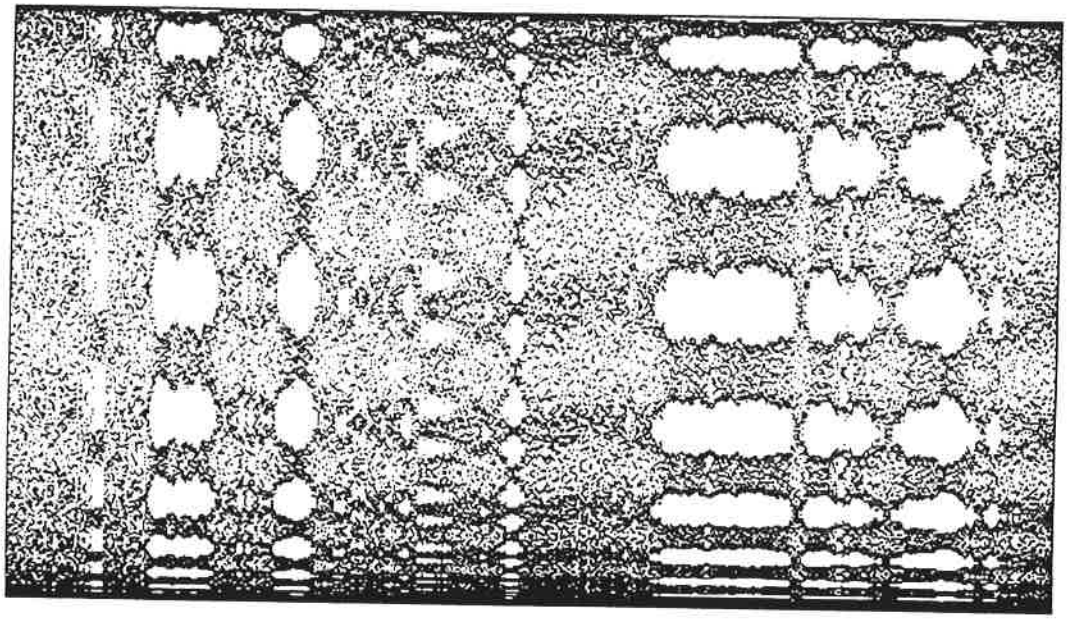
UNTERSUCHUNGEN VON EXAKT-REVERSIBLEN ALGORITHMEN FÜR DYNAMIK-SIMULATIONEN



DIPLOMARBEIT
VON
HANS H. DIEBNER

VORGELEGT DEM
DIPLOMPRÜFUNGSAUSSCHUß
DER
FAKULTÄT FÜR PHYSIK.

ERSTELLT AM
INSTITUT FÜR PHYSIKALISCHE
UND THEORETISCHE CHEMIE
DER EBERHARD-KARLS-UNIVERSITÄT
TÜBINGEN

OKTOBER 1993

Inhalt

Vorwort

Nach gängiger Lehrmeinung leben wir in einer irreversiblen Welt, eine Tatsache(?), die sich durch Einführung und Verhalten der "Entropie" niederschlägt. Wie sich die "Entropie" zu verhalten hat, wird ihr durch den sogenannten zweiten Hauptsatz der Wärmelehre nahegelegt. Ungeschickterweise gibt es aber gar zahlreiche, verschiedene Vorstellungen, wie diese "Entropie" auszusehen habe. Auf BOLTZMANN geht einerseits die Erkenntnis zurück, daß die "Entropie" proportional zum Logarithmus des Phasenraumvolumens sein muß. In seinen reiferen Jahren hat BOLTZMANN andererseits versucht, die "Entropie" eines Vielteilchen-Systems anhand einer "Verteilungsfunktion", für die vielen Teilchenzustände im *Ein*-Teilchen-Phasenraum zu definieren. GIBBS benutzte in einem "Verbesserungs-" Vorschlag zwar denselben funktionalen Zusammenhang zwischen "Entropie" und einer Verteilungsfunktion, letztere aber beschreibt bei ihm eine Verteilung von Zuständen in einem fiktiven Ensemble von (wahren) Vielteilchen-Phasenräumen.

E. T. JAYNES[$] zeigte 1965, daß die zweite BOLTZMANNsche Entropieformel weder zur GIBBSschen noch zu seiner ursprünglichen äquivalent ist und daß weiterhin gilt:

> *From this we see that entropy is an antropomorphic concept, not only in the well-known statistical sense that it measures the extent of human ignorance as to the microstates. Even at purely phenomenological level, entropy is an antropomorphic concept. For it is a property, not of the physical system, but of the particular experiments you or I choose to perform.*

Zu ähnlicher Erkenntnis gelangt auch F. SCHLÖGL in seinem Lehrbuch "Propability and Heat - Fundamentals of Thermostatistics":

> *Entropy depends on which knowledge is taken notice of by the observer....*
>
> *For two observers who applied different methods the entropy change is different.*

Macht eine physikalische Größe, die von der Farbenblindheit des Beobachters abhängt, Sinn in einer objektiven Naturwissenschaft? Hat die "Entropie" gleichen Rang wie die Masse und die Energie, in dem Sinne, daß durch die Angabe ihrer Transformations- und Transporteigenschaften ein System vollständig und eindeutig charakterisiert ist?

Sehen wir vom Solipsismusproblem sowie der KANTschen Antinomienlehre, die natürlich auch Entitäten wie Masse und Energie betreffen, einmal ab, so ist die GIBBSsche "Entropie" doch in einem wesentlich objektiveren Sinne subjektiv, als "Masse" und "Energie". Dies liegt daran, daß gemäß SHANNON - dem später sogenannten Begründer der Informationstheorie - die GIBBSsche Entropieformel zu einer "informatorischen Entropie" verallgemeinerbar ist, wobei sich letztere auf ein "Informationsmaß" stützt, welches heute trotz entgegngesetzter Warnung seitens des Urhebers, als "Information" schlechthin gehandelt wird. Shannons Ableitung der "Entropie" fußt auf einer Analogie von nachrichtentechnischen Systemgrößen zur menschlichen Kommunikation und ist daher, wie man zeigen kann[$], abhängig vom menschlichen Handeln. Information ist daher

---

$Edwin T. Jaynes: "Papers on Probability, Statistics and Statistical Physics", edited by R. D. Rosenkrantz, 1983

$Wer das nicht glaubt, möge sich Peter Janich: "Grenzen der Naturwissenschaft", Kapitel 7. "Ist Information ein Naturgegenstand? - Menschliches Handeln als Ursprung des Informationsbegriffes", zu Gemüte führen.

kein Naturgegenstand in dem Sinne, daß er unabhängig vom Menschen existierte und kann daher nicht Grundlage einer Beschreibung der Reversibilitätseigenschaften der Welt sein. Daß die informatorische Entropie weiterhin status quo der Schulphysik ist bedeutet, daß

> *Die Verstehbarkeit von Theorien und der durch sie fingierten Entitäten im Sinne einer Nachvollziehbarkeit ihrer Entstehungsgeschichte als Mittel zur Problembewältigung...absichtsvoll ausgeschlossen wird.*

Außerdem:

> *Die Auffassung, Information sei ein Naturgegenstand und Informationsaustausch ein Naturprozeß, verdankt sich damit dem vorsätzlichen Ignorieren aller Herkunfts- und Entstehungszusammenhänge und einer Umdeutung einer Theorie zu etwas, als das sie weder intendiert noch erzeugt worden ist.*[&]

Etwas physikalischer argumentiert heißt das, daß nicht jede Entropieübertragung auch mit Informationsübertragung verknüpft ist. Die aus dem Informationsmaß berechnete "Entropie"-Übertragung ist stets kleiner als die (wahre) Entropieübertragung. Dies ist die sogenannte *SZILARD-BRILLOUIN-Relation.*[*]

Damit aber werfen sich für mich Fragen auf, die ich nicht durch bloße Absorption von Lehrmeinungen, mit denen man in den prüfungsrelevanten Pflichtveranstaltungen gefüttert wird, lösen möchte. Manche Erkenntnisse sind für mich nicht so plausibel, wie sie erachtet werden. Plausibel ist höchstens, daß man oft das für plausibel hält, was gar nicht plausibel ist. *Ein* Grund, mich für die Bearbeitung des vorliegende Themas zu entscheiden, liegt unter anderem in der maßgeblich von BENNETT und LANDAUER vertretenen Ansicht, Rechnen sei ein prinzipiell irreversibler Vorgang, wobei sie sich auf den zweiten Hauptsatz und damit auf die informatorische Entropie beziehen. Aber: Geschickt formulierte Subjektivität, d. h. Journalismus, macht noch keine Physik.

Ich möchte Herrn O. Rössler herzlich danken, daß ich meinen infantilen Fragen nach dem "Wie" und "Warum" im Rahmen dieser Diplomarbeit nachgehen durfte, obwohl die Arbeit selbst meine Fortschritte in obiger Sache nicht zu dokumentieren vermag. Meinen Spaß, den ich an der Sache hatte, sei ihm hiermit versichert. Wolfgang Schweizer vom Institut für Theoretische Astrophysik war so freundlich, sich als Mitbetreuer zur Verfügung zu stellen. Hierfür, sowie für gewinnbringenden Austausch insbesondere während der von ihm veranstalteten Vorlesung zu numerischen Algorithmen sei ihm herzlich gedankt. Auch Herrn H. Ruder, der die erforderliche Verantwortung für die Mitbetreuung übernimmt, schulde ich meinen Dank. Wichtig ist mir fernerhin für die zahlreichen Privataudienzen beim "Pabst" und für seine unermüdlichen Versuche, mich auf den rechten Weg zu bringen, zu danken. Großen Gewinn brachten mir auch die vielen Diskussionen mit Herrn D. Hoffmann, dem ich genauso wie allen anderen Mitgliedern des Instituts für Theoretische Chemie für ihre Hilfsbereitschaft und die angenehme Atmosphäre großen Dank schulde.

H. H. D.

&ebd. p.146

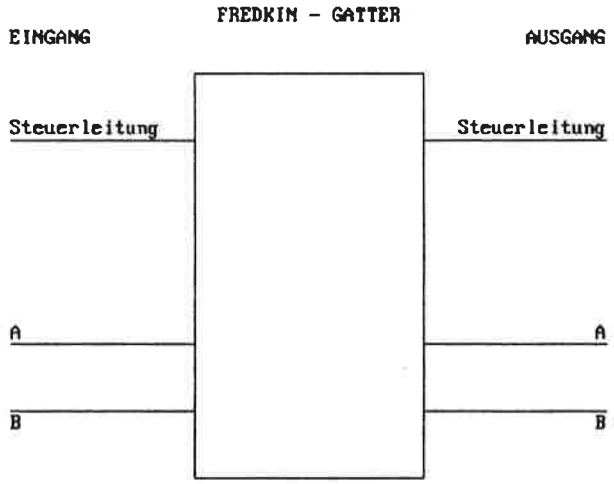
*Siehe z.B. Werner Ebeling: Chaos - Ordnung - Information, p.60

1. Einleitung

In dieser Arbeit wird ein allgemeines Schema zur Konstruktion von exakt-reversiblen Integrationsalgorithmen, also Methoden zur numerischen Lösung von Differentialgleichungen, angegeben und untersucht. Wie alle numerischen Verfahren für den Digital-Computer beruht auch dieses auf einer Diskretisierung des kontinuierlichen Problems, d.h. einer Approximation durch diskrete Iterationswerte. Die mit diesem Schema erzeugten Algorithmen ermöglichen die exakte Reproduktion aller dieser Iterationswerte ohne sie abzuspeichern. Die Algorithmen enthalten sozusagen einen *Gedächtnisterm*, der jedem Iterationswert eine Information über den vorhergehenden Wert aufprägt.

Die Idee zu diesen Algorithmen lehnt sich an die Idee des sogenannten *FREDKIN-Gatters* [1]. Dieses logische Gatter ermöglicht es, aus dem Output auf den Input zurückzuschließen, d.h. diesen exakt zu reproduzieren. Normalerweise läßt sich aus dem Ergebnis einer Rechnung nicht mehr eindeutig ermitteln, wie es zustande kam. Beim FREDKIN-Gatter jedoch wird eine Zusatzinformation so mit auf den Ausgang gegeben, daß dadurch ermöglicht wird, durch Neueingabe in den Eingang am Ausgang exakt den ursprünglichen Input zu reproduzieren. In Abb.1 ist das Prinzip des FREDKIN-Gatters anschaulich wiedergegeben.

Abb.1:Skizze zum FREDKIN - Gatter



EINGANG			AUSGANG		
A	B	Steuer-leitung	A	B	Steuer-leitung
0	0	1	0	0	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	1	1	1	1
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	1	1	0

Da sich nun aus FREDKIN-Gattern alle anderen logischen Gatter durch geeignete Verschaltung aufbauen lassen, kann jede beliebige Rechnung damit *exakt-reversibel* ausgeführt werden - jedenfalls im Prinzip. In praxi konnte bisher ein solches FREDKIN-Gatter noch nicht realisiert werden, denn mit jedem Rechenschritt müsste ein Gedächtnisterm abgespeichert werden. Es wird auch befürchtet, daß eine unmittelbare physikalische Realisierung am 2.Hauptsatz der Thermodynamik scheitern könnte, da auch ein Rechenprozeß letztendlich ein physikalischer Prozeß ist. Ein realer physikalischer Prozeß aber ist mit einer Energiedissipation verbunden, die sich im Rechenprozeß als Informationsverlust zeigt [2].

Allerdings sollen solche Fragen nicht Gegenstand dieser Arbeit sein, obwohl die Reversibilität bzw. Irreversibilität von physikalischen Prozessen auch hier eine gewisse Rolle spielt. Die Reversibilität von dynamischen Systemen spiegelt sich in der Invarianz der die Dynamik beschreibenden Differentialgleichung gegenüber Zeitumkehr wieder. Die Art, wie der Rechenprozeß zur Lösung sowie die Reversibilitätseigenschaften des zugrundeliegenden physikalischen Problems, repräsentiert durch eine Differentialgleichung, zusammenhängen, läßt sich in der vorliegenden Problemstellung am besten wie folgt beschreiben: Durch den exakt-reversibel konstruierten Integrationsalgorithmus wird dem zu simulierenden physikalischen System - genauer: der iterativen Lösung des Systems - eine Rückrechenbarkeit aufgezwungen,

und es erhebt sich die Frage, wie das ursprüngliche System durch den Algorithmus verändert, d.h. verfälscht wird. Natürlich erscheint eine Integration von irreversiblen Systemen mit reversiblen Algorithmen absurd, deshalb läßt sich der Anwendungsbereich von exakt-reversiblen Integrationsalgorithmen schon an dieser Stelle auf reversible Systeme einengen.

Interessant werden könnten solche Algorithmen dort, wo viele Freiheitsgrade im Spiel sind, also in typischen statistisch mechanischen Vielteilchenproblemen. Bekanntlich gibt es hier den (Schein-) Widerspruch zwischen der Reversibilität der mikroskopischen Beschreibung des zugrundeliegenden Systems durch eine HAMILTONsche Gleichung und der makroskopischen Irreversibilität, d.h. der Entropie-Zunahme eines solchen Systems, mit anderen Worten: der Existenz eines Zeitpfeils. J. HURLEY [3] konnte den Widerspruch durch eine angemessene stochastische Beschreibung solcher komplexer Systeme ausräumen. Wählt man nämlich einen beliebigen Anfangszustand eines Vielteilchensystems, dann wird sich eine Trajektorie im Phasenraum ausbilden, die zu einem überwiegenden Teil innerhalb eines Bereiches im Phasenraum verläuft, dessen Punkte mehr oder weniger dem Gleichgewichtszustand entsprechen. Sticht man nun gewissermaßen in dieses Trajektorienknäuel, so ist die Wahrscheilichkeit dafür, daß man auf einem Trajektorienstück landet, von dem aus man in beiden Zeitrichtungen sich noch mehr in Richtung Gleichgewichtszustand bewegt so groß, daß dies makroskopisch als Irreversibilität des Prozesses gedeutet wird.

Verwendet man nun einen Standard-Integrationsalgorithmus wie z.B. das RUNGE-KUTTA-Verfahren, dann wird man schon nach einer beliebig kleinen Integrationszeit nach Invertierung der Differentialgleichung trotzdem nicht vom obigen Gleichgewichtszustand wegkommen, sondern im Gegenteil noch weiter auf ihn zurechnen. Dies ist nun aber absolut verständlich und hat nichts mit "Hexerei" oder Irreversibilität der zugrundeliegenden Mechanik zu tun. Bei einer numerischen Simulation stimmen Iterationspunkt des Algorithmus und zugehörige Zeit i. a. nicht mehr überein. Mit anderen Worten: Ein Algorithmus veranstaltet eine dauernde *Trajektoriensprungerei* und landet gemäß HURLEY stets mit einer so enorm hohen Wahrscheinlichkeit auf einem solchen Trajektorienstück, das in beiden Richtungen zum Gleichgewicht führt, so daß also die vermeintliche Irreversibilität verständlich wird.

Für Untersuchungen durch Computer-Simulationen von solchen komplexen thermodynamischen Systemen, und zwar im Hinblick auf die geschilderte Problemstellung, sind reversible Integrationsalgorithmen unerläßlich. Natürlich lassen sich auch mit solchen Algorithmen die Trajektoriensprünge nicht vermeiden, allerdings finden diese Sprünge geregelt statt, in dem Sinne, daß sie exakt reproduzierbar sind.

Aber abgesehen von diesen tiefsinnigeren Fragestellungen bieten reversible Integrationsalgorithmen auch von einem rein pragmatischen Gesichtspunkt aus betrachtet Vorteile, insbesondere bei der Anwendung auf Systeme mit vielen Freiheitsgraden, wie sie typischerweise in Molekular-Dynamik-Simulationen vorkommen. Es läßt sich nämlich an beliebigen Stellen auf Rückrechnung umschalten, eine Änderung z.B. in Zwangsbedingungen vornehmen, und schließlich wieder vorwärts integrieren. Detailuntersuchungen werden dadurch erheblich erleichtert. Das Integrationsprogramm muß nicht jedesmal neu gestartet werden. Der Phantasie für weitere Anwendungen sind keine Grenzen gesetzt. Schließlich sei schon an dieser Stelle darauf hingewiesen, daß die behandelten Algorithmen einen Zusammenhang zu den sogenannten *symplektischen* Algorithmen aufweisen, die in HAMILTONschen Systemen eine bedeutsame Rolle spielen.

Im 2.Kapitel wird nun zunächst auf einige wichtige Grundbegriffe zu Integrationsalgorithmen eingegangen. Im 3. Kapitel wird dann anhand eines einfachen Beispiels das Konstruktionsprinzip der exakt-reversiblen Integrationsalgorithmen erläutert und der Begriff der *exakten* Reversibilität genauer betrachtet. Hierbei wird der wichtige Unterschied zur Umkehrbarkeit im mathematischen Sinne erläutert. Graphiken ermöglichen einen anschaulichen Zugang zum Mechanismus der

exakt-reversiblen Algorithmen. Im 4. und 5.Kapitel werden erste explizite Beispiele angegeben und die erste Modellrechnungen mit weitgehend exemplarischem Charakter durchgeführt. Auch hier soll, insbesondere durch Graphiken, die Anschaulichkeit der Wirkmechanismen im Mittelpunkt stehen. Im 6.Kapitel sollen schließlich weitgehend mathematische Betrachtung zu den reversiblen Algorithmen angestellt werden. Im Mittelpunkt stehen dabei der Zusammenhang bzw. der Unterschied zwischen der zu lösenden Differentialgleichung und der durch Anwendung des Algorithmus entstehenden Differenzengleichung. Anschließend werden im 7. Kapitel die Ergebnisse von Untersuchungen an zwei exemplarischen dynamischen System angegeben und erläutert. Aus diesen Untersuchungen lassen sich Schlüsse über die Anwendbarkeit der reversiblen Algorithmen ziehen. Ein Gütekriterium in Anwendung auf konservative physikalische Systeme stellt die Erhaltung der Energie durch die Algorithmen dar. Der Energieerhaltung wird daher ein besonderes Augenmerk geschenkt. Eine wichtige Rolle spielt hierbei auch der *Gedächtnisterm*, dessen Größe Aufschluß über die Güte des Algorithmus gibt. Ein ganz spezieller Algorithmus von der hier vorgestellten reversiblen Machart - der sogenannte VERLET-Algorithmus - wird sich als ein gewisser Sonderfall erweisen. Möglicherweise liegt dies daran, daß dieser Algorithmus aus einem sehr fundamentalen Prinzip der klassischen Physik heraus, namentlich dem HAMILTONschen Prinzip der extremalen Wirkung, abgeleitet werden kann. Dieser Ableitung sei das 8. Kapitel gewidmet. Ein weiterer Grund für die besondere Güte des VERLET-Algorithmus könnte in seiner *symplektischen* Eigenschaft zu suchen sein. Der Begriff der Symplektizität soll im 9. Kapitel erläutert werden. Dort wird auch eine neue (Unter-) Klasse von exakt-reversiblen Algorithmen angegeben, namentlich die symplektischen Algorithmen. Die Arbeit schließt mit einer Zusammenfassung und Diskussion im 10. Kapitel.

2. Einige Grundbegriffe zu Integrationsalgorithmen

In dieser Arbeit geht es um Integrationsalgorithmen, d.h. also um Vorschriften zur iterativen Lösung von Differentialgleichungen. Die Differentialgleichungen seien hierbei, falls nicht anders vermerkt, stets zusammen mit einem Anfangswert $\mathbf{x}(t_0) = \mathbf{x}_0$ in der Form

$$d/dt(\mathbf{x}) = \mathbf{f}(\mathbf{x}); \quad \mathbf{x}_0 \tag{2.1}$$

gegeben, d.h. als Anfangswertproblem erster Ordnung, und zwar nicht explizit von der Zeit abhängig. Explizit zeitabhängige Differentialgleichungen können immer in diese Form gebracht werden, wenn man die Zeit als zusätzliche Komponente zum Vektor $\mathbf{x}$ schlägt. Der vektorielle Charakter der Funktionen $\mathbf{x}(t)$ und $\mathbf{f}(\mathbf{x})$ deuten darauf hin, daß es sich im allgemeinen um Systeme von Differentialgleichungen handeln wird. Differentialgleichungen zweiter Ordnung lassen sich durch geeignete Umformung ebenfalls in ein solches System von Differentialgleichungen erster Ordnung bringen. Bei bestimmten Algorithmen kann es allerdings zweckmäßig sein, Differentialgleichungen zweiter Ordnung in ihrer Form zu belassen. Es wird dann an der entsprechenden Stelle darauf hingewiesen.

Eine Möglichkeit, die Differentialgleichung zu charakterisieren, stellt der Fluß $\mathbf{f}^t$ dar, welcher jedem Anfangswert $\mathbf{x}_0$ einen Funktionswert $\mathbf{x}(t)$ zur Zeit t zuordnet. Es gilt also

$$\mathbf{x}(t) = \mathbf{f}^t(\mathbf{x}_0). \tag{2.2}$$

Es ist allerdings nicht immer möglich, den Fluß einer Differentialgleichung anzugeben; dennoch läßt sich anhand dieser Darstellung der Wirkungsmechanismus eines Algorithmus gut verstehen. Wählt man nämlich ein sehr kleines Zeitintervall h, so lassen sich aufgrund elementarer mathematischer Überlegungen immer Näherungen $\mathbf{F}^h$ für den Fluß $\mathbf{f}^h$ angeben, so daß jeder Funktionswert $\mathbf{F}^h(\mathbf{x}_0)$ in der Nähe des wahren Wertes $\mathbf{x}(t_0+h)$ liegt.

Die entscheidende Idee ist nun, den Näherungswert

$$y_1 := y(t_0+h) = F^h(x_0) \tag{2.3}$$

wieder in die Näherungsfunktion F^h einzugeben, um so die nächste Näherungslösung

$$y_2 := y(t_0+2h) = F^h(y_1) \tag{2.4}$$

zu erhalten. In Verallgemeinerung ergibt sich also die Differenzengleichung

$$y_{n+1} = F^h(y_n). \tag{2.5}$$

Betrachten wir nun einige explizite Beispiele, wobei wir einstweilen auf die vektorielle Darstellung verzichten, da die Verallgemeinerung keine Probleme bereit. Hierzu erinnern wir uns zunächst an die TAYLOR-Reihe einer Funktion x(t) um einen Entwicklungspunkt t_0:

$$x(t) = x(t_0) + (t-t_0)\,(d/dt)\,x(t_0) + (1/2)\,(t-t_0)^2\,(d^2/dt^2)\,x(t_0) + \ldots.$$

Aus der TAYLOR-Reihe ergibt sich der Integrationsalgorithmus m-ter Ordnung

$$y_{n+1} = y_n + h\,f(y_n) + (1/2)\,h^2\,f'(y_n) + \ldots + (1/m!)\,h^m\,f^{(m-1)}(y_n), \tag{2.6}$$

wobei f die durch die Differentialgleichung (2.1) definierte Funktion darstellt und $f'(x) = (d/dt)f(x(t))$. Die *TAYLOR-Methode* (2.6) zur Integration von Differentialgleichungen wird im folgenden noch eine entscheidende Rolle spielen.

Bricht man die Taylor-Reihe bereits nach dem zweiten Glied ab, so erhält man den bekannten *EULER-Algorithmus*:

$$y_{n+1} = y_n + h\,f(y_n). \tag{2.7}$$

Es ist dies der einfachste (aber unexakte) Algorithmus. Es handelt sich um einen Algorithmus von erster Ordnung, d.h. der globale Fehler dieses Verfahrens wächst linear mit der Größe von h. Entsprechendes gilt für die TAYLOR-Methode höherer Ordnung, wo der globale Fehler also mit h^m wächst. Das Restglied der TAYLOR-Reihe, $(1/(m+1)!)\,h^{(m+1)}\,f^{(m)}(\mu)$, mit $y_{n-1} < \mu < y_n$, gibt den lokalen Fehler des Algorithmus wieder und wird oft mit der englischen Terminus *Truncation-Error* belegt. Den meisten anderen Algorithmen ist ihre Ordnung nicht so leicht anzusehen. Man erhält sie im Prinzip durch einen Vergleich mit einer TAYLOR-Reihe. Näheres hierzu findet sich in jedem Standardwerk über numerische Methoden (z.B.[6]).

Großer Beliebtheit erfreuen sich die *RUNGE-KUTTA-Methoden*. Das Konstruktionsprinzip dieser Methoden ist, nicht wie bei dem EULER-Verfahren lediglich die Tangente an den letzten Iterationspunkt zu legen und auf ihr nach dem Zeitschritt h den nachfolgenden Iterationswert zu entnehmen, sondern stattdessen mehrere Zwischenschritte auszuführen. Es sei hier exemplarisch das RUNGE-KUTTA-Verfahren vierter Ordnung (RK4) angegeben:

$$x_{n+1} = x_n + h\,\{(1/6)\,k_1 + (1/3)\,k_2 + (1/3)\,k_3 + (1/6)\,k_4\}$$

mit

$$\begin{aligned} k_1 &= f(x_n) \\ k_2 &= f(x_n + h\,(k_1/2)) \\ k_3 &= f(x_n + h\,(k_2/2)) \\ k_4 &= f(x_n + hk_3). \end{aligned} \qquad (2.8)$$

Alle genannten Verfahren sind sogenannte *Ein-Schritt-Verfahren,* d.h. daß die nachfolgenden Iterationspunkte x_{n+1} explizit nur durch die unmittelbaren Vorgänger x_n, nicht aber durch weitere Vorgänger x_i $(i < n)$, bestimmt sind. Weiterhin sei in den obigen Verfahren eine konstante Schrittweite h gewählt, was im allgemeinen nicht notwendig, für die Konstruktion der reversiblen Algorithmen aber unerläßlich ist.

Es sei ausdrücklich betont, daß jeder Algorithmus in Anwendung auf eine bestimmte Differentialgleichung immer auf eine Differenzengleichung führt, deren Lösung eine Punktfolge ist, die möglichst nahe an der wahren Trajektorie durch den gegebenen Anfangswert liegen soll. Wenn nun die Lösung einer Differenzengleichung, die aus einem bestimmten Algorithmus in Anwendung auf eine ganz bestimmte Differentialgleichung entstanden ist, in der Nähe der wahren Trajektorie liegt, so muß das nicht zwangsläufig bei Anwendung auf jede andere Differentialgleichung der Fall sein. Die meisten Algorithmen sind nur in Anwendung auf eine bestimmte Klasse von Differentialgleichungen als gut zu bezeichnen. Ein gewisses Kriterium für die Güte eines Algorithmus ist u.a. die Übereinstimmung von wesentlichen Eigenschaften des Algorithmus mit dem Fluß der Differentialgleichung. Beispielsweise ist der Fluß einer Differentialgleichung, die ein konservatives (d.h. energieerhaltendes) physikalisches dynamisches System beschreibt, zwangsläufig zeitreversibel. Es ist daher einerseits zu erwarten, daß zeitreversible Algorithmen hier besonders gute Dienste tun - natürlich nur, wenn sie auch andere wesentliche Gütekriterien erfüllen -, andererseits kann man auch erwarten, daß eben diese Algorithmen *nur* dort befriedigende Resultate liefern werden.

Abschließend zu diesen grundlegenden Vorbemerkungen zu den Integrationsalgorithmen sei der wichtige Zusammenhang zwischen Differentialgleichung und Differenzengleichung anhand eines konkreten Beispiels erläutert. Wählen wir hierzu

$$(d/dt)\,x = \mu\,x\,,\ x_0 = 1 \qquad (2.9)$$

als zu lösendes Anfangswertproblem, dann führt der EULER-Algorithmus auf die Differenzengleichung

$$x_{n+1} = x_n + \mu\,h\,x_n\,,\ x_0 = 1. \qquad (2.10)$$

Die Differentialgleichung hat die eindeutige Lösung

$$x(t) = c\,e^{\mu t}, \qquad (2.11)$$

wobei der Koeffizient c sich aus dem Anfangswert ergibt:

$$x(0) = c = 1.$$

Die Differenzengleichung hat als eindeutige Lösung

$$x_n = c' r^n, \tag{2.12}$$

wobei sich auch hier mit dem Anfangswert wiederum $c' = 1$ ergibt und für $n = 0$ folgt

$$r = 1 + h \mu. \tag{2.13}$$

Man erhält die Punktfolge $(n h, (1 + h \mu)^n)$ als Näherungslösung für den Graphen $(t, e^{\mu t})$ der wahren Funktion. Mehr über den Zusammenhang von Näherungslösung und wahrer Lösung findet sich in Kap.6.

3. Das Konstruktionsprinzip für die reversiblen Algorithmen

Der vektorielle Charakter der Differentialgleichung wird in diesem Kapitel zunächst unterdrückt. Die Verallgemeinerung auf vektorwertige Funktionen bereitet keine Schwierigkeiten.

Wie oben bereits angedeutet, wollen wir für die folgende Konstruktion von reversiblen Algorithmen einen gegebenen Ein-Schritt-Algorithmus

$$x_{n+1} = F^h(x_n) \tag{3.1}$$

mit konstanter Schrittweite h voraussetzen. Die zu lösende Differentialgleichung sei vom Typ (2.1). Es läßt sich dann durch einen Vorzeichenwechsel im Zeitschritt h ein Rückrechenalgorithmus

$$x'_{n-1} = F^{-h}(x_n) \tag{3.2}$$

angeben, der einer Integration der invertierten Differentialgleichung $(d/dt)x = -f(x)$ mit dem Vorwärtsalgorithmus entspricht. Es sei aber ausdrücklich betont, daß die Iterationswerte x'_{n-1} im allgemeinen nicht mit den Iterationswerten x_{n-1} des Vorwärtsalgorithmus übereinstimmen. Auf diese Weise lassen sich daher im allgemeinen die zurückliegenden Iterationswerte nicht mehr exakt reproduzieren.

Geht man nun aber davon aus, daß der durch Rückrechnung mit dem Ausgangsalgorithmus durch Vorzeicheninvertierung von h gewonnene Iterationswert x'_{n-1} sich nur sehr minimal von dem vorhergehenden Iterationswert unterscheidet, dann dürfte die Differenz $x_{n-1} - x'_{n-1}$ sehr klein sein. Fügt man diese kleine Differenz nach jedem Iterationsschritt dem erhaltenen x_{n+1} hinzu, dann erwartet man, daß sich der Wert immer nur sehr minimal ändert, d.h. der ursprüngliche Algorithmus nur minimal verändert wird. Der so entstandene Zwei-Schritt-Algorithmus lautet also:

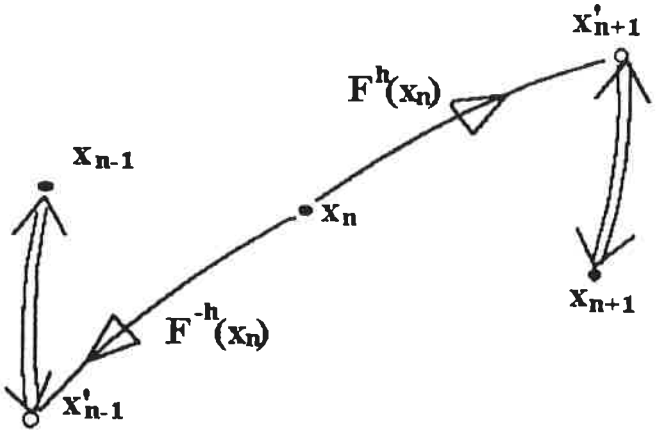
$$x_{n+1} = F^h(x_n) + \{F^{-h}(x_n) - x_{n-1}\}. \tag{3.3}$$

Die in der Klammer stehende Differenz fungiert hier sozusagen als *Gedächtnisterm*, denn, wie man sieht, hat man auf diese Weise einen in der Vertauschung von x_{n+1} und x_{n-1} symmetrischen Algorithmus erzeugt, mit dem sich nun auf einfachste Weise alle zurückliegenden Iterationswerte reproduzieren lassen. Es handelt sich um einen Zwei-Schritt-Algorithmus den man abgekürzt in der folgenden Form darstellen kann:

$$x_{n+1} = F(x_n) - x_{n-1}. \tag{3.4}$$

Um diesen Algorithmus zu starten, muß außer dem gegebenen Anfangswert x_0 zunächst der folgende Iterationswert x_1 mit einem Ein-Schritt-Algorithmus, z.B. mit dem zugrundeliegenden Ausgangsalgorithmus F^h, ermittelt werden. Die folgende Graphik (Abb.2) soll das weitere Vorgehen veranschaulichen:

Abb.2: Skizze zur Funktionsweise der reversiblen Algorithmen.



Ausgehend von einem gegebenen Iterationspunkt x_n wird zunächst durch Anwendung des Grundalgorithmus ein Zwischenwert

$$x'_{n+1} = F^h(x_n)$$

ermittelt. Dann wird von dem Punkt x_n aus mit F^{-h} zurückgerechnet, d.h. der Zwischenwert

$$x'_{n-1} = F^{-h}(x_n)$$

ermittelt, und die Differenz dieses Wertes zum vorhergehenden Iterationspunkt x_{n-1} berechnet. Die durch den Doppelstrich dargestellte Differenz wird sodann am Zwischenpunkt x'_{n+1} abgetragen und dadurch der neue Iterationspunkt x_{n+1} festgelegt. Man erkennt hier besonders schön die Symmetrie in Vorwärts- und Rückwärtsrichtung.

Es sei dieses Schema an einer besonders einfachen Rechnung demonstriert. Als Grundalgorithmus wählen wir hierzu

$$x_{n+1} = (1/2)\, x_n.$$

Diese Differenzengleichung ist sozusagen eine Diskretisierung eines exponentiellen Zerfalls. Nach jeder Zeiteinheit - d.h. in diesem Fall Iterationsschritt - wird ein bestimmter Wert halbiert. Die Umkehrung dieses Sachverhaltes würde eine Verdoppelung des jeweiligen gegebenen Wertes nach jedem Schritt bedeuten:

$$x_{n-1} = 2\, x_n.$$

Nun ist klar, daß es sich hier um eine - im mathematischen Sinne - ohnedies umkehrbare Differenzengleichung handelt. Die Symmetriesierung nach obigem Konstruktionsprinzip scheint hier also zunächst sinnlos zu sein. Dennoch bietet das Verfahren einen entscheidenden

Unterschied zu der mathematischen Umkehrformel, wie man gleich sieht. Bilden wir zunächst die symmetriesierte Differenzengleichung:

$$x_{n+1} = (1/2)\, x_n + 2\, x_n - x_{n-1} = (5/2)\, x_n - x_{n-1}.$$

Nehmen wir nun im Hinblick auf spätere Anwendung auf Integrationsalgorithmen an, daß die Rechnung von einem Computer durchgeführt wird; wobei wir zur Verdeutlichung der auftretenden numerischen Komplikationen eine extreme Rundung betrachten wollen, und zwar in dem Sinne, daß alle Dezimalen gänzlich unterdrückt werden. Die Rechnung mit den Anfangswerten $x_0 = 80$ und $x_1 = 40$ liefert dann nach 10 durchgeführten Schritten die folgenden Werte:

Tabelle 1: Einfaches Beispiel zur Demonstration des Konstruktionsprinzips für die reversiblen Algorithmen.

n	Rechnung	genaues Ergebnis	Gerundetes Ergebnis ($=x_{n+1}$)
0			80
1			40
2	(5/2)*40 - 80	20	20
3	(5/2)*20 - 40	10	10
4	(5/2)*10 - 20	5	5
5	(5/2)*5 - 10	2.5	2
6	(5/2)*2 - 5	0	0
7	(5/2)*0 - 2	-2	-2
8	(5/2)*(-2) - 0	-5	-5
9	(5/2)*(-5) + 2	-10.5	-10
10	(5/2)*(-10) + 5	-20	-20

Rechnet man nun zurück durch Umstellen des Algorithmus, dann erhält man unter jeweiliger Durchführung desselben Rundungsfehlers exakt alle vorhergehenden Werte wieder zurück:

Tabelle 2: Rückrechnung

10			-20
9			-10
8	(5/2)*(-10) + 20	-5	-5
7	(5/2)*(-5) + 10	-2.5	-2
6	(5/2)*(-2) + 5	0	0
5	(5/2)*0 + 2	2	2
4	(5/2)*2 - 0	5	5
3	(5/2)*5 - 2	10.5	10
2	(5/2)*10 - 5	20	20
1	(5/2)*20 - 10	40	40
0	(5/2)*40 - 20	80	80

Mit anderen Worten: Die nach obigem Schema erzeugten Algorithmen rechnen gewissermaßen genauso falsch zurück wie sie vorwärts gerechnet haben und verdienen daher den Namen *exakt-reversibel*. Es sei aber ausdrücklich betont, daß die so konstruierten Algorithmen nicht zwangsläufig exakt in dem Sinne sind, daß sie Näherungslösungen von hoher Qualität erzeugen. Das Attribut *exakt* bezieht sich rein auf die Reversibilität.

Die folgende Tabelle 3 enthält eine analoge Rechnung (Differenzengleichung vom gleichen Typ: Reduktion um 99% nach jedem Schritt), jedoch mit dem Computer ausgeführt. Das angenommene Anfangswertproblem lautet:

$$x_{n+1} = 1.957\, x_n - x_{n-1}; \quad x_0 = 10, \quad x_1 = 9.9$$

Insgesamt wurden 20000 Rechenschritte durchgeführt. Die oberen beiden Spalten der Tabelle enthalten die ersten 20 berechneten Werte aus der Vor- bzw. die letzten 20 aus der Rückwärtsrechnung im sogenannten *Single Precision* Modus. In diesem Modus, wie auch im *Double Precision* Modus (beide werden unten erläutert), sind die durchgeführten Rundungen bei der Rückrechnung nicht mehr exakt reproduzierbar wie man sieht. Dies läßt sich nur mit einem Trick bewerkstelligen, und zwar indem man auf Integerzahlen übergeht. Im Bereich der Integerzahlen führt der Computer Additionen reversibel durch, so daß die Rückrechnung mit obigem Algorithmus (3.3) exakt durchgeführt werden kann. Die unteren beiden Spalten zeigen Rechenwerte aus dieser Rechnung, wobei wiederum die ersten 20 Werte von der Vorwärtsrechnung bzw. die letzten 20 aus der Rückwärtsrechnung wiedergegeben sind. Aus der folgenden Erläuterung zu diesem Trick wird man sofort ersehen, daß dieser zunächst zu gewissen Nachteilen führt, die aber im Prinzip ausräumbar sind und daher keine Einschränkung für das Konzept der reversiblen Algorithmen darstellen.

Tabelle 3: Zum Rundungsproblem

Vorwärts in single precision		**Rückwärts nach 20000 Schritten**	
		0	10.00007
		1	9.900022
2	9.374299	2	9.37427
3	8.44504	3	8.445436
4	7.153553	4	7.153442
5	5.553999	5	5.553851
6	3.71562	6	3.715444
7	1.717476	7	1.717273
8	-.3545233	8	-.3547401
9	-2.41128	9	-2.4115
10	-4.36437	10	-4.364565
11	-6.1297	11	-6.129954
12	-7.631574	12	-7.631755
13	-8.80524	13	-8.80539
14	-9.600281	14	-9.600394
15	-9.98251	15	-9.982581
16	-9.935491	16	-9.935517
17	-9.461246	17	-9.461226
18	-8.580167	18	-8.580103
19	-7.330141	19	-7.330035

Vorwärts mit 'FIX'		**Rückwärts nach 20000 Schritten**	
		0	10
		1	9.89999999
2	9.37429998	2	9.37429998
3	8.445505069999999	3	8.445505069999999
4	7.15355344	4	7.15355344
5	5.55399901	5	5.55399901
6	3.71562262	6	3.71562262
7	1.71747445	7	1.71747445
8	-.35452513	8	-.35452513
9	-2.41128012	9	-2.41128012
10	-4.36435006	10	-4.36435006

11	-6.12975294	11	-6.12975294
12	-7.63157644	12	-7.63157644
13	-8.80524215	13	-8.80524215
14	-9.600282440000001	14	-9.600282440000001
15	-9.98251058	15	-9.98251058
16	-9.93549076	16	-9.93549076
17	-9.46124483	17	-9.46124483
18	-8.58016537	18	-8.58016537
19	-7.33013879	19	-7.33013879
20	-5.76491624	20	-5.76491624
21	-3.95180229	21	-3.95180229

Alle in dieser Arbeit durchgeführten Computerprogramme wurden in QuickBASIC 4.0 (qBASIC) programmiert. Sie befinden sich sämtlich im Anhang. Das oben ausgeführte Programm findet sich unter dem Namen NUMTEST.BAS.

Die elementaren Datentypen von qBASIC sind 2 Typen von Integerzahlen und 2 Typen sogenannter *Floating-Point*-Zahlen. Der *INTEGER*-Typ hat eine Länge von 2 Byte, die *LONG INTEGER*-Zahlen verfügen über 4 Byte. In letzterem Modus lassen sich die ganzen Zahlen im Bereich von -2,147,483,648 bis 2,147,483,647 darstellen.

Die *SINGLE PRECISION* Floating Point-Zahlen haben einen Umfang von 4 Byte. Sie bestehen, wie die *DOUBLE PRECISION* Floating Point-Zahlen, die einen Umfang von 8 Byte haben, aus der Mantisse und dem Exponenten. Die Double Precision-Zahlen haben den Darstellungsbereich

-1.797693134862316 E 308 bis -4.94065 E-324 für negative Zahlen

+4.94065 E-324 bis + 1.797693134862316 E308 für positive Zahlen.

Für die Floating Point-Zahlen gilt eine Genauigkeit von 7 Dezimalstellen im Single Precision Modus und 15 Stellen im Double Precision Modus. Mit dieser Genauigkeit wird jede Teilrechnung ausgeführt. Besteht eine Rechnung, wie z.B. ein Integrationsalgorithmus, aus mehreren Teilrechnungen, so erhält man i.a. für zwei Rechnungen verschiedene Resultate, wenn diese zwar im arithmetischen Sinne gleich sind, aber veränderte Operationsreihenfolgen haben. Wird zum Beispiel eine Multiplikationsreihenfolge geändert, so kann bei Iterationsrechnungen das Ergebnis nach wenigen Schritten schon sehr stark differieren, insbesondere wenn es sich um chaotische Abbildungen handelt. Die folgende Tabelle 4 liefert ein Beispiel für diese Tatsache. Die erste Spalte besteht aus hintereinanderfolgenden Werten der logistischen Abbildung

$$x_{n+1} = 3.86\, x_n\, (1 - x_n),$$

in der angegebenen Reihenfolge multipliziert, während die zweite Spalte wie folgt berechnet wurde:

$$x_{n+1} = (3.86\, (1-x_n))\, x_n.$$

Tabelle 4: Zur Abhängigkeit eines Produkts von der Multiplikationsreihenfolge.

	3.86 x (1-x)	(3.86 (1-x)) x		3.86 x (1-x)	(3.86 (1-x)) x
0	.5	.5			
1	.965	.965	46	.4750422385664435	.4750423527063459
2	.1303715000000001	.1303715000000001	47	.962595645156711	.962595667148443
3	.4376266198727153	.4376266198727153	48	.1389803386566987	.1389802601189795
4	.9499829072027793	.9499829072027794	49	.4619061439169725	.4619059250266874
5	.1834093792497926	.1834093792497923	50	.9593985923768807	.9593985280044406
6	.5781136623725737	.5781136623725728	51	.1503583226234626	.1503585509239834
7	.9414472671978702	.9414472671978706	52	.4931176921235004	.4931183083597699
8	.2127798376944343	.2127798376944327	53	.9648171666158112	.9648171993558514
9	.6465676724895348	.6465676724895314	54	.131027706250302	.1310275887663131
10	.8820791610906616	.8820791610906656	55	.4394974632780341	.4394971286288203
11	.401499886588589	.4014998865885773	56	.9508702501737997	.950870093865552
12	.9275492287596453	.9275492287596363	57	.1803238275817111	.1803243716465998
13	.2593983959699455	.2593983959699751	58	.5705355788816506	.5705369215782747
14	.7415479510133163	.7415479510133712	59	.9457954659516655	.9457947347998448
15	.7397867072144966	.7397867072143941	60	.1978883137925785	.1978908300794812
16	.7430590132668651	.7430590132670547	61	.6126921221596666	.6126979908767495
17	.7369601400291883	.7369601400288325	62	.915979874428162	.9159747686100482
18	.7482605832641612	.7482605832648122	63	.2970684721130201	.297084868656093
19	.7270953955977264	.7270953955964788	64	.8060405486592996	.806066234960704
20	.7659308498114848	.765930849813672	65	.6034692447445753	.6034085548935395
21	.6920238228375701	.6920238228330799	66	.9236752854130704	.9237237491908431
22	.8226696466465836	.8226696466532398	67	.2721271147663918	.2719685916745902
23	.5631133946532599	.5631133946366792	68	.7645654399566998	.7642864725141226
24	.9496244597432195	.9496244597512983	69	.694819794004799	.6953892693217217
25	.1846540824743854	.1846540824463432	70	.8184946567547609	.8176369090549455
26	.5811498358777231	.5811498358094551	71	.5734460529680548	.5755522248632384
27	.9395807579688917	.9395807580116602	72	.9441779143911768	.9429665846883085
28	.2191274028826893	.2191274027375518	73	.203445084217023	.2075931347193618
29	.6604868549679066	.6604868546531997	74	.6255330022303124	.6349631490278577
30	.8655817218164891	.8655817222063978	75	.9041720562550313	.8946899008413093
31	.449111018040448	.449111016940012	76	.3344495029183582	.3636887451911669
32	.9550038024483647	.9550038020160442	77	.8592091070558423	.8932782733960237
33	.1658701434640639	.1658701449826457	78	.4669396351954375	.3679822907441198
34	.5340589424291206	.5340589463462756	79	.9607810673968984	.8977253123493233
35	.960522355380754	.9605223543507967	80	.1454479186042994	.3544042630382043
36	.1463679583510112	.1463679620127491	81	.4797702912909893	.8831752621212091
37	.4822853033999878	.4822853133966763	82	.9634203292982175	.3982621334036028
38	.9637886915640662	.9637886929311876	83	.1360325697974881	.9250467090915968
39	.134714191368345	.1347141864734369	84	.4536569596418694	.2676342390489574
40	.4499458331275473	.4499458193238748	85	.9567099652760076	.7565837511113017
41	.9553290802618	.9553290749278094	86	.1598657894038894	.7108760456494227
42	.1647271546582612	.1647271734079874	87	.5184316544991402	.7933507924130736
43	.5311055800211145	.5311056285511487	88	.9636886580739579	.6328289065998002
44	.9612652295613832	.9612652179076105	89	.1350723175219886	.896896018865996
45	.1437247376710621	.143724779169768	90	.450955256126339	.3569479038041265

Man erkennt, daß die Iterationswerte nach ungefähr 80 Iterationsschritten nichts mehr miteinander zu tun haben. Vgl. hierzu auch [4].

Grundvoraussetzung für eine exakte Rückrechenbarkeit ist somit, daß mindestens Multiplikationen in der Vor- und Rückwärtsrechnung in derselben Weise auftreten, wie das bei

der Konstruktion (3.3) tatsächlich der Fall ist. Lediglich Additionen lassen sich, wie bereits erwähnt, im Integer-Modus exakt umkehren. Im Konstruktionsprinzip (3.3) ist daher in Verallgemeinerung statt der positiven auch eine negative Korrektur möglich, d.h.

$$x_{n+1} = F^h(x_n) - \{F^{-h}(x_n) - x_{n-1}\}. \qquad (3.5)$$

Auch hier unterscheiden sich Vor- und Rückwärtsrechnung nur durch additive Terme. Multiplikative Ausdrücke bleiben identisch.

Wie erwähnt, muß man sich allerdings immer eines Tricks bedienen. Die Vorgehensweise ist wie folgt: Man führt die im Vorwärts- wie im Rückwärtsalgorithmus vorkommende Abbildungsvorschrift $F(x_n)$ (siehe (3.4)) durch, multipliziert den erhaltenen Wert mit 10^m und nimmt davon den abgeschnittenen ganzzahligen Anteil mittels des Befehls "FIX". Die restlichen Rechnungen (Additionen) werden im Integerbereich durchgeführt und sind daher reversibel. Den erhaltenen Iterationswert muß man vor erneuter Eingabe in die Abbildungsvorschrift F durch 10^m teilen. Im Programm NUMTEST.bas (siehe Anhang) ist diese Prozedur für das obige Beispiel (Tabelle 3) durchgeführt. Der Exponent m bestimmt die Rundungsgenauigkeit. Er muß in Konsistenz mit dem Zahlenbereich der Long-Integer-Zahlen gewählt werden. Wenn die Integrationswerte unter 1 bleiben kann bei qBasic m = 8 verwendet werden.

Zur Erinnerung sei hier angegeben, daß die mathematische Bedingung für die Umkehrbarkeit einer Abbildung f(x) lautet, daß die Funktionaldeterminante der Abbildung nirgends verschwindet:

$$\det D(f(x)) \neq 0 \text{ für alle } x.$$

Für die numerische exakte Reversibilität ist diese Bedingung zwar notwendig, aber keineswegs hinreichend, wie in den Rechenbeipielen (Tabellen 3,4) deutlich wurde, da man sich des angegebenen Tricks bedienen muß. Natürlich darf an dieser Stelle nicht verschwiegen werden, daß dieser Trick eine gewaltige Einsschränkung des Zahlenspektrums liefert. Auf den Exponenten einer Zahl muß in diesem Falle ganz verzichtet werden. Allerdings ist dieser Mangel, 9 statt 309 zur Verfügung stehenden Stellen, im Prinzip behebbar, indem man per entsprechende Software den Darstellungsbereich von Zahlen, insbesondere den Integerzahlen, erweitert. Dieses Problem stellt also keine prinzipielle Beschränkung für die exakt reversiblen Algorithmen dar.

4. Explizite Beispiele für exakt-reversible Integrationsalgorithmen

Das Konstruktionsprinzip (3.3) bzw. (3.5) gestattet nun also, aus jedem beliebigen Ein-Schritt-Algorithmus mit konstanter Schrittweite einen exakt-reversiblen Algorithmus zu basteln. Alle Zwei-Schritt-Algorithmen, die sich so zerlegen lassen, daß sie dem Konstruktionsprinzip (3.3) respektive (3.5) entsprechen, werden im folgenden als *Algorithmen FREDKINscher Art*[1] bezeichnet. Als ein erstes konkretes Beispiel für einen exakt-reversiblen Algorithmus von FREDKINscher Art sei als Grundalgorithmus die TAYLOR-Methode (2.6) betrachtet. Wir haben dann

$$F^h(x_n) = x_n + h\, f(x_n) + (1/2)\, h^2\, f'(x_n) + \ldots + (1/m!)\, h^m\, f^{(m-1)}(x_n)$$

[1]Die Grundidee zu dem Konstruktionsprinzip stammt von E. Fredkin (aus einer Mitteilung an O. E. Rössler im Juni 1992).

und

$$F^{-h}(x_n) = x_n - h\, f(x_n) + (1/2)\, h^2\, f'(x_n) + \ldots (-1)^m (1/m!)\, h^m\, f^{(m-1)}(x_n)$$

als Vorwärts- und Rückwärts-Grundalgorithmus. Hierbei bedeutet f wiederum die durch die Differentialgleichung (2.1) definierte Funktion; m ist eine beliebige positive ganze Zahl.

Die Anwendung des Konstruktionsprinzips (3.3) mit dem positiven Gedächtnisterm liefert den exakt-reversiblen Integrationsalgorithmus

$$x_{n+1} = 2\, x_n + h^2\, f'(x_n) + h^4\, f^{(3)}(x_n)/12 + \ldots + 2\, h^{2m}\, f^{(2m-1)}(x_n)/(2m)! - x_{n-1}. \qquad (4.1)$$

In diesem Algorithmus heben sich bei der Addition der Vor- und Rückwärts-TAYLOR-Methode alle ungeraden Terme weg. Für die Funktion f bedeutet dies, daß sie nur durch ihre ungeraden Ableitungen in den Algorithmus eingeht. Diese Tatsache hat Konsequenzen, die noch zu besprechen sind.

Brechen wir den Algorithmus bereits nach dem ersten Glied m = 1 ab, dann erhalten wir den sinnlosen Algorithmus

$$x_{n+1} = 2\, x_n - x_{n-1}.$$

Dieser Algorithmus hängt nur noch von den Anfangswerten der Differentialgleichung ab, nicht aber von dem gegebenen Kraftfeld f. Man erhält daher unabhängig von dem wahren Trajektorienverlauf immer eine Gerade.

Nimmt man den nächsten Term hinzu, so ergibt sich

$$x_{n+1} = 2x_n + h^2\, f'(x_n) - x_{n-1}. \qquad (4.2)$$

Dieser Algorithmus wurde unabhängig von diesem Schema entwickelt und ist in der Literatur unter dem Namen *VERLET-Algorithmus* [5] bekannt. Er wird im weiteren eine größere Rolle spielen und Grundlage zu einigen näheren Untersuchungen sein.

Zunächst sei jedoch der negative Gedächtnisterm (3.5) in der Anwendung auf die TAYLOR Methode betrachtet. Man erhält hier einen Algorithmus mit den ungeraden Termen der TAYLOR Reihe (bzw. mit den geraden Ableitungen der Funktion f), da sich bei der Differenz die geraden Terme wegheben:

$$x_{n+1} = 2\, h\, f(x_n) - h^3\, f''(x_n)/3 - \ldots - 2\, h^{2m-1}\, f^{(2m-2)}(x_n)/(2m-1)! + x_{n-1}. \qquad (4.3)$$

Für den einfachsten Fall m = 1 erhält man hier

$$x_{n+1} = 2\, h\, f(x_n) + x_{n-1}. \qquad (4.4)$$

Auch dieser Algorithmus ist in der Literatur bekannt, und zwar unter dem Namen *Midpoint-Methode*. (siehe z.B.[6]) Es handelt sich letztendlich um den EULER-Algorithmus mit Gedächtnisterm. Auch dieser Algorithmus soll im weiteren Verlauf noch näher untersucht werden.

Bei anderen Ein-Schritt-Algorithmen als Grundalgorithmus, ergeben die Konstruktionsschemata (3.3) oder (3.5) keine so eleganten Ausdrücke wie im Fall der TAYLOR-Reihe. Hier bleibt uns nichts anderes übrig als stur nach dem Konstruktionsschema vorzugehen, d.h. auch so zu programmieren. Als Untersuchungsobjekt wird das RUNGE-KUTTA-Verfahren vierter Ordnung diskutiert werden.

5. Wirkungsweise des VERLET-Algorithmus der Midpoint-Methode und des RK4-Verfahrens mit Gedächtnisterm

Betrachten wir die Wirkung der reversiblen Algorithmen noch etwas genauer. In Abb.2 ist skizziert wie *ein* Iterationsschritt durchgeführt wird. Jetzt wollen wir anhand dieser Skizze den weiteren Verlauf der Algorithmen studieren. Die Abb.3 zeigt den prinzipiellen Verlauf eines Algorithmus von FREDKINscher Art mit positivem Gedächtnisterm für die Anwendung in dem konvergenten Feld einer Exponentialfunktion. Die zugehörige Differentialgleichung lautet

$$(d/dt)x = \mu x, \quad \mu < 0. \tag{5.1}$$

Die punktierten Linienstücke geben das Richtungsfeld der Differentialgleichung wieder. Die Iterationswerte des Algorithmus sind die ausgefüllten Kreise. Ausgehend von irgendeinem dieser Iterationspunkte (x_n) folgen dem Richtungsfeld in Vorwärts- und in Rückwärtsrichtung durchgezogene Linien über einen Zeitschritt (hier: $h = 1$) bis zu je einem kleinen Kreis - diese stellen die Zwischenwerte $F^h(x_n)$ bzw. $F^{-h}(x_n)$ des Grundalgorithmus dar. Die Differenz des rückgerechneten Zwischenwertes zum wahren vorhergehenden Iterationspunkt (Doppellinien) wird dann am vorwärtsgerechneten Zwischenpunkt abgetragen und man erhält den neuen Iterationspunkt x_{n+1}. Es sei darauf hingewiesen, daß diese, sowie die folgenden Überlegungen, allgemeine Gültigkeit für die Algorithmen von FREDKINscher Art mit positivem Gedächtnisterm haben.

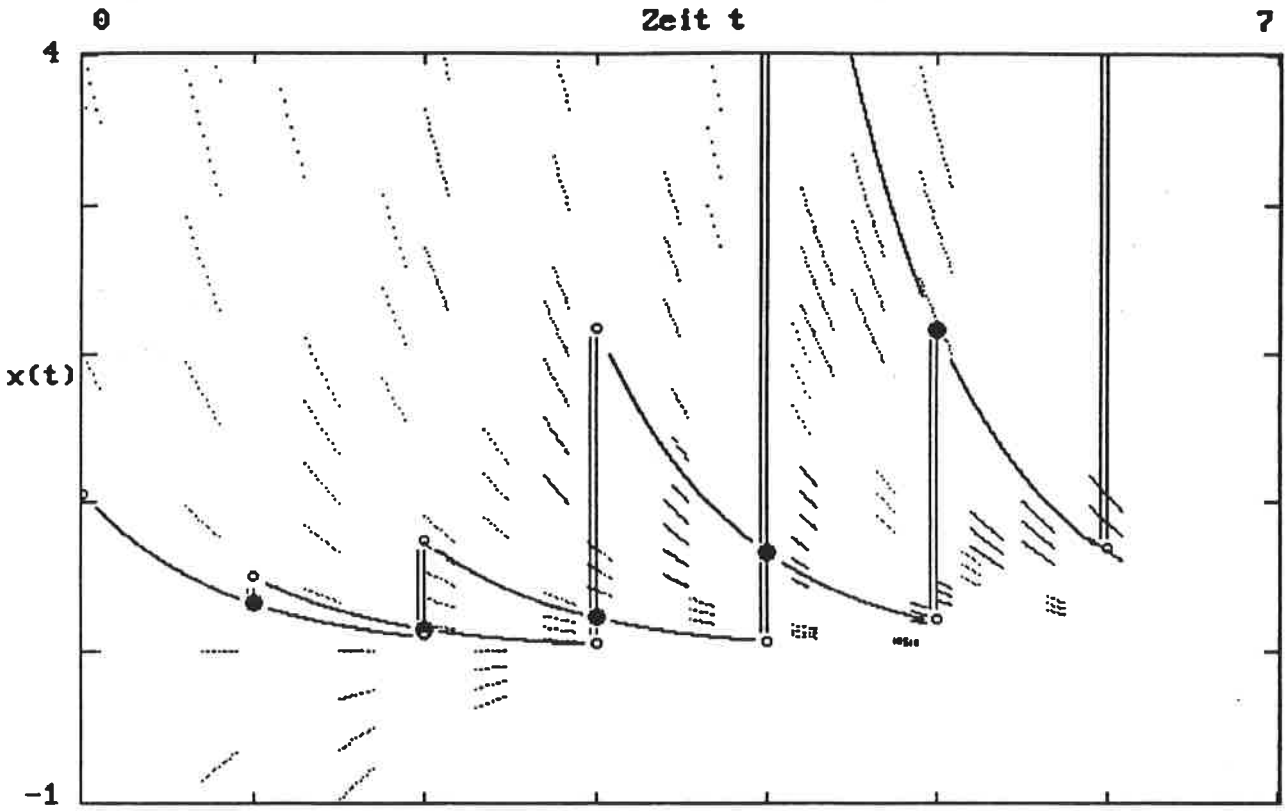


Abb.3: Skizze zur Funktionsweise eines Algorithmus von FREDKINscher Art in einem konvergenten Feld (vgl. Text).

Man erkennt nun an dieser Skizze mit übertrieben großem Zeitschritt h sehr schön, daß die reversiblen Algorithmen im konvergenten Feld zu einem Wegdriften der Trajektorie aus dem lokalen Richtungsfeld führen. Die Differenzterme (Gedächtnisterme) werden immer größer. Die Algorithmen machen hier einen systematischen Fehler. Mathematisch gesprochen schleichen sich hier sogenannte *parasitäre Lösungen* mit ein (diese Begriffsbildung wird später genauer erläutert). Es deutet sich hier bereits die in der Einleitung erwähnte Einschränkung der Anwendbarkeit dieser Algorithmen an, nämlich daß keine dissipativen Systeme damit bearbeitet werden können. In dem skizzierten Fall liegt eine Exponentialfunktion vor, die asymptotisch auf den Fixpunkt null zuläuft. Ein (nicht-reversibler) Standard-Algorithmus würde hier in der Computerrechnung wegen der Endlichkeit und Diskretheit des darstellbaren Zahlenbereichs irgendwann (zu einer endlichen Zeit) zum Iterationswert null führen und in alle Zukunft null bleiben. Zur exakten Reproduzierbarkeit steht der Informationsverlust über den Zeitpunkt des *Nullwerdens* im Widerspruch. Dies erklärt schon rein anschaulich, warum ein solcher Algorithmus nie auf einen Attraktor läuft, selbst wenn das zugrundeliegende dynamische System einen stabilen Attraktor besitzt.

Bei divergenten Feldern ist das Verhalten praktisch umgekehrt. Die Gedächtnisterme konvergieren augenscheinlich gegen null, wie man anhand der Abb.4 schön erkennt. Untersucht wurde hier die Differentialgleichung

$$(d/dt)x = \mu x; \quad \mu > 0. \tag{5.2}$$

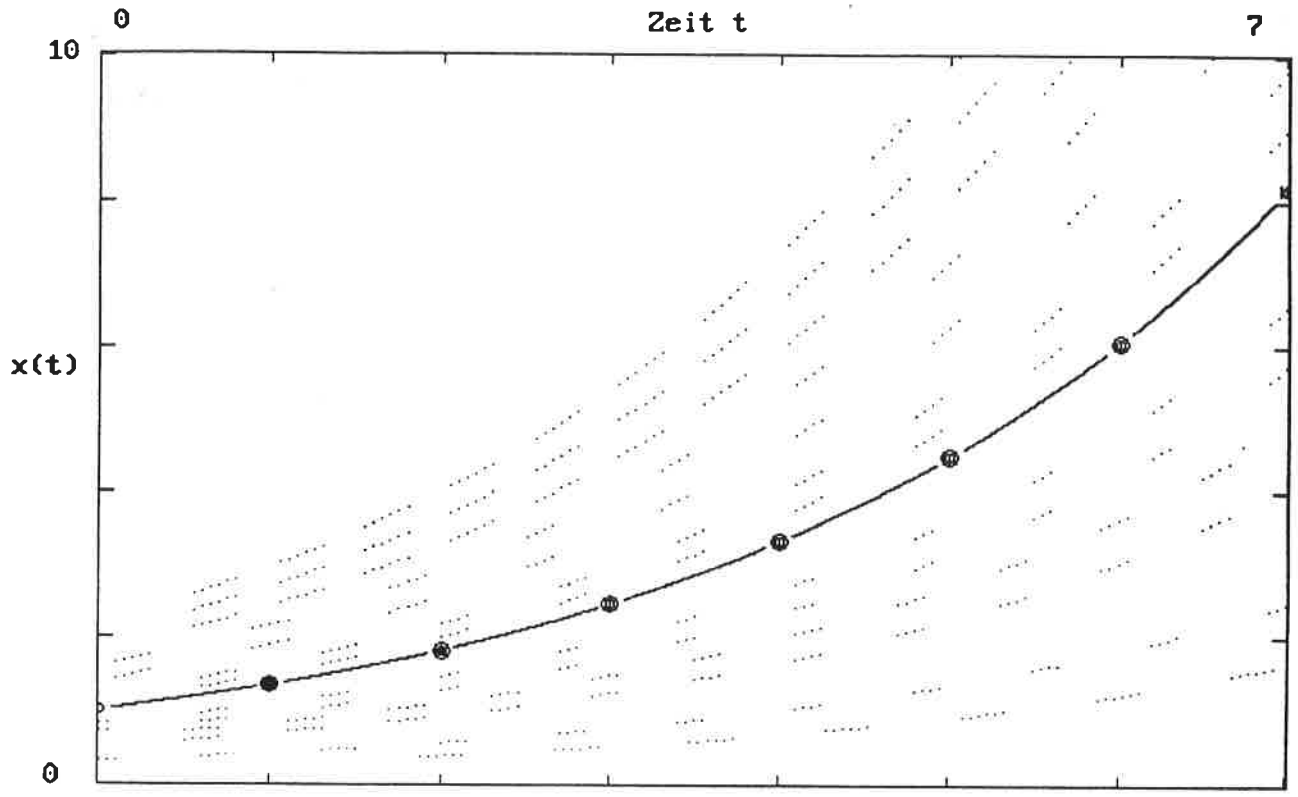


Abb.4: Skizze zur Funktionsweise eines Algorithmus von FREDKINscher Art im divergenten Feld.

Allerdings ist dadurch nicht garantiert, daß die resultierende Lösung gut ist in dem Sinne, daß sie nahe einer wahren Trajektorie verläuft. Vielmehr ist die Lösung höchst empfindlich von den beiden Startwerten x_0 und x_1 abhängig, d.h. von dem ersten Gedächtnisterm der Rechnung. Macht man eine übertrieben falsche Vorgabe dergestalt, daß $x_0 = x_1$ ist, dann stellt sich eine gewisse Symmetrie bzgl. Zeitspiegelung ein. Mit anderen Worten: Die Lösungskurven für die Differentialgleichung (2.1) und die invertierte Differentialgleichung $(d/dt)x = -f(x)$ (was der Gleichung (5.2) entspricht) sind identisch. Abb.5 zeigt dies für die Differentialgleichung (5.1) bzw.(5.2). Beide Richtungsfelder sind skizziert. Die Anfangswerte lauten $x_1 = x_2 = 1$. Die entstandenen Lösungskurven liegen übereinander; sie fallen mit der Kurve $1/2(e^{ct} + e^{-ct})$, also mit dem Mittelwert aus der zeitinvertierten und der nichtinvertierten wahren Trajektorie, zusammen. Diese Mittelwertskurve ist durch die Kreise in dem Diagramm angedeutet.

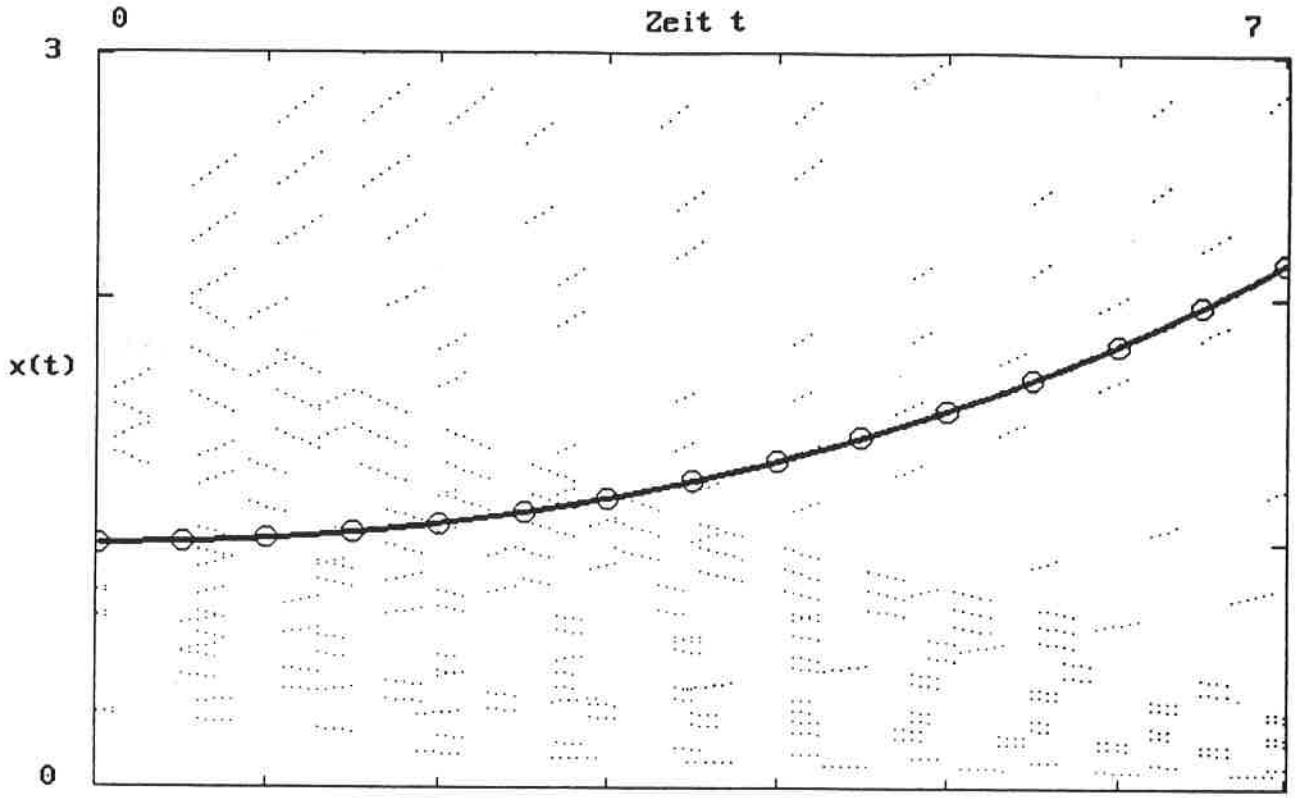


Abb.5: Trajektiorienverlauf für identische Anfangswerte $x_0=x_1$. Zur Erläuterung siehe Text.

Verfolgt man diesen Gedankengang weiter, dann läßt sich das folgende Phänomen plausibel machen: Besitzt die wahre Lösung einen Extremalpunkt, dann müssen zwei aufeinanderfolgende Iterationspunkte, wenn sie in die Nähe dieses Extremalwertes gelangen, zwangsläufig, zumindest annähernd, gleich werden. Das bedeutet dann aber konkret, daß von da an die Iterationskurve zeitspiegelsymmetrisch zur bisherigen Kurve verlaufen muß. Das dies tatsächlich so ist, sieht man anhand Abb.6, wo wiederum die Differentialgleichung (5.2) für die abfallende Exponentialfunktion untersucht wurde. Miteingezeichnet sind auch die Zwischenabbildungen $F^h(x_n)$ und $F^{-h}(x_n)$ des Grundalgorithmus. Man erkennt, daß diese Abbildungen zu immer größeren Differenztermen führen. Mehr zu diesem Phänomen im folgenden Kapitel über die mathematische Analyse der Algorithmen.

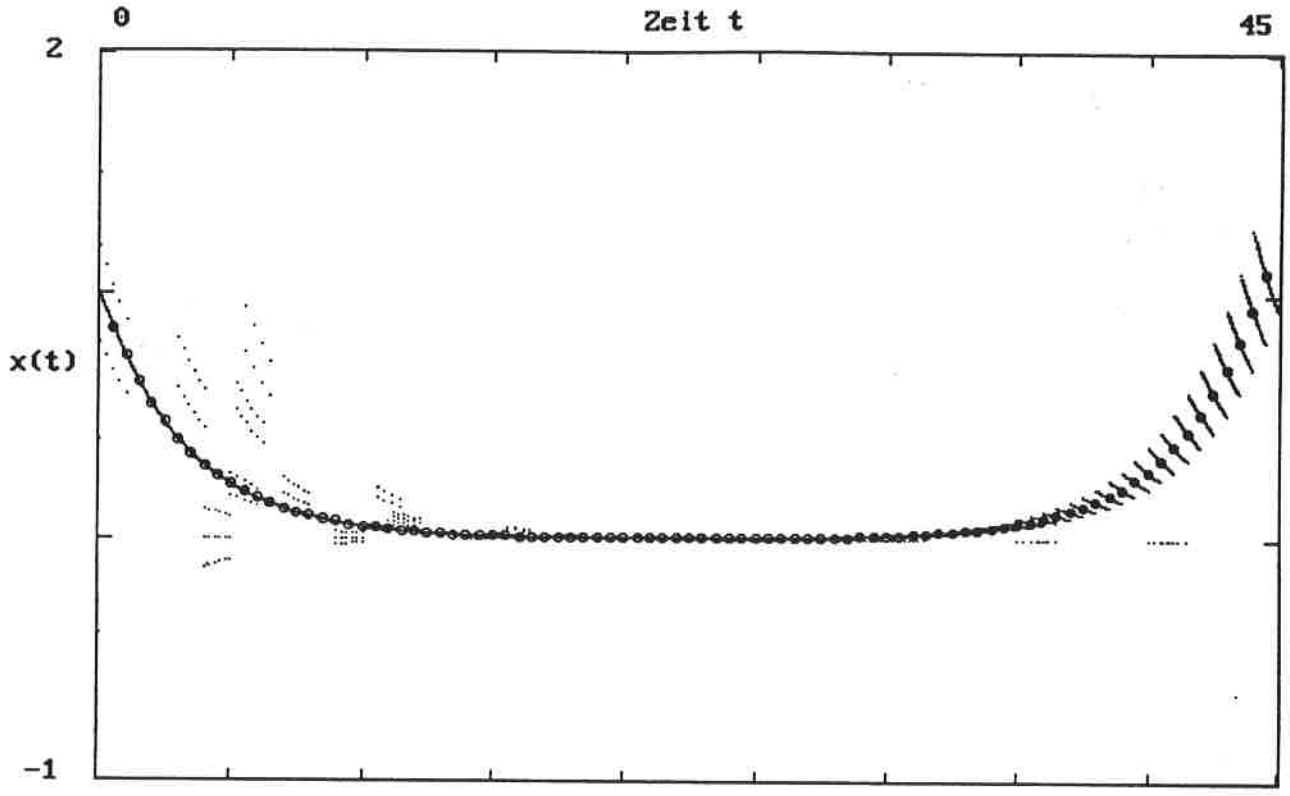


Abb.6: Zur Erläuterung der Symmetrie der Trajektorien bzgl. Zeitspiegelung mit Symmetriezentrum in Extremalpunkten.

Eine weitere Konsequenz obiger Überlegung führt nun zu einem Phänomen, das als typisch für die Algorithmen vom FREDKINschen Typ zu bezeichnen ist. Obiges Phänomen läßt sich nämlich auch fogendermaßen deuten: Besitzt eine Differentialgleichung einen stabilen Fixpunkt, wie z.B. die Null bei der oben untersuchten Differentialgleichung, dann wird dieser Fixpunkt bei Anwendung der exakt-reversiblen Integrationsalgorithmen instabil. Nimmt man nun eine Differentialgleichung (zunächst mit nur einem Freiheitsgrad), die zwei Fixpunkte hat und wählt einen Anfangswert zwischen diesen beiden Fixpunkten, dann erhält man das beachtenswerte Resultat, daß die berechnete Trajektorie zwischen diesen Fixpunkten hin- und herpendelt, und zwar im Rahmen der Rechengenauigkeit des Computers nahezu periodisch. Als Beispiel für dieses Phänomen möge die logistische Differentialgleichung dienen:

$$(d/dt)x = x - x^2. \qquad (5.3)$$

Abb.7 zeigt das Ergebnis der Integration der logistischen Differentialgleichung mit einem Algorithmus von FREDKINscher Art.

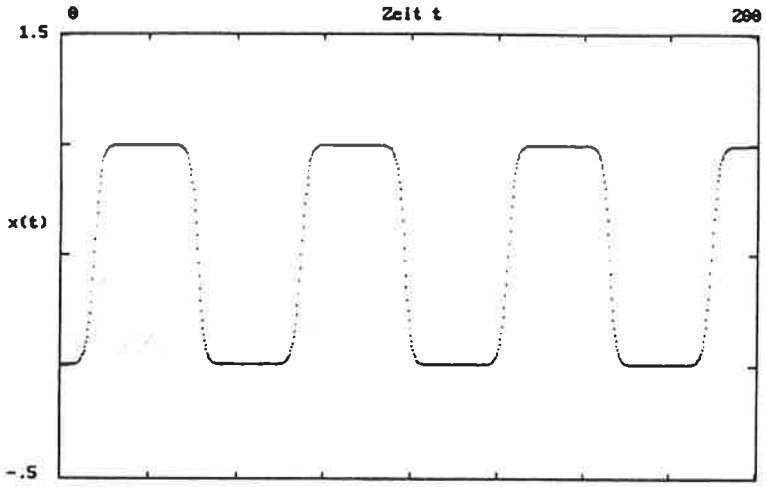


Abb.7: Zur Erläuterung der Oszillation zwischen zwei Fixpunkten - hier die logistische Differentialgleichung mit dem stabilen Fixpunkt $x=1$ und dem instabilen Fixpunkt $x=0$. Der Startwert ist $x_0=.0001$, der Zeitschritt $h=.25$.

Dasselbe Resultat läßt sich übrigens auch mit der logistischen Map, d. h. mit der Differenzengleichung

$$x_{n+1} = 2\, x_n\, (1 - x_n) \tag{5.4}$$

herbeiführen. Man braucht hierzu nur die (zweideutige) Umkehrabbildung

$$x_{n-1} = 1/2 \mp (1 - 2\, x_n)^{1/2}/2 \tag{5.5}$$

z.B. mit dem negativen Vorzeichen als Rückwärts-Zwischenalgorithmus zu wählen und erhält nach Anwendung der Konstruktionsformel (3.3) die exakt reversible Abbildung

$$x_{n+1} = 2\, x_n\, (1 - x_n) + 1/2 + (1 - 2\, x_n)^{1/2}/2 - x_{n-1}. \tag{5.6}$$

Die logistische Map besitzt die beiden Fixpunkte $x=0$ und $x=1/2$, wobei letzterer stabil, $x=0$ dagegen ein instabiler Fixpunkt ist.

Gerade an diesen beiden Fixpunkten ist die logistische Map aber nicht bijektiv ergo nicht umkehrbar. Ansonsten dürfte sich nämlich zwischen (5.4) und (5.6) nicht der geringste Unterschied ergeben. Indiesem Fall aber wird verständlich, daß auch hier die erzeugte Punktfolge zwischen Vorwärts- und Rückwärts-Map hin- und herpendelt, wobei die Umkehrung jeweils bei der Annäherung an die Fixpunkte stattfindet.

Trägt man die Iterationpunkte x_n auf der Ordinate ab und führt auf der Abszisse eine äquidistante Zeitsequenz ein, dann ergibt sich ein Bild, wie es auf der Titelseite dieser Arbeit zu sehen ist. Die äquidistanten Zeitschritte sind hierbei sehr klein gewählt, so daß sich ein etwas bizarres Muster ergibt, dessen Theorie zurückgestellt wird.

Die bisherige Untersuchung läßt sich schlagwortartig wie fogt zusammenfassen: Die Algorithmen von FREDKINscher Art verhindern wegen ihrer Rückrechenbarkeit ein Hineinlaufen in asymptotische Limespunkte oder Fixpunkte. Diese werden instabil und sorgen für eine Zeitspiegelung, d.h., die Algorithmen integrieren dadurch quasi die zeitinvertierte Differentialgleichung.

Es muß allerdings darauf aufmerksam gemacht werden, daß nicht immer ein so elegantes Verhalten wie im Falle der logistischen Gleichung auftritt, wo die Tajektorie zwischen den beiden Fixpunkten oszilliert. Tatsächlich kann die Lösungskurve auch in den Aussenbeich gelangen und divergiert dann gegen Unendlich. Dies hängt gewissermaßen vom Zufall, sprich davon ab, wohin die entscheidende *Korrektur* am Scheitelpunkt führt. Durch Variieren des Zeitschrittes h lassen sich beide Resultate herbeiführen.

Gehen wir nun an die Untersuchung der exakt-reversiblen Algorithmen mit *negativem* Gedächtnisterm (3.5). Die Anwendung auf die TAYLOR-Reihe ergibt, wie bereits erwähnt, im einfachsten Falle die sogenannte Midpoint-Methode (4.4). Dieser, wie auch der negativ korrigierten RK4-Methode soll der Rest dieses Kapitels gewidmet sein. Wiederum gelten die folgenden Ausführungen zunächst für beide Algorithmen.

Wir starten wieder mit einer erklärenden Skizze zur Funktionsweise dieses Algorithmustyps. (Auf eine Skizze im divergierenden Feld wird verzichtet, da sich keine prinzipiellen Unterschiede zu den positiv korrigierten Algorithmen ergeben.) Für den Fall von konvergierenden Feldern verhält sich die Drift, bzw. die Trajektoriensspringerei aber anders. Die aufeinanderfolgenden Iterationspunkte springen gewissermaßen zwischen zwei divergierenden Tajektorien hin- und her. In der graphischen Lösung einer Differentialgleichung sieht dies so aus, als ob die Trajektorie aufspalte. Abb.8 zeigt den Mechanismus des Algorithmus, und Abb.9 stellt die Lösungskurve für die Differentialgleichung (5.1) der Exponentialfunktion dar. Insbesondere in Abb.9 erkennt man, daß sich analog zu den Algorithmen mit positivem Gedächtnisterm eine parasitäre Lösung einschleicht, die sich in der Nähe der Nullstelle, bzw. des asymptotischen Fixpunkts, bemerkbar macht und schließlich dominiert. Anders als bei den positiven Fällen aber alternieren hier die aufeinanderfolgenden Iterationspunkte zwischen zwei Kurven, die Lösungen für die invertierte Differentialgleichung sind; wobei die eine oberhalb die andere unterhalb der Asymptote verläuft.

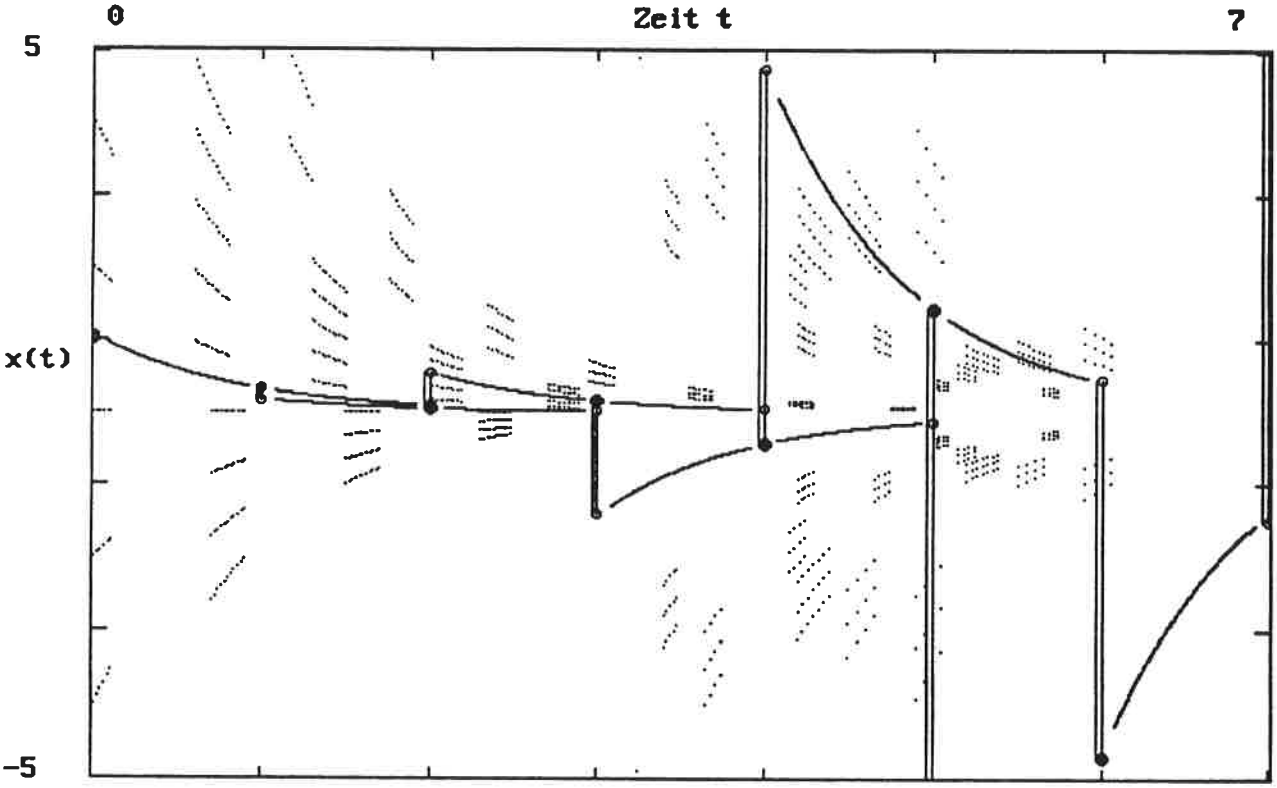


Abb.8: Funktionsweise der Algorithmen mit negativem Gedächtnisterm im konvergenten Feld.

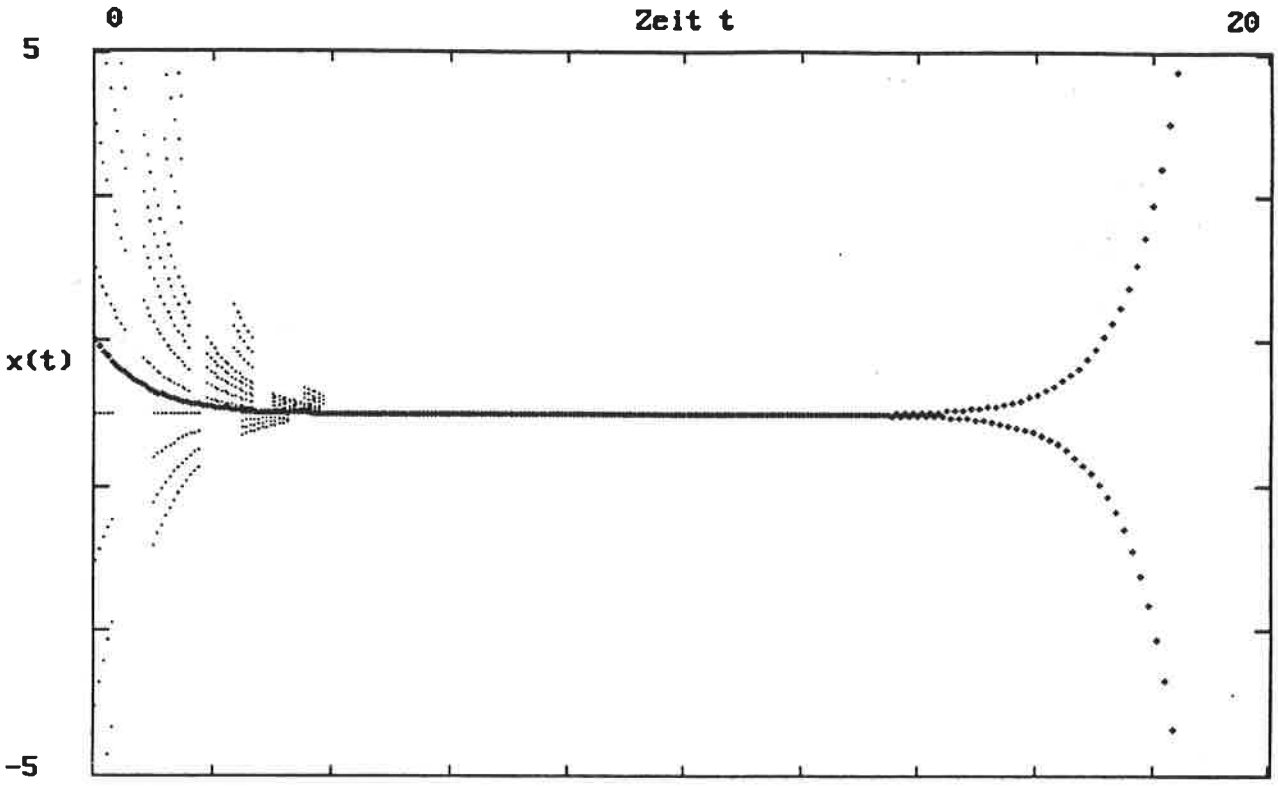


Abb.9: Lösungskurve eines Algorithmus FREDKINscher Art im Feld einer Exponentialfunktion. Zeitschritt h= 0.7.

In den weiteren Untersuchungen werden die bisher angegebenen Eigenschaften der Algorithmen von FREDKINscher Art, die sie gewissermaßen unter Extrembedingungen zeigen, nochmals aufgegriffen. Das Verhalten des Gedächtnistermes kann gewissermaßen als Prüfinstanz eingesetzt werden. Denn eines ist bisher klar: Ein Einsatz von exakt reversiblen Integrationsalgorithmen vom angegebenen Typ ist bei dissipativen Systemen auf eine sehr kurze Integrationszeit beschränkt. Eine stetige Kontrolle des Gedächtnistermes könnte aber zum Einsatz bei schwach dissipativen bzw. konservativen Systemen verhelfen. Insbesondere wenn die konservativen Systeme chaotisch sind, könnten die Gedächtnisterme als Kriterium herangezogen werden, ob die Integration noch mit einer gewissen Güte verläuft, da auch hier lokale Konvergenzen und Divergenzen auftreten.

Im folgenden Kapitel sollen zunächst die aufgezeigten Eigenschaften in einen mathematischen Rahmen gebracht werden. Explizite Beispiele werden herangezogen, um das beobachtete Wegdriften von der wahren Lösung auch mathematisch verstehen zu können.

6. Die parasitären Lösungen der Algorithmen von FREDKINscher Art (linearer Fall). Eine mathematische Analyse

An dieser Stelle greifen wir die bereits erwähnte Tatsache wieder auf, daß die Integrationsalgorithmen nichts anderes als Differenzengleichungen darstellen. Diese Differenzengleichungen sind diskrete Approximationen an die Differentialgleichungen. Die Lösungen der zugehörigen Anfangswertprobleme sind Punktfolgen, die möglichst nahe an den kontinuierlichen Lösungsfunktionen der entsprechenden kontinuierlichen Anfangswertprobleme liegen sollten.

Erfüllen die Differentialgleichungen bestimmte Bedingungen, wie z.B. die LIPSCHITZ-Bedingung, so besitzen die zugehörigen Anfangswertprobleme eindeutige Lösungen. Die Theorie der linearen Differentialgleichungen besagt, daß die Menge der linear unabhängigen Lösungen durch die Ordnung der Differentialgleichung gegeben ist. Dasselbe gilt auch für lineare Differenzengleichungen. Hierin liegt nun genau das Problem der parasitären Lösungen für Multi-Step-Algorithmen, insbesondere der Algorithmen von FREDKINscher Art. Diese Algorithmen führen nämlich auf Differenzengleichungen zweiter Ordnung, deren Lösungsmannigfaltigkeiten zweidimensional sind. Die Zahl linear unabhängiger Lösungen ist für die Differenzengleichung also immer doppelt so groß wie die Zahl der linear unabhängigen Lösungen der zugehörigen Differentialgleichung.

Ein erstes Beispiel zum Zusammenhang von Differentialgleichung und Differenzengleichung ist bereits am Ende von Kapitel 2 gegeben worden. Hier führte der EULER-Algorithmus in Anwendung auf die einfache Differentialgleichung (2.9) auf die Differenzengleichung (2.10). Diese Gleichung ist vom allgemeinen Typ

$$x_n = a_n x_{n-1} + f_n, \tag{6.1}$$

wo a_n und f_n gegebene beliebige Zahlenfolgen sind. Gleichung (6.1) hat die eindeutige Lösung (vgl. z.B. [7])

$$x_n = a_n a_{n-1} \ldots a_0 (x_0 + f_1/a_1 + f_2/(a_1 a_2) + \ldots + f_n/(a_n a_{n-1} \ldots a_1). \tag{6.2}$$

x_0 ist hierbei der gegebene Anfangswert. Für die Gleichung (2.10) ist $a_n = (1 + \mu h)$ und $f_n = 0$ für alle n, so daß man in Übereinstimmung mit der bereits in Kapitel 2 angegebenen Gleichung (2.13) die Lösung

$$x_n = (1 + h \mu)^n x_0$$

erhält. Der globale Fehler dieser Approximation kann mit

$$F_n = x_0 (e^{\mu h} - (1 + \mu h)^n)$$

angegeben werden.

Greifen wir zunächst die Midpoint-Methode (4.4) als ein konkretes Beispiel für einen Algorithmus von FREDKINscher Art heraus und untersuchen diese ebenfalls in Anwendung auf die Differentialgleichung (2.9). Wir erhalten die Differenzengleichung

$$x_{n+1} = 2 h \mu x_n + x_{n-1}. \tag{6.3}$$

Da eine weitgehende Übereinstimmung mit dem Grundalgorithmus - hier der EULER-Algorithmus - erwartet wird, ist ein Potenzansatz der Form $x_n = r^n$ für eine Lösung sinnvoll. Einsetzen in die Gleichung (6.3) mit $n = 1$ ergibt eine Bedingungsgleichung für die Existenz einer solchen Lösung. Man erhält die zweidimensionale Lösungsmannigfaltigkeit

$$x_n = c_1 r_1^n + c_2 r_2^n,$$

mit

$$r_1 = h\,\mu + (1 + h^2\mu^2)^{1/2}; \quad r_2 = h\,\mu - (1 + h^2\mu^2)^{1/2}. \tag{6.4}$$

Die Koeffizienten c_1 und c_2 ergeben sich durch die Anfangsbedingungen x_0 und x_1 mittels der Gleichungen

$$x_0 = c_1 + c_2$$

und

$$x_1 = c_1\, r_1 + c_2\, r_2,$$

zu

$$c_1 = (x_1 - r_2 x_0)/(r_1 - r_2); \quad c_2 = (x_0\, r_1 - x_1)/(r_1 - r_2). \tag{6.5}$$

Entwickelt man r_1 und r_2 nach Potenzen von $h\mu$, so erhält man in der ersten Näherung für r_1 die Lösung (2.13) der EULERschen Differenzengleichung und für r_2 die Lösung ((2.13) mit -h statt +h) der Rückwärts-EULER-Gleichung. Beide Lösungen sind hier also in der Gesamtlösung der Midpoint-Methode vertreten. Mit welchem Gewicht diese Lösungen vertreten sind, betimmen die Koeffizienten c_1 und c_2, also die Anfangsbedingungen. Betrachten wir hierzu den Fall, daß x_1 durch den exakten Wert der analytischen Lösung der Differentialgleichung gegeben sei. O.B.d.A. wählen wir $x_0 = 1$, dann ist $x_1 = e^{h\mu}$. Mit diesen Werten erhält man für die Koeffizienten

$$c_1 = (e^{h\mu} - r_2)/(2\,(1 + h^2\mu^2)^{1/2})$$

und

$$c_2 = (r_1 - e^{h\mu})/(2\,(1 + h^2\mu^2)^{1/2}).$$

Entwickelt man diese Koeffizienten ebenfalls nach Potenzen von $h\mu$, dann erhält man für c_1 gleich eins (bis auf ein Restglied, das mit $h^2\mu^2$ verschwindet); c_2 wird null bis auf ein Restglied, welches in $h^3\mu^3$ verschwindet:

$$\begin{aligned} c_1 &= 1 + O(h^2\mu^2), \\ c_2 &= O(h^3\mu^3). \end{aligned} \tag{6.6}$$

Vergleiche hierzu [6].

Dies erklärt für den Fall $\mu < 0$ (abfallende Exponentialfunktion), daß zwar zunächst die Vorwärtslösung dominiert, sich nach einer genügend großen Anzahl von durchgeführten Iterationsschritten schließlich aber eine Dominanz der Rückwärtslösung einstellt, da ja die Vorwärtslösung in dem betrachteten Beispiel gegen null strebt. Da ein anderer beliebiger Anfangswert x_0 nur zu einer Multiplikation beider Koeffizienten mit diesem Startwert führt, ändert sich an der Allgemeinheit der Argumentation nichts. Für $\mu > 0$ strebt die Rückwärtslösung selbst gegen null und erklärt daher, daß solche divergenten Fälle eine vernünftige Lösung haben. Beachtenswert ist weiterhin, daß das in c_2 auftretende Restglied immer

negativ ist. Das erklärt das alternierende Verhalten der Midpoint-Methode nun auch aus der mathematischen Perspektive.

Die Verallgemeinerung für reversibel gestaltete TAYLOR-Methoden mit negativem Gedächtnisterm liegt nun natürlich auf der Hand. Die Vorwärts-TAYLOR-Methode liefert in Anwendung wiederum auf die Exponentialfunktion die Näherungslösung

$$x_n = x_0 [1 + h\mu + (1/2)h^2\mu^2 + \ldots (1/m!)h^m\mu^m]$$

und für die Rückwärts-TAYLOR-Methode die Lösung

$$x_n = x_0 [1 - h\mu + (1/2)h^2\mu^2 - \ldots (-1)^m (1/m!)h^m\mu^m]$$

Der daraus konstruierte exakt reversible Algorithmus beinhaltet wieder beide Lösungen in Superposition mit entsprechenden Koeffizienten. Obige Rechnung verläuft dann analog, nur mit etwas umfangreicheren Termen. Bemerkenswert ist hierbei die Tatsache, daß das Einschleichen der Rückwärtslösung mit derselben Dominanz stattfindet, d.h. unabhängig von der Ordnung der Taylorreihe ist. Durch höhere Ordnung läßt sich die Genauigkeit des Vorwärts- wie des Rückwärtsanteiles steigern, nicht aber beinflussen, wann welcher Anteil dominiert. Mit anderen Worten: Der systematische Fehler der Algorithmen von FREDKINscher Art ist von der Ordnung unabhängig.

Für die Algorithmen mit positivem Gedächtnisterm läßt sich obige Analyse entsprechend durchführen. Auch hier schleichen sich parasitäre Lösungen ein - die auftretenden Rückwärtslösungen. Betrachten wir hierzu den VERLET-Algorithmus (4.2). Dieser Algorithmus setzt sich aus der Vorwärts- und Rückwärts-TAYLOR-Methode bis zum jeweils quadratischen Glied zusammen. Die Lösungen hierfür sind, in Anwendung wiederum auf die Exponentialfunktion, durch

$$x_n = (1 + h\mu + h^2\mu^2/2)^n \quad \text{für TAYLOR-Vorwärts}$$

bzw. (6.7)

$$x_n = (1 - h\mu + h^2\mu^2/2)^n \quad \text{für TAYLOR-Rückwärts}$$

gegeben.

Ein Potenzansatz für die Lösung des VERLET-Algorithmus führt auf die Lösungsmannigfaltigkeit

$$x_n = c_1 r_1^n + c_2 r_2^n$$

mit

$$r_1 = 1 + h^2\mu^2/2 + h\mu(1 + h^2\mu^2)^{1/2},$$

$$r_2 = 1 + h^2\mu^2/2 - h\mu(1 + h^2\mu^2)^{1/2}.$$

Mit r_1 und r_2 lassen sich nach einer TAYLOR-Entwicklung in der 1.Näherung die obigen Lösungen (6.7) reproduzieren. Für die Koeffizienten c_1 und c_2 gelten wiederum die Gleichungen (6.5) und nach Einsetzen von r_1 und r_2, sowie den Anfangswerten $x_0=1$ und $x_1=e^{h\mu}$ ergeben sich dieselben Werte wie in den Gleichungen (6.6), mit dem kleinen aber wichtigen Unterschied, daß das Restglied für c_2 stets positiv ist und daher nicht zu alternierender Trajektoriensspringerei führt. Wählt man für die Anfangswerte $x_0=1$ und $x_1=1$, so werden die Koeffizienten bis auf kleine Restglieder beide 1/2, was die in Kapitel 5 beobachtete Symmetrie (vgl. Abb.5) erklärt.

Mit dieser Untersuchung hat man nun im Prinzip die Wirkung des VERLET-Algorithmus sowie der Midpoint-Methode auch auf das lineare Differentialgleichungssystem

$$(d/dt)\mathbf{x} = A\,\mathbf{x} \tag{6.8}$$

erfaßt, hierbei $\mathbf{x} = (x_1,x_2,\ldots,x_n)^T$ darstellt und A eine n-dimensionale Matrix mit nichtverschwindender Determinante. Allerdings befinden sich hierunter auch Spezialfälle, die zu periodischen Lösungen führen. Das einfachste Beispiel ist durch das konservative zweidimensionale System

$$\begin{aligned}(d/dt)x &= p\\ (d/dt)p &= -x\end{aligned} \tag{6.9}$$

gegeben. Dieses Differentialgleichungssystem ist äquivalent zur Differentialgleichung zweiter Ordnung

$$(d^2/dt^2)x = -x. \tag{6.10}$$

Sie besitzt die Sinus- und Cosinusfunktion als linear unabhängige Lösungen. Mit den beiden Anfangswerten $x_0 = 0$ und $(d/dt)x_0 = p_0 = 1$ erhält man den reinen Sinus als Lösung für den Ort x(t).

Eine Anwendung des VERLET-Algorithmus liefert für die x-Koordinate die Rekursionsgleichung

$$x_{n+1} = 2\,x_n - h^2\,x_n - x_{n-1}. \tag{6.11}$$

Eine explizite Lösung auch der Impulskoordinate p(t) ist hier nicht unbedingt erforderlich. Auch in der Differentialgleichung selbst sind Ort und Impuls unabhängig. Nach einem Vorschlag von VERLET [5] lassen sich die Impulse approximativ durch

$$p_n = (x_{n+1} - x_{n-1})/(2\,h)$$

im Nachhinein berechnen. Man hat dann also die Differentialgleichung zweiter Ordnung (6.10) durch die Differenzengleichung (6.11), ebenfalls von zweiter Ordnung, ersetzt. Dies erklärt, warum hier keine parasitären Lösungen auftreten. Anders formuliert: Vorwärts und Rückwärtslösung sind hier wegen der Zeitspiegelungsinvarianz identisch gleich, so daß sie zu einer einzigen Lösung verschmelzen. Außerdem führt die Reversibilität zu einer Stabilisierung der Lösung, wie man an der folgenden genauen Analyse erkennt:

Der übliche Potenzansatz $x_n = r^n$ für eine spezielle Lösung der Differenzengleichung (6.11) führt wie immer auf eine quadratrische Gleichung für r, in diesem Fall mit den Lösungen

$$r_{1,2} = 1 - h^2/2 \mp \{(2 - h^2)^2 - 4\}^{1/2}/2.$$

Der wichtige Unterschied zu den bisherigen Fällen, daß nämlich die Diskriminante negativ wird, bedeutet nichts anderes, als daß r_1 und r_2 zueinander konjugiert komplexe Zahlen sind. Definiert man $r_1 =: r$ und $r_2 = r^*$ (konjugiert komplex), dann lautet die allgemeine Lösung für die Differenzengleichung (6.11)

$$x_n = r^n (x_1 - r^* x_0)/(r - r^*) + r^{*n} (x_0 r - x_1)/(r - r^*)$$

$$= \frac{x_1}{i((2 - h^2)^2 - 4)^{1/2}} (r^n - r^{*n}).$$

wobei der Einfachheit halber aber o.B.d.A. $x_0 = 0$ eingesetzt wurde. Dieser Ausdruck läßt sich durch Anwenden der EULERschen Formeln für die komplexen Zahlen umformen zu

$$x_n = \frac{2 x_1}{((2-h^2)^2 - 4)^{1/2}} \sin(n \varphi).$$

Hierbei ist der Winkel φ gegeben durch $\tan z = ((2 - h^2)^2 - 4)^{1/2}/(2 - h^2)$. Wichtig an diesem Ausdruck ist, daß die Amplitude von x_n unabhängig von n ist und die Ungenauigkeit der iterativen Lösung sich daher in einer reinen Phasendrift gegenüber der wahren Lösung bemerkbar macht. Die Energiegröße $E = x_n^2 + p_n^2$ bleibt daher praktisch konstant. Diese hervorragende Stabilität konnte in einer numerischen Rechnung bestätigt werden. Vergleiche hierzu auch die Befunde von [8] zur Stabilität von zeitsymmetrischen Algorithmen.[2]

Eine entsprechende Rechnung läßt sich auch für die Midpoint-Methode ausführen. Hier erhält man in Anwendung auf die Differentialgleichung (6.9) das Gleichungssystem

$$x_{n+1} = 2 h p_n - x_{n-1}$$

$$p_{n+1} = - 2 h x_n - p_{n-1}.$$

Am besten wählt man hier eine komplexe Darstellung und schreibt $z_n = x_n + i p_n$. Das Gleichungssystem geht dann über in die komplexe Differenzengleichung

$$z_{n+1} = 2 h e^{-i \pi/2} z_n + z_{n-1}$$

und liefert mit dem Potenzansatz $z_n = r^n$ die charakteristische Gleichung

---

[2]Vor einer voreiligen, verallgemeinernden Schlußfogerung muß allerdings gewarnt werden. In Kap. 7 werden genauere Untersuchungen erörtert werden, deren Ergebnisse eine Verallgemeinerung in Frage stellen.

$$r^2 = 2\,h\,e^{-i\,\pi/2}\,r + 1.$$

Diese Gleichung hat die Lösungen

$$r_{1,2} = -i\,h \mp (1 - h^2)^{1/2}.$$

Es gilt auch hier wieder

$$|r_{1,2}| = 1,$$

so daß auch diesesmal die Amplitude der allgemeinen Lösung gemäß

$$z_n = r_2^n\,(z_1 - r_1 z_0)/(r_1 - r_2) + r_2^n\,(r_2 z_0 - z_1)/(r_1 - r_2)$$

unabhängig von n wird. Interessant ist der Zusammenhang

$$r_1^n = e^{in\varphi};\quad r_2^n = e^{in\pi}\,e^{-in\varphi} = (-1)^n\,e^{-in\varphi}.$$

Auch hier geht die Rückwärtslösung durch den Faktor $e^{in\pi}$ alternierend in die Gesamtlösung ein, wenngleich sie auch nicht störend wirkt, da alles stabil bleibt.

Schließlich sei noch das System

$$\begin{aligned} (d/dt)x &= p \\ (d/dt)p &= x \end{aligned} \qquad (6.12)$$

betrachtet. Der Unterschied im Vorzeichen der zweiten Gleichung gegenüber System (6.9) macht sich im Vorzeichen der Diskriminante in den beiden Lösungen der charakteristischen Gleichung, die aus dem Potenzansatz $x_n = r^n$ entsteht, bemerkbar. Die Lösungen sind somit reell und daher nicht zu einer periodischen Gesamtlösung zusammenfaßbar. Das System (6.12) besitzt einen Sattelpunkt in $(x,p) = (0,0)$. Abb. 10 gibt das Feld dieses Systems wieder. Die Richtung, in der die Trajektorien in der Zeit durchlaufen werden, ist durch Pfeile gekennzeichnet. Bei Vorzeichenumkehr in dem Differentialgleichungssystem (6.12), d.h. bei Zeitumkehr, dreht sich der Richtungssinn um.

Wählt man nun einen Anfangspunkt auf der Winkelhalbierenden im zweiten bzw. vierten Quadranten, dann verläuft die zugehörige Trajektorie auf der Winkelhalbierenden in den Fixpunkt. Rechnet man das System mit einem reversiblen Algorithmus von FREDKINscher Art, und zwar mit positivem Gedächtnisterm, so läuft die Trajektorie mit Anfangspunkt auf einer der besagten Winkelhalbierenden nach einer gewissen Zeit wieder aus dem Fixpunkt heraus, und zwar auf der Winkelhalbierenden des gegenüberliegenden Quadranten, wie man in Abb.11 erkennt. Dies entspricht der Integration der zeitinvertierten Differentialgleichung; das Ergebnis ist somit konsistent mit den bisherigen Erkenntnissen, daß alle Fixpunkte instabil werden. Immer dann, wenn $(d/dt)x = 0$ wird, d.h. wenn die Vorwärtslösung einen Extremalwert durchläuft, macht sich die mitlaufende Rückwärtslösung bemerkbar. Schließt sich an diesen Extremalwert

eine Vorwärtslösung, die ohnedies mit der Rückwärtslösung identisch ist - wie es z.B. bei periodischen Funktionen der Fall ist - so führt die Verwendung eines reversiblen Algorithmus von FREDKINscher Art also zur Stabilisierung des Systems. In Kapitel 8 wird auf diese Eigenschaft nochmals explizit eingegangen.

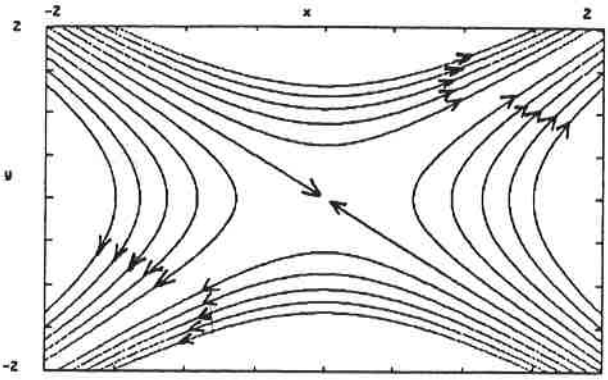


Abb.10: Phasenraum zur Differentialgleichung (6.12), gerechnet mit VERLET. Zeitschritt h = .005, Anfangswerte: $p_0 = 2$, $x_0 = -2.5 \ldots -1.5$; $p_0 = -2$, $x_0 = 1.5 \ldots 2.5$.

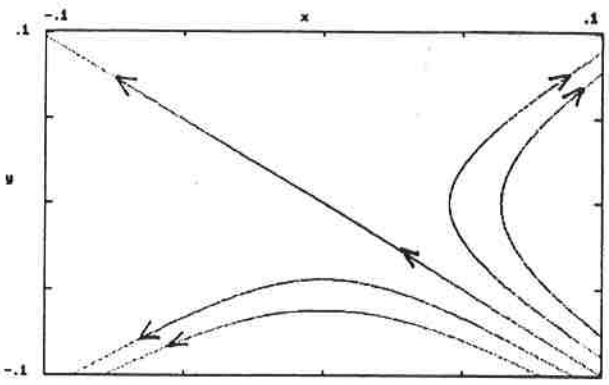


Abb.11: Ausschnitt aus dem Phasendiagramm um den Ursprung zur Differentialgleichung (6.12). Algorithmus: VERLET. Anfangswerte: $p_0 = -.2$, $x_0 = .19 \ldots .21$. Zeitschritt h=.01.

Es sei an dieser Stelle auf eine interessante Arbeit von I. BIALINICKI-BIRULA [9] hingewiesen. In dieser Publikation leitet der Autor aus der Bedeutung der Wellenfunktion und ihrer Komplex-Konjugierten, die Nichtexistenz eines Zeitpfeils in der Quantenmechanik ab. Die Wellenfunktion beschreibt demnach einen quantenmechanischen Übergang, und die komplex konjugierte Wellenfunktion den dazu inversen Übergang, und zwar deshalb, weil die SCHRÖDINGER-Gleichung der komplex-konjugierten Wellenfunktion mit der zeitinvertierten ursprünglichen SCHRÖDINGER-Gleichung zusammenfällt. Welcher Vorgang als Vorwärts- und welcher als Rückwärtsprozeß gedeutet wird, ist willkürlich. Beide Prozesse sind nach der Interpretation des genannten Autors aber in der Raum-Zeit existent. (Eine Messung bewirkt dann eine Umkehr der intrinsischen Zeit).

Nach BIALYNICKI-BIRULA ist dies ein auf die Quantenmechanik beschränktes Phänomen. In der klassischen Physik tritt diese Zweideutigkeit komplexer Zahlen in der Formulierung von Theorien nirgends auf. Ein interessantes Faktum scheint zu sein, daß sich die bei der zeitreversiblen Diskretisierung entstehende Zweideutigkeit in Anwendung auf periodische Systeme im Auftreten von Vor- und Rückwärtslösung in Form zueinander konjugiert komplexer Zahlen bemerkbar macht. Gewissermaßen als Ausblick für zukünftige Anstrengungen in dieser Richtung sei die Idee geäußert, durch eine Einführung einer zeitlichen Zweideutigkeit (= Nichtexistenz eines Zeitpfeils) in der klassischen Physik möglicherweise die Existenz eines Wirkungsquantums aus einem tieferen Prinzip heraus abzuleiten. Die Bedeutung eines solchen Vorgehens wäre die konsequente Erweiterung des EINSTEINschen Gedankenganges, nämlich nicht nur den (makroskopischen) Bewegungszustand des Beobachters in die objektive Physik einzubinden, sondern auch seinen (mikroskopischen) thermodynamischen Zustand. Eine statistische Schwankung des Zeitpfeils in obigem Sinne könnte das Resultat einer solchen Untersuchung sein [10]. Die Verwendung von komplexen Zahlen scheint für einen solchen Zweck geradezu prädestiniert, denn eine komplexe (zeitabhängige) Exponentialfunktion beschreibt ja bekanntlich einen in der komplexen Ebene rotierenden Pfeil, die konjugierte Exponentialfunktion eine Rotation in entgegengesetzter Richtung.

Nach diesen mehr oder weniger spekulativen Äußerungen sei abschließend zu diesem Kapitel bemerkt, daß alle durchgeführten Computer-Simulationen mit den verschiedenen Algorithmen von FREDKINscher Art in Anwendung auf die verschiedensten Differentialgleichungen zur Bestätigung der bisherigen Erkenntnisse führten. Die Verallgemeinerung der in diesem Kapitel

durchgeführten Modellrechnungen ist daher nicht nur empirisch gesichert, sondern auch vom mathematischen Standpunkt gesehen wegen der Zeitsymmetrie plausibel. Im folgenden Kapitel sollen die Algorithmen anhand von Simulationen zweier HAMILTONscher Systeme untersucht und beurteilt werden.

7. Energetische Stabilität von Algorithmen der FREDKINschen Art und die Kontrollfunktion des Gedächtnistermes

Anhand einer Simulation eines eindimensionalen Modellgases sollen nun die Algorithmen von FREDKINscher Art auf ihre praktische Brauchbarkeit hin untersucht werden. Die Modellsituation ist hierbei sehr einfach gehalten, um die Anschaulichkeit zu bewahren, aber dennoch bereits chaotisch im Sinne von SINAI [12] - es handelt sich um ein *Billiard*-System mit zwei ausgedehnten Teilchen, wobei eines eine konvexe Oberfläche besitzt. Die Stöße werden in der Simulation weich ausgeführt, wodurch das System neutral-stabile periodische und quasiperiodische Lösungen besitzt, aber auch durch Chaotizität bestimmt ist.

Die Abb. 12 zeigt eine Skizze des Modellgases mit den zwei Teilchen, wobei sich das eine Teilchen nur horizontal (entlang der x-Achse) zwischen $x = 0$ und $x = 1$ und das andere, konvex geformte Teilchen nur vertikal (entlang der y-Achse) zwischen $y = 0$ und $y = 0.2$ bewegt. Man erhält auf diese Weise den in Abb. 13 gezeigten Konfigurationsraum. Die Breite des x-Teilchens wurde dabei als beliebig dünn angenommen. Der Impulsraum ist durch eine Kreisfläche mit dem Radius $r = (p_{x0}^2 + p_{y0}^2)^{1/2}$ gegeben, wo p_{x0} und p_{y0} die Anfangsimpulse der beiden Teilchen bedeuten.

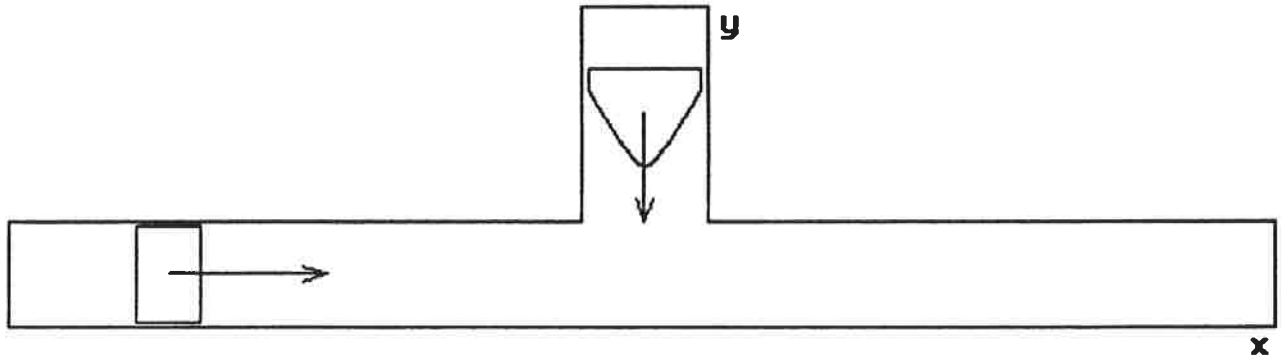


Abb. 12: Skizze zu dem chaotischen 1D - Modellgas

Die HAMILTON-Funktion zu diesem Modellgas lautet

$$H = p_x^2/2 + p_y^2/2 + \mu/x + \mu/(1 - x) + \mu/y + \mu/(0.2 - f(x) - y). \qquad (7.1)$$

Der Parameter μ beschreibt die Weichheit des Stosses. Die Funktion f(x) trägt der Rundung des vertikalen Teilchens Rechnung und lautet

$$f(x) = [(x - 0.45)^2 + 5\times10^{-6}]^{1/2} + [(x - 0.55)^2 + 5\times10^{-6}]^{1/2} - 2[(x - 0.5)^2 + 10^{-4}]^{1/2}. \qquad (7.2)$$

Die Massen der beiden Teilchen wurden als gleich vorausgesetzt und zur Vereinfachung $m = 1$ gewählt.

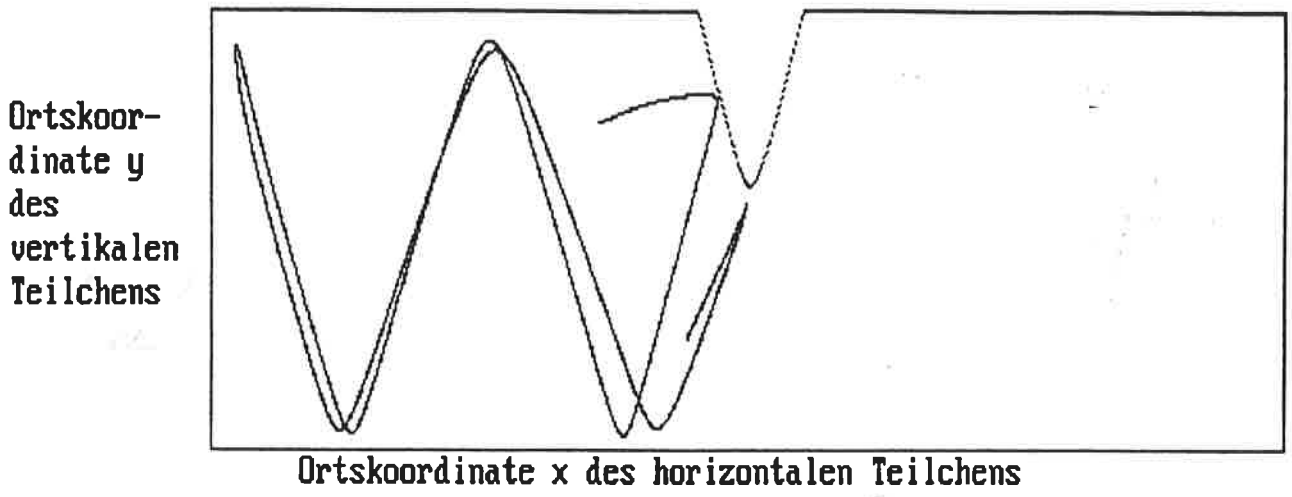


Abb.13: Der Konfigurationsraum zum 1D-Modellgas mit einem Trajektorienstück .

Für Detailfragen und Verallgemeinerungen zu diesem System siehe [13].

Die Bewegungsgleichungen zu diesem System erhält man aus den kanonischen (HAMILTONschen) Gleichungen zu

$$\begin{aligned} (d/dt)x &= p_x \\ (d/dt)y &= p_y \\ (d/dt)p_x &= \mu/x^2 - \mu/(1 - x)^2 - (\mu\, f')/(0.2 - f(x) - y)^2 \\ (d/dt)p_y &= \mu/y^2 - \mu/(0.2 - f(x) - y)^2 \end{aligned} \qquad (7.3)$$

Die Ableitung f'(x) der Funktion f(x) ist hierbei gegeben durch

$$f'(x) = \frac{x - 0.45}{[(x - 0.45)^2 + 5\times10^{-6}]^{1/2}} + \frac{x - 0.55}{[(x - 0.55)^2 + 5\times10^{-6}]^{1/2}} - 2\,\frac{x - 0.5}{[(x - 0.5)^2 + 10^{-4}]^{1/2}}.$$

Wir haben also ein konservatives HAMILTONsches System vorliegen, das demzufolge durch die zeitreversiblen Algorithmen besonders gut zu simulieren sein müsste. Zumindest ist dies der Schluß vieler Autoren, für die stellvertretend hier nur auf [8] verwiesen werden soll. Das Betätigungsfeld dieser Autoren ist die Molekular-Dynamik-Simulation, d.h. unter anderem die Untersuchung von Modellgasen, wie sie dem hier beschriebenen durchaus ähnlich sind, jedoch mit einer enorm höheren Zahl von Freiheitsgraden. Tatsächlich zeigen sich aber bereits bei dem hier beschriebenen niedrigdimensionalen System Auswirkungen des systematischen Fehlers der reversiblen Algorithmen, so daß ihre Anwendbarkeit ohne Modifikationen auf eine sehr kurze

Integrationszeit beschränkt bleibt, wenn nicht sogar unmöglich wird. Im folgenden soll für die Folge der erzeugten Iterationspunkte, die eine Trajektorie simuliert, kurz aber ungenau der Begriff *Trajektorie* verwendet werden.

Beginnen wir mit der Integration dieses Systems mit dem bekannten RK4-Verfahren. In den Abb. 14 und 15 sind die Teiltrajektorien im Konfigurationsraum bzw. im Impulsraum gezeigt. Der gewählte Parameter μ für die Weichheit des Stoßes lautet hier $\mu = 0.05$. Der Zeitschritt wurde h = 0.005 gewählt. Die eine Teiltrajektorie im Konfigurationsraum bleibt bei diesen Werten stets auf den richtigen Konfigurationsraum beschränkt und füllt diesen mit der Zeit vollständig aus; die Impulsraum - Teiltrajektorie füllt mit der Zeit die besagte Scheibe mit dem Radius $r = (p_{x0}^2 + p_{y0}^2)^{1/2}$ aus.

Die entscheidende Frage ist nun, wie sich die Energie mit fortschreitender Integrationszeit verhält? Abb. 16 gibt Aufschluß über diese Frage. Gezeigt ist das zeitliche Verhalten der Energie, die nach jedem Zeitschritt gemäß Gleichung (7.1) mit den jeweils berechneten Iterationswerten für x und y berechnet wurde. Man sieht, daß die Energie nach einer gewissen Integrationszeit vom Anfangswert $E_0 \approx 3.35$ wegzudriften beginnt und fernerhin Schwankungen aufweist. In diesem Maßstab, der im Hinblick auf spätere Vergleiche so groß gewählt wurde, ist dies gerade eben erkennbar.

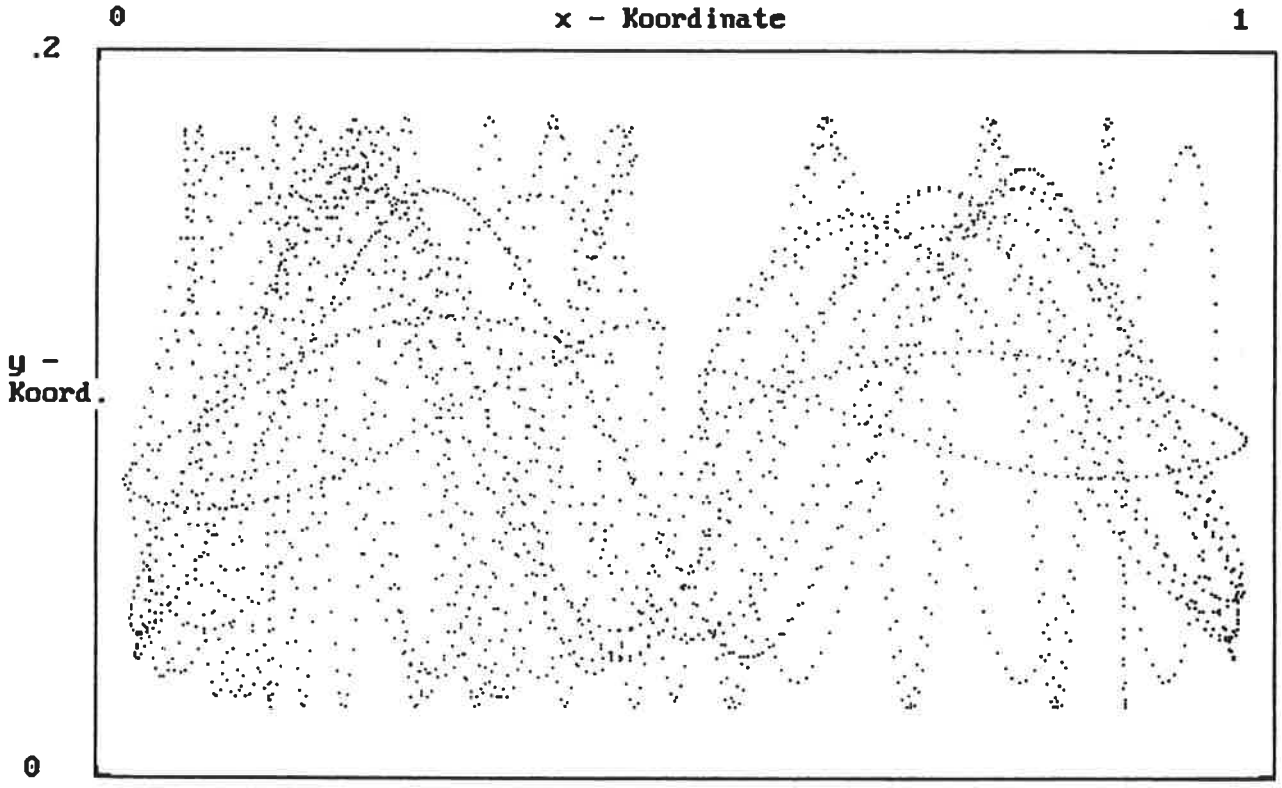


Abb. 14: Teiltrajektorie im Konfigurationsraum des 1D-Modellgases mit dem RK4-Verfahren gerechnet. Zugrundeliegende Daten: Zeitschritt h = 0.005; Weichheit $\mu = 0.05$; Anfangswerte $x_0 = y_0 = 0.05$; $p_{x0} = p_{y0} = -1$.

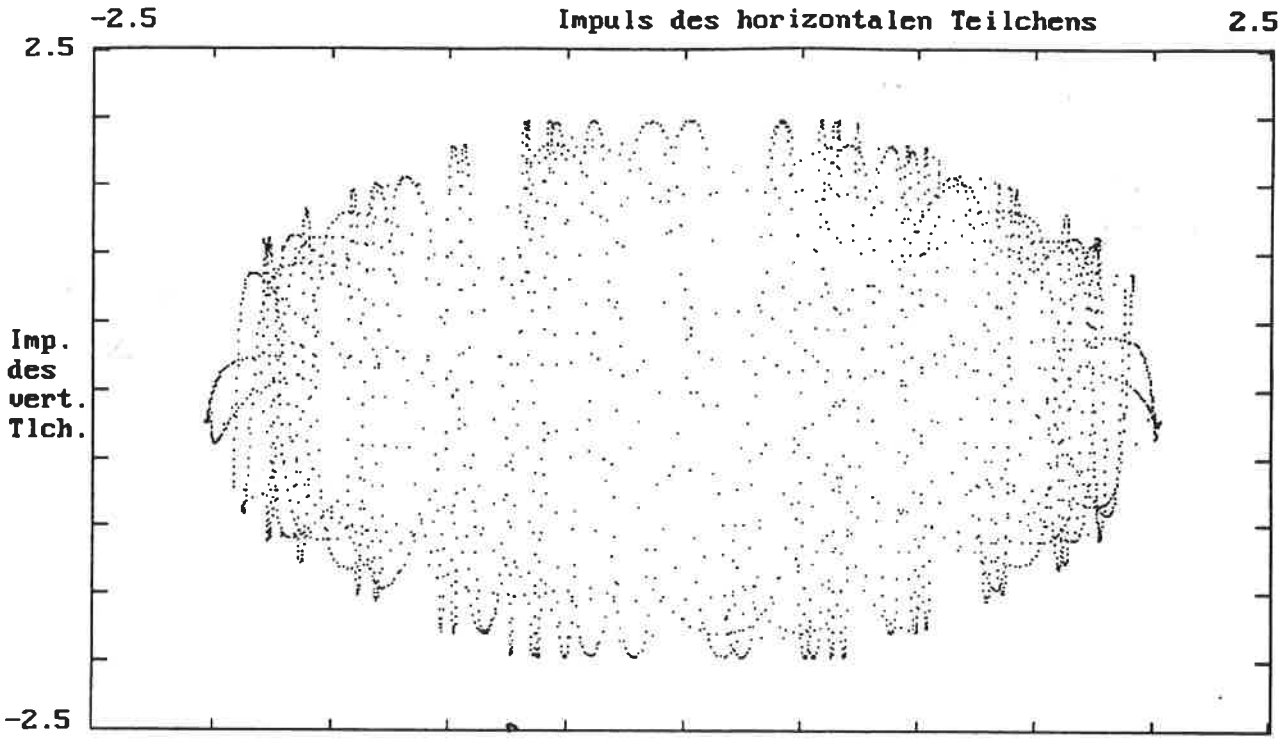


Abb. 15: Teiltrajektorie im Impulsraum des 1D-Modellgases. Daten wie in Abb. 14.

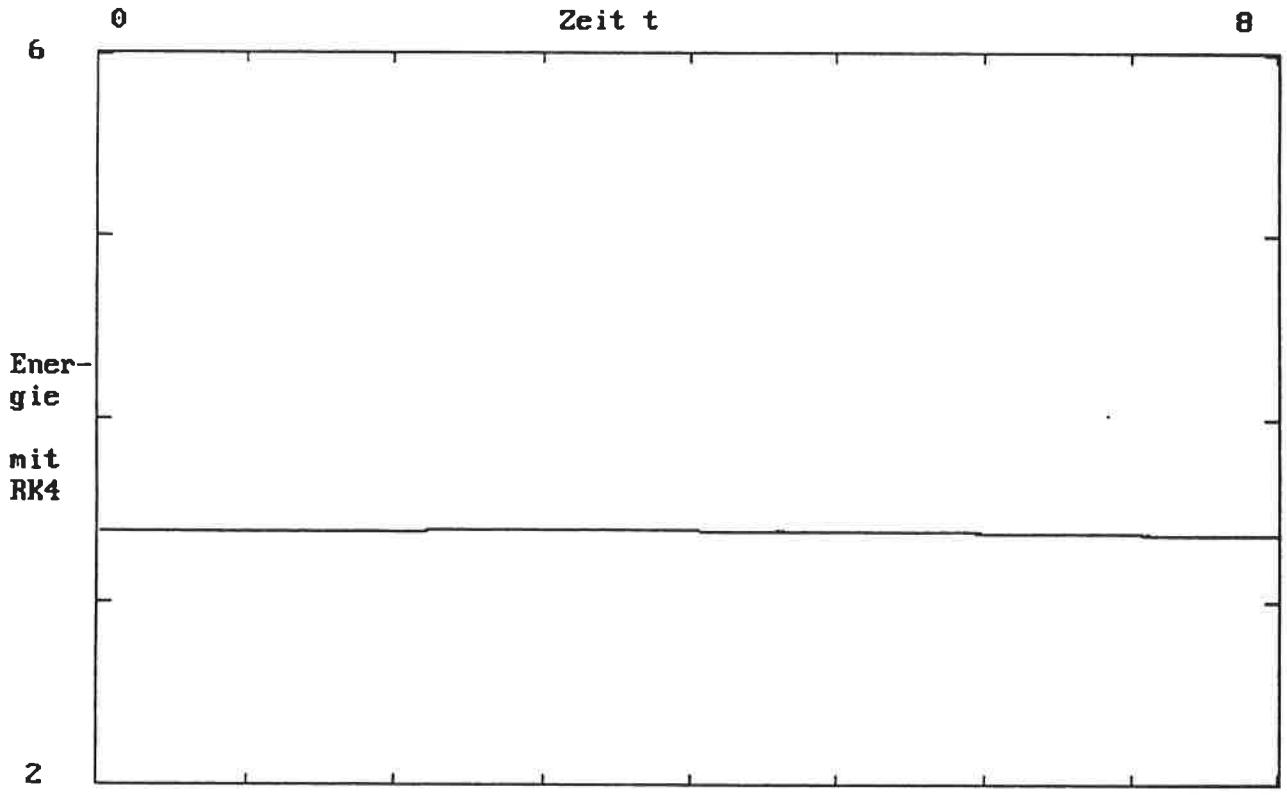


Abb. 16: Das zeitliche Verhalten der Energie bei Verwendung des RK4-Verfahrens. Parameter und Anfangswerte wie in Abb. 14. (Der ungünstige Maßstab wurde wegen einem später zu erfolgenden Vergleich mit Abb. 19 gewählt).

Wie verhält sich nun ein zeitreversibler Algorithmus in Anwendung auf dieses System? Zunächst soll zur Klärung dieser Frage der aus dem RK4-Verfahren konstruierte reversible Algorithmus FREDKINscher Art herangezogen werden. Man hat dann einen unmittelbaren Vergleich von reversiblem Algorithmus zu seinem nicht-reversiblen Ausgangsalgorithmus. Die Parameter h und μ wurden unverändert übernommen. Auch die Anfangswerte sind gleich den für die Abb. 14 und 15 zugrunde liegenden. Das Ergebnis ist in den Abb. 17, 18 und 19 zu sehen. Abb. 17 stellt den Konfigurationsraum, Abb. 18 den Impulsraum dar. Wie man erkennt, bleibt zwar der Konfigurationsraum stabil, die Impulse bleiben aber nicht auf den sphärischen Bereich beschränkt. Das Wegdriften der Impulse aus dem sphärischen Bereich verläuft synchron zum Wegdriften der Energie (vgl. Abb. 19). Besonders interessant ist die Tatsache, daß die Energie hierbei noch viel weniger stabil bleibt, als bei der Anwendung des nicht-reversiblen Ausgangsalgorithmus.

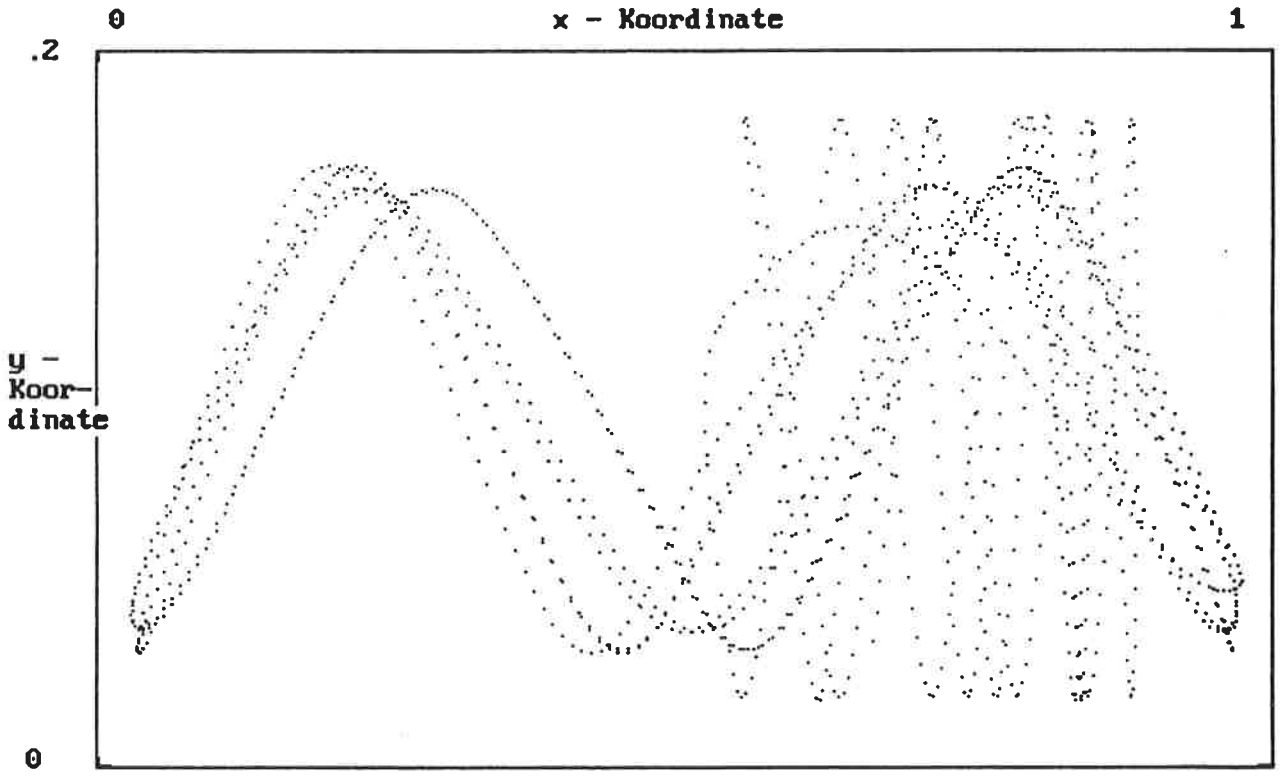


Abb. 17:Trajektorie im Konfigurationsraum des 1D-Modellgases, gerechnet mit dem reversibel konstruierten RK4-Algorithmus. Daten wie in Abb. 14.

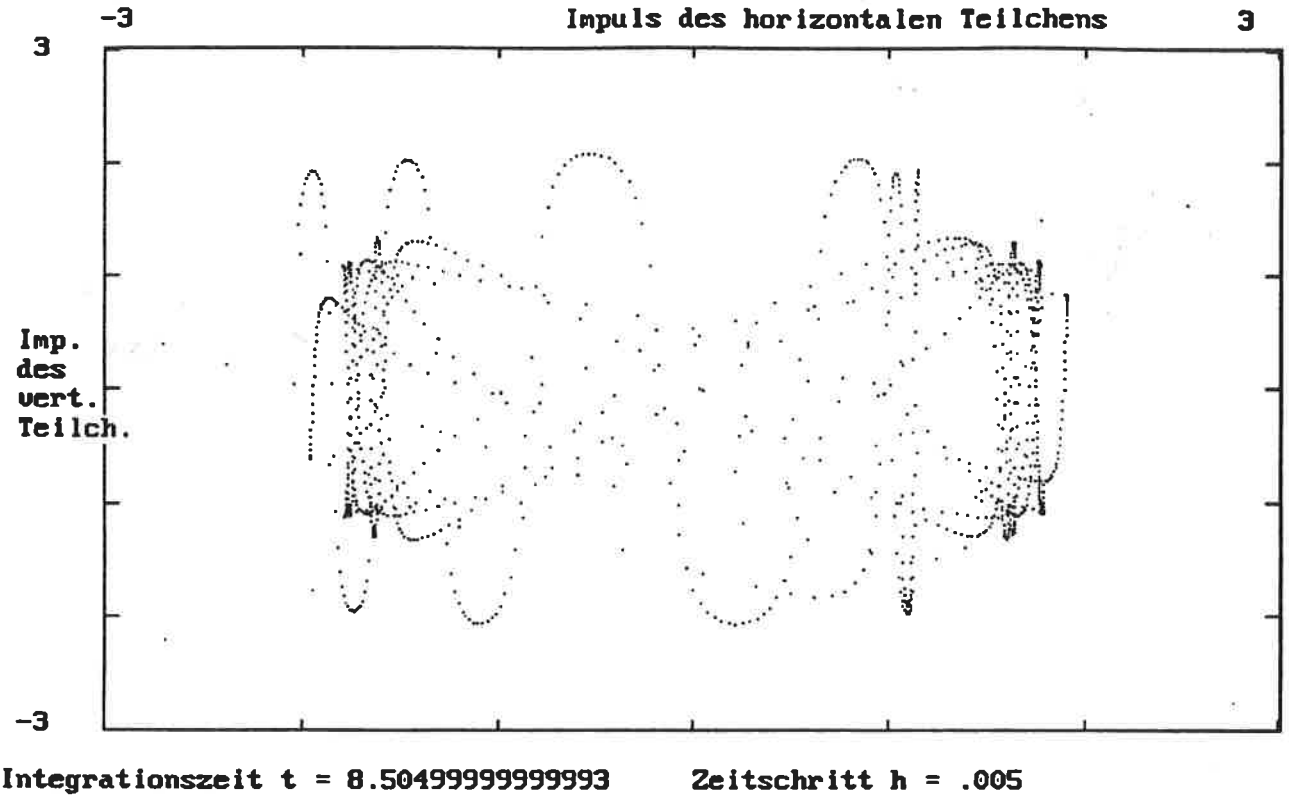


Abb. 18: Trajektorie im Impulsraum des 1D-Modellgases mit dem reversibel konstruierten RK4-Algorithmus gerechnet. Daten wie in Abb.14.

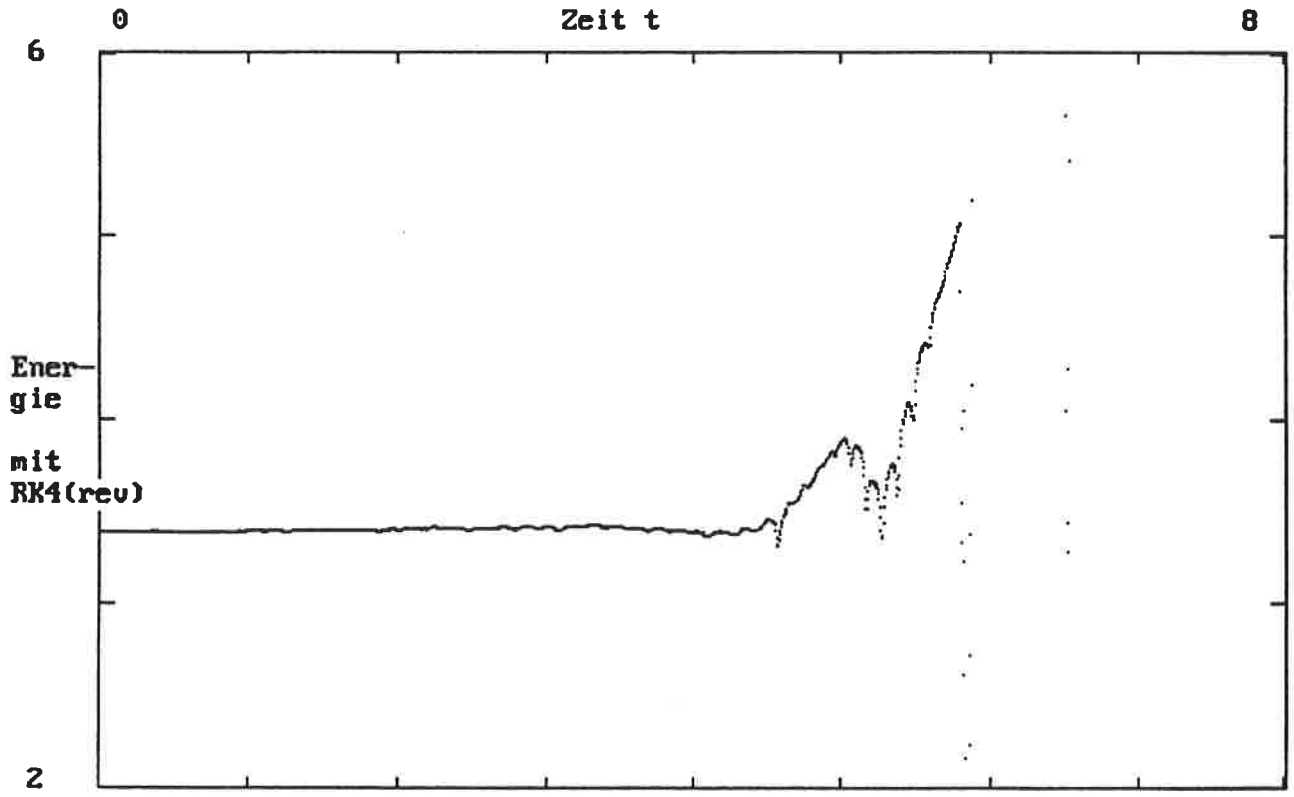


Abb. 19: Das zeitliche Verhalten der Energie des aus dem RK4-Verfahren konstruierten reversiblen Algorithmus. Zugrundeliegende Daten wie in Abb. 14.

Ein weiteres Merkmal des reversibel gestalteten RK4-Algorithmus ist die Tendenz, die Trajektorie zu symmetrisieren (vgl. Abb. 17). Diese Beobachtung läßt sich auch an Rechnungen von anderen dynamischen Systemen und anderen Algorithmen von FREDKINscher Art bestätigen.

Für das vorliegende System bedeutet dies, daß die Trajektorien öfter als bei der Integration mit konventionellen Algorithmen fast exakt in sich zurücklaufen. Durch Verkleinern des Zeitschrittes läßt sich diese Tendenz verstärken. Ein exaktes In-Sich-Zurücklaufen wird freilich durch die Rundungsfehler verhindert. [Möglicherweise können exakt-reversible Integrationsalgorithmen dazu dienen, gezielt periodische Lösungen von chaotischen Systemen aufzusuchen; dies soll hier allerdings noch zurückgestellt werden.]

Als nächstes soll der VERLET-Algorithmus unter dem Aspekt der Energieerhaltung getestet werden. Diese Untersuchung ist von besonderem Interesse, da dem VERLET-Algorithmus in gewissem Sinne eine Sonderstellung zukommt. Denn erstens läßt sich mit diesem Algorithmus ein NEWTONsches System, d. h. ein System von Differentialgleichungen 2. Ordnung, unmittelbar integrieren (wird unten erläutert). Zweitens wird dieser Algorithmus standardmäßig in der Molekular-Dynamik-Simulation verwendet und schließlich läßt er sich drittens aus bewährten Prinzipien ableiten, die von allgemeingültiger und fundamentaler Natur sind. Es ist dies das noch zu besprechend TROTTER-Theorem (siehe Kap. 9), sowie das Variationsprinzip der Wirkung. Eine Anwendung der Minimierung des Wirkungsintegrals auf ein vorher diskretisiertes dynamisches System zur Ableitung des VERLET-Algorithmus wird von R. E. GILLILAN und K. R. WILSON [14] diskutiert.

Nun liefert diese Ableitung aus dem HAMILTONschen Prinzip der Wirkungsminimierung zwar die Berechtigung der Anwendung des VERLET-Algorithmus auf ganz bestimmte Problemkreise, schränkt diese aber gleichzeitig ganz erheblich ein. Diese Tatsache ist von so großer Bedeutung, daß die Ableitung von GILLILAN und WILSON samt ihren Konsequenzen im nächsten Kapitel diskutiert werden soll.

Zunächst aber die experimentellen Fakten: Wie früher bereits erläutert, läßt sich der VERLET-Algorithmus auf zwei Weisen implementieren. Der Ausgangsalgorithmus wäre hier die TAYLOR-Methode 2. Ordnung gemäß Gleichung (2.6) mit m = 2. Man kann dann ein System von Differentialgleichungen von 1. Ordnung stets so bearbeiten, daß jede einzelne Gleichung streng nach dem Konstruktionsprinzip (3.3) gelöst wird, wobei also an jeder Iterationsstelle n immer der TAYLOR-Algorithmus vorwärts (d. h. mit positivem h) und rückwärts (d. h. mit -h) berechnet und die Summe nach (3.3) gebildet wird. Die Summe des Vorwärts- und Rückwärts-TAYLOR-Algorithmus läßt sich aber auch kompakt zum Algorithmus (4.2) zusammenfassen, was in praxi deshalb sinnvoll ist, weil dadurch erheblich an CPU-Zeit gespart wird. Eine weitere Erleichterung ergibt sich speziell für NEWTONsche Gleichungssysteme, d. h. für Differentialgleichungen 2. Ordnung wie

$$\frac{d^2}{dt^2} x = \frac{F}{m}, \quad \text{mit } F = -\text{grad } V(x). \tag{7.4}$$

Das Potential V muß selbstverständlich geschwindigkeitsunabhängig sein, da man sonst Dissipationen erhält, die (noch?) nicht reversibel gerechnet werden können. Differentialgleichungen vom Typ (7.4) lassen sich besonders bequem mit dem VERLET-Algorithmus integrieren, da sie direkt, d. h. ohne vorherige Berechnung der Impulse integriert werden können. In der Molekular-Dynamik-Simulation sind die Impulse oft von untergeordneter Bedeutung, so daß durch die Verwendung des VERLET-Algorithmus erheblich an CPU-Zeit

gespart wird. Wie schon in Kapitel 6 angegeben, lassen sich die Impulse, falls doch benötigt, approximativ durch

$$p_n = \frac{x_{n+1} - x_{n-1}}{2\,h} \qquad (7.5)$$

berechnen; eine Formel, die sich aus der Umstellung der zum VERLET-Algorithmus komplementären Midpoint-Methode ergibt.

Wir werden in Kürze sehen, daß sich hierin eine Besonderheit verbirgt, denn mit dieser kompakten Programmierweise des VERLET-Algorithmus, bei der auf die unabhängige iterative Berechnung der Impulse verzichtet wird, kann der Gedächtnisterm als Ausdruck für den systematischen Fehler nicht studiert werden. Außerdem erweist sich der Algorithmus in dieser Form als symplektisch, worauf in Kapitel 9 näher eingegangen wird.

Unser hier bearbeitetes Modell-System (8.3) ist vom NEWTONschen Typ (8.5) und eignet sich daher für solche Studien. Auf Abbildungen von Trajektorien, die mit der TAYLOR-Methode integriert wurden, sei hier verzichtet, da sich nichts wesentlich Neues gegenüber dem RK4-Verfahren ergibt. Nach Wahl eines unterkritischen Zeitschrittes (z. B. h = 0.0005) bleibt sowohl der Konfigurationsraum als auch der Impulsraum stabil. Eine Tatsache, die natürlich keineswegs eine Auskunft über die Güte der Iteration gibt. Allerdings ist eine Fehlerrechnung für die TAYLOR-Methode einfach durchzuführen und standardmäßig in der Literatur zu numerischen Methoden für Differentialgleichungen angegeben.

Abb. 20 zeigt nun die zeitliche Entwicklung der Energie bei Rechnung mit dem TAYLOR-Verfahren. Zum Vergleich hierzu sieht man in Abb. 21 die Energie in Abhängigkeit von der Integrationszeit für den "vollständigen" VERLET-Algorithmus, d. h. den sowohl auf die Orte als auch auf die Impulse angewandten Algorithmus. Konfigurationsraum und Impulsraum zu diesem Algorithmus sind für eine Integrationszeit von ca. 10 Zeiteinheiten in den Abb. 22 bzw. 23 wiedergegeben. Der Konfigurationsraum bleibt stabil, d. h. die Trajektorie auf den durch die geometrischen Randbedingungen des Systems gegebenen limitierten Bereich (vgl. Abb. 13) beschränkt. Nicht so die Teiltrajektorie im Impulsraum. Diese verbleibt nicht in dem theoretisch auf die oben bereits angegebene Sphäre beschränkten Bereich. Wie kann der Konfigurationsraum, nicht aber der Impulsraum stabil bleiben?

Die Antwort auf diese Frage findet sich in einer genaueren Analyse dessen, was bei jedem einzelnen Iterationsschritt geschieht: Der Vorwärts-TAYLOR-Algorithmus ergibt einen Zwischenpunkt, der wegen der großen Impulse tatsächlich weit außerhalb des limitierten Konfigurationsraums liegt. Der Rückwärts-TAYLOR-Algorithmus liefert ebenfalls einen weit außerhalb liegenden Zwischenpunkt. Dieser liegt aber nahezu exakt "gegenüber" dem Vorwärts-Zwischenpunkt, d. h., bei der zu bildenden Summe beider Zwischenpunkte heben sich die relativ großen Beträge nahezu weg (da sich in der Summe die Impulsterme wegheben). Der resultierende nachfolgende Iterationspunkt liegt daher innerhalb der Konfigurationsraumgrenzen, verläuft aber in eine Richtung, die nahezu senkrecht auf dem durch die Vorwärts- und Rückwärts-Integration mittels des TAYLOR-Algorithmus gegebenen Richtungsfeld steht. Mit anderen Worten: Die reversibel gerechnete Trajektorie hat die bereits in Kapitel 4 angesprochene Tendenz, aus dem Richtungsfeld des dynamischen Systems heraus zu driften.

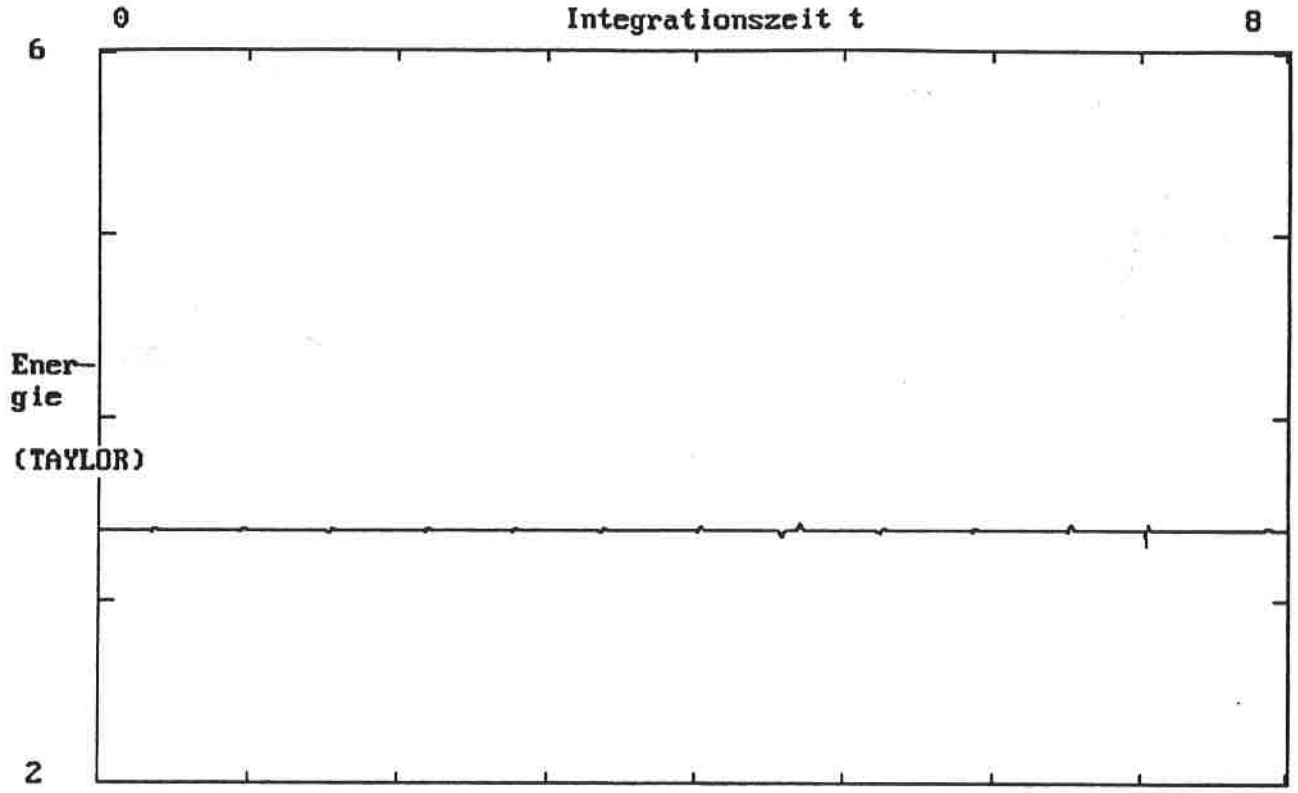


Abb. 20: Zeitlicher Energieverlauf für den TAYLOR-Algorithmus 2. Ordnung. Zeitschritt h = 0.0005; Weichheit $\mu = 0.05$; Anfangswerte wie in Abb. 14.

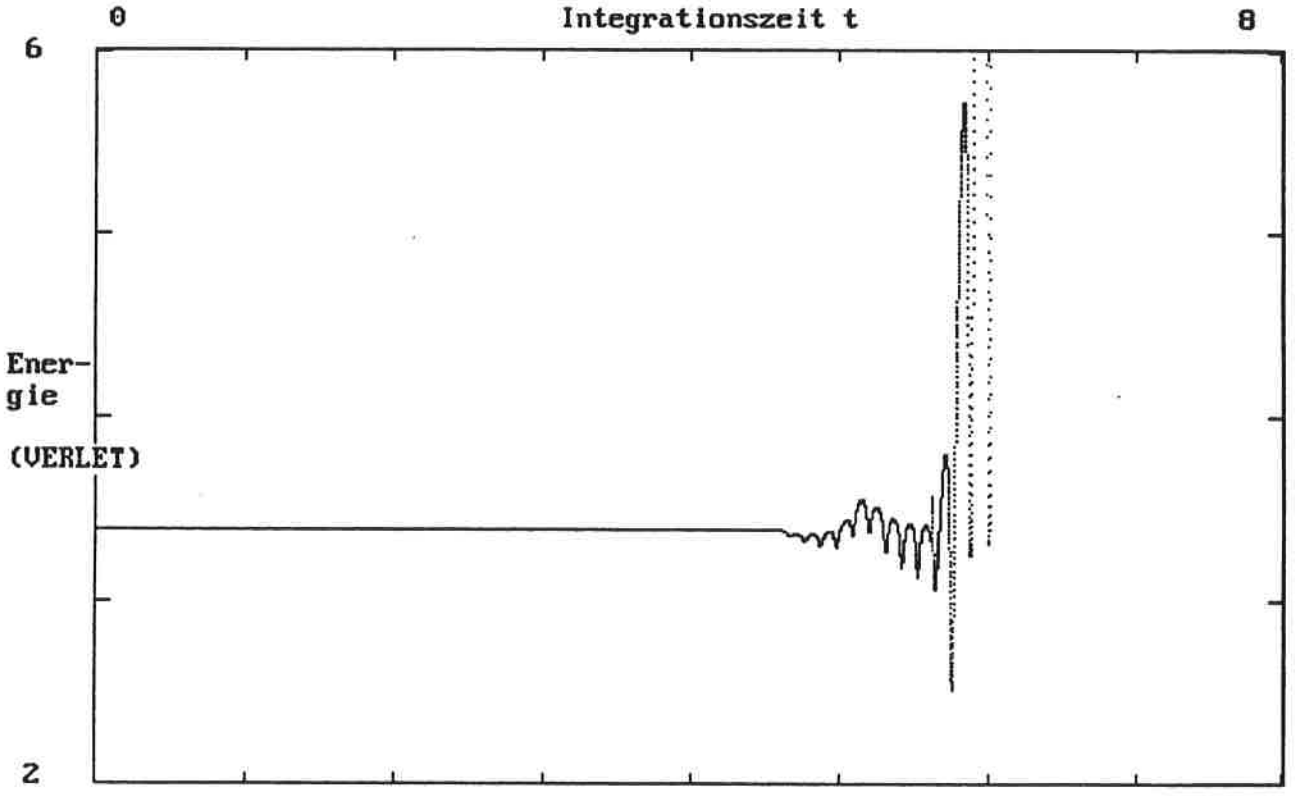


Abb. 21: Zeitlicher Energieverlauf für den VERLET-Algorithmus mit iterativer Impulsberechnung ebenfalls mittels des VERLET-Algorithmus. Daten wie in Abb. 20.

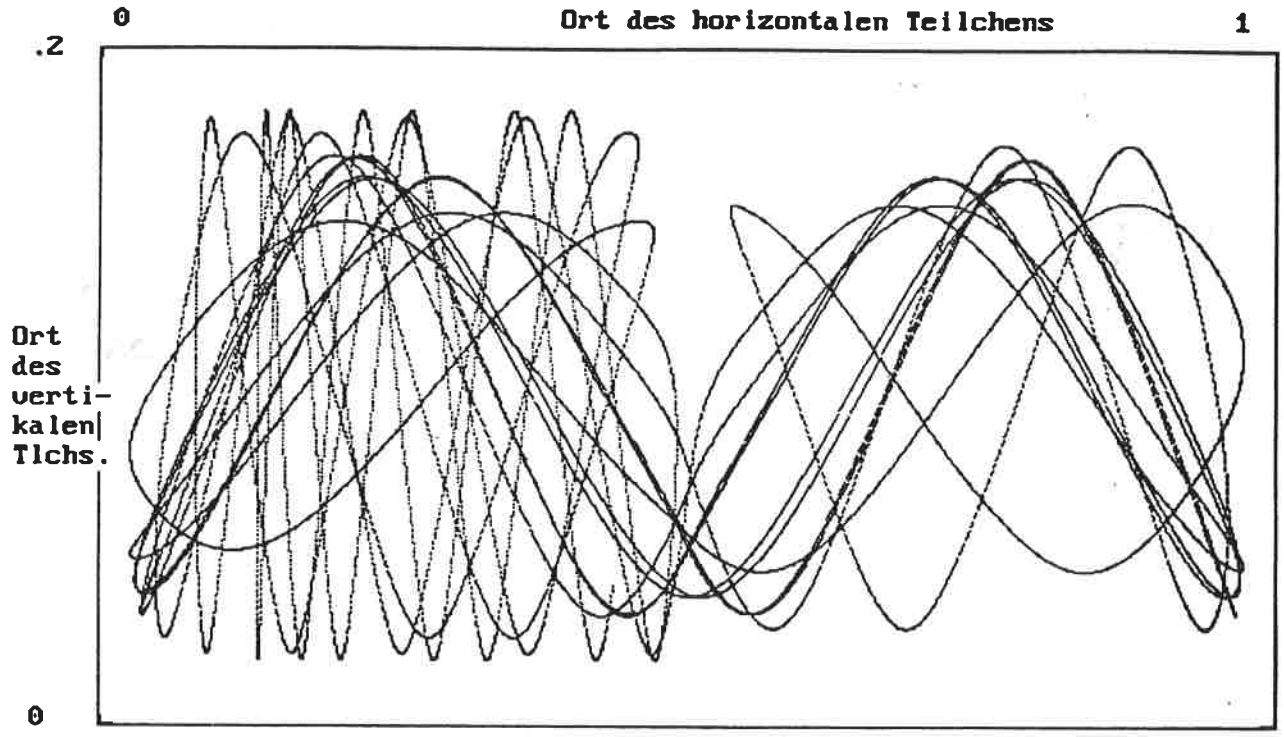


Abb. 22: Trajektorie im Konfigurationsraum. Algorithmus: VERLET für Orte und Impulse. Anfangswerte und Parameter wie in Abb 20.

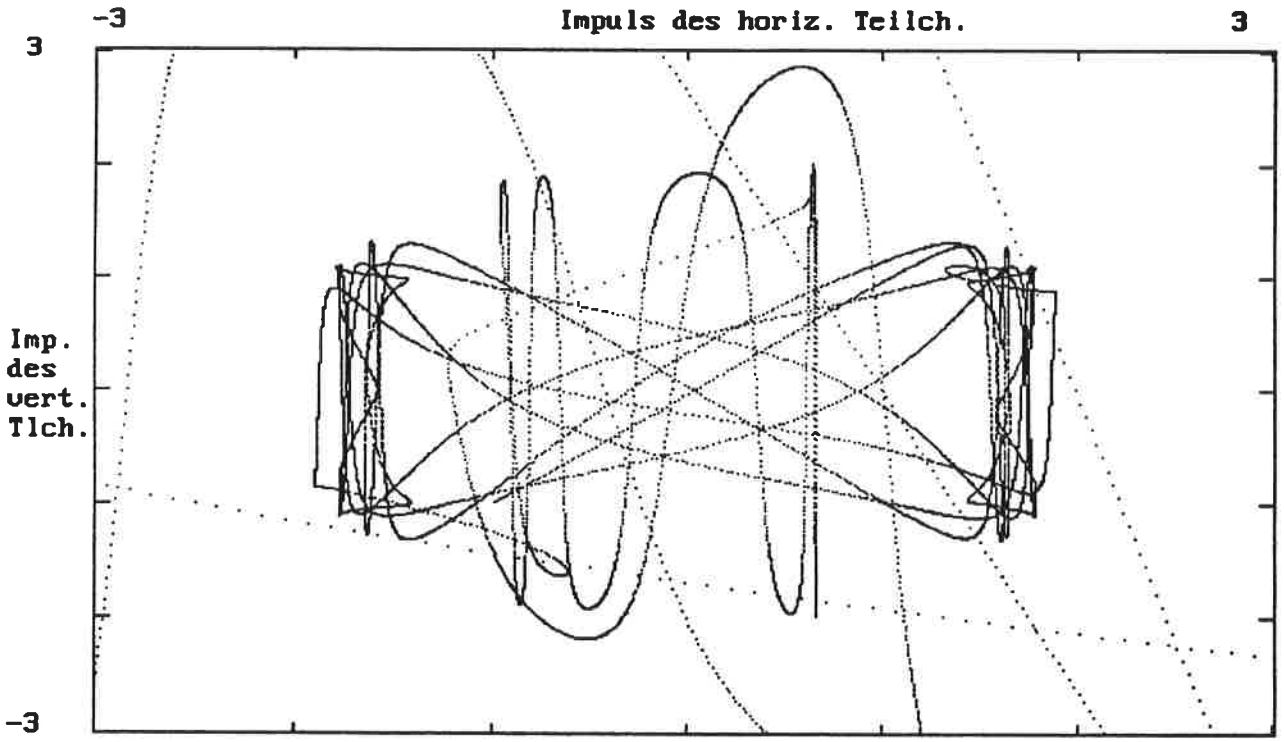


Abb. 23: Trajektorie im Impulsraum. Algorithmus: VERLET für Orte und Impulse. Anfangswerte und Parameter unverändert aus Abb. 20 übernommen.

Man muß nun allerdings vorsichtig mit der Interpretation dieses Ergebnisses sein. Das Richungsfeld, von dem hier die Rede ist, muß nicht zwangsläufig das wahre Richtungsfeld sein, denn es wird ja an jeder Iterationsstelle aus den fehlerhaften iterativen Impulsen bestimmt. Eine objektive Aussage über den lokalen Fehler kann kaum gemacht werden.

Eine paradox anmutende Situation ergibt sich, wenn man nun andererseits die Impulse durch die approximative Formel (8.5) jeweis im Nachhinein aus den unabhängig von den Impulsen berechneten Orten (mithilfe des kompakten VERLET-Algorithmus) ermittelt. Man erhält auf diese Weise Werte, die tatsächlich auf den sphärischen Bereich beschränkt bleiben. Zieht man diese approximativen Impulse fernerhin zur Berechnung der Energie heran, dann erhält man den in Abb. 24 gezeigten zeitlichen Verlauf. Diese Abbildung zeigt einen repräsentativen zeitlichen Ausschnitt. Man erkennt eine gewisse Schwankungsbreite der Energie, aber um einen (langzeitig) stabilen Mittelwert. Der TAYLOR-Ausgangsalgorithmus weist zwar Schwankungen nur in kleineren Amplituden auf; der Mittelwert driftet aber deutlich von der Anfangsenergie weg (vgl. Abb. 20).

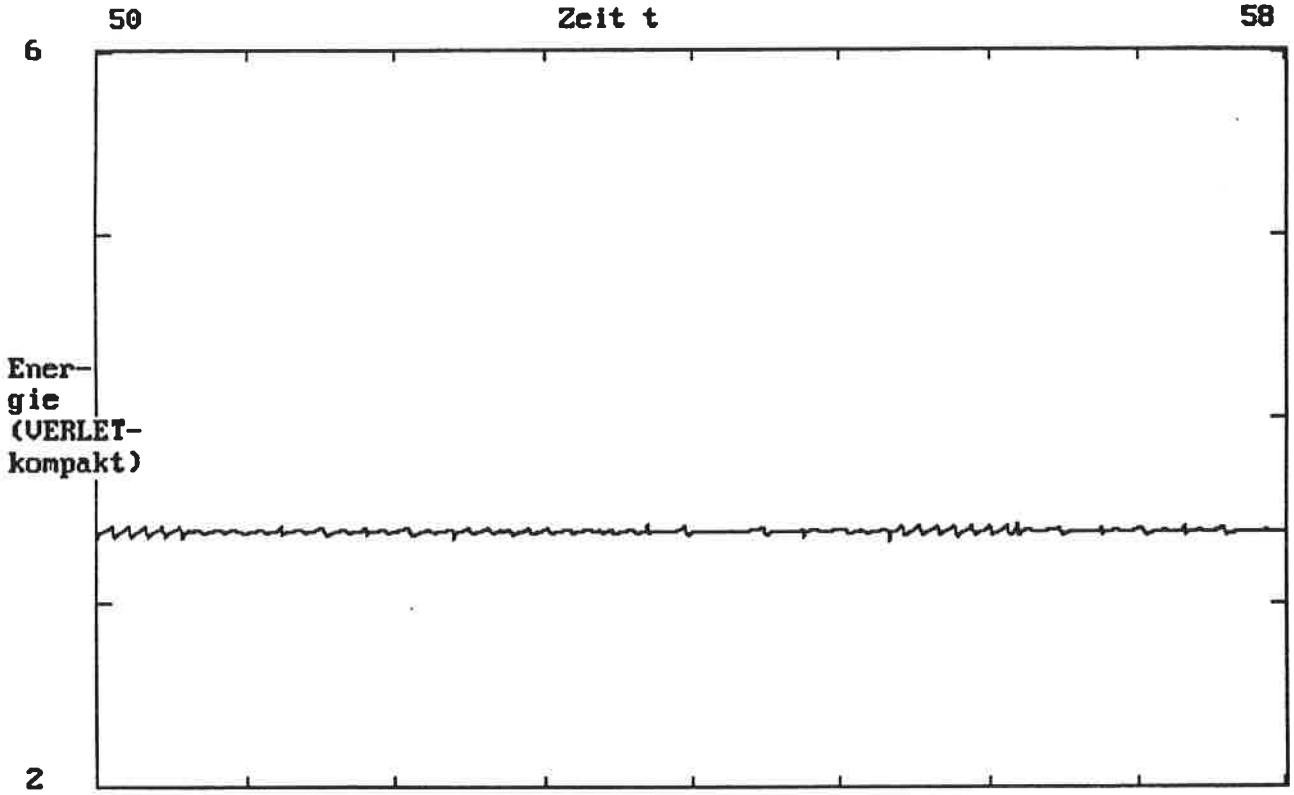


Abb. 24: Der energetische Verlauf des 1D-Modellgases mit dem kompakten VERLET-Algorithmus. Repräsentativer Ausschnitt von t = 50 bis t = 58 Zeiteinheiten. Sonstige Daten wie in Abb. 20.

Um den Preis der u. U. tragbaren höheren Schwankungsbreite und um den Preis des Verzichts auf "exakte" Impulsberechnung, scheint der kompakt implementierte VERLET-Algorithmus also eine brauchbare Möglichkeit zu bieten, exakt reversibel rechnen zu können - zumindest Systeme vom NEWTONschen Typ (7.4). Durch die approximative Impulsformel (7.5) werden also stets solche Impulswerte ermittelt, die die symplektische Beziehung zu den zugehörigen Orten und insbesondere die Energie erhalten. Allerdings ist dadurch nicht die geringste Garantie für die Güte der Rechnung gegeben. Für den vorliegenden Fall läßt sich lediglich sagen, daß die Teiltrajektorie für die Orte stets in dem vorgegebenen Konfigurationsraum verbleibt. Fraglich bleibt, ob eventuell vorhandene Strukturen im Phasenraum durch die Rechnung mit dem VERLET-Algorithmus richtig oder überhaupt wiedergegeben werden.

Obige Besonderheit des (kompakten) VERLET-Algorithmus büßt man übrigens wieder ein, wenn man im TAYLOR-Grundalgorithmus zwei Ordnungen höher geht. Man erhält dann nämlich in Anwendung auf das NEWTONsche Gleichungssystem (7.4) mit der Masse $m \equiv 1$ den reversiblen Algorithmus

$$x_{n+1} = 2x_n + h^2F(x_n) + (h^4/12)(d^2/dt^2)F(x_n) - x_{n-1}.$$

Die zweite zeitliche Ableitung der Kraft im dritten Summanden der rechten Seite dieser Gleichung lautet hierbei explizit:

$$(d^2/dt^2)F(x_n) = (d^2/dx^2)F(x_n)\,[(d/dt)x_n]^2 + (d/dx)F(x_n)\,F(x_n).$$

Dieser Ausdruck wird nur für $(d^2/dx^2)F(x) \equiv 0$ impulsunabhängig. Schließt man triviale Fälle aus, so ist diese Identität für (quadratische) Oszillatorpotentiale erfüllt. Wir haben also eine weitere Bestätigung, daß die Algorithmen von FREDKINscher Art besondere Dienste im periodischen Fall tun. Es verdient festgehalten zu werden, daß der VERLET-Algorithmus einen gewissen Sonderfall darzustellen scheint.

Als Repräsentant eines negativ korrigierten reversiblen Algorithmus gemäß Gleichung (3.5) sei hier noch die Midpoint-Methode betrachtet. Wie zu erwarten, spaltet die Trajektorie nach einer gewissen Integrationszeit auf (d. h. alterniert zwischen Vor- und Rückwärtslösung des Ausgangsalgorithmus). Mithin beginnt natürlich auch die Energie zu alternieren, und zwar zwischen Werten die ungefähr spiegelsymmetrisch zur Anfangsenergie liegen. Vergleiche hierzu die Abb. 25. Auf eine graphische Darstellung von Konfigurations- und Impulsraum sei hier verzichtet, da sich nichts Neues ergibt. Stattdessen sei mithilfe der Abb. 26 auf die zeitliche Korrelation des Anwachsens des Gedächtnistermes (o. B. d. A. für die Ortskomponente des horizontalen Teilchens)

$$[x_n + hp_n] - x_{n-1}$$

und der Aufspaltung der Energie hingewiesen. Man kann bei Verwendung eines negativ korrigierten reversiblen Algorithmus den Beginn der (sichtbaren oder durch Vorgabe einer Schwelle ermittelten) Aufspaltung oder aber die Größe des Gedächtnistermes gewissermaßen als Güteindikatior heranziehen. Letzteres gilt auch für die positiv korrigierten Algorithmen. Z. B. verläuft für das korrigierte RK4-Verfahren das Anwachsen des Gedächtnistermes (o. B. d. A. für die Ortskomponente des horizontalen Teilchens, siehe Abb. 27) simultan zum Anwachsen der Energie. Man vergleiche Abb. 27 mit Abb. 19.

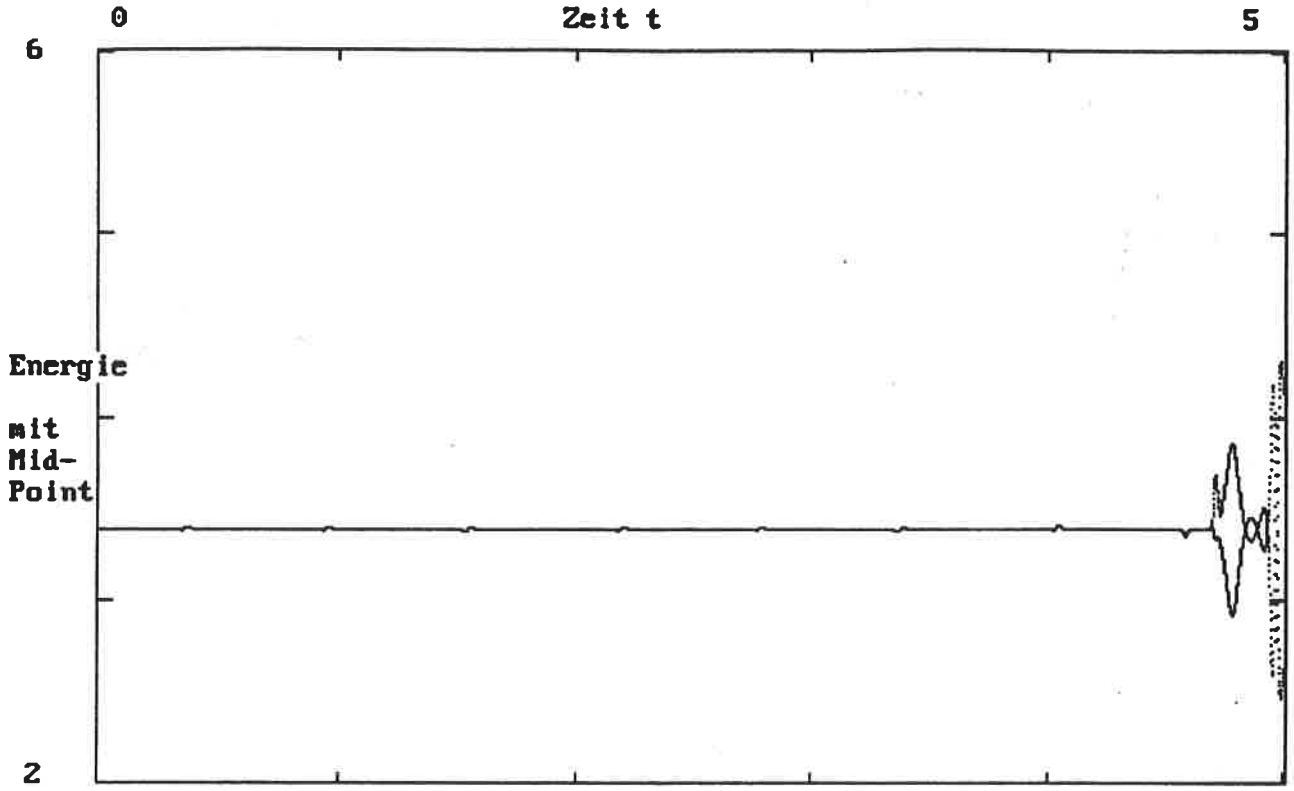


Abb. 25: Die Energie in Abhängigkeit von der Integrationszeit für die Midpoint-Methode. Zeitschritt h = 0.0005; Weichheit μ = 0.05.

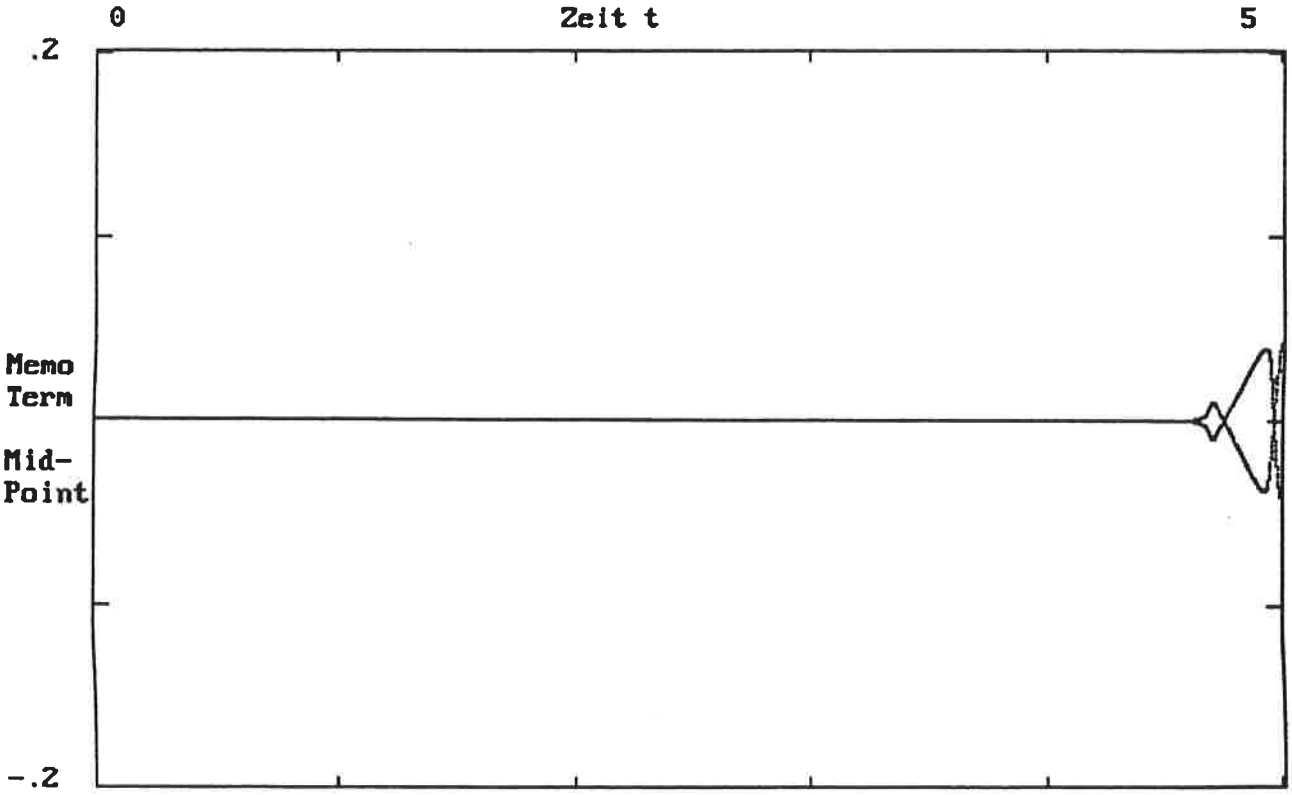


Abb. 26: Der Gedächtnisterm versus Integrationszeit (absolut aufgetragen) für die Midpoint-Methode. Man vergleiche die zeitliche Korellation zum energetischen Verhalten in Abb. 25.

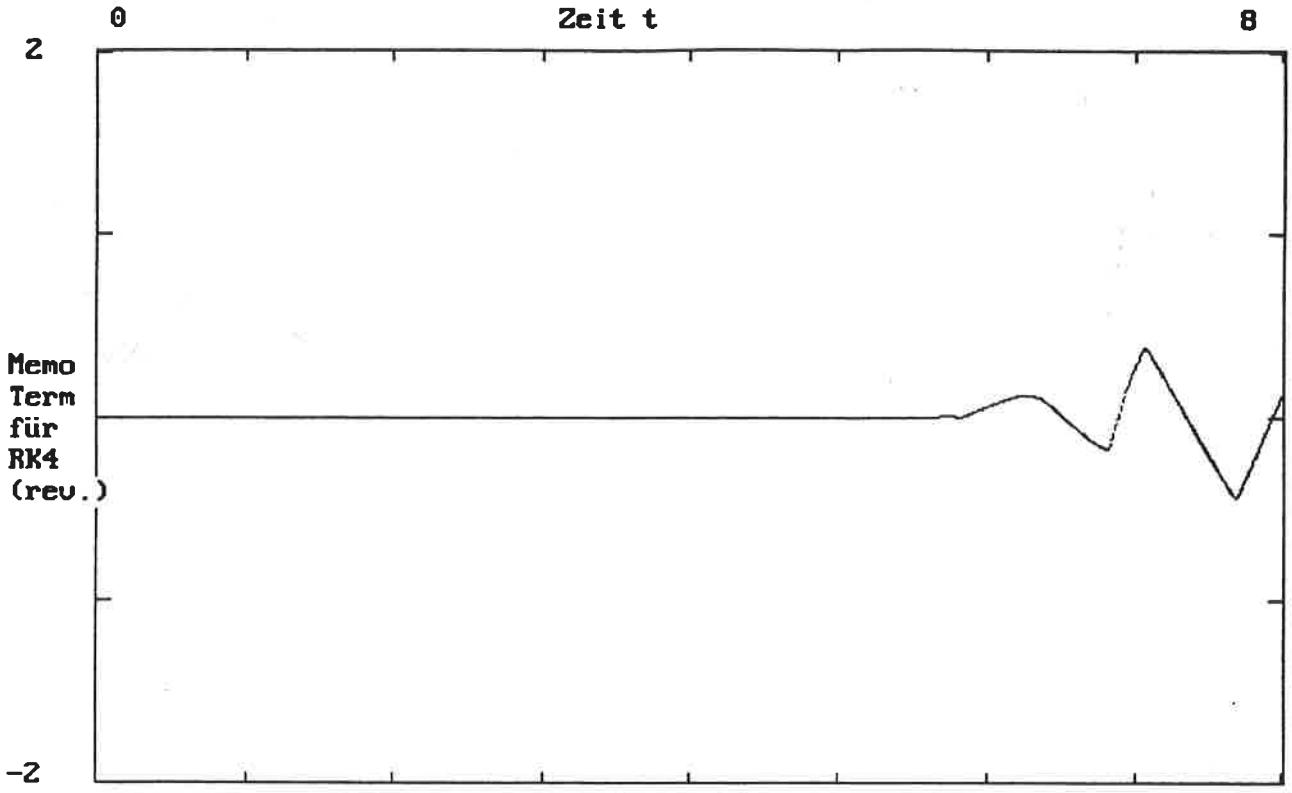


Abb. 27: Der Gedächtnisterm für die Ortskomponente des horizontalen Teilchens des reversibel konstruierten RK4-Verfahrens. Zeitschritt h = 0.005; Weichheit μ = 0.05 (vgl. hierzu Abb. 19).

Zur Unterstützung der bisherigen Ergebnisse bezüglich der energieerhaltenden Eigenschaften der reversiblen Algorithmen sei noch die Anwendung auf ein zweites konservatives System betrachtet. Die HAMILTON-Funktion dieses Sytems ist durch

$$H = (p_x^2 + p_y^2)/2 - \varepsilon(x^2 + y^2) + x^2y^2(x^2 + y^2)/2 \qquad (7.6)$$

gegeben. Die ersten beiden Summanden auf der rechten Seite dieser Gleichung stellen für sich genommen einen harmonischen Oszillator dar. Der dritte Summand bringt nichtlineare Anteile in die Bewegungsgleichungen und führt damit zu einer Irregulariät, die für betragsmäßig kleiner werdenden Parameter ε zunimmt. Wählt man den Parameter ε = -1, so erhält man ein weitgehend reguläres System. Für ε = -0.1 ist praktisch keine Regularität mehr vorhanden.

Das durch die HAMILTON-Funktion (8.6) gegebene System ergibt sich nach einer Transformation eines KEPLER-Problems auf semiparabolische Koordinaten und repräsentiert damit z.B. ein Wasserstoffatom im Magnetfeld. Der Parameter ε steht hierbei mit Energie E und Magnetfeldstärke η durch $\varepsilon = E\,\eta^{-2/3}$ im Zusammenhang. Näheres zu diesem System ist für die hier verfolgten Zwecke nicht relevant und kann beispielsweise in [4,19] nachgeschlagen werden. Das System ist hinlänglich genau untersucht und die Ergebnisse hier sollen im Hinblick auf Konsistenz mit diesen Untersuchungsergebnissen erörtert werden.

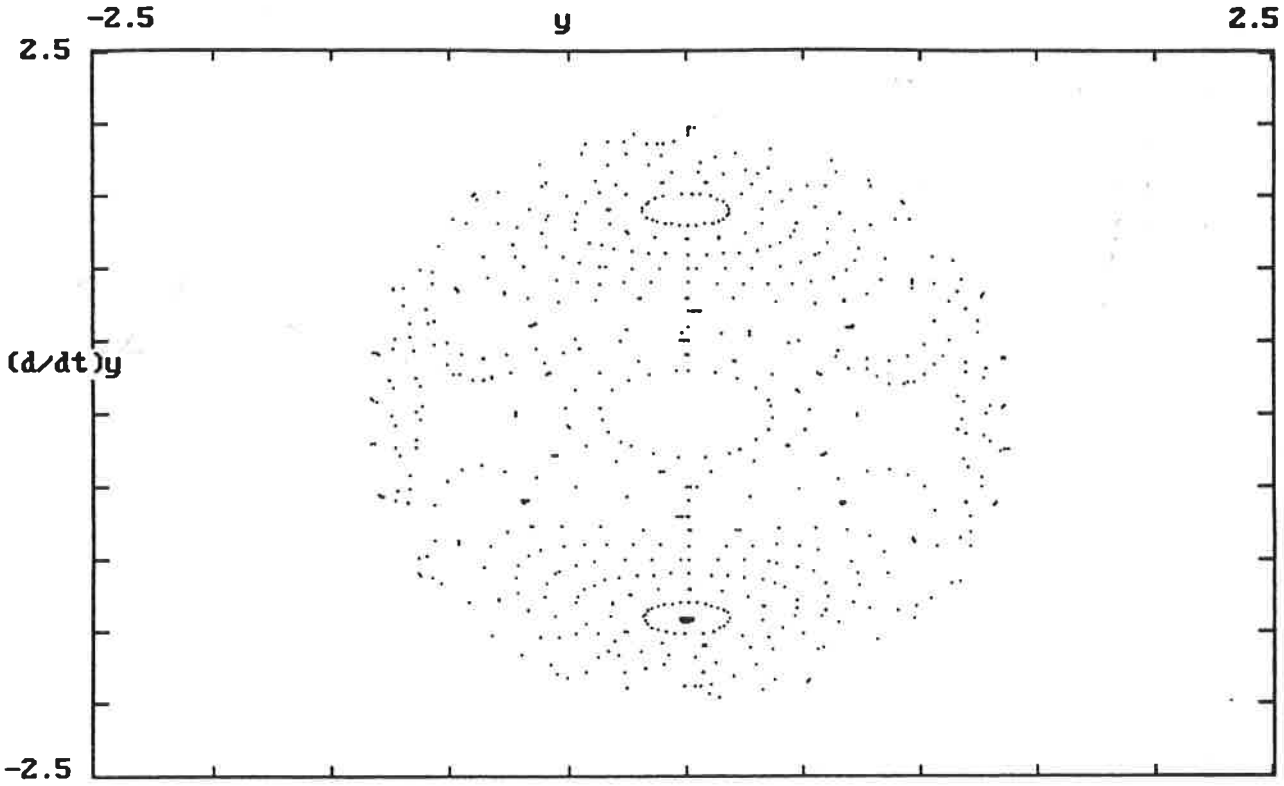


Abb. 28: POINCARE-Schnitt durch die y-p_y-Ebene (x = 0) für das diamagnetische H-Atom, gerechnet mit dem kompakt implementierten VERLET-Algorithmus, d. h. mit der approximativen Impulsformel (7.5). Anfangswerte: $x_0 = 0$, $y_0 = -1.9....2.0$, $p_{x0} = (4 + 2\varepsilon y_0^2 - p_{y0}^2)^{1/2}$, $p_{y0} = 0.01$; Parameter $\varepsilon = -1$ (regulärer Fall); Zeitschritt h = 0.01.

Die zu integrierenden Bewegungsgleichungen berechnen sich aus den kanonischen Gleichungen zu

$$\begin{aligned} (d/dt)x &= p_x \\ (d/dt)y &= p_y \\ (d/dt)p_x &= 2\varepsilon x - 2x^3y^2 - xy^4 \\ (d/dt)p_y &= 2\varepsilon y - 2y^3x^2 - yx^4. \end{aligned} \qquad (7.7)$$

Die Gesamtenergie des Systems sei H = 2 gewählt. Das System läßt sich dann zweckmäßigerweise durch einen POINCARE-Schnitt, z.B. durch die y-p_y-Ebene, darstellen. Hierbei setzt man beispielsweise x = 0 fest und p_x ergibt sich aus der Umformung von (7.6) mit H = 2. Abb. 28 stellt einen solchen POINCARE-Schnitt dar, und zwar für den regulären Fall ε = -1. Gerechnet wurde hier mit dem VERLET-Algorithmus bei approximativer Impulsberechnung. Diese Phasenraumdarstellung deckt sich mit Berechnungen mittels anderer (Standard-) Algorithmen. Von besonderem Interesse aber ist die Energieerhaltung unter dem reversiblen Algorithmus im direkten Vergleich zu einem nicht reversiblen (Standard-) Algorithmus und insbesondere zu seinem Ausgangsalgorithmus. Abb. 29 gibt den zeitlichen Verlauf der Energie für den kompakt implementierten VERLET-Algorithmus über 100 Zeiteinheiten wieder. Die Energie weist erhebliche Schwankungen auf und kann damit mit der "Energieerhaltung" des RK4-Verfahrens nicht konkurrieren, wie man durch einen Vergleich mit Abb. 30 feststellt. Den beiden Prints liegen dieselben Daten für die Anfangswerte, Parameter und Zeitschritt zugrunde; außerdem wurde auf gleichen Maßstab der Wiedergabe geachtet. Um die

Wirkung der Konstruktion eines nicht-reversiblen Algorithmus zu einem reversiblen angemessen beurteilen zu können, darf natürlich der Vergleich zum nicht-reversiblen Ausgangsalgorithmus selbst nicht fehlen. Dieser Vergleich ist anhand Abb. 31 möglich. Dort ist - wiederum mit denselben Daten und demselben Maßstab - die Energie unter dem TAYLOR-Algorithmus zweiter Ordnung in ihrem zeitlichen Verlauf dargestellt. Die Schwankungen um die mittlere Energie ist von gleicher Größenordnung wie beim VERLET-Algorithmus, die Drift der mittleren Energie ist aber bei diesem Zeitschritt von h = 0.01 untragbar hoch. Schließlich sei auch die "Energiekonservierung" unter dem vollständig implementierten VERLET-Algorithmus betrachtet. Man erhält durch die Abb. 32 die Bestätigung des obigen Ergebnisses, daß hier der systematische Fehler seine Auswirkung zeigt und das System instabil macht, was sich u. a. in einem enormen Anwachsen der Amplitude der Energieschwankung bemerkbar macht.

Dieses Ergebnis wird durch die Anwendung des reversibel konstruierten RK4-Verfahrens ebenfalls bestätigt. Wählt man wiederum den Zeitschritt h = 0.01, so ergibt sich unter diesem Algorithmus der in Abb. 33 dargestellte zeitliche Energieverlauf für das diamagnetische H-Atom. Ganz deutlich ist auch hier das - offenbar exponentielle - Anwachsen der Schwankungsamplitude zu erkennen, d. h., die Akkumulation des systematischen Fehlers. Noch einmal muß also ausdrücklich erwähnt werden, daß die hier untersuchten reversiblen Algorithmen von FREDKINscher Art, mit Ausnahme des kompakt implementierten VERLET-Algorithmus, nur für kleine Integrationszeiten verwendet werden können. Am besten führt man stets den Gedächtnisterm bei der Iteration explizit mit und gibt eine Schwelle für den Abbruch der Rechnung vor.

Der Vollständigkeit halber sei schließlich noch eine Rechnung des diamagnetischen Wasserstoffatoms für den nicht-regulären (d. h. weitgehend chaotischen) Fall mit dem kompakten VERLET-Algorithmus durchgeführt. Die Abb. 34 zeigt die Energie des Systems in ihrem zeitlichen Verlauf unter diesem Algorithmus. Es wurde wieder der Zeitschritt h = 0.01 und dieselben Anfangswerte wie in den früheren Rechnungen verwendet. Die Energie weist unregelmäßige Schwankungen auf, d. h. die Irregularität macht sich hier bemerkbar. Quasi sicherheitshalber ist in Abb. 35 noch eine Langzeitrechnung für denselben Fall zu sehen. Man erkennt keine merklichen Auswirkungen eines etwaigen systematischen Fehlers; das System bleibt stabil.

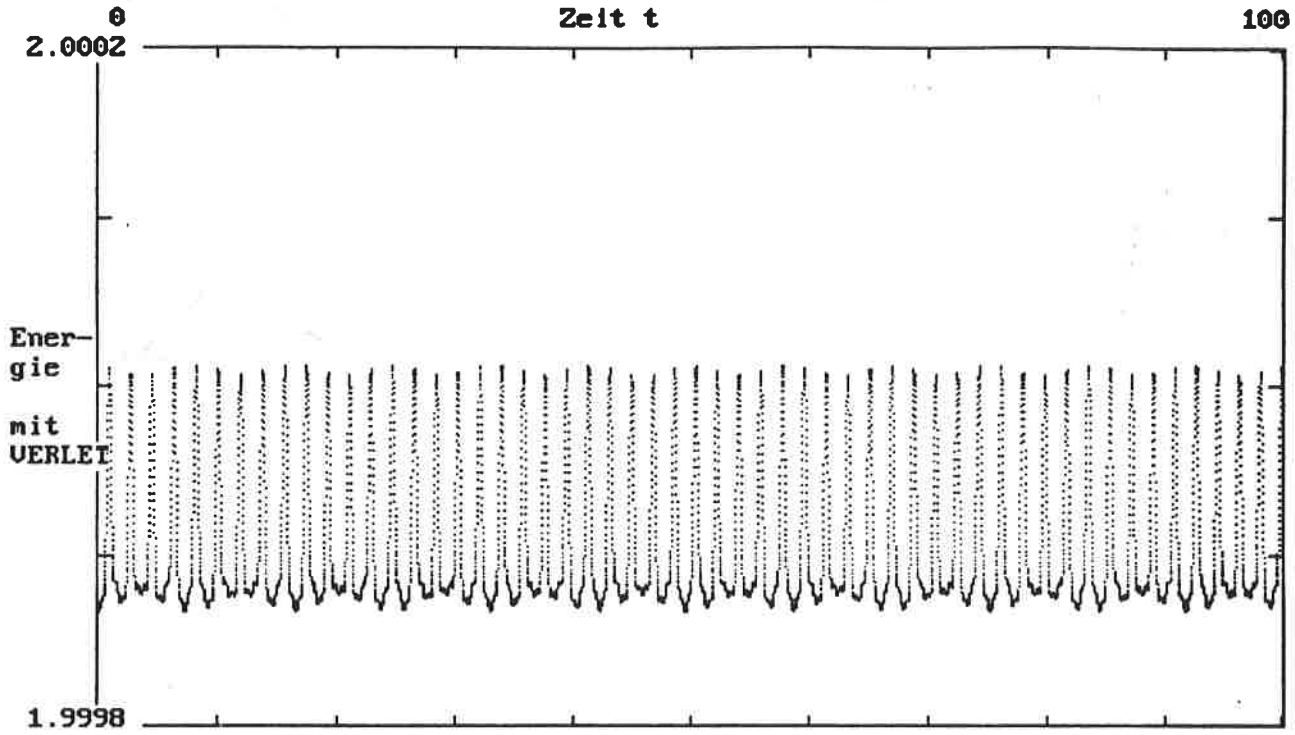


Abb. 29: "Energieerhaltung" unter dem VERLET-Algorithmus mit approximativer Impulsberechnung für das diamagnetische H-Atom. Zeitschritt h = 0.01; Parameter ε = -1 (regulärer Fall); Anfangswerte: $x_0 = 0$, $y_0 = 0.1$, $p_{y0} = 1$, p_{x0} aus H = 2 berechnet.

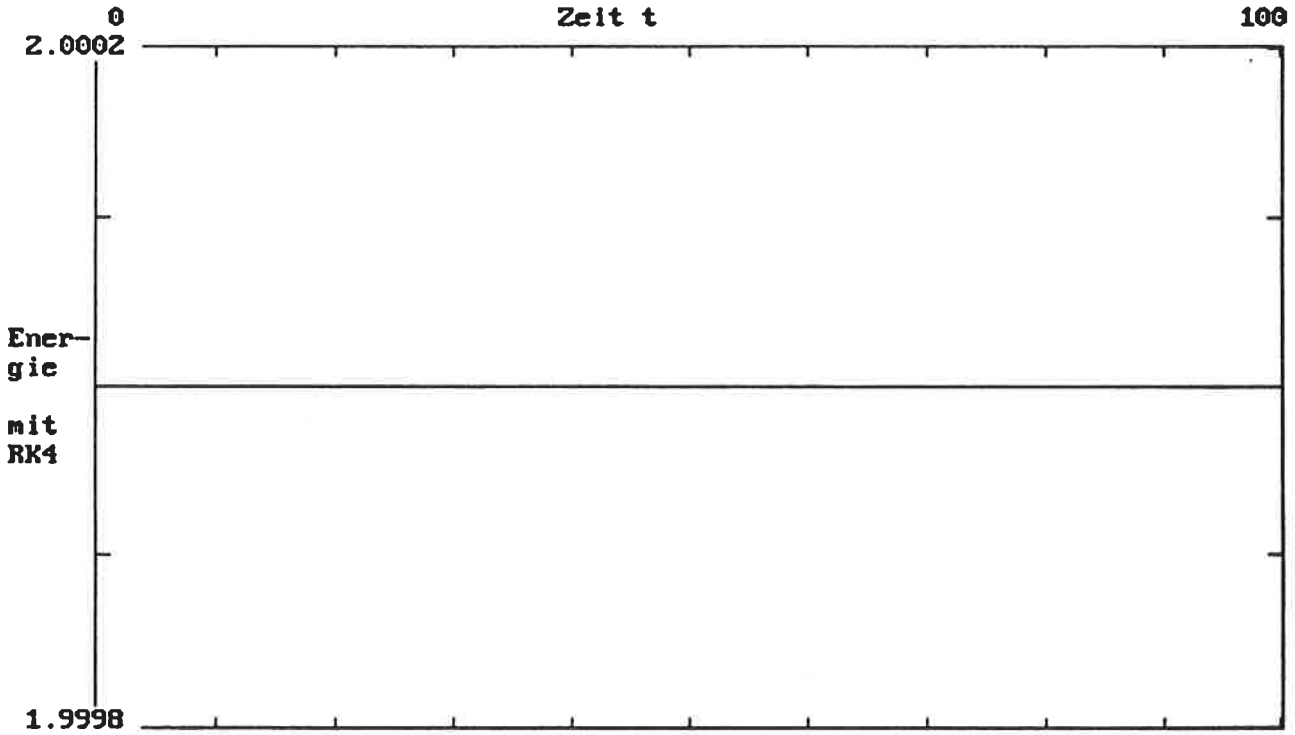


Abb. 30: Zeitliches Energieverhalten beim diamagnetischen H-Atom unter dem RK4-Algorithmus zum Vergleich mit dem VERLET-Algorithmus (Abb. 29). Dieselben Daten wie in Abb. 29.

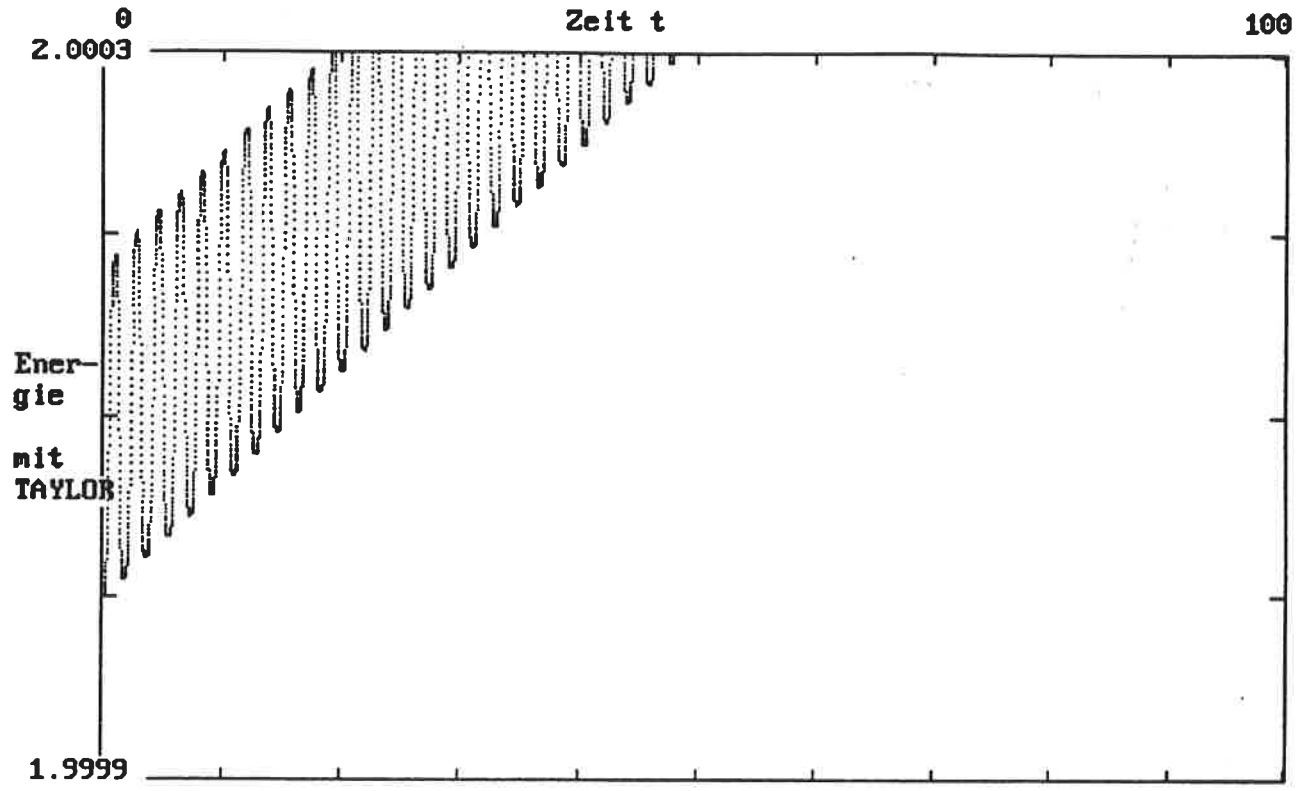


Abb. 31: Energieverlauf beim diamagnetischen H-Atom mit dem TAYLOR-Algorithmus zweiter Ordnung. Zeitschritt h = 0.01 Anfangswerte und Parameter wie in Abb. 29.

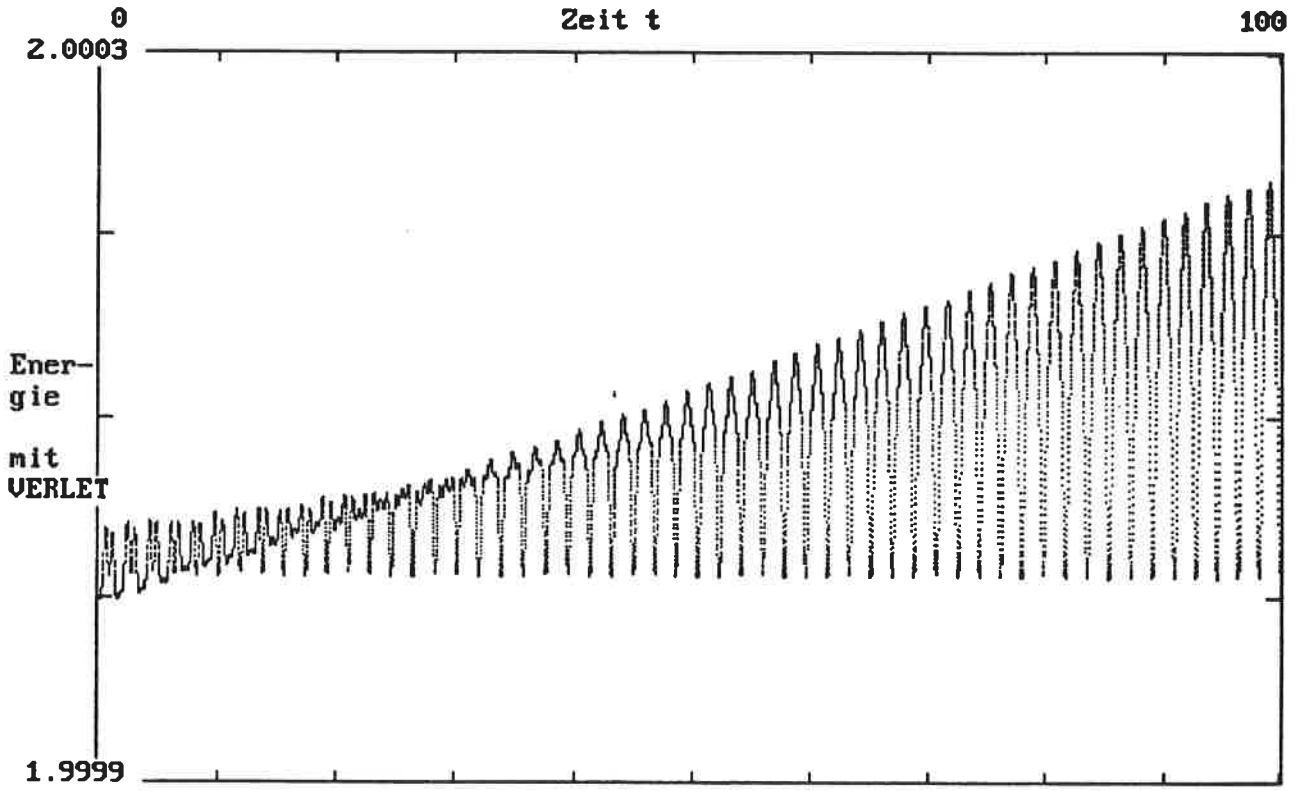


Abb. 32: Energie beim diamagnetischen H-Atom für den vollständig implementierten VERLET-Algorithmus. Anfangswerte und sonstige Daten wurden beibehalten (vgl. Abb. 29 und 31).

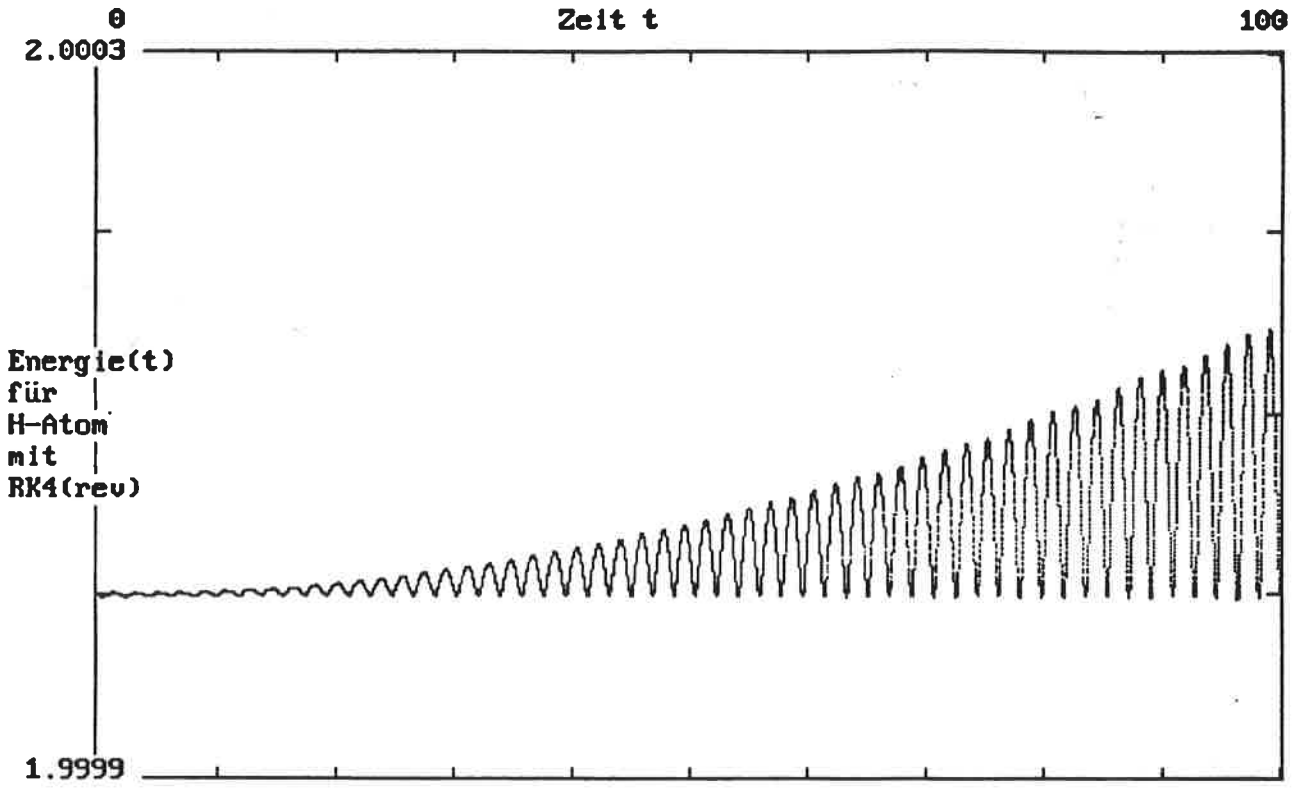


Abb. 33: Zeilicher Verlauf der Energie des diamagnetischen H-Atoms im regulären Fall, gerechnet mit dem reversibel konstruierten RK4-Algorithmus.

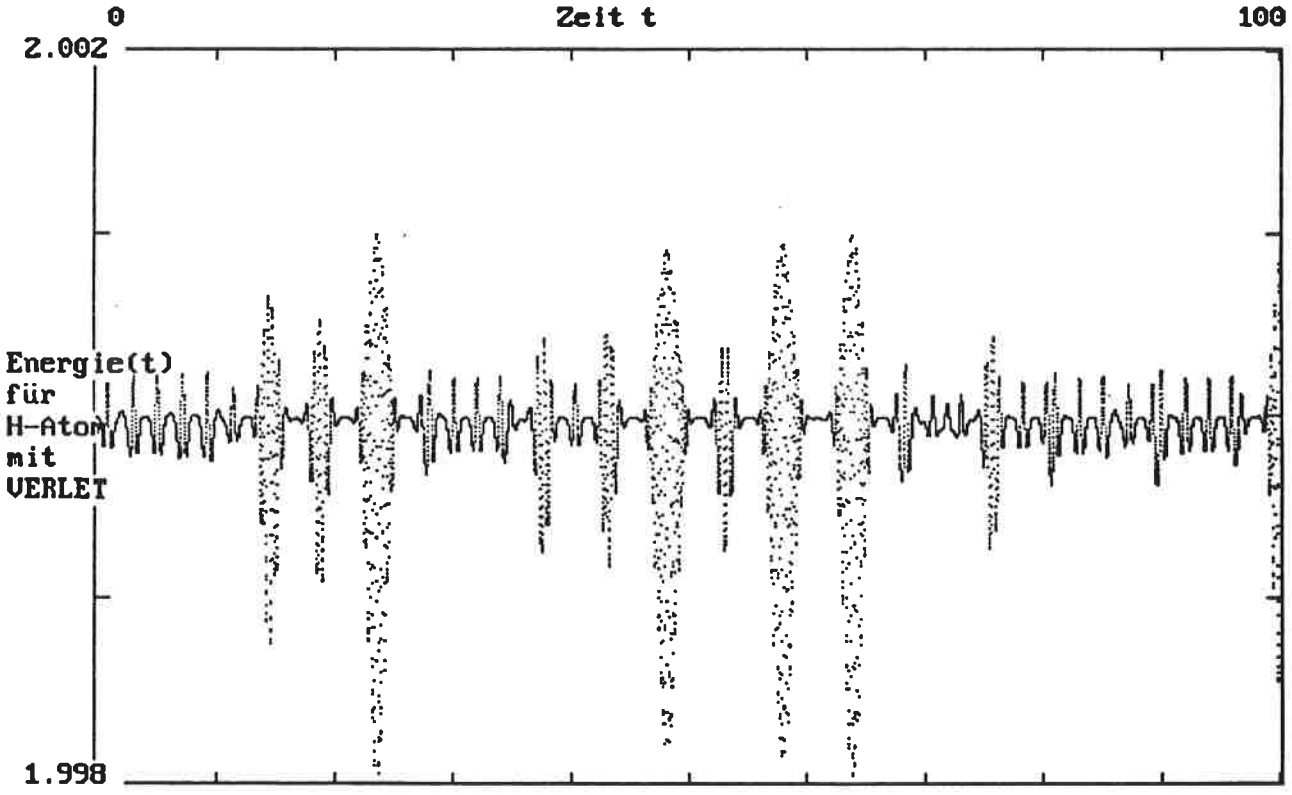


Abb. 34: Die Energie in ihrem zeitlichen Verhalten für das diamagnetische H-Atom im nicht-regulären Fall, gerechnet mit dem kompakten VERLET-Algorithmus (Kurzzeitrechnung).

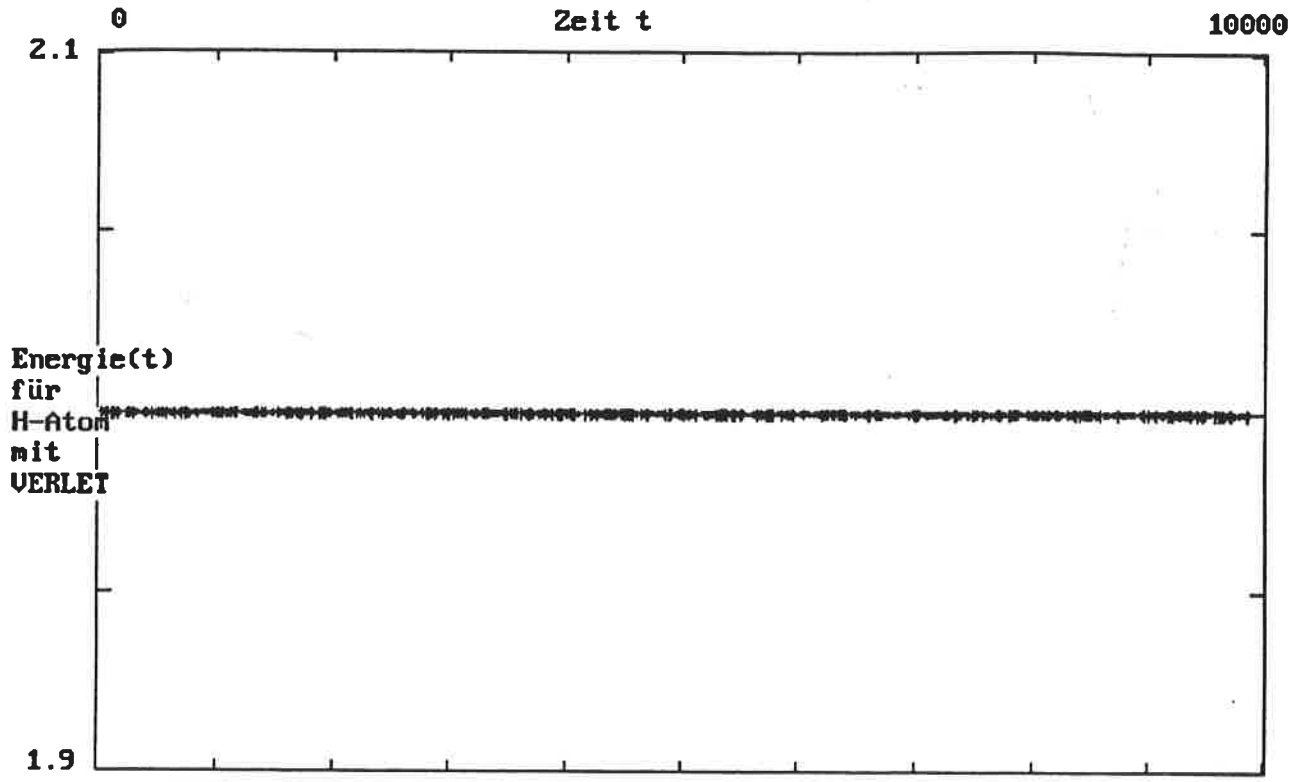


Abb. 35: Energie des H-Atoms mit dem kompakten VERLET-Algorithmus in einer Langzeitstudie mit denselben Daten wie in Abb. 34 gerechnet.

8. Die Ableitung des VERLET-Algorithmus aus der Variation der Wirkung

In diesem Kapitel soll über die Arbeit von R. E. GILLILAN und K. R. WILSON [14] berichtet werden, die mit ihrer Ableitung des VERLET-Algorithmus aus dem Prinzip der Wirkungsminimierung (HAMILTONsches Prinzip) gezeigt haben, daß diesem Algorithmus eine gewisse Sonderstellung zukommt. Wie die Autoren zeigen, lassen sich damit spezielle Einsatzgebiete für den Algorithmus rechtfertigen, die für die Anwendung des Variationsprinzips prädestiniert sind. Namentlich sind dies Reaktionen, dernen Anfangs- *und* Endzustand bekannt sind.

Zunächst aber zur Ableitung selbst. Gegeben sei ein dynamisches System, repräsentiert durch die LAGRANGE-Funktion

$$L(q,p) = 1/2\ p^2 - V(q). \qquad (8.1)$$

Sei weiterhin die Trajektorie q(t) in einer Basis mit Parametern α_i gegeben, d. h. durch

$$q \equiv q(\{\alpha_i\},t).$$

Das HAMILTONsche Prinzip besagt dann, daß die durch

$$dS = Ldt$$

definierte Wirkung S(q(t),p(t)) bezüglich der "wahren" Parameter minimiert wird, d. h.

$$\frac{\partial S}{\partial \alpha_i} = 0 \quad \text{für alle i.} \tag{8.2}$$

Wertet man nun das gegebene Potential nur zu äquidistanten Zeitpunkten aus, und zwar mittels

$$L = 1/2\, p^2 - h\, V(q)\, \Sigma_{n\epsilon Z}\ \delta(t - nh), \tag{8.3}$$

so beschreibt man damit ein freies Teilchen, welches nach jeder Zeiteiheit h einen Impuls proportional zu V(q) zu der gegebenen Zeit erfährt. Während eines Zeitintervalls mit der Dauer h, d. h. zwischen $t_n = nh$ und $t_{n+1} = (n+1)h$, bleibt der Impuls

$$p = \frac{q(t_{n+1}) - q(t_n)}{h}$$

konstant. Bildet man mit dieser zeitlich äquidistant zerlegten LAGRANGE-Funktion das Wirkungsintegral von der Anfangszeit $t = t_n$ bis zu der Endzeit $t = t_{n+m}$, dann ergibt sich wegen der δ-Funktion die Summe

$$S = \frac{1}{2} h \sum_{k=n}^{n+m-1} \left| \frac{q_{k+1} - q_k}{h} \right|^2 - h \sum_{k=n}^{n+m-1} V(q_k).$$

wobei $q(t_n) = q_n$ gesetzt wurde. Diese Wirkung soll nun minimiert werden, wobei die Orte q_k an den Übergangsstellen der Zeitintervalle als Parameter aufgefaßt werden können. Man erhält dann mit (8.2)

$$h \frac{\partial S}{\partial q_k} = -q_{k+1} + 2q_k - q_{k-1} - h^2 \frac{dV(q_k)}{dq_k} = 0. \tag{8.4}$$

Das aber ist wegen $-dV/dq_k = (d/dt)p_k$ nichts anderes als der VERLET-Algorithmus.

Soweit die sehr knapp gehaltene Ableitung. Für nähere Ausführungen sei nochmals auf die Originalarbeit [14] verwiesen. Das wesentliche für die folgenden Anmerkungen ist in dieser Darstellung enthalten. Denn es gilt, daß das hier verwendete Variationsprinzip für die Wirkung nach der "wahren" *Teil*-Trajektorie im Konfigurationsraum frägt. Der Impulsraum wird dabei (explizit) nicht betrachtet. Gewiss hängt p und q über die zeitliche Ableitung miteinander zusammen, variiert aber wird nur q(t). Die auftretende Variaton δp ist lediglich die zeitliche Ableitung der Variation von q und daher nicht unabhängig. Die Folgerung daraus ist somit, daß die hier betrachtete Ableitung eine Berechtigung für die Anwendung des VERLET-Algorithmus auf den Ort, aber eben *nur* auf den Ort, liefert, und zwar durch die in (8.4) gegebene Gleichung. Diese Formel kann nicht ohne weiteres auf die Impulse übertragen werden; diese müssen unabhängig, z. B. durch die Formel (7.5), berechnet werden. Im Lichte des Resultats von Kapitel

8 wurde eine neue Interpretation des mathematischen Resultats von GILLILAN und WILSON ermöglicht.

9. Eine neue Klasse von symplektischen Algorithmen

Dieses Kapitel widmet sich noch einmal der Frage, warum der VERLET-Algorithmus mit approximativer Impulsberechnung als einziger der exakt-reversiblen Algorithmen eine über lange Zeiten brauchbare Integration von HAMILTONschen Systemen gestattet. Außer der im vorhergenden Kapitel gezeigten Ableitbarkeit dieses Algorithmus aus dem HAMILTONschen Prinzip muß eine Eigenschaft des Algorithmus existieren, die ihn von den anderen abhebt. Die Vermutung, daß dies auf die Symplektizität des Algorithmus zurückzuführen ist, liegt nahe. Dieser Begriff soll hier erläutert werden. Eine genaue Analyse des Zusammenhangs der Algorithmen von FREDKINscher Art mit dem Begriff Symplektizität gestattet schließlich die Ableitung einer Transformationsformel, mit deren Hilfe aus der Klasse der exakt-reversiblen Algorithmen eine neue Teilklasse von *symplektischen* Algorithmen erzeugt werden kann. Diese neue Klasse von Algorithmen enthält wiederum exakt-reversible Algorithmen in dem hier behandelten Sinne und stellt insofern eine Unterklasse der Algorithmen von FREDKINscher Art dar.

HAMILTONsche Systeme sind durch Angabe der HAMILTON-Funktion H als Funktion der verallgemeinerten Orte x_μ und Impulse p_μ vollständig beschrieben, wobei der Index μ über die n Freiheitsgrade des Systems läuft. Die Dynamik ist dann durch die kanonischen Gleichungen

$$(d/dt)\, x_\mu = \partial H/\partial p_\mu, \qquad (d/dt)\, p_\mu = -\partial H/\partial x_\mu, \qquad \mu = 1,\ldots,n \tag{9.1}$$

zusammen mit Anfangsbedinungen für die Orte und Impulse determiniert. Die Orts- und Impulsverktoren $\mathbf{x} = (x_1,\ldots,x_n)^T$ bzw. $\mathbf{p} = (p_1,\ldots,p_n)^T$ werden häufig zu einem Phasenraumpunkt $\mathbf{z}$ aus dem Phasenraum Z zusammengefaßt, und die Dynamik dann durch die vektorielle Differentialgleichung

$$(d/dt)\, \mathbf{z} = \mathbf{J}\, \mathrm{grad}_z\, H \tag{9.2}$$

beschrieben. Die 2n-dimensionale Matrix **J** ist hierbei die sogenannte symplektische Einheitsmatrix

$$\mathbf{J} = \begin{vmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{-1} & \mathbf{0} \end{vmatrix} \tag{9.3}$$

mit der n-dimensionalen Einheitsmatrix 1.

Betrachtet man nun eine Transformation T auf dem Phasenraum Z, so wird diese als symplektisch (oder kanonisch) bezeichnet, wenn sie die durch die Gleichungen (9.1) gegebene kanonische Form aufrechterhält. Eine einfache Rechnung zeigt, daß dies durch die Invarianzbedingung für die symplektische Einheitsmatrix

$$\mathbf{J} = (DT)\, \mathbf{J}\, (DT)^T \tag{9.4}$$

gewährleistet ist, wo DT die JACOBI-Matrix der Transformation T darstellt; $(DT)^T$ ist ihre Transponierte.

Nun gibt es für die Beschreibung der Dynamik auf dem Phasenraum, wie im einleitenden zweiten Kapitel bereits erwähnt, die Möglichkeit der Angabe eines Phasenraumflusses (2.2), welcher in dieser speziellen Anwendung auf HAMILTONsche Systeme auch als Propagator bezeichnet wird und die folgende explizite Form hat:

$$\mathbf{z}(t) = e^{-iLt}\,\mathbf{z}(0), \tag{9.5}$$

mit dem LIOUVILLE-Operator

$$L = \{\,.\,,H\} = \Sigma_\mu\; i(\partial H/\partial x_\mu)(\partial/\partial p_\mu) - i(\partial H/\partial p_\mu)(\partial/\partial x_\mu), \tag{9.6}$$

wobei $\{.,H\}$ auch die *POISSON-Klammer* genannt wird (vgl. z. B. [11]).

Ein Integrationsalgorithmus ist eine Diskretisierung eines solchen Propagators und sollte möglichst viele seiner Eigenschaften erhalten. Da der Propagator e^{-iLt} bereits symplektisch ist, ist es naheliegend, dies auch von einem Algorithmus zur iterativen Lösung des Problems (9.1) zu fordern. Berechnet man auf beiden Seiten der Gleichung (9.4) die Determinante, so ergibt sich

$$\det(DT) = 1. \tag{9.7}$$

Unsere reversiblen 2-Schritt-Algorithmen haben die allgemeine Form $\mathbf{z}_{n+1} = \mathbf{T}(\mathbf{z}_n,\mathbf{z}_{n-1})$; es ist also die Determinante der JACOBI-Matrix

$$D(\mathbf{T}(\mathbf{z}_n,\mathbf{z}_{n-1}))$$

zu berechnen, um eine Aussage bezüglich der Symplektizität des zeitdiskreten Algorithmus machen zu können.

Die Algorithmen der FREDKINschen Art haben wegen der Symmetrie die nützliche Eigenschaft, daß

$$\mathbf{z}_n = \mathbf{F}(\mathbf{z}_{n-1}) - \mathbf{z}_{n-2}$$

und daß (9.8)

$$\mathbf{z}_n = \mathbf{F}(\mathbf{z}_{n+1}) - \mathbf{z}_{n+2}.$$

Daraus folgt

$$\partial \mathbf{z}_n/\partial \mathbf{z}_{n-1} = \partial \mathbf{z}_n/\partial \mathbf{z}_{n+1},$$

wobei diese Ableitungen in Kurzschreibweise die Funktionalmatizen symbolisieren sollen. Für die Umkehrmatizen gilt dann

$$\partial z_{n+1}/\partial z_n = \partial z_{n-1}/\partial z_n. \tag{9.9}$$

Mit der Kettenregel schreibt sich die Funktionalmatrix wie folgt:

$$\partial z_{n+1}/\partial z_n = \partial \mathbf{F}(z_n)/\partial z_n - \partial z_{n-1}/\partial z_n.$$

Mit (9.9) läßt sich dieser Ausdrück umformen zu

$$\partial z_{n+1}/\partial z_n = 1/2\ \partial \mathbf{F}(z_n)/\partial z_n. \tag{9.10}$$

Für Symplektizität muß daher gemäß (9.3) gefordert werden:

$$\det[\partial \mathbf{F}/\partial z_n] = 2. \tag{9.11}$$

Das Argument in **F** soll wie in (9.11) im folgenden unterdrückt werden.

Für den allgemeinen Fall wird (9.11) nicht erfüllt sein; durch die Vorschrift

$$\mathbf{F}' = \frac{2^{1/k}}{(\det[\partial \mathbf{F}/\partial z_n])^{1/k}}\ \mathbf{F}, \tag{9.12}$$

wobei k die Anzahl der Variablen ist, läßt sich ein symplektischer Algorithmus

$$z_{n+1} = \mathbf{F}'(z_n) - z_{n-1} \tag{9.13}$$

herstellen, der weiterhin exakt reversibel bleibt, da die Transformation (9.12) nur Glieder in z_n betrifft. Es ist außerdem möglich, nur $j < k$ Komponenten von **F** jeweils mit dem vor **F** stehenden Faktor zu multiplizieren, wobei im Exponenten k durch j ersetzt werden muß.

Betrachten wir speziell hierzu den VERLET-Algorithmus, der noch einmal explizit in seiner Anwendung auf HAMILTONsche Systeme angeführt sei:

$$\begin{aligned} x_{n+1} &= 2x_n + h^2 f(x_n) - x_{n-1} \\ p_{n+1} &= 2p_n + h^2[\partial f(x_n)/\partial x_n]p_n - p_{n-1}, \end{aligned} \tag{9.13}$$

wobei zur Vereinfachung nur ein Freiheitsgrad berücksichtigt wird. Die Funktion f ist die wirkende Kraft. Die Teilabbildungsvorschrift $\mathbf{F}(z_n)$ mit $z_n = (x_n, p_n)^T$ lautet hier

$$F_1(x_n,p_n) = 2x_n + h^2 f(x_n)$$
$$F_2(x_n,p_n) = 2p_n + h^2[\partial f(x_n)/\partial x_n]p_n, \qquad (9.14)$$

mit $F = (F_1,F_2)^T$. Die JACOBI-Matrix für dieses F lautet

$$\partial F/\partial z_n = \begin{bmatrix} \partial F_1/\partial x_n & \partial F_1/\partial p_n \\ \partial F_2/\partial x_n & \partial F_2/\partial p_n \end{bmatrix}. \qquad (9.15)$$

Für diesen speziellen Fall ist $\partial F_1/\partial p_n = 0$. Die Komponente $\partial F_2/\partial x_n$ ist daher unerheblich, da nur die Determinante dieser Matrix von Interesse ist. Die Komponenten von Interesse sind also

$$\partial F_1/\partial x_n = 2 + h^2 \partial f/\partial x_n = \partial F_2/\partial p_n.$$

Hält man an der ersten Komponente $\partial F_1/\partial x_n$ fest, so muß für Symplektizität gefordert werden, daß die zweite Diagonalkomponente lautet:

$$\partial F_2'/\partial p_n = \frac{2}{\partial F_1/\partial x_n}. \qquad (9.16)$$

Hierdurch wird ein F_2' definiert, das sich wie folgt berechnet:

$$F_2' = \frac{2}{\partial F_2/\partial p_n} F_2$$
$$= \frac{2p_n[2 + h^2(\partial f/\partial x_n)]}{2 + h^2(\partial f/\partial x_n)} \qquad (9.17)$$
$$= 2p_n.$$

Mit der so erhaltenen neuen Formel für die Impulsberechnung,

$$p_{n+1} = 2p_n - p_{n-1}, \qquad (9.18)$$

erhält man einen symplektischen Algorithmus. Betrachten wir jetzt die von VERLET vorgeschlagene Formel zur approximativen Impulsberechnung (7.5). Aus ihr ergibt sich

$$p_{n+1} = (x_{n+2} - x_n)/2h$$

und

$$p_{n-1} = (x_n - x_{n-2})/2h. \tag{9.19}$$

Setzt man (9.19) in (9.18) ein, so folgt

$$p_n = (x_{n+2} - x_{n-2})/4h$$

also eine Übereinstimmung mit (9.18). Der VERLET-Algorithmus ist in dieser Form also symplektisch.

Diese Symplektizität wurde mithilfe der Transformationsformel (9.17) hergestellt, die einen Spezialfall der allgemeinen Transformationsvorschrift (9.12) darstellt, und zwar unter der Bedingung, indem die Abbildungsvorschrift für die x-Komponente beibehalten wird. Dieser Speziallfall ist selbstverständlich nur bei HAMILTONschen Systemen verwirklichbar. Für nicht-HAMILTONsche Systeme läßt sich übrigens mithilfe der Vorschrift

$$z_{n+1} = \frac{2^{1/k}\,[F^h(z_n) + F^{-h}(z_n)]}{\left| \det \dfrac{\partial [F^h(z_n) + F^{-h}(z_n)]}{\partial z_n} \right|^{1/k}} - z_{n-1} \tag{9.20}$$

immer ein Algorithmus von FREDKINscher Art herstellen, dessen Funktionaldeterminante eins wird. F^h und F^{-h} bezeichnen hier einen beliebigen Einschritt-Algorithmus bzw. seine numerische Umkehrung, wie in Kapitel 3 bereits zur Konstruktion der Algorithmen der FREDKINschen Art genau erläutert wurde. Es handelt sich dann um divergenzfreie Differenzengleichungen, die demzufolge divergenzfreie dynamische Systeme zu simulieren imstande sein müßten. Auch hier gilt, wie im Anschluß an Gleichung (9.12) erläutert, daß auch j Komponenten mit dem Faktor multipliziert werden können, wobei n durch $j < n$ ersetzt wird. Mit Gleichung (9.20) ist eine neue Klasse von symplektischen bzw. allgemein divergenzfreien Algorithmen beschrieben, die zugleich exakt reversibel sind.

Es gibt eine Anzahl neuerer Arbeiten über symplektische Algorithmen anderer Art (z. B. [16,17]. Der Tenor dieser Arbeiten ist, daß von symplektischen Algorithmen eine Langzeitstabilität erwartet wird, denn Nicht-Symplektizität zerstört die Langzeitstatistik des Flusses, was auf die Nicht-Invarianz des sogenannten POINCARE-Integrals zurückzuführen sei. Die Invarianz dieses Integrals bedeutet im wesentlichen die Erhaltung des Phasenraumvolumens; es handelt sich um die Verallgemeinerung des LIOUVILLEschen Satzes. Mit der obigen Konstruktionsvorschrift besteht nun erstmals die Möglichkeit, aus jedem beliebigen Einschrittalgorithmus einen sowohl symplektischen wie zusätzlich exakt-reversiblen Algorithmus zu konstruieren.

Eine wichtige Frage bleibt noch zu klären: Verbleibt die mit dem symplektischen VERLET-Algortihmus simulierte Trajektorie lokal stets im "wahren" Richtungsfeld im Phasenraum?

Erinnern wir uns an die Tatsache, daß die symmetrischen Algorithmen - aufgrund ihrer Abhängigkeit von zwei vorhergehenden Iterationspunkten - parasitäre Lösungen einbringen können. Im Extremfall rechnen diese Algorithmen sogar rückwärts - entgegen dem gegebenen

Richtungsfeld. Das einfachste Beispiel zur Demonstration dieses Sachverhaltes ist durch die Differentialgleichung

$$\begin{aligned} (d/dt)\, x &= p \\ (d/dt)\, p &= x \end{aligned} \qquad (9.21)$$

gegeben. Dieses HAMILTONsche System besitzt einen Sattelpunkt im Ursprung. In Kapitel 6 wurde das Verhalten der (nicht-symplektischen) reversiblen Algorithmen für den Fall, daß ein Anfangswert auf der Nebendiagonalen (durch den zweiten und vierten Quadranten) gewählt wird, geschildert. Die Simulationstrajektorie läuft durch den Sattelpunkt und von da an gegen das Richtungsfeld.

Zieht man ins Kalkül, daß die Punkte auf einer Trajektorie, die für das zeitinvertierte System simuliert wird, sowohl der Energieerhaltung unterliegen als auch symplektisch sein sollten, d. h. Ort und Impuls kanonisch konjugiert sind, so stellen etwaige parasitäre Lösungen eines symplektischen zeitreversiblen Algorithmus keinen Widerspruch zur Symplektizität dar. Dennoch wäre eine solch Simulation unbrauchbar, da sie definitiv falsch ist. Das Verhalten von symplektisch-reversiblen Algorithmen ist im Hinblick auf die parasitären Lösungen daher zu prüfen.

Bleiben wir zunächst bei dem einfachen System (9.21) mit dem Sattelpunkt. Wählt man die Anfangswerte $x_0 = 1$ und $p_0 = -1$, d. h. einen Punkt auf der Diagonalen im vierten Quadranten, so zeigt sich das in Abb. 36 wiedergegebene Verhalten. Für verschiedene Zeitschritte h (hier: h = 0.5, 0.1, 0.02, 0.004) "knickt" die Trajektorie in unmittelbarer Nähe des Fixpunktes ab und verläuft anschließend auf der anderen Diagonalen (durch den dritten Quadranten) weiter, d. h. stets "richtig" im Sinne des Richtungsfeldes. Eine Verkleinerung von h führt dazu, daß noch näher an den Fixpunkt herangerechnet und der "Knick" entsprechend schärfer wird. Die simulierte Trajektorie weicht dem Richtungsfeld in der Nähe des Fixpunktes auf eine eng benachbarte Trajektorie aus, verbleibt aber anschließend im "wahren" Richtungsfeld.

Der Fixpunkt verhält sich, anschaulich gesprochen, als ob von ihm ein repellierendes Potential ausginge, das mit dem Zeitschritt h verkleinert werden kann und im Limes h gegen 0 verschwindet. Dieses Verhalten ist durchaus akzeptabel, da Strukturen im Phasenraum richtig wiedergegeben werden. Zu erklären ist dieser Unterschied zum nicht-symplektischen VERLET-Algorithmus damit, daß die Impulsberechnung auf die Ortsiterierten zurückgeführt wird. Die Folge der Ortsiterierten verläuft monoton von positiven zu negativen Werten, so daß die in der approximativen Impulsformel auftretende Differenz $(x_{n+1} - x_{n-1})$ und damit die Impulse selbst immer negativ sind. Beim vollständigen VERLET-Algorithmus werden die Impulse unabhängig iterativ bestimmt, so daß auch hier eine monotone Folge von negativen zu positiven Impulswerten auftritt.

In komplexen Systemen ist es nicht mehr möglich, anhand des Simulationsergebnisses selbst zu beurteilen, ob dieses richtig ist oder nicht, denn i. a. ist der Verlauf des Richtungsfeldes nicht bekannt und soll gerade mittels einer solchen Simulation ermittelt werden. Anhand der in Kapitel 8 untersuchten Systeme, dem eindimensionalen Modellgas und dem diamagnetischen Wasserstoffatom, soll daher der symplektische VERLET-Algorithmus bezüglich der Abweichung seiner Simulationstrajektorie vom Richtungsfeld getestet werden. Als Testkriterium möge die Winkelabweichung eines nach jedem Zeitschritt ermittelten Trajektorienstückes zur Tangente an das lokale Richtungsfeld dienen. Die Tangente soll mittels des EULER-Verfahrens berechnet werden, dessen Genauigkeit für die hier verfolgten Zwecke genügt. In den Abbildungen 37 - 39 sind die Ergebnisse dieser Überprüfung dargestellt. Abb. 37 zeigt die Winkelabweichung für das diamagnetische H-Atom (im irregulären Fall mit $\varepsilon = -0.1$), und zwar wurde der Kosinus des

Winkels gegen die Zeit aufgetragen. Mit dem gewählten Zeitschritt h = 0.01 ergeben sich statistische Schwankungen der Winkelabweichung, die aber durch die Wahl eines kleineren h unter jede beliebige Schranke gedrückt werden können, wie sich in Abb. 39 andeutet (es wurde h = 0.005 verwendet). Es "schleichen" sich keine parasitären Lösungen ein, die zu einem ständigen Wegdriften aus dem Richtungsfeld führen würden.

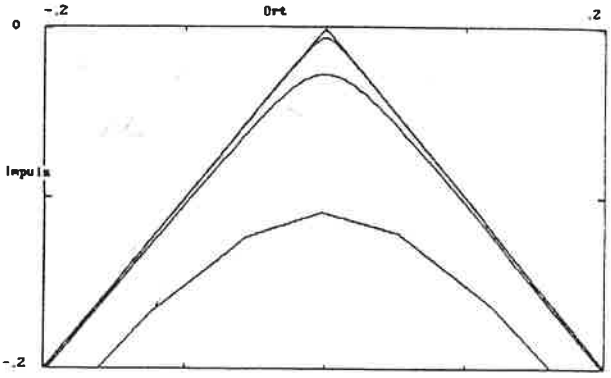


Abb. 36: Das Verhalten des symplektischen VERLET-Algorithmus in der Nähe des Sattelpunktes von System (9.21). Gewählte Zeitschritte: h = 0.5, 0.1, 0.02, 0.004. Die Trajektorie verbleibt im "wahren" Richtungsfeld.

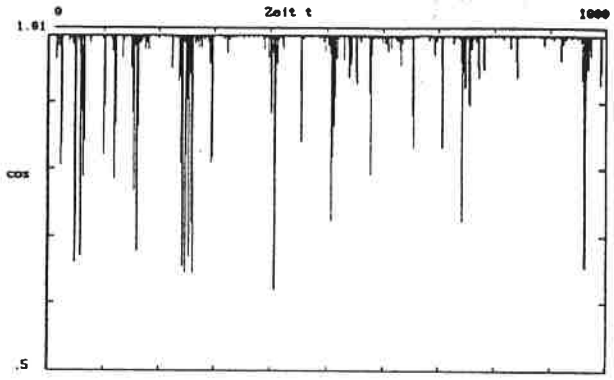


Abb. 37: Die Winkelabweichung vom Richtungsfeld für das H-Atom (irregulärer Fall) mit dem Zeitschritt h = 0.01.

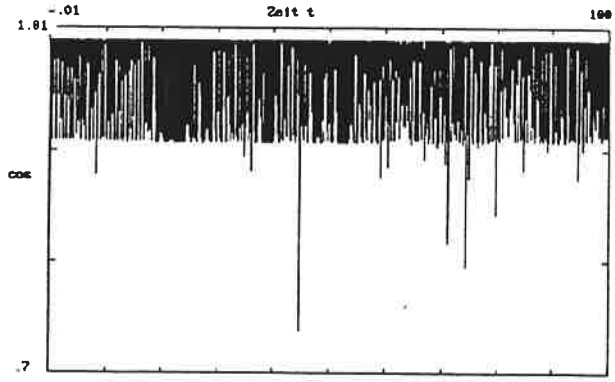


Abb. 38: Winkelabweichung im Richtungsfeld für das 1D-Modellgas mit dem Zeitschritt h = 0.001 gerechnet. Weichheit μ = 0.01.

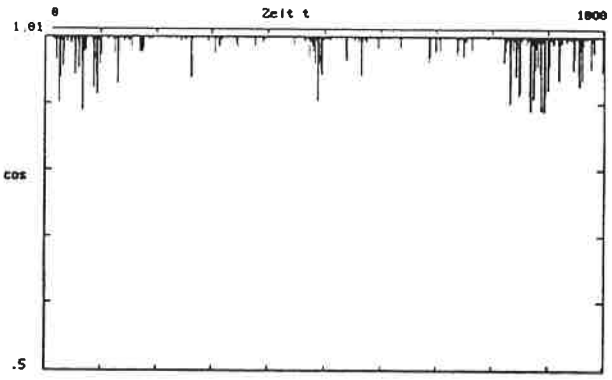


Abb. 39: Winkelabweichung für das H-Atom mit h = 0.005 zum Vergleich mit Abb. 36.

Für das eindimensionale Modellgas ist die Winkelabweichung vom lokalen Richtungsfeld in Abb. 38 dargestellt. Mit dem gewählten Zeitschritt h = 0.001 weicht der Kosinus des Winkels nur unwesentlich von eins ab, d. h. auch hier verläuft die Simulationstrajektorie lokal immer im "wahren" Richtungsfeld. Die Integrationszeit von 100 Zeiteinheiten (was einer realen Rechenzeit von ca. 24 Stunden auf dem PC entspricht) ist sehr lang und daher wohl ebenfalls repräsentativ.

Ergebnis dieser Studie über die Abweichung der Simulationstrajektorie vom Richtungsfeld ist somit, daß der symplektische VERLET-Algorithmus seinem nicht-symplektischen Pendant außer der Symplektizität noch die weitere Eigenschaft voraus hat, daß keine systematischen Abweichungen vom Richtungsfeld vorhanden sind.

Es ist dies zugleich ein weiteres Indiz dafür, daß durch die Konstruktion von Algorithmen mit bestimmten Dissipationsfaktoren auch dissipative Systeme "vernünftig" simuliert werden können.

Es sei zum Abschluß dieses Kapitels noch auf eine interessante Ableitung von M. TUCKERMAN et al [11] hingewiesen, die in eindrucksvollerweise Weise die enge Verknüpfung von Symplektizität und Zeitreversibilität schildern (ohne allerdings die Symplektizität explizit zu erwähnen).

Die Autoren bedienen sich dabei des sogenannten TROTTER-Theorems, welches z. B. in Anwendung auf den Propagator e^{-iLt} eine diskrete Approximation für diesen liefert. Das Theorem besagt, daß die Elemente von kontinuierlichen Halbgruppen, deren Generatoren sich in eine Summe zerlegen lassen, faktorisierbar sind. Für unseren Fall bedeutet das für die additive Zerlegung des LIOUVILLE-Operators

$$iL = iL_1 + iL_2$$

eine Faktorisierbarkeit des Propagators in der Form

$$\begin{aligned} e^{-i(L_1+L_2)t} &= [e^{-i(L_1+L_2)t/P}]^P \\ &= [e^{-iL_1h/2}\, e^{-iL_2h}\, e^{-iL_1h/2}]^P + O(t^3/P^2), \end{aligned}$$

wobei $h = t/P$ bedeutet und O ein verschwindendes Restglied ist. Mit der Notation $U_\mu = e^{-iL_\mu t}$, wobei $\mu = 1,2$, läßt sich hieraus der diskrete Zeitpropagator

$$G(h) = U_1(h/2)\, U_2(h)\, U_1(h/2)$$

definieren, der durch explizites Einsetzen von (7.6) und Anwendung auf den Phasenraumpunkt **z** den VERLET-Algorithmus liefert.

Daß die Verwendung von zeitreversiblen Zerlegungen von ursprüglich zeitreversiblen Propagatoren wiederum auf zeitreversible Algorithmen führt, ist klar. Daß dabei aber ein Algorithmus speziell von FREDKINscher Art abgeleitet werden kann, ist zunächst erstaunlich. Zieht man ins Kalkül, daß der Phasenraumfluß samt seiner Diskretisierung symplektisch ist, so wird dieses Ergebnis allerdings verständlich. Erneut erweist sich der VERLET-Algorithmus als Besonderheit in dem Sinne, daß er aus ganz unterschiedlichen Gesichtspunkten heraus abgeleitet werden kann; hier durch eine unmittelbare Diskretisierung des Flusses.

10. Zusammenfassung und Diskussion

Es wurde eine Klasse von exakt-reversiblen Algorithmen für Dynamik-Simulationen vorgestellt und untersucht. Die Algorithmen dieser Klasse haben die allgemeine, symmetrische Struktur

$$\mathbf{z}_{n+1} = \mathbf{F}^h(\mathbf{z}_n) \mp [\mathbf{F}^{-h}(\mathbf{z}_n) - \mathbf{z}_{n-1}]. \qquad (10.1)$$

Hierbei symbolisieren die Abbildungsvorschriften $\mathbf{F}^h$ beliebige Einschrittalgorithmen und $\mathbf{F}^{-h}$ die zugehörigen numerischen Umkehrungen, d. h. dieselben Algorithmen, jedoch unter Verwendung negativer Zeitschritte. Der Ausdruck in der eckigen Klammer in (10.1) stellt einen Korrektur- oder besser: Gedächtnisterm dar, der eine Information über den vorhergehenden Iterationspunkt einbringt.

Die Algorithmen dieser Form (10.1), die *Algorithmen FREDKINscher Art*, sind im Integer-Modus *exakt-reversibel* programmierbar; alle Iterationspunkte lassen sich exakt reproduzieren. Die Frage der exakten Rückrechenbarkeit von Algorithmen wurde offenbar zuerst von FREDKIN (unpubliziert) und RÖSSLER [10] gestellt. Zwei unerwartete Antworten wurden gefunden. Erstens: sie existiert, zweitens: sie kann beliebig lang ohne Verschlechterung durchgeführt werden. Das erste Teilergebnis folgt bereits aus FREDKINS oben im vierten Kapitel erwähnter persönlicher Mitteilung, das zweite ist neu und unerwartet.

Allerdings können die Algorithmen dieses Typs i. a. nicht für dissipative Systeme verwendet werden. Dissipative Systeme besitzen einen Attraktor und sind daher nicht auf dem gesamten Definitionsbereich umkehrbar. Mathematisch wird es im Limes t gegen ∞ problematisch. Numerisch bekommt man bereits nach endlicher Rechenzeit Schwierigkeiten, da man immer auf einem diskreten Raster rechnet. Eine Verkleinerung der Rasterstruktur hat ihre Grenzen beim endlichen Speicherplatz und an der endlichen Rechenzeit. Aus demselben Grund läßt sich mit diesen Algorithmen auch nicht beliebig nahe an Extremalstellen heranrechnen. Die Folge ist ein flacherer Verlauf der simulierten Funktion an solchen Stellen; eine Tatsache, die jedoch keine wesentliche Rolle in der Anwendung spielt.

Ein erstes Ergebnis war also, daß die Algorithmen von FREDKINscher Art nur für die Integration von divergenzfreien Systemen verwendet werden können. Für divergenzfreie Systeme

$$(d/dt)\mathbf{z} = \mathbf{f}(\mathbf{z}); \qquad \mathrm{div}\ \mathbf{f} = 0$$

erwartet man eine besonders gute Simulation durch Algorithmen, die auf ihrerseits divergenzfreie Differenzengleichungen

$$\mathbf{z}_{n+1} = \mathbf{T}(\mathbf{z}_n, \mathbf{z}_{n-1}); \qquad \det[\mathbf{DT}] = \det[\mathbf{D}\mathbf{z}_{n+1}] = 1$$

führen. (Hierbei symbolisiert $\mathrm{D}[\mathbf{z}_{n+1}]$ die Funktionalmatrix der Abbildung $\mathbf{T}$, die hier aus naheliegenden Gründen speziell als Zweischrittalgorithmus gewählt wurde). Sei nun diese Abbildung speziell von der symmetrischen Struktur wie sie durch die FREDKINschen Algorithmen gegeben sind, d. h.

$$\mathbf{z}_{n+1} = \mathbf{F}(\mathbf{z}_n) - \mathbf{z}_{n-1},$$

dann gilt für ihre Funktionaldeterminante

$$\det[\mathbf{D}\mathbf{z}_{n+1}] = 1/2\ \det[\mathbf{DF}(\mathbf{z}_n)].$$

Mit diesem Zusammenhang ließ sich eine neue Klasse von *divergenzfreien Algorithmen* ableiten, die im besonderen auch die *symplektischen Algorithmen* für HAMILTONsche Systeme enthalten. Als zu dieser Klasse gehörend erwies sich der bekannte VERLET-Algorithmus in seiner

kompakten Form mit approximativer Impulsberechnung. In den durchgeführten "Computer-Experimenten" ließ sich die hervorragende Güte dieses Algorithmus bestätigen. Es wurde auf eine Arbeit von TUCKERMAN et al verwiesen, die den VERLET-Algorithmus direkt aus einer Diskretisierung des symplektischen Phasenraumflusses ableiteten. Diese Ableitung wird im Lichte der hier durchgeführten Betrachtungen zur Symplektizität verständlich. Auch die Arbeit von GILLILAN und WILSON, die das HAMILTONsche Prinzip zur Ableitung des VERLET-Algorithmus verwendeten, erscheint in neuem Licht. Das Variationsprinzip des Wirkungsintegrals scheint ein recht mächtiges Instrumentarium und die nähere Beschäftigung mit diesem Prinzip eine lohnende Aufgabe zu sein.

Der Anwendungsbereich der Algorithmen von FREDKINscher Art ist also i. a. auf nicht-dissipative Systeme beschränkt. Rechnungen an Systemen mit ungerader Dimension können i. a. ebenfalls nicht befriedigend ausgeführt werden. Z. B. hat das divergenzfreie Gleichungssystem

$$\begin{aligned} (d/dt)x &= -x -y \\ (d/dt)y &= x \\ (d/dt)z &= a(y - y^2) \end{aligned} \qquad (10.2)$$

mit den Anfangswerten $x_0 = 0.3$, $y_0 = 0$, $z_0 = -0.25$ und dem Parameter $a = 0.2$ toroidales Verhalten [15]. Tatsächlich liefern alle Algorithmen von FREDKINscher Art Trajektorien, die anfänglich auf dem Torus liegen, aber dann nach gewisser Integrationszeit von ihm wegdriften. Das gilt auch für den VERLET-Algorithmus, der in seiner approximativen Form hier natürlich nicht verwendet werden kann, da es sich um ein Nicht-HAMILTONsches System handelt.

Verwendet man für die Integration eines solchen Systems aber Algorithmen aus der Teilklasse der FREDKINschen Algorithmen, namentlich die divergenzfreien Algorithmen nach der obigen Konstruktionsvorschrift (9.20), so sind befriedigende Ergebnisse zu erwarten.

Darüber hinaus kann sogar an die Integration von dissipativen Systemen mithilfe exakt-reversibler Algorithmen gedacht werden. Denn eine zu (9.20) analoge Formel könnte auch auf Algorithmen transformieren, die über eine von eins verschiedene Funktionaldeterminante verfügen. Es ist zu erwarten, daß jeder Dissipationsfaktor hergestellt werden kann, um eine Differentialgleichung mit demselben Dissipationsfaktor "systemgerecht" zu integrieren. Beispielsweise wäre das Gleichungssystem (10.2) dissipativ mit dem Dissipationsfaktor -b, wenn man die dritte Gleichung ändert zu

$$(d/dt)z = a(y - y^2) -bz.$$

Es besteht also die berechtigte Hoffnung, mit den hier untersuchten exakt-reversiblen Integrationsalgorithmen und ihren geplanten Erweiterungen dem "Geheimnis" der Reversibilitätseigenschaften von dynamischen Systemen auf die Spur zu kommen. Insbesondere erscheint dem Autor eine Überprüfung der Ableitungen von HURLEY interessant, wie bereits in der Einleitung angesprochen wurde. Die Thermodynamik der (makroskopisch) irreversiblen Prozesse ist von so bizarrer Natur, daß jeder Versuch, zu einem besseren Verständnis beizutragen, lohnend erscheint.

Das vielleicht wichtigste Ergebnis der vorliegenden Arbeit ist die Erkenntnis, daß beliebig lange HAMILTONsche Trajektorien in gleichbleibender approximativer Qualität erzeugt werden können, die exakt reversibel sind. Dadurch wird es möglich, kritische oder interessante Übergänge in klassisch-mechanischen Vielteilchen-Problemen exakt durchzuuntersuchen mithilfe

numerischer und chaostheoretischer Untersuchungen. Die kinetische Theorie wird dadurch wieder aktuell.

Anhang

Hier sind die für die Arbeit verwendeten Q-Basic-Programme zusammengestellt. Im wesentlich wurde mit dem Programm VARINT.BAS gearbeitet. Es wurde so konzipiert, daß alle hier relevanten Algorithmen darin zur Auswahl enthalten sind. Fernerhin enthält dieses Programm einige Differentialgleichungs-Systeme zur Auswahl. Die Auswahl hat hierbei über die Aktivierung bzw. Stillegung bestimmter Programmzeilen zu erfolgen. Das Programm H-ATOM.BAS ist eine umgestellte Version von VARINT.BAS speziell für das diamagnetische H-Atom zugeschnitten. Das Titelbild wurde mit dem Programm LOG.BAS erstellt. Gerechnet wurde stets auf dem PC "MICROWAY AT 386". Die Programme mögen umständlich erscheinen; hierbei ist zu bedenken daß sie sozusagen im Laufe der Zeit wuchsen und nicht mehr eigens aufbereitet wurden.

VARINT.BAS:

```
CLS
stopp = 0
PRINT "VARINT ist ein Programm zur Integration von ODE's, wobei der"
PRINT "Algorithmus aus untenstehender Auswahl beliebig gewaehlt werden kann."
PRINT "Zur Verfuegung stehen:"
PRINT
PRINT "RK4=1,  RK4(rev+)=2,  RK4(rev-)= 3,  VERLET1=4,  EULER=5,"
PRINT "Midpoint,  VERLET2=7,  TAYLOR(2.Ordnung)=8"
PRINT
PRINT "VERLET1 berechnet jede Variable mit x(n+1) = 2*x(n) + h^2 * f(x(n)) - x(n-1),"
PRINT "wogegen VERLET2 die Impulse approximativ mit p(n) = (x(n+1) - x(n-1)) / (2*h)"
PRINT "berechnet."
PRINT
INPUT "gewaehlter Algorithmus = "; alg

'*****************************************************************************
'Dimensionierung der Variablen:
'*****************************************************************************

DIM Anfangswert#(13)  'Die Anfangswerte.
DIM f#(12, 6)         'Die Kraefte. Es bedeutet: erste 6 = Anzahl der ODE's
                      'zweite 6 = Grad der auftretenden Ableitungen
DIM x#(12, -1 TO 1)   'Die Variablen.Die zweite dim-Zahl gibt die
                      '3 aufeinanderfolgenden Iterationswerte an
DIM k#(4, 12), d#(12)  'Parameter fuer RK4
DIM y#(12, 2)         'Zwischenterme fuer RK4p; VERLET1 und EULERrev
DIM p#(12)            'Die approximativen Impulse in VERLET2
DIM yBeschriftung$(5)

'*****************************************************************************
'Die alten Daten fuer Parameter und Bildschirmdefinition
'aus den Dateien varint1.dat, varint2.dat:
'*****************************************************************************
OPEN "varint1.dat" FOR BINARY AS #1
```

```
  GET #1, , h#
  GET #1, , e#
  GET #1, , t#

  FOR j = -1 TO 1 STEP 1
   FOR i = 1 TO 12
    GET #1, , x#(i, j)
   NEXT i
  NEXT j

  FOR i = 1 TO 13
   GET #1, , Anfangswert#(i)
  NEXT

  FOR j = 1 TO 6
   FOR i = 1 TO 12
    GET #1, , f#(i, j)
   NEXT i
  NEXT j

  GET #1, , xmax
  GET #1, , xmin
  GET #1, , ymax
  GET #1, , ymin
  GET #1, , xAbschnitt
  GET #1, , yAbschnitt

 CLOSE #1

 PRINT "Die alten Daten lauten:"
 PRINT "Zeitschritt h="; h#, "Parameter e="; e#

'*****************************************************************************
'Abfragen, ob Aenderungen vorgenommen werden sollen:
'*****************************************************************************

  INPUT "Soll der Timestep h geaendert werden?   ja=1   nein=0  "; Aenderung1
  IF Aenderung1 = 0 THEN GOTO 10
  INPUT "h=", h#                '******  Zeitschritt ******

10 INPUT "Soll der Parameter e geaendert werden?  ja=1    nein=0", Aenderung2
  IF Aenderung2 = 0 THEN GOTO 30
  INPUT "e=", e#      '*****  Wird fuer chaotische Glch. nach Roessler und H-Atom gebraucht

30 INPUT "Soll ein Neustart vorgenommen oder mit den alten Daten weitergerechnet werden?   Neu=1   weiter=0
", neu
  IF neu = 0 THEN GOTO 70

35 PRINT " Die bisherigen Anfangswerte lauten:"

  FOR i = 1 TO 13
   PRINT "x"; SPC(0); i; SPC(0); ",0 = "; Anfangswert#(i)
  NEXT

  PRINT "x13,0 = Anfangszeit"
  INPUT "Sollen die Anfangswerte geaendert werden?   ja=1    nein=0", Aenderung5
  IF Aenderung5 = 0 THEN GOTO 60

  FOR i = 1 TO 13
```

```
    PRINT "x"; i; "0 = "; : INPUT Anfangswert#(i)
  NEXT

'*****************************************************************
60 'Bestimmung des ersten Wertepaars:
'*****************************************************************

  t# = Anfangswert#(13)

  FOR i = 1 TO 12
    x#(i, 0) = Anfangswert#(i)
  NEXT

  GOSUB force
  FOR i = 1 TO 12
    k#(1, i) = h# * f#(i, 1)
    d#(i) = x#(i, 0)
    x#(i, 0) = d#(i) + k#(1, i) / 2#
  NEXT

  GOSUB force
  FOR i = 1 TO 12
    k#(2, i) = h# * f#(i, 1)
    x#(i, 0) = d#(i) + k#(2, i) / 2#
  NEXT

  GOSUB force
  FOR i = 1 TO 12
    k#(3, i) = h# * f#(i, 1)
    x#(i, 0) = d#(i) + k#(3, i)
  NEXT

  GOSUB force
  FOR i = 1 TO 12
    k#(4, i) = h# * f#(i, 1)
    x#(i, 1) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
    x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
  NEXT

  t# = t# + h#

'*****************************************************************
'Hier beginnt die Bildschirmdefinition:
'*****************************************************************

70 PRINT "Die bisherigen Bildschirmdaten:"
  PRINT "xmax="; xmax, "xmin="; xmin, "ymax="; ymax, "ymin="; ymin, "x-Abschnitt="; xAbschnitt, "y-
Abschnitt="; yAbschnitt
  INPUT "Sollen die Achsen geaendert werden?   ja=1    nein=0", Aenderung
  IF Aenderung = 0 THEN GOTO 75
  INPUT "maximaler x-Wert=", xmax
  INPUT "minimaler x-Wert=", xmin
  INPUT "maximaler y-Wert=", ymax
  INPUT "minimaler y-Wert=", ymin
  INPUT "x-Abschnitt=", xAbschnitt
  INPUT "y-Abschnitt=", yAbschnitt

75 OPEN "varint2.dat" FOR INPUT AS #2
```

```
    INPUT #2, xBeschriftung$

    FOR j = 1 TO 5
      INPUT #2, yBeschriftung$(j)
    NEXT j

   CLOSE #2

  PRINT
  PRINT "Die bisherige Beschriftung lautet:"
  PRINT "   für die x-Achse:   "; xBeschriftung$
  PRINT "   für die y-Achse:   ";

  FOR i = 1 TO 5
    PRINT yBeschriftung$(i);
  NEXT

  PRINT : PRINT
  INPUT "Soll die Beschriftung geaendert werden?  ja = 1    nein = 0", Aenderung
  IF Aenderung = 0 THEN GOTO 80
  INPUT "Die x-Beschriftung:", xBeschriftung$
  PRINT "Die y-Beschriftung (es stehen 5 Zeilen mit je ca. 5 Buchstaben zur Verfügung):"

  FOR i = 1 TO 5
    INPUT yBeschriftung$(i)
  NEXT

80 GOSUB Bildschirm

'*******************************************************************************
'Anweisungen zu welchem Algorithmus gesprungen wird:
'*******************************************************************************

  IF alg = 1 THEN GOTO RK4
  IF alg = 2 THEN GOTO RK4p
  IF alg = 3 THEN GOTO RK4n
  IF alg = 4 THEN GOTO VERLET1
  IF alg = 5 THEN GOTO EULER
  IF alg = 6 THEN GOTO Midpoint
  IF alg = 7 THEN GOTO VERLET2
  IF alg = 8 THEN GOTO Taylor

'*******************************************************************************
VERLET1:
'*******************************************************************************

  KEY(2) ON
  GOSUB force

  GOTO 200  'Dieser Sprung muß stillgelegt werden, wenn man kompakt rechnen will!!!

'Der Algorithmus kompakt:
'------------------------

  FOR i = 1 TO 4 STEP 1
    x#(i, 1) = 2# * x#(i, 0) + (h# ^ 2#) * f#(i, 2) - x#(i, -1)
  NEXT i

  GOTO 300
```

```
200
'Der Algorithmus mit Zwischentermen (fuer den "Gedaechtnisterm"):
'-------------------------------------------------------
 FOR i = 1 TO 6
  y#(i, 1) = x#(i, 0) + h# * f#(i, 1) + (h# ^ 2# / 2#) * f#(i, 2)
  y#(i, 2) = x#(i, 0) - h# * f#(i, 1) + (h# ^ 2# / 2#) * f#(i, 2)
  x#(i, 1) = y#(i, 1) + y#(i, 2) - x#(i, -1)
 NEXT i

300
 t# = t# + h#
'Berechnung der Energie für Hamiltonsche Systeme:
'-------------------------------------------
 Energie# = x#(4, 1) ^ 2# / 2# + x#(6, 1) ^ 2# / 2# + e# / x#(1, 1) + e# / (1# - x#(1, 1)) + e# / x#(3, 1) + e# /
(.2# - f1# - x#(3, 1))

'Anweisungen fuer die Graphik (Auswahl durch Enfernung der REM-Marken):
'-------------------------------------------------------------------

  'PSET (xmin + t#, ymin + (y#(1, 2) - x#(1, -1)))
  'PSET (x#(3, 1), x#(4, 1))
  'PSET (x#(1, 1), x#(2, 1))
  PSET (t#, Energie#)

  'Poincare-Schnitt:
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

'Zum Zeichnen eines lokalen Richtungsfeldes:
'-------------------------------------
 GOTO 301
  h1# = .01
  t1# = t#: t2# = t#: q#(1, 1) = x#(1, 0): q#(1, 2) = x#(1, 0)
  FOR i = 1 TO 2
   FOR j = 1 TO 100
    q#(1, 1) = q#(1, 1) + h1# * e# * q#(1, 1) + (h1# ^ 2# / 2#) * e# ^ 2# * q#(1, 1)
    q#(1, 2) = q#(1, 2) - h1# * e# * q#(1, 2) + (h1# ^ 2# / 2#) * e# ^ 2# * q#(1, 2)
    t1# = t1# + h1#
    t2# = t2# - h1#
    PSET (t1#, q#(1, 1)), 1
    PSET (t2#, q#(1, 2)), 2
   NEXT
  NEXT

301
  FOR i = 1 TO 6
   x#(i, -1) = x#(i, 0): x#(i, 0) = x#(i, 1)
  NEXT i

 ON KEY(2) GOSUB Ende1
 'IF t# >= 100 THEN gosub Ende1
 IF stopp = 1 THEN GOSUB Ende

GOTO VERLET1

'*************************************************************************
VERLET2:
'*************************************************************************
```

```
  KEY(2) ON
  GOSUB force
  t# = t# + h#

   FOR i = 1 TO 3 STEP 1
    x#(i, 1) = 2# * x#(i, 0) + h# ^ 2# * f#(i, 2) - x#(i, -1)
   NEXT i

'Poincare-Schnitt:
'-----------------
  'IF x#(1, 0) >= 0 AND x#(1, -1) <= 0 THEN GOSUB pcs
  'IF x#(1, 0) >= 0 AND x#(1, -1) <= 0 THEN GOSUB pcs

'Zur Berechnung der approximativen Impulse:
'-----------------------------------
   FOR i = 1 TO 3 STEP 1
    p#(i) = (x#(i, 1) - x#(i, -1)) / (2# * h#)
    x#(i, -1) = x#(i, 0): x#(i, 0) = x#(i, 1)
   NEXT i

'Die Berechnung der Energie:
'--------------------------
  Energie# = p#(1) ^ 2# / 2# + p#(3) ^ 2# / 2# + e# / x#(1, 1) + e# / (1# - x#(1, 1)) + e# / x#(3, 1) + e# / (.2# -
f1# - x#(3, 1))

'Für die Graphik:
'----------------
  PSET (t#, Energie#)
  'PSET (p#(1), p#(2))
  'PSET (x#(1, 0), p#(1))

  ON KEY(2) GOSUB Ende1
  IF stopp = 1 THEN GOSUB Ende

GOTO VERLET2

'*******************************************************************************
EULER:
'*******************************************************************************

  KEY(2) ON
  GOSUB force
  t# = t# + h#

   FOR i = 1 TO 4
    x#(i, 1) = x#(i, 0) + h# * f#(i, 1)
   NEXT

   FOR i = 1 TO 4
    x#(i, 0) = x#(i, 1)
   NEXT

  PSET (x#(1, 0), x#(2, 0))
  'PSET (x#(1, 0) - x#(4, 0), x#(2, 0) - x#(5, 0))
  ON KEY(2) GOSUB Ende

GOTO EULER
```

```
'*****************************************************************************
RK4:
'*****************************************************************************
 KEY(2) ON
 GOSUB force
 t# = t# + h#

 FOR i = 1 TO 6
  k#(1, i) = h# * f#(i, 1)
  d#(i) = x#(i, 0)
  x#(i, 0) = d#(i) + k#(1, i) / 2#
 NEXT

 GOSUB force

 FOR i = 1 TO 6
  k#(2, i) = h# * f#(i, 1)
  x#(i, 0) = d#(i) + k#(2, i) / 2#
 NEXT

 GOSUB force

 FOR i = 1 TO 6
  k#(3, i) = h# * f#(i, 1)
  x#(i, 0) = d#(i) + k#(3, i)
 NEXT

 GOSUB force

 FOR i = 1 TO 6
  k#(4, i) = h# * f#(i, 1)
  x#(i, 1) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
  x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
 NEXT

'Verschiedene Energieterme:
'-------------------------
 GOSUB force
  'Energie# = x#(4, 0) ^ 2# / 2# + x#(6, 0) ^ 2# / 2# + e# / x#(1, 0) + e# / (1# - x#(1, 0)) + e# / x#(3, 0) + e# /
(.2# - f1# - x#(3, 0))
  'Energie# = x#(3, 0) ^ 2# / 2# + x#(4, 0) ^ 2# / 2#

'Poincare-Schnitt:
'-----------------
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

'Graphikanweisungen:
'-------------------
  'PSET (t#, Energie#)
  PSET (x#(4, 0), x#(6, 0))

  ON KEY(2) GOSUB Ende

GOTO RK4

'*****************************************************************************
RK4p:
'*****************************************************************************
```

```
KEY(2) ON
t# = t# + h#

 FOR j = 1 TO 2
  GOSUB force

  FOR i = 1 TO 6 STEP 1
   k#(1, i) = h# * f#(i, 1)
   d#(i) = x#(i, 0)
   x#(i, 0) = d#(i) + k#(1, i) / 2#
  NEXT

  GOSUB force

  FOR i = 1 TO 6 STEP 1
   k#(2, i) = h# * f#(i, 1)
   x#(i, 0) = d#(i) + k#(2, i) / 2#
  NEXT

  GOSUB force

  FOR i = 1 TO 6 STEP 1
   k#(3, i) = h# * f#(i, 1)
   x#(i, 0) = d#(i) + k#(3, i)
  NEXT

  GOSUB force

  FOR i = 1 TO 6 STEP 1
   k#(4, i) = h# * f#(i, 1)
   y#(i, j) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
  NEXT

  h# = -h#

  IF j = 1 THEN
   FOR i = 1 TO 6 STEP 1
    x#(i, 0) = d#(i)
   NEXT
  END IF

 NEXT j

 FOR i = 1 TO 6 STEP 1
  x#(i, 1) = y#(i, 1) + y#(i, 2) - x#(i, -1)
 NEXT

'Zur Berechnung der Energie und Anweisungen für die Graphik:
'-----------------------------------------------------
 'Energie# = x#(4, 0) ^ 2# / 2# + x#(6, 0) ^ 2# / 2# + e# / x#(1, 0) + e# / (1# - x#(1, 0)) + e# / x#(3, 0) + e# /
(.2# - f1# - x#(3, 0))
 'PSET (t#, Energie#)
 'PSET (x#(4, 1), x#(6, 1))
 PSET (t#, y#(1, 2) - x#(1, -1))

 FOR i = 1 TO 6 STEP 1
  x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
 NEXT
```

```
 GOTO r
 'Zeichnen eines lokalen Richtungsfeldes:
 '-----------------------------------
  z1# = x#(1, -1): z2# = x#(1, -1): t1# = t#: t2# = t#
  FOR i = 0 TO 92 STEP 1
   z1# = z1# + h# / 100# * z1# * e#
   z2# = z2# - h# / 100# * z2# * e#
   t1# = t1# + h# / 100#
   t2# = t2# - h# / 100#
   IF i > 8 THEN
   PSET (t1#, z1#)
   PSET (t2#, z2#)
   END IF
  NEXT i
r:
'Poincare-Schnitt:
'-----------------
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

  IF t# >= 8 THEN GOSUB Ende1
  ON KEY(2) GOSUB Ende1
  IF stopp = 1 THEN GOTO Ende

GOTO RK4p

'*****************************************************************************
Midpoint:
'*****************************************************************************

t# = t# + h#
GOSUB force
KEY(2) ON

GOTO 1000   'Stillegen, falls kompakt gerechnet werden soll!

'Der kompakte Algorithmus:
'------------------------
 FOR i = 1 TO 4
   x#(i, 1) = 2# * h# * f#(i, 1) + x#(i, -1)
 NEXT

 GOTO 2000

1000 'Der Algorithmus mit Zwischentermen:
'---------------------------------------
 FOR i = 1 TO 6
   y#(i, 1) = x#(i, 0) + h# * f#(i, 1)
   y#(i, 2) = x#(i, 0) - h# * f#(i, 1)
   x#(i, 1) = y#(i, 1) - y#(i, 2) + x#(i, -1)
 NEXT

2000
'Energieberechnung und Anweisung für die Graphik:
'------------------------------------------------
  'Energie# = x#(4, 1) ^ 2# / 2# + x#(6, 1) ^ 2# / 2# + e# / x#(1, 1) + e# / (1# - x#(1, 1)) + e# / x#(3, 1) + e# /
(.2# - f1# - x#(3, 1))
  'PSET (x#(3, 1), x#(4, 1))
```

```
  'PSET (x#(1, 1), x#(2, 1))
  'PSET (t#, y#(1, 2) - x#(1, -1))
  PSET (t#, Energie#)

'Poincare-Schnitt:
'----------------
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs


  FOR i = 1 TO 6
    x#(i, -1) = x#(i, 0): x#(i, 0) = x#(i, 1)
  NEXT

  IF t# > 5 THEN GOSUB Ende1
  ON KEY(2) GOSUB Ende1
  IF stopp = 1 THEN GOSUB Ende

GOTO Midpoint

'*************************************************************************
Taylor:
'*************************************************************************

  KEY(2) ON
  ON KEY(2) GOSUB Ende
  GOSUB force

  FOR i = 1 TO 6
    x#(i, 1) = x#(i, 0) + h# * f#(i, 1) + (h# ^ 2# / 2#) * f#(i, 2)
    x#(i, 0) = x#(i, 1)
  NEXT i

  PSET (x#(1, 0), x#(1, 2))
GOTO Taylor

'************************************************************************
RK4n:
'************************************************************************

KEY(2) ON
t# = t# + h#

  FOR j = 1 TO 2
    GOSUB force

    FOR i = 1 TO 4
      k#(1, i) = h# * f#(i, 1)
      d#(i) = x#(i, 0)
      x#(i, 0) = d#(i) + k#(1, i) / 2#
    NEXT

    GOSUB force

    FOR i = 1 TO 4
      k#(2, i) = h# * f#(i, 1)
      x#(i, 0) = d#(i) + k#(2, i) / 2#
    NEXT
```

```
    GOSUB force

    FOR i = 1 TO 4
      k#(3, i) = h# * f#(i, 1)
      x#(i, 0) = d#(i) + k#(3, i)
    NEXT

    GOSUB force

    FOR i = 1 TO 4
      k#(4, i) = h# * f#(i, 1)
      y#(i, j) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
    NEXT

    h# = -h#

    IF j = 1 THEN

      FOR i = 1 TO 4
        x#(i, 0) = d#(i)
      NEXT

    END IF

  NEXT j

    FOR i = 1 TO 4
      x#(i, 1) = y#(i, 1) - y#(i, 2) + x#(i, -1)
    NEXT

    FOR i = 1 TO 4
      x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
    NEXT

GOTO k
'Richtungsfeld zeichnen:
'----------------------
  z1# = x#(1, -1): z2# = x#(1, -1): t1# = t#: t2# = t#

  FOR i = 0 TO 92 STEP 1
    z1# = z1# + h# / 100# * z1# * e#
    z2# = z2# - h# / 100# * z2# * e#
    t1# = t1# + h# / 100#
    t2# = t2# - h# / 100#
    IF i > 8 THEN
    PSET (t1#, z1#)
    PSET (t2#, z2#)
    END IF
  NEXT i

k:
'Poincare-Schnitt:
'-----------------
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

'Graphik:
'--------
```

```
IF t# >= 7 THEN GOSUB Ende1
ON KEY(2) GOSUB Ende1
IF stopp = 1 THEN GOSUB Ende

GOTO RK4n

'*****************************************************************************
force:    'Hier müssen die ODE's eingegeben werden:
'*****************************************************************************

'Zu welchem System soll gesprungen werden?
'---------------------------------------
 GOTO 7

'Die in den ODE's auftretenden Kräfte sowie deren Ableitungen:
'-----------------------------------------------------

1  'Die folgende Zeile beinhaltet Kräfte für einen Sinus zum Testen der Algorithmen

 f#(1, 1) = x#(2, 0): f#(2, 1) = -x#(1, 0) - x#(2, 0)
 'f#(1, 2) = f#(2, 1)': f#(2, 2) = -f#(1, 1)
 'f#(1, 1) = x#(2, 0)
 'f#(2, 1) = -x#(1, 0)
 'f#(3, 1) = (-.1#) * x#(3, 0) '+ .005# * x#(1, 0)
 f#(1, 2) = f#(2, 1)
 f#(2, 2) = -f#(1, 1) - f#(2, 1)
 'f#(3, 2) = .0001# * x#(3, 0) + .05# * f#(1, 1)
 GOTO 15

2  'Reversibel konstruierte logistische DGL:

 f#(1, 1) = x#(1, 0) - x#(1, 0) ^ 2#
 f#(1, 2) = -f#(1, 1) * (2# * x#(1, 0) - 1#)
 GOTO 15

3  'Coulombanziehung für zwei Teilchen:

 q# = -e# * ((x#(1, 0) - x#(4, 0)) ^ 2# + (x#(2, 0) - x#(5, 0)) ^ 2# + (x#(3, 0) - x#(6, 0)) ^ 2#) ^ (-3# / 2#)
 f#(1, 1) = x#(7, 0)
 f#(2, 1) = x#(8, 0)
 f#(3, 1) = x#(9, 0)
 f#(4, 1) = x#(10, 0)
 f#(5, 1) = x#(11, 0)
 f#(6, 1) = x#(12, 0)

 f#(7, 1) = q# * (x#(1, 0) - x#(4, 0)) '- .02 * x#(7, 0) ^ 2
 f#(8, 1) = q# * (x#(2, 0) - x#(5, 0)) '- .02 * x#(8, 0) ^ 2
 f#(9, 1) = q# * (x#(3, 0) - x#(6, 0)) '- .02 * x#(9, 0) ^ 2
 f#(10, 1) = -.0001 * f#(7, 1)
 f#(11, 1) = -.0001 * f#(8, 1)
 f#(12, 1) = -.0001 * f#(9, 1)

 f#(1, 2) = f#(7, 1)
 f#(2, 2) = f#(8, 1)
 f#(3, 2) = f#(9, 1)
 f#(4, 2) = f#(10, 1)
 f#(5, 2) = f#(11, 1)
 f#(6, 2) = f#(12, 1)
 GOTO 15
```

```
4 'Exponentialfunktion:

  f#(1, 1) = e# * x#(1, 0)
  f#(1, 2) = e# ^ 2# * x#(1, 0)
  GOTO 15

6 'Diamagnetisches H-Atom:

  f#(1, 1) = x#(3, 0): f#(2, 1) = x#(4, 0)
  f#(3, 1) = 2# * x#(1, 0) * e# - 2# * x#(2, 0) ^ 2# * x#(1, 0) ^ 3# - x#(2, 0) ^ 2# * x#(1, 0)
  f#(4, 1) = 2# * x#(2, 0) * e# - 2# * x#(1, 0) ^ 2# * x#(2, 0) ^ 3# - x#(1, 0) ^ 2# * x#(2, 0)
  f#(1, 2) = f#(3, 1): f#(2, 2) = f#(4, 1)
  f#(3, 2) = 2# * f#(1, 1) * e# - 4# * x#(2, 0) * f#(2, 1) * x#(1, 0) ^ 3# - 6# * x#(2, 0) ^ 2# * f#(1, 1) * x#(1, 0) ^
2# - 4# * x#(2, 0) ^ 3# * f#(2, 1) * x#(1, 0) - x#(2, 0) ^ 4# * f#(1, 1)
  f#(4, 2) = 2# * f#(2, 1) * e# - 4# * x#(1, 0) * f#(1, 1) * x#(2, 0) ^ 3# - 6# * x#(1, 0) ^ 2# * f#(2, 1) * x#(2, 0) ^
2# - 4# * x#(1, 0) ^ 3# * f#(1, 1) * x#(2, 0) - x#(1, 0) ^ 4# * f#(2, 1)
  GOTO 15

7 'Chaotische Viel-Teilchen-Gleichung nach Roessler

  f1# = SQR((x#(1, 0) - .45#) ^ 2# + 5# * 10# ^ -6#) + SQR((x#(1, 0) - .55#) ^ 2# + 5# * 10# ^ -6#) - 2# *
SQR((x#(1, 0) - .5#) ^ 2# + 10# ^ -4#)
  g1# = (x#(1, 0) - .45#) / SQR((x#(1, 0) - .45#) ^ 2# + 5# * 10# ^ -6#) + (x#(1, 0) - .55#) / SQR((x#(1, 0) -
.55#) ^ 2# + 5# * 10# ^ -6#) - 2# * (x#(1, 0) - .5#) / SQR((x#(1, 0) - .5#) ^ 2# + 10# ^ -4#)
  k1# = ((x#(1, 0) - .45#) ^ 2# + 5# * 10# ^ -6#) ^ (-.5#) - (x#(1, 0) - .45#) ^ 2# * ((x#(1, 0) - .45#) ^ 2# + 5# *
10# ^ -6#) ^ (-1.5#) + ((x#(1, 0) - .55#) ^ 2# + 5# * 10# ^ -6#) ^ (-.5#) - (x#(1, 0) - .55#) ^ 2# * ((x#(1, 0) - .55#)
^ 2# + 5# * 10# ^ -6#) ^ (-1.5#) - 2# * ((x#(1, 0) - .5#) ^ 2# + 10# ^ -4#) ^ (-.5#) + 2# * (x#(1, 0) - .5#) ^ 2# *
((x#(1, 0) - .5#) ^ 2# + 10# ^ -4#) ^ (-1.5#)
  'f2# = SQR((x#(2, 0) - .45#) ^ 2# + 5# * 10# ^ -6#) + SQR((x#(2, 0) - .55#) ^ 2# + 5# * 10# ^ -6#) - 2# *
SQR((x#(2, 0) - .5#) ^ 2# + 10# ^ -4#)
  'g2# = (x#(2, 0) - .45#) / SQR((x#(2, 0) - .45#) ^ 2# + 5# * 10# ^ -6#) + (x#(2, 0) - .55#) / SQR((x#(2, 0) -
.55#) ^ 2# + 5# * 10# ^ -6#) - 2# * (x#(2, 0) - .5#) / SQR((x#(2, 0) - .5#) ^ 2# + 10# ^ -4#)
  'k2# = ((x#(2, 0) - .45#) ^ 2# + 5# * 10# ^ -6#) ^ (-.5#) - (x#(2, 0) - .45#) ^ 2# * ((x#(2, 0) - .45#) ^ 2# + 5# *
10# ^ -6#) ^ (-1.5#) + ((x#(2, 0) - .55#) ^ 2# + 5# * 10# ^ -6#) ^ (-.5#) - (x#(2, 0) - .55#) ^ 2# * ((x#(2, 0) - .55#)
^ 2# + 5# * 10# ^ -6#) ^ (-1.5#) - 2# * ((x#(2, 0) - .5#) ^ 2# + 10# ^ -4#) ^ (-.5#) + 2# * (x#(2, 0) - .5#) ^ 2# *
((x#(2, 0) - .5#) ^ 2# + 10# ^ -4#) ^ (-1.5#)
  f#(1, 1) = x#(4, 0)
  'f#(2, 1) = x#(5, 0)
  f#(3, 1) = x#(6, 0)

  f#(4, 1) = e# / x#(1, 0) ^ 2# - e# / (1# - x#(1, 0)) ^ 2# - (e# / (.2# - f1# - x#(3, 0)) ^ 2#) * g1#
  'f#(5, 1) = e# / x#(2, 0) ^ 2# - e# / (1# - x#(2, 0)) ^ 2# - (e# / (.2# - f2# - x#(3, 0)) ^ 2#) * g2#
  f#(6, 1) = e# / x#(3, 0) ^ 2# - e# / (.2# - f1# - x#(3, 0)) ^ 2# '- e# / (.2# - f2# - x#(3, 0)) ^ 2#
  GOTO 15

  f#(1, 2) = f#(4, 1)
  'f#(2, 2) = f#(5, 1)
  f#(3, 2) = f#(6, 1)
  'f#(4, 2) = -2# * e# * f#(1, 1) / x#(1, 0) ^ 3# - 2# * e# * f#(1, 1) / (1# - x#(1, 0)) ^ 3# - 2# * e# * f#(1, 1) * g1# ^
2# / (.2# - f1# - x#(3, 0)) ^ 3# - e# * f#(1, 1) * k1# / (.2# - f1# - x#(3, 0)) ^ 2# - 2# * e# * f#(3, 1) * g1# / (.2# -
f1# - x#(3, 0)) ^ 3#
  'f#(5, 2) = -2# * e# * f#(2, 1) / x#(2, 0) ^ 3# - 2# * e# * f#(2, 1) / (1# - x#(2, 0)) ^ 3# - 2# * e# * f#(2, 1) * g2# ^
2# / (.2# - f2# - x#(3, 0)) ^ 3# - e# * f#(2, 1) * k2# / (.2# - f2# - x#(3, 0)) ^ 2# - 2# * e# * f#(3, 1) * g2# / (.2# -
f2# - x#(3, 0)) ^ 3#
  'f#(6, 2) = -2# * e# * f#(3, 1) / x#(3, 0) ^ 3# - 2# * e# * f#(3, 1) / (.2# - f1# - x#(3, 0)) ^ 3# - 2# * e# * g1# *
f#(1, 1) / (.2# - f1# - x#(3, 0)) ^ 3#  '- 2# * e# * g2# * f#(2, 1) / (.2# - f2# - x#(3, 0)) ^ 3#- 2# * e# * f#(3, 1) / (.2#
- f2# - x#(3, 0)) ^ 3#
  GOTO 15
```

```
8  'Sattelpunkt:

  f#(1, 1) = x#(2, 0): f#(2, 1) = x#(1, 0): f#(1, 2) = f#(2, 1): f#(2, 2) = f#(1, 1)
  GOTO 15

9  'Logistische ODE:

  f#(1, 1) = x#(1, 0) - x#(1, 0) ^ 2#
  f#(1, 2) = f#(1, 1) - 2# * x#(1, 0) * f#(1, 1)
  GOTO 15

11  'Lorenz-System:

  f#(1, 1) = s# * (x#(2, 0) - x#(1, 0))
  f#(2, 1) = x#(1, 0) * (r# - x#(3, 0)) - x#(2, 0)
  f#(3, 1) = x#(1, 0) * x#(2, 0) - b# * x#(3, 0)
  f#(1, 2) = s# * (f#(2, 1) - f#(1, 1))
  f#(2, 2) = f#(1, 1) * (r# - x#(3, 0)) - f#(2, 1) - x#(1, 0) * f#(3, 0)
  f#(3, 2) = f#(1, 1) * x#(2, 0) + x#(1, 0) * f#(2, 1) - b# * f#(3, 1)
  GOTO 15

12  'Roessler-System:

  f#(1, 1) = -x#(2, 0) - x#(3, 0)
  f#(2, 1) = x#(1, 0)
  f#(3, 1) = a# * (x#(2, 0) - x#(2, 0) ^ 2#) - b# * x#(3, 0)
  GOTO 15

13  'van der Pol:

  f#(1, 1) = sig# * x#(2, 0)
  f#(2, 1) = sig# * (-x#(1, 0) + e# * (1# - x#(1, 0) ^ 2#) * x#(2, 0))
  f#(1, 2) = sig# * f#(2, 1)
  f#(2, 2) = sig# * (-f#(1, 1) + f#(2, 1) * (1# - x#(1, 0) ^ 2#) - x#(2, 0) * 2# * x#(1, 0) * f#(1, 1))
  GOTO 15

14  'Elektron im Magnetfeld:

  f#(1, 1) = x#(3, 0)
  f#(2, 1) = x#(4, 0)
  f#(3, 1) = x#(4, 0)
  f#(4, 1) = -x#(3, 0)
  f#(1, 2) = f#(3, 1)
  f#(2, 2) = f#(4, 1)
  f#(3, 2) = f#(4, 1)
  f#(4, 2) = -f#(3, 1)

15 RETURN

'*************************************************************************
Bildschirm:        'Setup für den Bildschirm
'*************************************************************************

SCREEN 12
VIEW (50, 40)-(620, 410), , 16
WINDOW (xmin, ymin)-(xmax, ymax)

  FOR i = 1 TO 100
```

```
    LINE (xmin + i * xAbschnitt, ymax)-(xmin + i * xAbschnitt, -ABS(ymin - ymax) / 100 + ymax)
    LINE (xmin, ymin + i * yAbschnitt)-(xmin + ABS(xmin - xmax) / 100, ymin + i * yAbschnitt)
    LINE (xmin + i * xAbschnitt, ymin)-(xmin + i * xAbschnitt, ABS(ymin - ymax) / 100 + ymin)
    LINE (xmax, ymin + i * yAbschnitt)-(xmax - ABS(xmin - xmax) / 100, ymin + i * yAbschnitt)
  NEXT

PRINT
PRINT TAB(7); xmin; TAB(35); xBeschriftung$; TAB(75); xmax
PRINT ; TAB(2); ymax

  FOR i = 1 TO 9
    PRINT
  NEXT

  FOR i = 1 TO 5
    PRINT TAB(2); yBeschriftung$(i)
  NEXT

  FOR i = 1 TO 8
    PRINT
  NEXT

PRINT TAB(2); ymin

RETURN


Ende:
PRINT : PRINT "Integrationszeit t ="; t#, "Zeitschritt h ="; h#
KILL "varint1.dat"


OPEN "varint1.dat" FOR BINARY AS #1

  PUT #1, , h#
  PUT #1, , e#
  PUT #1, , t#

  FOR j = -1 TO 1 STEP 1
    FOR i = 1 TO 12
      PUT #1, , x#(i, j)
    NEXT i
  NEXT j

  FOR i = 1 TO 13
    PUT #1, , Anfangswert#(i)
  NEXT

  FOR j = 1 TO 6
    FOR i = 1 TO 12
      PUT #1, , f#(i, j)
    NEXT i
  NEXT j

  PUT #1, , xmax
  PUT #1, , xmin
  PUT #1, , ymax
  PUT #1, , ymin
  PUT #1, , xAbschnitt
```

```
  PUT #1, , yAbschnitt

 CLOSE #1

 KILL "varint2.dat"
 OPEN "varint2.dat" FOR OUTPUT AS #2
  PRINT #2, xBeschriftung$
  FOR i = 1 TO 5
   PRINT #2, yBeschriftung$(i)
  NEXT
 CLOSE #2

 END
 RETURN

 pcs:
 IF ABS(f#(1, 1)) > .0000001# THEN h1# = -x#(1, -1) / f#(1, 1) ELSE h1# = 0#
 ' h1# = (-2# * f#(1, 1) + SQR(4# * f#(1, 1) ^ 2# - 8# * f(1, 2) * x#(1, -1))) / (2# * f#(1, 2)) ELSE h1# = 0#
 x# = x#(2, -1) + h1# * f#(2, 1) '+ (h1# ^ 2# / 2#) * f#(2, 2)
 p# = x#(4, -1) + h1# * f#(4, 1) '+ (h1# ^ 2# / 2#) * f#(4, 2)
 'p#(2) = (x#(2, -1) - x#(2, -1)) / (2# * h#)
 PSET (x#, p#)
 RETURN

 Zwischenalgorithmus: 'RK4
 FOR j = 1 TO 2
  FOR i = 1 TO 4
  GOSUB force
  k#(1, i) = h# * f#(i, 1)
  d#(i) = x#(i, 0)
  x#(i, 0) = d#(i) + k#(1, i) / 2#
  NEXT

  GOSUB force
  FOR i = 1 TO 4
  k#(2, i) = h# * f#(i, 1)
  x#(i, 0) = d#(i) + k#(2, i) / 2#
  NEXT

  GOSUB force
  FOR i = 1 TO 4
  k#(3, i) = h# * f#(i, 1)
  x#(i, 0) = d#(i) + k#(3, i)
  NEXT

  GOSUB force
  FOR i = 1 TO 4
  k#(4, i) = h# * f#(i, 1)
  y#(i, j) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
  x#(i, 0) = d#(i)
  NEXT

 h# = -h#
 NEXT j
 RETURN

 Ende1:
 stopp = 1
 RETURN
```

H-ATOM.BAS:

```
CLS
stopp = 0
count = 0
Zeit# = 10000#
PRINT "H-Atom ist ein Programm zur Integration des diamagnatischen H-Atoms, wobei der"
PRINT "Algorithmus aus untenstehender Auswahl beliebig gewaehlt werden kann."
PRINT "Zur Verfuegung stehen:"
PRINT
PRINT "RK4=1,  RK4(rev+)=2,  RK4(rev-)= 3,  VERLET1=4,  EULER=5,"
PRINT "Midpoint=6,  VERLET2=7,  TAYLOR(2.Ordnung)=8"
PRINT
PRINT "VERLET1 berechnet jede Variable mit x(n+1) = 2*x(n) + h^2 * f(x(n)) - x(n-1),"
PRINT "wogegen VERLET2 die Impulse approximativ mit p(n) = (x(n+1) - x(n-1)) / (2*h)"
PRINT "berechnet."
PRINT
INPUT "gewaehlter Algorithmus = "; alg

'*****************************************************************************
'Dimensionierung der Variablen:
'*****************************************************************************

DIM Anfangswert#(5)  'Die Anfangswerte.
DIM f#(4, 6)         'Die Kraefte. Es bedeutet: erste 6 = Anzahl der ODE's
                     'zweite 6 = Grad der auftretenden Ableitungen
DIM x#(4, -1 TO 1)   'Die Variablen.Die zweite dim-Zahl gibt die
                     '3 aufeinanderfolgenden Iterationswerte an
DIM k#(4, 12), d#(12)  'Parameter fuer RK4
DIM y#(4, 2)         'Zwischenterme fuer RK4p; VERLET1 und EULERrev
DIM p#(4)            'Die approximativen Impulse in VERLET2
DIM yBeschriftung$(5)

'*****************************************************************************
'Die alten Daten fuer Parameter und Bildschirmdefinition
'aus den Dateien H-Atom1.dat, H-Atom2.dat:
'*****************************************************************************
OPEN "H-Atom1.dat" FOR BINARY AS #1

  GET #1, , h#
  GET #1, , e#
  GET #1, , t#

  FOR j = -1 TO 1 STEP 1
    FOR i = 1 TO 4
      GET #1, , x#(i, j)
    NEXT i
  NEXT j
```

```
    FOR i = 1 TO 5
     GET #1, , Anfangswert#(i)
    NEXT

    FOR j = 1 TO 6
     FOR i = 1 TO 4
      GET #1, , f#(i, j)
     NEXT i
    NEXT j

    GET #1, , xmax
    GET #1, , xmin
    GET #1, , ymax
    GET #1, , ymin
    GET #1, , xAbschnitt
    GET #1, , yAbschnitt

   CLOSE #1

   PRINT "Die alten Daten lauten:"
   PRINT "Zeitschritt h="; h#, "Parameter e="; e#

'*************************************************************************
'Abfragen, ob Aenderungen vorgenommen werden sollen:
'*************************************************************************

    INPUT "Soll der Timestep h geaendert werden?   ja=1   nein=0  "; Aenderung1
    IF Aenderung1 = 0 THEN GOTO 10
    INPUT "h=", h#                  '******  Zeitschritt *****

10 INPUT "Soll der Parameter e geaendert werden?  ja=1    nein=0", Aenderung2
   IF Aenderung2 = 0 THEN GOTO 30
   INPUT "e=", e#      '*****  Wird fuer chaotische Glch. nach Roessler und H-Atom gebraucht

30 INPUT "Soll ein Neustart vorgenommen oder mit den alten Daten weitergerechnet werden?   Neu=1   weiter=0
", neu
   IF neu = 0 THEN GOTO 70

35 PRINT " Die bisherigen Anfangswerte lauten:"

   FOR i = 1 TO 5
    PRINT "x"; SPC(0); i; SPC(0); ",0 = "; Anfangswert#(i)
   NEXT

   PRINT "x5,0 = Anfangszeit"
   INPUT "Sollen die Anfangswerte geaendert werden?   ja=1    nein=0", Aenderung5
   IF Aenderung5 = 0 THEN GOTO 60

   FOR i = 1 TO 5
    PRINT "x"; i; "0 = "; : INPUT Anfangswert#(i)
   NEXT
60
DO
  Anfangswert#(4) = Anfangswert#(4) + .1#
  Diskriminante# = 4# - Anfangswert#(4) ^ 2# + 2# * e# * Anfangswert#(2) ^ 2#
  IF Diskriminante# > 0 THEN GOSUB Anfangswerte
LOOP WHILE Anfangswert#(4) <= 1.3
```

```
  Anfangswert#(4) = 1.9#
  Zeit# = 150#

DO
  Anfangswert#(4) = Anfangswert#(4) + .02#
  Diskriminante# = 4# - Anfangswert#(4) ^ 2# + 2# * e# * Anfangswert#(2) ^ 2#
  IF Diskriminante# > 0 THEN GOSUB Anfangswerte
LOOP WHILE Anfangswert#(4) <= 1.99

  Anfangswert#(4) = -.5#
  Zeit# = 100#

DO
  Anfangswert#(4) = Anfangswert#(4) - .1#
  Diskriminante# = 4# - Anfangswert#(4) ^ 2# + 2# * e# * Anfangswert#(2) ^ 2#
  IF Diskriminante# > 0 THEN GOSUB Anfangswerte
LOOP WHILE Anfangswert#(4) >= -1.4

GOTO Ende
'*************************************************************************
Anfangswerte: 'Bestimmung des ersten Wertepaars:
'*************************************************************

  t# = Anfangswert#(5)
  Anfangswert#(3) = SQR(Diskriminante#)
  FOR i = 1 TO 4
    x#(i, 0) = Anfangswert#(i)
  NEXT

  GOSUB force
  FOR i = 1 TO 4
    k#(1, i) = h# * f#(i, 1)
    d#(i) = x#(i, 0)
    x#(i, 0) = d#(i) + k#(1, i) / 2#
  NEXT

  GOSUB force
  FOR i = 1 TO 4
    k#(2, i) = h# * f#(i, 1)
    x#(i, 0) = d#(i) + k#(2, i) / 2#
  NEXT

  GOSUB force
  FOR i = 1 TO 4
    k#(3, i) = h# * f#(i, 1)
    x#(i, 0) = d#(i) + k#(3, i)
  NEXT

  GOSUB force
  FOR i = 1 TO 4
    k#(4, i) = h# * f#(i, 1)
    x#(i, 1) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
    x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
  NEXT

  t# = t# + h#
IF count = 1 THEN GOTO Anweisung
'*************************************************************
'Hier beginnt die Bildschirmdefinition:
```

```
'*****************************************************************

70 PRINT "Die bisherigen Bildschirmdaten:"
  PRINT "xmax="; xmax, "xmin="; xmin, "ymax="; ymax, "ymin="; ymin, "x-Abschnitt="; xAbschnitt, "y-
Abschnitt="; yAbschnitt
  INPUT "Sollen die Achsen geaendert werden?   ja=1   nein=0", Aenderung
  IF Aenderung = 0 THEN GOTO 75
  INPUT "maximaler x-Wert=", xmax
  INPUT "minimaler x-Wert=", xmin
  INPUT "maximaler y-Wert=", ymax
  INPUT "minimaler y-Wert=", ymin
  INPUT "x-Abschnitt=", xAbschnitt
  INPUT "y-Abschnitt=", yAbschnitt

75 OPEN "H-Atom2.dat" FOR INPUT AS #2

   INPUT #2, xBeschriftung$

   FOR j = 1 TO 5
    INPUT #2, yBeschriftung$(j)
   NEXT j

  CLOSE #2

 PRINT
 PRINT "Die bisherige Beschriftung lautet:"
 PRINT "   für die x-Achse:   "; xBeschriftung$
 PRINT "   für die y-Achse:   ";

 FOR i = 1 TO 5
  PRINT yBeschriftung$(i);
 NEXT

 PRINT : PRINT
 INPUT "Soll die Beschriftung geaendert werden?  ja = 1   nein = 0", Aenderung
 IF Aenderung = 0 THEN GOTO 80
 INPUT "Die x-Beschriftung:", xBeschriftung$
 PRINT "Die y-Beschriftung (es stehen 5 Zeilen mit je ca. 5 Buchstaben zur Verfügung):"

 FOR i = 1 TO 5
  INPUT yBeschriftung$(i)
 NEXT

80
count = 1
GOSUB Bildschirm
'*****************************************************************************
Anweisung: 'Anweisungen zu welchem Algorithmus gesprungen wird:
'*****************************************************************************

 IF alg = 1 THEN GOSUB RK4
 IF alg = 2 THEN GOSUB RK4p
 IF alg = 3 THEN GOSUB RK4n
 IF alg = 4 THEN GOSUB VERLET1
 IF alg = 5 THEN GOSUB EULER
 IF alg = 6 THEN GOSUB Midpoint
 IF alg = 7 THEN GOSUB VERLET2
 IF alg = 8 THEN GOSUB Taylor
RETURN
```

```
'*****************************************************************************
VERLET1:
'*****************************************************************************

DO UNTIL t# >= 100
 KEY(2) ON
 GOSUB force

GOTO 200  'Dieser Sprung muß stillgelegt werden, wenn man kompakt rechnen will!!!

'Der Algorithmus kompakt:
'------------------------

 FOR i = 1 TO 4
  x#(i, 1) = 2# * x#(i, 0) + (h# ^ 2#) * f#(i, 2) - x#(i, -1)
 NEXT i

 GOTO 300

200
'Der Algorithmus mit Zwischentermen (fuer den "Gedaechtnisterm"):
'---------------------------------------------------------
 FOR i = 1 TO 4
  y#(i, 1) = x#(i, 0) + h# * f#(i, 1) + (h# ^ 2# / 2#) * f#(i, 2)
  y#(i, 2) = x#(i, 0) - h# * f#(i, 1) + (h# ^ 2# / 2#) * f#(i, 2)
  x#(i, 1) = y#(i, 1) + y#(i, 2) - x#(i, -1)
 NEXT i

300
 t# = t# + h#
'Berechnung der Energie für das System:
'--------------------------------------
 Energie# = x#(4, 1) ^ 2# / 2# + x#(3, 1) ^ 2# / 2# - e# * (x#(1, 1) ^ 2# + x#(2, 1) ^ 2#) + .5# * x#(1, 1) ^ 2# *
x#(2, 1) ^ 2# * (x#(1, 1) ^ 2# + x#(2, 1) ^ 2#)
 PSET (t#, Energie#)

'Poincare-Schnitt:
'-----------------
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

 FOR i = 1 TO 4
  x#(i, -1) = x#(i, 0): x#(i, 0) = x#(i, 1)
 NEXT i

 ON KEY(2) GOSUB Ende1
 IF stopp = 1 THEN GOSUB Ende

LOOP
RETURN

'*****************************************************************************
VERLET2:
'*****************************************************************************

DO UNTIL t# > Zeit#
 KEY(2) ON
 GOSUB force
```

```
  t# = t# + h#

    FOR i = 1 TO 2
      x#(i, 1) = 2# * x#(i, 0) + h# ^ 2# * f#(i, 2) - x#(i, -1)
    NEXT i

'Poincare-Schnitt:
'-----------------
  'IF x#(1, 0) >= 0 AND x#(1, -1) <= 0 THEN GOSUB pcs
  'IF x#(1, 0) >= 0 AND x#(1, -1) <= 0 THEN GOSUB pcs

'Zur Berechnung der approximativen Impulse:
'-----------------------------------------
    FOR i = 1 TO 2
      p#(i) = (x#(i, 1) - x#(i, -1)) / (2# * h#)
      x#(i, -1) = x#(i, 0): x#(i, 0) = x#(i, 1)
    NEXT i

'Die Berechnung der Energie:
'--------------------------
  Energie# = p#(1) ^ 2# / 2# + p#(2) ^ 2# / 2# - e# * (x#(1, -1) ^ 2# + x#(2, -1) ^ 2#) + .5# * x#(1, -1) ^ 2# *
x#(2, -1) ^ 2# * (x#(1, -1) ^ 2# + x#(2, -1) ^ 2#)

'Für die Graphik:
'----------------
  PSET (t#, Energie#)
  'PSET (p#(1), p#(2))
  'PSET (x#(1, 0), p#(1))

  ON KEY(2) GOSUB Ende1
  IF stopp = 1 THEN GOSUB Ende

LOOP
GOTO Ende
RETURN

'*******************************************************************************
EULER:
'*******************************************************************************

  KEY(2) ON
  GOSUB force
  t# = t# + h#

    FOR i = 1 TO 4
      x#(i, 1) = x#(i, 0) + h# * f#(i, 1)
    NEXT

    FOR i = 1 TO 4
      x#(i, 0) = x#(i, 1)
    NEXT

  PSET (x#(1, 0), x#(2, 0))
  'PSET (x#(1, 0) - x#(4, 0), x#(2, 0) - x#(5, 0))
  ON KEY(2) GOSUB Ende

GOTO EULER

'*******************************************************************************
```

```
RK4:
'*****************************************************************************

DO UNTIL t# >= 100
 KEY(2) ON
 GOSUB force
 t# = t# + h#

 FOR i = 1 TO 4
  k#(1, i) = h# * f#(i, 1)
  d#(i) = x#(i, 0)
  x#(i, 0) = d#(i) + k#(1, i) / 2#
 NEXT

 GOSUB force

 FOR i = 1 TO 4
  k#(2, i) = h# * f#(i, 1)
  x#(i, 0) = d#(i) + k#(2, i) / 2#
 NEXT

 GOSUB force

 FOR i = 1 TO 4
  k#(3, i) = h# * f#(i, 1)
  x#(i, 0) = d#(i) + k#(3, i)
 NEXT

 GOSUB force

 FOR i = 1 TO 4
  k#(4, i) = h# * f#(i, 1)
  x#(i, 1) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
  x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
 NEXT

'Energie:
'-----------------------
 Energie# = x#(3, 0) ^ 2# / 2# + x#(4, 0) ^ 2# / 2# - e# * (x#(1, 0) ^ 2# + x#(2, 0) ^ 2#) + .5# * x#(1, 0) ^ 2# *
x#(2, 0) ^ 2# * (x#(1, 0) ^ 2# + x#(2, 0) ^ 2#)

'Poincare-Schnitt:
'----------------
 'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
 'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

'Graphikanweisungen:
'------------------
 PSET (t#, Energie#)
 'PSET (x#(2, 0), x#(4, 0))

 ON KEY(2) GOSUB Ende


LOOP
RETURN


'*****************************************************************************
```

```
RK4p:
'*************************************************************************

DO UNTIL t# >= 100

KEY(2) ON
t# = t# + h#

 FOR j = 1 TO 2
  GOSUB force

  FOR i = 1 TO 4
   k#(1, i) = h# * f#(i, 1)
   d#(i) = x#(i, 0)
   x#(i, 0) = d#(i) + k#(1, i) / 2#
  NEXT

  GOSUB force

  FOR i = 1 TO 4
   k#(2, i) = h# * f#(i, 1)
   x#(i, 0) = d#(i) + k#(2, i) / 2#
  NEXT

  GOSUB force

  FOR i = 1 TO 4
   k#(3, i) = h# * f#(i, 1)
   x#(i, 0) = d#(i) + k#(3, i)
  NEXT

  GOSUB force

  FOR i = 1 TO 4
   k#(4, i) = h# * f#(i, 1)
   y#(i, j) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
  NEXT

  h# = -h#

  IF j = 1 THEN
   FOR i = 1 TO 4
    x#(i, 0) = d#(i)
   NEXT
  END IF

 NEXT j

 FOR i = 1 TO 4
  x#(i, 1) = y#(i, 1) + y#(i, 2) - x#(i, -1)
 NEXT

'Zur Berechnung der Energie und Anweisungen für die Graphik:
'-------------------------------------------------------------
 Energie# = x#(3, 0) ^ 2# / 2# + x#(4, 0) ^ 2# / 2# - e# * (x#(1, 0) ^ 2# + x#(2, 0) ^ 2#) + .5# * x#(1, 0) ^ 2# *
x#(2, 0) ^ 2# * (x#(1, 0) ^ 2# + x#(2, 0) ^ 2#)
 PSET (t#, Energie#)

 FOR i = 1 TO 4
```

```
  x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
 NEXT

'Poincare-Schnitt:
'-----------------
 'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
 'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

 ON KEY(2) GOSUB Ende1
 IF stopp = 1 THEN GOTO Ende

LOOP
RETURN

'*****************************************************************************
Midpoint:
'*****************************************************************************

DO WHILE t# <= Zeit#
t# = t# + h#
GOSUB force
KEY(2) ON

GOTO 1000  'Stillegen, falls kompakt gerechnet werden soll!

'Der kompakte Algorithmus:
'------------------------
 FOR i = 1 TO 4
  x#(i, 1) = 2# * h# * f#(i, 1) + x#(i, -1)
 NEXT

 GOTO 2000

1000 'Der Algorithmus mit Zwischentermen:
'---------------------------------------
 FOR i = 1 TO 6
  y#(i, 1) = x#(i, 0) + h# * f#(i, 1)
  y#(i, 2) = x#(i, 0) - h# * f#(i, 1)
  x#(i, 1) = y#(i, 1) - y#(i, 2) + x#(i, -1)
 NEXT

2000
'Energieberechnung und Anweisung für die Graphik:
'----------------------------------------------
 'Energie# = x#(4, 1) ^ 2# / 2# + x#(6, 1) ^ 2# / 2# + e# / x#(1, 1) + e# / (1# - x#(1, 1)) + e# / x#(3, 1) + e# /
(.2# - f1# - x#(3, 1))
 'PSET (x#(3, 1), x#(4, 1))
 'PSET (x#(1, 1), x#(2, 1))
 'PSET (t#, y#(1, 2) - x#(1, -1))
 PSET (t#, Energie#)

'Poincare-Schnitt:
'-----------------
 'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
 'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs


 FOR i = 1 TO 6
  x#(i, -1) = x#(i, 0): x#(i, 0) = x#(i, 1)
```

```
  NEXT

  IF t# > 5 THEN GOSUB Ende1
  ON KEY(2) GOSUB Ende1
  IF stopp = 1 THEN GOSUB Ende

LOOP
RETURN

'*****************************************************************************
Taylor:
'*****************************************************************************

DO UNTIL t# >= 100
  KEY(2) ON
  ON KEY(2) GOSUB Ende
  t# = t# + h#
  GOSUB force

  FOR i = 1 TO 4
    x#(i, 1) = x#(i, 0) + h# * f#(i, 1) + (h# ^ 2# / 2#) * f#(i, 2)
    x#(i, 0) = x#(i, 1)
  NEXT i

'Energie:
'-----------------------
  Energie# = x#(3, 0) ^ 2# / 2# + x#(4, 0) ^ 2# / 2# - e# * (x#(1, 0) ^ 2# + x#(2, 0) ^ 2#) + .5# * x#(1, 0) ^ 2# *
x#(2, 0) ^ 2# * (x#(1, 0) ^ 2# + x#(2, 0) ^ 2#)
  PSET (t#, Energie#)

LOOP
RETURN

'*****************************************************************************
RK4n:
'*****************************************************************************

KEY(2) ON
t# = t# + h#

  FOR j = 1 TO 2
    GOSUB force

    FOR i = 1 TO 4
      k#(1, i) = h# * f#(i, 1)
      d#(i) = x#(i, 0)
      x#(i, 0) = d#(i) + k#(1, i) / 2#
    NEXT

    GOSUB force

    FOR i = 1 TO 4
      k#(2, i) = h# * f#(i, 1)
      x#(i, 0) = d#(i) + k#(2, i) / 2#
    NEXT

    GOSUB force

    FOR i = 1 TO 4
```

```
    k#(3, i) = h# * f#(i, 1)
    x#(i, 0) = d#(i) + k#(3, i)
   NEXT

   GOSUB force

   FOR i = 1 TO 4
    k#(4, i) = h# * f#(i, 1)
    y#(i, j) = d#(i) + (k#(1, i) + k#(4, i)) / 6# + (k#(2, i) + k#(3, i)) / 3#
   NEXT

   h# = -h#

   IF j = 1 THEN

    FOR i = 1 TO 4
     x#(i, 0) = d#(i)
    NEXT

   END IF

  NEXT j

   FOR i = 1 TO 4
    x#(i, 1) = y#(i, 1) - y#(i, 2) + x#(i, -1)
   NEXT

   FOR i = 1 TO 4
    x#(i, -1) = d#(i): x#(i, 0) = x#(i, 1)
   NEXT

GOTO k
'Richtungsfeld zeichnen:
'-----------------------
 z1# = x#(1, -1): z2# = x#(1, -1): t1# = t#: t2# = t#

 FOR i = 0 TO 92 STEP 1
  z1# = z1# + h# / 100# * z1# * e#
  z2# = z2# - h# / 100# * z2# * e#
  t1# = t1# + h# / 100#
  t2# = t2# - h# / 100#
  IF i > 8 THEN
  PSET (t1#, z1#)
  PSET (t2#, z2#)
  END IF
 NEXT i

k:
'Poincare-Schnitt:
'-----------------
  'IF x#(1, -1) >= 0 AND x#(1, 0) <= 0 THEN GOSUB pcs
  'IF x#(1, -1) <= 0 AND x#(1, 0) >= 0 THEN GOSUB pcs

'Graphik:
'--------


 IF t# >= 7 THEN GOSUB Ende1
 ON KEY(2) GOSUB Ende1
```

```
  IF stopp = 1 THEN GOSUB Ende

 GOTO RK4n

'*****************************************************************************
force:     'Die ODE:
'*****************************************************************************

  f#(1, 1) = x#(3, 0): f#(2, 1) = x#(4, 0)
  f#(3, 1) = 2# * x#(1, 0) * e# - 2# * x#(2, 0) ^ 2# * x#(1, 0) ^ 3# - x#(2, 0) ^ 4# * x#(1, 0)
  f#(4, 1) = 2# * x#(2, 0) * e# - 2# * x#(1, 0) ^ 2# * x#(2, 0) ^ 3# - x#(1, 0) ^ 4# * x#(2, 0)
  f#(1, 2) = f#(3, 1): f#(2, 2) = f#(4, 1)
  'f#(3, 2) = 2# * f#(1, 1) * e# - 4# * x#(2, 0) * f#(2, 1) * x#(1, 0) ^ 3# - 6# * x#(2, 0) ^ 2# * f#(1, 1) * x#(1, 0) ^
2# - 4# * x#(2, 0) ^ 3# * f#(2, 1) * x#(1, 0) - x#(2, 0) ^ 4# * f#(1, 1)
  'f#(4, 2) = 2# * f#(2, 1) * e# - 4# * x#(1, 0) * f#(1, 1) * x#(2, 0) ^ 3# - 6# * x#(1, 0) ^ 2# * f#(2, 1) * x#(2, 0) ^
2# - 4# * x#(1, 0) ^ 3# * f#(1, 1) * x#(2, 0) - x#(1, 0) ^ 4# * f#(2, 1)

 RETURN

'*****************************************************************************
Bildschirm:        'Setup für den Bildschirm
'*****************************************************************************

 SCREEN 12
 VIEW (50, 40)-(620, 410), , 16
 WINDOW (xmin, ymin)-(xmax, ymax)

  FOR i = 1 TO 100
    LINE (xmin + i * xAbschnitt, ymax)-(xmin + i * xAbschnitt, -ABS(ymin - ymax) / 100 + ymax)
    LINE (xmin, ymin + i * yAbschnitt)-(xmin + ABS(xmin - xmax) / 100, ymin + i * yAbschnitt)
    LINE (xmin + i * xAbschnitt, ymin)-(xmin + i * xAbschnitt, ABS(ymin - ymax) / 100 + ymin)
    LINE (xmax, ymin + i * yAbschnitt)-(xmax - ABS(xmin - xmax) / 100, ymin + i * yAbschnitt)
  NEXT

 PRINT
 PRINT TAB(7); xmin; TAB(35); xBeschriftung$; TAB(74); xmax
 PRINT ; TAB(2); ymax

  FOR i = 1 TO 9
    PRINT
  NEXT

  FOR i = 1 TO 5
    PRINT TAB(2); yBeschriftung$(i)
  NEXT

  FOR i = 1 TO 8
    PRINT
  NEXT

 PRINT TAB(2); ymin

 RETURN


 Ende:
 PRINT : PRINT "Integrationszeit t ="; t#, "Zeitschritt h ="; h#
 KILL "H-Atom1.dat"
```

```
OPEN "H-Atom1.dat" FOR BINARY AS #1

 PUT #1, , h#
 PUT #1, , e#
 PUT #1, , t#

 FOR j = -1 TO 1 STEP 1
  FOR i = 1 TO 4
   PUT #1, , x#(i, j)
  NEXT i
 NEXT j

 FOR i = 1 TO 5
  PUT #1, , Anfangswert#(i)
 NEXT

 FOR j = 1 TO 6
  FOR i = 1 TO 4
   PUT #1, , f#(i, j)
  NEXT i
 NEXT j

 PUT #1, , xmax
 PUT #1, , xmin
 PUT #1, , ymax
 PUT #1, , ymin
 PUT #1, , xAbschnitt
 PUT #1, , yAbschnitt

CLOSE #1

KILL "H-Atom2.dat"
OPEN "H-Atom2.dat" FOR OUTPUT AS #2
 PRINT #2, xBeschriftung$
 FOR i = 1 TO 5
  PRINT #2, yBeschriftung$(i)
 NEXT
CLOSE #2

END
RETURN

pcs:
IF ABS(f#(1, 1)) > .0000001# THEN h1# = -x#(1, -1) / f#(1, 1) ELSE h1# = 0#
' h1# = (-2# * f#(1, 1) + SQR(4# * f#(1, 1) ^ 2# - 8# * f(1, 2) * x#(1, -1))) / (2# * f#(1, 2)) ELSE h1# = 0#
x# = x#(2, -1) + h1# * f#(2, 1) '+ (h1# ^ 2# / 2#) * f#(2, 2)
'p# = x#(4, -1) + h1# * f#(4, 1) '+ (h1# ^ 2# / 2#) * f#(4, 2)
p# = (x#(2, 1) - x#(2, -1)) / (2# * h#)
PSET (x#, p#)
RETURN

Ende1:
stopp = 1
RETURN
```

NUMTEST.BAS

```
SCREEN 12
WIDTH 80, 50
KEY(1) ON
CLS
'*************************************************************************
'**********  Rechnung ohne "FIX" in single precision  **********
'*************************************************************************
LOCATE 1, 1
PRINT "Vorwärts in single precision"
LOCATE 1, 40
PRINT "Rückwärts nach 20000 Schritten"

y! = 10: z! = .99 * 10                    'Anfangswerte
'************************** Vorwärtsschleife **********************
FOR n& = 2& TO 20000&
 x! = 1.957 * z! - y!

 IF n& < 20 THEN
  LOCATE n& + 2&, 1
  PRINT n&, x!
 END IF

 IF n& = 20000 THEN EXIT FOR
 y! = z!: z! = x!
 ON KEY(1) GOSUB ende
NEXT
'************************** Rückwärtsschleife *********************
y! = x!
FOR n& = 19998& TO 0& STEP -1&
 x! = 1.957 * z! - y!

 IF n& < 20 THEN
  LOCATE n& + 2, 40
  PRINT n&, x!
 END IF

 y! = z!: z! = x!
 ON KEY(1) GOSUB ende
NEXT
'*************************************************************************ii
'********************** Rechnung mit "FIX"  ************************
'*************************************************************************
y& = FIX(10# * 100000#): z& = FIX(100000# * .99# * 10#)
LOCATE 24, 1
PRINT "Vorwärts mit 'FIX'"
LOCATE 24, 40
PRINT "Rückwärts nach 20000 Schritten"
'************************ Vorwärtsschleife ***************************
FOR n& = 2& TO 20000&
 x& = FIX(1.957 * z&) - y&
 u# = x& / 100000

 IF n& < 22 THEN
  LOCATE n& + 25, 1
  PRINT n&, u#
 END IF

 IF n& = 20000 THEN EXIT FOR
 y& = z&: z& = x&
 ON KEY(1) GOSUB ende
NEXT
'************************** Rückwärtsschleife *********************
y& = x&
FOR n& = 19998& TO 0& STEP -1&
 x& = FIX(1.957 * z&) - y&
 u# = x& / 100000

 IF n& < 22 THEN
  LOCATE n& + 25, 40
```

```
    PRINT n&, u#
  END IF

  y& = z&: z& = x&
  ON KEY(1) GOSUB ende
NEXT

ende: END
```

RUNDUNG.BAS

```
SCREEN 12
WIDTH 80, 50
COLOR 0, 15
CLS
DIM x AS DOUBLE: DIM y AS DOUBLE: DIM u AS DOUBLE: DIM v AS DOUBLE: DIM n AS INTEGER
x = .5: u = .5

DO UNTIL n = 91

    IF n < = 45 THEN
      LOCATE n + 3, 1: PRINT n
      LOCATE n + 3, 4: PRINT x
      LOCATE n + 3, 22: PRINT u
    ELSE
      LOCATE n - 42, 41: PRINT n
      LOCATE n - 42, 44: PRINT x
      LOCATE n - 42, 62: PRINT u
    END IF

  n = n + 1
  y = 3.86# * x
  y = y * (1# - x)
  v = 3.86# * (1# - u)
  v = v * u
  x = y: u = v
LOOP

END
```

LOG.BAS:

```
SCREEN 12
PRINT "Die logistische Gleichung als invertierbarer Algorithmus F+ ausgeführt:"
e# = -.000001997#
neu:
CLS
e# = e# - .000000001#
z = 150000                    'Iterationsschritte
WINDOW (-z / 50, .7)-(z + z / 10, -.05)
LINE (0, 0)-(0, .5)
LINE (0, 0)-(z, 0)
LINE (0, .5)-(z, .5)
```

```
LINE (z, 0)-(z, .5)
y0# = .0001#       'Anfangswert
y# = 2# * y0# * (1# - y0#)
y# = INT(y# * 100000000#)
y# = y# / 100000000#
yn# = y# + e#
y# = INT(y0# * 100000000#)
y# = y# / 100000000#
yvor# = y#
PSET (0#, yvor#)
PSET (1#, yn#)
tvor# = 0#
tn# = 1#
'************************** Vorwärtsrechnung **********************************
Start:
KEY(2) ON
ON KEY(2) GOSUB Ende
y# = 2# * yn# * (1# - yn#) + (1# / 2#) - (SQR((4# - 8# * yn#))) / 4# - yvor#
y# = INT(y# * 100000000#)
y# = y# / 100000000#
ynach# = y#
t# = 2# * tn# - tvor#
t& = t# * 100#
t# = t& / 100#
tnach# = t#
PSET (tnach#, ynach#)
IF ynach# < -1 OR ynach# > 1 THEN GOTO neu
IF tnach# >= z THEN END 'GOTO Back
yvor# = yn#
yn# = ynach#
tvor# = tn#
tn# = tnach#
GOTO Start
'************************** Rückrechnung **************************************
Back: tvor# = tnach#
yvor# = ynach#
Start1:
y# = 2# * yn# * (1# - yn#) + (1# / 2#) - (SQR(4# - 8# * yn#)) / 4# - yvor#
y# = INT(y# * 100000000#)
y# = y# / 100000000#
ynach# = y#
t# = 2# * tn# - tvor#
t& = t# * 1000#
t# = t& / 1000#
tnach# = t#
PRESET (tnach#, ynach#)
IF tnach# <= 10 THEN PRINT tnach#, ynach#
IF tnach# <= 0 THEN END
yvor# = yn#
yn# = ynach#
tvor# = tn#
tn# = tnach#
GOTO Start1
Ende: END
RETURN
```

Literaturverzeichnis

[1] Charles H. Bennett, Rolf Landauer: "Grundsätzliche physikalische Grenzen beim Rechnen", Spekt. d. Wiss. Sept. 1985, p.94

[2] R. Landauer, M. Büttiker: "Drift and Diffusion in Reversible Computations", Phys. Scripta T9(1985)155

[3] James Hurley: "Resolution of the Time-Asymmetry Paradox", Phys. Rev. A22(1980)1205

[4] Wolfgang Schweizer: "Praktische numerische Algorithmen für klassische chaotische Systeme", Scriptum zur Vorlesung im WS 1992/93 an der Universität Tübingen

[5] Loup Verlet: "Computer 'Experiments' on Classical Fluids...", Phys. Rev. 159(1967)98

[6] Kendall E. Atkinson: "An Introduction to Numerical Analysis", John Wiley N.Y. 1978

[7] Lothar Berg: "Differenzengleichungen zweiter Ordnung mit Anwendungen", UTB 1980

[8] H. J. C. Berendsen, W. F. van Gunsteren: "Practical Algorithms for Dynamic Simulations" in: "Molecular-Dynamics Simulation of Statistical-Mechanical Systems; Proceedings of the International School of Physics 'Enrico Fermi'", North Holland 1986

[9] Iwo Bialynicki-Birula: "Does Measurement reverse the Direction of Intrinsic Time?", Physica B 151(1988)302

[10] Otto E. Rössler: "Endophysik - Die Welt des inneren Beobachters", Editor: Peter Weibel, Merve Berlin 1992

[11] M. Tuckerman et al: "Reversible Multiple Time Scale Molecular Dynamics", J. Chem. Phys. 97(1992)1990

[12] D. V. Anosov, Ya. G. Sinai: "Some Smooth Ergodic Systems" Russian Math. Surveys 22(1967)103

[13] O. E. Rössler, M. Hoffmann: "Quasiperiodization in Classical Hyperchaos", J. Comp. Chem. 8(1987)510

[14] Richard E. Gillilan, Kent R. Wilson: "Shadowing, Rare Events, and Rubber Bands. A Variational Verlet Algorithm for Molecular Dynamics", J. Chem. Phys. 97(1992)1757

[15] Otto E. Rössler: "Continuous Chaos - Four Prototype Equations" in: Annals New York Academy of Sciences 1979

[16] Robert I. McLachlan, Pan Atela: "The Accuracy of Symplectic Integrators", Nonlinearity 5(1992)541

[17] J. M. Sanz-Serna: "Symplectic Runge-Kutta and Related Methods: Recent Results", Physica D 60(1992)293;
R. D. Skeel, G. W. Gear: "Does Variable Step Size Ruin a Symplectic Integrator?" Physica D 60(1992)311;
D. Okunbor: "Canonical Methods for Hamiltonian Systems : Numerical Experiments", Pysica D 60(1992)314

[18] Soren Toxvaerd: "A New Algorithm for Molecular Dynamics Calculation", J. Comp. Phys. 47(1982)444

[19] Hanns Ruder, Wolfgang Schweizer: "Atomare Strukturen und Quantenchaos", Phys. in unserer Zeit 24(Nr 4,1993)163