Kapitel 3

Lösung der Newtonschen Bewegungsgleichung

3.1 Der Harmonische Oszillator

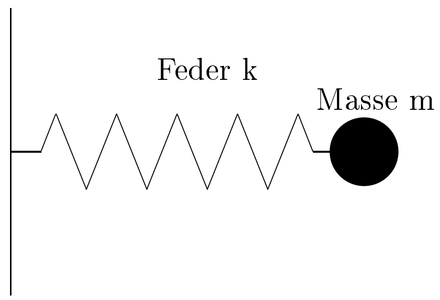


Abbildung 3.1: Schwingungsfeder

Wir gehen davon aus, daß die Masse punktförmig (die Federmasse ist vernachlässigt), die Gravitation gleich Null ist und die Bewegung reibungslos. Dann gilt das Newtonsche Bewegungsgesetz für den harmonischen Oszillator:

$$m\ddot{x} = -kx \tag{3.1}$$

Dabei ist k die Federkonstante. Es gelten die Anfangsbedingungen $x(0) = x_0$ und $\dot{x}(0) = v_0$. Die Lösung dieser Gleichung ist

$$x(t) = A\sin(\omega t + \delta) \qquad \text{mit} \quad \omega^2 = \frac{k}{m} \tag{3.2}$$

Die Anfangsbedingungen ergeben sich durch das Einsetzen von $t = 0$ zu $x_0 = A \sin \delta$ und $v_0 = A\omega \cos \delta$. Daraus werden dann die Konstanten A und δ berechnet:

$$\tan \delta = \frac{x_0 \omega}{v_0} \quad \text{und} \quad A = \frac{x_0}{\sin \delta} \tag{3.3}$$

Um eine Funktion in einem Programm verwenden zu können, muß sie diskretisiert werden, d.h. eine kontinuierliche Kurve wird in Teilstücke zerlegt. Dafür benutzen wir ein Feld `x[i]`, wobei `i` nur ganzzahlige Werte annehmen kann. Unsere Zeit t berechnet sich dann durch $t =$`i`dt (dt ist das Zeitintervall zwischen zwei Stützstellen). Damit entspricht der Übergang $t \to t + dt$ dem Übergang `i` $\to$ `i+1`.

Das folgende Programm gibt die exakte Lösungsgleichung (2) wieder:

```
#include<iostream>
using std::cin;
using std::cout;
#include<fstream>
using std::ostream;
#include<cmath>

int main()
{
// define x(t)
   double x[1000], t;  // output-variables

// parameter definition
   double A, delta;     // A=amplitude, delta=phase-angle
   double mass, kfeder; // mass=mass, kfeder=spring-const.
   double t_max, dt;    // t_max=total time, dt=time-interval

// read parameters from std::cin
   cout << "Amplitude      ";  cin >> A;
   cout << "Phasenwinkel   ";  cin >> delta;
   cout << "Masse          ";  cin >> mass;
   cout << "Federkonstante ";  cin >> kfeder;
   cout << "Gesamtzeit     ";  cin >> t_max;
   cout << "Zeitintervall  ";  cin >> dt;
   double omega=sqrt(kfeder/mass);

// loop from t=0 to t=t_max
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) {
        x[i]=A*sin(omega*t+delta);  // calculate the funktion
        t=t+dt;                     // increase time
```

```
   }

// output 2 columns to file harmon.dat
   ofstream outfile("harmon.dat");
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) {
      outfile << t << " " << x[i] << "\n";
      t=t+dt;
   }
   outfile.close();                  // close file
}
```

Die Funktion plottet man dann am besten mit dem Programm `gnuplot` und dem Befehl `plot 'harmon.dat'`. Auf dem Bildschirm sollte dann die in Abb. 3.2 dargestellte Zeichnung erscheinen.

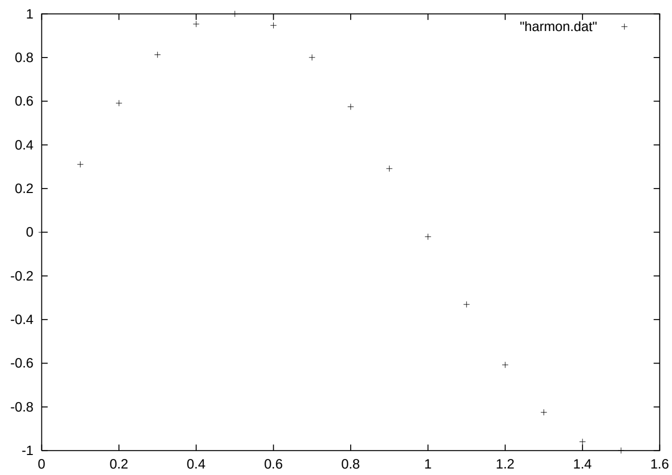


Abbildung 3.2: Ausdruck von gnuplot für $A = 1$, $\delta = 0$, $m = 1$, $k = 10$, $t_{\max} = 1.5$, und $\Delta t = 0.1$, direkt aus der analytischen Lösung berechnet.

3.1.1 Das Euler-Verfahren

Das Euler-Verfahren ist die einfachste Möglichkeit eine Differentialgleichung numerisch zu lösen. Zunächst muß man die Gleichung umformen. Das Newtonsche Bewegungsgesetz lautet $\ddot{x} = -\omega^2 x$. Mit der Geschwindigkeit $v = \dot{x}$ ensteht die Gleichung $\dot{v} = -\omega^2 x$. Die Ableitungen von x und v können für $\Delta t \to 0$ durch den Differenzenquotienten dargestellt werden:

$$\dot{x} = \frac{x(t+\Delta t) - x(t)}{\Delta t} \quad \Rightarrow \quad x(t+\Delta t) = x(t) + v\Delta t \tag{3.4}$$

$$\dot{v} = \frac{v(t+\Delta t) - v(t)}{\Delta t} \quad \Rightarrow \quad v(t+\Delta t) = v(t) - \omega^2 x\Delta t \tag{3.5}$$

Mit Hilfe dieser Gleichungen kann nun eine numerische Approximation im Computer durchgeführt werden. Zuerst werden die Formeln wieder diskretisiert und anschließend wird der Einfachheit halber der Index `i` um Eins verringert. Dies entspricht der Berechnung der aktuellen Werte `i` aus den alten Werten `i-1`. Die Formeln haben einen Fehler in der Größenordnung $O(\Delta t^2)$. Dieser Fehler wirkt sich im Verlauf des Verfahrens durch ansteigende Energie aus. Die Gesamtenergie ist wegen Energieerhaltung aber konstant (Energie = kinetische Energie + Federenergie): $E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2$. Zur Kontrolle der Energieerhaltung wird die Gesamtenergie E ebenfalls im Programm diskret berechnet.

```
// ...
   const int nx=1000;                                    // fieldlength
   double x[nx], v[nx], e[nx], t;
// ...
// make sure that the number of intervals is smaller than nx
   if( t_max/dt >= nx) {
      cout << "ERROR: dt is too small, set to: " << t_max/nx << '\n';
      return -1;
   }
// setup initial conditions
   x[0]=A*sin(delta);
   v[0]=A*omega*cos(delta);
// loop from t=0 to t=t_max
   for( int i=1; i<=t_max/dt; i++ ) {
        v[i]=v[i-1]-omega*omega*x[i-1]*dt;               // next velocity
        x[i]=x[i-1]+v[i-1]*dt;                           // next position
        e[i]=0.5*(mass*v[i]*v[i]+kfeder*x[i]*x[i]);
   }
// output to file
   ofstream outfile("eulerh.dat");
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) {
      outfile << t << " " << x[i] << ' ' << v[i] << ' ' << e[i] << "\n";
      t=t+dt;
   }
   outfile.close();
```

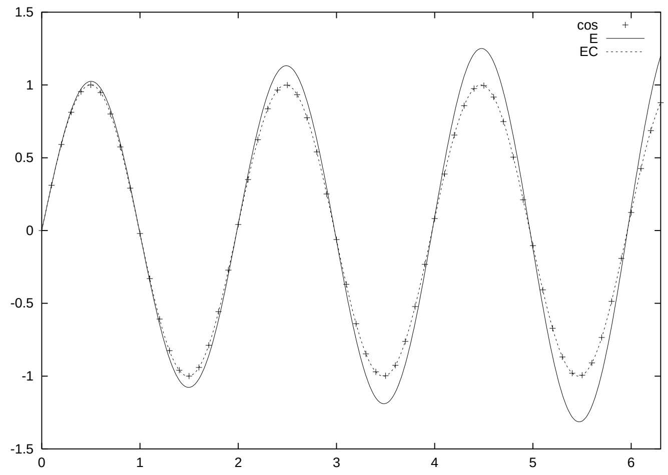


Abbildung 3.3: Ausdruck von gnuplot für $A = 1$, $\delta = 0$, $m = 1$, $k = 10$, $t_{\max} = 6.3$, und $\Delta t = 0.01$, berechnet mit dem Euler-Verfahren (E) im Vergleich mit der exakten Lösung aus Abb. 3.2, und dem Euler-Cromer Verfahren (EC).

3.1.2 Die Euler-Cromer-Methode

Eine besseres Ergebnis erhält man mit der Euler-Cromer-Methode. In der Zeile:

```
x[i]=x[i-1]+v[i-1]*dt;              // next position
```

ersetzt man den alten Wert `v[i-1]` durch den neuen Wert `v[i]`, der ja gerade eben berechnet wurde. Dies ist erlaubt, da durch die Diskretisierung zu jedem Wert ein Bereich von Δt gehört. In diesem Fall ist die numerische Lösung viel besser, wie man leicht sehen kann wenn man die analytische Lösung mit der numerischen vergleicht. Ausserdem ist beim Euler-Cromer-Verfahren die Energie (bis auf kleine Schwankungen) besser erhalten.

3.2 Das Pendel

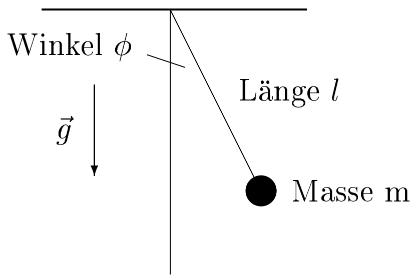


Abbildung 3.4: Das Pendel mit masseloser Aufhängung

Bei einem Pendel ist die (punktförmige) Masse m an einem starren, masselosen Stab der Länge l aufgehängt und schwingt reibungsfrei im Schwerefeld $\vec{g}$. $\varphi(t)$ ist der Auslenkwinkel des Pendels zur Zeit t aus der Ruhelage heraus. Um die Verlaufsfunktion aufzustellen benutzen wir den Energieerhaltungssatz: Die Gesamtenergie E, besthend aus kinetischer und potentieller Energie bleibt erhalten, d.h. $\dot{E} = 0$. Der maximale Auslenkwinkel ist dann $\varphi_{\max}$, mit der zugehörigen Winkelgeschwindigkeit $\omega(\varphi_{\max}) = \dot{\varphi}_{\max} = 0$. Die Gesamtenergie ist also

$$E = \frac{1}{2} m l^2 \dot{\varphi}^2 - mgl \cos\varphi = -mgl \cos\varphi_{\max} \ . \tag{3.6}$$

Die analytische Lösung erhält man durch Auflösen nach $\dot{\varphi}$

$$\dot{\varphi}^2 = \frac{2g}{l}(\cos\varphi - \cos\varphi_{\max}) \ , \tag{3.7}$$

Wurzel ziehen und Inversion

$$\frac{dt}{d\varphi} = \frac{1}{\sqrt{\frac{2g}{l}(\cos\varphi - \cos\varphi_{\max})}} \ , \tag{3.8}$$

Trennung der Variablen t und φ

$$dt = \frac{d\varphi}{\sqrt{\frac{2g}{l}(\cos\varphi - \cos\varphi_{\max})}} \ , \tag{3.9}$$

und anschließender Integration von der Zeit t_0 bis t und den entsprechenden Winkeln φ_0 bis $\varphi(t)$

$$t(\varphi) - t_0 = \sqrt{\frac{l}{2g}} \int_{\varphi_0}^{\varphi(t)} \frac{d\psi}{\sqrt{\cos\psi - \cos\varphi_{\max}}} \ . \tag{3.10}$$

Somit kann man dann die Schwingungsdauer T des Pendels bestimmen. Für eine Periode benötigt das Pendel viermal die Dauer der Bewegung von $\varphi_0 = 0$ nach $\varphi_{\max}$. Damit ergibt sich

$$T = 4\sqrt{\frac{l}{2g}} \int_0^{\varphi_{\max}} \frac{d\psi}{\sqrt{\cos\psi - \cos\varphi_{\max}}} \ . \tag{3.11}$$

Der Integrand divergiert, sobald $\psi \to \varphi_{\max}$. Das Integral ist ein sog. elliptisches Integral ohne geschlossene analytische Lösung. In einem der folgenden Kapitel wird die numerische Integration solcher Funktionen behandelt werden. Die Lösungen der elliptischen Integrale sind für die verschiedenen Gattungen tabelliert. Gleichung (3.11) läßt sich durch Variablensubstitution $\sin\psi^* = \frac{\sin(\psi/2)}{\sin(\varphi_{\max}/2)}$ auf ein elliptisches Integral erster Gattung in der Legendreschen Normalform zurückführen:

$$F(k, \varphi) = \int_0^{\varphi} \frac{d\psi}{\sqrt{1 - k^2 \sin^2\psi}} \ , \tag{3.12}$$

wobei $k = \sin(\varphi_{\max}/2)$ ist. Ein weiteres Beispiel ist die Normalform der 2.Gattung

$$E(k,\varphi) = \int_0^\varphi d\psi \sqrt{1 - k^2 \sin^2 \psi} \tag{3.13}$$

Für die numerische Lösung der Pendelbewegung werden wir die Energieerhaltungsgleichung (3.6) umformen, um wieder die Gleichung eines harmonischen Oszillators zu erhalten. Durch Ableitung von Glg. (3.6) ergibt sich

$$\dot{E} = 0 = ml^2 \dot{\varphi}\ddot{\varphi} + mgl\dot{\varphi}\sin\varphi \tag{3.14}$$

und nach Umformung erhält man die nichtlineare Gleichung:

$$\ddot{\varphi} = -\frac{g}{l}\sin\varphi \ . \tag{3.15}$$

In der Nährung für kleine $\varphi \ll 1$ ist $\sin\ \varphi \approx \varphi$ und man erhält die lineare Differentialgleichung

$$\ddot{\varphi} = -\frac{g}{l}\varphi \ , \tag{3.16}$$

die mathematisch identisch zu Gleichung (3.1) ist. Diese kann durch einfaches Umbenennen der Parameter wieder numerisch mit den Verfahren aus dem letzten Abscnitt gelöst werden. Die Anfangsbedingungen seien z.B. $\varphi(0) = \varphi_{\max}$ und $\dot{\varphi}(0) = 0$.

3.2.1 Die Verlet-Methode

Ein verbessertes Verfahren, zur Lösung von Gleichung (3.15) ist die Verlet-Methode. Sie ist allerdings nur gültig, wenn die Kräfte unabhängig von der Geschwindigkeit sind. Zur Herleitung schreibt man die Taylor-Entwicklung von $\varphi(t+\Delta t)$ und $\varphi(t-\Delta t)$ auf:

$$\varphi(t+\Delta t) = \varphi(t) + \Delta t\omega(t) + \frac{\Delta t^2}{2}a(t) + \cdots \tag{3.17}$$

$$\varphi(t-\Delta t) = \varphi(t) - \Delta t\omega(t) + \frac{\Delta t^2}{2}a(t) + \cdots \tag{3.18}$$

wobei $a(t) = \dot{\omega}(t) = -\frac{g}{l}\sin\varphi$ verwendet wurde. Addiert man die beiden Gleichungen (3.17) und (3.18) ergibt sich

$$\varphi(t+\Delta t) = 2\varphi(t) - \varphi(t-\Delta t) + \Delta t^2 a(t) + \cdots \tag{3.19}$$

Der Vorteil liegt darin, daß durch die Addition die ungeraden Potenzen von Δt wegfallen und damit der Fehler auf die Größenordnung $O(\Delta t^4)$ schrumpft, anstatt wie zuvor bei $O(\Delta t^2)$ zu liegen. Für die Formel benötigt man nun zwei Werte der $\varphi(t)$–Funktion, den Anfangswert $\varphi(0) = \varphi_0$ und z.B. den Wert $\varphi(-\Delta t)$, welcher sich wiederum durch den Differenzenquotienten $\dot{\varphi}(0) = \frac{\varphi_0 - \varphi(-\Delta t)}{\Delta t}$ näherungsweise errechnet zu:

$$\varphi(-\Delta t) = \varphi_0 - \Delta t\dot{\varphi}_0 \ . \tag{3.20}$$

Wir haben wieder das übliche Berechnungsprogramm für die numerische Lösung:

```
   const int nx=10001;   // fieldlength
   double p[nx], t;

// parameter definition
   double g, l, m;     // g=gravity, l=length of pendulum, m=mass
   double p0, dp0;     // p0=initial angle, dp0=initial angular velocity
   double t_max, dt;   // t_max=total time, dt=time step

// read parameters
   cout << "Gravitation    ";  cin >> g;
   cout << "Pendellaenge   ";  cin >> l;
   cout << "Masse          ";  cin >> m;
   cout << "Anf.Auslenkung ";  cin >> p0;
   cout << "Anf.Geschw.    ";  cin >> dp0;
   cout << "Gesamtzeit     ";  cin >> t_max;
   cout << "Zeitintervall  ";  cin >> dt;
   double c=-(g/l)*dt*dt;

// make sure that the number of intervals is smaller than nx
   if( t_max/dt >= nx) {
      cout << "ERROR: dt is too small, set to: " << t_max/nx << '\n';
      return -1;
   }

// setup initial conditions
   p[0]=p0-dt*dp0;
   p[1]=p0;
// loop from t=0 to t=t_max
   for( int i=2; i<t_max/dt; i++ ) {
        p[i]=2.*p[i-1]-p[i-2]+c*sin(p[i-1]);
   }

// output to file
   ofstream outfile("verletp.dat");
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) {
      outfile << t << ' ' << p[i+1] << '\n';
      t=t+dt;
   }
   outfile.close();
```

Dabei wird der Anfangsindex auf `i=2` verschoben, um negative Indizes zu vermeiden. Mit den Parametern $g = 10$, $l = m = 1$, $\varphi_0 = 0.1$, $\dot{\varphi}_0 = 0$, $t_{\max} = 7.5$ und $\Delta t = 0.01$, erhält man bereits das Bild einer harmonischen Schwingung. Setzt man die Anfangsbedingung höher, z.B. $\varphi_0 = 3$, so ergibt sich ein deutlich anderes Schwingverhalten, das nicht mehr durch einen einfachen Cosinus genähert werden kann, wie in Abb. 3.5 zu sehen.

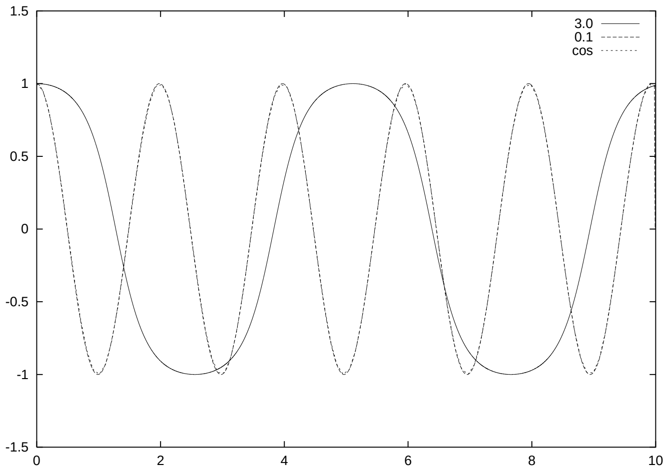


Abbildung 3.5: Ausdruck von $\varphi(t)/\varphi_0$ für $g = 10$, $l = 1$, $m = 1$, $\dot{\varphi}_0 = 0$, $t_{\max} = 10$ und $\Delta t = 0.01$, berechnet mit dem Verlet-Verfahren für $\varphi_0 = 0.1$ und $\varphi_0 = 3$, verglichen mit $\cos(\sqrt{10}t)$.

3.2.2 Das allgemeine Pendel

Zu dem speziellen Pendel aus dem vorherigen Kapitel fügen wir noch eine zur Geschwindigkeit proportionale Reibungskraft $-q\dot{\varphi}$ und eine treibende Kraft $F = F_0 \sin(\Omega t)$ hinzu, wobei die Frequenz Ω nicht gleich der Eigenfrequenz des Pendels sein darf ($\Omega \neq \omega_0 = \sqrt{\frac{g}{l}}$). Somit existieren die zusätzlichen Parameter q, F_0 und Ω. Das dazugehörige Bewegungsgesetz lautet

$$\ddot{\varphi} = -\frac{g}{l}\sin\varphi - q\dot{\varphi} + F_0 \sin\Omega t \tag{3.21}$$

Zur Lösung dieser Gleichung ist das Verlet-Verfahren nicht mehr geeignet, da eine Kraft von der Geschwindigkeit abhängt. Deswegen wird ein neues Verfahren zur numerischen Lösung von Gleichungen vorgestellt.

3.2.3 Das Runge-Kutta-Verfahren

Als Beispiel wird jetzt von einer Gleichung $\dot{\mathbf{x}} = f[\mathbf{x}, t]$ ausgegangen, wobei $\mathbf{x}$, einen Vektor, bestehend aus verschiedenen Größen, darstellen kann. Beim Euler-Verfahren wurde der

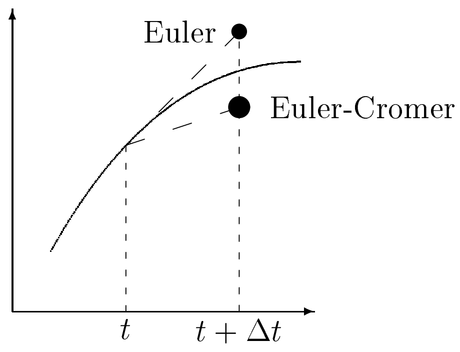


Abbildung 3.6: Fehler beim Euler- / Euler-Cromer-Verfahren

nächste Kurvenstützpunkt mit Hilfe der Steigung des alten Stützpunktes, beim Euler-Cromer-Verfahren mit der Steigung des neuen Stützpunktes berechnet. Beide Verfahren besitzen einen relativ großen Fehler (siehe Abb. 3.6).

Um die Genauigkeit zu verbessern wird beim Runge-Kutta-Verfahren die Steigung in der Mitte des Intervalls bestimmt. Der Funktionswert $\bar{x}$ in der Intervallsmitte wird durch eine Taylorentwicklung berechnet

$$\bar{x} = x\left(t + \frac{\Delta t}{2}\right) = x(t) + \frac{1}{2}\Delta t \; f[x(t), t]. \tag{3.22}$$

Dieser Zwischenwert wird dann in die allgemeine Formel eingesetzt.

$$x(t + \Delta t) = x(t) + \Delta t \; f\left[x\left(t + \frac{\Delta t}{2}\right), t + \frac{\Delta t}{2}\right] \tag{3.23}$$

Für die Anwendung auf das Pendel benötigen wir wieder die beiden Gleichungen

$$\dot{\omega} = f[\varphi, \omega, t] = -\frac{g}{l}\sin\varphi - q\omega + F_0 \sin\Omega t \tag{3.24}$$

$$\dot{\varphi} = \omega \tag{3.25}$$

Für das Runge-Kutta-Verfahren (R-K) werden dann die Zwischenwerte $\bar{\omega}$ und $\bar{\varphi}$ berechnet.

$$\bar{\omega} = \omega(t) + \frac{\Delta t}{2} \; f[\varphi(t), \omega(t), t] \tag{3.26}$$

$$\bar{\varphi} = \varphi(t) + \frac{\Delta t}{2} \; \omega(t) \tag{3.27}$$

Die Lösungen ergeben sich dann durch das R-K-Verfahren.

$$\varphi(t + \Delta t) = \varphi(t) + \Delta t \; \bar{\omega} \tag{3.28}$$

$$\omega(t + \Delta t) = \omega(t) + \Delta t \; f\left[\bar{\varphi}, \bar{\omega}, t + \frac{\Delta t}{2}\right] \tag{3.29}$$

Im folgenden Programmsegment wird zuerst $\bar{\omega}$ berechnet und in einer Zwischenvariablen `obar` gespeichert. Damit wird der neue Winkel $\varphi(t + \Delta t)$ berechnet, $\bar{\varphi}$ wird in einer Zwischenvariablen `pquer` abgelegt und zur die Berechnung von ω benutzt. Die Zeit t wird zweimal um $\frac{\Delta t}{2}$ erhöht und somit bei jedem Durchlauf um Δt.

```
   const int nx=10001;  // fieldlength
   double p[nx], dp[nx], t;

// parameter definition
   double g, l;        // g=gravity, l=length of pendulum
   double q, om, am;   // q=friction, om=driving frequency, am=amplitude
   double p0, dp0;     // p0=initial angle, dp0=initial angular velocity
   double t_max, dt;   // t_max=total time, dt=time step
   double fp, obar, pbar;

// read parameters
   cout << "Gravitation    ";  cin >> g;
   cout << "Pendellaenge   ";  cin >> l;
   cout << "Reibung        ";  cin >> q;
   cout << "Anreg.Frequenz ";  cin >> om;
   cout << "Anreg.Amplitude";  cin >> am;
   cout << "Anf.Auslenkung ";  cin >> p0;
   cout << "Anf.Geschw.    ";  cin >> dp0;
   cout << "Gesamtzeit     ";  cin >> t_max;
   cout << "Zeitintervall  ";  cin >> dt;
   double c=g/l;
   double dt2=dt/2.;

// make sure that the number of intervals is smaller than nx
   if( t_max/dt >= nx) {
      cout << "ERROR: dt is too small, set to: " << t_max/nx << '\n';
      return -1;
   }

// setup initial conditions
   p[0]=p0;
   dp[0]=dp0;
// loop from t=0 to t=t_max
   t=0.0;
   for( int i=1; i<t_max/dt; i++ ) {
        fp=-c*sin(p[i-1])-q*dp[i-1]+am*sin(om*t);
        obar=dp[i-1]+dt2*fp;
        p[i]=p[i-1]+dt*obar;
```

```
         t=t+dt2;
         pbar=p[i-1]+dt2*dp[i-1];
         fp=-c*sin(pbar)-q*obar+am*sin(om*t);
         dp[i]=dp[i-1]+dt*fp;

         t=t+dt2;
      }

// output to file
   ofstream outfile("ruku.dat");
   t=0.0;
   for( int i=0; i<=t_max/dt; i++ ) {
      outfile << t << ' ' << p[i+1] << ' ' << dp[i+1] << '\n';
      t=t+dt;
   }
   outfile.close();
}
```

3.2.4 Chaos

Um die Ergebnisse des Progamms besser untersuchen zu können, legen wir folgende Werte fest: $g = l = 9.81$, $\Omega = 2/3$, $q = 0.5$. Die einzigen jetzt noch frei wählbaren Variablen sind: F_0 und φ_0. In Abbildung 3.7 sehen wir den zeitlichen Verlauf des Auslenkwinkels φ für verschiedene Zeitschritte Δt und verschiedene Amplituden F_0 zwischen 0 und 1.3.

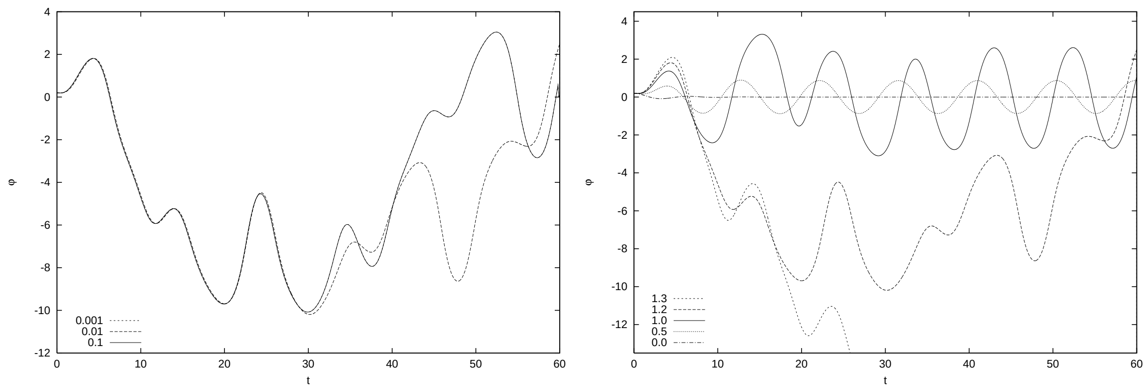
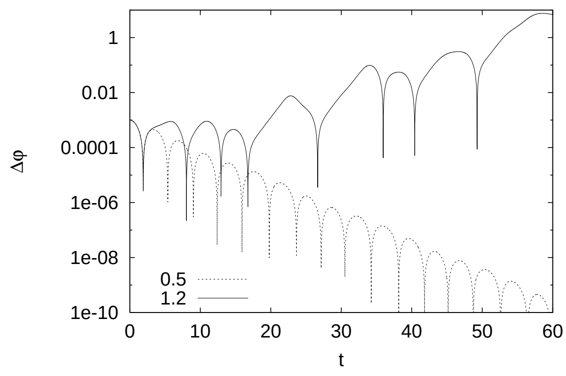


Abbildung 3.7: Links: Pendelausschlag φ als Funktion der Zeit t für verschiedene Δt. Rechts: φ als Funktion von t für verschiedene F_0.

Ist die Genauigkeit $\Delta t \leq 0.01$ so fallen die Kurven aufeinander, bei größerem Δt stellt man starke Abweichungen fest. Ohne externe Anregung $F_0 = 0$, kommt das Pendel aufgrund der Reibungskraft q schon nach kurzer Zeit zum Stillstand. Für einen kleinen Wert, zum Beispiel $F_0 = 0.5$, entsteht das gleichmäßige Schwingungsbild einer periodischen

Bewegung. Die Frequenz entspricht jedoch nicht der Eigenfrequenz des Pendels, sondern entspricht ungefähr der Frequenz der antreibenden Kraft. Wird die Antreibkraft weiter erhöht, dann bewegt sich das Pendel nicht mehr gleichmäßig. Die Ausschläge lassen sich nicht mehr vorraussagen, das Verhalten des Pendels wird *chaotisch*. Es ist bemerkenswert, daß trotz des durch die Rechnung gegebenen Determinismus das Verhalten des Pendels chaotisch ist.

Als nächstes halten wir den Wert von F_0 fest und verändern den anfänglichen Auslenkwinkel φ_0. Im ersten Fall sei der Anfangsauslenkwinkel $\varphi_{01} = 0.2$, im zweiten Fall $\varphi_{02} = 0.201$. Der Unterschied beträgt also nur 0.5 Prozent. Die Differenz ("damage spreading" genannt) dieser beiden Auslenkwinkel $\Delta\varphi(t) = \varphi_2(t) - \varphi_1(t)$ wird nun im zeitlichen Verlauf untersucht. Die Grafiken der Abbildung 3.8 sind halblogarithmische Darstellungen, d.h. die x-Achse hat einen normalen, linearen Maßstab, während die y-Achse logarithmisch ist. Der Abstand zwischen den Auslenkungen wurde einmal mit der Euler-Cromer Methode (links) und dann mit dem Runge-Kutta Verfahren (rechts) berechnet. Die jeweils obere Kurve zeigt den chaotischen Fall: Der Fehler vergrößert sich, im Mittel auf einer steigenden Geraden. Eine Linie mit positiver Steigung bedeutet in der halblogarithmischen Darstellung, daß die Funktion exponentiell wächst, der Abstand läßt sich also durch $\Delta\varphi \approx e^{\lambda t}$, mit $\lambda > 0$ nähern. Im periodischen Fall (untere) verringert sich der Abstand in der Darstellung linear, also wieder exponentiell, aber diesmal mit negativer Steigung $\lambda < 0$. λ wird als der Lyapunovexponent bezeichnet. Bei $\lambda = 0$ ist der Übergang vom Periodischen ins Chaotische mit $\lambda > 0$.



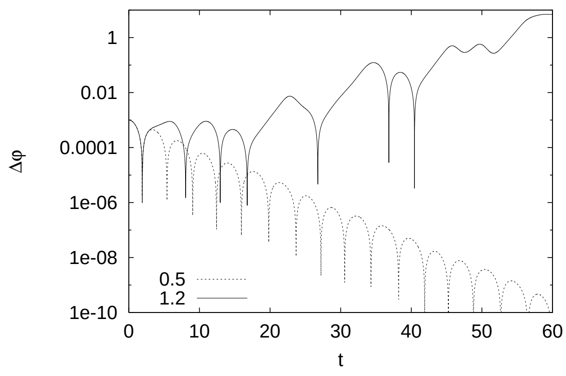


Abbildung 3.8: Damage Spreading: Der Abstand zwischen den Anfangsauslenkungen $\varphi_{01} = 0.2$ und $\varphi_{01} = 0.201$ wurde links mit Euler-Cromer, rechts mit Runge-Kutta berechnet.

3.2.5 Der Poincaréschnitt

Eine andere Darstellung der Pendelbewegung ist durch die Abbildung in einem sog. *Phasendiagramm* möglich. Ein Phasendiagramm ist ein Geschwindigkeit-Ort-Diagramm, in das die Winkelgeschwindigkeit ω über den Auslenkwinkel φ aufgetragen wird. Für den harmonischen Oszillator entsteht dann als Bild eine Ellipse. Diese geschlossene Kurve zeigt die Energieerhaltung beim harmonischen Oszillator. Für das allgemeine Pendel entsteht im periodischen Fall nach der Einschwingzeit ebenfalls eine Ellipse (linke Grafik, Abbildung 3.9). Im chaotischen Fall weist das Phasendiagramm eine endlose Anzahl unterschiedlicher Linien auf, die irgendwann (fast) das ganze Diagramm überdeckt haben (rechte Grafik).

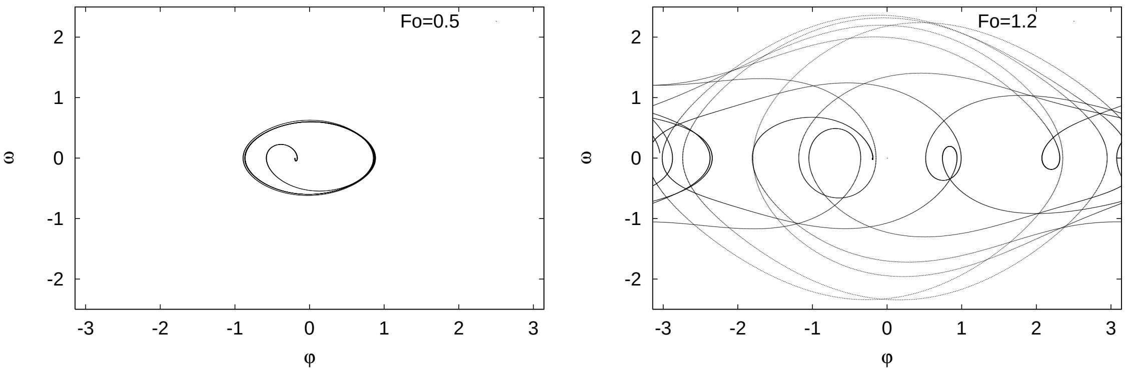


Abbildung 3.9: Phasendiagramme des allgemeinen Pendels, links für $F_0 = 0.5$, rechts für $F_0 = 1.2$.

Die Untersuchung des Bildes im chaotischen Fall ist sehr schwierig. Deswegen wird hier meist eine stroboskopische Analyse durchgeführt: Es werden nur Punkte in das Diagramm gezeichnet, die zu der gleichen Phase der angelegten periodischen Kraft vorkommen, beispielsweise für $\Omega t = 0, 2\pi, 4\pi$ etc.. Diese stroboskopischen "Schnitte" werden *Poincaréschnitte* genannt. Für den harmonischen Oszillator entsteht in dieser Darstellungsart im Diagramm nur ein einziger Punkt. Das entstehende Bild wird *Attraktor* genannt. Dieser erste Fall ist der punktförmige Attraktor. Ist die Bewegung chaotisch, dann entsteht eine Menge von Punkten. Dieses Bild heißt dann seltsamer oder fraktaler Attraktor. Für unser Pendel ist dies in der Abbildung 3.10 links zu sehen. Als drittes gibt es noch die geschlossenen Attraktoren: ein geschlossener Linienzug im Poincaréschnitt. Für ein Doppelpendel (ein Pendel, das an der Masse eines anderen Pendels aufgehängt ist) ensteht ein Diagramm, bei dem gleichzeitig zwei verschiedene Attraktoren, d.h. zwei verschiedene Fälle im gleichen Phasenraum, gezeigt werden können, ein geschlossener und ein fraktaler Attraktor.

3.2.6 Übergang periodisch – chaotisch

Betrachtet man das Pendel für noch größere Antreibkräfte, entsteht wieder periodisches Verhalten. Abbildung 3.11 zeigt das Pendelverhalten für die Antreibkräfte $F_0 = 1.35$, $F_0 = 1.44$ und $F_0 = 1.465$. Es fällt auf, daß die Perioden für die letzten beiden Werte doppelt, bzw. viermal so groß sind. Die Periode wird also jeweils verdoppelt. Dieses Verhalten wird Bifurkation genannt.

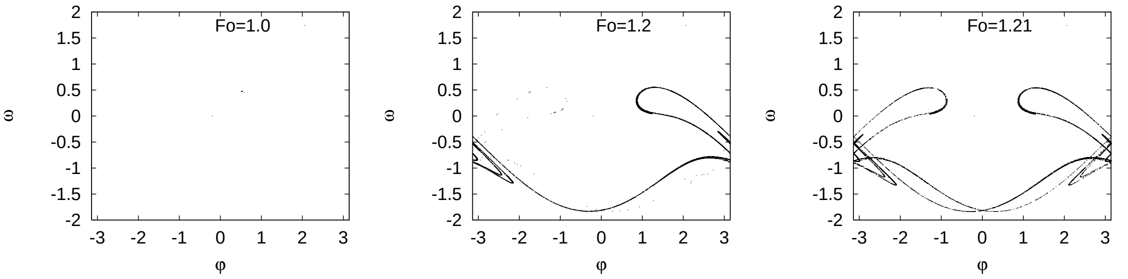


Abbildung 3.10: Poincaréschnitte für das allg. Pendel

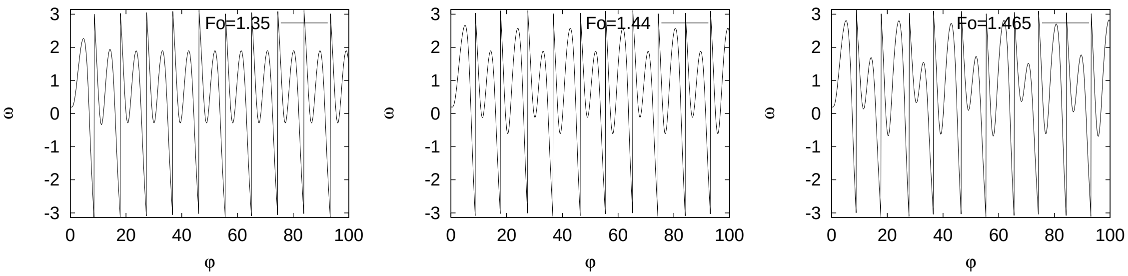


Abbildung 3.11: Unterschiedliche Perioden bei verschieden starken Antreibkräften

Das Bifurkationsdiagramm aus Abbildung 3.12 zeigt die Punkte des Poincaréschnittes in Abhängigkeit der Antreibfrequenz F_0. Die Bifurkationen sind in diesem Fall an den Stellen $F_0 \approx 1.43$ und $F_0 \approx 1.46$. Ab dem Wert $F_0 \approx 1.47$ beginnt das Chaos. Dieser Übergang zum Chaos wird auch Periodenverdopplungs– oder Kaskadenszenario genannt.

Die Längen der Perioden sind im Poincaréschnitt durch die Anzahl der Punkte gegeben. Eine zweifache Periode entspricht so z.B. einem Attraktor aus zwei Punkten. Sei jetzt f_n der Wert, an dem eine Bifurkation zu 2^n besteht, d.h. das Diagramm sich in 2^n Linien aufspaltet. Dann ist die Funktion

$$\delta_n = \frac{f_{n-1} - f_n}{f_n - f_{n+1}} \tag{3.30}$$

das Verhältnis der Abstände zwischen zwei aufeinanderfolgenden Bifurkationen. δ_n geht für große n gegen den Wert 4.669. Dieser Wert ist universell, d.h. für alle Zweier-Kaskadenszenarios derselbe. Man nennt ihn Feigenbaumkonstante.

3.3 Himmelsmechanik

Im Folgenden betrachten wir die Berechnung der Planetenbewegung, zunächst einmal der Erde, um die Sonne. Entscheidend hierbei ist die Gravitationskraft zwischen der Masse

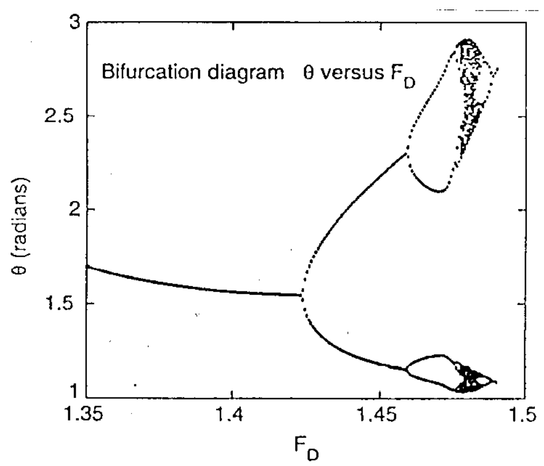


Abbildung 3.12: Bifurkationsdiagramm des Pendels

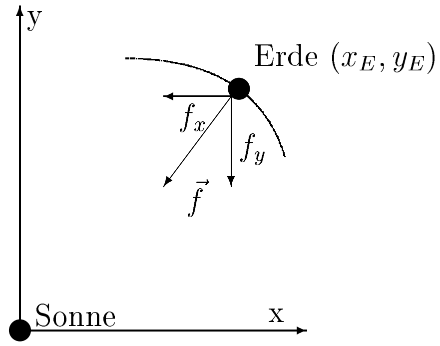


Abbildung 3.13: System Erde–Sonne

der Sonne (m_S) und der Masse der Erde (m_E). Die Formel für diese Kraft lautet :

$$\vec{f} = -\frac{Gm_Sm_E}{r^2} * \frac{\vec{r}}{r} \tag{3.31}$$

Der Faktor G ist die Gravitationskonstante und der Vektor $\vec{r} = (x,\, y)$ ist der Abstandsvektor zwischen den Massenschwerpunkten von Sonne und Erde. Zur Vereinfachung betrachten wir die Bewegung nur im Zweidimensionalen. Für die Berechnung benutzen wir die Formel in der Komponentenschreibweise für die x-Richtung f_x und in die y-Richtung f_y mit

$$f_x = -\frac{Gm_Sm_Ex}{r^3}\ ,\ f_y = -\frac{Gm_Sm_Ey}{r^3} \text{ und } r = \sqrt{x^2+y^2} \tag{3.32}$$

Um geschicktere und angenehmere Zahlen zu bekommen, verwenden wir hier nicht das MKS-System (Meter–Kilo–Sekunde), sondern die Astronomische Einheit (1 AE= $1.5 * 10^{11}$ m≈ Abstand Erde – Sonne) für Längen und das Jahr (1 a) für die Zeit. Die Bahngeschwindigkeit bei der Kreisbewegung ist somit $v_0 = 2\pi * \frac{1\mathrm{AE}}{1\mathrm{a}}$. Da die Gravitationskraft betragsmäßig gleich der Zentrifugalkraft ist, haben wir

$$f = \frac{m_E v_0^2}{r} = \frac{G m_S m_E}{r^2} \Rightarrow G m_S = v_0^2 r = 4\pi^2 \frac{\mathrm{AE}^3}{\mathrm{a}^2} \tag{3.33}$$

Für die numerische Berechnung verwenden wir wieder die Newtonsche Bewegungsgleichung

$$\ddot{x} = \frac{f_x}{m_E} = -\frac{4\pi^2 x}{r^3} \text{ und } \ddot{y} = \frac{f_y}{m_E} = -\frac{4\pi^2 y}{r^3} \tag{3.34}$$

Wir benutzen in diesem Fall wieder die Verlet-Methode für unser Programm und lösen für die numerische Rechnung die beiden Gleichungen

$$x(t+\Delta t) = -x(t-\Delta t) + 2x(t) - 4\pi^2 x(t) r_3(t) \Delta t^2 \tag{3.35}$$

$$y(t+\Delta t) = -y(t-\Delta t) + 2y(t) - 4\pi^2 y(t) r_3(t) \Delta t^2 \tag{3.36}$$

Dabei ist die Abkürzung $r_3(t) = \left(\sqrt{x(t)^2 + y(t)^2}\right)^{-3}$ verwendet worden.

```
int main()
{
   double p0x, p1x, p2x, dp0x, dp1x, dp2x, t;
   double p0y, p1y, p2y, dp0y, dp1y, dp2y, r3;
   double alpha, t_max, dt;

// read parameters
   cout << "Anf.x  ";  cin >> p1x;
   cout << "Anf.y  ";  cin >> p1y;
   cout << "Anf.vx ";  cin >> dp1x;
   cout << "Anf.vy ";  cin >> dp1y;
   cout << "Alpha  ";  cin >> alpha;
   cout << "Gesamtzeit     ";  cin >> t_max;
   cout << "Zeitintervall  ";  cin >> dt;

   const double beta=(alpha)/2.;
   const double dt2=dt*dt;
   const double pi24=4.*M_PI*M_PI;

// setup initial conditions
   p0x=p1x-dt*dp1x;
   p0y=p1y-dt*dp1y;
   dp0x=dp1x;
   dp0y=dp1y;

// loop from t=0 to t=t_max
```

```
    t=0.0;
    ofstream outfile("orbit.dat");
    for( int i=0; i<t_max/dt; i++ ) {
         r3=1./pow(p1x*p1x+p1y*p1y,beta);
         p2x=2.*p1x-p0x-pi24*dt2*r3*p1x;  // Verlet algorithm
         p2y=2.*p1y-p0y-pi24*dt2*r3*p1y;
         outfile << t << ' ' << p1x << ' ' << p1y << '\n';
         t=t+dt;
         p0x=p1x;                         // change new->old
         p0y=p1y;
         p1x=p2x;
         p1y=p2y;
    }
    outfile.close();
}
```

Für ideale Anfangsbedingungen, das wäre $x_0 = 1$, $y_0 = 0$, $vx_0 = 0$ und $vy_0 = 2\pi$, erhält man für die Erdbewegung eine Kreisbahn, siehe Abbildung 3.14 (links). Verändert man die Anfangsgeschwindigkeit auf $vy_0 = 5$, so ergibt sich eine elliptische Bahn (rechts).

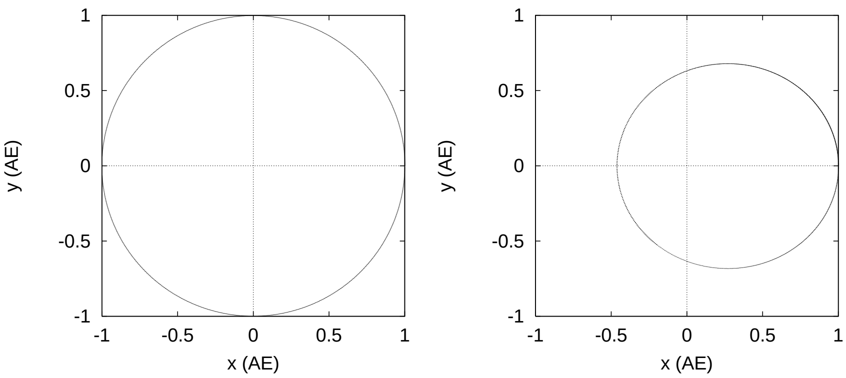


Abbildung 3.14: Bahnkurven der Erde um die Sonne für verschiedene Anfangsgeschwindigkeiten $vy_0 = 2\pi$ (links) und $vy_0 = 5$ (rechts).

Eine weitere Variationsmöglichkeit besteht in der Änderung des Gravitationsgesetzes. Anstatt die Gravitationskraft proportional zu $1/r^2$ abfallen zu lassen, kann man sie auch proportional zu $1/r^{\alpha-1}$ machen. Allgemein wird im obigen Programm `r=1/sqrt(pow(r, alpha))` verwendet, wobei `pow(r, alpha)` der Potenzierung r^α entspricht. Abbildung 3.15 zeigt vier verschiedene Beispiele mit Werten für β zwischen 3.00 und 2.01, wobei $\beta = \alpha - 1$. Die Rotation der Ellipse bei den Beispielen für $\beta \approx 2$ nennt man Präzesion.

Für $\beta = 3.00$ fällt auf, daß die Erde, nachdem sie in die Nähe der Sonne "geflogen" ist, sich mit hoher Geschwindigkeit aus dem System entfernt. Dies ist allerdings ein numerisches Problem. Für diesen Effekt ist die Wahl eines zu großen Δt verantwortlich, daß bei großen

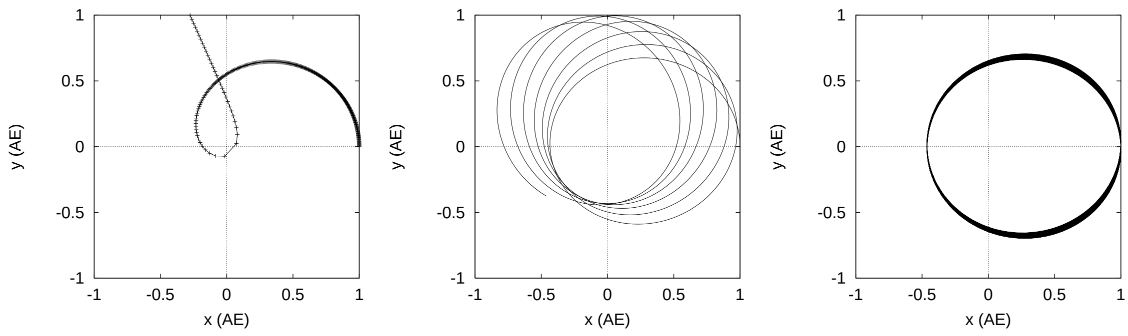


Abbildung 3.15: Bahnkurven bei versch. Gravitationsgesetzen $\beta = 3$ (links), $\beta = 2.1$ (mitte) und $\beta = 2.01$ (rechts).

Geschwindigkeitsänderungen kleiner sein sollte. Allerdings würde bei einem kleineren Δt die Simulation zu lange dauern. Man benötigt also ein automatisches Verfahren, das die Größe von Δt regelt. Eine Möglichkeit ist es, die Änderung der Energie für verschiedene Δt zu vergleichen und den Zeitschritt dementsprechend zu verändern (adaptives Verfahren). Dazu bietet sich z.B. das Runge-Kutta Verfahren an, da dort sowieso Werte zur Zeit $t + \Delta t/2$ berechnet werden. Die Energie der Erde ist

$$E_{Erde} = -\frac{Gm_S m_E}{r} + \frac{m_E}{2}v^2 \tag{3.37}$$

Die Energie ergibt sich also aus den Koordinaten x und y für $r = \sqrt{x^2 + y^2}$ direkt und für $\vec{v} = \Delta\vec{r}/\Delta t$ indirekt. Wir berechnen zuerst die Energie E_1 aus

$$x_1 = x(t + \Delta t) = f_{\Delta t}(x(t), y(t)) \tag{3.38}$$
$$y_1 = y(t + \Delta t) = g_{\Delta t}(x(t), y(t)) \tag{3.39}$$

und die Energie E_2 mit halb so großem Δt durch

$$x_2 = f_{\frac{\Delta t}{2}}(x(t + \frac{\Delta t}{2}), y(t + \frac{\Delta t}{2})) \tag{3.40}$$
$$y_2 = g_{\frac{\Delta t}{2}}(x(t + \frac{\Delta t}{2}), y(t + \frac{\Delta t}{2})) \tag{3.41}$$

Die Funktionen f und g berechnen die neuen x- bzw. y-Werte für Δt oder $\Delta t/2$. Das entscheidende Kriterium ist die Differenz

$$\Delta E = |E_2 - E_1| \tag{3.42}$$

Ist ΔE groß, so ist Δt zu klein gewählt, ist ΔE sehr klein, so kann Δt größer gewählt werden. Im Programm könnte z.B. bei $\Delta E > 10^{-5} \rightarrow \Delta t' = \Delta t/2$ und bei $\Delta E < 10^{-7} \rightarrow \Delta t' = \Delta t * 2$ gewählt werden.

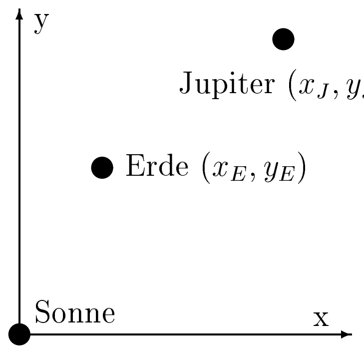


Abbildung 3.16: System Erde–Jupiter

3.3.1 Erde und Jupiter

Kommt zu den Körpern Sonne und Erde noch ein dritter (beispielsweise der Jupiter) hinzu, so ist dieses Problem nicht mehr analytisch lösbar. Die Keplerschen Gesetze zur Planetenbewegung gelten nicht mehr. Allgemein ist das beim N-Körperproblem ($N > 2$) so. Numerisch läßt sich dieses Problem immer noch lösen. Wir benötigen als erstes das Gravitationsgesetz für Erde-Jupiter:

$$f_{EJ} = -\frac{Gm_E m_J}{r_{EJ}^2} \text{ mit } r_{EJ} = \sqrt{(x_E - x_J)^2 + (y_E - y_J)^2} \tag{3.43}$$

Wieder stellen wir für die Erdbewegung (und auch für die Jupiterbewegung) eine komponentenweise Gleichung auf, die analog für y gilt.

$$\ddot{x}_E = -\frac{4\pi^2 x_E}{r_E^3} - \frac{Gm_J(x_E - x_J)}{r_{EJ}^3} \tag{3.44}$$

$$\ddot{x}_J = -\frac{4\pi^2 x_J}{r_E^3} - \frac{Gm_E(x_J - x_E)}{r_{EJ}^3} \tag{3.45}$$

Der rechte Ausdruck ist der Einfluß vom dritten Körper, also jeweils des Jupiters, bzw. der Erde. Für die Rechnung benötigen wir jetzt auch die Massen der einzelnen Körper.

$$\begin{aligned} m_E &= 6 * 10^{24}\text{kg} \\ m_S &= 2 * 10^{30}\text{kg} \\ m_J &= 2 * 10^{27}\text{kg} * M_J \end{aligned}$$

Den Faktor M_J benutzen wir später in der Simulation um auszuprobieren, wie sich unser System verhalten würde, wenn die Masse des Jupiters geändert würde. Der Radius der Jupiterbahn ist $r_J = 5.2\,\text{AE}$ und seine Bahngeschwindigkeit ist $v = 2.76\,\text{AE/a}$. Man kann nun die Konstanten

$$Gm_E = 12 * \pi^2 * 10^{-6} \tag{3.46}$$

$$Gm_J = 4 * \pi^2 * 10^{-3} * M_J \tag{3.47}$$

berechnen. Als numerisches Verfahren verwenden wir wieder die Verlet-Methode. Die Gleichungen

$$\begin{aligned} x_E(t+\Delta t) &= -x_E(t-\Delta t) + 2x_E(t) - \\ &\quad -\Delta t^2 4\pi^2 \left(\frac{x_E(t)}{r_E^3(t)} + \frac{10^{-3}(x_E(t) - x_J(t)) * M_J}{r_{EJ}^3(t)} \right) \qquad (3.48) \\ x_J(t+\Delta t) &= -x_J(t-\Delta t) + 2x_J(t) - \\ &\quad -\Delta t^2 4\pi^2 \left(\frac{x_J(t)}{r_J^3(t)} + \frac{3 * 10^{-6}(x_J(t) - x_E(t))}{r_{EJ}^3(t)} \right) \qquad (3.49) \end{aligned}$$

gelten für die x-Komponente, die y-Komponente ist analog dazu.

```
   double p0x, p1x, p2x, dp1x, t;
   double p0y, p1y, p2y, dp1y, r3e;
   double q0x, q1x, q2x, dq1x, r3j;
   double q0y, q1y, q2y, dq1y, r3ej;
   double alpha, massj, t_max, dt;
// ...
// loop from t=0 to t=t_max
   t=0.0;
   ofstream outfile("orbit.dat");
   for( int i=0; i<t_max/dt; i++ ) {
        r3e=1./pow(p1x*p1x+p1y*p1y,beta);
        r3j=1./pow(q1x*q1x+q1y*q1y,beta);
        r3ej=1./pow((p1x-q1x)*(p1x-q1x)+(p1y-q1y)*(p1y-q1y),beta);
// Verlet sums
        p2x=2.*p1x-p0x-pi24*dt2*
              (r3e*p1x+r3ej*1.e-3*massj*(p1x-q1x));  // Earth
        p2y=2.*p1y-p0y-pi24*dt2*
              (r3e*p1y+r3ej*1.e-3*massj*(p1y-q1y));
        q2x=2.*q1x-q0x-pi24*dt2*
              (r3j*q1x-r3ej*3.e-6*(p1x-q1x));        // Jupiter
        q2y=2.*q1y-q0y-pi24*dt2*
              (r3j*q1y-r3ej*3.e-6*(p1y-q1y));

        outfile << t << ' ' << p1x << ' ' << p1y <<
                         ' ' << q1x << ' ' << q1y << '\n';
        t=t+dt;
        p0x=p1x;                        // change new->old
        p0y=p1y;
        p1x=p2x;
        p1y=p2y;
        q0x=q1x;                        // change new->old
        q0y=q1y;
```

```
        q1x=q2x;
        q1y=q2y;
    }
```

Abbildung 3.17 zeigt die Bahnen für verschiedene Massen des Jupiters. Gerade in der letzten Grafik ist deutlich zu sehen, daß bei diesem Drei–Körper–Problem die Erde, die deutlich weniger Masse besitzt, als der Jupiter, einen chaotischen Verlauf hat. Ebenso ist das allgemeine N–Körper–Problem chaotisch!

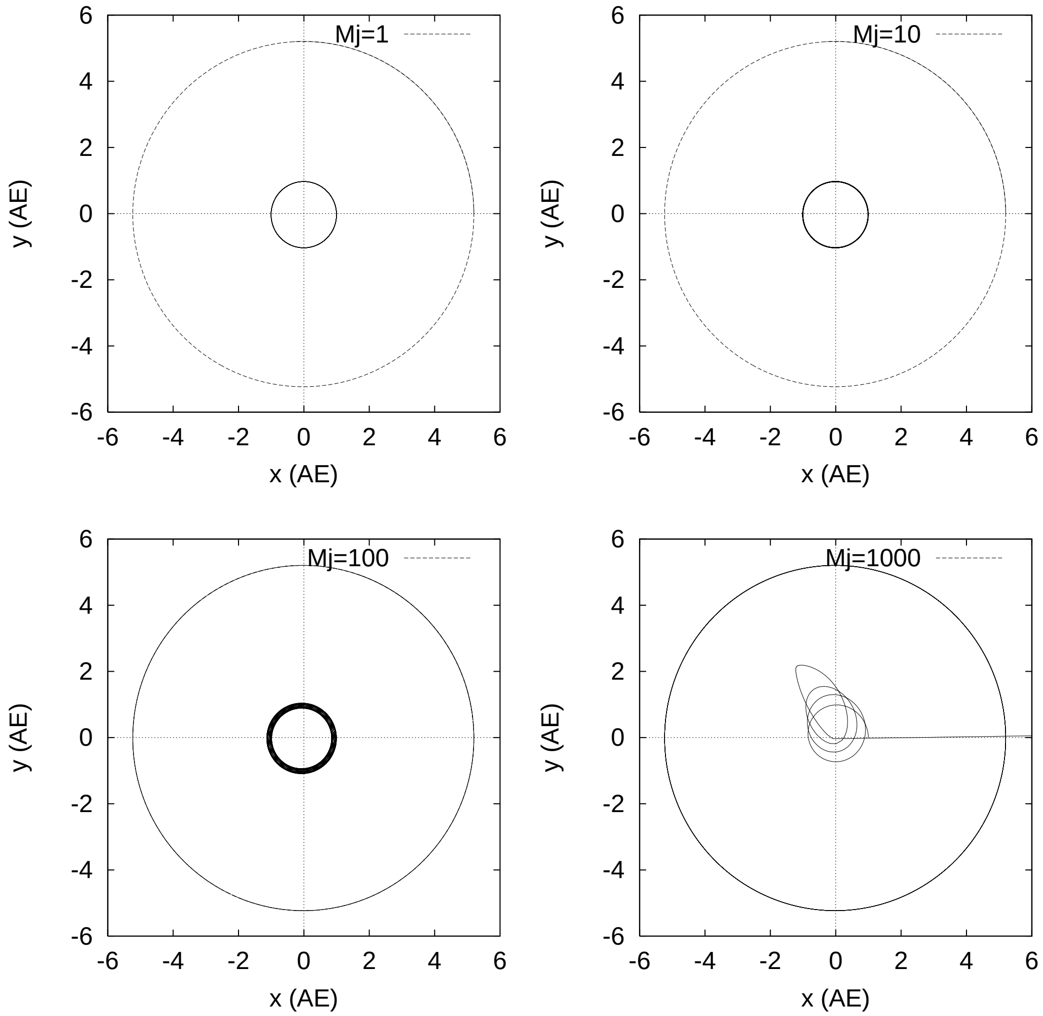


Abbildung 3.17: 2–Körper–Simulation mit verschiedenen Jupiter-Massen mit fixierter Sonne.

3.3.2 Das Drei–Körper–Problem

Bei einer Masse des Jupiter von $M_j = 1000$ ist allerdings zu bedenken, das die Annahme einer unbeweglichen Sonne falsch wird. Simuliert man das wirkliche 3–Körper–Problem, mit beweglicher Sonne, so umkreisen Sonne und Jupiter den gemeinsamen Schwerpunkt und die Erde wird nach kurzer Zeit aus dem System geschleudert. In Abb. 3.18 ist die Simulation für verschiedene Zeitschritte dargestellt. Eigentlich entspricht diese Simulation einem Doppelsternsystem mit zwei großen Sonnen und einem Planeten. Schon bei einem Zeitschritt von $\Delta t \leq 0.01$ ist die Lösung beinahe nicht mehr von der Lösung für $\Delta t \leq 0.001$ zu unterscheiden. Als Kontrolle kann man die Energie im Planetensystem berechnen. Die kinetische Energie im N–Körper–System ist

$$E_k = \sum_{i=1}^{N} \frac{1}{2} m_i v_i^2 \ , \tag{3.50}$$

und die potentielle Energie errechnet sich aus allen paarweisen Wechselwirkungen zu

$$E_p = \sum_{i=1}^{N} \sum_{j=i+1}^{N} 4\pi^2 m_i m_j \frac{1}{r_{ij}} \ , \tag{3.51}$$

wobei wie oben alle Längen in AE, alle Geschwindigkeiten in v_0, alle Zeiten in Jahren, und alle Massen in Sonnenmassen m_S gemessen werden. In Abb. 3.18 erkennt man, daß die Energieerhaltung nur für große Zeitschritte verletzt wird.

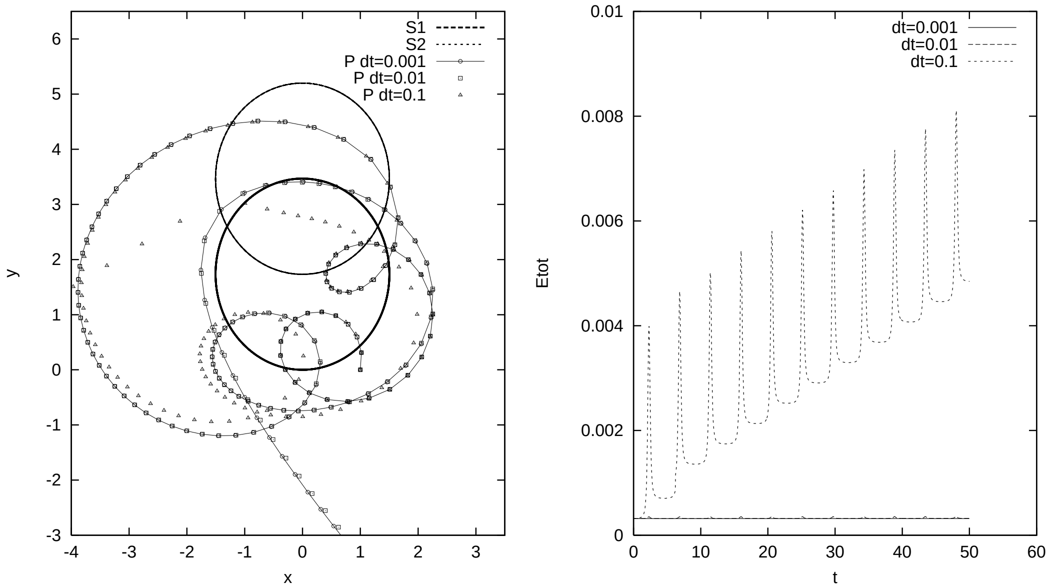


Abbildung 3.18: 3–Körper–Simulation mit Jupiter-Masse $M_j = 1000$ und beweglicher Sonne. Links: Trajektorien und rechts: Energie als Funktion der Zeit.

Die gleiche Methode, bei der jeder Körper mit jedem anderen wechselwirkt, verwendet man auch bei Simulation von Molekülen (Molekulardynamik), siehe Kapitel 3.4.

3.3.3 Perihelbewegung des Merkurs

Ein Beispiel für die Anwendung der Relaticitätstheorie in der Himmelsmechanik ist die Präzession des Perihels (der Umlaufbahn) des Merkurs, die aufgrund seiner Sonnennähe relativ groß ist. Die Bahn des Merkurs dreht sich um 566 Arksek/Jahrhundert (1 Arksek = (1/3600)°). Für eine volle Umdrehung benötigt die Bahn insgesamt 230000 Jahre! Wird die Präzession allerdings selbst mit dem Einfluß des Jupiters errechnet, kommt man nur auf 523 Arksek/Jahrhundert! Albert Einstein erhält durch seine Allgemeine Relativitätstheorie (Raum wird durch Masse gekrümmt) ein erweitertes Bewegungsgesetz. Für die Kraft zwischen Sonne und Merkur gilt nun:

$$f \approx \frac{G m_S m_M}{r_{SM}^2}\left(1 + \frac{\alpha}{r_{SM}^2}\right), \text{ mit } \alpha \approx 1.1 * 10^{-8}\mathrm{AE}^2 \tag{3.52}$$

Der Unterschied in der Berechnung durch diese Formel (und mit dieser Konstanten α) ist 43 Arksek/Jahrhundert, genau die Differenz zwischen dem vorher berechneten Wert und dem beobachteten. Dieses Ergebnis bescherte der Allgemeinen Relativitätstheorie den Durchbruch, da sie ein bis dahin unlösbares Problem der Physik korrekt erklärt.

3.3.4 Starre Körper

Man kann auch die Drehung der Planeten um sich selbst als starren Körper berechnen. Zur Vereinfachung betrachten wir wieder nur eine Drehung im 2-Dimensionalen. Statt Ort $\vec{r}$ und Geschwindigkeit $\vec{v}$ wird nun auch der Drehwinkel φ und die Winkelgeschwindigkeit ω verwendet. Das Bewegungsgesetz lautet:

$$\ddot{\varphi} = \dot{\omega} = \frac{M}{J} \text{ , mit} \tag{3.53}$$

$$J = \sum_i m_i (\vec{r}_i - \vec{r}_M)^2 \text{ (Trägheitsmoment)} \tag{3.54}$$

$$M = \sum_i (\vec{r}_i - \vec{r}_M) \times \vec{f}_i \text{ (Drehmoment)} \tag{3.55}$$

m_i sind die Massenpunkte, aus denen sich der Körper zusammensetzt, $\vec{r}_i$ ihre jeweiligen Koordinaten, f_i ihre anliegenden Kräfte und $\vec{r}_M$ die Schwerpunktskoordinate. Ein Beispiel für diese Drehung ist die Bewegung des Saturnmondes Hyperion, dessen äußere Form der eines Eis gleicht. Man stellt fest, daß seine Bewegung chaotisch ist.

3.4 Molekulardynamik

Mit der sog. Molekulardynamik kann man nicht nur Moleküle wie z.B. das Helimatom simulieren, sondern ganz allgemeine N–Teilchen–Systeme wie Gase, Flüssigkeiten, Membranen oder Schüttgüter.

Ein weiteres großes Gebiet der numerischen Physik ist die Molekulardynamik: eine Menge von Teilchen wird für eine bestimmte Zeit mit allen auf sie wirkenden Kräften simuliert. Dies ist ein typisches N–Körper-Problem. Dabei kann eine riesige Anzahl an Gleichungen auftreten, die alle berechnet werden müssen. Im November 1997 wurde durch den Zusammenschluß von zwei Supercomputern eine Molekularsimulation mit über einer Milliarde (10^9) Punkte eines Kristallgitters durchgeführt. Allerdings besitzt ein Kubikzentimeter Materie ungefähr 10^{22} Moleküle, also einen Faktor 10^{13} mehr. Man könnte also nur $1/10^{13}\,\mathrm{cm}^3$ simulieren.

In der Praxis ist es sowieso nur üblich bis zu 10^6 Teilchen zu simulieren. Weiterhin muß man wegen der großen auftretenen Kräfte die Zeitintervalle Δt sehr klein wählen, üblicherweise in der Größenordnung von 10^{-12}s= 1ps. Bei normalerweise 10^6 Iterationen kommt man dann auf eine Zeit von 1μs. Wegen dieser Einschränkungen werden nur kleine und kurze Vorgänge auf diese Weise simuliert wie Bruchspitzen, Schmelzen und Erstarren von Materie oder "head-crash", dem Aufprall des Lese-/Schreibkopfes auf die Magnetscheibe einer Festplatte.

3.4.1 Das Heliumatom

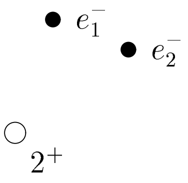


Abbildung 3.19: Elektronenbewegung im Heliumatom

Die elektrostatischen Kräfte ähneln den Gravitationskräften. So kann man auch die Bahn der beiden Elektronen in einem Heliumatom konstruieren. Neben der Anziehung von entgegengesetzten Ladungen kommt noch die Abstoßung durch gleiche Ladungen hinzu, die mit umgekehrtem Vorzeichen in die Berechnung eingehen. Wir setzen Masse und Ladung des Elektrons auf 1 ($m_e = q_e = 1$) und erhalten die beiden Kräfte für die Elektronen :

$$f_1 = -2\frac{\vec{r}_1}{r_1^3} + \frac{\vec{r}_1 - \vec{r}_2}{r_{12}^3} \tag{3.56}$$

$$f_2 = -2\frac{\vec{r}_2}{r_2^3} + \frac{\vec{r}_2 - \vec{r}_1}{r_{12}^3} \tag{3.57}$$

$$\tag{3.58}$$

Durch die möglichen starken Kräfte ist hierbei ein adaptiver Zeitschritt notwendig. Die entstehende Bewegung ist wieder chaotisch.

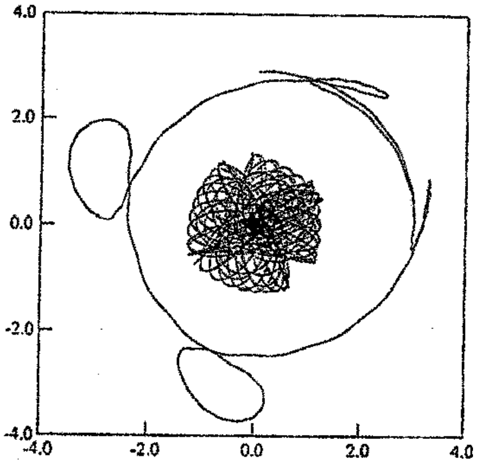


Abbildung 3.20: Bewegung der zwei Elektronen im Heliumatom.

3.4.2 Optimierung

Durch die gegenseitige Wechselwirkung ist der Aufwand bei N Teilchen $O(N^2)$, da der Einfluß von allen Teilchen auf jedes Teilchen berücksichtigt wird. Nun ist es aber so, daß die Stärke der Kraft von der Entfernung der Teilchen abhängt. Nur die Teilchen in näherer Umgebung haben einen wirklichen Einfluß auf die Bewegung. Um das auszunutzen gibt es verschiedene Ansätze:

1. Nachbarschaftstafeln
   Für jedes Teilchen i gibt es eine Liste von Teilchen die sich innerhalb einer Kugel (oder eines Kreises) mit dem Radius R um Teilchen i befinden. R muß dabei größer als die Reichweite der Kräfte sein. Für das Teilchen i werden nur die Kräfte aus den in der Liste stehenden Teilchen berechnet und die Liste wird regelmäßig erneuert. Der Erneuerungsprozeß kann zwar $O(N^2)$ sein, aber der Aufwand muß nur nach vielen Zeitschritten betrieben werden.

2. linked-cell (verbundene Zellen)
   Der Raum (die Ebene) wird in einzelne Zellen unterteilt. Jede Zelle besitzt eine Liste der Teilchen in ihr und jedes Teilchen in einer Zelle wechselwirkt nur mit Teilchen in umgebenden Zellen. Für die Berechnung werden nur die Teilchen aus der eigenen und den benachbarten Zellen herangezogen.

Durch Kombination dieser beiden Methoden kann man einen Aufwand von $O(N)$ erreichen. Außerdem lassen sich die Methoden sehr einfach parallelisieren und so auch auf Systemen mit mehreren Recheneinheiten effektiv realisieren.

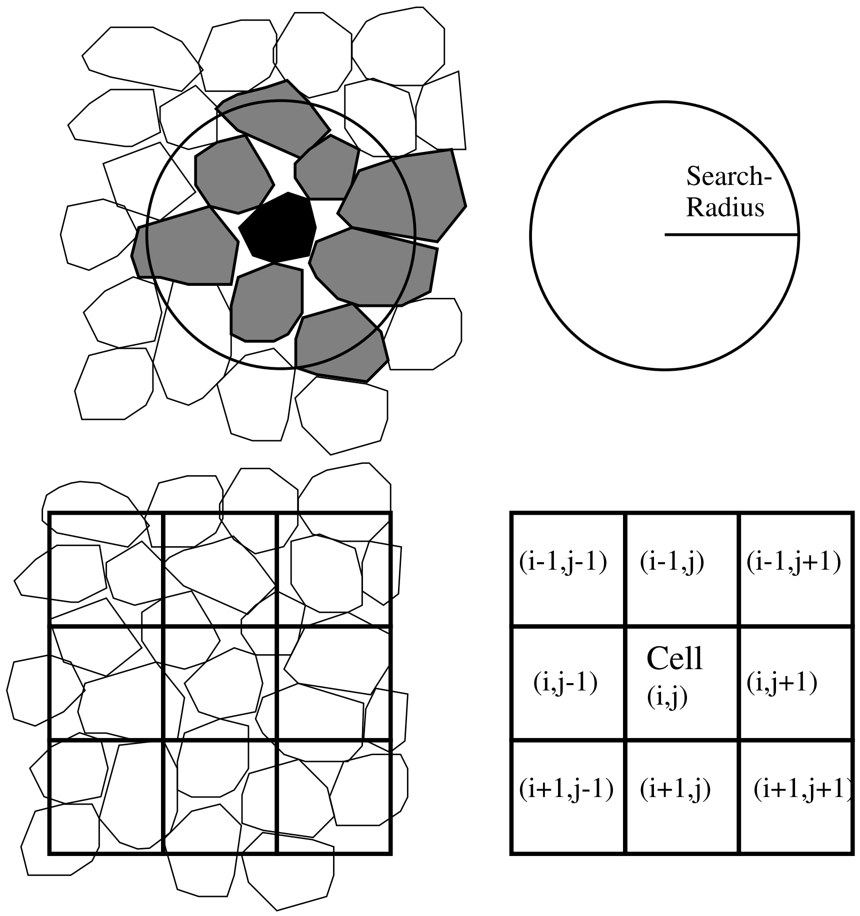


Abbildung 3.21: Verlet–Nachbarschaftstafeln (oben) und linked–cell (unten)

3.4.3 Weitere Vielteilchensysteme

Sind die Teilchen größer, wie Beispielsweise bei Sand, wo in einem Kubikzentimeter ungefähr 10^5 Körner sind, und durch kleinere auftretende Kräfte die Zeitintervalle größer, also $\Delta t \approx 10^{-5}s$, sein können, sind realistische Mengen und Zeitdauern von mehreren Minuten möglich.

Hierzu gehört beispielsweise die Simulation von Sand in einem Trichter, Fragmentation einer Wand mit dem Hammer, Teilchenvermischung in Flüssigkeiten oder die Zersplitterung zweier Kugeln nach einem Zusammenstoß.

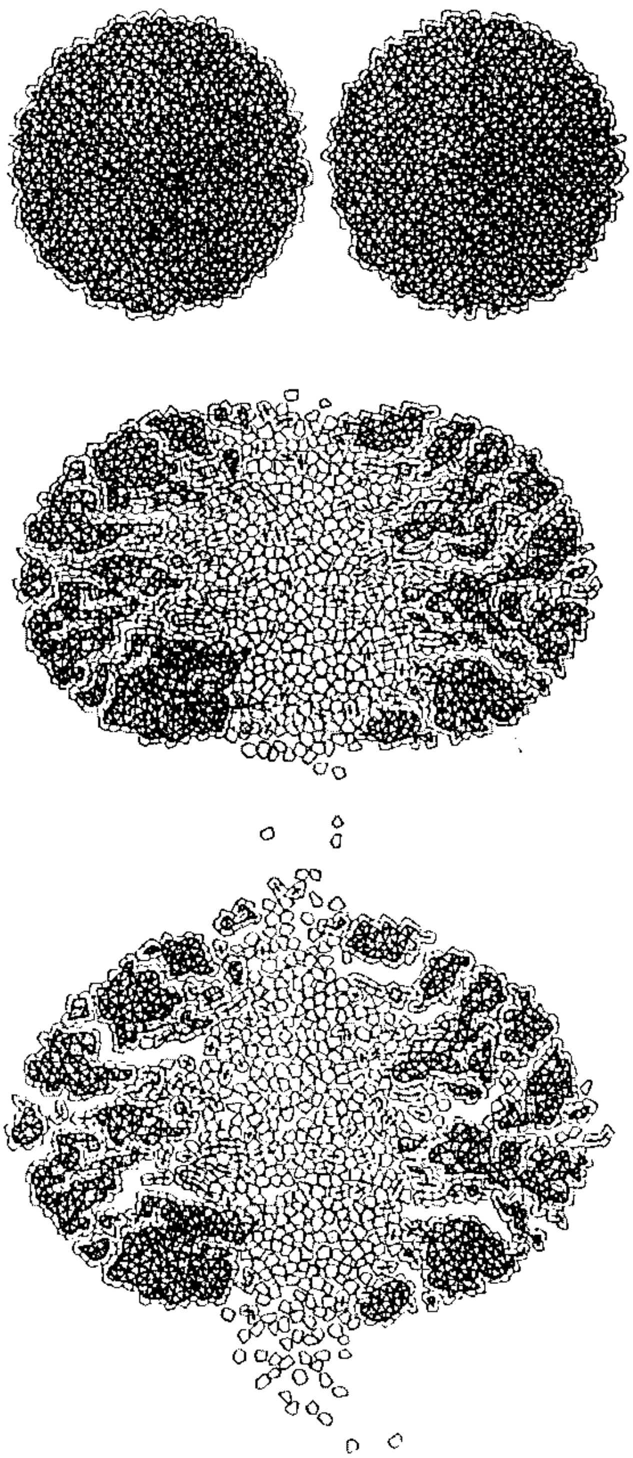


Abbildung 3.22: Zusammenstoß zweier Atomkerne

Kapitel 4

Populationsdynamik

- Quetelet (1835): “Essai de Physique Sociale”, Abhandlung über Populationsphysik
- Verhulst (1838): Modell für das dichteabhängige Wachstum einer Population (Deterministisch)
- Lotka, Volterrra, Kostizin (1923-1940): Grundlagen der theoretischen Ökologie (Deterministisch)
- Heute: Durch Einsatz von Computern ist die Einbeziehung stochastischer Faktoren in das Modell möglich

Die Populationsdynamik besteht aus der Modellierung eines Zustandes und seiner Dynamik und nicht aus einem Gesetz (wie das Newtonsche Gravitationsgesetz).

4.1 Betrachtung einer einzelnen Spezies

Der einfachste Fall ist eine einzelne Spezies, die sich ausbreitet oder, je nach Randbedingungen auch wieder schrumpfen kann. Ein Beispiel sind Bakterien, die sich vermehren solange Nahrung vorhanden ist.

4.1.1 Iterative Modelle

Zuerst kann man ein iteratives Modell formulieren, in dem die neue Population direkt und in einem Schritt aus der vorherigen hervorgeht.

Exponentielles Wachstum

Die gewöhnlichen Wachstumsvorgänge in der Natur werden meistens durch eine exponentielle Iteration gelöst, da zum Beispiel jedes Tier wieder mehrere Nachkommen besitzt. Jede Folgegeneration P_{n+1} ist das Produkt der Vorgängergeneration mit einem konstanten Faktor a:

$$P_{n+1} = aP_n \ . \tag{4.1}$$

Beginnt man mit einer Anfangspopulation P_0 ergibt sich $P_n = a^n P_0$. Hierbei würde allerdings die Population im Laufe der Zeit bis ins unendliche wachsen, so daß diese Form der Iteration nicht sinnvoll ist.

Logistische Gleichung

Bei Überbevölkerung geht die Population wegen Nahrungsmittelmangel und zu wenig Platz wieder zurück. Um dies zu berücksichtigen, wird in die Formel eine "umweltbedingte Kapazität" k ("Carrier Capacity") eingefügt, die die Population wieder verkleinert:

$$P_{n+1} = (a - k * P_n) * P_n \tag{4.2}$$

Durch die Normierung von $a = \mu$ und $y_n = (k/a)P_n$ entsteht die logistische Gleichung:

$$y_{n+1} = \mu y_n(1 - y_n) \tag{4.3}$$

μ ist in diesem Fall die Reproduktionsrate. Durch Anwenden der Gleichung entsteht in der Population ein Gleichgewicht. Trägt man das Gleichgewicht über der Reproduktionsrate auf, entsteht das Bild einer Bifurkation. Das heißt, wenn die Reproduktionsrate höher wird, entstehen mehrere verschiedene Gleichgewichte. Es folgt eine Kaskade von Bifurkationen von Populationen, bis diese in Chaos übergehen. Dieser Zustand ist für eine Populationssimulation nicht sinnvoll.

4.1.2 Gewöhnliche Differentialgleichungen

Wird die Population durch Differentialgleichungen simuliert, entsteht ein kontinuierliches Zeitverhalten. Die Zustandsvariable ändert sich stetig. Bei hoher Individuenanzahl ist gewährleistet, daß die ganzen Zahlen (die Individuen) problemlos als reele Zahlen genähert werden können. Allerdings ist auch bei dieser Methode keine Simulation statischer Fluktuationen durch Änderung der Bedingungen während der Simulation möglich, sondern nur eine Variation der Anfangsbedingungen.

Exponentielles Wachstum

Die Änderung des Systems ist hier die Differenz der Zahl der Geburten und der Zahl der Todesfälle, die beide proportional zu der Gesamtpopulation sind. Die Zahl der Geburten ist $b * y$ und die Zahl der Todesfälle $m * y$, wobei y die Individuenanzahl ist.

$$\frac{dy}{dt} = by - my = (b - m)y \tag{4.4}$$

Die Lösung dieser DGL besitzt wieder ein Exponentialverhalten, diesmal aber für kontinuierliche Zeiten:

$$y(t) = y_0 * e^{(b-m)*t} \tag{4.5}$$

Die Lösung ist entweder konstant (bei $y_0 = 0$ immer $y(t) = 0$), wächst ins unendliche (für $b - m > 0$), oder sie verschwindet exponentiell (für $b - m < 0$). Gleichung 4.5 ist also auch keine sinnvolle Beschreibung der meisten Populationen.

Logistisches Wachstum

Wird die Kapazität berücksichtigt (bei Verhulst 1838 und Schäfer 1954), entsteht die nichtlineare DGL

$$\frac{dy}{dt} = \mu y(1 - \frac{y}{K}) \tag{4.6}$$

Die Lösung lautet hierfür:

$$y_1(t) = \frac{K}{1 + (\frac{K}{y_0} - 1)e^{-\mu t}} \tag{4.7}$$

sowie $y_2 = 0$ und $y_3 = K$. Ohne Rauschen ist die DGL stabil, d.h. wenn in Gleichung 4.6 für y der Wert 0 eingegeben wird, bleibt das Ergebnis 0 und ist $y > 0$, ergibt sich als oberer Grenzwert die Kapazität K. Wird aber zu Glg. 4.6 ein Rauschterm (z.B. $A*\cos(t)$, mit sehr kleiner Amplitude A) hinzuaddiert, zeigt sich, daß die Lösung y_2 instabil ist: nach kurzer Zeit steigt die Funktion trotz des Anfangswertes 0 bis zu einem gewissen Wert exponentiell an (Abbildung 1, links). Der obere Grenzwert (y_3) ist auch inklusive Rauschterm stabil: bei einer Störung kehrt die Funktion exponentiell zur Ausgangslösung zurück (Abb. 4.1, rechts).

Es gibt noch weitere logistische Gleichungen, zum Beispiel mit konstanter Ernterate v:

$$\frac{dy}{dt} = \mu y(1 - \frac{y}{K}) - v \ , \tag{4.8}$$

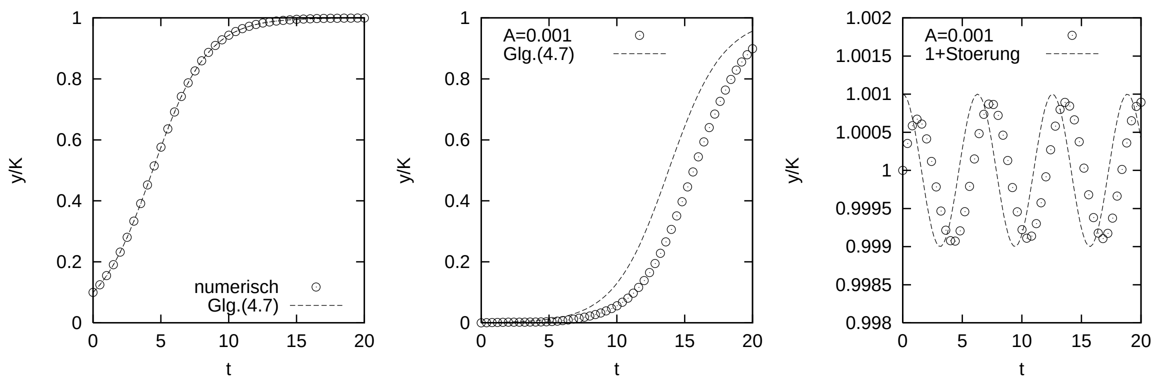


Abbildung 4.1: Numerische Lösung der Populationsgleichung 4.6 mit $\mu = 0.5$ und $K = 1$. Links: $y_0 = 0.1$, $A = 0$, Mitte: $y_0 = 0$, $A = 0.001$, Rechts: $y_0 = K$, $A = 0.001$.

wobei von einer Population immer ein Teil abgezogen ("geerntet") wird. Eine andere Möglichkeit zur Einbeziehung der Kapazität ist das "Mitscherliche Ertragsgesetz" von 1908:

$$\frac{dy}{dt} = \mu(K - y) \tag{4.9}$$

Die Kapazität K ist hier der maximal mögliche Ertrag. Die Gleichung hat den Vorteil, daß sie nicht quadratisch ist und somit am Anfang auch nicht so stark ansteigt, wie die anderen logistischen Gleichungen.

Vergleich logistisches -und exponentielles Wachstum

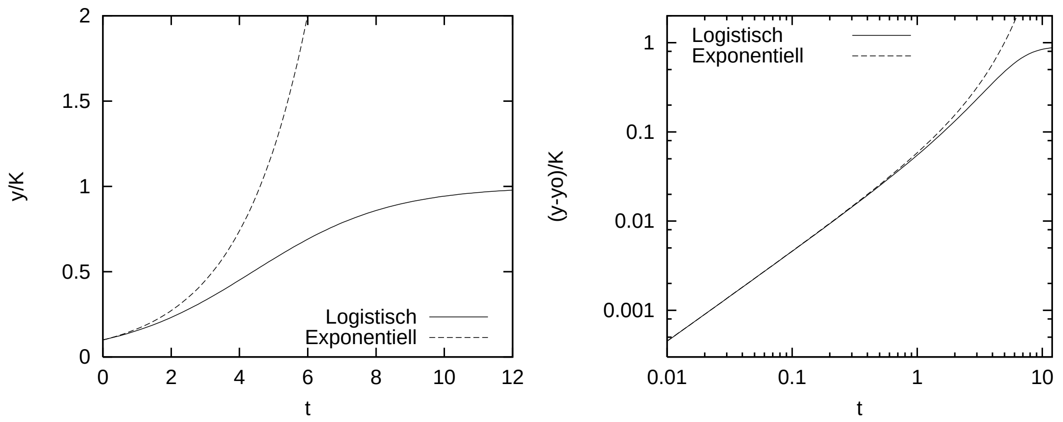


Abbildung 4.2: Vergleich von exponentiellem und logistischem Wachstum, Links: linear, Rechts: Halblogarithmisch. Gezeigt ist $(y(t) - y_0)/K$ aus Gleichung 4.6 (durchgezogene Linie) und exponentielles Wachstum $(1 - y_0/K)y_0(\exp(\mu t) - 1)$ (gestrichelte Linie).

In Abbildung 4.2 sieht man, daß die Exponentialfunktion und die Logistische Funktion anfangs gleichschnell wachsen. Man kann also am Anfang nicht unterscheiden, ob ein Wachstumsprozess exponentiell ist und ins unendliche wächst, oder logistisch und sich nach einer Zeit einem Grenzwert nähert. Derartige Prognosen sind bei Phänomenen wie beispielsweise der Verbreitung von AIDS, die es erst seit relativ kurzer Zeit gibt, bedeutungslos.

4.2 Zwei und mehr Spezies

In der Natur gibt es normalerweise mehr als nur eine Art von Lebewesen. Man unterscheidet zwischen verschiedenen Systemen wie z.B. *Konkurrenten* oder *Räuber–Beute–Systeme*. Die Räuber–Beute–Systeme ergeben meistens oszillierende Populationen, wobei der Räuber der Beute "hinterherhinkt", da die Population der Räuber von derjenigen der Beute abhängt. In Abbildung 3 ist dies teilweise nicht der Fall, da sie die Zahl der gefangenen Tiere zeigt, die auch abhängig vom Absatzmarkt sind (Preisentwicklung, Nachfrage).

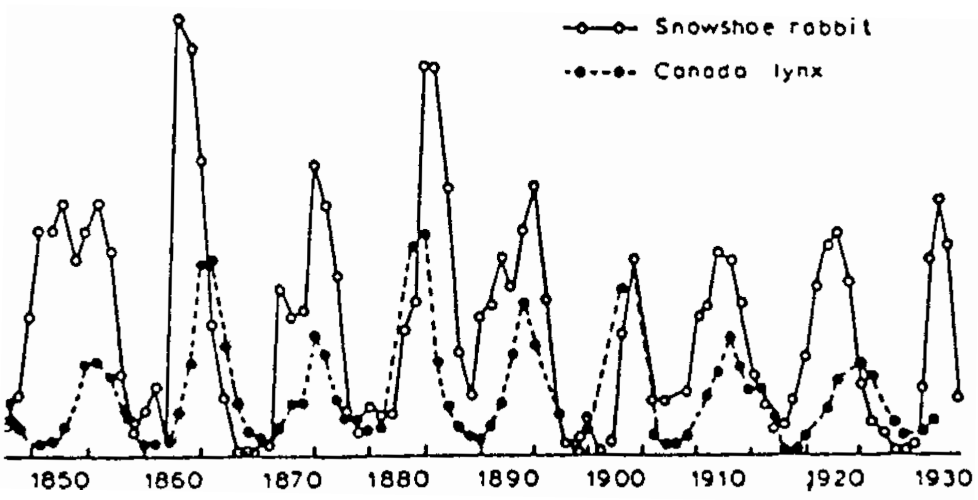


Abbildung 4.3: Räuber–Beute–System

4.2.1 Volterra–Gleichung (1925)

In der Volterra–Gleichung besitzen die Populationen y_i $(i = 1, 2)$ eine unbegrenzte Kapazität. Ausfälle geschehen bei der Beute durch "gefressen" werden, sowie bei den Räubern durch Todesfälle. Die Geburten sind bei der Beute proportional zur Population y_1, und bei den Räubern sowohl proportional zur eigenen Population y_2 als auch zur Population der Beutetiere y_1.

Damit ergibt sich das Gleichungssystem:

$$\frac{dy_1}{dt} = (b_1 - r_{21}y_2)y_1$$
$$\frac{dy_2}{dt} = (-m_2 + r_{12}y_1)y_2 \tag{4.10}$$

b_1 ist die Geburtenrate der Beute, m_2 die Sterberate der Räuber, r_{21} die "gefressene" Beute und r_{12} die "satten" Räuber.

Die in Abb. 4.4 abgebildete Lösung zeigt, das die Räuberpopulation zeitlich nach der Beutepopulation reagiert. Dieses Gleichungssystem besitzt immer periodische Lösungen, und es existieren deshalb keine Gleichgewichtszustände, d.h. auch das "Aussterben" einer Art ist nicht möglich.

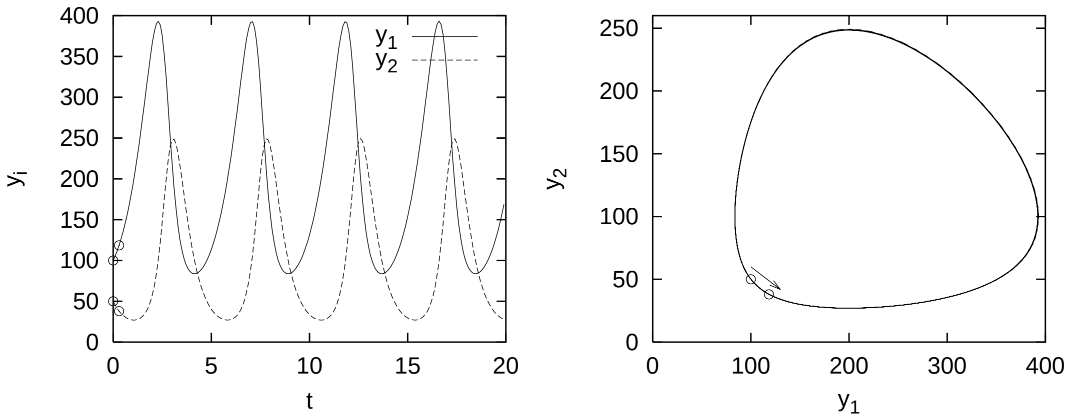


Abbildung 4.4: Räuber–Beute als Funktion der Zeit (links) und im Phasenraum (rechts). Gleichung 4.10 wurde mit den Parametern $b_1 = 1$, $m_2 = 2$, $r_{21} = r_{12} = 0.01$ und den Anfangswerten $y_1(0) = 100$ und $y_2(0) = 50$ numerisch gelöst. Die Kreise zeigen die Populationen zur Zeit $t = 0$ und $t = 0.3$ an, und der Pfeil im rechten Bild deutet die Orientierung im Phasenraum an.

4.2.2 Lotka–Volterra–Gleichung (1926)

Bei der Lottka–Volterra–Gleichung hat die Beutepopulation zusätzlich noch eine begrenzte Kapazität K, sonst sind die Gleichungen unverändert.

$$\frac{dy_1}{dt} = (b_1 - r_{21}y_2 - \frac{y_1}{K})y_1$$
$$\frac{dy_2}{dt} = (-m_2 + r_{12}y_1)y_2 \tag{4.11}$$

Die begrenzte Kapazität führt dazu, daß nach einer Übergangszeit (“Transiente”) immer ein Gleichgewicht erreicht wird. Es gibt also keine periodischen Lösungen mehr und es ist auch möglich, daß eine Art ausstirbt, wie in Abb. 4.5 für $K = 200$ zu sehen.

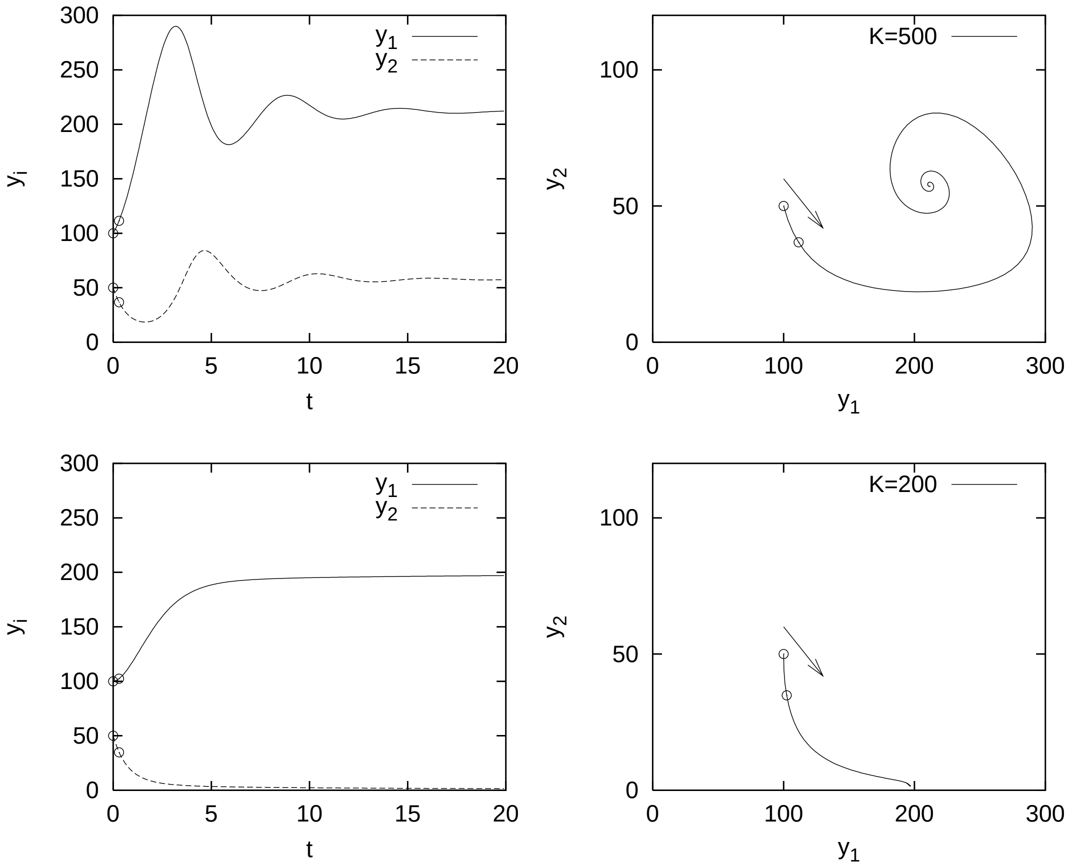


Abbildung 4.5: Räuber–Beute als Funktion der Zeit (links) und im Phasenraum (rechts). Gleichung 4.11 wurde mit den Parametern $b_1 = 1$, $m_2 = 2$, $r_{21} = r_{12} = 0.01$ und den Anfangswerten $y_1(0) = 100$ und $y_2(0) = 50$ numerisch gelöst. Oben hatte die Beutepopulation eine hohe ($K = 500$), unten eine relativ niedrige ($K = 200$) Kapazität. Die Kreise zeigen die Populationen zur Zeit $t = 0$ und $t = 0.3$ an, und der Pfeil im rechten Bild deutet die Orientierung im Phasenraum an.

4.2.3 Folgerungen

Vorteile von Differentialgleichungen

Die Modellierung der Populationsdynamik durch DGL’en ist beliebig allgemein möglich. Bestimmte Prozesse sind sehr gut nachvollziehbar und auch die Zeitabhängigkeit von Parametern (jahreszeitabhängige Schwankungen der Kapazität o.ä.) bei der numerischen

Lösung ist einfach zu implementieren.

Nachteile der differentiellen Modellierung

Bereits kleine Modifikationen können das Verhalten der DGL auf unvorhergesehene (und unerwünschte) Weise beeinflussen, wobei auch ein Erzwingen eines bestimmten Verhaltens sehr schwierig ist. Auswirkungen von Fluktuationen können nicht berücksichtigt werden und die differentiellen Amplituden sind nicht immer sinnvoll (z.B. 0.02 Elefanten).

Hauptsächlich fehlt jedoch die räumliche Auflösung, die im Folgenden genauer betrachtet werden soll.

4.3 Ausblick: Diskrete Modelle

Um die Nachteile der differentiellen Modellierung auszugleichen und die "unendliche" Reichweite der Räuber (jeder Räuber kann jedes Beutetier fressen) einzuschränken, kann man als Alternative auf ein diskretes System ausweichen. Der Algorithmus für ein Räuber–Beute System sieht folgendermaßen aus:

1. Das biologische System wird auf einem Gitter konstruiert.
2. Die Zahl der Individuen pro Gitterplatz wird abgespeichert.
3. *Migrationsschritt*: Räuber und Beute bewegen sich mit einer bestimmten Wahrscheinlichkeit auf einen anderen Gitterplatz.
4. *Jagdschritt*: Die Räuber versuchen in ihrer Umgebung, also auf ihrem Gitterplatz, Beute zu machen.
5. *Geburts/Todesrate*: Mit einer gewissen Wahrscheinlichkeit stirbt der Räuber / das Beutetier, oder bringt Nachkommen hervor. "Satte" Räuber bringen mit höherer Wahrscheinlichkeit Nachkommen hervor und sterben mit niedriger Wahrscheinlichkeit als "hungrige" Räuber.
6. Zurück zu Schritt 2.

Unterhalb einer gewissen Systemgröße passiert es nun, daß die Räuber die Beute ausrotten und anschließend selbst verhungern. Bei mittlerer Systemgröße zeigen sich Oszillationen in der Gesamtpopulation und für große Systeme bleibt sie konstant und fluktuiert nur noch lokal.