangsläufig betragsgroße Einträge in den Koeffizienten c_i, a_{ij} und γ_{ij}, wodurch die Verfahren sehr empfindlich werden.

Es ist in einigen Fällen möglich, höhere Fehlerkonstanten zum Verschwinden zu bringen, nur sind dafür große Werte für den Parameter γ nötig. Bei praktischen Rechnungen waren jedoch stets Verfahren mit kleinerem γ robuster und effizienter.

3.4 Ausflug: Explizite Verfahren hoher Ordnung

Setzen wir in der Verfahrensvorschrift der PTSW-Verfahren (2.4) die Matrix $T_m = 0$ ein, so erhalten wir explizite Zweischritt-Runge-Kutta-Verfahren (2.3) (EPTRK-Methoden), wie sie von Cong [19] vorgeschlagen wurden. Verwenden wir hierfür die PTSW-Verfahren der vorangehenden Abschnitte dieser Arbeit, so sind die zugeordneten expliziten Verfahren wegen der schlechteren Genauigkeitseigenschaften nicht konkurrenzfähig zu den EPTRK-Methoden in [20] mit $s = 5$ und $s = 8$ Stufen. Die Wahl der Knoten c_i für die EPTRK-Verfahren in [20] wurde in umfangreichen numerischen Tests optimiert. Es blieb offen, warum gerade diese Knoten so günstig waren und ob weitere Verbesserungen möglich sein würden. Dabei waren zwei Richtungen interessant: eine Vergrößerung des Stabilitätsgebiets und eine Erhöhung der Genauigkeit. Unter Verzicht auf Genauigkeit ließen sich die Stabilitätsgebiete dieser Verfahren vergrößern, Details findet man in [36, 39, 42]. Hier möchten wir uns mit der zweiten Frage befassen, nämlich welche Knoten Verfahren mit optimaler Genauigkeit liefern. Motiviert durch numerische Experimente mit PIRK-Verfahren konzentrieren wir uns dabei auf den Stufenfehler X.
Wir betrachten s-stufige EPTRK-Verfahren (2.3) mit B($s + 2$) und C(s). Der Vektor der Hauptstufenfehler X ergibt sich als Residuum in C($s + 1$). Unter Verwendung von Lemma 2.2, den vereinfachenden Konsistenzbedingungen (2.15) mit $\gamma = 0, \beta = A$ und $C^{s-1}\mathbb{1} = V_0 e_s$ erhalten wir die Darstellung

$$X = \left[\frac{C^{s+1}}{s+1} - A(C-I)^s\right]\mathbb{1} = V_0[1/(s+1)F^2 - FD^{-1}(PFP^{-1} - I)]e_s =: V_0 x. \tag{3.12}$$

Durch die Struktur des globalen Fehlers konvergiert das EPTRK-Verfahren für konstante Schrittweiten mit Ordnung $s + 2$, falls die Bedingung

$$b^T X = 0 \tag{3.13}$$

erfüllt ist, [42]. Eine analoge Bedingung wurde in [21] für Zweischritt-Runge-Kutta-Nyströmverfahren formuliert. Ziel der folgenden Überlegungen ist eine Minimierung des Stufenfehlers $\|X\|_2$ für Verfahren der Ordnung $s + 2$. Damit haben wir insgesamt vier Bedingungen an die Knoten c_i zu erfüllen: zwei Quadraturbedingungen B($s + 2$), Gleichung (3.13) und $X^T X \to \min$. Damit können wir für $s = 4$ ein eindeutig bestimmtes Verfahren erhalten. Um dieses bestimmen zu können, formulieren wir unsere vier Bedingungen als Ausdrücke in den symmetrischen Koeffizienten φ_i, $i = 0, \ldots, s - 1$. Sei nun $s = 4$. Für die Quadraturbedingungen erhalten wir

$$\int_0^1 \prod_{i=1}^{s}(x - c_i)\,dx = \varphi_0 + \frac{1}{2}\varphi_1 + \frac{1}{3}\varphi_2 + \frac{1}{4}\varphi_3 + \frac{1}{5} = 0 \tag{3.14}$$

$$\int_0^1 x\prod_{i=1}^{s}(x - c_i)\,dx = \frac{1}{2}\varphi_0 + \frac{1}{3}\varphi_1 + \frac{1}{4}\varphi_2 + \frac{1}{5}\varphi_3 + \frac{1}{6} = 0, \tag{3.15}$$

aus Bedingung (3.13) wird mit (3.12) und B(s)

$$0 = b^T X = \mathbb{1}^T D^{-1}[1/(s+1)F^2 - FD^{-1}(PFP^{-1} - I)]e_s. \tag{3.16}$$

Für $s = 4$ erhalten wir daraus schließlich

$$-1/20\,\varphi_0\varphi_3 - 3/5\,\varphi_0 - 1/40\,\varphi_1\varphi_3 + 5/3 + 1/10\,\varphi_1 + \frac{8}{15}\,\varphi_2 + \varphi_3 - \frac{1}{60}\,\varphi_3\varphi_2 - \frac{1}{80}\,{\varphi_3}^2 = 0. \tag{3.17}$$

Um $X^T X$ in φ-Parametern zu schreiben, benötigen wir wegen

$$\|X\|_2^2 = X^T X = x^T V_0^T V_0 x$$

eine Darstellung von $V_0^T V_0$, die das folgende Lemma liefert.

Lemma 3.2. *Gegeben sei die Vandermonde-Matrix $V_0 = [C^0\mathbb{1}, \ldots, C^{s-1}\mathbb{1}]$ zu den Knoten c_i mit $C := \operatorname{diag}(c_1, \ldots, c_s)$. Dann gilt*

$$V_0^T V_0 = \begin{pmatrix} \mathbb{1}^T V_0 \\ \mathbb{1}^T V_0 F \\ \vdots \\ \mathbb{1}^T V_0 F^{s-1} \end{pmatrix}, \tag{3.18}$$

wobei F die Frobeniusmatrix zum Knotenpolynom $\varphi(x)$ aus Lemma 2.2 ist. Weiter ist

$$\mathbb{1}^T V_0 = \sum_{i=1}^{s} e_i^T F^{i-1}. \tag{3.19}$$

Beweis. Mit obigen Voraussetzungen gilt $CV_0 = V_0 F$ nach Lemma 2.2. Rekursives Anwenden liefert (3.18). Für den zweiten Teil der Behauptung nutzen wir die Beziehung $C^i = V_0 F^i V_0^{-1}$. Damit ist Gleichung (3.19) äquivalent zu

$$\mathbb{1}^T = \sum_{i=1}^{s} e_i^T V_0^{-1} C^{i-1} \tag{3.20}$$

Wir überprüfen (3.20) komponentenweise. Für den j-ten Eintrag erhalten wir

$$\sum_{i=1}^{s} e_i^T V_0^{-1} C^{i-1} e_j = \sum_{i=1}^{s} e_i^T c_j^{i-1} V_0^{-1} e_j = e_j^T V_0 V_0^{-1} e_j = 1,$$

und damit für $j = 1, \ldots, s$ die Behauptung. □

Mit Hilfe des Lemmas 3.2 können wir nun Ausdrücke der Form $\sum_{i=1}^{s} c_i^l$ in φ-Parametern ausdrücken, z.B. ist $c_1^4 + c_2^4 + c_3^4 + c_4^4 + c_5^4 = -4\,\varphi_1 + 4\,\varphi_2\varphi_4 + 2\,{\varphi_3}^2 - 4\,\varphi_3{\varphi_4}^2 + {\varphi_4}^4$.
Für $s = 4$ errechnen wir mit Hilfe des Lemmas 3.2 und der Definition (3.12) eine Darstellung des Stufenfehlers $X^T X$ in φ-Parametern. Substituieren wir mit Hilfe von (3.14), (3.15) und (3.17) die φ_1, φ_2 und φ_3 in dieser Darstellung, so ergibt sich schließlich der Stufenfehler als Polynom in φ_0

$$\begin{aligned}\|X\|_2^2 = \frac{319603072}{36905625} - \frac{83603182}{2460375}\,\varphi_0 + \frac{61357147}{1093500}\,{\varphi_0}^2 + \frac{5813179}{145800}\,{\varphi_0}^3 + \frac{1843733}{12960}\,{\varphi_0}^4 + \\ \frac{1286296}{2025}\,{\varphi_0}^5 + \frac{215365}{72}\,{\varphi_0}^6 + \frac{109375}{18}\,{\varphi_0}^7 - \frac{190625}{24}\,{\varphi_0}^8 - \frac{78125}{3}\,{\varphi_0}^9 + \frac{390625}{16}\,{\varphi_0}^{10}.\end{aligned} \tag{3.21}$$

Die einzige reelle Nullstelle von $\frac{d\|X\|_2^2}{d\varphi_0}$ und damit die Minimalstelle der konvexen Funktion ist $\varphi_0 = 0.1775140422832315$. Die Knoten des resultierenden Verfahrens sind in Tabelle 3.2 dargestellt. Numerische Ergebnisse findet man in [42]. Weil sich die Rechenschritte in diesem Abschnitt nur mit großem Aufwand manuell verifizieren lassen, haben wir im Anhang der Arbeit im Abschnitt A.2 ein Maple Arbeitsblatt entworfen, das sämtliche Rechenschritte enthält.

Bemerkung 3.7 (Fünfstufige Verfahren). Für $s = 5$ erhält man bei analogem Vorgehen bereits ein Polynom in zwei Variablen für $X^T X$. Unangenehmerweise werden die Knoten c_i komplex für die zugehörige Minimalstelle. Eine gute Wahl wurde numerisch gefunden, siehe Tabelle 3.2. ◊

Die zugehörigen Stabilitätsgebiete sind in Abbildung 3.12 dargestellt.

Tabelle 3.2: Explizite Verfahren der Ordnung $s + 2$ mit minimalem Stufenfehler.

EPTRK4

$s = 4$

$c_1 = 0.14896081139065023954118388536785985962 5$

$c_2 = 0.65193532739747708194173563879895300614 7$

$c_3 = 1.11723717693925119216234065444499569005 7$

$c_4 = 1.63610356235544614593415888300150765845 9$

EPTRK5

$s = 5$

$c_1 = .13721610394689589671676413664008418113 95$

$c_2 = .62875278636252338606564992495615394570 89$

$c_3 = 1.32870213148691591399035473852015631720 4$

$c_4 = 1.42545578665512924728614738911037573387 5$

$c_5 = 1.62020354869139269879822666791608696493 0$

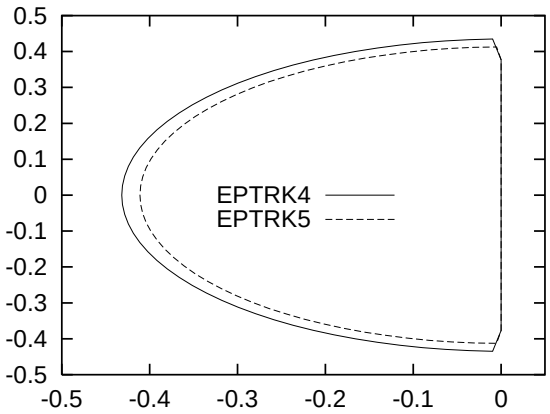

Abbildung 3.12: Stabilitätsgebiete für die expliziten Verfahren.

4 Implementierung von PTSW-Verfahren

4.1 Einleitung

Als ein Vorteil von sequentiellen W-Methoden gilt deren relativ einfache Implementierung. Im Gegensatz zu impliziten Runge-Kutta-Verfahren gibt es keine Iteration, deren Konvergenz abgesichert werden muß. Dadurch sind die W-Methoden robust und effizient, besonders auch für geringe Genauigkeitsanforderungen. Die Implementierung einer PTSW-Methode (2.4) wird durch den Zweischritt-Charakter schwieriger: wir benötigen Startwerte für den ersten Schritt, und bei Schrittweitenwechsel müssen die Verfahrenskoeffizienten neu bestimmt werden. Der Aufwand für diese Neuberechnung ist, schon bei moderater Dimension des Differentialgleichungssystems, vernachlässigbar gering; jedoch reagieren die PTSW-Verfahren empfindlich bei häufigen Schrittweitenwechseln. Mit einer angepaßten Heuristik versuchen wir deshalb, die Schrittweite möglichst über mehrere Zeitschritte konstant zu halten.
Das Generieren und Lösen der Stufengleichungssysteme unterscheidet sich, abgesehen von der Möglichkeit zur Parallelisierung, nicht von der Situation bei W-Methoden. Damit sind Standardtechniken, z.B. zur Approximation der Jacobimatrix oder zur Schätzung des lokalen Integrationsfehlers mit Hilfe von eingebetteten Verfahren, auf den PTSW-Fall übertragbar. Für große Differentialgleichungssysteme diskutieren wir die Lösung der Gleichungssysteme durch Krylovapproximation.
Aus der Konvergenzuntersuchung für singulär gestörte Probleme [54] folgt die Konvergenz der PTSW-Methoden für semi-explizite DAEs vom Index 1. Wir beschreiben die Implementierung für DAEs der Form $Gy' = f(t, y)$. Dabei ändern sich, verglichen mit dem ODE-Fall, nicht nur die Stufengleichungssysteme, sondern auch die Berechnungen für die Startwerte $k_{0,i}$.
Wir implementieren die PTSW-Verfahren in FORTRAN-77. Für die Parallelisierung setzen wir sogenannte shared-memory Compiler-Direktiven ein.

4.2 Auswahl der Verfahren

Um nicht die Übersicht zu verlieren, beschränken wir uns in dieser Arbeit bei der Implementierung und bei den numerischen Tests auf die in Tabelle 4.1 dargestellten PTSW-Verfahren. Dabei haben wir (nach umfangreichen Tests), die (hoffentlich) effizientesten ausgewählt. Sämtliche Verfahren erfüllen B(s), C(s), $\Gamma(s)$, haben also mindestens Ordnung s und sind bereits durch die Angabe der Knoten c_i und γ, durch die Steifgenauigkeit (2.22) oder durch die Wahl $v^T = 0$ und durch die vereinfachenden Konsistenzbedingungen (2.7) bestimmt. Gilt B($s+1$) nur für konstante Schrittweiten, so schreiben wir B*($s+1$). Zur Bezeichnung der Methoden verwenden wir folgendes Schema:

- PTSWsA: L(α)-stabile steifgenaue Verfahren mit $0 < \varrho(M(\infty)) < 1$. Für PTSW2A und PTSW3A gilt B$^*(s+1)$.
- PTSWsB: L(α)-stabile steifgenaue Verfahren mit $\varrho(M(\infty)) = 0$, konstruiert nach Satz 3.1.
- PTSWsC: PTSW2C und PTSW3C sind A(α)-stabil mit B$(s+1)$ und $v^T = 0$. PTSW4C ist L(α)-stabil, steifgenau und erfüllt $\varrho(M(\infty)) = 0$.

Uns interessiert bei der getroffenen Auswahl vor allem, wie sich die unterschiedlichen Forderungen nach Nilpotenz der Stabilitätsmatrix $M(\infty)$, Steifgenauigkeit oder Superkonvergenz $B(s+1)$ für $s = 2, 3$ und 4 auswirken.
Verfahren mit $s = 5$ und $s = 6$ wurden auch getestet. Obwohl sich die erwartete Ordnung numerisch zeigte und sie stabil waren, waren sie sehr empfindlich bei Schrittweitenwechseln und erwiesen sich in der getesteten Implementierung als nicht konkurrenzfähig.

Bemerkung 4.1. Bei den PTSW-Verfahren aus Tabelle 4.1 liegen einige Knoten c_i außerhalb des Intervalls $[0,1]$. Ist $c_{\max} = \max_{i=1}^{s} c_i > 1$, so setzen wir die Existenz einer eindeutig bestimmten Lösung für das Differentialgleichungssystem (2.1) bis zum Zeitpunkt $t = t_e + h_{N_{\text{Step}}}(c_{\max} - 1)$ voraus. Man beachte, daß wir im Gegensatz dazu durch die spezielle Wahl der Startprozedur keine zusätzlichen Annahmen benötigen, falls $c_{\min} = \min_{i=1}^{s} c_i < 0$ ist. $\Diamond$

Tabelle 4.1: Übersicht der implementierten PTSW-Verfahren.

s	Name	Stabilität, Extras	$\lvert C_{p+1}\rvert$	γ	Knoten c_i	$\varrho(M(\infty))$
2	PTSW2A	L(90), B*($s+1$)	$\lvert C_4\rvert \approx 0.18$	$4/5$	$23/9$ 1	0.8799
2	PTSW2B	L(90), $\varrho(M_\infty) = 0$	$\lvert C_3\rvert \approx 0.25$	0.63397459	-0.46410161 0.63397459 1	0
2	PTSW2C	A(81.1), B($s+1$)	$\lvert C_4\rvert \approx 0.12$	0.48	-1 $5/9$	0.6796
3	PTSW3A	L(88.4), B*($s+1$)	$\lvert C_5\rvert \approx 0.27$	0.85	-1 2.34246575 1	0.9178
3	PTSW3B	L(84.8), $\varrho(M_\infty) = 0$	$\lvert C_4\rvert \approx 0.19$	0.51554560	-2.02539286 -0.33469671 1	0
3	PTSW3C	A(87.3), B($s+1$)	$\lvert C_5\rvert \approx 0.39$	1	-0.96 0.52766970 3.28	0.9366
4	PTSW4A	L(87.9)	$\lvert C_5\rvert \approx 0.22$	$4/5$	-1 $-1/2$ 4 1	0.8844
4	PTSW4B	L(68.5), $\varrho(M_\infty) = 0$	$\lvert C_5\rvert \approx 0.17$	0.45645866	-3.32526780 -2.10534506 -0.26604845 1	0
4	PTSW4C	L(89.9), $\varrho(M_\infty) = 0$	$\lvert C_5\rvert \approx 0.53$	0.87242087	-2.74944216 -0.70716296 4.41533918 1	0

4.3 Der Startschritt

Die Berechnung des Startschrittes ist im Prinzip nicht schwierig, man integriert mit einem Standardintegrator solange, bis genügend viele Startdaten $y(t)$ und $y'(t)$ an den benötigten Zeitpunkten erzeugt sind. Wir verwenden folgenden Algorithmus:

Gegeben sei t_0, h_0, $u_0 = y(t_0)$ sowie $u_0' = y'(t_0)$,
gesucht $t_1 \geq t_0$ und $u_1 \approx y(t_1)$ und $k_{0,i} \approx y'(t_1 + (c_i - 1)h_0)$

1) Wir sortieren die Knotenmenge $T = \{c_1 - 1, \ldots, c_s - 1, 0\}$.
T enthält $\#T \quad (= s,$ oder $s + 1)$ verschiedene Elemente.
Bezeichne $T_1 \leq T_2 \leq \cdots \leq T_{\#T}$ diese Knoten.
2) Setze $t_1 := t_0 - h_0 T_1$
3) **For** $i = 1, \ldots, \#T$
4) integriere ODE bis $t = t_0 + h_0(T_i - T_1)$,
d.h. berechne $u_{T_i} \approx y(t_0 + h_0(T_i - T_1))$ und $u'_{T_i} = f(t, u_{T_i})$.
5) Falls $T_i = 0$, setze $u_1 := u_{T_i}$
6) Falls für ein $j \in \{1, \ldots, s\}$ $T_i = c_j - 1$ ist, setze $k_{0,j} := u'_{T_i}$.
7) Setze $h_1 = h_0$ für den ersten PTSW-Integrationsschritt.

Algorithmus 4.1: Berechnung der Startwerte

Zur Kontrolle sieht man, daß die Integrationszeitpunkte in Zeile 4) gerade die gewünschten Werte $t_1 + (c_i - 1)h_0$ bzw. t_1 annehmen. Die spezielle Wahl von t_1 ist die kleinstmögliche überhaupt, wenn wir eine (eventuell instabile) Integration in negative Zeitrichtung vermeiden wollen. Der Integrator ist beliebig, wir setzen für Rechnungen mit LU-Zerlegung bei kleinen Problemen RADAU [31] und für Rechnungen mit Krylovapproximation ROWMAP [53] ein. Auf den ersten Blick wirkt der Algorithmus etwas kompliziert, weil wir nicht die Zeitpunkte $t_1 + (c_i - 1)h_0$, sondern die „relativen Zeitpunkte“ T_i sortieren. Der Grund dafür ist, daß wir die Zeitschrittweite h_0 dadurch später z.B. in einer Anweisung

1b) $h_0 = \frac{t_e - t_0}{N_{\text{step}} - T_1}$,

festlegen können. Dies ermöglicht Rechnungen mit konstanter Schrittweite $h = h_0$, bei denen es (z.B. für Ordnungstests) wichtig ist, den Integrationsendpunkt t_e mit einer vorgegebenen Schrittzahl N_{step} genau zu treffen. Denn dann ergibt sich $t_1 + N_{\text{step}} h = t_0 - [T_1 - N_{\text{step}}] \frac{t_e - t_0}{N_{\text{step}} - T_1} = t_e$.

Bemerkung 4.2. Als Alternative zur obigen Startprozedur ist folgendes denkbar:

1. eine Anlaufrechnung mit sehr kleiner Anfangsschrittweite h_0 und den groben Näherungen $k_{0,i} = u_0'$, $u_1 = u_0$,
2. die Verwendung eines Startintegrators mit stetiger Erweiterung,
3. ein Selbststart im Rahmen einer Ordnungssteuerung.

Die erste Möglichkeit ist zwar einfach, liefert aber ungenaue Startwerte bzw. zahlreiche Schrittvergrößerungen und ist daher weniger interessant. Die anderen beiden versprechen einigen Effizienzvorteil und kommen für eine „Endanwenderversion“ der PTSW-Implementierung in Betracht. Bei den Testbeispielen in dieser Arbeit ist der Zeitaufwand des Startens jedoch vernachlässigbar gering und Algorithmus 4.1 ausreichend. $\Diamond$

4.4 Lösung der Stufengleichungen

4.4.1 Vorbetrachtungen

In den Stufen der PTSW-Verfahren (2.4) sind lineare Gleichungssysteme zu lösen. Dabei können die gleichen Techniken zur Implementierung wie bei sequentiellen, linear-impliziten Runge-Kutta-Verfahren (z.B.

[49]) eingesetzt werden. Zunächst stellen wir die Stufengleichung etwas um

$$(I - h_m\gamma T_m)x_i = r_i, \quad \text{mit } r_i = f(t_m + h_m c_i, Y_{mi}) + \sum_{j=1}^{s} \frac{\gamma_{ij}}{\gamma} k_{m-1,j} \text{ und } k_{mi} = x_i - \sum_{j=1}^{s} \frac{\gamma_{ij}}{\gamma} k_{m-1,j}, \quad (4.1)$$

und sparen somit die Multiplikation mit T_m auf der rechten Seite. Zur Abkürzung haben wir Variablen $x_i = k_{mi} + \sum_{j=1}^{s} \frac{\gamma_{ij}}{\gamma} k_{m-1,j}$ und r_i für die rechte Seite eingeführt (und dabei zugunsten der Übersichtlichkeit auf den zusätzlichen Index m verzichtet). Für die (approximative) Lösung der Stufen (4.1) haben wir viele Möglichkeiten. Bewährt haben sich LU-Zerlegung, Krylovapproximation [46, 38] oder z.B. approximierende Matrixfaktorisierung [28]. Solange die berechnete Näherungslösung als exakte Lösung mit einer gestörten Matrix $T_{m,i} \approx T_m$ (und $T_{m,i} = \mathcal{O}(1)$) interpretiert werden kann, bleibt die Konvergenzordnung des Integrationsverfahrens erhalten (vgl. Bemerkung 2.1). Wesentlichen Einfluß hat der Gleichungssystemlöser auf die Stabilität des Integrationsverfahrens und (weniger gravierend) auch auf die Größe der lokalen Fehlerkonstanten.

4.4.2 Lösung mit LU-Zerlegung

Bei kleinen Problemen oder bei Problemen mit geringer Bandbreite der Jacobimatrix lösen wir die linearen Gleichungssysteme mit LU-Zerlegung. Die Jacobimatrix berechnen wir approximativ mit Differenzen-Quotienten. Hat die Jacobimatrix Bandbreite m_b so benötigen wir $m_b + 1$ Funktionsauswertung, nämlich $f(t_m, u_m)$ und $f(t_m, u_m + \varepsilon\xi_l), \quad l = 1, \ldots, m_b$, mit geeigneten Testvektoren ξ_l aus Nullen und Einsen (im Abstand m_b). Die Einträge der Jacobimatrix erhalten wir in den Differenzen-Quotienten $dq_l = (f(t_m, u_m + \varepsilon\xi_l) - f(t_m, u_m))/\varepsilon$. Diese Berechnung läßt sich leicht parallelisieren. Entweder in einer Schleife (links) oder in zweien (rechts):

Jacobimatrixberechnung, Variante 1
$f_0 = f(t_m, u_m)$
do parallel $l = 1, \ldots, m_b$
  $f_l = f(t_m, u_m + \varepsilon\xi_l)$
  $dq_l = (f_l - f_0)/\varepsilon$
end do

Jacobimatrixberechnung, Variante 2
$x_0 = 0$
do parallel $l = 0, \ldots, m_b$
  $f_l = f(t_m, u_m + \varepsilon\xi_l)$
end do
do parallel $l = 1, \ldots, m_b$
  $dq_l = (f_l - f_0)/\varepsilon$
end do

Variante 1 ist von Vorteil, wenn f_0 bereits zur Verfügung steht (z.B. durch die letzte Stufe im vorangegangenen Schritt $f_0 \approx k_{m-1,s}$), oder der Startaufwand für eine Parallelisierung hoch ist. Bei Variante 2 erwarten wir (für teures f und kleine Bandbreite m_b) deutlich bessere Speedups, weil sämtliche $m_b + 1$ Funktionsauswertungen nun parallel ausgeführt werden können. In unserer Implementierung entscheiden wir uns für die Variante 1 bei der Berechnung von T_m.

Bemerkung 4.3. Eine interessante Möglichkeit zur Berechnung der Jacobimatrix bietet das automatische Differenzieren. Aus dem Fortran-Quelltext der rechten Seite der Differentialgleichung erstellt z.B. ADIFOR eine Unterroutine für die Jacobimatrix bzw. Jacobimatrix-Vektor-Produkte durch symbolisches Differenzieren, z.B. [24]. Die generierten Unterroutinen sind i.a. etwas langsamer als handgeschriebene, bei eigenen Messungen mit semidiskretisierten Diffusionsgleichungen war der Unterschied allerdings sehr gering. Z.B. wird automatisches Differenzieren z.B. in DAEPACK verwendet (http://yoric.mit.edu/daepack/daepack.html). ◊

Mit T_m bilden wir die Matrix $(I - h_m\gamma T_m)$. Für die anschließend benötigte LU-Zerlegung von $(I - h_m\gamma T_m)$ verwenden wir die LAPACK-Unterprogramme `DGETRF` bzw. `DGBTRF` (für Bandmatrizen). Obwohl wir auf der verwendeten Testplattform (HP/Convex X-Class) vom Hersteller parallelisierte LAPACK-Routinen verwenden, erweist sich die LU-Zerlegung, zumindest bei kleinen Problemen oder Problemen mit Bandstruktur, als sequentieller Flaschenhals. Ein ähnliches Ergebnis erhielten wir auch auf einer Workstation Sun Enterprise 420 unter Verwendung der parallelen SUN-Performance-Bibliothek. Auch Pulch [43] hat, bei Verwendung von ScaLAPACK, trotz dichter Jacobimatrizen mit Dimension 1000×1000, kaum Speedups bei der LU-Zerlegung erzielen können.

Wichtig für die Effizienz der PTSW-Verfahren ist, daß die Jacobimatrix nicht in jedem Schritt neu berechnet werden muß, sondern daß sie einige Zeitschritte lang wiederverwendet werden kann. Wir verwenden die folgende, einfache Heuristik: Sei vor j Schritten die jüngste Jacobimatrix berechnet und zerlegt worden $T_{m-j} \approx f_y(t_{m-j}, u_{m-j})$. Dann verwenden wir diese auch im aktuellen Schritt, wenn

$$\left| \frac{h_m - h_{m-j}}{h_{m-j}} \right| \leq 0.1, \quad \text{und} \quad j \leq s$$

ist. Andernfalls berechnen und zerlegen wir eine neue Matrix $(I - h_m\gamma T_m)$.

4.4.3 Lösung mit Krylovverfahren

In den zurückliegenden Jahren ist der Einsatz von Krylovunterraumtechniken in Zeitintegrationsverfahren intensiv untersucht worden, z.B. [8, 9, 16, 45]. Dabei hat sich eine hervorragende Eignung für sehr große, steife Probleme herausgestellt. Weil bei der Lösung der linearen Gleichungssysteme innerhalb des Integrationsprozesses i.a. nur moderate Genauigkeiten erreicht werden müssen, genügen oft kleine Krylovdimensionen. Bei einer sogenannten „matrixfreien Implementierung“, ohne explizite Erzeugung und Speicherung der Jacobimatrix, sind die Speicherplatzanforderungen entsprechend gering. Durch adaptive, residuumkontrollierte Wahl der Krylovdimension erhält man eine automatische Steifheitserkennung: ist das Problem nichtsteif, so bleibt die Dimension gering, und es wird fast explizit integriert. Erhöht sich die Steifheit, so steigt auch die Krylovdimension und das Verfahren rechnet implizit. Z.B. wird in der Arbeit von Botchev u.a. [6] allein aus einer Stabilitätsbedingung die notwendige Krylovdimension bestimmt.

Zur Lösung der Stufen (4.1) setzen wir bei den PTSW-Verfahren die Methode der vollständigen Orthogonalisierung [44] (*fully orthogonalization method, FOM*) ein. Wir generieren für jede Stufe i einen Krylovraum $\mathcal{K}_i$ zur Jacobimatrix $A = f_y(t_m, u_m)$

$$\mathcal{K}_i = \text{span}(r_i, Ar_i, \ldots A^{\kappa_i - 1} r_i) = \text{span}(r_i, (I - h_m\gamma A)r_i, \ldots, (I - h_m\gamma A)^{\kappa_i - 1} r_i), \tag{4.2}$$

und bestimmen $x_i \in \mathcal{K}_i$ durch die Orthogonalitätsbedingung

$$\delta_i := [r_i - (I - h_m\gamma A)x_i] \perp \mathcal{K}_i \tag{4.3}$$

an das Residuum δ_i. Die verwendete Krylovdimension κ_i wird so gewählt, daß für das Residuum $\|\delta_i\|_2 <$ `KTOL` für eine vorgegebene Toleranz `KTOL` gilt oder die Maximaldimension $\kappa_i = \kappa_{\max} = 50$ erreicht ist. Wir koppeln `KTOL` an die Toleranz der Schrittweitensteuerung mit `KTOL` $=$ `ATOL`$/h_m$. Eine effiziente Bestimmung von x_i erlaubt der Arnoldi-Alorithmus, der die Berechnung der Jacobimatrix-Vektor-Produkte mit einer Gram-Schmidt-Orthogonalisierung verbindet und so eine Orthogonalbasis Q_i zum Krylovraum $\mathcal{K}_i$ generiert, d.h. $\mathcal{K}_i = \text{span}(q_1, \ldots, q_{\kappa_i})$. Wir können das Gesamtverfahren mit folgendem Lemma als PTSW-Lösung mit der Matrix $T_{mi} = Q_i Q_i^T A$ interpretieren.

Lemma 4.1. *Seien die Spalten der Matrix Q_i eine Orthogonalbasis zum Krylovunterraum*

$$\mathcal{K}_i = \mathit{span}(r_i, Ar_i, \dots A^{\kappa_i - 1} r_i).$$

Dann liegt die Lösung x_i des linearen Gleichungssystems

$$(I - h_m \gamma Q_i Q_i^T A) x_i = r_i \tag{4.4}$$

im Krylovraum, d.h. $x_i = Q_i Q_i^T x_i$, und es gilt (4.3)*. Effektiv berechnen läßt sich x_i durch Lösung eines linearen Gleichungssystems der Dimension $\kappa_i \times \kappa_i$, denn es gilt*

$$x_i = Q_i l_i, \quad \textit{mit} \quad (I - h_m \gamma Q_i^T A Q_i) l_i = Q_i^T r_i. \tag{4.5}$$

Beweis. Wegen $x_i = r_i - h_m \gamma Q_i Q_i^T A x_i = Q_i Q_i^T \left[r_i - h_m \gamma A x_i \right]$ liegt x_i im Krylovraum $\mathcal{K}_i$. Und wegen

$$Q_i^T [r_i - (I - h_m \gamma A) x_i] = Q_i^T [(I - h_m Q_i Q_i^T A) x_i - (I - h_m \gamma A) x_i] = 0$$

gilt Bedingung (4.3).
Setzen wir $x_i = Q_i Q_i^T x_i$ und $r_i = Q_i Q_i^T r_i$ in Gleichung (4.4) ein, so erhalten wir durch Linksmultiplikation mit Q_i^T das System (4.5) für $l_i = Q_i^T x_i$. □

Bei der Berechnung der Orthogonalbasis Q_i durch den Arnoldiprozeß hat die Matrix $Q_i^T A Q_i$ Hessenbergform. Durch Anwendung von Givens-Drehungen bringt man die Subdiagonale zum Verschwinden und erhält eine QR-Zerlegung von $(I - h_m \gamma Q_i^T A Q_i)$. Während der sukzessiven Vergrößerung des Krylovraums kann diese QR-Zerlegung einfach aktualisiert werden. Dies erlaubt auch eine effiziente Berechnung des aktuellen Residuums δ_i in jedem Krylovschritt. Eine ausführliche Beschreibung der Implementierung des Arnoldiprozesses gibt Saad [44].

Für den exponentiellen Krylovintegrator `exp4` in der Arbeit von Hochbruck u.a. [33] werden die rechten Seiten so transformiert, daß sie 'klein' sind, um rasche Konvergenz und damit niedrige Krylovdimensionen zu erziehlen. Auch in (4.4) ist die rechte Seite r_i klein, was zumindest bei nichtsteifen Problemen kleine Krylovdimensionen ermöglicht.

Lemma 4.2. *Für die rechte Seite r_i in den Stufen* (4.1) *eines PTSW-Verfahrens* (2.4)*, das die Voraussetzungen von Satz 2.8 (a) erfüllt (C(q),B(p) und Γ(q) mit $p \geq q$), gilt*

$$r_i = f(t_m + h_m c_i, Y_{mi}) + \sum_{j=1}^{s} \frac{\gamma_{ij}}{\gamma} k_{m-1,j} = \mathcal{O}(h^q).$$

Beweis. Mit dem Konvergenzsatz 2.8 folgt

$$\begin{aligned} r_i &= f(t_m + h_m c_i, y(t_m + c_i h_m) + \mathcal{O}(h^q)) + \sum_{j=1}^{s} \frac{\gamma_{ij}}{\gamma} y'(t_m + (c_j - 1) h_{m-1}) + \mathcal{O}(h^q) \\ &= y'(t_m + c_i h_m) + \sum_{j=1}^{s} \frac{\gamma_{ij}}{\gamma} y'(t_m + (c_j - 1) h_{m-1}) + \mathcal{O}(h^q) \end{aligned}$$

und mit $\Gamma(q)$

$$= y'(t_m + c_i h_m) - y'(t_m + c_i h_m) + \mathcal{O}(h^q).$$

□

Die im Arnoldi-Algorithmus benötigten Jacobimatrix-Vektor-Produkte berechnen wir matrixfrei mittels finiter Differenzen

$$Av = f_y(t_m, u_m)v = \frac{1}{\varepsilon}\left(f(t_m, u_m + \varepsilon v) - f(t_m, u_m)\right) + \mathcal{O}(\varepsilon\|v\|^2), \tag{4.6}$$

mit $\varepsilon = 10^{-7} * \max(10^{-5}, \|\frac{1}{\sqrt{n}}u_m\|_2)$. Den für alle Stufen benötigten Funktionsaufruf $f(t_m, u_m)$ müssen wir in jedem Schritt vorab berechnen, was bei kleinen Krylovdimensionen den maximalen Speedup deutlich verringert. Für solche Beispiele wären bei steifgenauen Verfahren mit $c_s = 1$ die Approximationen $f(t_m, u_m) \approx k_{m-1,s}$, $f(t_m, u_m) \approx f(Y_{m-1,s})$ zu diskutieren.

Bemerkung 4.4 (BiCGStab). Ein Nachteil der Methode der vollständigen Orthogonalisierung (FOM) ist der mit der Krylovdimension quadratisch wachsende Orthogonalisierungsaufwand. Für bestimmte Probleme ist daher die BiCGStab-Iteration von van der Vorst [52] eine interessante Alternative. Wir haben zur Lösung der Stufengleichungen bei den PTSW-Verfahren testweise BiCGStab eingesetzt (indem wir die *reverse-communication* Routine `BiCGSTABREVCOM.f` aus [1] eingebunden haben). Die Rechenzeiten für die Testprobleme dieser Arbeit sind dabei, insbesondere für strengere Toleranzen, deutlich angestiegen. Der benötige Arbeitsspeicher war jedoch, wegen der kurzen Rekursion von BiCGStab, viel geringer als bei der FOM-Implementierung. ◊

Bemerkung 4.5 (Vorkonditionierung). Um die Krylovdimension und damit den Aufwand zu reduzieren, ist es möglich Vorkonditionierungstechniken einzusetzen. Sei P_m eine Approximation an die Jacobimatrix $A = f_y(t_m, u_m)$, für die eine Multiplikation mit dem Vorkonditionierer $B := (I - h_m\gamma P_m)^{-1}$ leicht berechenbar ist. Für eine Linksvorkonditionierung müßten wir das System

$$B(I - h_m\gamma A)x_i = Br_i$$

approximativ durch FOM mit dem Krylovraum $\mathcal{K}_i$

$$\mathcal{K}_i = \text{span}\left(Br_i, B(I - h_m\gamma A)r_i, \ldots, (B(I - h_m\gamma A))^{\kappa_i - 1}\, Br_i\right)$$

lösen. Einige Beispiele dafür, wie effektiv die Krylovdimension bei guter Vorkonditionierung reduziert werden kann, finden sich in der Arbeit [40] zur Lösung von Reaktions-Diffusionsgleichungen mit dem Krylovintegrator ROWMAP. ◊

4.5 Schrittweitensteuerung

Für eine Implementierung der PTSW-Verfahren mit variablen Schrittweiten benötigen wir ein Unterprogramm, welches bei Schrittweitenwechsel neue PTSW-Koeffizienten gemäß den vereinfachenden Konsistenzbedingungen (2.14) berechnet, eine Schätzung des lokalen Integrationsfehlers durch ein eingebettetes Verfahren niedrigerer Ordnung und, darauf basierend, einen Schrittweitenvorschlag.
Die Neuberechnung der Koeffizienten erfordert zwei Multiplikationen von $s \times s$ Matrizen und das Skalieren mit $S = \text{diag}(1, \sigma, \ldots, \sigma^{s-1})$. Schon für moderate Dimension des Differentialgleichungssystems ist der Aufwand hierfür im Vergleich mit den übrigen Operationen vernachlässigbar. Wie man ein solches Unterprogramm automatisch mittels Maple generiert, zeigen wir im Anhang dieser Arbeit.
Um den lokalen Fehler Err_{emb} während der Integration abzuschätzen, verwenden wir, wie üblich (z.B. [29]), ein eingebettetes Verfahren der Ordnung $s - 1$. Wir wählen als Gewichte für die Einbettung

$$b_e^T = 0.95b^T \quad \text{und} \quad v_e^T = ((\mathbb{1}^T + \varepsilon e_s^T)SD^{-1} - b_e^T V_0 S)V_1^{-1}, \tag{4.7}$$

und haben dann ein von σ unabhängiges Residuum in B(s) (Gl. (2.7)) von $\varepsilon := 0.1$. Den lokalen Fehler Err_{emb} messen wir in einer gewichteten Norm

$$Err_{\text{emb}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{u_{m+1,i} - \tilde{u}_{m+1,i}}{\mathtt{ATOL} + \mathtt{RTOL}|u_{m+1,i}|}\right)^2}, \quad \tilde{u}_{m+1} = u_m + h_m \sum_{i=1}^{s}(b_{e,i}k_{m,i} + v_{e,i}k_{m-1,i}),$$

mit vorgegebener absoluter und relativer Toleranz. Bei den numerischen Experimenten in dieser Arbeit wählen wir stets $\mathtt{ATOL} = \mathtt{RTOL} = 10^{-i}, i = 2, \ldots, 8$. Aus dem lokalen Fehler Err_{emb} berechnen wir einen Schrittweitenfaktor

$$\sigma^* = \min\{2, \max\{0.5, Err_{\text{emb}}^{-1/s} \cdot 0.85\}\},$$

und aus diesem eine neue Schrittweite h^* mit

$$h^* = \begin{cases} h_m & \text{falls } \sigma^* > 1.05 \text{ und für ein } j \in \{0, \ldots, s-1\} \text{ ist } \sigma_{m-j} > 1 \\ h_m & \text{falls } \sigma^* \in [0.95, 1.05] \\ h_m\sigma^* & \text{sonst,} \end{cases}$$

um häufige Schrittweitenwechsel zu vermeiden. Wir akzeptieren den Schritt, falls Err_{emb} kleiner oder gleich eins ist und setzen $h_{m+1} := h^*, \sigma_{m+1} := h_{m+1}/h_m$. Ist der geschätzte Fehler größer, wiederholen wir den Schritt mit $h_m := h^*, \sigma_m := h_m/h_{m-1}$. Wie man sieht, lassen wir in s aufeinanderfolgenden Schritten nur maximal eine Vergrößerung zu. Durch das „Quasikonstanthalten“ werden die PTSW-Verfahren robuster. Außerdem können wir alte LU-Zerlegungen wiederverwenden, vergleiche Abschnitt 4.4.

Bemerkung 4.6 (Not a Number). Im Programm testen wir noch einen dritten Fall

$$\text{if } (Err_{\text{emb}} \neq Err_{\text{emb}}) \text{ then } h_m := 1/2h_m; \text{ reject step; end if.}$$

Auf den ersten Blick erscheint diese Anweisung überflüssig, doch bei IEEE-konformer Implementierung der Fließkommaarithmetik wird sie dann ausgeführt, wenn Err_{emb} eine „Nicht-Zahl“ (*not a number, NaN*) ist. Treten in einem Zeitschritt illegale Operationen auf (Exponentenüberläufe, Divisionen durch Null, etwa weil das lineare Gleichungssystem singulär ist, oder $f(t, u)$ außerhalb des Definitionsbereichs ausgewertet wird), so reagieren wir angemessen: mit Schrittweitenreduktion und Wiederholung des Zeitschritts und verhindern damit ein „Festlaufen“ des Integrators mit $h_{m+1} = \text{NaN}$. ♢

4.6 Anwendung auf differential-algebraische Systeme

Wir betrachten ein differential-algebraisches Anfangswertproblem in der Form

$$Gy' = f(t, y), \quad y(t_0) = y_0 \tag{4.8}$$

mit einer konstanten Matrix $G \in \mathbb{R}^{n,n}$ und konsistenten Anfangswerten. Eine für dieses Problem geeignete Formulierung der PTSW-Verfahren liefert der sogenannte „direkte Weg“ [30]: wir nehmen zunächst an, G sei regulär, erhalten analog zu Gleichung (4.1) die Stufengleichung

$$(I - h_m G^{-1}\gamma T_m)(k_{mi} + \sum_{j=1}^{s}\frac{\gamma_{ij}}{\gamma}k_{m-1,j}) = G^{-1}f(t_m + h_m c_i, Y_{mi}) + \sum_{j=1}^{s}\frac{\gamma_{ij}}{\gamma}k_{m-1,j},$$

oder, äquivalent dazu,

$$(G - h_m\gamma T_m)(k_{mi} + \sum_{j=1}^{s}\frac{\gamma_{ij}}{\gamma}k_{m-1,j}) = f(t_m + h_m c_i, Y_{mi}) + G\sum_{j=1}^{s}\frac{\gamma_{ij}}{\gamma}k_{m-1,j}. \tag{4.9}$$

In dieser Darstellung (4.9) können wir uns von der Regularität von G lösen. Obwohl das lineare Gleichungssystem (4.9) prinzipiell auch mittels Krylovapproximation gelöst werden könnte, was für große DAE-Systeme attraktiv erscheint und eine interessante Aufgabe für weitere Untersuchungen der PTSW-Verfahren ist, möchten wir uns hier bei der Implementierung nur auf LU-Zerlegungen beschränken. Aus der Analyse für singulär gestörte Probleme [54] folgt die Konvergenz für Probleme vom Index 1 in semi-expliziter Form. Eine eingehende theoretische Untersuchung der PTSW-Verfahren für Probleme von höherem Index bleibt offen. Bei der Bestimmung der numerischen Ordnung für Testprobleme mit Index 2 und 3 zeigte sich jedoch keine Ordnungsreduktion in den differentiellen Komponenten [41].
Schwierigkeiten bereitet der Start, weil die Approximationen an die Ableitung der Lösung für die Stufenwerte $k_{0,i}$ (im Startalgorithmus in Schritt 4) weder durch einfache f-Auswertung, noch als Rückgabeparameter der Integration, beispielsweise in RADAU, gewonnen werden können. Allerdings werden in RADAU (vgl. [30], S. 118) intern transformierte Stufen $Z_{\text{radau}} = (z_1^T, \ldots, z_s^T)^T$ berechnet, die die benötigte Information enthalten. Es ist

$$Z_{\text{radau}} = h_{\text{radau}}(A_{\text{radau}} \otimes I)K_{\text{radau}}, \tag{4.10}$$

d.h., wir finden wegen $c_{s,\text{radau}} = 1$ unsere gewünsche Näherung $u'_{T_i} = k_{s,\text{radau}}$ mit

$$k_{s,\text{radau}} = \frac{1}{h_{\text{radau}}}(e_s^T A_{\text{radau}^{-1}}^{-1} \otimes I)Z_{\text{radau}} \tag{4.11}$$

In unserer Implementierung der PTSW-Verfahren für DAEs verwenden wir RADAU ohne Ordnungssteuerung, mit 3 Stufen (`iwork`(11)=`iwork`(12)=3), extrahieren die internen Stufen Z_{radau} aus dem übergebenen Arbeitsfeld `work` und multiplizieren sie mit dem Vektor $e_s^T A_{\text{radau},3}^{-1} = \left(-1 + 8/3\sqrt{6}, \quad -1 - 8/3\sqrt{6}, \quad 5\right)$.

4.7 Programmierung und Parallelisierung

4.7.1 Einleitung

Bei der Programmierung von Zeitintegrationsverfahren haben sich über die Jahre gewisse Standardtechniken etabliert, von denen wir auch Gebrauch machen werden. Als gute Vorlagen gelten beispielsweise die Fortran-Codes RADAU und RODAS von Hairer und Wanner [30] und die VODE-Familie (VODE, VODPK, PVODE) von Hindmarsh, Brown, u.a., z.B. [7]. Wir orientieren uns bei der Programmierung der PTSW-Codes `ptsw.f` (LU-Zerlegung, DAEs) und `ptswk.f` (Krylovapproximation) an diesen Standards und beschreiben hier einige Besonderheiten der PTSW-Verfahren. Wir verwenden FORTRAN-77. Die anwendungsbereiten Codes können vom Autor bei Bedarf angefordert werden.

4.7.2 Nutzerschnittstelle

Die Schnittstelle für den Nutzer von `ptsw.f` und `ptswk.f` besteht aus einer Subroutine der Form

```
      subroutine ptswk (n,f,t,u,uprime,h,te,rtol,atol,
     $     work,lwork,iwork,liwork,mf,rpar,ipar,info)
      implicit none
      integer n,lwork,liwork,iwork(liwork),ipar(*),info
      real*8  t,u(n),uprime(n),h,te,rtol,atol,work(lwork),rpar(*)
      external f
      character (len=*) mf
```

zur Integration. Außer Anfangswerten und einigen Steuerparametern wird lediglich eine Funktion `f` mit folgender Deklaration

```
subroutine f(n,t,u,uprime,rpar,ipar)
```

für die rechte Seite der Differentialgleichung (2.1) benötigt. Die Felder `rpar` und `ipar` werden im Integrator nicht verwendet und können, z.B. zur Parameterübergabe an `f`, vom Nutzer selbst verwaltet werden. Zur Übergabe von zusätzlichen Optionen und zur Integrationsstatistik dienen die ersten Einträge der Felder `work` und `iwork`. Eine detaillierte Dokumentation der vollständigen Schnittstelle ist in den Fortran-Quelltexten `ptsw.f` und `ptswk.f` als Kommentar enthalten.

4.7.3 Speicherverwaltung

Durch die statische Speicherverwaltung von FORTRAN-77 müssen sämtliche Vektoren und Matrizen der Dimension n im Arbeitsfeld `work` abgelegt werden, auch wenn sie nur temporär benötigt werden. Der Integrator berechnet zunächst für jeden Vektor einen Startindex p im `work`-Feld. Mit `work`$(p-1+i)$ kann dann auf die i-te Komponente zugegriffen werden. Diese, insbesondere bei mehrdimensionalen Daten, umständliche Manipulation, kann durch die Vergabe von Aliasnamen in einer inneren Subroutine mit entsprechenden Übergabeparameter wesentlich vereinfacht werden. Es ergibt sich folgende Programmstruktur,

```
subroutine ptswk(...)
integer p_k,p_gmat
....
p_gmat= ...
p_k=p_gmat + sizeof_gmat
...
call ptswkcore &
& (..,work(p_gmat),work(p_k),..)...
end
```

```
subroutine ptswkcore(...,gmat,k,...)
integer ldgmat,n,s
real*8 gmat(n,ldgmat),k(n,s,2)
...
gmat(i,j) = ...
! Neue Stufen:
k(l,i,flag)=... k(l,i,3-flag)
end
```

wobei im „Core“-Integrator auf die Daten in der gewohnten Weise zugegriffen werden kann. Das Feld `k(n,s,2)` enthält übrigens die Stufen K_m und K_{m-1}. Um Kopieraufwand beim nächsten Zeitschritt zu sparen, wird der Zugriffsindex `flag` $\in \{1,2\}$ in einer Anweisung der Form `flag=3-flag` gespiegelt. Für lokale, temporäre Vektoren in den parallelen Stufen benötigen wir s-fache Kopien, z.B. `tmp(len,s)`, weil bei der Parallelisierung wegen der statischen Speichervergabe keine Vektoren privatisiert werden können (siehe unten). Der Zugriff auf die jeweilige Kopie der Länge `len` erfolgt mit Stufenindex i, also `tmp(1,i)`.

4.7.4 Parallelisierung

Weil wir Rechner mit gemeinsamem Speicher betrachten, ist die Parallelisierung der PTSW-Verfahren einfach. Arbeitsmittel sind Direktiven in Form von Fortran-Kommentaren für parallelisierende Compiler. Ein sequentieller Standardcompiler ignoriert diese Direktiven, so daß nur ein Quelltext (für parallele und sequentielle Kompilierung) erstellt werden muß. Leider unterscheiden sich Syntax und Semantik der von verschiedenen Herstellern (wie Sun, Cray oder Hewlett-Packard/Convex) entwickelten Compilererweiterungen, obwohl eine ähnliche Funktionalität implementiert wird. Abhilfe schafft der im Oktober 1997 verabschiedete Standard OpenMP 1.0 (http://www.openmp.org, im November 2000 OpenMP 2.0), der jedoch auf der Testplattform dieser Arbeit (Convex X-Class) nicht verfügbar ist.
Wir haben in unsere Codes `ptsw.f` und `ptswk.f` die herstellerspezifischen CPS-Direktiven (von Convex/HP) und standardisierte OpenMP-Direktiven integriert und erläutern hier die Parallelisierung der Stufen von `ptswk.f` mittels OpenMP. Im Quelltext genügt eine einzige OpenMP-Direktive, hier dargestellt in Fortran mit Pseudocode,

```
!omp parallel do private (j) shared (f)
 do i=1,s
   Y_mi = u_m + sum_{j=1}^{s} a_ij k_{m-1,j}
   rechte Seite r_i = f(t_m + c_i h_m, Y_mi) + sum_{j=1}^{s} (gamma_ij / gamma) k_{m-1,j}
   Krylovlösung x_i = FOM(r_i, u_m, t_m, f, tmp(1,i))
   neue Stufen k_mi = x_i - sum_{j=1}^{s} (gamma_ij / gamma) k_{m-1,j}
 end do
!omp end parallel do
```

um die zu parallelisierende Schleife für den Compiler zu markieren. Jeder Thread arbeitet mit einer „privaten Kopie“ der inneren Variablen j, gekennzeichnet für den Compiler durch `private(j)`. Die `shared`-Anweisung ist weniger plausibel. Alle Threads benötigen die gleiche Subroutine `f` zur Berechnung der rechten Seite der Differentialgleichung, und wir müssen das explizit angeben, weil `f` nur eine Referenz (vom Typ `external`) und nicht die Subroutine selbst ist.
Bei Vektorvariablen haben wir beim Programmdesign darauf geachtet, daß es nicht zu Kollisionen zwischen den Threads kommt (z.B. durch `tmp(1,i)`). Bei skalaren Variablen (wie j) innerhalb der Schleife können wir `private` einsetzen. Probleme bereiten jedoch die lokalen Variablen in `f` und in `FOM`, wenn sie (FORTRAN-77-typisch) global und statisch vom Compiler angelegt werden. Deshalb bieten OpenMP-fähige Übersetzer eine Option (z.B. `-stackvar`) an, Daten nicht global, sondern auf einem (Thread-eigenen) Stack anzulegen. Verwenden wir größere Felder, so ist die vorgegebene Stackgröße schnell zu klein und somit Ursache von rätselhaften, kaum zu lokalisierenden Programmabstürzen. Für eine erfolgreiche Parallelisierung müssen wir daher i.a. die Stackgröße erhöhen. Die Anzahl der Threads für eine Programmausführung wird durch die Umgebungsvariable `OMP_NUM_THREADS` festgelegt. Als Referenzbeispiel geben wir hier die notwendigen Kommandos für eine OpenMP-Parallelisierung von `ptswk.f` auf einem SUN-Fire-Server unter Solaris 8 zum Kompilieren und Ausführen an:

```
$ f90 -o ptswk -fast -openmp -noautopar \
$    -stackvar driver.f bsp.f ptswk.f -lsunperf_mt
$
$ ulimit -s unlimited
$ export STACKSIZE=65535
$ export OMP_NUM_THREADS=4  # Anzahl Threads
$ ./ptswk                   # Programm starten!
```

Für weitere Beispiele, auch für die Convex X-Class, verweisen wir auf das `Makefile` in der PTSW-Software sowie auf die System- und Compilerdokumentation des jeweiligen Herstellers.

Bemerkung 4.7 (Automatische Felder in Fortran 90, synchrone Caches). Wie oben erläutert ist, arbeiten parallele Threads in unseren Codes `ptsw.f` und `ptswk.f` mit globalen Feldern. Konflikte zwischen den Threads treten nicht auf, weil Threads nur in eigenen Bereichen Einträge ändern. Die Analyse eines Compilers kann diese Kollisionsfreiheit nicht feststellen, deshalb versagt die automatische Parallelisierung und helfende Direktiven sind notwendig. Würden wir, z.B. mit automatischen Feldern in Fortran 90, lokale Felder anstelle der globalen verwenden, so wären die Threads deutlich besser getrennt, und der Compiler hätte mehr Optimierungsmöglichkeiten. Z.B. könnte ein aufwendiges Cache-Synchronisieren zwischen den Prozessoren entfallen. Leider konnten bei entsprechenden Untersuchungen auf der Testplattform keine Geschwindigkeitsvorteile gemessen werden, weil die automatischen Arrays nicht auf dem Stack, sondern

via `allocate` im Heap angelegt wurden. Hier zeigt sich die Kehrseite der faszinierend einfachen Shared-Memory-Parallelisierung mit OpenMP: man kann als Anwender keinen Einfluß auf Interna wie Speicherlayout oder Cache-Synchronisierungen nehmen. Man kann sich, selbst bei unbelasteter Maschine, nicht einmal darauf verlassen, daß jeder Thread auf einer eigenen CPU läuft. Wir erhalten daher bei der Messung der Speedups im folgenden Kapitel also nur untere Schranken, die sich vermutlich durch hardwarenahe Programmierung verbessern ließen. ◊

5 Numerische Tests

5.1 Benchmarks für ODE-Integratoren

In diesem Kapitel vergleichen wir die PTSW-Verfahren untereinander und mit Standardintegratoren hinsichtlich ihrer Praxistauglichkeit. Uns interessiert, wie zuverlässig sie mit unterschiedlichen Toleranzen vorgegebene Testprobleme lösen und wie hoch der Aufwand zum Erreichen einer bestimmten Genauigkeit ist. Die Ergebnisse, dargestellt in Rechenzeit-Fehler-Diagrammen (wie in der Literatur üblich, z.B. [26, 23, 30]), müssen vorsichtig interpretiert werden, weil ein derartiger Vergleich Schwächen hat:

- Die gemessenen Rechenzeiten sind auf anderen Plattformen eventuell schlecht reproduzierbar. Wie überraschend groß die Unterschiede sein können, zeigt Beispiel 5.1 unten. In älteren Arbeiten findet man anstelle der Rechenzeiten eine Statistik der benötigten Operationen: f-Auswertungen, Matrixfaktorisierungen und Rücksubstitutionen. Diese Werte sind gut reproduzierbar. Das Problem ist damit nicht gelöst, sondern nur verschoben: für eine praktische Beurteilung benötigen wir nun eine Wichtung der unterschiedlichen Kosten für die Einzeloperationen. Bei parallelen Rechnungen ist eine solche Einzelstatistik wenig aussagekräftig, hier kann man nur die Echtzeit (‚*wallclock time*‘) als Bewertungsgrundlage heranziehen, um Wartezeiten, die durch schlechte Synchronisation hervorgerufen werden, zu bestrafen.

- Die Auswahl der Testbeispiele mag einen Integrator begünstigen. Gerade weil bestimmte Testaufgaben sehr häufig zum Vergleich benutzt werden, sind die Steuerstrategien in manchen Implementierungen auf diese Probleme speziell angepaßt. Außerdem können zwei verschiedene Integratoren, auch bei recht ähnlichen Testproblemen, sehr unterschiedlich abschneiden. Einen „Universalintegrator“, der stets die besten Ergebnisse liefert, gibt es nicht.

Kaum betroffen von diesen Einwänden ist der Vergleich der PTSW-Verfahren untereinander. Wir hoffen aber auch, durch die getroffene Auswahl der Referenzverfahren und der Beispiele eine realistische Einschätzung der Effizienz der PTSW-Verfahren für typische Anwendungen zu erhalten.

Beispiel 5.1. Wir lösen das Testproblem MEDAKZO sequentiell auf vier verschiedenen Computern mit RADAU, VODE und PTSW3A (Beschreibung siehe unten). Hinsichtlich der erreichten Genauigkeit, der dafür benötigten Schritte, Jacobimatrizen und f-Auswertungen sind die Unterschiede zwischen den Plattformen minimal. Alle Integratoren sind in FORTRAN-77 geschrieben und verwenden die gleichen LAPACK-Routinen zur Lösung der linearen Gleichungssysteme. Trotzdem sind die Rechenzeiten sehr verschieden, Abbildung 5.1. Besonders deutlich wird es, wenn man RADAU und VODE auf USparc-II und PA8000 vergleicht: auf dem SUN-Rechner benötigt VODE 55 % der Rechenzeit von RADAU und auf der HP/Convex-Maschine 143 %, um die Genauigkeit 10^{-4} zu erreichen. Interessant ist auch, daß der Pentium II konkurrenzfähig zur USparc-II ist (deren *peak performance* mit 900 *mflops* nach Herstellerangaben fast dreifach höher liegt).
Die folgenden Untersuchungen beziehen sich auf die HP/Convex X-Class mit PA8000 Prozessoren.

Bemerkung 5.1 (Tuning). In der Arbeit von Bendtsen [5] wird behauptet, daß der parallele Code ParSODE, basierend auf MIRK-Verfahren, deutlich effizienter als VODE und RADAU sei. Speedups von drei bis fünf werden erzielt. Bendtsen verwendet Testprobleme vom CWI-Testset [23] und vergrößert die Systeme künstlich. Die aufgeblähten Systeme haben sehr große, dicht besetzte Jacobimatrizen und der LU-Zerlegungsaufwand dominiert die Integration. Weil die MIRK-Verfahren die Jacobimatrix sehr selten berechnen, schneiden sie exzellent ab. Vergleicht man RADAU, VODE und ParSODE anhand der unmodifizierten, kleinen Differentialgleichungssysteme, so ist ParSODE nicht annäherend konkurrenzfähig (dies

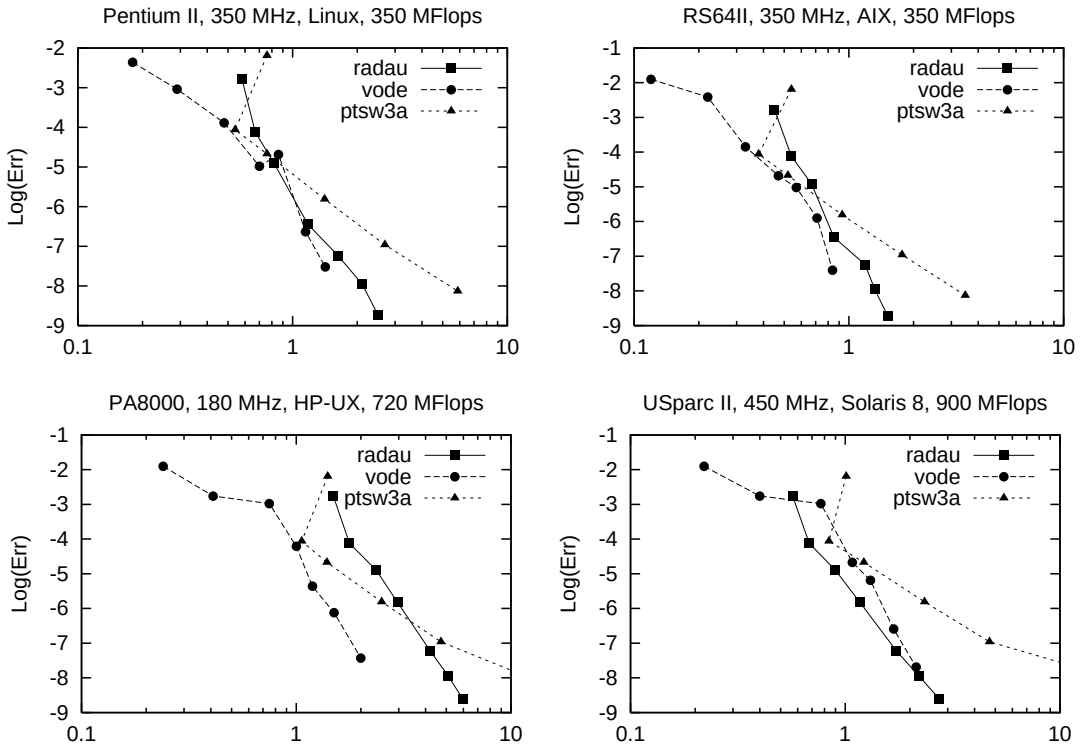

Abbildung 5.1: Gleiche Rechnung auf unterschiedlicher Hardware für Beispiel MEDAKZO, Diskussion in Beispiel 5.1.

muß man selbst tun, der Vergleich fehlt in der Arbeit von Bendtsen [5]). Wir erwähnen dieses Beispiel nicht, um vor ParSODE zu warnen, sondern um deutlich zu machen, daß strategische Entscheidungen im Integrator, z.B. wie oft die Jacobimatrix berechnet oder zerlegt wird, sehr kritisch für die Performance sind und eigentlich nur mit Kenntnis der (dimensions- und hardwareabhängigen) Kosten optimal getroffen werden können. Es ergibt sich daher die interessante Frage, ob ein automatisches Tuning, also das Anpassen der Steuerstrategien eines Integrators an eine bestimmte Problemklasse, nicht eine wesentliche Effizienzverbesserung ermöglicht, vielleicht sogar in ganz ähnlicher Weise, wie bei ATLAS (http://www.netlib.org/atlas), dem Paket zur Erzeugung von optimierten BLAS-Routinen für Operationen der lineare Algebra. Ein manuelles Tuning wird von vielen Codes bereits unterstützt, so gibt es in RADAU die Möglichkeit einen Kostenfaktor `work(3)` für die Berechnung der Jacobimatrix anzugeben, der bei der Steuerung berücksichtigt wird. ◊

Bemerkung 5.2 (Testprobleme für parallele Codes). Beim Design der Testsets, z.B. des CWI-Testsets [23], werden möglichst einfache Beispiele bevorzugt, die typische Schwierigkeiten für Integrationsverfahren enthalten. Die Differentialgleichungen bleiben überschaubar und sind in manchen Fällen sogar noch analytisch lösbar. Dadurch ist die Auswertung der rechten Seite wenig rechenaufwendig. Legt man derartige Testprobleme bei der Abschätzung des praktisch möglichen Speedups zugrunde, so ist das zu pessimistisch, denn gerade die Komplexität größerer, realitätsnäherer Probleme läßt bessere Speedups erwarten. Rechenaufwendige Testprobleme sind im Report von Bellen [4] speziell für parallele Tests (aus eher „akademischen Beispielen“) konstruiert worden. Auch FEKETE [23] wurde speziell als „paralleles Testproblem“ entworfen. Eine interessante Klasse von Problemen, die sich auch gut als Benchmarks für parallele Krylov-Integratoren eignen würden, sind Reaktions-Diffusionsgleichungen mit komplizierten Reaktionstermen, wie sie z.B. auch

bei der Modellierung der Kalziumdynamik innerhalb einer Herzzelle eingesetzt werden, z.B. [56, 55]. Die Zusammenstellung von typischen Problemen mit einer interessanten Dynamik (z.B. wandernden Wellen in den Lösungen) zu einem Testset wäre eine aufwendige, aber für die Bewertung der Integratoren sehr nützliche Arbeitsaufgabe. ♢

5.2 Die Testumgebung

Für die numerischen Tests benutzen wir einen X-Class Server SPP2000 von HP/Convex unter HP-UX 10.01. Die Maschine besitzt insgesamt 32 Prozessoren, von denen wir bei parallelen Rechnungen bis zu 4 Prozessoren exklusiv nutzen. Die Rechenzeit messen wir über die Bibliotheksfunktion `dtime` bzw. im Parallelbetrieb über die Fortran-90-Intrinsic Funktion `date_and_time`, wobei Schwankungen von etwa 3 % durch den Mehrbenutzerbetrieb auftreten können. Die erreichte Genauigkeit der numerischen Lösung $\tilde{u}$ im Endpunkt t_e der Integration geben wir in einer gewichteten Norm an

$$Err = \sqrt{\frac{1}{n}\sum_{i=1}^{s}\left(\frac{\tilde{u}_i - u_{\mathrm{ref},i}}{1+|u_{\mathrm{ref},i}|}\right)^2}. \tag{5.1}$$

Die Referenzlösung u_{ref}l haben wir mit RADAU bzw. VODPK bei Vorgabe strenger Toleranzen (im allgemeinen `ATOL`=`RTOL`=10^{-12}) berechnet. Wir verwenden den FORTRAN-77 Compiler `fort77` von Hewlett-Packard mit Optimierung (`+DS2.0a +DA2.0N +O3 +Oparallel +Onoautopar +U77`) und die herstelleroptimierten BLAS- und LAPACK-Routinen.
Als Referenzverfahren wählen wir für die Beispiele mit LU-Zerlegung die Fortran-Codes

- RADAU – implizite Runge-Kutta-Verfahren implementiert von Hairer und Wanner, mit Ordnungssteuerungen $p = 1, \ldots, 13$, Version vom 6. Mai 1999, [31],
- RODAS – Rosenbrock-Methode der Ordnung 4 implementiert von Hairer und Wanner, Version vom 28. Oktober 1996, [30],

mit den Standardeinstellungen. Für die Rechnungen mit Krylovapproximation vergleichen wir mit

- ROWMAP – 4-stufige Krylov-W-Methode mit mehrfachem Arnoldiprozeß, Version vom 10. Januar 1997, Weiner u.a. [53],
- VODPK – BDF-Verfahren mit Ordnungssteuerung und GMRES(5) zur Lösung der linearen Gleichungssysteme, Version vom 15. Mai 1997, Byrne u.a. [17],

auch mit Standardeinstellungen.

5.3 Vergleich mit RODAS und RADAU

Wir betrachten Testbeispiele von moderater Größe, für die relativ günstig eine Jacobimatrix berechnet und zerlegt werden kann:

- PLATE – nichtautonomes Problem mit $n = 80$ und voller Jacobimatrix, [30],
- MEDAKZO – 1D-Reaktions-Diffusionssystem, $n = 400$, 5-Band-Jacobimatrix, [23],
- CUSP – 1D-Reaktions-Diffusionssystem, $n = 96$, mit periodischen Randbedingungen, zyklische Bandstruktur der Jacobimatrix, [30],

- FEKETE – Fekete-Punkte auf einer Kugel, $n = 160$, DAE vom Index 2, [23].

Eine Parallelisierung ergibt nur beim letzten Problem, FEKETE, eine erhebliche Verbesserung. Bei den anderen Beispielen überwiegt der Zusatzaufwand zur Synchronisierung den Vorteil der Parallelisierung vollständig. Deshalb geben wir auch nur bei FEKETE parallele Rechenergebnisse und Speedups an.

Ergebnisse für PLATE

Wir vergleichen zunächst die PTSW-Methoden gleicher Stufenzahl untereinander, Abbildung 5.2 oben. Für $s = 2$ ist PTSW2A am effizientesten. PTSW2C ist für grobe und PTSW2B für strenge Toleranzen deutlich langsamer.
Alle drei 3-stufigen Verfahren zeigen ein glattes Lösungsverhalten. Die Effizienzunterschiede sind nicht groß, es gibt einen leichten Vorteil für die A-Variante. Bei kleineren Toleranzen zwischen 10^{-5} und 10^{-8} fällt PTSW3B etwas zurück.
Die 4-stufigen Verfahren haben Schwierigkeiten mit PLATE. Verfahren PTSW4B versagt fast völlig. Ursache ist die hohe Empfindlichkeit bei variablen Schrittweiten, denn für konstantes $h_m = h$ ist das gleiche Verfahren (mit Label PTSW4B-FIX) exzellent.
Verglichen mit RADAU ist nur PTSW2A für grobe Toleranzen konkurrenzfähig. Für kleine Toleranzen sind PTSW3A und PTSW2A ähnlich gut geeignet wie RODAS.

Ergebnisse für MEDKAZO

Auch bei diesem Beispiel sind PTSW2A und PTSW3A unter den betrachteten $2-$ und 3-stufigen Verfahren wieder am besten, Abbildung 5.3. Für $s = 2$ ist der Abstand deutlich, für $s = 3$ verhält sich PTSW3A fast identisch zu PTSW3C. Bei den vierstufigen Verfahren ist PTSW4B, besonders für kleine Toleranzen, gut geeignet. Für grobe Genauigkeiten gibt es „Ausreißer“: PTSW2A, PTSW3A, PTSW4A und PTSW4C sind mit $\mathtt{ATOL} = \mathtt{RTOL} = 10^{-3}$ schneller und genauer als mit $\mathtt{ATOL} = \mathtt{RTOL} = 10^{-2}$. Kein PTSW-Verfahren versagt bei MEDAKZO, alle erreichen bei entsprechender Toleranz auch hohe Genauigkeiten.
RODAS wird nicht erreicht, aber für geringe bis mittlere Toleranzen liegen PTSW2A und PTSW3A zwischen RADAU und RODAS.

Ergebnisse für CUSP

CUSP entsteht durch Diskretisierung einer Reaktions-Diffusionsgleichung mit periodischen Randbedingungen. Durch Weglassen eines Blocks kann die Jacobimatrix gut durch eine siebenbändige Matrix approximiert werden, siehe [30]. Diese (leicht invertierbare) Approximation verwenden wir bei den folgenden Rechnungen für die PTSW-Methoden und für RADAU. Für RODAS jedoch benötigen wir für die Konvergenzordnung 4 die ‚richtige‘ 96×96-Jacobimatrix f_y und damit ist $(I - h_m\gamma f_y)^{-1}$ vollbesetzt, d.h., die LU-Zerlegung ist deutlich aufwendiger.
PTSW2A und PTSW2C sind für CUSP deutlich genauer als PTSW2B, Abbildung 5.4. Die dreistufigen Verfahren sind fast gleichauf, nur bei kleineren Toleranzen ist PTSW3B weniger genau als PTSW3A und PTSW3C. Die PTSW-Verfahren mit $s = 4$ haben Schwierigkeiten bei groben und mittleren Toleranzen, etwa bis $\mathtt{ATOL} = \mathtt{RTOL} = 10^{-4}$, bei schärferen Genauigkeitsanforderungen lösen sie das Problem zuverlässig, aber nicht so effizient wie die dreistufigen. Ursache für die Empfindlichkeit sind vermutlich wieder die Schrittweitenwechsel. Man kann dies jedoch kaum verifizieren: eine Vergleichsrechnung mit konstanter (moderat groß gewählter) Schrittweite $h_m = h$ läßt die Dynamik des Systems nicht zu, man benötigt die Schrittweitensteuerung.

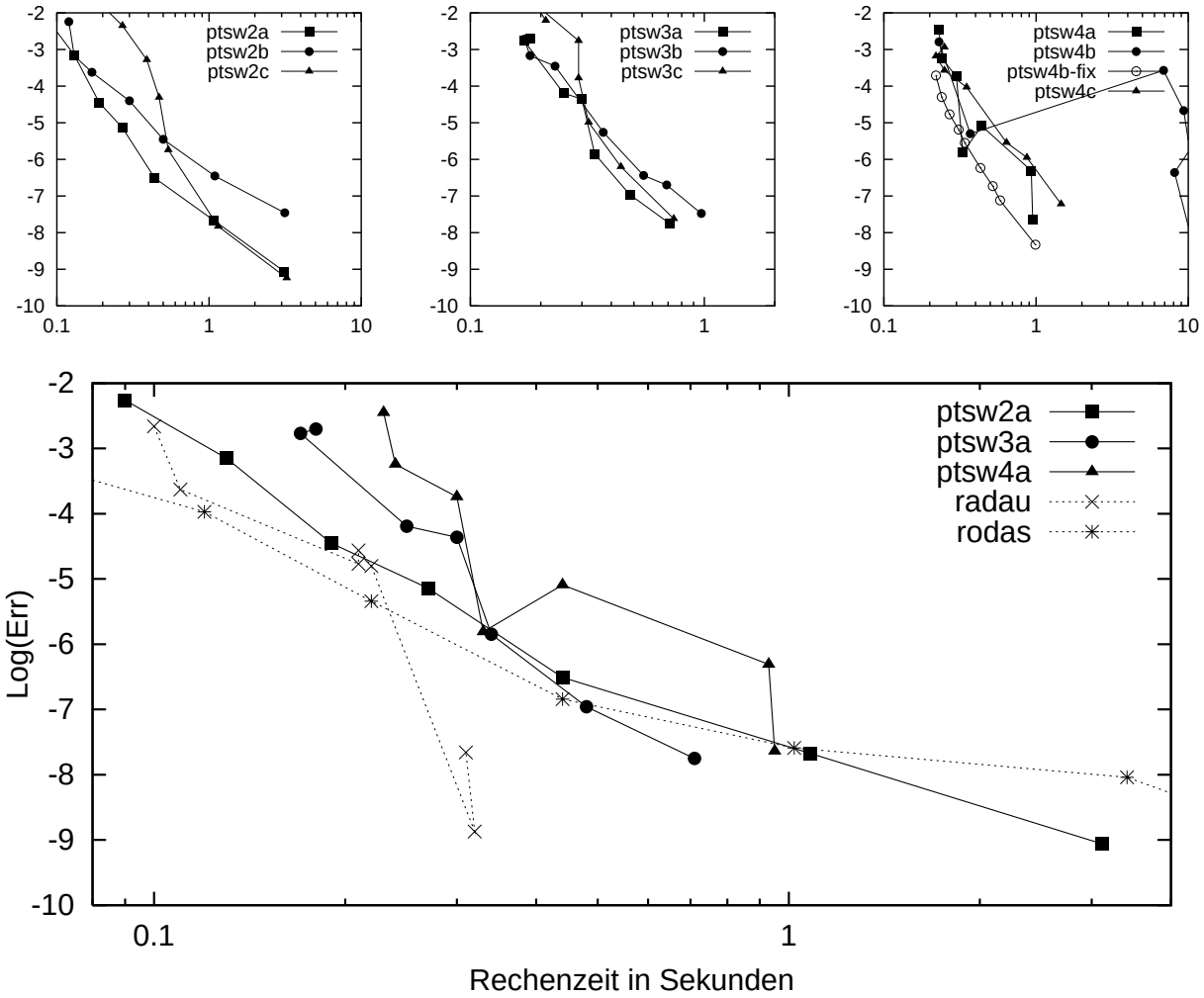

Abbildung 5.2: Ergebnisse für PLATE.

RODAS erreicht mit relativ wenigen Schritten eine hohe Genauigkeit, ist aber wegen der aufwendigeren linearen Algebra nicht konkurrenzfähig zu RADAU. Die Verfahren PTSW2A und PTSW3A sind bis zur Genauigkeit 10^{-7} signifikant schneller als RADAU.
Dieses Beispiel zeigt deutlich den Vorteil von W-Methoden gegenüber Rosenbrock-Methoden, wenn eine gute und einfach zu invertierende Approximation an $(I - h_m \gamma f_y)$ zur Verfügung steht.

Ergebnisse für FEKETE

Mit Hilfe des differential-algebraischen Systems FEKETE vom Index 2 können wir zwei Fragen nachgehen und letztendlich positiv beantworten: konvergieren PTSW-Verfahren bei der Lösung von DAEs vom Index 2 numerisch, wie man aufgrund der hohen Stufenordnung eventuell vermuten könnte und lassen sich auch bei kleineren Beispielen mit LU-Zerlegung Speedups erzielen, wenn die rechte Seite der Differentialgleichung etwas ‚teurer' ist?
Wir betrachten hier nur den (gewichteten) Fehler der differentiellen Komponenten (1 bis 120), Abbildung 5.5. Bei den zweistufigen PTSW-Verfahren hat PTSW2C, im Gegensatz zu PTSW2A und PTSW2B, anscheinend Stabilitätsprobleme. Die naheliegende Vermutung, daß dies eine Folge der fehlenden Steifgenauigkeit von PTSW2C ist, wird leider beim Blick auf $s = 3$ nicht verstärkt: auch PTSW3C fehlt diese

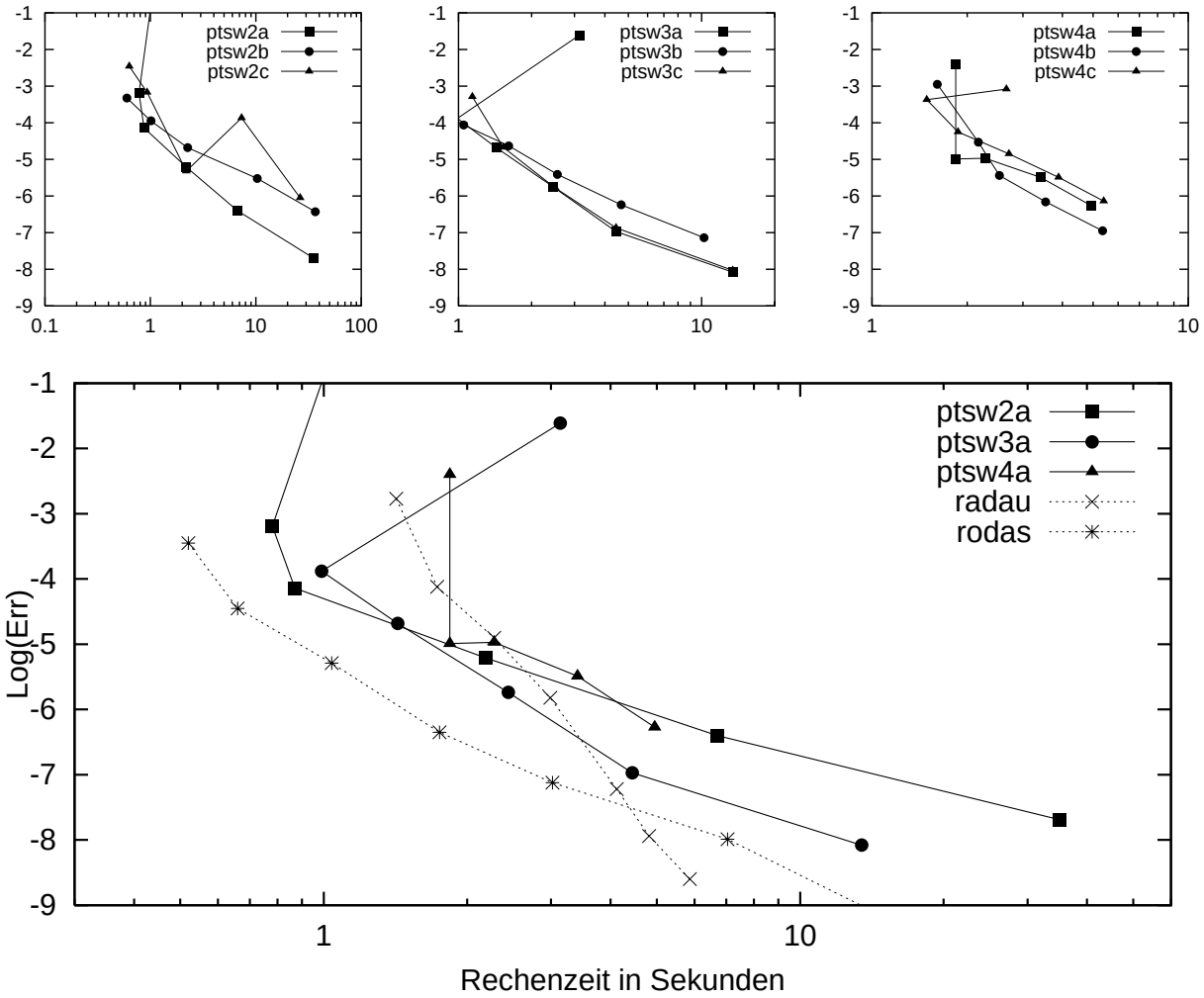

Abbildung 5.3: Ergebnisse für MEDAKZO.

Eigenschaft und es integriert FEKETE ohne Schwierigkeiten, wenn auch nicht so effizient wie PTSW3A. Von den vierstufigen Verfahren lösen nur PTSW4A und PTSW4C das Problem akzeptabel. Man könnte auch mutmaßen, daß die Größe des A(α)-Winkels auschlaggebend ist, nur fehlt bei der Beschreibung von FEKETE [23] leider eine Aussage zur Lage der Eigenwerte der Jacobimatrix.
Im Vergleich mit RADAU sehen wir, daß PTSW2A, PTSW3A und PTSW4C bereits sequentiell konkurrenzfähig sind. Bei Parallelisierung sind die PTSW-Methoden durch Speedups von ca. 1.5 für PTSW2A, 1.7 bis 1.9 für PTSW3A und 1.9 bis 2.2 für PTSW4C deutlich schneller als das sequentielle RADAU. Durch die logarithmische Achseneinteilung sieht man gut, daß der Speedup wenig von der Toleranz abhängt, die Linienzüge für parallele und sequentielle Ausführung liegen in beinahe konstantem Abstand.

5.4 Vergleich mit ROWMAP und VODPK

Leider gibt es, abgesehen vom zweidimensionalen Brusselator, kaum anerkannte Testaufgaben, die sich für Zeitintegratoren mit Krylovapproximation eignen. Im IVP-Testset [23] findet sich kein einziges Problem. Wir vergleichen daher die PTSW-Verfahren anhand von Diffusionsgleichungen, die aus vorgegebenen ‚exakten Lösungen' $u(t, x, y)$ durch Differentiation geeignet konstruiert wurden. Für die Ortsdiskretisierung verwenden wir für die Diffusionsterme zentrale Differenzenquotienten 2. Ordnung. Als Referenzlösung be-

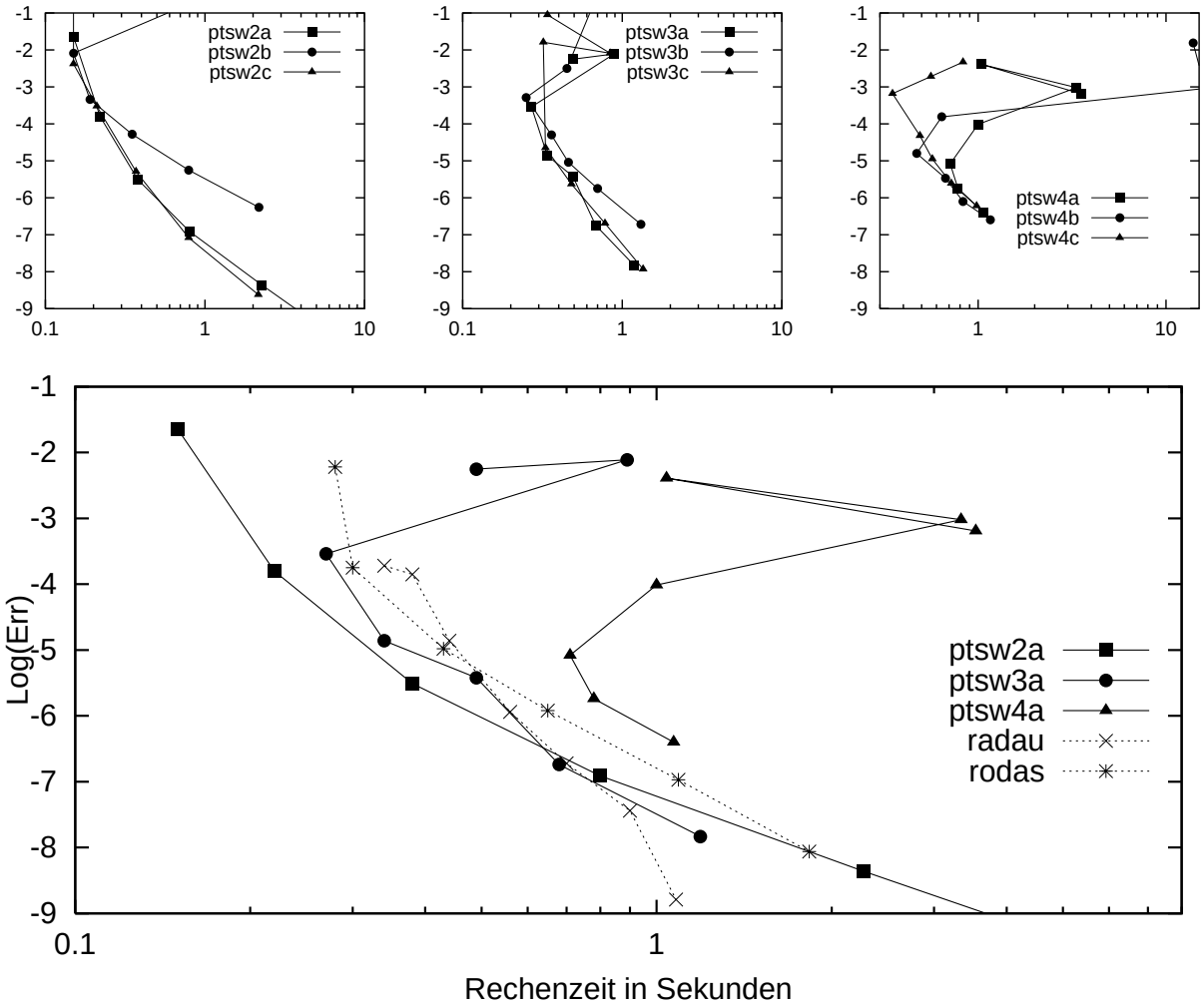

Abbildung 5.4: Ergebnisse für CUSP.

nutzen wir eine genaue numerische Lösung des semidiskreten Systems.
Wir betrachten folgende Beispiele:

- NILIDI – Nichtlineare Diffusionsgleichung, [16],

$$u_t = e^u \Delta u + u(18e^u - 1), \quad u(t,x,y) = e^{-t}\sin(3x)\sin(3y), \quad t \in [0,1], \Omega = [0,\pi/3]^2$$

  mit Dirichlet-Randbedingungen. Wir diskretisieren mit $n = 100 \times 100 = 10000$ Gitterpunkten.

- DIFFU2 – Nichtautonome Diffusionsgleichung, [53]

$$u_t = \Delta u + g(t,x,y), \quad \Omega = [0,1]^2, \quad u(t,x,y) = \sin(\pi x)\sin(\pi y)(1 + 4xy\sin t),$$

  wieder mit Dirichlet-Randbedingungen und $n = 100 \times 100 = 10000$ Gitterpunkte. Die Quellterme $g(t,x,y)$ in der rechten Seite werden geeignet aus $u(t,x,y)$ bestimmt und sind durch das Auftreten von Sinus- und Kosinusfunktionen etwas aufwendiger zu berechnen, als z.B. die von NILIDI.

- BRUSSELATOR – Reaktions-Diffusionsgleichung, z.B. in [33],

$$u_t = 1 + u^2 v - 4u + \alpha\Delta u, \quad v_t = 3u - u^2 v + \alpha\Delta u, \quad \alpha = 0.2, \quad \Omega = [0,1]^2$$

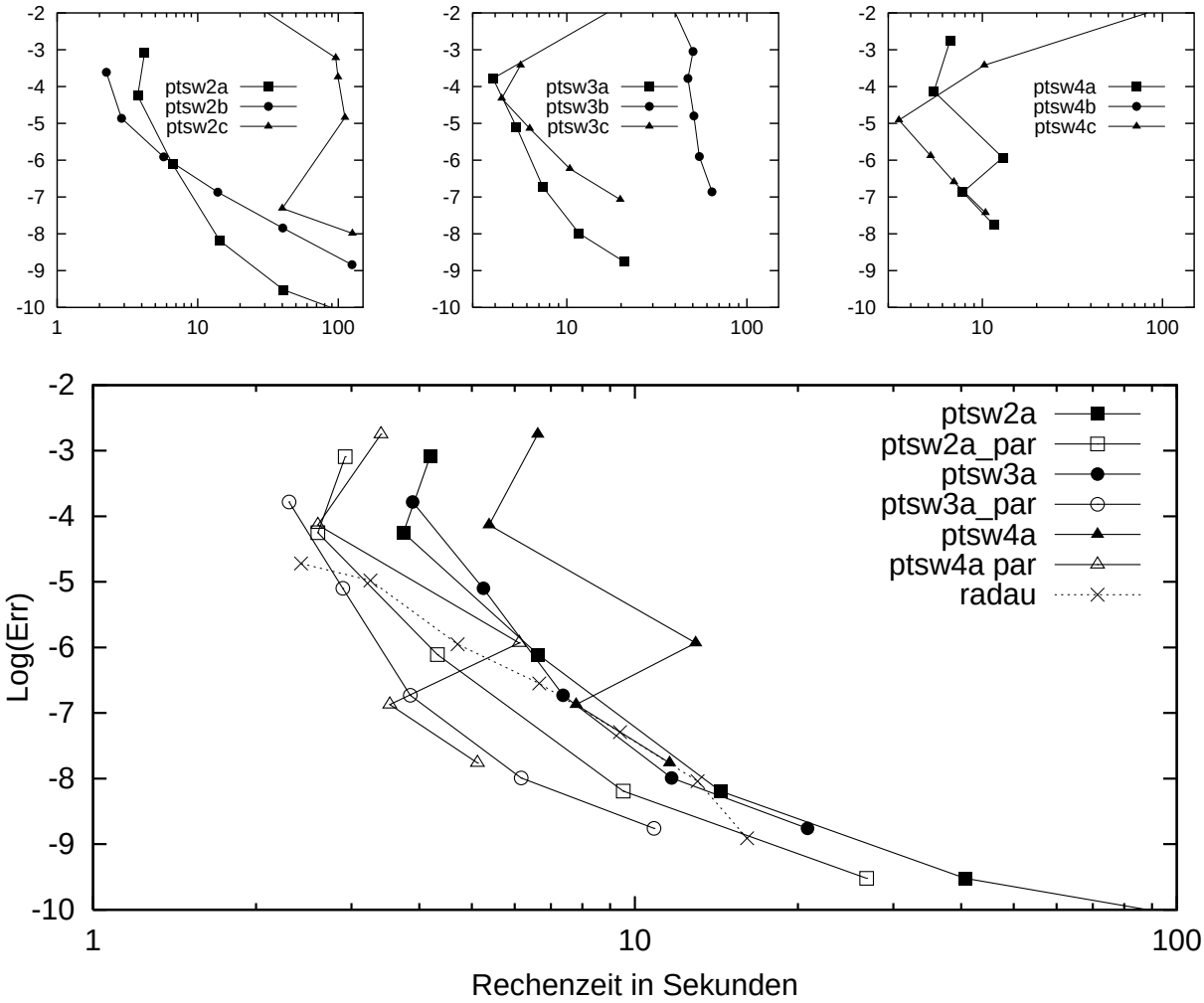

Abbildung 5.5: Ergebnisse für FEKETE.

mit Anfangswerten $u(0,x,y) = 0.5 + y$, $v(0,x,y) = 1 + 5x$, $n = 2 \times 100 \times 100 = 20000$ und homogenen Neumannbedingungen. Durch die Wahl der Diffusionskonstanten $\alpha = 0.2$ wird das System steif.

Ergebnisse für NILIDI

Die Ergebnisse für die PTSW-Verfahren mit Krylovapproximation unterscheiden sich deutlich von denen mit LU-Zerlegung. Bei den kleinen Beispielen waren die Verfahren PTSW2A, PTSW3A und PTSW4A besonders gut geeignet, jetzt sind sie, besonders für $s = 3$ und $s = 4$, vergleichsweise ungünstig, Abbildung 5.6 oben. So benötigen PTSW3A und PTSW3B für die gleiche Genauigkeit in etwa gleich viele Schritte, jedoch ist PTSW3A mit einer deutlich höher liegenden durchschnittlichen Krylovdimension wesentlich langsamer. Die Ursache für diese Beobachtung ist der große Spektralradius $\varrho(M(\infty)) \doteq 0.917$ und die daraus resultierende schlechte Fehler-Dämpfung von PTSW3A.
Besonders effizient sind die nilpotenten Verfahren PTSW2B und PTSW3B, sie sind bereits bei sequentieller Rechnung günstiger als VODPK. ROWMAP ist für dieses Beispiel sehr gut geeignet und wird nicht erreicht. Die PTSW-Verfahren werden bei NILIDI durch die Parallelisierung deutlich beschleunigt. Dabei variiert der Speedup eng korreliert mit der schwankenden (von der Toleranz nichtmonoton abhängenden) Krylov-

dimension. Zur Illustration verweisen wir auf Abbildung 5.7. Man sieht, wie bei Problemvergrößerung um Faktor 2 (durch eine Ortsdiskretisierung mit $n = 140 \times 140$ Gitterpunkten) auch der Speedup zunimmt. Als Maximalwerte erhalten wir hohe Speedups von 1.7, 2.5 und 3.4 für zwei, drei und vier Prozessoren.

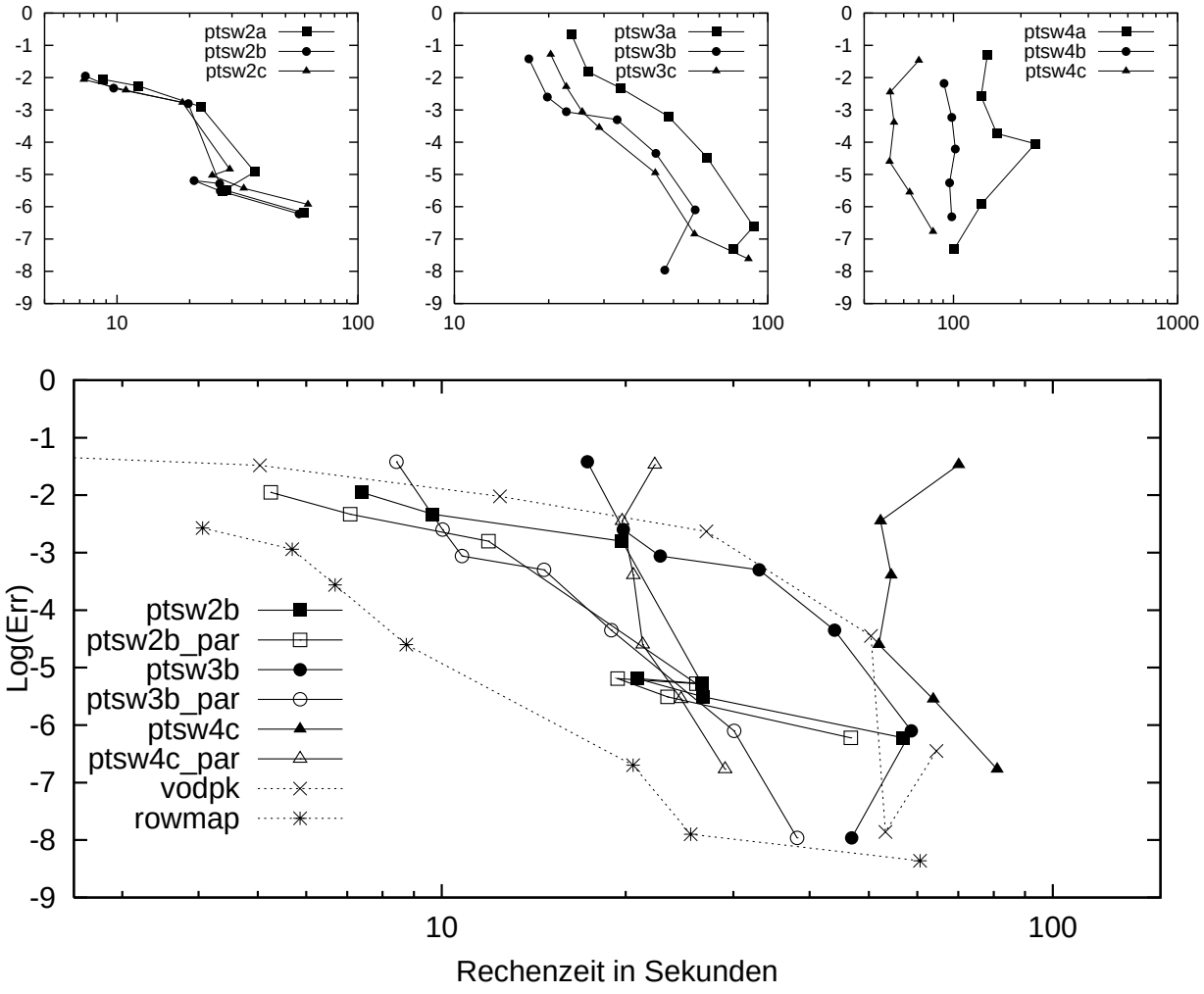

Abbildung 5.6: Ergebnisse für NILIDI.

Ergebnisse für DIFFU2

Für DIFFU2 sind die Ergebnisse ähnlich wie bei NILIDI: von den zweistufigen PTSW-Verfahren ist das nilpotente PTSW2B, bei den dreistufigen PTSW3B und bei den vierstufigen PTSW4C am effizientesten, Abbildung 5.8. Bei sequentieller Rechnung ist PTSW2B für grobe und PTSW3B für kleine Toleranzen am günstigsten. Die Vergleichsverfahren ROWMAP und VODPK sind gegenüber PTSW2B und PTSW3B, auch bei sequentieller Rechnung, nicht konkurrenzfähig. Der Abstand ist, auch wenn man verschiedene Genauigkeiten betrachtet, beachtlich und zeigt das Potential von Verfahren der PTSW-Klasse bei der numerischen Lösung derartiger Probleme.
Obwohl die rechte Seite der Differentialgleichung relativ aufwendig zu berechnen ist, sind die Speedups für DIFFU2 wegen der niedrigeren Krylovdimensionen kleiner als bei NILIDI (ca. 1.5, 2.0 und 2.5 für $s = 2, 3$ und 4). Nach Parallelisierung ist PTSW3B das beste Verfahren, PTSW2B und PTSW4C liegen nur für sehr grobe bzw. sehr feine Toleranzen in einem ähnlichen Bereich.

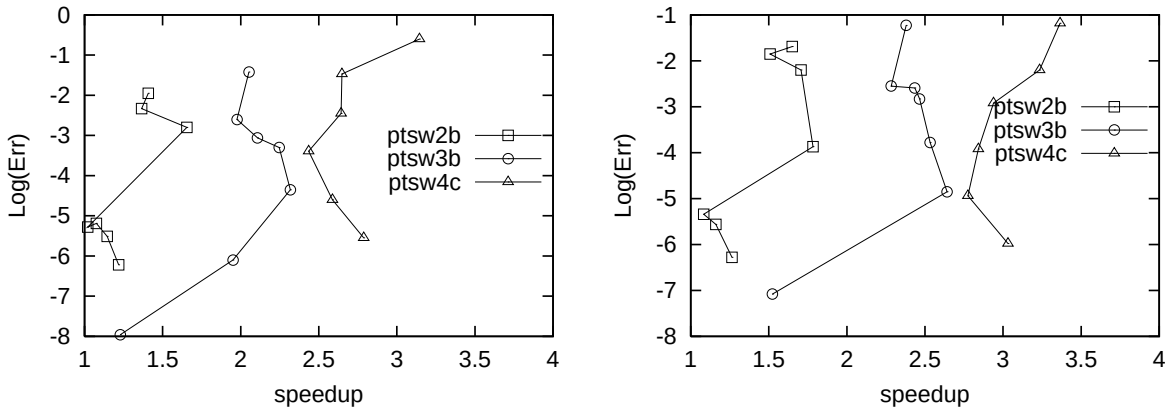

Abbildung 5.7: Speedup für NILIDI. Er steigt mit Problemvergrößerung, links $n = 10000$, rechts $n = 19600$.

Ergebnisse für BRUSSELATOR

Die Unterschiede zwischen den zweistufigen Verfahren sind gering, PTSW2C ist bei groben und PTSW2A bei kleinen Toleranzen führend, Abbildung 5.9. Für $s = 3$ liegt PTSW3B mit Abstand vor PTSW3A und PTSW3C. Leider ist PTSW4C, das beste vierstufige Verfahren für NILIDI und DIFFU2, beim BRUSSELATOR langsamer als PTSW4B und PTSW4A.
Die Speedups bei einer Parallelisierung sind durch die besonders einfache rechte Seite der Differentialgleichung und durch die niedrigen Krylovdimensionen trotz $n = 20000$ etwas kleiner als bei NILIDI: PTSW2B (1.3 .. 1.6), PTSW3B (1.7 .. 2) und PTSW4B (2.1 .. 2.5) .
Wir diskutieren zunächst die sequentiellen Ergebnisse. PTSW2B, ROWMAP und VODPK sind bis zur Genauigkeit 10^{-6} etwa gleich schnell. Für sehr kleine Toleranzen ist VODPK mit knappem Vorsprung vor PTSW3B am effizientesten. Durch Parallelisierung liegt PTSW4B für alle Genauigkeiten deutlich vor PTSW3B, und dieses vor VODPK. PTSW2B-PAR ist bis 10^{-4} mit dem dreistufigen Verfahren PTSW3B-PAR vergleichbar.

5.5 Diskussion und Konsequenzen der numerischen Experimente

Unsere numerischen Tests zeigen deutlich, daß steifgenaue L(α)-stabile PTSW-Verfahren den A(α)-stabilen Verfahren mit $v^T = 0$ überlegen sind, obwohl letztere für variable Schrittweiten eine höhere Ordnung besitzen. Dieses Ergebnis erhalten wir unabhängig davon, ob wir LU-Zerlegung oder Krylovapproximation zur Lösung der Stufengleichungen verwenden (z.B. bei PLATE, FEKTE und DIFFU2). PTSW-Verfahren, die auf steife Probleme angewendet werden, sollten daher stets die Steifgenauigkeitsbedingung (2.22) erfüllen.
Die zwei- und dreistufigen PTSW-Verfahren sind robuster als die vierstufigen, besonders bei groben Toleranzvorgaben. Ursachen für die Empfindlichkeit der vierstufigen Verfahren sind zum einen die ungünstigeren (betragsgroßen) Verfahrensparameter und zum anderen die starken Abhängigkeiten von den Schrittweitenverhältnissen σ_m bei der Formulierung für variable Zeitschritte h_m. Um effiziente PTSW-Verfahren mit $s > 3$ konstruieren zu können, müssen wir daher auf die hohe Stufenordnung s verzichten.
Für Rechnungen mit LU-Zerlegung sind die optimierten Verfahren PTSW2A und PTSW3A gut geeignet. Besonders bei geringen und mittleren Genauigkeitsanforderungen sind sie bereits sequentiell konkurrenzfähig zu RADAU. Bemerkenswert ist die Konvergenz für das Index-2 Problem FEKETE. Bei den Testbeispielen dieser Arbeit zahlt sich eine Parallelisierung auf der verwendeten Hardware nicht aus, weil die

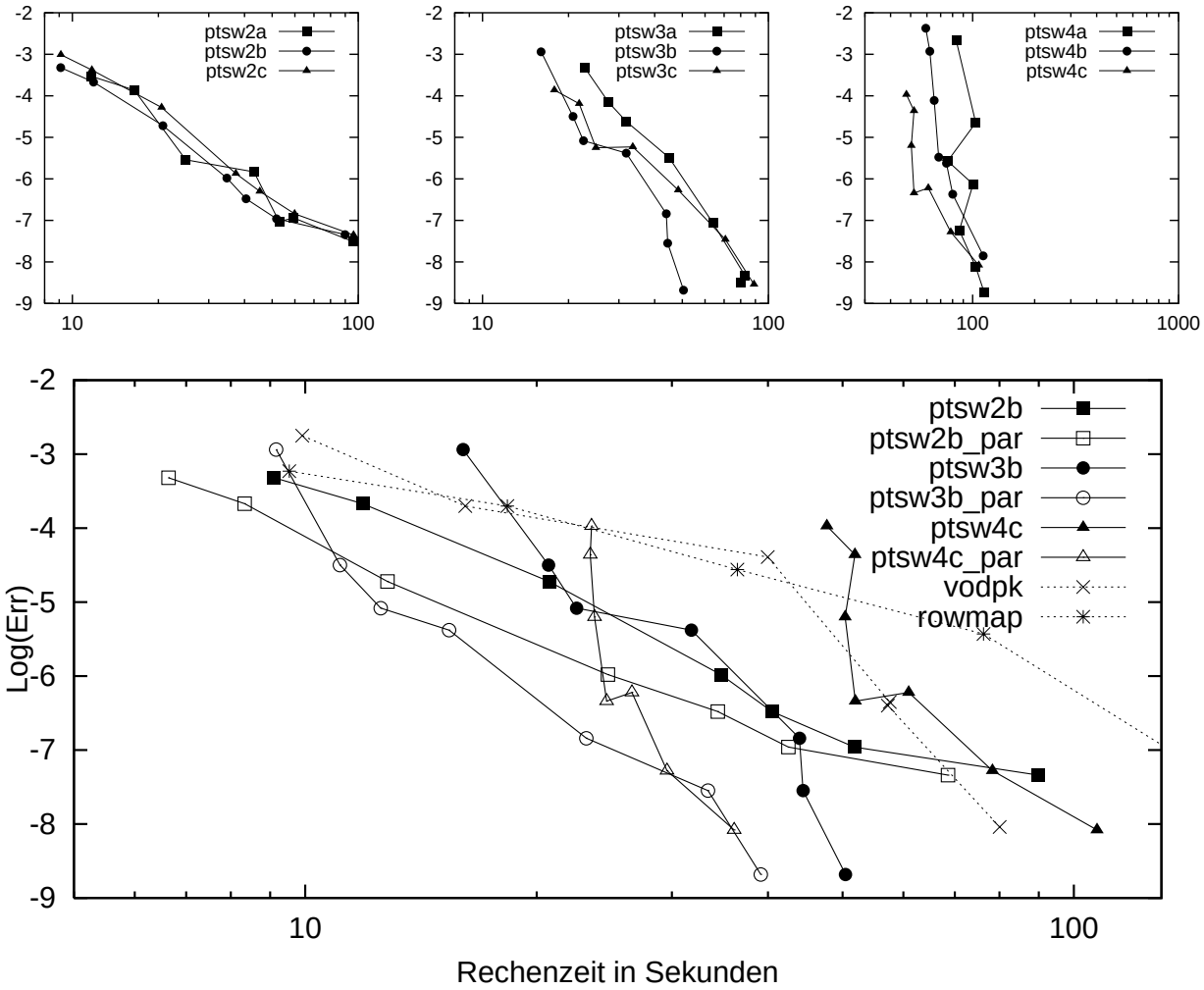

Abbildung 5.8: Ergebnisse für DIFFU2.

LU-Zerlegung für diese Dimensionen nicht skaliert und somit einen sequentiellen Flaschenhals darstellt. Bei Rechnung mit Krylov-Approximation wird die Krylovdimension pro Stufe wesentlich durch den Spektralradius der Stabilitätsmatrix bestimmt: bei den Verfahren PTSW2B und PTSW3B mit $\varrho(M(\infty)) = 0$ wird die vorgegebene Toleranz für das Residuum der linearen Gleichungssysteme mit deutlich niedrigerem Aufwand als bei PTSW2A oder PTSW3A erreicht. Das beste PTSW-Verfahren bei den Testbeispielen mit Krylovlösung ist PTSW3B. Es ist bei NILIDI und bei BRUSSELATOR bereits sequentiell mit dem Mehrschrittcode VODPK vergleichbar und bei DIFFU2 sogar deutlich effizienter. Mit drei Prozessoren erreichen wir mit PTSW3B, abhängig vom Beispiel, Speedups zwischen 1.5 und 2.5. Man beachte, daß wir bei niedrigen Krylovdimensionen nur einen mittleren Speedup erwarten dürfen, weil wir in jedem Schritt zunächst einen sequentiellen Funktionsaufruf (vgl. Gleichung (4.6)) benötigen. Hinzu kommt die sequentielle Startprozedur und die unterschiedlich hohe Krylovdimension in den parallelen Stufen. Die parallelisierten Verfahren PTSW2B-PAR und PTSW3B-PAR sind mit Krylovapproximation für die Probleme NILIDI, BRUSSELATOR und DIFFU2 für alle Genauigkeiten effizienter als das sequentielle VODPK.

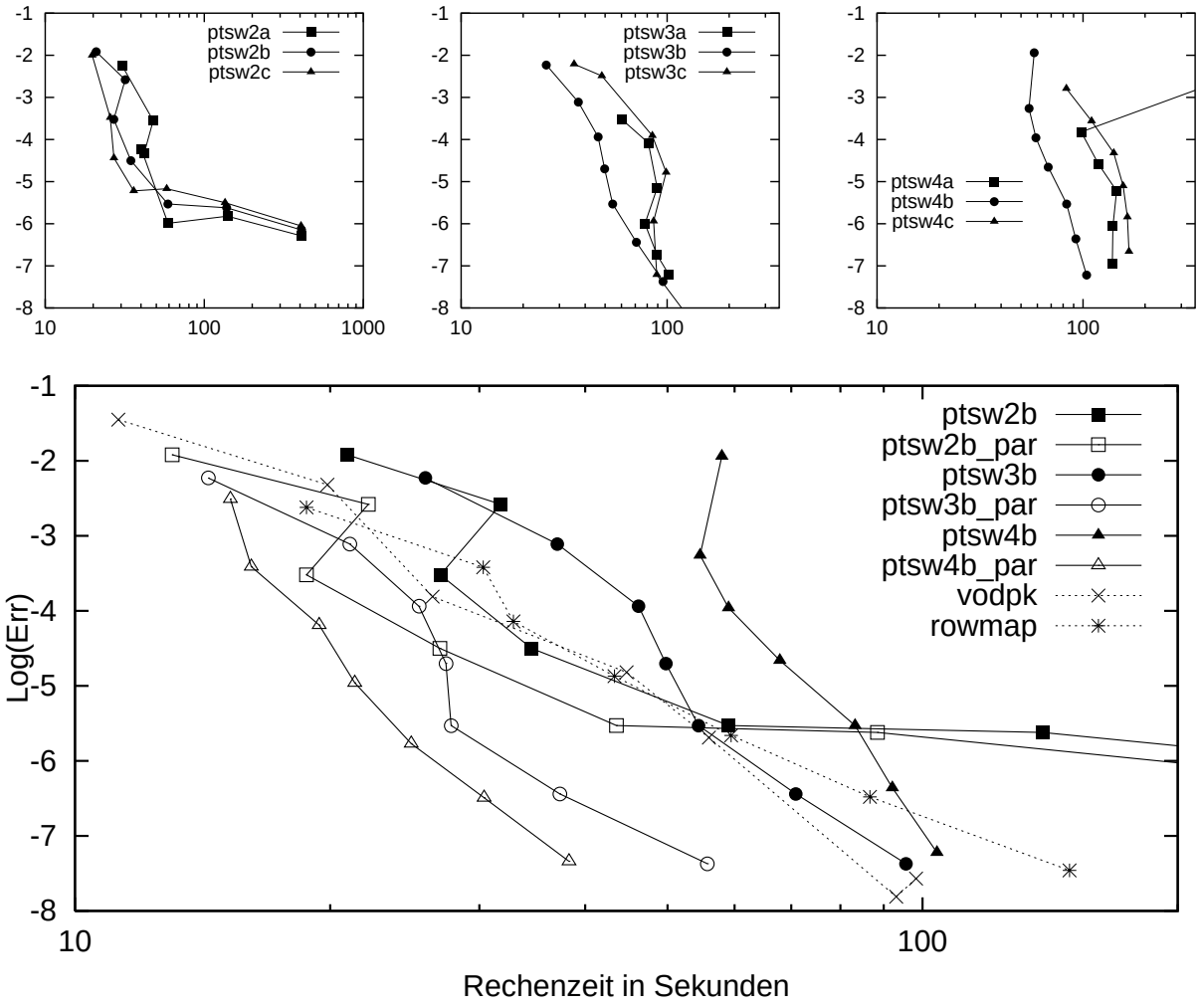

Abbildung 5.9: Ergebnisse für BRUSSELATOR.

6 Zusammenfassung und weiterführende Bemerkungen

In dieser Arbeit haben wir parallele Zweischritt-W-Methoden (PTSW-Methoden) zur numerischen Lösung gewöhnlicher Differentialgleichungssysteme entworfen und untersucht. Der zugrundeliegende Ansatz, eine W-Methode mit externen Stufen zu betrachten, wurde wesentlich durch die expliziten Zweischritt-Runge-Kutta-Verfahren motiviert und führte uns auf ein s-stufiges linear-implizites Schema, in dem die Stufen unabhängig voneinander berechnet werden können.
Die Konvergenz der PTSW-Verfahren mit Ordnung $p^* = \min(p, q+1)$ ließ sich mit Hilfe von vereinfachenden Konsistenzbedingungen $\mathrm{C}(q)$, $\Gamma(q)$ und $\mathrm{B}(p)$ zeigen, die wir aus einer asymptotischen Entwicklung der lokalen Fehler ableiten konnten. Durch den Zweischritt-Charakter waren im Gegensatz zur Situation bei sequentiellen W-Methoden hohe Stufenordnungen erreichbar.
Durch die Anwendung der PTSW-Verfahren auf die Testgleichung $y' = \lambda y$ erhielten wir eine Matrizen-Rekursion, für die sich eine angepaßte Definition der A-Stabilität formulieren ließ. Aus der Übertragung der L-Stabilität ergab sich für Verfahren hoher Ordnung eine Steifgenauigkeitsbedingung.
Für s-stufige Verfahren mit Ordnung $p = s$ und Stufenordnung $q = s$ konnten wir die Existenz von L-stabilen Verfahren bis $s = 12$ nachweisen. Der Beweis ist konstruktiv und liefert Verfahren mit nilpotenter Stabilitätsmatrix $M(\infty)$. Durch umfangreiche numerische Suche erhielten wir für $s = 2$ und $s = 3$ weitere $\mathrm{L}(\alpha)$-stabile PTSW-Verfahren, die für konstante Schrittweiten zusätzlich die Bedingung $\mathrm{B}(s+1)$ erfüllen. Bei Verzicht auf die Steifgenauigkeit konnten wir $\mathrm{A}(\alpha)$-stabile Verfahren konstruieren, die auch für variable Schrittweiten mit Ordnung $s+1$ konvergieren.
Ausführlich haben wir die Implementierung der PTSW-Verfahren mit Schrittweitensteuerung beschrieben. Die Stufen wurden mittels Compilerdirektiven parallelisiert. Ein Nutzer unserer PTSW-Codes muß lediglich eine Subroutine zur Auswertung der rechten Seite der Differentialgleichung zur Verfügung stellen, Zusatzaufwand für die Parallelisierung entsteht nicht.
In numerischen Tests konnten wir zeigen, daß zwei- und dreistufige PTSW-Verfahren in der vorliegenden Implementierung robust und effizient sind und bereits sequentiell konkurrenzfähig zu Standardintegratoren sind. Für Probleme mit LU-Zerlegung erwiesen sich steifgenaue Verfahren, die $\mathrm{B}(s+1)$ für konstante Schrittweiten erfüllen (PTSW2A und PTSW3A), und für Probleme mit Krylovapproximation Verfahren mit nilpotenter Stabilitätsmatrix (PTSW2B und PTSW3B) als besonders geeignet. Durch die Parallelisierung (mit Speedups von bis zu 1.6 bzw. 2.5 mit zwei bzw. drei Prozessoren) waren PTSW2B und PTSW3B für alle betrachteten Beispiele mit iterativer Stufenlösung deutlich effizienter als VODPK für alle Toleranzen.
Vermutlich ist es schwer, innerhalb der untersuchten Klasse von PTSW-Verfahren bessere Koeffizientensätze mit $s = 2$ oder $s = 3$ Stufen zu finden. Für eine Fortführung der Untersuchungen an Zweischritt-W-Methoden müßten wir daher die Klasse verallgemeinern oder die Stufenzahl erhöhen. Eine wichtige Frage ist dabei, ob die Parallelität erhalten werden soll. Wir können sie systematisch ausbauen und mehr Prozessoren voraussetzen oder, ganz im Gegenteil, sequentielle Zweischritt-Verfahren untersuchen. Beide Richtungen erscheinen sinnvoll und sollen im folgenden kurz diskutiert werden:
Wenn die Stufenzahl der PTSW-Verfahren z.B. auf $s = 8$ erhöht wird, so können wir praktisch nützliche Verfahren nur dann gewinnen, wenn wir die Ordnungs- und Stufenordnungsbedingungen, und damit die Kopplung der Stufen aneinander, nicht zu stark wählen. Vorstellbar wäre etwa ein Verfahren mit $s = 8$ Stufen und Ordnung $p = 4$ mit $\mathrm{C}(4)$, $\Gamma(4)$ und $\mathrm{B}(4)$. Die Freiheitsgrade könnten dafür genutzt werden, eine einfache Struktur der Stabilitätsmatrix $M(w)$ vorzuschreiben, um so robuste Verfahren mit hoher Genauigkeit (auch für variable Schrittweiten) zu finden. Der zusätzliche Aufwand für jeden Schritt könnte durch die entsprechend erhöhte Prozessorenzahl teilweise kompensiert werden.
Wie wir gesehen haben, sind die PTSW-Verfahren mit Krylovlösung bereits sequentiell mit Standardinte-

gratoren konkurrenzfähig. Es ist daher interessant, sequentielle Zweischritt-W-Methoden der Form

$$Y_{mi} = u_m + h_m \sum_{j=1}^{s} a_{ij} k_{m-1,j} + h_m \sum_{j=1}^{i-1} \tilde{a}_{ij} k_{m,j} \tag{6.1a}$$

$$(I - h_m \gamma T_m) k_{m,i} = f(Y_{mi}) + h_m T_m \sum_{j=1}^{s} \gamma_{ij} k_{m-1,j} + h_m T_m \sum_{j=1}^{i-1} \tilde{\gamma}_{ij} k_{mj} \tag{6.1b}$$

$$u_{m+1} = u_m + h_m \sum_{i=1}^{s} (b_i k_{m,i} + v_i k_{m-1,i}) \tag{6.1c}$$

zu untersuchen. Die neu eingeführten Parameter $\tilde{\gamma}_{ij}$ stellen eine sequentielle Kopplung der Stufen dar. Mit ihrer Hilfe können wir die PTSW-Verfahren deutlich verbessern: z.B. gilt Satz 3.3 für die Verfahren (6.1) nicht, d.h. Steifgenauigkeit und Konvergenz B($s+1$) schließen sich nun nicht mehr aus. Vorläufige Ergebnisse zeigen weiter, daß es möglich ist, L-Stabilität mit kleinen Werten γ zu erzielen und die σ_m-Abhängigkeit der Matrix $M(w)$ deutlich zu reduzieren.

Literatur

[1] R. Barrett, M. Berry, T.F. Chan, and et al., *Templates for the solution of linear systems: building blocks for iterative methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.

[2] Z. Bartoszewski and Z. Jackiewicz, *Construction of two-step Runge-Kutta methods of high order for ordinary differential equations*, Numerical Algorithms **18** (1998), 51–71.

[3] P. Bastian, K. Birken, K. Johannsen, S.Lang, V. Reichenberger, C. Wieners, G. Wittum, and C. Wrobel, *Parallel solution of partial differential equations with adaptive multigrid methods on unstructured grids*, High performance computing in science and engineering '99, HLRS Stuttgart, 2000, pp. 496–508.

[4] A. Bellen, *PADETEST – a set of real-life test differential equations for parallel computing*, Tech. Report 103, Universita di Trieste, 1992.

[5] C. Bendtsen, *Parallel Numerical Algorithms for the Solution of Systems of Ordinary Differential Equations*, Ph.D. thesis, IMM, DTU, Denmark, 1996.

[6] M. A. Botchev, G. L. G. Sleijpen, and H. A. van der Vorst, *Stability control for approximate implicit time stepping schemes with minimum residual iterations*, Appl. Numer. Math. **31** (1999), no. 3, 239–253.

[7] P.N. Brown, D.G. Byrne, and A.C. Hindmarsh, *VODE: a variable-coefficient ODE solver*, SIAM J. Sci. Statist. Comput. **10** (1989), no. 5, 1038–1051.

[8] P.N. Brown and A.C. Hindmarsh, *Matrix-free methods for stiff systems of ODE's*, SIAM J. Numer. Anal. 23 (1986), no. 23, 610–638.

[9] P.N. Brown, A.C. Hindmarsh, and R.L. Petzold, *Using Krylov methods in the solution of large-scale differential algebraic systems*, SIAM J. Sci. Comput. **15** (1994), no. 6, 1467–1488.

[10] L. Brugnano and D. Trigiante, *Solving differential problems by multistep initial and boundary value methods*, Gordon and Breach, 1998.

[11] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations*, Oxford University Press, 1995.

[12] K. Burrage and H. Suhartanto, *Parallel iterated methods based on variable step-size multistep Runge-Kutta methods of Radau type for stiff problems*, Adv. Comput. Math. **13** (2000), no. 3, 257–270.

[13] J.C. Butcher, *The numerical analysis of ordinary differential equations*, John Wiley & Sons, 1987.

[14] J.C. Butcher and P. Chartier, *Parallel general linear methods for stiff ordinary differential and differential algebraic equation*, Appl. Num. Math. **17** (1995), 213–222.

[15] J.C. Butcher and A.D. Singh, *The choice of parameters in parallel general linear methods for stiff problems* , Appl. Num. Math. **34** (2000), 59–84.

[16] M. Büttner, B.A. Schmitt, and R. Weiner, *Automatic partitioning in linearly-implicit Runge-Kutta methods*, Appl. Num. Math. **13** (1993), 41–55.

[17] G.D. Byrne, *Pragmatic experiments with Krylov methods in the stiff ODE setting*, Computational Ordinary Differential Equations, Clarendon Press, Oxford, 1992, pp. 323–356.

[18] N.H. Cong, *A general family of pseudo two-step Runge-Kutta methods*, submitted for publication, 1997.

[19] ———, *Explicit pseudo two-step Runge-Kutta methods for parallel computers*, Int. J. Comput. Math. **73** (1999), 77–91.

[20] N.H. Cong, H. Podhaisky, and R. Weiner, *Numerical experiments with some explicit pseudo two-step RK methods on a shared memory computer*, Computers & Mathematics with Applications **36** (1998), 107–116.

[21] ———, *Performance of explicit pseudo two-step RKN methods on a shared memory computer*, submitted for publication, 2001.

[22] N.H. Cong, K. Strehmel, R. Weiner, and H. Podhaisky, *Runge–Kutta-Nyström-type parallel block predictor-corrector methods*, Advances in Computational Mathematics **10** (1999), 115–133.

[23] *CWI Testset for IVP solvers*, 1996-98, http://www.cwi.nl/cwi/projects/IVPtestset, release 2.1.

[24] P. Eberhard and C. Bischof, *Automatic differentiation of numerical integration algorithms*, Math. Comp. **68** (1999), no. 226, 717–731.

[25] R. F. Enenkel and K. R. Jackson, *DIMSEMs-diagonally implicit single-eigenvalue methods for the numerical solution of stiff ODEs on parallel computers*, Adv. Comput. Math. **7** (1997), no. 1-2, 97–133, Parallel methods for ODEs.

[26] W.H. Enright and J.D. Pryce, *Two FORTRAN Packages for Assessing Initial Value Methods*, ACM TOMS **13** (1987), 28–34.

[27] W. Gander and J. Hrebicek, *Solving problems in scientific computing using Maple and MATLAB*, 3. edition ed., Springer, 1997.

[28] A. Gerisch and H. Podhaisky, *Splitting Methods for the Simulation of Tumor Angiogenesis Models*, Proceedings of the 16th IMACS World Congress, Lausanne,Switzerland (M. Deville and R. Owens, eds.), 2000, published on CDROM.

[29] E. Hairer, S.P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*, Springer, 1993.

[30] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*, Springer, 1996.

[31] ———, *Stiff differential equations solved by Radau methods*, J. Comput. Appl. Math. **111** (1999), no. 1-2, 93–111.

[32] A.C. Hindmarsh and A.G. Taylor, *PVODE and KINSOL: Parallel Software for Differential and Nonlinear Systems*, Tech. Report UCRL-ID-129739, Lawrence Livermore National Laboratory, 1998.

[33] M. Hochbruck, C. Lubich, and H. Selhofer, *Exponential integrators for large systems of differential equations*, SIAM J. Sci. Comput. **19** (1998), no. 5, 1552–1574.

[34] A. Iserles and S. P. Nørsett, *On the theory of parallel Runge-Kutta methods*, IMA J. Numer. Anal. **10** (1990), no. 4, 463–488.

[35] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, and S.M. Vorkoetter, *Maple V Programming Guide*, Springer, 1996.

[36] H. Podhaisky, *Parallele explizite Zweischritt Runge-Kutta-Methoden*, Diplom-Thesis, Universität Halle, 1998.

[37] H. Podhaisky, B.A. Schmitt, and R. Weiner, *Two-step W-methods with parallel stages*, Tech. Report 22, Universität Halle, FB Mathematik/Informatik, 1999.

[38] ———, *Design, analysis and testing of some parallel two-step W-methods for stiff systems*, Appl. Num. Math. (2001), accepted for publication.

[39] H. Podhaisky and R. Weiner, *A Class of Explicit Two-Step Runge-Kutta Methods with Enlarged Stability Regions for Parallel Computers*, Parallel Computation (P. Zinterhofer, M. Vajteršic, and Andreas Uhl, eds.), Lecture Notes in Computer Science, 1557, Springer, 1999, pp. 68–77.

[40] H. Podhaisky, R. Weiner, and B.A. Schmitt, *Numerical experiments with Krylov integrators*, Appl. Num. Math. **28** (1998), 413–425.

[41] ———, *Numerical experiments with parallel two-step W-methods for DAEs*, submitted to ZAMM (2001).

[42] H. Podhaisky, R. Weiner, and J. Wensch, *High Order Explicit Two-Step Runge-Kutta Methods for Parallel Computers*, Journal of Computing and Information Technology **8** (2000), no. 1, 13–18.

[43] R. Pulch, *A Parallel RADAU5 Code*, Tech. Report 13, IWRMM Karlsruhe, 2000.

[44] Y. Saad, *Iterative methods for sparse linear systems*, The PWS series in computer science, PWS, 1996.

[45] B.A. Schmitt and R. Weiner, *Matrix-free W-methods using a multiple Arnoldi iteration*, Appl. Numer. Math. **18** (1995), 307–320.

[46] B.A. Schmitt, R. Weiner, and H. Podhaisky, *Parallel two-step W-methods*, ZAMM **81** (2001), no. Suppl. 3, 749–752.

[47] T. L. Sterling, J. Salmon, and D. J. Becker, *How to Build a Beowulf*, MIT press, 1999.

[48] K. Strehmel and R. Weiner, *Linear-implizite Runge-Kutta-Methoden und ihre Anwendung*, Teubner, Stuttgart-Leipzig, 1992.

[49] ———, *Numerik gewöhnlicher Differentialgleichungen*, Teubner, 1995.

[50] P.J. van der Houwen and J.J.B. de Swart, *Parallel linear system solver for Runge-Kutta methods*, Advances in Computational Mathematics (1997), 157–181.

[51] P.J. van der Houwen and B. P. Sommeijer, *CWI contributions to the development of parallel Runge-Kutta methods*, Appl. Num. Math. **22** (1996), 327–344.

[52] H. van der Vorst, *Minimum residual modifications to Bi-CG and to the preconditioner*, Recent advances in iterative methods, Springer, New York, 1994, pp. 217–225.

[53] R. Weiner, B.A. Schmitt, and H. Podhaisky, *ROWMAP – a ROW-code with Krylov techniques for large stiff ODEs*, Appl. Num. Math. **25** (1997), 303–319.

[54] ———, *Two-step W-methods on singular perturbation problems*, Tech. report, Philipps-Universität Marburg, 2000.

[55] M. Wussling, K. Krannich, V. Drygalla, and H. Podhaisky, *Calcium waves in agarose gel with cell organelles: Implications of the velocity curvature relationship*, Biophysical Journal **80** (2001), 2658–2666.

[56] M. Wussling, K. Krannich, G. Landgraf, A. Herrmann-Frank, D. Wiedenmann, and H. Podhaisky, *Sarcoplasmic reticulum vesicles embedded in agarose gel exhibit propagating calcium waves*, FEBS Letters **463** (1999), 103 – 109.

A Maple-Skripte

A.1 Einleitung

Bei der Ausführung „mechanischer“ Rechnungen können Computeralgebra-Programme sehr nützlich werden. Wenn man genau weiß, was eingesetzt, umgeformt, zusammengefaßt oder entwickelt werden soll und dann noch die richtigen Kommandos findet, kann man sich z.B. von Maple V sehr gut helfen lassen. Allerdings ist die Bedienung stark gewöhnungsbedürftig und vieles klappt nicht gleich auf Anhieb. Unser Ziel ist hier, für vier Beispiele aus dem Themenkreis dieser Arbeit voll funktionsfähige Maple-Arbeitsblätter zu entwerfen, die als nützliche Vorlagen beim Lösen ähnlicher Probleme dienen können.
Wir verwenden einen funktionalen Programmierstil und setzen mit Hilfe von Operationen (wie `map`, `@` und `unapply`) Grundfunktionen geeignet zusammen. Das Ergebnis mag auf den ersten Blick etwas kryptisch erscheinen, ist dafür aber relativ kompakt und effizient ausführbar.
Eine umfassende Darstellung der Programmierung in Maple mit vielen Tricks und Kniffen findet man im Handbuch zur Software [35]. Wie man dieses Wissen sinnvoll für nichttriviale praktische Probleme anwenden kann, zeigen Gander u.a. [27] anhand von einigen besonders instruktiven Beispielen.

Bemerkung A.1 (Kompatibilität zu Maple 6/7). Die hier vorgestellten Programme wurden für Maple V R4 entwickelt. In den jüngsten Maple-Versionen 6 und 7 wurden leider einige Konzepte geändert, so daß alte Programme nicht mehr lauffähig sind. Ein kurzer Überblick, was geändert werden müßte:

Maple V R4	Maple 6/7	Bemerkung
`"`	`%`	ditto-Operator, liefert letztes Ergebnis
`.`	`||`	Stringkonkatination
`stack`	`stackmatrix`	Setzt zwei Matrizen übereinander
`fortran`	`codegen[fortran]`	Ausgelagert in ein neues Paket, weil es vorher Namenskollisionen (mit `stack`) gab.
`'`	`"`	Stringbegrenzer

◊

A.2 Explizite Zweischritt-Runge-Kutta-Verfahren

Wir wollen das explizite Verfahren mit $s = 4$ mit $p = 6$ bestimmen, für welches die 2-Norm des Stufenfehlers minimal wird (vgl. Abschnitt 3.4). Neben der Einheitsmatrix und den Einheitsvektoren

```
> restart:with(linalg):s:=4:
> eins:=vector([1$i=1..s]):Id:=diag(1$s):
> assign({seq(e[i]=vector([0$i-1,1,0$s-i]),i=1..s)}):
```

benötigen wir die Matrizen

```
> Di:=diag(1/i$i=1..s):
> P:=matrix(s,s,[seq((binomial(j-1,i-1)$j=1..s),i=1..s)]):
> F:=matrix(s,s,[0$s*s]): assign({seq(evaln(F[i+1,i])=1,i=1..s-1),
> seq(evaln(F[i,s])=-phi[i-1],i=1..s)}):
```

sowie die Äquivalente zu den Gleichungen (3.14)-(3.17).

```
> r:=evalm(add(evalm(e.i&*F**(i-1)),i=2..s)+e1):
> V0tV0:=stack(r,(r&*F**i)$i=1..s-1):
> phi[s]:=1:phix:=add(x**l*phi[l],l=0..s):
> bs1:=int(phix,x=0..1):bs2:=int(x*phix,x=0..1):
> x:=evalm((1/(s+1)*F**2-F&*Di&*(P&*F&*inverse(P)-Id))&*e.s):
> bx:=simplify(evalm(eins&*Di&*x));
> XTX:=simplify(evalm(x&*V0tV0&*x)):
```

Wir erhalten die Lösung mit minimalem Stufenfehlervektor X. Erst als Koeffizienten des $\varphi(x)$-Polynoms,

```
> sol:=solve({bx,bs1,bs2},{phi[i]$i=1..3});
> poly:=unapply(subs(phi[0]=x,simplify(subs(sol,XTX))),x);
> sol0:={phi[0]=fsolve(D(poly),0.1..0.2)};
> sol_end:=subs(sol0,sol) union sol0:
```

$$\{\varphi_0 = 0.1775140423\}$$

dann daraus auch die Knoten c_i.

```
> knoten=[fsolve(subs(sol_end,phix),x,complex)];
```

$$\text{knoten} = [0.1489608115, 0.6519353270, 1.117237175, 1.636103565].$$

Wesentlich ist die Darstellung der Knoten c_i als Nullstellen des Polynoms $\varphi(x) = \prod_{i=1}^{s}(x - c_i)$, weil die Quadraturbedingungen dann als lineare Gleichungen in den Polynomkoeffizienten φ_i leichter erfüllt werden können. Das naheliegende Vorgehen, die Knoten gleich aus den nichtlinearen Bedingungen bestimmen zu lassen, überfordert die Möglichkeiten von Maple sofort.

A.3 Plotten von Stabilitätsgebieten

Sei `cp` das charakteristische Polynom der Stablitätsmatrix $\texttt{cp} = \det(xI - M(z))$ als Funktion in x und z. Wir definieren eine Funktion f, die zu gegebenem x auf dem Einheitskreis die Menge der Randpunkte des Stabilitätsgebiets z berechnet mit $\texttt{cp}(x, z) = 0$. Dabei sind von der gesamten Wurzelortskurve nur die stabilen Zweige für eine Darstellung interessant, deshalb lösen wir zwei Nullstellenprobleme und selektieren mit der Funktion r.

```
> restart:with(linalg):with(plots):
> f1:=x->{fsolve(cp(x,z),z,complex)}:
> f2:=z->{fsolve(cp(x,z),x,complex)}:
> r:=S->max(op(map(abs,S)))<1.0001:
> f:=x->(op@select)(r@f2,f1(x)):
```

Wir wenden f nun für ein gegebenes Polynom auf Punkte des Einheitskreises an und zeichnen das Ergebnis in der komplexen Ebene.
Z.B. liefert

```
> cp:=(x,z)->x^5+(-4.8886*z-1.)*x^4+(7.5099*z^2+3.8886*z)*x^3+
> (-3.8710*z^3-4.1213*z^2)*x^2+(1.0273*z^3+.44139*z^4)*x-.73964e-2*z^4:
>
> complexplot([seq(f(exp(2*I*Pi*i/100)),i=1..100)],style=point);
```

das Stabilitätsgebiet von EPTRK4, siehe Abbildung 3.12. Da `fsolve` relativ langsam ist, kann man zur effizienteren Berechnung von `f1` bzw. `f2` auch die Eigenwerte der Begleitmatrix numerisch berechnen, was jedoch leider erst ab Maple 6 effizient möglich ist (da der in Maple VR4 implementierte QR-Algorithmus für komplexe Matrizen schlecht, teilweise sogar überhaupt nicht, konvergiert).

A.4 Konstruktion einer vierstufigen steifgenauen PTSW-Methode nach Satz 3.1

Wir setzen aus den entsprechenden Blöcken die transformierte Stabilitätsmatrix

$$\tilde{M}(z) = \mathrm{diag}(V_0^{-1}, 1) M(z) \,\mathrm{diag}(V_0, 1)$$

zusammen.

```
> restart:with(linalg):s:=4:
> Id:=diag(1$s):Di:=diag(1/i$i=1..s):F:=matrix(s,s,[0$s*s]):
> P:=matrix(s,s,[seq(binomial(j-1,i-1)$j=1..s,i=1..s)]):
> F:=matrix(s,s,[0$s*s]):assign({seq(evaln(F[i+1,i])=1,i=1..s-1),
> seq(evaln(F[i,s])=-phi[i-1],i=1..s)}):
> beta:=evalm((F&*Di-g*Id)&*P):
> eins:=vector([1$s]):
> phi[s]:=1:phi[0]:=-add(phi[i],i=1..s):
> Mw:=augment(stack(w*beta,(g*w+1)*eins&*beta),
> matrix(s+1,1,[w,0$s-1,g*w+1]));
```

Das Ergebnis paßt kaum auf die Zeile.

$$\begin{bmatrix} -gw & -gw & -gw & w\,(-g+1/4+1/4\,\varphi_1+1/4\,\varphi_2+1/4\,\varphi_3) & w \\ w & w\,(1-g) & w\,(1-2\,g) & w\,(1-3\,g-1/4\,\varphi_1) & 0 \\ 0 & 1/2\,w & w\,(1-g) & w\,(3/2-3\,g-1/4\,\varphi_2) & 0 \\ 0 & 0 & 1/3\,w & w\,(1-1/4\,\varphi_3-g) & 0 \\ (gw+1)\,(1-g) & (gw+1)\,(-2\,g+3/2) & (gw+1)\,(-4\,g+7/3) & (gw+1)\left(-8\,g+\frac{15}{4}\right) & gw+1 \end{bmatrix}$$

Diese Matrix enthält wichtige Informationen: für $z \to 0$ die Konsistenzbedingungen bzw. Fehlerkonstanten. Wir bilden das charakteristische Polynom, differenzieren und setzen $w = 0, x = 1$, stellen nach den Differentialen um und erhalten so die implizite Taylorentwicklung von $x(w)|_{w=0,x=1}$, die wir mit der analytischen Lösung vergleichen. Eigentlich können wir den untenstehenden Block unter Verwendung von `implicitdiff` wesentlich vereinfachen, nur ab $s \geq 3$ sind wir aus Effizienzgründen zu diesen Kopfständen gezwungen.

```
> cp:=unapply(charpoly(Mw,x),x,w):
> null:=expr->subs({x=1,w=0,dx[0]=1},expr):dx[0]:=1:
> eq:={seq(map(null,subs({seq((D@@i)(x)(w)=dx[i],i=0..8)},
> convert(diff(cp(x(w),w),w$j),D)))=0,j=1..8)}:
> sol_dx:=solve(eq,{seq(dx[i],i=1..8)}):
> ntc:=i->subs(sol_dx,dx[i]):
> x_von_w:=add(ntc(i)*w**i/i!,i=1..6)+1:
> x_von_z:=map(simplify,taylor(subs(w=z/(1-g*z),x_von_w),z));
> Cs1:=1/(s+1)!-coeff(x_von_z,z,s+1):
```

$$1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 + \left(-\frac{5}{24} + 5/8\,g + 1/8\,g\varphi_2 - \frac{11}{96}\,\varphi_3 + \frac{7}{24}\,g\varphi_3 + 1/24\,\varphi_2 g - 1/48\,\varphi_1 - 1/18\,\varphi_2\right) z^5 + O\left(z^6\right) \quad \text{(A.1)}$$

Für $z = \infty$ setzen wir die Koeffizienten des charakteristischen Polynoms null. Dies garantiert Nilpotenz und somit Stabilität. Durch schrittweises Auflösen ergeben sich die φ-Parameter als Funktion von γ entsprechend dem Beweis von Satz 3.1.

```
> cpinf:=collect(charpoly(subs(w=-1/g,evalm(g*Mw)),x),x);
> sol:=solve({(coeff(cpinf,x,i)=0)$i=2..s},{phi[i]$i=1..3});
> pg:=collect(subs(sol,coeff(cpinf,x,1)),g);
> gg:=fsolve(pg,g,complex)[1];
```

$$\left\{\varphi_3 = 12 - 16\,g, \varphi_2 = 48 - 132\,g + 72\,g^2, \varphi_1 = -288\,g + 64 + 336\,g^2 - 96\,g^3\right\}.$$

Für γ bleibt eine Bedingung vom Polynomgrad s. Wir wählen die kleinste Nullstelle $\gamma = 0.4564586$, setzen alles ein und erhalten Fehlerkonstanten:

```
> Einsetzen:=expr->subs({g=gg},subs(sol,expr));
> HauptFehlerTerm=Einsetzen(Cs1);
```

$$HauptFehlerTerm = -0.1718166884$$

Aus dem charakterischen Polynom zu $\tilde{M}(z)$ lassen sich die Randpunkte des Stabilitätsgebiets S berechnen und schließlich der L(α)-Winkel mit $\alpha = \inf\{|\arg(z) - \pi| : z \in \partial S\}$.

```
> ccp:=Einsetzen(cp(x,w));
> winkel:=w->180/Pi*abs(Pi-argument(w/(1+gg*w))):
> ff:=unapply('fsolve'(ccp,w,complex),x):
> alpha=evalf((min@op@map)(winkel,map(ff,[seq(exp(2*I*Pi*i/400),i=0..400
> )])));
```

$$\alpha = 68.49750431$$

A.5 Fortran-Koeffizientensätze

Um die berechneten Koeffizienten in hoher Genauigkeit in die Implementierung des Integrators einzubauen, eignen sich die Fortran-Exportfunktionen von Maple vorzüglich. Für $s = 2$ geben wir uns $\gamma = 4/5$ vor. Aus $B(3)$ folgt $c_1 = 1 + (1/3 - \gamma)/(1/2 - \gamma)$. Die Matrizen für die vereinfachenden Konsistenzbedingungen (2.13) sind:

```
> restart:with(linalg):mf:='ptsw2a':
> s:=2:g:=4/5:eins:=vector(s,[1$s]):
> cl:=[1+(1/3-g)/(1/2-g),1]:c:=vector(cl):C:=diag(op(cl)):
> V0:=vandermonde(cl):Di:=diag(1/i$i=1..s):
> S:=diag(si**(i-1)$i=1..s):
> P:=matrix(s,s,[seq((binomial(j-1,i-1)$j=1..s),i=1..s)]):
> PiV:=evalm(P&*inverse(V0)):
```

Die Verfahrenskoeffizienten für variable Schrittweiten ergeben sich aus den vereinfachenden Konsistenzbedingungen (2.14).

```
> a:=evalm(C&*V0&*S&*Di&*PiV);
> gg:=evalm(-g*V0&*S&*PiV);b:=vector(s,[0$s-1,g]);
> v:=evalm( (eins&*S&*Di-b&*V0&*S) &* PiV);
> err:=1/10: be:=evalm(95/100*b);
> ve:=evalm( ((eins+vector(s,[0$s-1,err]))&*S&*Di-be&*V0&*S) &* PiV);
```

Das Verfahren ist vollständig berechnet und wir können die Koeffizienten als Fortran-Block ausgeben.

```
> f77:=expr->fortran(expr,precision=double,optimized):
> file:=cat(mf,‘.f‘):writeto(file);
> printf(‘      if (mf.eq.’%s’) then\n‘,mf);
> fortran([’s’=s,’ord’=s+1,’eord’=s-1]);f77([’g’=g]);
> f77(a);f77(b);f77(c);f77(gg);f77(v);f77(ve);f77(be);
> printf(‘      return\n‘);
> printf(‘      end if\n‘);
> writeto(terminal);
```

Das Ergebnis steht in der Datei `’ptsw2a.f’` und kann ohne Änderungen ins Fortranprogramm mittels `include` eingebunden werden.

```
if (mf.eq.’ptsw2a’) then
s = 2
ord = 3
eord = 1
g = 4.D0/5.D0
a(1,1) = 529.D0/252.D0*si
a(1,2) = 23.D0/9.D0-529.D0/252.D0*si
a(2,1) = 9.D0/28.D0*si
a(2,2) = 1.D0-9.D0/28.D0*si
b(1) = 0.D0
b(2) = 4.D0/5.D0
c(1) = 23.D0/9.D0
c(2) = 1.D0
gg(1,1) = -46.D0/35.D0*si
gg(1,2) = -4.D0/5.D0+46.D0/35.D0*si
gg(2,1) = -18.D0/35.D0*si
gg(2,2) = -4.D0/5.D0+18.D0/35.D0*si
v(1) = -27.D0/140.D0*si
v(2) = 1.D0/5.D0+27.D0/140.D0*si
ve(1) = -27.D0/200.D0*si
ve(2) = 6.D0/25.D0+27.D0/200.D0*si
be(1) = 0.D0
be(2) = 19.D0/25.D0
return
end if
```

Danksagung

Die vorliegende Arbeit entstand im Institut für Numerische Mathematik am Fachbereich für Mathematik und Informatik der Martin–Luther–Universität Halle–Wittenberg.

Mein herzlicher Dank gilt meinem Betreuer und akademischen Lehrer Prof. Dr. Rüdiger Weiner für freundliche Überlassung des Themas, für seine umfassende Unterstützung meiner Arbeit und die vielen anregenden Diskussionen.

Danken möchte ich auch Prof. Dr. Bernhard Schmitt, für die originellen e-mails aus Marburg, die für die Analyse der PTSW-Verfahren eine große Hilfe waren.

Mein besonderer Dank gilt Dr. Alf Gerisch und Dr. Jörg Wensch für Ihre wertvollen Anregungen. Allen Mitarbeiterinnen und Mitarbeitern am Institut danke ich für das freundliche und kreative Arbeitsklima.

Hans-Jürgen Walter hat als Systembetreuer im Rechenzentrum häufig Prozessoren zu privaten Subkomplexen für mich zusammengefaßt. Dafür und für seine Bereitschaft, mir bei diversen Systemproblemen, die im Umgang mit modernen Parallelrechnern auftreten, weiterzuhelfen, danke ich ihm sehr.

Erklärung

Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die aus anderen Werken wörtlich oder inhaltlich entnommenen Daten, Fakten und Konzepte sind unter Angabe der entsprechenden Quellen als solche gekennzeichnet.

Diese Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem anderen Prüfungsverfahren vorgelegt.

Halle (Saale), im Oktober 2001

Helmut Podhaisky

Lebenslauf

Helmut Podhaisky
geboren am 30.6.1973 in Halberstadt,
verheiratet mit Stephanie Podhaisky, eine Tochter

1980 – 1988	26. POS „Gotthold Ephraim Lessing“ in Halle-Neustadt
1988 – 1992	Spezialschule „Georg Cantor“ in Halle, Abitur
1.7.1992 – 30.9.1993	Zivildienst
29.9.1993-26.5.1998	Mathematikstudium mit Nebenfach Informatik an der Martin-Luther-Universität Halle, Abschluß: Diplommathematiker
1.8.1995 – 31.7.1997	Wissenschaftliche Hilfskraft im DFG-Projekt „Krylov-W-Methoden“ unter Leitung von Prof. Schmitt (Universität Marburg) und Prof. R. Weiner.
15.6.1998 – 31.3.2000	Wissenschaftliche Hilfskraft im DFG-Projekt „Raum-Zeit-Strukturen von Ca^{2+}-Signalen in Säugermyokartzellen“ unter Leitung von Prof. Wußling, Julius-Bernstein-Institut für Physiologie.
1.10.1998 – 31.10.2001	Doktorand am Institut für Numerische Mathematik, Thema der Dissertation: „Parallele Zwei-Schritt-W-Methoden“ betreut von Prof. R. Weiner
seit 1.11.2001	Mitarbeiter im Institut für Informatik der Martin-Luther-Universität Halle
10. 01. 2002	Verteidigung der Dissertation