4 Integration gewöhnlicher DGln; Anfangswertproblem

4.1 Einleitung

Physikalische *Relevanz* ist offensichtlich:

• Systeme von wenigen DGln

Punktmechanik: Wurfbahnen, ICBMs, Satelliten, Sonnensystem, Pendel, Doppelpendel, getriebene Systeme (z.B. getriebenes Teilchen im Doppelmuldenpotential = Duffing-Oszillator).

Systeme, die nach geeigneter Vereinfachung nur noch wenige Freiheitsgrade haben, z.B. Lorenz-Oszillator und andere dissipative Systeme aus diversen Gebieten der nonlinear science.

• Systeme von vielen DGln

„Molekulardynamik", d.h. Simulation (durch Integration der Newtonschen Bewegungsgleichungen) von Systemen aus vielen Teilchen mit gegebenen WW-Potentialen aus Chemie, FK-Physik und Stat. Mechanik:

- Phasenübergänge (auch Glasübergang)
- Strahlenschäden im FK, z.B. Halbleiterbauelementen.

Für diese Systeme mit *vielen* Teilchen oft spezielle Probleme, z.B. kann hier die Berechnung der WW-Kräfte wegen der vielen Teilchen besonders „teuer" sein, weshalb hier andere Verfahren verwendet werden als für Systeme mit wenigen Teilchen.

4.2 Reduktion auf DGl-Systeme 1. Ordnung

Typisches Beispiel: gedämpfter (nichtlinearer) Oszillator

$$m\ddot{x} + k\dot{x} - F(x) = 0 \tag{4.1}$$

DGl 2. Ordnung wegen Newton. Alle endlichdimensionalen gewöhnlichen DGln endlicher Ordnung lassen sich auf DGln *erster* Ordnung zurückführen, wie man am obigen Beispiel sieht: definiere

$$\dot{x} =: v, \tag{4.2}$$

dann ergibt sich

$$\begin{aligned} \dot{v} &= -\frac{k}{m}v + \frac{1}{m}F(x) \\ \dot{x} &= v \end{aligned} \tag{4.3}$$

also ein System von 2 DGln erster Ordnung. Also genügt es, DGl-Systeme der Form

$$\dot{\vec{x}} = \vec{f}(\vec{x}) \tag{4.4}$$

mit Anfangsbedingungen

$$\vec{x}(0) = \vec{x}_0 \tag{4.5}$$

zu betrachten. Interessant unter dem Aspekt numerischer DGl-Löungen sind aber auch nur die *nichtlinearen* DGln (also all die, die Sie *nicht* als Übungsaufgaben in Physik I-IV hatten!).

4.2.1 Lineare Systeme

Sei nämlich $f(\vec{x})$ *linear*, also

$$f(\vec{x}) = A \cdot \vec{x}, \tag{4.6}$$

(A ist eine Matrix) dann ist die Lösung von (4.4/4.5)

$$\vec{x}(t) = \exp(At) \cdot \vec{x}_0, \tag{4.7}$$

wonach man sich nur noch Gedanken über die praktische Auswertung von

$$\exp(At) = 1 + At + \frac{A^2 t^2}{2} + \cdots \tag{4.8}$$

machen muss. Letztere gelingt am einfachsten, wenn man $\vec{x}_0$ nach (Rechts-) Eigenvektoren von A entwickelt. In diesem Fall entpuppt sich die Lösung des DGl-Systems also als Eigenwertproblem, $\rightarrow \S 8$. (Im Spezialfall gekoppelter Schwingungen haben Sie das natürlich alle schon oft getan, als Sie nach Eigenmoden entwickelt haben.)

4.3 Euler-Methode

Bleiben also die nichtlinearen Systeme als einzig interessante, und wir befassen uns nun wirklich mit den Möglichkeiten, sie numerisch zu integrieren. Der Einfachheit halber arbeiten wir in einer Dimension; wenn nicht anderes gesagt ist, gilt alles mutatis mutandis auch für höhere Dimensionen.

$$\dot{x} = f(x,t) \quad ; \quad x(0) = x_0. \tag{4.9}$$

(Wollen auch getriebene Systeme behandeln können.) Die naive Methode, eine DGl numerisch zu lösen besteht nun darin, sich daran zu erinnern, dass Differentialquotienten als Limites von Differenzenquotienten aufgefasst werden können, und zu hoffen, dass man für kleine Differenzen dem Limes schon hinreichend nahe ist. Wir entwickeln also nach Taylor bis zur 1. Ordnung:

$$x(t + \Delta t) = x(t) + f(x(t), t)\Delta t + \mathcal{O}(\Delta t^2). \tag{4.10}$$

Daraus folgt

$$\begin{aligned} x(\Delta t) &\approx x_0 + f(x_0, t = 0)\Delta t \\ x((n+1)\Delta t) &\approx x(n\Delta t) + f(x(n\Delta t), n\Delta t)\Delta t. \end{aligned} \tag{4.11}$$

(4.11) heißt Euler-Cauchy-Polygon-Approximation und ist ein „Einschritt-Verfahren“ 1. Ordnung: es verwendet nur Information aus dem vorhergehenden Punkt und der *lokale* Fehler in jedem Schritt ist offensichtlich $\mathcal{O}(\Delta t^2)$; dazu kommen aber immer noch die akkumulierten (und fortgepflanzten !) Fehler aus (vielen) vorhergehenden Schritten. Da gerade für nichtlineare DGln kleine Fehler in den Anfangwerten exponentiell mit der Zeit anwachsen können, ist es wichtig, Verfahren mit möglichst kleinen Fehlern zu entwickeln. Natürlich kann man immer den Fehler durch Halbierung der Schrittweite Δt verringern, aber das verdoppelt die Rechenzeit und erhöht den Einfluss von Rundungsfehlern. Wir sollten also über das naive Verfahren (4.11) hinausgehen, da es weder besonders genau, noch besonders stabil ist. (Es gibt jedoch Fälle, in denen es sich nicht lohnt, aufwändigere Verfahren einzusetzen, z.B. wenn die Bewegung von *zufälligen* Kräften getrieben wird, wie bei stochastischen DGln, die sowohl in der statistischen Mechanik als auch in der Finanzmathematik (Black-Scholes-Modell) auftreten.)

4.4 Runge-Kutta 2. Ordnung

Besser ist ein Verfahren, welches als „midpoint method“ oder Runge-Kutta-Verfahren 2. Ordnung (RK2) bekannt ist; es beruht auf der Gleichung

$$x(t+\Delta t) = x(t) + \Delta t f(x + \frac{\Delta t}{2} f(x,t), t + \frac{\Delta t}{2}) + \mathcal{O}(\Delta t^3) \tag{4.12}$$

oder, “algorithmischer” geschrieben:

$$\begin{aligned} k_1 &= \Delta t f(x_n, t_n) \\ k_2 &= \Delta t f(x_n + \frac{k_1}{2}, t_n + \frac{\Delta t}{2}) \\ x_{n+1} &= x_n + k_2 + \mathcal{O}(\Delta t^3). \end{aligned} \tag{4.13}$$

Die Idee ist: man benutzt zur Extrapolation der Lösung über den *ganzen* Schritt die Ableitung an der Stelle, bis zu der man in einem *halben* Euler Schritt gekommen wäre. Das Verfahren ist eine Ordnung besser als Euler, braucht allerdings auch zwei Funktionsaufrufe von f. (Wir nehmen an, das sei der „teure“ Schritt.)
Um zu sehen, dass (4.13) tatsächlich ein Verfahren 2. Ordnung ist, entwickeln wir $x(t)$ um den Mittelpunkt des Intervalls:

$$x(t+\frac{\Delta t}{2} \pm \frac{\Delta t}{2}) = x(t+\frac{\Delta t}{2}) \pm \frac{\Delta t}{2} f(x(t+\frac{\Delta t}{2}), t+\frac{\Delta t}{2}) + \frac{\Delta t^2}{8} f'(x(t+\frac{\Delta t}{2}), t+\frac{\Delta t}{2}) + \mathcal{O}(\Delta t^3) \tag{4.14}$$

$$\Rightarrow x(t+\Delta t) - x(t) = \Delta t f(x(t+\frac{\Delta t}{2}), \frac{\Delta t}{2}) + \mathcal{O}(\Delta t^3) \tag{4.15}$$

denn die Terme gerader Ordnung heben sich heraus. Der Unterschied zu (4.11) liegt darin, dass das unbekannte $x(t+\frac{\Delta t}{2})$ à la Euler approximiert wurde, mit einem Fehler $\mathcal{O}(\Delta t^2)$. Zusammen mit dem Vorfaktor Δt ergibt sich also ein Fehler $\mathcal{O}(\Delta t^3)$, q.e.d.

4.5 Runge-Kutta 4. Ordnung

Das Verfahren kann verfeinert werden, indem geeignete Näherungen für die Ableitung kombiniert werden, so dass die Fehler sich gerade bis zu einer gewünschten Ordnung kompensieren. Ein sehr gebräuchliches Verfahren ist *Runge-Kutta 4. Ordnung* (RK4):

$$\begin{aligned} k_1 &= \Delta t f(x_n, t_n) \\ k_2 &= \Delta t f(x_n + \frac{k_1}{2}, t_n + \frac{\Delta t}{2}) \\ k_3 &= \Delta t f(x_n + \frac{k_2}{2}, t_n + \frac{\Delta t}{2}) \\ k_4 &= \Delta t f(x_n + k_3, t_n + \Delta t) \\ x_{n+1} &= x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathcal{O}(\Delta t^5). \end{aligned} \tag{4.16}$$

Hier wird also die Ableitung viermal berechnet: am Startpunkt, an zwei versuchsweisen Mittelpunkten, und an einem versuchsweisen Endpunkt des Intervalls. Der Beweis, dass der Fehler im Einzelschritt $\mathcal{O}(\Delta t^5)$ ist, sei dem Leser überlassen. (Vgl. z.B. [Schwarz, 1997] für allgemeine Beweise für ganze Klassen von Verfahren.)
Mit vier Auswertungen von f hätte man auch zwei RK2-Schritte bestreiten können; RK4 ist also nur dann besser als RK2, wenn man die Schrittweite mindestens verdoppeln kann, bei gleichbleibender Genauigkeit. Das ist zwar meistens so, aber nicht

garantiert. RK4 ist der „alte Ackergaul“ [Press et al., 1992] unter den numerischen Lösungsverfahren: zuverlässig, aber nicht übermäßig genau, und nicht übermäßig schnell. Für spezielle Probleme kann es besser sein, ein hochgezüchtetes „Rennpferd“-Verfahren zu benutzen (s. weiter unten, oder [Press et al., 1992]).
Die Runge-Kutta-Verfahren und andere lassen sich gut auf einheitliche Weise herleiten, indem man die zeitliche Entwicklung in einem Schritt zunächst formal exakt als Integral schreibt, und dann unterschiedliche Methoden (Trapezregel, Simpsonformel, vgl. §3) zur approximativen Berechnung des Integrals verwendet. Diese Herleitung deckt auch die so genannten Prädiktor-Korrektor-Verfahren ab. Wie der Name schon sagt, wird dort im einfachsten Fall eine einfache (Euler-) Vorhersage über das Ergebnis eines Schritts gemacht, die dann verfeinert (korrigiert) wird. Eine schöne Zusammenfassung findet sich bei [Kierfeld, 2016].

4.6 Schrittweitenanpassung

Da bei RK-Verfahren die einzelnen Schritte unabhängig voneinander sind, kann die Schrittweite Δt in jedem Schritt beliebig geändert werden. Dies kann man sich zur *Schrittweitenanpassung* zunutze machen, um bei vorgegebener Genauigkeit maximale Geschwindigkeit zu erreichen. In dem schönen Buch von Korsch und Jodl [Korsch and Jodl, 1994] werden beispielsweise für die dort behandelten hochempfindlichen chaotischen DGln keine vornehmeren Algorithmen verwendet als RK4 mit Schrittweitenanpassung. Für das Doppelpendel kann man mit den im Buch mitgelieferten Programmen mit Euler, RK2 und RK4 mit und ohne Schrittweitenanpassung experimentieren. Für den häufigen Fall, dass nur mal eben ein Plot der Lösung $x(t)$ erzeugt werden soll, um ihr qualitatives Verhalten zu sehen, benutzt man RK4 mit irgendeiner nach Gefühl gewählten Schrittweite, und anschließend wiederholt man mit der halben oder doppelten Schrittweite und überprüft die erzielte Genauigkeit.
Laufende Schrittweitenanpassung, einfachste Variante: benutze RK4 zwei Schritte mit Δt, und, ausgehend vom selben Startpunkt, einen Schritt mit $2\Delta t$; Differenz: Δx, zu vergleichen mit $\Delta x_{\max}$, dem zulässigen „Fehler“. (Natürlich ist Δx nicht der Fehler; den kennt man leider nicht.) Je nachdem Δx betragsmäßig größer oder wesentlich kleiner ist, wird Δt vergrößert oder verkleinert, ausgehend von der Daumenregel

$$\Delta x \sim \Delta t^5. \tag{4.17}$$

Mehraufwand für die Kontrolle: 2 kleine Schritte: 8 f-Auswertungen; 1 großer Schritt: zusätzlich 3 f-Auswertungen (Anfangspunkt ist schon erledigt!) $\Rightarrow$ Overhead-Faktor $11/8 = 1.375$. Die zusätzlichen f-Auswertungen können sogar genutzt werden, um aus RK4 eine Formel 5. Ordnung zu machen, vgl. [Press et al., 1992].
Effizientere Möglichkeit der Schrittweitenanpassung nach Runge-Kutta-Fehlberg: aus 6 f-Auswertungen kann man eine Formel 5. Ordnung *und* eine Formel 6. Ordnung generieren (daher auch der Name “embedded RK”, den diese Methode auch trägt). Wenn Δx die Differenz zwischen den beiden Lösungen in einem Schritt Δt ist, dann ändere man Δt auf

$$\Delta t' = \Delta t \left(\frac{\Delta x_{\max}}{\Delta x} \right)^{1/5} \ldots \tag{4.18}$$

... und wiederhole den Schritt falls $\Delta t' < \Delta t$
... und gehe zum nächsten Schritt falls $\Delta t' > \Delta t$.
Es gibt mehrere Varianten von RK-Fehlberg; mit unterschiedlichen Stützstellen zur f-Auswertung und dto. Gewichtsfaktoren. [Press et al., 1992] empfehlen die Variante von Cash und Karp, die dort als Tabelle aufgeführt ist und hier nicht reproduziert werden muss.
Welche Genauigkeit $\Delta x_{\max}$ soll man verlangen ? — Hängt vom Problem ab. Bei mehrdimensionalen Problemen muss man natürlich für jede Komponente ein $\Delta x_{\max}$ vorschreiben, und Δt so wählen, dass *alle* Komponenten ok sind. Es wird oft sinnvoll sein, eine *relative* Genauigkeit zu fordern:

$$\Delta x_{\max} = \varepsilon |x|. \tag{4.19}$$

Dies kann allerdings eine Vollbremsung zur Folge haben, wenn x eine Nullstelle hat. Vgl. [Press et al., 1992] für weitere nützliche Hinweise und einen nach den geschilderten Methoden gebauten RK-Fehlberg-Integrator ODEINT.

4.7 Bulirsch-Stoer-Methode

Eine andere Methode, die hier nur ganz grob geschildert werden soll (Details: [Press et al., 1992]): Bulirsch-Stoer-Methode. Hohe Genauigkeit bei vergleichsweise geringem Aufwand, aber ungeeignet für DGl-Systeme mit stark variierenden Funktionen, oder gar Singularitäten im Innern des Integrationsintervalls. (Dies ist ein „Rennpferd"-Verfahren.)
Idee: Betrachte ein Intervall Δt (muss gar nicht mal so klein sein) und benutze ein einfaches Verfahren 2. Ordnung (sog. modified midpoint method) mit Unterteilung von Δt in Unterschritte $\frac{\Delta t}{n}; n = 2, 4, 6, 8, \ldots$. Aus der Folge der Ergebnisse kann man sowohl nach $n \to \infty$ extrapolieren als auch Fehlerabschätzungen gewinnen, ähnlich wie bei der Romberg-Methode (§3) zur Berechnung von Integralen.
Offensichtlich muss hier die Funktion f sehr oft ausgewertet werden, dafür können aber sehr große Schrittweiten erzielt werden. Das Extrapolieren funktioniert natürlich nur, wenn f innerhalb der Schrittweite nicht stark variiert. Für die Anwendung auf wechselwirkende Vielteilchensysteme (MD-Simulation, §5) erscheint das Verfahren weniger geeignet, da dann jede Auswertung von f in Wirklichkeit $\frac{n(N-1)}{2}$ Funktionsaufrufe bedeutet.

4.8 Steife DGl-Systeme

Verschiedene Freiheitsgrade können sehr unterschiedliche Zeitskalen haben; z.B. Doppelpendel mit kleiner Masse an kurzem Faden oben und großer Masse an langem Faden unten. (Bildchen)
Betrachten hier ein etwas simpleres Beispiel (nur 2 lineare DGln, während es beim Doppelpendel 4 nichtlineare wären)

$$\begin{aligned} \dot{u} &= 998u + 1998v \\ \dot{v} &= -999u - 1999v \end{aligned} \tag{4.20}$$

mit Anfangsbedingungen

$$u(0) = 1 \quad ; \quad v(0) = 0. \tag{4.21}$$

Eine kleine Transformation enthüllt den heimtückischen Charakter dieses Systems:

$$\begin{aligned} y &:= u + v \\ z &:= u + 2v \end{aligned} \tag{4.22}$$

$\Longrightarrow$

$$\begin{aligned} \dot{y} &= -y \\ \dot{z} &= -1000z. \end{aligned} \tag{4.23}$$

Die Lösung ist im vorliegenden Fall natürlich trivial:

$$y(t) = e^{-t} \qquad z(t) = e^{-1000t} \tag{4.24}$$

und damit

$$\begin{aligned} u(t) &= 2e^{-t} - e^{-1000t} \\ v(t) &= -e^{-t} + e^{-1000t}. \end{aligned} \tag{4.25}$$

Die typischen Zeitskalen unterscheiden sich um einen Faktor 1000. Der "schnelle" Anteil der Lösung ist nach kurzer Zeit völlig ausgestorben. Trotzdem muss man aber mit einer kleinen Schrittweite arbeiten, damit der schnelle Anteil der DGl nicht zur Instabilität führt.
Um zu sehen, wie es zu solchen Instabilitäten durch zu große Schrittweite kommt, und was man dagegen unternehmen kann, betrachten wir eine (lineare) DGl

$$\dot{x} = -cx \quad (c > 0) \tag{4.26}$$

und behandeln sie mit dem simpelsten Algorithmus à la Euler (4.11)

$$x_{n+1} = x_n + \Delta t \dot{x}_n = (1 - c\Delta t)x_n. \tag{4.27}$$

Diese Variante des Euler-Algorithmus nennt man explizites (oder Vorwärts-) Euler-Schema.
Wenn man $\Delta t > 2/c$ wählt, divergiert die Folge der x_n, denn dann ist

$$|1 - c\Delta t| > 1.$$

Diesen Defekt kann man leicht beheben, indem man die rechte Seite der DGl nicht bei x_n auswertet, sondern bei x_{n+1} (implizites oder Rückwärts-Schema)

$$\begin{aligned} x_{n+1} = x_n + \Delta t\, \dot{x}_{n+1} = x_n - c\, \Delta t\, x_{n+1} \\ \Rightarrow \qquad x_{n+1} = \frac{x_n}{1 + c\Delta t}. \end{aligned} \tag{4.28}$$

Damit

$$\lim_{n\to\infty} x_n = 0, \tag{4.29}$$

wie es sich gehört, und zwar für beliebige Δt. Diese Betrachtung verallgemeinert sich leicht auf mehrere Dimensionen; sei ein (immer noch lineares) DGl-System gegeben:

$$\dot{\vec{x}} = -C\vec{x} \tag{4.30}$$

mit einer positiv definiten Matrix C.
Explizites Schema:

$$\vec{x}_{n+1} = (1 - \Delta t\ C)\vec{x}_n, \tag{4.31}$$

stabil solange der größte EW von $(1 - \Delta t\ C)$ betragsmäßig kleiner als 1, also $\Delta t < 2/\lambda_{\max}$, wo $\lambda_{\max}$ der größte EW von C.
Implizites Schema:

$$\vec{x}_{n+1} = (1 + \Delta t\ C)^{-1}\vec{x}_n \tag{4.32}$$

ist absolut stabil, denn alle EW von $(1+\Delta t\ C)^{-1}$ sind kleiner als 1. Allerdings muss man nun eine Matrix invertieren (oder äquivalent dazu: ein lineares Gleichungssystem lösen).
Unglücklicherweise interessieren wir uns aber eigentlich für nichtlineare DGln, also z.B.

$$\dot{\vec{x}} = \vec{f}(\vec{x}). \tag{4.33}$$

Nach dem impliziten Schema gibt das

$$\vec{x}_{n+1} = \vec{x}_n + \Delta t\ \vec{f}(\vec{x}_{n+1}) \tag{4.34}$$

und die Komponenten von $\vec{x}_{n+1}$ ergeben sich als Lösung eines ekligen nichtlinearen Gleichungssystems, das evtl. viel Aufwand erfordert. Wir versuchen nun, die rechte Seite von (4.34) zu linearisieren, was für halbwegs friedliches $\vec{f}$ und nicht zu große Schritte gehen sollte:

$$\vec{f}(\vec{x}_{n+1}) = \vec{f}(\vec{x}_n) + \left.\frac{\partial \vec{f}}{\partial \vec{x}}\right|_{\vec{x}_n} (\vec{x}_{n+1} - \vec{x}_n). \tag{4.35}$$

Hierbei bedeutet $\left.\frac{\partial \vec{f}}{\partial \vec{x}}\right|_{\vec{x}_n}$ die Matrix der partiellen Ableitungen von Komponenten von $\vec{f}$ nach Komponenten von $\vec{x}$, also die Jacobimatrix.

$$\Longrightarrow \vec{x}_{n+1} = \vec{x}_n + \Delta t \left[\vec{f}(\vec{x}_n) + \left.\frac{\partial \vec{f}}{\partial \vec{x}}\right|_{\vec{x}_n} (\vec{x}_{n+1} - \vec{x}_n)\right] \tag{4.36}$$

oder

$$(\vec{x}_{n+1} - \vec{x}_n)\left(\mathbf{1} - \Delta t\ \left.\frac{\partial \vec{f}}{\partial \vec{x}}\right|_{\vec{x}_n}\right) = \Delta t \vec{f}(\vec{x}_n).$$

Das ist äquivalent zu

$$\vec{x}_{n+1} = \vec{x}_n + \Delta t \left(\mathbf{1} - \Delta t\ \left.\frac{\partial \vec{f}}{\partial \vec{x}}\right|_{\vec{x}_n}\right)^{-1} \vec{f}(\vec{x}_n). \tag{4.37}$$

Für den speziellen linearen Fall $\vec{f}(\vec{x}) = -C\vec{x}$ erhält man die alte DGl (4.30) und aus (4.37) wird die alte Lösung (4.32). Wenn die in einem impliziten (Rückwärts-) Schema auftretenden nichtlinearen Gleichungen durch Linearisierung gelöst werden, nennt man das Schema "semi-implizit". In jedem Schritt wird ein lineares Gleichungssystem gelöst. Wir haben also gerade die semi-implizite Euler-Methode vorgestellt. Die Euler-Methode ist eine Methode erster Ordnung. Man kann auch Methoden höherer Ordnung (wie RK oder Bulirsch-Stoer) so verallgemeinern, dass durch semi-implizite Schemata die Stabilität verbessert wird. Nähere Diskussion solcher Verallgemeinerung und wichtige Hinweise zur Handhabung („Risiken und Nebenwirkungen") finden sich in [Press et al., 1992].

4.9 Zurück zur Natur: Integration Newtonscher Bewegungsgleichungen

Betrachte ein Kraftfeld ohne Geschwindigkeitsabhängigkeit (keine Reibung, keine Magnetfelder):

$$\dot{\vec{r}} = \vec{v} \tag{4.38}$$

$$\dot{\vec{v}} = \frac{1}{m}\vec{F}(\vec{r},t) = \vec{a}(\vec{r},t) \tag{4.39}$$

ist dann die als DGl 1. Ordnung formulierte Bewegungsgleichung. Für N Teilchen sprechen wir dann über $3N$-dimensionale Vektoren $\vec{r},\vec{v}$ und $\vec{a}$. Mit $\vec{r}_n = \vec{r}(t_n) = \vec{r}(t_0 + n\Delta t)$, $\vec{v}_n$ und $\vec{a}_n$ analog liefert Euler dann z.B.

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_n\Delta t + \mathcal{O}(\Delta t^2) \tag{4.40}$$

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n\Delta t + \mathcal{O}(\Delta t^2), \tag{4.41}$$

oder ein einfaches Prädiktor-Korrektor-Verfahren (ohne Beweis): Prädiktor:

$$\vec{r}_{n+1,p} = \vec{r}_n + \vec{v}_n\Delta t + \mathcal{O}(\Delta t^2) \tag{4.42}$$

$$\vec{a}_{n+1,p} = \vec{a}(\vec{r}_{n+1,p}, t_{n+1}) \tag{4.43}$$

und nach Korrektur:

$$\vec{v}_{n+1} = \vec{v}_n + \frac{1}{2}(\vec{a}_{n+1,p} + \vec{a}_n)\Delta t + \mathcal{O}(\Delta t^3) \tag{4.44}$$

$$\vec{r}_{n+1} = \vec{r}_n + \frac{1}{2}(\vec{v}_{n+1} + \vec{v}_n)\Delta t + \mathcal{O}(\Delta t^3). \tag{4.45}$$

(In der letzten Zeile sollte systematischerweise der Prädiktor $\vec{v}_{n+1,p}$ auftauchen; es ist aber $\vec{v}_{n+1,p} = \vec{v}_n + \vec{a}_n\Delta t = \vec{v}_{n+1} + \mathcal{O}(\Delta t^2)$ (vgl. (4.41)), so dass das im Ergebnis (nach Multiplikation mit Δt) keinen Unterschied macht und die Berechnung von $\vec{v}_{n+1,p}$ überflüssig ist und eingespart werden kann.
Speziell für MD-Simulationen mit vielen Teilchen eignen sich zwei Algorithmen, die nun noch vorgestellt werden sollen.

4.9.1 Verlet-Algorithmen (Loup Verlet 1967)

Wir betrachten zwei Taylorentwicklungen

$$\vec{r}_{n\pm1} = \vec{r}_n \pm \vec{v}_n\Delta t + \frac{1}{2}\vec{a}_n\Delta t^2 \pm ... \tag{4.46}$$

und addieren; dann folgt

$$\vec{r}_{n+1} = 2\vec{r}_n - \vec{r}_{n-1} + \vec{a}_n\Delta t^2 + \mathcal{O}(\Delta t^4). \tag{4.47}$$

(Das hätte man auch durch Diskretisierung der 2. Ableitung in $\ddot{\vec{r}} = \vec{a}(\vec{r},t)$ à la (3.4) erhalten.) Die Geschwindigkeiten, wenn erwünscht (z.B. für die kinetische Energie), müssen nachträglich berechnet werden, z.B. mit der symmetrischen 2-Punkt-Formel (3.3) für die Ableitung:

$$\vec{v}_n = \frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2\Delta t} + \mathcal{O}(\Delta t^2). \tag{4.48}$$

Eigenschaften des Algorithmus:

- Einzelschrittfehler $\mathcal{O}(\Delta t^2)$ in $\vec{v}$ und $\mathcal{O}(\Delta t^4)$ in $\vec{r}$.
- Benötigt pro Zeitschritt nur eine Kraftberechnung $m\vec{a}_n$, genau wie Euler, ist aber genauer, und weniger aufwändig als Prädiktor-Korrektor.
- Ist in der angegebenen Form nicht „selbststartend", da $\vec{r}_0$ und $\vec{r}_{-1}$ benötigt, aber in der Regel $\vec{r}_0$ und $\vec{v}_0$ gegeben. Abhilfe:

$$\vec{r}_{-1} := \vec{r}_0 - \vec{v}_0\Delta t + \frac{1}{2}\vec{a}_0\Delta t^2.$$

Äquivalent zum Verlet-Algorithmus und selbststartend ist der Geschwindigkeits-Verlet-Algorithmus mit den drei Rechenschritten (in dieser Reihenfolge)

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_n\Delta t + \frac{1}{2}\vec{a}_n\Delta t^2 \tag{4.49}$$

$$\vec{a}_{n+1} = \vec{a}(\vec{r}_{n+1}, t_{n+1}) \tag{4.50}$$

$$\vec{v}_{n+1} = \vec{v}_n + \frac{1}{2}(a_{n+1} + \vec{a}_n)\Delta t. \tag{4.51}$$

Die Äquivalenz der beiden Algorithmen sieht man, wenn man nach dem Geschwindigkeits-Verlet-Algorithmus $\vec{r}_{n+2}+\vec{r}_n$ ausgehend von $\vec{r}_{n+1}$ ausdrückt und mit dem ursprünglichen Verlet-Algorithmus vergleicht; vgl. [Kierfeld, 2016].

4.9.2 Leapfrog-Algorithmus (Leapfrog = Bocksprung)

Bei diesem Algorithmus werden $\vec{r}$ und $\vec{v}$ abwechselnd in Halbschritten berechnet:

$$\vec{v}_{n+1/2} = \vec{v}_{n-1/2} + \vec{a}_n\Delta t \tag{4.52}$$

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_{n+1/2}\Delta t, \tag{4.53}$$

und dann

$$\vec{a}_{n+1} = \vec{a}(\vec{r}_{n+1}, t_{n+1}). \tag{4.54}$$

Das liefert identische Trajektorien $\{\vec{r}_n\}$ wie Verlet; Nachteil: potentielle und kinetische Energie können nie zum gleichen Zeitpunkt berechnet werden, so dass die Energieerhaltung als Kontrollmöglichkeit entfällt.
Alle hier vorgestellten Algorithmen benötigen nur eine einzige Kraftberechnung (aus $\frac{N(N-1)}{2}$ Wechselwirkungen!) pro Zeitschritt.

4.10 Weitere Methoden

Es gibt noch einige weitere Methoden zur Lösung von DGl-Anfangswertproblemen; insbesondere Verfahren, die nicht nur Information aus dem letzten Schritt benutzen, um den nächsten zu machen. Besonders wichtig darunter sind die so genannten *Prädiktor-Korrektor-Verfahren*, die die Lösung polynomial aus der Vergangenheit extrapolieren und dann korrigieren. Wegen dieser Vergangenheits-Abhängigkeit sind sie nicht ganz trivial zu starten. Trotz vieler Entwicklungsarbeit, die in diese Verfahren gesteckt wurde, scheinen sie insgesamt zunehmend von RK (auf der simplen Seite) und Bulirsch-Stoer (auf der ausgefeilten Seite) verdrängt zu werden. Für eine allgemeine Beschreibung dieser Verfahren konsultiere man [Press et al., 1992]; dort finden sich auch weiter führende Literaturhinweise.

4.11 Softwarepakete, Mathematica etc.

Praktisch alle wissenschaftlichen Softwarepakete, etwa die Programmbibliotheken von CERN, NAG, Numerical Recipes enthalten (meist mehrere) Routinen zur Lösung von DGln. Es lohnt sich für ernstzunehmende wissenschaftliche Arbeiten nicht, diese gut rollenden Räder neu zu erfinden; statt dessen sollte man lernen, wie man solche Routinen korrekt bedient und in eigene Programme einbindet. Natürlich kann es Freude machen, etwa die drei gekoppelten DGln des Lorenz-Oszillators „mal eben“ mit einem RK4-Verfahren zu programmieren und die Trajektorie im 3D Phasenraum auf dem Bildschirm zu verfolgen...
An dieser Stelle sei auch auf Computer-Mathematik-Systeme hingewiesen, wie Mathematica, Maple, Reduce, Axiom, MuPAD, ... Viele dieser Systeme enthalten Möglichkeiten zum Lösen von DGln. Zusammen mit den meist auch vorhandenen Graphik-Paketen bieten diese Systeme komfortable Möglichkeiten, rasch einen qualitativen Überblick über das Verhalten der Lösung eines DGl-Systems zu bekommen.

Literatur

[Kierfeld, 2016] Kierfeld, J. (2016). Computational physics. Skriptum zur Vorlesung SS 2016.

[Korsch and Jodl, 1994] Korsch, H. and Jodl, H.-J. (1994). *Chaos — A Program Collection for the PC*. Springer-Verlag, Berlin.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in FORTRAN: the Art of Scientific Computing*. Cambridge U.P., Cambridge.

[Schwarz, 1997] Schwarz, H. (1997). *Numerische Mathematik*. B.G. Teubner Verlag, Stuttgart.