ANHANG A

Maple

MAPLE ist ein Computermathematiksystem, mit dem Probleme aus vielen Bereichen der Mathematik computergestützt gelöst werden können. MAPLE unterstützt sowohl die sogenannte symbolische als auch die numerische Rechnung. Angewendet auf Differentialgleichungen bedeutet die symbolische Rechnung, dass eine explizite Lösungsformel durch Anwendung von analytischen Lösungstechniken – oder genauer durch Anwendung von Algorithmen, die aus diesen entwickelt wurden – berechnet wird. Insbesondere können solche Algorithmen basierend auf den Methoden aus Kapitel 5 entwickelt werden- Die numerische Berechnung hingegen benutzt die in Kapitel 6 vorgestellten Verfahren, um Lösungen approximativ zu berechnen.

In diesem Anhang geben wir eine Einführung in eine Auswahl von MAPLE-Routinen zur Lösung gewöhnlicher Differentialgleichungen und der graphischen Darstellung der Lösungen. Da eine umfassende Einführung leicht ein eigenes Buch füllen würde, beschränken wir uns auf die Erklärung einiger grundlegender Methoden und geben im abschließenden Abschnitt einige kurze Ausblicke auf weitere Funktionen. Zur weitergehenden Einführung in MAPLE gibt es eine Vielzahl von Büchern und Internet-Seiten, z.B. den Kurs des Rechenzentrums der Universität Bayreuth [**13**]. Speziell mit dynamischen Systemen beschäftigt sich das Buch von S. Lynch [**15**].

Alle Beispiele wurden mit MAPLE 10.04 getestet und stehen unter `http://www.dgl-buch.de` als Worksheets zum Download zur Verfügung. Die hier im Buch abgedruckten MAPLE-Eingaben sind im klassischen „Maple Notation“-Eingabeformat angegeben, da dieses eine direkte Wiedergabe der einzugebenden Befehlsfolgen im Druck ermöglicht. In MAPLE kann dieses Format unter dem Menüpunkt „Tools → Options → Display“ durch Auswahl von „Input Display: Maple Notation“ ausgewählt werden. Alternativ können alle Eingaben aber natürlich

auch in der „2-D Math Notation“ eingegeben werden, sehen dann am Bildschirm aber anders aus als hier im Druck.

A.1. Grundlegendes

Um gewöhnliche Differentialgleichungen mit MAPLE zu lösen, müssen wir zuerst klären, wie man Differentialgleichungen und Anfangsbedingungen in MAPLE eingibt. Bevor die folgenden Definitionen eingegeben werden, empfiehlt es sich, einmal die Anweisung `restart;` auszuführen, um alle bereits definierten Variablen zu löschen.

Eingabe von Differentialgleichungen. Um Differentialgleichungen mit MAPLE zu lösen, müssen diese zunächst eingegeben werden. Die dabei benötigten Ableitungen werden mit der `diff`-Anweisung definiert. Um sie später weiterverarbeiten zu können, wird die gesamte Gleichung einer Variablen zugewiesen.

Betrachten wir als erstes Beispiel die Gleichung

$$\dot{x}(t) = x(t)^2. \tag{A.1}$$

Mit der MAPLE-Anweisung

```
> bsp1 := diff(x(t),t) = x(t)^2;
```

wird diese definiert und der Variablen `bsp1` zugewiesen. Um mehrdimensionale Gleichungen einzugeben, werden diese als System eindimensionaler Gleichungen geschrieben. Als Beispiel betrachten wir die lineare zweidimensionale Gleichung

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x. \tag{A.2}$$

mit $x = (x_1, x_2)^T$. Ausgeschrieben in zwei Gleichungen lautet diese

$$\dot{x}_1(t) = x_2(t), \quad \dot{x}_2(t) = -x_1(t)$$

und genauso kann dieses zweidimensionale System mittels

```
> bsp2 := diff(x1(t),t)=x2(t), diff(x2(t),t)=-x1(t);
```

eingegeben und der Variablen `bsp2` zugewiesen werden.

Darüberhinaus kann MAPLE auch Gleichungen höherer Ordnung direkt verarbeiten, ohne dass diese in ein System erster Ordnung (3.3) umgeformt werden müssen. Das Beispiel (A.2) z.B. ist äquivalent zu

$$\ddot{y}(t) = -y(t) \tag{A.3}$$

mit $(x_1, x_2) = (y, \dot{y})$. Gleichung (A.3) kann in MAPLE direkt als

```
> bsp3 := diff(y(t),t,t) = -y(t);
```

eingegeben und der Variablen `bsp3` zugewiesen werden.

Zu beachten ist bei der Eingabe von Differentialgleichungen, dass das unabhängige Argument (in der Notation dieses Buches also zumeist das t) stets mit eingegeben werden muss. Dies erkennt man sofort an den – oben nicht explizit aufgeführten – Ausgaben der entsprechenden MAPLE-Anweisungen in. Die für Gleichung (A.1) durchgeführte Definition liefert die Ausgabe

```
> bsp1 := diff(x(t), t) = x(t)^2;
```

$$\frac{d}{dt}x\left(t\right) = \left(x\left(t\right)\right)^2$$

während bei der Eingabe ohne „t“ die Ausgabe

```
> bsp1 := diff(x, t) = x^2;
```

$$0 = x^2$$

erscheint: MAPLE leitet den nicht von t abhängigen Ausdruck „x“ in `diff(x, t)` sofort nach t ab und liefert konsequenterweise „0“ als Ergebnis.

Eingabe von Anfangsbedingungen. Wollen wir ein Anfangswertproblem lösen, so muss zusätzlich eine Anfangsbedingung eingegeben und wiederum einer Variablen zugewiesen werden. Dies kann eine ganz abstrakte Anfangsbedingung der Form $x(t_0) = x_0$ sein, die für Gleichung (A.1) als

```
> ab1 := x(t0) = x0;
```

eingegeben und der Variablen `ab1` zugewiesen wird. Die Notwendigkeit dieser abstrakten Definition wird im folgenden Abschnitt klar werden. Ebenso kann man aber auch konkrete numerische Zahlenwerte benutzen, z.B.

```
> ab1num := x(1) = 10;
```

Für die Beispiele (A.2) und (A.3) geht dies analog, indem man die Dimension bzw. Ordnung des Beispiels berücksichtigt. Wollen wir z.B. für Gleichung (A.2) den abstrakten Anfangswert `x0=(x01,x02)` zur Anfangszeit `t0` definieren, so geht dies mit

```
> ab2 := x1(t0) = x10, x2(t0) = x20;
```

konkrete Zahlenwerte weist man mit

```
> ab2num := x1(0) = 5, x2(0) = 2;
```

zu. Um für Gleichung (A.3) eine eindeutige Lösung zu erhalten, muss man sowohl $y(t_0)$ als auch $\dot{y}(t_0)$ festlegen, was für die abstrakten Werte `y0, dy0` mittels

```
> ab3 := y(t0) = y0, D(y)(t0) = dy0;
```

und für konkrete Zahlenwerte (die hier äquivalent zu `x0=(x01,x02)` für (A.2) gewählt werden) mit

```
> ab3num := y(0) = 5, D(y)(0) = 2;
```

geschieht. Die hierbei verwendete Anweisung `D(y)` berechnet ähnlich wie das oben verwendete `diff(y(t),t)` die Ableitung $\dot{y}$. Während aber `diff(y(t),t)` den Ausdruck $\dot{y}(t)$ in Form eines MAPLE-Terms berechnet, liefert $D(y)$ die Funktion $t \mapsto \dot{y}(t)$ in Form eines MAPLE-Operators. Damit lässt sich die Anfangsbedingung der Ableitung kompakter und intuitiver schreiben.

A.2. Analytische Lösungen

In diesem Abschnitt wollen wir erläutern, wie die `dsolve`-Anweisung benutzt werden kann, um gewöhnliche Differentialgleichungen analytisch zu lösen. Für die computergestützte analytische Lösung gelten natürlich die gleichen prinzipiellen Beschränkungen, wie wir sie am Anfang von Kapitel 5 skizziert haben. Allerdings kann die Fähigkeit des Computers, viele verschiedene Methoden in sehr kurzer Zeit systematisch durchzuprobieren, bei der Berechnung analytischer Lösungen eine enorme Hilfe sein.

Die einfachste Form des Aufrufs zur Lösung einer Differentialgleichung lautet für (A.1)

```
> dsolve(bsp1);
```

und liefert die Ausgabe

$$x\,(t) = (-t + \_C1\,)^{-1}$$

Hier erkennt man die typische Struktur der Ausgabe dieser Anweisung: Die allgemeine Lösung wird abhängig von einer Variablen `_C1` dargestellt, die beliebige reelle Werte annehmen kann. Ähnlich ist das bei den beiden anderen Gleichungen (A.2) und (A.3), bei denen Ein- und Ausgabe wie folgt lauten:

```
> dsolve({bsp2});
```

$$\begin{aligned}\{x1\,(t) &= \_C1\,\sin(t) + \_C2\,\cos(t)\,,\\ x2\,(t) &= \_C1\,\cos(t) - \_C2\,\sin(t)\}\end{aligned}$$

und

```
> dsolve(bsp3);
```

$$y\,(t) = \_C1\,\sin(t) + \_C2\,\cos(t)$$

Hier hängen die Lösungen nun von jeweils zwei Konstanten `_C1` und `_C2` ab. Zu beachten ist, dass das Argument `dgl2` von `dsolve` im Fall von (A.2) in geschweifte Klammern gesetzt werden muss. Der Grund dafür ist, dass es sich hier um ein System von zwei Gleichungen handelt, das auf diese Weise als ein einziges Argument an die Routine übergeben wird. Etwas uneinheitlich ist dabei die Ausgabereihenfolge der beiden Lösungekomponenten in MAPLE impementiert; es kann vorkommen, dass die beiden Lösungskomponenten in umgekehrter Reihenfolge ausgegeben werden.

Um nun die Lösung für eine gegebene Anfangsbedingung der Form $x(t_0) = x_0$ zu berechnen, müsste man nur noch `_C1` bzw. `_C1` und `_C2` so berechnen, dass diese Bedingung erfüllt ist. Dies kann man aber MAPLE automatisch erledigen lassen, indem wir die oben definierten „abstrakten“ Anfangsbedingungen in den `dsolve`-Befehl einsetzen. Für unsere drei Beispiele erhalten wir so

```
> dsolve({bsp1,ab1});
```

$$x\,(t) = \frac{x0}{1 - tx0 + t0\;x0}$$

```
> dsolve({bsp2,ab2});
```

$$\begin{aligned}\{x1\,(t) &= (\cos(t0)\,x20 + \sin(t0)\,x10)\sin(t)\\ &\quad + (-\sin(t0)\,x20 + x10\,\cos(t0))\cos(t)\,,\\ x2\,(t) &= (\cos(t0)\,x20 + \sin(t0)\,x10)\cos(t)\\ &\quad - (-\sin(t0)\,x20 + x10\,\cos(t0))\sin(t)\}\end{aligned}$$

```
> dsolve({bsp3,ab3},y(t));
```

$$\begin{aligned} y(t) &= (y0\,\sin(t0) + \cos(t0)\,dy0)\sin(t) \\ &\quad + (-\sin(t0)\,dy0 + \cos(t0)\,y0)\cos(t) \end{aligned}$$

Die Anfangsbedingung wird bei der Übergabe also mit der Differentialgleichung in geschweiften Klammern zusammengefasst und so an `dsolve` übergeben. Bei `bsp3` wird hier zudem mit dem weiteren Argument `y(t)` explizit festgelegt, nach welchem Term die Lösung aufgelöst werden soll. Dies ist bei diesem Beispiel notwendig, damit MAPLE die Differentialgleichung und die Anfangsbedingung auseinanderhalten kann, denn in beiden treten hier ja Ableitungen auf. Bei `bsp1` und `bsp2` ist diese Angabe nicht notwendig, da MAPLE hier selbst erkennt, nach welchem Term die Lösung sinnvollerweise aufgelöst werden muss.

Wiederholen wir diese Anweisungen mit den entsprechenden numerischen Anfangsbedingungen, so erhalten wir die zugehörigen Lösungen mit den angegebenen Zahlenwerten als Anfangsbedingung:

```
> dsolve({bsp1,ab1num});
```

$$x(t) = -10\,(10\,t - 11)^{-1}$$

```
> dsolve({bsp2,ab2num});
```

$$\{x1(t) = 5\cos(t) + 2\sin(t),\, x2(t) = -5\sin(t) + 2\cos(t)\}$$

```
> dsolve({bsp3,ab3num},y(t));
```

$$y(t) = 5\cos(t) + 2\sin(t)$$

Wie geht MAPLE intern vor? Wir haben in Kapitel 5 verschiedene Möglichkeiten kennen gelernt, Lösungen von Differentialgleichungen analytisch zu lösen. Für lineare Differentialgleichungen haben wir bereits vorher in Kapitel 2 gesehen, dass sich die Lösungen explizit über die Matrix-Exponentialfunktion berechnen lassen. Vereinfacht gesagt prüft MAPLE all diese Möglichkeiten – und noch viele mehr – beim Aufruf

von `dsolve` systematisch auf ihre Anwendbarkeit, um dann im Erfolgsfall die entsprechende Methode auszuführen. Welche Methoden MAPLE dabei probiert, kann man sehen, wenn man die Anweisung

```
> infolevel[dsolve]:=3;
```

eingibt. Diese bewirkt, dass `dsolve` bei allen nachfolgenden Ausführungen detailliert Auskunft über die geprüften Methoden gibt – die Voreinstellung ist 1, damit werden keinerlei Informationen ausgegeben. Wiederholen wir nach der Anweisung z.B. den Aufruf

```
> dsolve(bsp1);
```

so erhalten wir die Ausgabe:

```
Methods for first order ODEs:
--- Trying classification methods ---
trying a quadrature
trying 1st order linear
trying Bernoulli
<- Bernoulli successful
```

$$x\left(t\right) = -\left(t - \_C1\right)^{-1}$$

Zuerst wird also die Quadratur ausprobiert, d.h. es wird getestet, ob sich die Gleichung – gegebenenfalls nach Umformungen – durch einfaches Integrieren (also durch Quadratur) des Vektorfeldes gelöst werden kann. Dies kann funktionieren, wenn die rechte Seite nur von einer der beiden Variablen t oder x abhängt, scheitert aber oft daran, dass keine Stammfunktion gefunden werden kann, was offenbar auch hier der Fall ist. Als zweites wird geprüft, ob die Gleichung linear ist, was hier ebenfalls nicht zutrifft. Als drittes wird getestet, ob die eingegebene Gleichung eine Bernoulli-Differentialgleichung ist. Dies ist hier der Fall, weswegen diese Lösungsmethode hier angewendet wird. Weitere Methoden, die MAPLE zur Lösung probiert sind z.B. die Ermittlung von Symmetrien des Vektorfeldes, Lösungsformeln für homogene Gleichungen oder Gleichungen vom d'Alembertschen Typ und viele mehr. Eine ausführliche Übersicht findet sich auf der Hilfe-Seite zur Anweisung `odeadvisor`, vgl. auch Abschnitt A.5.

Im Zusammenhang mit der `dsolve`-Routine sollte darauf hingewiesen werden, dass MAPLE jedes nicht weiter definierte Symbol standardmäßig als komplexe Zahl behandelt. Beim Lösen einer reellen

gewöhnlichen Differentialgleichung kann das dazu führen, dass die berechnete Lösung nicht so weit vereinfacht werden kann, dass eine allgemeine Lösung ausgegeben werden kann. Als Beispiel betrachte die Gleichung

$$\dot{x}(t) = x(t)^3 - x(t)$$

aus Beispiel 5.9. Mit

```
> dgl := diff(x(t),t)=x(t)^3-x(t);
> ab:= x(t0)=x0;
> dsolve({dgl,ab},x(t));
```

erhält man kein Ergebnis, da sich die aus der Methode für Bernoulli-Gleichungen ergebende allgemeine Lösungsformel nicht weit genug vereinfachen lässt. Dies sieht man z.B. indem man die Anfangsbedingung weglässt, da dann eine Lösung ausgegeben wird. Ebenso kann man durch Heraufsetzen von `infolevel[dsolve]` sehen, dass die Gleichung korrekt als Bernoulli-Gleichung erkannt wird.

Abhilfe schafft hier die explizite Angabe, dass die Gleichung als reell zu lösen ist, was mit der nachgestellten Anweisung `assuming real` realisiert wird. Folgerichtig liefert der Aufruf

```
> dsolve({dgl,ab},x(t)) assuming real;
```

mit

$$x\left(t\right) = -\frac{\sqrt{-\left(-x0^2 e^{2\,t0} - e^{2\,t} + e^{2\,t} x0^2\right) e^{2\,t0}}\, x0}{-x0^2 e^{2\,t0} - e^{2\,t} + e^{2\,t} x0^2}$$

die richtige allgemeine Lösung.

A.3. Numerische Lösungen

Numerische Lösungen werden in MAPLE ebenfalls mit `dsolve` berechnet, indem der Anweisung die Option `numeric` übergeben wird.

Die `numeric` Option. Für das Beispiel (A.1) erhält man mit der `numeric`-Option

```
> numlsg1:=dsolve({bsp1,ab1num}, numeric);
```

die Ausgabe

$$numlsg1 := \mathbf{proc}(x\_rkf45) \ldots \mathbf{end\ proc}$$

Dies erscheint auf den ersten Blick unverständlich, ist aber leicht erklärt: Der Variablen `numlsg1` werden hier keine Zahlenwerte sondern eine Berechnungsvorschrift zugewiesen, die wie jede andere MAPLE-Funktion

ausgewertet werden kann. Um z.B. den numerisch berechneten Wert an der Stelle $t = 0$ zu erhalten, gibt man ein:

```
> numlsg1(0);
```

$$[t = 0.0, x\,(t) = 0.909091248236105320]$$

Für die beiden anderen Beispiele lauten die Aufrufe und Ausgaben entsprechend

```
> numlsg2:=dsolve({bsp2,ab2num}, numeric):
> numlsg2(0);
```

$$[t = 0.0, x1\,(t) = 5.0, x2\,(t) = 2.0]$$

und

```
> numlsg3:=dsolve({bsp3,ab3num},numeric):
> numlsg3(0);
```

$$[t = 0.0, y\,(t) = 5.0, \frac{d}{dt}y\,(t) = 2.0]$$

Bei der Gleichung zweiter Ordnung wird hier also nicht nur der Wert sondern auch die Ableitung im gewählten Zeitpunkt ausgegeben. Bei der Definition von `numlsg2` und `numlsg3` haben wir hier einen Doppelpunkt ans Ende der Anweisung gesetzt, wodurch die Ausgabe unterdrückt wird.

Intern wird der numerische Algorithmus zur Berechnung der Lösung bei jeder Auswertung neu ausgeführt, was bei vielen aufeinanderfolgenden Auswertungen recht lange Rechenzeiten nach sich ziehen kann. Alternativ kann daher die `range`-Option übergeben werden, mit der ein Intervall von t-Werten festgelegt wird, auf dem die Lösung berechnet wird. Mit

```
> numlsg1_range:=dsolve({bsp1,ab1num}, numeric,
          range = -10..1):
```

wird die Lösung auf einem Gitter $\mathcal{T}$ vorausberechnet, das den angegebenen Bereich – hier also $t \in [-10, 1]$ – abdeckt. Bei jeder Auswertung wird der gewünschte Wert dann durch Interpolation aus den gespeicherten Werten in den Gitterpunkten berechnet.

Weitere Ausgabeformate. Neben der standardmäßigen Definition der Lösung als auswertbarer Algorithmus gibt es viele weitere Möglichkeiten der Ausgabe der Lösung, die mit der Option `output` gewählt werden können und von denen wir hier zwei besprechen wollen.

Zum Beispiel ist es möglich, die Lösung der Differentialgleichung mit `dsolve` direkt für eine Menge vorgegebener Zeiten auszuwerten. Dies erreicht man, indem man `dsolve` mit der `output`-Option wie folgt ein Array mit Zeiten übergibt:

```
> dsolve({bsp2,ab2num}, numeric,
          output=array([0,0.25,0.5,0.75,1]));
```

$$\left[\begin{array}{c} [t, x1\,(t), x2\,(t)] \\ \left[\begin{array}{ccc} 0.0 & 5.0 & 2.0 \\ 0.2500000000 & 5.3393701809 & 0.7008049215 \\ 0.5000000000 & 5.346764055 & -0.6419626068 \\ 0.7500000000 & 5.021722044 & -1.944816217 \\ 1.0 & 4.384453794 & -3.126750498 \end{array}\right] \end{array}\right]$$

Eine weitere Variante ist, die berechnete Funktion als eine stückweise definierte MAPLE-Funktion auszugeben. Dazu wird `output=piecewise` und zusätzlich mittels `range` ein Bereich übergeben, also z.B.

```
> dsolve({bsp1,ab1num}, numeric, output=piecewise,
          range=0..0.2);
```

$$\left[t = t, x\,(t) = \begin{cases} \mathit{undefined} & t < 0.0 \\ 0.9069526370 + 0.9085562153\,t \\ +0.8661496472\,(t - 0.05088529784)^2 \\ +0.8294192240\,(t - 0.05088529784)^3 \\ +0.7659454715\,(t - 0.05088529784)^4, & t \le 0.1017705956 \\ 0.8854291406 + 1.114621428\,t \\ +1.177021463\,(t - 0.1528157176)^2 \\ +1.249739507\,(t - 0.1528157176)^3 \\ +1.271889983\,(t - 0.1528157176)^4, & t \le 0.2 \\ \mathit{undefined} & \mathit{otherwise} \end{cases}\right]$$

Die approximative Lösung wird so als stückweise definierte Funktion zurückgegeben, deren Abschnitte gerade Polynome vierten Grades sind.

Der Grad vier wird dabei gewählt, da Polynome vierten Grades eine Approximation fünfter Ordnung auf jedem Teilintervall liefern und damit zwischen den Gitterpunkten die gleiche Genauigkeit liefern wie das verwendete Runge-Kutta Verfahren der Konvergenzordnung fünf, vgl. den folgenden Abschnitt.

Numerische Schemata. Das numerische Standardschema in MAPLE ist das Runge-Kutta-Fehlberg-Verfahren `rkf45`, ein eingebettetes explizites Runge-Kutta-Verfahren mit Konsistenzordnungen 4 und 5. Die Schrittweite wird gemäß der in Abschnitt 6.5 beschriebenen Methode gesteuert: sie wird in jedem Schritt so gewählt, dass der geschätzte lokale Fehler eine Fehlerschranke der Form

$$\|\tilde{x}(t_{i+1}) - x(t_i + h_i; t_i, \tilde{x}(t_i))\| \leq \texttt{abserr} + \texttt{relerr} \cdot \|\tilde{x}(t_{i+1})\| \quad \text{(A.4)}$$

erfüllt[1], wobei die Werte `abserr` und `relerr` optional vorgegeben werden können. Defaultwerte für `rkf45` sind $\texttt{abserr} = 10^{-7}$ und $\texttt{relerr} = 10^{-6}$.

Für steife Differentialgleichungen bietet MAPLE als weiteres Standardschema das Rosenbrock-Verfahren an, ein eingebettetes implizites Runge-Kutta-Verfahren mit Konsistenzordnungen 3 und 4. Mit der Option `stiff = true` wird auf dieses Verfahren umgeschaltet.

Über diese Standardschemata hinaus sind in MAPLE eine ganze Reihe weiterer Verfahren implementiert, die mit der Option `method` ausgewählt werden können. Einen Überblick gibt die Hilfeseite, die mit `?dsolve, numeric` aufgerufen wird. An dieser Stelle sei nur noch auf das explizite Euler-Verfahren hingewiesen, das in MAPLE mit der Option `method = classical[foreuler]` ausgewählt wird. Dieses Verfahren verwendet keine Schrittweitensteuerung sondern eine konstante Schrittweite, die mit der Option `stepsize` ausgewählt wird. Ein Beispielaufruf für (A.1) lautet

```
> numlsg_euler:=dsolve({bsp1,ab1num}, numeric,
          method=classical[foreuler], stepsize=0.01);
```

A.4. Grafische Darstellung der Lösungen

Eine wesentliche Stärke der computergestützten Lösung von Differentialgleichungen – sei es analytisch oder numerisch – ist die Möglichkeit, die erhaltenen Ergebnisse sofort graphisch darzustellen.

[1] Beachte, dass nur der *geschätzte* lokale Fehler diese Schranke einhält. Auch wenn diese Verfahren im Allgemeinen recht zuverlässige Ergebnisse liefern, ist eine rigorose Einhaltung der Schranke durch den *tatsächlichen* Fehler nicht gewährleistet.

Hier beschreiben wir, wie das in MAPLE gemacht wird. Alle mit den Methoden dieses Abschnittes erzeugten Grafiken können dabei durch Anklicken mit der rechten Maustaste menügesteuert nachträglich modifiziert und in verschiedenen Formaten exportiert werden.

Darstellung analytischer Lösungen. Da analytische Lösungen als übliche MAPLE-Ausdrücke geliefert werden, können die Standardgrafikanweisungen `plot` und `plot3d` zu ihrer graphischen Darstellung verwendet werden. Wir veranschaulichen den Gebrauch dieser Routinen anhand unserer Beispiele (A.1) und (A.2), also `bsp1` und `bsp2`.

Zunächst müssen wir die Lösungsausdrücke an Variablen zuweisen, um diese dann plotten zu können. Für `bsp1` geschieht dies mittels

```
> lsg1:=dsolve({bsp1,ab1num});
```

$$lsg1 := x(t) = -10\,(10\,t - 11)^{-1}$$

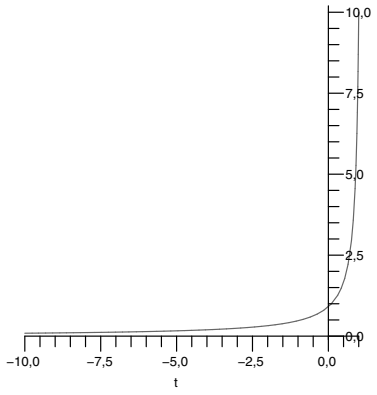
```
> xt := rhs(lsg1);
```

$$xt := -10\,(10\,t - 11)^{-1}$$

Damit wird zunächst die Lösungsgleichung der Variablen `lsg1` zugewiesen und dann die rechte Seite der Gleichung (`rhs` = right hand side) der Variablen `xt`. Diese kann nun mittels

```
> plot(xt,t=-10..1);
```

auf einem ausgewählten Intervall (hier $[-10, 1]$) graphisch dargestellt werden. Das Ergebnis ist



Für das zweidimensionale Beispiel `bsp2` funktioniert dies ähnlich, allerdings müssen hier die rechten Seiten der beiden Lösungsgleichungen separat an Ausdrücke `x1t` und `x2t` zugewiesen werden. Dies geschieht mittels

```
> lsg2:=dsolve({bsp2,ab2num});
```

$$lsg2 := \{x1\,(t) = 2\,\sin(t) + 5\,\cos(t)\,, x2\,(t) = 2\,\cos(t) - 5\,\sin(t)\}$$

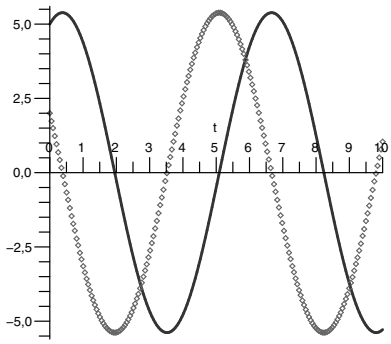
```
> x1t:=rhs(lsg2[1]); x2t:=rhs(lsg2[2]);
```

$$x1t := 2\,\sin(t) + 5\,\cos(t)$$
$$x2t := 2\,\cos(t) - 5\,\sin(t)$$

Hier kann die oben bereits erwähnte Eigenart von MAPLE, die Lösungskomponenten gelegentlich in umgekehrter Reihenfolge auszugeben, Probleme bereiten. Passiert dies, so müssen die Definitionen von `x1t` und `x2t` vertauscht werden.

Im zweidimensionalen gibt es nun verschiedene Möglichkeiten, die Lösung darzustellen. Die erste besteht darin, die beiden Komponenten der Lösung als Funktionen in t in eine Grafik zu plotten. Dies geschieht mittels[2]

```
> plot([x1t, x2t], t=0..10, color=[red, blue],
        style=[point, line], thickness=[1,2],
        numpoints=200);
```



[2]Verschiedene Farben sind hier und im Folgenden in Graustufen gedruckt.

Die beiden Lösungen werden innerhalb einer eckigen Klammer (also als eine MAPLE-Liste) an `plot` übergeben. Zudem nutzen wir hier noch einige Optionen, die die Farbe (`color`), den Zeichenstil (`style`), die Dicke der Linie (`thickness`) und die Auflösung über die Anzahl der Punkte (`numpoints`) festlegen. Hierbei erhalten beide Kurven unterschiedliche Optionen, wenn die zugehörigen Parameter ebenfalls als Listen übergeben werden (wie hier bei `color`, `style` und `thickness`). Wird nur ein Parameter für eine Option übergeben (wie hier bei `numpoints`), so gilt diese für alle Kurven. Zu beachten ist hier, dass `numpoints` nicht nur bei der expliziten Verwendung von `point` als Stil wirksam ist, sondern auch die Auflösung bei der Darstellung der Linie beeinflusst, was nützlich sein kann, wenn die Standardauflösung in MAPLE zu grobe Plots erzeugt.

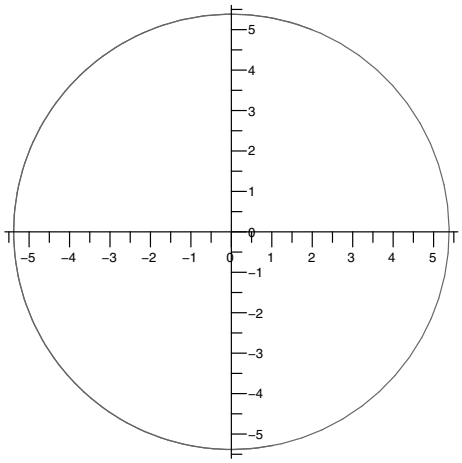
Will man mehrere verschiedene Lösungskurven mit vielen Optionen in einer Grafik darstellen, so kann es sehr unübersichtlich sein, diese in eine einzelne `plot`-Anweisung zu schreiben. In diesem Fall kann man die Kurven alternativ separat mit `plot` erzeugen, jeweils einer Variablen zuweisen und dann mittels `plots[display]` in eine Grafik plotten. Für die obige Grafik lautet diese alternative Anweisungsfolge

```
> kurve1:=plot(x1t, t=0..10, color=red, style=point,
        thickness=1, numpoints=200):
  kurve2:=plot(x2t, t=0..10, color=blue, style=line,
        thickness=2, numpoints=200):
  plots[display](kurve1,kurve2);
```

Die zweite Möglichkeit der Darstellung der Lösungen ist das Phasenportrait, in dem die Lösungskurve $\{(x_1(t), x_2(t))^T \mid t \in [t_1, t_2]\} \subset \mathbb{R}^2$ geplottet wird. Dies geht in MAPLE (hier mit $t_1 = 0$ und $t_2 = 10$) mit

```
> plot([x1t, x2t, t=0..10]);
```

und erzeugt die Grafik

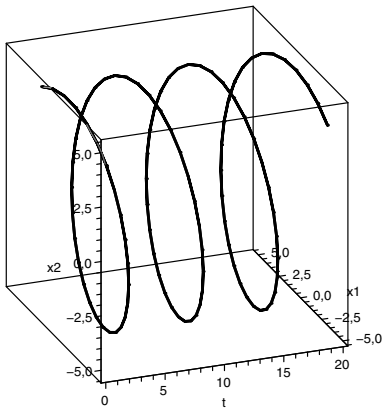


Auch hier können natürlich nach Wunsch die oben beschriebenen Optionen angewendet werden.

Die letzte Möglichkeit ist die dreidimensionale Darstellung der Lösungskurve $\{(t, x_1(t), x_2(t))^T \mid t \in [t_1, t_2]\} \subset \mathbb{R}^3$, die mit

```
> plot3d([t,x1t,x2t],t=0..20,x=-1..1,grid=[100,2],
        axes=boxed,labels=["t","x1","x2"],
        orientation=[-111,67], thickness=2);
```

das Bild



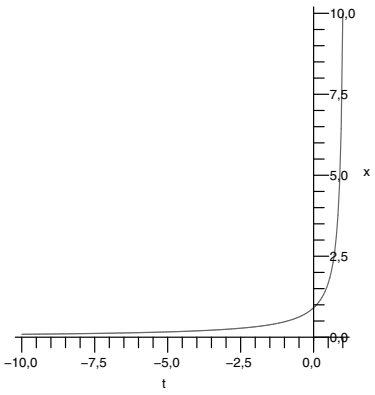
erzeugt. Da `plot3d` eigentlich zur Darstellung zweidimensionaler Flächen im $\mathbb{R}^3$ dient, wurde hier ein kleiner Trick angewendet, um die in den $\mathbb{R}^3$ eingebettete eindimensionale Lösungskurve zu zeichnen: es wurde eine

nicht verwendete Variable `x` als zweite Dimension übergeben. Die Option `grid=[100,2]` spielt hier die Rolle von `numpoints` in `plot`, wobei hier nur die erste Dimension (also t) fein aufgelöst werden muss, da die Kurve von der zweiten (also der „künstlichen Variablen“ x) ja gar nicht abhängt. Die `axes`- und `label`-Optionen bestimmen das Aussehen und die Beschriftung der Achsen, `orientation=[-111,67]` legt die Drehung der Grafik im Raum fest (diese kann am Bildschirm mit der Maus geändert werden) und `thickness` bestimmt wie in `plot` die Dicke der Kurve.

Darstellung numerischer Lösungen. Zur graphischen Darstellung der mittels `dsolve` mit der Option `numeric` berechneten numerischen Lösungen steht im `plots`-Paket von MAPLE mit `odeplot` eine eigene Routine zur Verfügung. Diese kann nach Aktivierung des Pakets mittels `with(plots)` direkt mit `odeplot` aufgerufen werden, alternativ kann sie nach den MAPLE-Konventionen auch ohne Aktivierung des Paketes mit der Langform `plots[odeplot]` aufgerufen werden[3]. Im Folgenden setzen wir voraus, dass die Anweisung `with(plots)` bereits ausgeführt wurde.

Für das eindimensionale Beispiel `bsp1` kann die graphische Darstellung nun wie folgt erzeugt werden

```
> numlsg1:=dsolve({bsp1,ab1num}, numeric,
          range=-10..1):
> odeplot(numlsg1,numpoints=200);
```



[3]Für die oben bereits angesprochene `display`-Anweisung gilt das Gleiche.

Hier wurde das Zeitintervall bereits in `dsolve` mittels `range=-10..1` festgelegt, alternativ kann der Zeitbereich auch in `odeplot` mittels `odeplot(numlsg1,-10..1,numpoints=200);` definiert oder abgeändert werden. Die Option `numpoints=200` bestimmt wie in `plot` die Auflösung der Grafik. Falls eines der beiden Standardverfahren `rkf45` oder Rosenbrock verwendet und in `dsolve` die `range`-Option benutzt wurde, kann die Auflösung alternativ mit `refine` festgelegt werden: die Option `refine=2` z.B. verdoppelt die Auflösung der Grafik.

Mit `odeplot` können auch animierte Darstellungen der Lösung erzeugt werden. Erweitert man den obigen Aufruf zu

```
> odeplot(numlsg1,numpoints=200,frames=20);
```

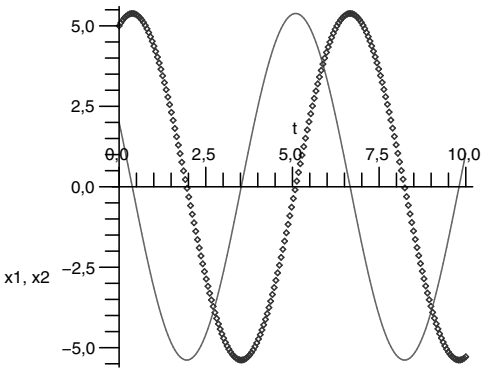
so erhält man eine Animation, die man nach Anklicken der Grafik über die Steuerelemente in der Kopfzeile von MAPLE starten kann.

Für höherdimensionale Beispiele wird neben der numerischen Lösung für jede zu zeichnende Kurve in einer Liste (also in einer in MAPLE durch eckige Klammern zusammengefassten Menge von Variablen) angegeben, welche Größen der Gleichung in Abhängigkeit von welcher anderen Größe gezeichnet werden sollen. Sollen Optionen wie z.B. `color` oder `style` nicht für alle Kurven sondern nur für eine Kurve gelten, werden diese mit in die entsprechende Liste geschrieben (dies ist anders als bei der `plot`-Anweisung). Sollen mehrere Kurven gezeichnet werden, werden die so erstellten Listen schließlich in einer weiteren Liste zusammengefasst. Das so erhaltene Konstrukt wird dann als zweites Argument an `odeplot` übergeben.

Wir illustrieren dies an unserem zweidimensionales Beispiel `bsp2`. Mit

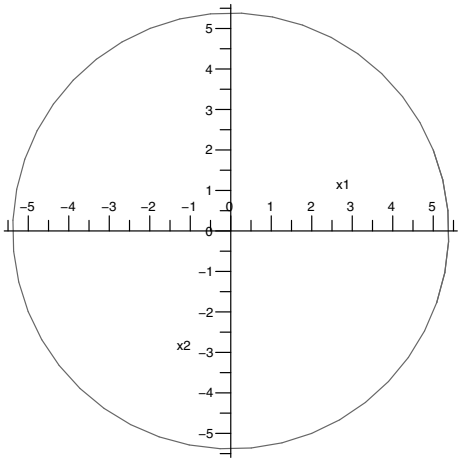
```
> numlsg2:=dsolve({bsp2,ab2num}, numeric,
          range=0..20);
```

```
> odeplot(numlsg2,[[t,x1(t),color=blue,style=point],
          [t,x2(t),color=red,style=line]],
          0..10, numpoints=200);
```



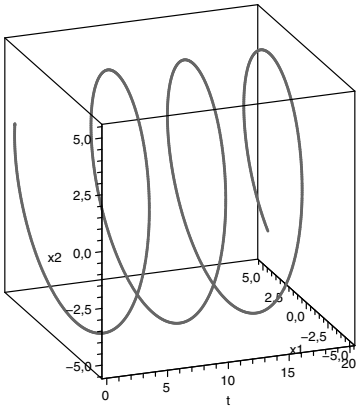
erhält man die Darstellung von $x_1(t)$ und $x_2(t)$ in Abhängigkeit von t. Die Phasenportrait-Darstellung erzeugt man mit

```
> odeplot(numlsg2,[x1(t),x2(t)],0..7);
```



und die dreidimensionale Darstellung mit

```
> odeplot(numlsg2,[t,x1(t),x2(t)],refine=2,axes=boxed,
        labels=["t","x1","x2"],orientation=[-111,70],
        thickness=2);
```



A.5. Weitere Routinen

Wie eingangs bereits erwähnt kann dieser Abschnitt nur eine kleine Auswahl der Möglichkeiten von MAPLE zur Lösung und Behandlung von Differentialgleichungen erläutern. Wir hoffen, dass die bis hier behandelten Themen einen Einstieg ermöglichen und dazu motivieren, sich die weiteren Möglichkeiten von MAPLE durch Studium der sehr ausführlichen Hilfe-Seiten oder weiterer Literatur selbst zu erarbeiten. Im Folgenden geben wir noch einige Anregungen dazu.

Interaktiver Modus von `dsolve`. Die `dsolve`-Anweisung bietet neben den hier beschriebenen Aufrufen innerhalb eines Worksheets auch eine interaktive Benutzeroberfläche, mit der menügesteuert auf praktisch alle Funktionalitäten inklusive der graphischen Darstellung zugegriffen werden kann. Der Aufruf lautet z.B. für `bsp1`

```
> dsolve[interactive]({bsp1,ab1num});
```

es ist aber auch möglich, die Oberfläche ganz ohne vorab definierte Gleichungen mittels `dsolve[interactive]();` aufzurufen und Differentialgleichung und Anfangsbedingung interaktiv einzugeben. Mittels der Menüpunkte „solve numerically“ und „solve symbolically“ kommt man in die entsprechenden Untermenüs, in denen die Lösung berechnet und graphisch dargestellt werden kann. In diesen Fenstern kann man mit dem Menüpunkt „On Quit Return . . . “ auswählen, was nach Beendigung mittels „Quit“ im Worksheet ausgegeben werden soll. Beispielsweise gibt die Auswahl „On Quit Return Maple Commands“ nach dem Schließen der

Oberfläche alle MAPLE-Anweisungen aus, die intern innerhalb der Oberfläche ausgeführt wurden und erlaubt so, die interaktiv durchgeführten Berechnungen im Worksheet nachzuvollziehen.

Das `DEtools`-Paket. Das `DEtools`-Paket liefert über die bereits vielfältigen Möglichkeiten des `dsolve`-Befehls hinaus eine große Anzahl weiterer Routinen zur analytischen Behandlung und graphischen Darstellung gewöhnlicher Differentialgleichungen. Insbesondere finden sich dort auch viele Routinen, bei denen nicht alleine die Berechnung von Lösungen im Vordergrund steht. Wie bei allen MAPLE-Paketen stehen die Routinen nach Eingabe von `with(DEtools);` zur Verfügung. Im Folgenden stellen wir einige dieser Routinen kurz anhand von Beispielen vor.

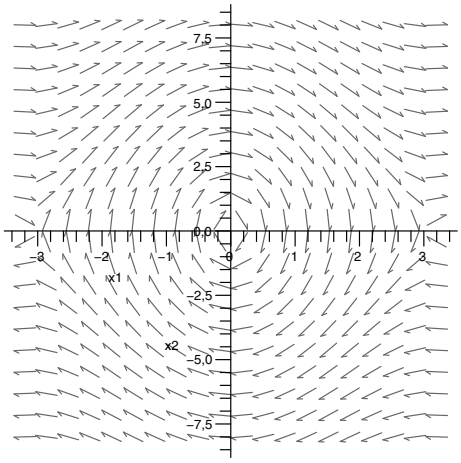
`dfieldplot` ist eine Routine mit der zweidimensionale Vektorfelder mit Pfeilen dargestellt werden können. Wir illustrieren die Routine anhand unserer Pendelgleichung ohne Reibung (1.5), die wir in MAPLE eingeben als

```
> pendel := diff(x1(t),t)=x2(t),
          diff(x2(t),t)=-9.81*sin(x1(t));
```

Der Aufruf für das Phasenportrait lautet dann

```
> dfieldplot([pendel],[x1(t),x2(t)], t=0..1,
          x1=-3.2..3.2, x2=-8..8);
```

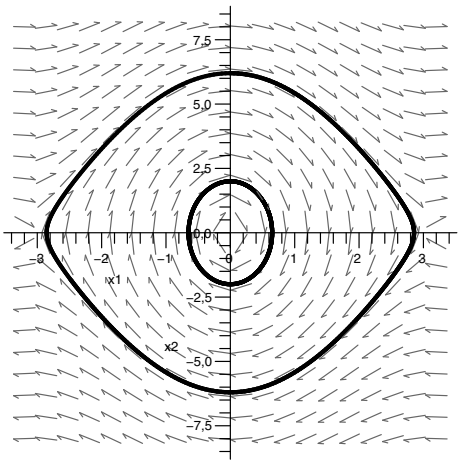
und liefert das Bild



Beachte, dass aus syntaktischen Gründen ein Bereich für t angegeben werden muss, auch wenn das Vektorfeld wie hier gar nicht von t abhängt.

`phaseportrait` ist eine Erweiterung von `dfieldplot` und erlaubt zusätzlich die gleichzeitige Darstellung von Lösungen für mehrere Anfangswerte in Form eines Phasenportraits. Diese Lösungen werden intern mit den bereits beschriebenen Routinen numerisch berechnet. Für das Pendel mit den Anfangswerte $(0, 6.2)^T$ und $(0, 2)^T$ erhalten wir

```
> phaseportrait([pendel],[x1(t),x2(t)], t=0..7,
        [[x1(0)=0,x2(0)=6.2],[x1(0)=0,x2(0)=2]],
        x1=-3.2..3.2, x2=-8..8,
        stepsize=0.01, linecolor=black););
```



Zu beachten ist hier, dass die Auflösung der Lösungskurven durch `stepsize` und nicht wie bei `plot` oder `odeplot` mit `numpoints` gewählt wird. Ohne Angabe eines hinreichend kleinen Wertes für `stepsize` tendieren die Ausgaben von `phaseportrait` dazu, etwas „ruckelig“ auszusehen. Sollen die Lösungen eines Phasenportraits ohne Pfeile ausgegeben werden, muss die Option `arrows=none` angegeben werden. Neben `phaseportrait` existiert noch die Routine `DEplot`, die aber im Wesentlichen äquivalent zu `phaseportrait` ist.

`odeadvisor` bietet eine Möglichkeit, den Typ einer Differentialgleichung zu erkennen. Die Beschreibung der von MAPLE verwendeten Klassifizierung kann über die Hilfe-Seite im Detail eingesehen werden. Beispielsweise sind Bernoulli-Differentialgleichungen ein Typ der Klassifizierung, wie das folgende Beispiel für die allgemeine Bernoulli-Gleichung (5.12) zeigt.

```
> dgl := diff(x(t),t)=a(t)*x(t)+b(t)*x(t)^alpha;
```

```
> odeadvisor(dgl);
```

$$[\_Bernoulli]$$

`intfactor` findet integrierende Faktoren für nicht exakte Differentialgleichungen. Für die gerade definierte Bernoulli-Gleichung `dgl` erhalten wir damit

```
> intfactor(dgl);
```

$$e^{(\alpha-1)\int a(t)dt}\left(x\left(t\right)\right)^{-\alpha}$$

also bis auf einen konstanten Vorfaktor genau (5.13).

`firint` schließlich findet Stammfunktionen exakter Differentialgleichungen. Der Name erinnert an „erstes Integral“ („first integral“), das in unserem Buch eine andere Bedeutung hat, welche aber eine enge Beziehung zu den Stammfunktionen besitzt, vgl. dazu die Diskussion zur Berechnung erster Integrale nach Satz 5.11. Multipliziert mit dem durch `intfactor(dgl)` berechneten integrierenden Faktor ist `dgl` exakt, weswegen `intfactor(dgl)*dgl` eine Stammfunktion besitzt. Diese wird berechnet durch

```
> firint(intfactor(dgl)*dgl);
```

$$-\frac{\left(x\left(t\right)\right)^{-\alpha+1}e^{(\alpha-1)\int a(t)dt}}{\alpha-1}-\int b\left(t\right)e^{(\alpha-1)\int a(t)dt}dt+\_C1=0$$

ANHANG B

Matlab

MATLAB ist ein auf numerische Rechnungen spezialisiertes Computermathematiksystem. Die Ursprünge von MATLAB (= „matrix laboratory“) liegen in der Matrizenrechnung und der numerischen linearen Algebra, allerdings stehen seit geraumer Zeit auch vielfältige und sehr leistungsfähige Algorithmen zur Lösung gewöhnlicher Differentialgleichungen in MATLAB zur Verfügung. Ebenfalls bietet MATLAB Erweiterungen (sogenante Toolboxes) an, in denen symbolische Methoden zur Verfügung stehen, diese sind jedoch weniger komfortabel zu bedienen als in MAPLE, weswegen wir uns hier auf die Beschreibung der numerischen Routinen beschränken. In diesem Bereich sind die Routinen in MATLAB unserer Erfahrung nach zumeist schneller als in MAPLE, zudem ist die Implementierung näher an den numerischen Algorithmen, deren Funktionsweise dadurch transparenter wird.

Alle im Folgenden aufgeführten Beispiele wurden mit MATLAB 7.2.0.294 (R2006a) getestet und stehen im Internet unter unter `http://www.dgl-buch.de` als M-Files zum Download zur Verfügung. Zur weitergehenden Einführung in MATLAB gibt es viel Literatur, wie z.B. der MATLAB-Guide von D. und N. Higham [**10**] oder das Skript von J. Behrens und A. Iske [**3**]. Speziell mit dynamischen Systemen und MATLAB beschäftigt sich das Buch von S. Lynch [**16**].

B.1. Grundlegendes

M-Files. Im Gegensatz zu MAPLE, in dem die Operationen zumeist interaktiv in Worksheets ausgeführt und als solche gespeichert werden, ähnelt MATLAB mehr einer Programmiersprache, in der die einzelnen Funktionen in kleinen Programmdateien, den sogenannten M-Files implementiert werden. Auch wenn in MATLAB viele Definitionen auch interaktiv im Kommandofenster (Command Window) durchgeführt werden können (was in etwa der Vorgehensweise in MAPLE entspricht), ist es

meist einfacher und bequemer, alle wesentlichen Definitionen in M-Files vorzunehmen.

Nach dem Start von MATLAB wird der M-File Editor mit `File->New->M-File` oder durch Öffnen eines bestehenden M-Files mittels `File->Open` geöffnet. In einem M-File kann nun entweder eine Folge von Anweisungen eingegeben werden oder es können ein oder mehrere MATLAB-Funktion mit Ein- und Ausgabeparametern definiert werden. Im ersten Fall spricht man von einem Skript, im zweiten von einer Funktion. Speichert man das M-File eines Skripts z.B. unter dem Namen `meinmfile.m`, so kann es im Command Window oder von einem anderen M-File aus einfach durch Eingabe von `meinmfile` – gegebenenfalls mit Funktionsparametern – aufgerufen werden.

Wichtig ist dabei, dass MATLAB das Verzeichnis „kennt“, in dem `meinmfile.m` abgelegt ist. Dies kann am einfachsten dadurch erreicht werden, dass dieses Verzeichnis im Adressfenster in der rechten Hälfte der Kopfzeile eingegeben wird. Diese Methode hat aber den Nachteil, dass MATLAB das Verzeichnis bei Eingabe eines neuen Verzeichnisses oder nach einem Neustart wieder „vergisst“. Soll sich MATLAB das Verzeichnis dauerhaft merken, so muss dieses unter `File->Set Path` dem sogenannten Pfad (Path) hinzugefügt werden. Wird der Pfad anschließend im `Set Path`-Menu mit `save` gespeichert, so ist das angegebene Verzeichnis auch nach einem Neustart von MATLAB bekannt.

Mathematische Operatoren. In MATLAB gibt es für Multiplikation, Division und Potenzieren je zwei verschiedene Varianten, nämlich `a*b`, `a/b` und `a^b` sowie `a.*b`, `a./b` und `a.^b`. Dies macht dann einen Unterschied, wenn es sich bei `a` und/oder `b` um Vektoren oder Matrizen handelt: im ersten Fall wird nämlich stets versucht, entsprechende Matrizenoperationen durchzuführen, während die mit dem Punkt versehenen Operationen stets komponentenweise zu verstehen sind. Wenn nicht explizit Matrizenrechnung verwendet werden soll, empfiehlt es sich in MATLAB stets, die mit dem Punkt versehenen Operatoren zu verwenden.

Eingabe von Differentialgleichungen. Nun kommen wir zur eigentlichen Implementierung von Differentialgleichungen in MATLAB. Wir betrachten dazu die bereits aus den MAPLE-Beispielen bekannten Gleichungen (A.1)

$$\dot{x}(t) = x(t)^2$$

mit $x(t) \in \mathbb{R}$ und (A.2)

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x.$$

mit $x(t) \in \mathbb{R}^2$. Da Gleichungen höherer Ordnung in MATLAB nicht direkt eingegeben werden können, betrachten wir Gleichung (A.3) hier nicht.

Zur numerischen Lösung dieser Gleichungen in MATLAB werden die zugehörigen Vektorfelder f, also

$$f(t, x) = x^2 \tag{B.1}$$

und

$$f(t, x) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x = \begin{bmatrix} x_2 \\ -x_1 \end{bmatrix} \tag{B.2}$$

als MATLAB-Funktionen in jeweils einem M-File eingegeben. Für (B.1) leistet dies das folgende M-File `bsp1_f.m`:

```
function y = bsp1_f(t,x)

y = x.^2;
```

Vom Command Window aus kann die Funktion nun z.B. mit `bsp1_f(0,3)` aufgerufen werden, was die Ausgabe `ans = 9` erzeugt.

Die lediglich zwei Zeilen der Funktion haben die folgende Bedeutung: Die erste Zeile legt fest, dass hier eine MATLAB-Funktion mit Namen `bsp1_f` definiert wird, die zwei Eingabeparameter t und x und einen Ausgabeparameter y besitzt. In der zweiten Zeile wird dann die Rechenregel für $y = f(t, x)$ definiert.

Hierbei sind verschiedene Sachen zu beachten:

- Der Name `bsp1_f` taucht hier zweimal auf, nämlich zum einen im Dateinamen des M-Files und zum anderen in der Deklaration der Funktion. Diese beide Namen sollten übereinstimmen, tun sie das aber nicht, gibt es keine Fehlermeldung. Statt dessen ist in diesem Fall der Dateiname ausschlaggebend: Eine Funktion, die im M-File `funktion1.m` gespeichert ist, die in der Deklaration aber z.B. `funktion2` heißt, kann nur unter dem Namen `funktion1` aber nicht mit `funktion2` aufgerufen werden.
- In einem M-File können verschiedene Funktionen nacheinander deklariert werden, dabei ist aber nur die erste „nach außen“ sichtbar, die weiteren können nur innerhalb dieses M-Files verwendet werden.

- Auch wenn das Vektorfeld – wie hier – nicht von t abhängt, muss es immer als Funktion in den zwei Variablen t und x deklariert werden, da ansonsten das Zusammenspiel mit den Differentialgleichungslösern in MATLAB nicht funktioniert. Die Zeitvariable t ist hierbei immer eindimensional, der Zustand x kann ein Vektor sein, vgl. die unten stehende Funktion für das Vektorfeld (B.2).

Für unser zweites Beispiel (B.2) lautet das entsprechende M-File `bsp2_f.m`:

```
function y = bsp2_f(t,x)

y = zeros(2,1);

y(1) =   x(2);
y(2) =  -x(1);
```

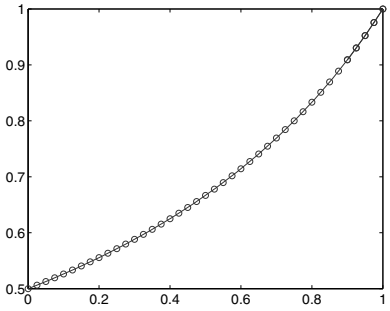
Statt der eindimensionalen Ein- und Ausgabeparameter x und y haben wir es nun mit Vektoren zu tun, deren Komponenten in MATLAB mittels Indizes in runden Klammern angesprochen werden. Zudem muss hier vor der Zuweisung der Werte die Anweisung `y = zeros(2,1);` stehen, damit die Variable y korrekt als zweidimensionaler Spaltenvektor definiert wird. In MATLAB ist ein solcher Spaltenvektor äquivalent zu einer 2×1 Matrix, welche mit der Anweisung `zeros(2,1)` (mit Nulleinträgen) erzeugt wird.

B.2. Numerische Lösungen

Der Standardlöser `ode45`. Die Standardroutine zur Lösung gewöhnlicher Differentialgleichungen in MATLAB ist die Routine `ode45`, hinter der sich ein von Dormand und Prince vorgeschlagenes Verfahren verbirgt, nämlich ein eingebettetes explizites Runge-Kutta-Verfahren mit Konsistenzordnungen 4 und 5. Beim Aufruf müssen der Funktionsname des Vektorfeldes mit vorangestelltem „`@`“, das Zeitintervall, auf dem die Gleichung gelöst werden soll, sowie die Anfangsbedingung angegeben werden. Als Ergebnis liefert die Routine das per Schrittweitensteuerung erzeugte Zeitgitter $\mathcal{T} = (t_0, t_1, \ldots)$, auf dem die Gleichung gelöst wurde, und die approximativen Lösungswerte $\tilde{x}(t_i)$ an den Gitterpunkten. Für unser erstes Beispiel (B.1) lautet der Aufruf zu Lösung der Gleichung auf dem Intervall $[0, 1]$ mit Anfangswert $x(0) = 0.5$

```
ode45(@bsp1_f, [0, 1], 0.5)
```

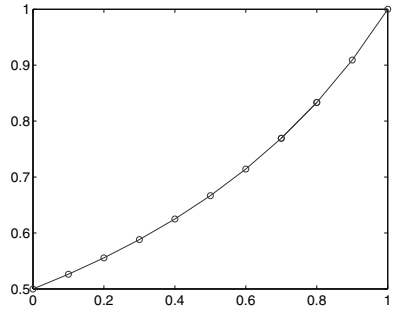
Wird dieser Befehl in dieser Form im Command Window eingegeben, so erzeugt MATLAB automatisch die folgende grafische Darstellung der Lösung.



Das hier sichtbare Gitter entspricht dem bei der Berechnung von der Schrittweitensteuerung verwendeten Gitter, das aber für die grafische Darstellung vierfach verfeinert wurde, d.h. zwischen je zwei Berechnungspunkte wurden in der Ausgabe je drei weitere Punkte gesetzt. Diese Verfeinerung kann man mit der `refine`-Option steuern, wie unten im Abschnitt „Optionen“ beschrieben. Alternativ kann man statt des Intervalls $[0, 1]$ ein Gitter vorgeben, was bewirkt, dass die approximierten Lösungswerte an diesen Gitterpunkten ausgegeben werden. Eindimensionale Gitter definiert man in MATLAB einfach mittels der Anweisung `[anfang:schrittweite:ende]`, der Aufruf

```
ode45(@bsp1_f, [0:0.1:1], 0.5)
```

liefert daher die Ausgabe auf dem gröberen Gitter $\{0, 0.1, 0.2, \ldots, 1\}$:



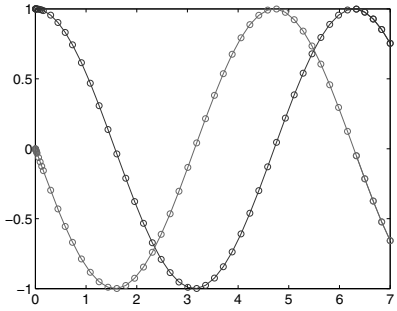
Zu beachten ist dabei, dass hiermit nur das Ausgabegitter und nicht das Rechengitter gesteuert wird. Das Rechengitter wird in MATLAB stets

per Schrittweitensteuerung mit den unten im Abschnitt „Optionen“ beschriebenen Fehlertoleranzen erzeugt.

Für unser zweites Beispiel funktioniert die Lösung z.B. für das Intervall $[0, 7]$ und den Anfangswert $x(0) = (1, 0)^T$ analog mit dem Aufruf

```
ode45(@bsp2_f, [0, 7], [1, 0])
```

und liefert



Oft möchte man statt einer grafischen Darstellung lieber die Werte t_i und $\tilde{x}(t_i)$ in Variablen geliefert bekommen, um diese weiterzuverarbeiten oder – vgl. dazu den Abschnitt B.3 – mittels eigener Plot-Anweisungen auf andere Arten darzustellen. Dazu schreibt man für das erste Beispiel z.B.

```
[t,x] = ode45(@bsp1_f, [0, 1], 0.5)
```

Damit erhält man die (hier gekürzt wiedergegebene) Ausgabe

```
t =
         0
    0.0250
    0.0500
    ...
    0.9500
    0.9750
    1.0000

x =
    0.5000
    0.5063
    0.5128
    ...
    0.9524
```

```
0.9756
1.0000
```

Wie bei allen MATLAB-Anweisungen kann man diese Ausgabe durch ein Semikolon nach der Anweisung unterdrücken. Für `bsp2_f` funktioniert dies analog mittels

```
[t,x] = ode45(@bsp2_f, [0, 7], [1, 0]);
```

mit dem Unterschied, dass x nun ein Vektor von Vektoren im $\mathbb{R}^2$ ist, was in MATLAB durch eine zweispaltige Matrix dargestellt wird, in der jede Zeile den Vektor des approximativen Lösungswertes zum entsprechenden Zeitpunkt enthält.

Optionen. Der Routine `ode45` kann (ebenso wie allen anderen im Folgenden vorgestellten Differentialgleichungslösern) eine Reihe von Optionen übergeben werden. Dazu wird mittels der Anweisung `odeset` zunächst eine Strukturvariable `options` mit den gewünschten Optionen erzeugt, die dann als weiteres Argument an `ode45` übergeben wird.

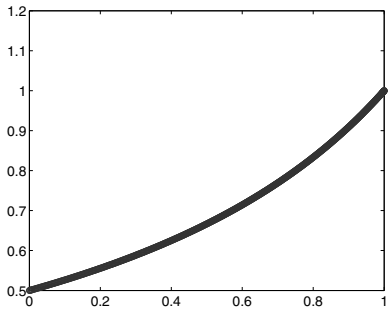
Wir illustrieren dieses Vorgehen anhand der in der Schrittweitensteuerung verwendeten Genauigkeiten `AbsTol` und `RelTol`, welche die analoge Bedeutung zu `abserr` und `relerr` in MAPLE haben, vgl. Formel (A.4)[1], und mit 10^{-3} und 10^{-6} voreingestellt sind. Wollen wir z.B. `bsp1_f` mit relativer und absoluter Toleranz 10^{-13} lösen, so lautet die entsprechende Anweisungsfolge

```
options = odeset('RelTol',1e-13);
options = odeset(options,'AbsTol',1e-13);

ode45(@bsp1_f, [0,1],0.5, options);
```

mit der Ausgabe

[1]Wie in der Fußnote auf Seite 203 bereits erwähnt, hält nur der *geschätzte* aber nicht unbedingt der *tatsächliche* Fehler diese Schranke ein.



Da bei MATLAB – im Gegensatz zu MAPLE – das von der Schrittweitensteuerung erzeugte Gitter in der Standard-Grafikdarstellung (vierfach verfeinert) sichtbar ist, sieht man hier deutlich den Effekt der Erhöhung der Genauigkeit: das Gitter ist im Vergleich zur Darstellung auf Seite 219 deutlich feiner geworden.

Das obige Beispiel zeigt, wie man mit `odeset` verschiedene Optionen beeinflusst: Beim ersten Aufruf wird lediglich der Name der zu ändernden Option und der gewünschte Wert angegeben und der Variablen `options` zugewiesen. Um weitere Optionen zu setzen bzw. zu ändern, wird danach bei jedem weiteren Aufruf die bereits definierte Optionsvariable `options` als erstes Argument an `odeset` übergeben.

Neben den Genauigkeiten gibt es eine Reihe von weiteren Optionen, die z.B. das Ausgabeverhalten der Routine oder weitere Details der Schrittweitensteuerung beeinflussen. Ausführliche Informationen hierüber finden sich in der Hilfe-Seite von `odeset` in MATLAB.

Erwähnt sei hier nur noch die `refine`-Option, mit der gesteuert wird, wie viele Gitterpunkte dem Rechengitter für die Ausgabe hinzugefügt werden. Der Parameter ist für `ode45` wie oben bereits erwähnt auf 4 voreingestellt, für alle im Folgenden beschriebenen anderen Verfahren auf 1. Will man also in `ode45` das wirkliche Rechengitter in der Ausgabe sehen, so muss

```
options = odeset(options,'refine',1);
```

gesetzt werden.

Andere numerische Verfahren. Neben dem Standardlöser `ode45` bietet MATLAB eine Reihe weiterer numerischer Verfahren an, von denen hier nur zwei erwähnt seien, die als Alternative zu `ode45` nützlich sein können:

Unter `ode113` steht ein kombiniertes Adams-Bashforth-Moulton Mehrschrittverfahren (ein sogenanntes Prädiktor-Korrektor-Verfahren) mit variabler Konsistenzordnung bis hin zu $p = 13$ zur Verfügung, das bei sehr hohen Genauigkeiten effizienter als `ode45` sein kann. So liefert z.B. der Aufruf

```
ode113(@bsp1_f, [0,1], 0.5, options);
```

mit den oben definierten Optionen ein deutlich gröberes Gitter, da `ode113` in jedem Zeitschritt genauer ist und die Schrittweitensteuerung daher größere Schritte wählen kann.

Mittels `ode15s` steht ein implizites Mehrschrittverfahren zur Verfügung, das eingesetzt werden sollte, wenn die zu lösende Differentialgleichung steif ist. Ob eine steife Gleichung vorliegt, kann man zumeist daran erkennen, dass `ode45` sehr kleine Schrittweiten wählt und daher sehr viel Rechenzeit braucht, obwohl die Lösung annähernd konstant verläuft.

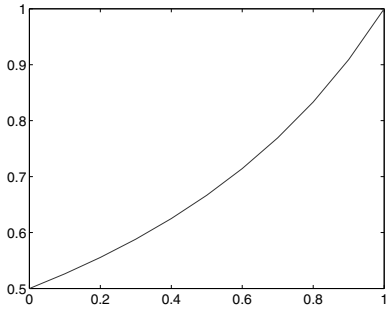
Für weitere Routinen siehe die Hilfe-Seite zu `ode45`, auf der alle Verfahren beschrieben sind. Alle numerischen Verfahren werden in MATLAB auf die gleiche Art und Weise angesprochen, so dass ein Verfahren leicht gegen ein anderes ausgetauscht werden kann.

B.3. Grafische Ausgabe

Die von `ode45` und den anderen Lösern direkt erzeugte grafische Ausgabe kann mittels der `OutputFcn`-Option auf verschiedene Weise gestaltet werden, vgl. die Hilfeseite zu `odeset`. In diesem Abschnitt beschreiben wir eine andere Vorgehensweise, in der wir das Gitter und die auf diesem berechneten approximativen Lösungwerte zunächst in zwei Variablen speichern und dann mit den Standard-Plot-Routinen von MATLAB grafisch darstellen. Auf diese Weise können wir alle in MATLAB zur Verfügung stehenden Methoden zur Bearbeitung von Grafiken für unsere Lösungsdarstellung verwenden.

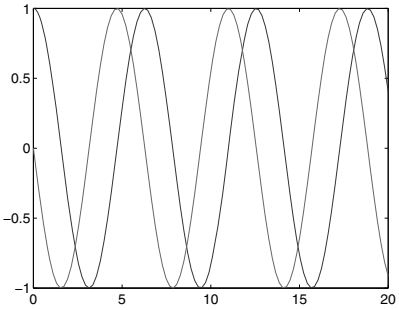
Grundlegende Plot-Anweisungen. Im einfachsten Fall geht dies mit den Anweisungen

```
[t1,x1] = ode45(@bsp1_f, [0:0.1:1],0.5);
plot(t1,x1);
```



bzw.

```
[t2,x2] = ode45(@bsp2_f, [0,20],[1, 0]);
plot(t2,x2);
```

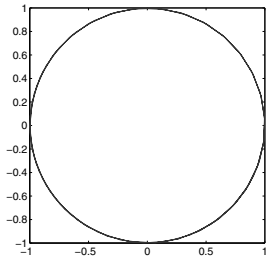


Der `plot`-Befehl in MATLAB stellt die auf den Gittern `t1` bzw. `t2` definierten Daten `x1` bzw. `x2` als interpolierte Funktionen in Abhängigkeit von t dar und behandelt die zweidimensionalen Datenpunkte in der Matrix `x2` automatisch als zwei separate reellwertige Funktionen. Zu beachten ist dabei, dass der `plot`-Anweisung stets diskrete Daten in Form von Gittern und auf diesen definierten Werten übergeben werden, also Vektoren oder Matrizen. Dies bedeutet insbesondere, dass die Auflösung allein durch die übergebenen Daten bestimmt ist und in der `plot`-Anweisung selbst nicht beeinflusst werden kann. In der Standard-Einstellung werden die Gitterpunkte (im Gegensatz zu den mittels `ode45` direkt erzeugten Grafiken) nicht markiert, dies kann aber wie unten im Abschnitt „Plotoptionen“ beschrieben geändert werden.

Will man diese zweidimensionale Funktion als Phasenportrait darstellen, so muss man die zweite Komponente in `x2` in Abhängigkeit der ersten plotten. Die Komponenten der zweispaltigen Matrix `x2` werden in MATLAB mittels `x2(i,j)` angesprochen, wobei der erste Index `i` der

Zeile, also dem Zeitpunkt im Gitter, und der zweite Index `j` der Spalte, also der Lösungskomponente entspricht. Die gesamte j-te Spalte von `x2` erhält man mit `x2(:,j)`. Daraus ergibt sich, dass das Phasenportrait mit der Anweisung

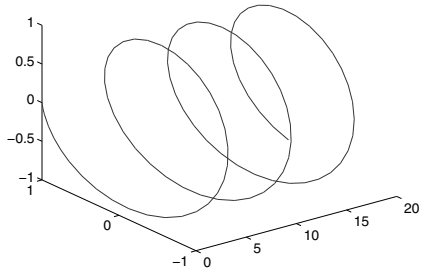
```
plot(x2(:,1),x2(:,2));
```



erzeugt wird.

Für eine dreidimensionale Darstellung dient die Anweisung `plot3`, der drei Vektoren übergeben werden müssen: das Zeitgitter sowie die erste und zweite Komponente der Lösung. Folglich erzeugt man die dreidimensionale Darstellung mit

```
plot3(t2,x2(:,1),x2(:,2));
```



In einer `plot`-Anweisung können mehrere Datensätze auf einmal dargestellt werden. So kann man die obige Darstellung von `x2` in Abhängigkeit von `t2` alternativ mit

```
plot(t2,x2(:,1),t2,x2(:,2));
```

erzeugen. Diese Form ist sinnvoll, wenn der Zeichenstil der beiden Lösungskomponenten unabhängig voneinander verändert werden soll, wie im übernächsten Abschnitt „Plotoptionen“ beschrieben.

Grafikfenster. In der Voreinstellung zeichnet MATLAB jede Grafik in das Koordinatensystem des aktuellen Grafikfensters und löscht dabei alle bereits darin dargestellten Objekte. Existiert kein aktuelles Grafikfenster, so wird beim Ausführen der `plot`-Anweisung ein neues Fenster geöffnet.

Möchte man mit einer `plot`-Anweisung zeichnen, ohne die bereits im aktuellen Fenster dargestellten Objekte zu löschen, so kann dies mit der Anweisung `hold on` erreicht werden. Alle dieser Anweisung folgenden `plot`-Anweisungen werden ausgeführt, ohne dass das Fenster vorher gelöscht wird, so dass man beliebig viele Lösungen in ein Fenster zeichnen kann. Mit `hold off` wird dieser Modus wieder ausgeschaltet. Die oben vorgestellte Anweisung

```
plot(t2,x2(:,1),t2,x2(:,2));
```

kann daher alternativ auch als

```
plot(t2,x2(:,1));
hold on;
plot(t2,x2(:,2));
hold off;
```

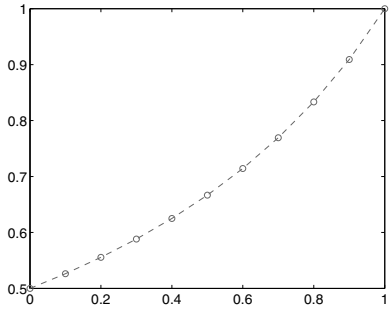
ausgeführt werden.

Mittels `figure` wird ein neues Grafikfenster geöffnet und als aktuelles Grafikfenster für die nachfolgenden `plot`-Befehle aktiviert. Soll eine `plot`-Anweisung in ein anderes als das aktuelle Fenster plotten, so kann man jederzeit mittels der Anweisung `h = gca;` einen Zeiger auf das Koordinatensystems des aktuellen Grafikfensters (den sogenannten „Handle“ der „CurrentAxes“) in der Variablen h speichern. Dann kann man im Folgenden mittels `plot(h,...)` die Grafik gezielt in dieses Fenster (an Stelle des aktuellen) umleiten.

Plotoptionen. MATLAB bietet vielfältige Möglichkeiten, das Aussehen des Plots, den Zeichenstil sowie die Beschriftung der Grafik zu beeinflussen. Wir geben hier nur die drei wesentlichen Möglichkeiten anhand von Beispielen wieder und verweisen wiederum auf die Hilfe-Seiten von MATLAB für die vollständigen Informationen.

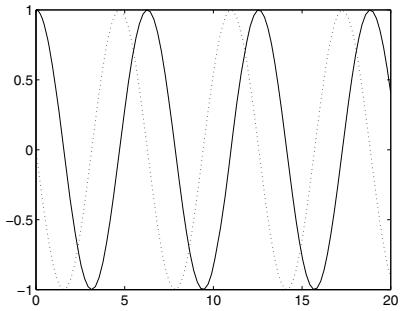
Die einfachste Art, die Darstellung zu beeinflussen, geschieht über die sogenannten LineSpec-Tripel. Mit diesen können die Farbe, der Zeichenstil der Linie und die Markierung der Gitterpunkte bestimmt werden. Z.B. bewirkt das Tripel `'r--o'`, dass die Linie in Rot (r) und gestrichelt (--) gezeichnet wird und die Gitterpunkte mit kleinen Kreisen (o) dargestellt werden, also

```
plot(t1,x1,'r--o');
```



Diese LineSpec-Festlegung kann in der `plot`-Anweisung nach jedem zu plottenden Datensatz angegeben werden, also z.B.

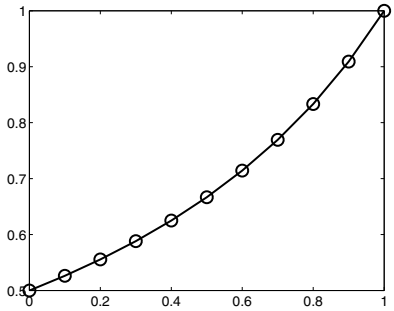
```
plot(t2,x2(:,1),'-k',t2,x2(:,2),':k');
```



Hier werden beide Lösungskomponenten in Schwarz (k) dargestellt, die erste durchgezogen (-) und die zweite gepunktet (:). Da keine Markierung für die Gitterpunkte angegeben wurde, werden diese – gemäß der Voreinstellung – nicht dargestellt.

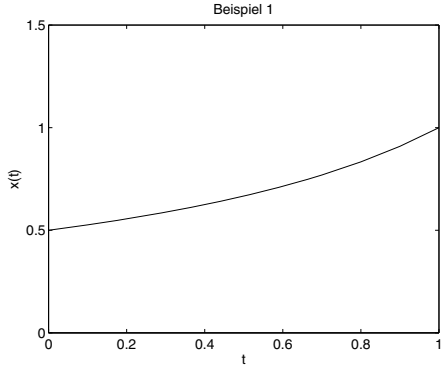
Die zweite Möglichkeit, eine Grafik zu beeinflussen, bieten die sogenannten Line-Properties, die der `plot`-Anweisung übergeben werden können. Mit diesen kann z.B. die Liniendicke und die Größe der Gitterpunkt-Markierungen wie folgt beeinflusst werden:

```
plot(t1,x1,'k-o','LineWidth',2,'MarkerSize',12);
```



Die dritte Möglichkeit, eine Grafik zu bearbeiten, bilden verschiedene Anweisungen, die nach Erzeugen der Grafik ausgeführt werden. Mit diesen kann das Koordinatensystem verändert sowie Achsenbeschriftungen und Titel hinzugefügt werden. Die folgende Anweisungsfolge erzeugt die nachfolgend dargestellte Grafik mit Beschriftung und geändertem Koordinatensystem.

```
plot(t1,x1,'k-');
xlabel('t');
ylabel('x(t)');
title('Beispiel 1');
axis([0, 1, 0, 1.5]);
```



Analog zur `plot`-Anweisung kann auch diesen Anweisungen ein Handle auf ein anderes Koordinatensystem als das im aktuellen Grafikfenster übergeben werden.

Neben der in diesem Beispiel verwendeten expliziten Angabe eines Koordinatensystems stehen weitere `axis`-Funktionen zur Verfügung, z.B. `axis tight`, mit der das Koordinatensystem genau an die dargestellten Objekte angepasst wird, oder `axis square`, mit der die Darstellung so

umskaliert wird, dass die Achsen ein Quadrat bilden. Diese Anweisung wurde z.B. für das Phasenportrait auf Seite 225 verwendet.

Über die beschriebene kommandogesteuerte Bearbeitung von Grafiken hinaus gibt es die Möglichkeit, praktisch sämtliche Eigenschaften einer Grafik mittels des interaktiven Property-Editors zu ändern. Diesen öffnet man, indem man im Menü des Grafik-Fensters den Pfeil („Edit Plot“) auswählt und das zu bearbeitende Objekt doppelt anklickt.

ANHANG C

Matrixnormen

In diesem Anhang fassen wir einige grundlegende Begriffe und Zusammenhänge über Normen, insbesondere Matrixnormen, zusammen.

Definition C.1 (Norm)**:** Es sei X ein reeller Vektorraum. Eine Abbildung $\|\cdot\| : X \to [0, \infty]$ heißt *Norm*, falls für $x, y \in X$ gilt:

(1) $\|x\| = 0$ genau dann, wenn $x = 0$ (*Definitheit*),
(2) $\|x + y\| \leq \|x\| + \|y\|$ (*Dreiecksungleichung*),
(3) $\|ax\| = |a|\|x\|$ für alle $a \in \mathbb{R}$ (*Homogenität*).

Typische Normen (*Vektornormen*) auf dem $\mathbb{R}^d$ sind

- die *p-Norm*

$$\|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{\frac{1}{p}},$$

insbesondere die 1-Norm

$$\|x\|_1 = \sum_{i=1}^{d} |x_i|$$

und die 2-Norm (oder *euklidische Norm*)

$$\|x\|_2 = \sqrt{\sum_{i=1}^{d} |x_i|^2},$$

- sowie die *Maximums-Norm*

$$\|x\|_\infty = \max_{i=1,\ldots,d} |x_i|.$$

Abbildung C.1 zeigt die Einheitskugeln $B_0(1) = \{x \in \mathbb{R}^2 \mid \|x\| \leq 1\}$ in der 1-, der 2- sowie der ∞-Norm.

Auch der Raum $\mathbb{R}^{n \times d}$ aller reellen $n \times d$-Matrizen ist ein Vektorraum und lässt sich mit einer Norm versehen (einer *Matrixnorm*).

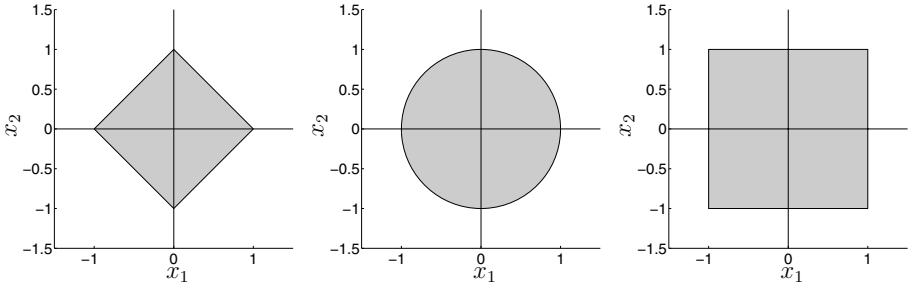


ABBILDUNG C.1: Einheitskugeln in der 1- (links), der 2- (Mitte) und der ∞-Norm (rechts) im $\mathbb{R}^2$

Definition C.2: Eine Matrixnorm heißt *verträglich* bzw. *konsistent* mit einer Vektornorm, falls

$$\|Ax\| \leq \|A\|\|x\|$$

für alle $x \in \mathbb{R}^d$.

Satz C.3: Zu einer Vektornorm $\|\cdot\|$ ist

$$\|A\| = \sup_{y \neq 0} \frac{\|Ay\|}{\|y\|} = \sup_{\|y\|=1} \|Ay\|$$

eine verträgliche Matrixnorm, die sog. *natürliche* oder *induzierte* Matrixnorm.

BEWEIS. Für $x = 0$ ist die Verträglichkeit klar, weil $\|Ax\| = 0 = \|x\|$ gilt. Für $x \neq 0$ gilt

$$\|A\| = \sup_{y \neq 0} \frac{\|Ay\|}{\|y\|} \geq \frac{\|Ax\|}{\|x\|}$$

und daher $\|A\|\|x\| \geq \|Ax\|$, also die Verträglichkeit.

Es bleibt zu zeigen, dass $\|A\|$ tatsächlich eine Norm ist, d.h. wir müssen Definition C.1 (1)–(3) nachweisen:
(1) Es gilt $\|A\| = 0$ genau dann, wenn $\|Ax\| = 0$ für alle $x \neq 0$, und dies wiederum genau dann, wenn $Ax = 0$ für alle $x \neq 0$. Letzteres ist aber äquivalent zu $A = 0$.
(2) Aufgrund der Verträglichkeit der Matrixnorm gilt

$$\|Ax\| \leq \|A\|\|x\|$$

für alle $x \in \mathbb{R}^d$, also

$$\|A + B\| = \sup_{\|x\|=1} \|Ax + Bx\| \leq \sup_{\|x\|=1} (\|Ax\| + \|Bx\|) \leq \|A\| + \|B\|.$$

(3) Es gilt

$$\|aA\| = \sup_{\|x\|=1} \|aAx\| = |a| \sup_{\|x\|=1} \|Ax\| = |a|\|A\|.$$

□

Die von den eingangs erwähnten Vektornormen induzierten Matrixnormen sind (vgl. z.B. Quarteroni et al. [**19**])

(1) die 1-Norm (oder *Spaltensummennorm*)

$$\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^{d} |a_{ij}|,$$

(2) die Maximums-Norm (oder *Zeilensummennorm*)

$$\|A\|_\infty = \max_{i=1,\dots,d} \sum_{j=1}^{n} |a_{ij}|,$$

(3) sowie die induzierte 2-Norm, für die gilt

$$\|A\|_2 = \sqrt{\rho(A^T A)} = \sqrt{\rho(AA^T)} = \sigma_1(A),$$

wobei $\rho(A) = \max_{\lambda\in\sigma(A)} |\lambda|$ den *Spektralradius*[1] und $\sigma_1(A)$ den größten Singulärwert von A bezeichnet.

Neben der induzierten Matrixnorm gibt es aber auch weitere, die ebenfalls verträglich sind:

Beispiel C.4: Die Matrixnorm

$$\|A\|_F = \sqrt{\sum_{i,j=1}^{d} |a_{ij}|^2} = \operatorname{spur} AA^T$$

auf dem $\mathbb{R}^{d\times d}$, die sog. *Frobeniusnorm*, ist mit der 2-Norm verträglich.

[1] $\sigma(A)$ bezeichnet die Menge aller Eigenwerte, das *Spektrum*, der Matrix A.

Literaturverzeichnis

[1] Aulbach, B.: *Gewöhnliche Differenzialgleichungen*. Elsevier-Spektrum, Heidelberg, 2. Aufl., 2004.

[2] Barnes, J. and P. Hut: *A hierarchical $o(n \log n)$ force-calculation algorithm*. Nature, 4(324):446–449, 1986.

[3] Behrens, J. und A. Iske: *MATLAB – Eine freundliche Einführung*. http://www-m3.ma.tum.de/twiki/pub/Allgemeines/Skripten/matlab.pdf, 1999.

[4] Deuflhard, P. und F. Bornemann: *Numerische Mathematik II*. de Gruyter, Berlin, 2. Aufl., 2002.

[5] Greengard, L. and V. Rokhlin: *A fast algorithm for particle simulations*. J. Comput. Phys., 73(2):325–348, 1987.

[6] Griebel, M., S. Knapek, G. Zumbusch und A. Caglar: *Numerische Simulation in der Molekülydynamik. Numerik, Algorithmen, Parallelisierung, Anwendungen*. Springer, Berlin, 2004.

[7] Guckenheimer, J. and P. Holmes: *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer, Heidelberg, 1983.

[8] Hairer, E., C. Lubich, and G. Wanner: *Geometric numerical integration. Structure-preserving algorithms for ordinary differential equations*. Springer, Heidelberg, 2nd ed., 2006.

[9] Hairer, E., S. P. Nørsett, and G. Wanner: *Solving ordinary differential equations I. Nonstiff problems*. Springer, Berlin, 2nd ed., 1993.

[10] Higham, D. J. and N. J. Higham: *MATLAB guide*. SIAM, Philadelphia, 2nd ed., 2005.

[11] Irwin, M.: *Smooth dynamical systems*. Academic Press, New York, 1980.

[12] Kelley, A.: *The stable, center-stable, center, center-unstable, unstable manifolds*. J. Differential Equations, 3:546–570, 1967.

[13] Kolinsky, H.: *Eine Einführung in das Computeralgebrasystem MAPLE*. http://www.rz.uni-bayreuth.de/lehre/maple/, 2007.

[14] La Salle, J. und S. Lefschetz: *Die Stabilitätstheorie von Ljapunow. Die direkte Methode mit Anwendungen.*. Bibliographisches Institut, Mannheim, 1967.

[15] Lynch, S.: *Dynamical systems with applications using MAPLE*. Birkhäuser, Boston, 2001.

[16] Lynch, S.: *Dynamical systems with applications using MATLAB*. Birkhäuser, Boston, 2004.

[17] Meiss, J. D.: *Differential dynamical systems*, vol. 14 of. *Mathematical Modeling and Computation*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.

[18] Palis, J. and W. de Melo: *Geometric theory of dynamical systems.*. Springer, Heidelberg, 1982.

[19] Quarteroni, A., R. Sacco und F. Saleri: *Numerische Mathematik 1*. Springer, Berlin, 2002.

[20] Sontag, E. D.: *Mathematical Control Theory*. Springer, New York, 2nd ed., 1998.

[21] Sotomayor, J.: *Generic bifurcations of dynamical systems*. In. *Dynamical systems (Proc. Sympos., Univ. Bahia, Salvador, 1971)*, pp. 561–582. Academic Press, New York, 1973.

[22] Sparrow, C.: *The Lorenz equations: bifurcations, chaos, and strange attractors*. Springer, New York, 1982.

[23] Stoer, J. und R. Bulirsch: *Numerische Mathematik 2*. Springer, Heidelberg, 5. Aufl., 2005.

[24] Tucker, W.: *The Lorenz attractor exists*. C. R. Acad. Sci. Paris Sér. I Math., 328(12):1197–1202, 1999.

[25] Walter, W.: *Gewöhnliche Differentialgleichungen*. Springer, Heidelberg, 7. Aufl., 2000.

Notationsverzeichnis

$\|\|x\|\|_p$	p-Norm auf dem $\mathbb{R}^d$, 229
$\|\|x\|\|_\infty$	∞-Norm auf $C(I, \mathbb{R}^d)$, 30
$C(I, \mathbb{R}^d)$	stetige Funktionen von I nach $\mathbb{R}^d$, 30
e_i	i-ter Einheitsvektor, 11
E^s, E^u, E^c	stabiler, instabiler, Zentrums-Unterraum, 93
$\exp(A)$, e^A	Matrix-Exponentialfunktion, 13
$f(t, x)$	Vektorfeld, 1, 25
$f(x)$	autonomes Vektorfeld, 2
$H(q, p)$	Hamilton-Funktion, 166
H^*	Hausdorff-Abstand, 151
I_{t_0, x_0}	maximales Existenzintervall, 31
L	Lipschitz-Konstante, 31
$L(q, \dot{q})$	Lagrange-Funktion, 164
$\omega(x)$, $\alpha(x)$	ω-, α-Limesmenge, 99
$O(x)$	Orbit, 92
$\Phi(t; t_0)$	Übergangsmatrix, 21
$\Phi(t_i, \tilde{x}(t_i), h_i)$	Einschrittverfahren, 74
$\varphi^t(x_0)$	Lösungsfluss, 91
t_0	Anfangszeit, 27
(t_0, x_0)	Anfangsbedingung, 27
$u(t)$	Kontrollfunktion, 114
$V(x)$	Lyapunov-Funktion, 122
W^c	Zentrums-Mannigfaltigkeit, 142
W^s, W^u	globale stabile, instabile Mannigfaltigkeit, 97
W^s_ε, W^u_ε	lokale stabile, instabile Mannigfaltigkeit, 96
x^*	Gleichgewicht, 3, 51
x_0	Anfangswert, 27
$x(t)$	Lösungskurve, 1
$\tilde{x}$	Gitterfunktion, numerische Approximation, 74
$x(t; t_0, x_0)$	Lösungskurve mit Anfangsbedingung (t_0, x_0), 27

Index