**Seminar „Dynamische Systeme per Computer“**

# Beschreibung einer Beispiel-Simulation

Wir beschreiben hier ein einfaches Beispiel für die Lösung gewöhnlicher Differentialgleichungen in Matlab. Am besten lesen Sie sich zunächst in der Matlab-Hilfe den Artikel über gewöhnliche Differentialgleichungen durch (zu finden unter dem Stichwort „ODEs“).

## Mathematische Modellierung eines physikalischen Pendels

Als Beispiel wollen wir ein physikalisches Pendel simulieren. Unter einem physikalischen Pendel versteht man eine (näherungsweise) punktförmige Masse $m$, die an einer starren und (näherungsweise) masselosen Stange der Länge $l$ aufgehängt ist und nach links und rechts ausschlagen kann. Im einfachsten Fall (den wir hier betrachten) nimmt man an, dass das Pendel reibungsfrei schwingt.

Mit $\phi$ bezeichnen wir den Auslenkungswinkel des Pendels. Auslenkungen nach rechts versehen wir mit positivem Vorzeichen. Die Bewegungsgleichung des Pendels lautet dann

$$(*) \qquad l\ddot{\phi} = -g \sin \phi,$$

wobei $g$ die Fallbeschleunigung (manchmal auch Ortsfaktor genannt) bezeichnet. Es gilt etwa $g \approx 9.81 \frac{m}{s^2}$. Auf die Herleitung der Bewegungsgleichung $(*)$ wollen wir hier nicht eingehen (wer physikalisch interessiert ist, findet Sie sofort, indem er mit Hilfe einer Skizze die auf das Pendel wirkende Rückstellkraft berechnet und sie nach dem zweiten Newtonschen Axiom gleich dem Produkt aus Masse und Beschleunigung des Pendels setzt; die Masse kürzt sich dann übrigens heraus).

Es besitzt $(*)$ eine eindeutige (sogar globale) Lösung, falls wir den Anfangswinkel $\phi_0$ und die Anfangswinkelgeschwindigkeit $\omega_0 := \dot{\phi}(t_0)$zum Anfangszeitpunkt $t_0$ festlegen. Wir möchten diese Lösung nun von Matlab numerisch berechnen lassen.

## Lösung der ODE in Matlab

Dazu schreiben wir eine Matlab-Funktion *berechneWinkel*, der wir die Länge des Pendels, die Anfangsdaten, den Start- und Endzeitpunkt der Simulation sowie eine kleine Zeitdifferenz übergeben; die Zeitdifferenz gibt den Abstand der einzelnen Zeitpunkt an, zu denen wir die Lösung erhalten wollen. Die Funktion *berechneWinkel* soll zum einen einen Vektor mit sämtlichen Zeitpunkten zurückgeben, zu denen die Lösung berechnet wurde; außerdem soll sie zwei Vektoren zurückgeben, die zu jedem dieser Zeitpunkte den Auslenkungswinkel des Pendel bzw. die momentane Winkelgeschwindigkeit enthalten.
Die Funktionsweise von *berechneWinkel* ist im Quellcode genauer erläutert; hier gehen wir noch einmal auf die wichtigsten Punkte ein:

- Zur Lösung der Differentialgleichung benutzen wir einen Matlab-ODE-Solver, in diesem Fall die Funktion *ode45*.

- Sämtliche Matlab-ODE-Solver können nur Differentialgleichungen erster Ordnung lösen. Die Differentialgleichung $(*)$ ist aber (wie die meisten Differentialgleichungen aus der klassischen Mechanik) von zweiter Ordnung. Wir schreiben Sie deshalb in ein System aus Differentialgleichungen erster Ordnung um, indem wir als weitere Variable die Winkelgeschwindigkeit $\omega = \dot{\phi}$ definieren. Damit wird $(*)$ zu

$$(**) \qquad \begin{pmatrix} \dot{\phi} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \omega \\ -\frac{g}{l} \sin \phi \end{pmatrix}.$$

- Dem ODE-Solver von Matlab müssen wir ein function-handle übergeben, das die rechte Seite der Differentialgleichung beschreibt. Wir fassen die beiden Variablen $\phi$ und $\omega$ zum Vektor $z := (\phi, \omega)$ zusammen. Unser Differentialgleichungssystem $(**)$ schreibt sich dann in der Form $\dot{z} = f(z)$, wobei

$$f(z) = \begin{pmatrix} z_2 \\ -\frac{g}{l} \sin z_1 \end{pmatrix}$$

  ist. Diese Funktion $f$ definieren wir im Quellcode und übergeben das function handle dem ODE-Solver.

## Visualisierung der Ergebnisse

Als Ausgabe der Funktion *berechneWinkel* erhalten wir drei Vektoren mit Zahlen, die uns in tabellarischer Form vermutlich nur wenig sagen. Deshalb sollten wir die Lösung in geeigneter Form visualisieren. In den beiden Funktionen *pendelAnimation* und *phasenAnimation* wurden zwei Möglichkeiten einer solchen Visualisierung implementiert:

- Bei der Pendel-Animation wird zum jedem Zeitpunkt, an dem die Lösung berechnet wurde, die das Pendel gezeichnet und die Bilder werden im Anschluss in einem Video abgespeichert. So kann man die Bewegung des Pendel direkt sehen.

- Ebenfalls interessant ist es, die Bewegung des Pendel im Phasenraum darzustellen. Als Phasenraum bezeichnet man hierbei die Menge aller möglichen Zustände des Systems, das heißt die Menge aller Tupel $(\phi, \omega)$. In der Phasen-Animation wird die Trajektorie des Pendels im Phasenraum beschrieben, das heißt zu jedem Zeitpunkt $t$ werden Winkel und Winkelgeschwindigkeit in ein Koordinaten-System eingetragen.

Im m-Skript *bsp_ausfuehren.m* wird mit einigen Beispiel-Parametern die Verwendung der oben beschriebenen Funktionen demonstriert.

Je nach dem, welche Effekte man mit seiner Simulationen veranschaulichen möchte, könnte man die oben beschriebenen Beispiel-Implementierungen weiter ausbauen und / oder abändern. Hier ein paar Beispiele:

- Die Frequenz des Pendels hängt von seiner Länge ab. Wenn man zwei Pendel mit verschiedener Länge in dergleichen Animation darstellt, kann man die Frequenz der beiden Pendel direkt miteinander vergleichen.

- Für kleine Auslenkungswinkel $\phi$ kann man in der Differentialgleichung $(*)$ die Klein-Winkel-Näherung $\sin\phi \approx \phi$ verwenden und erhält dann die lineare Differentialgleichung $l\ddot{\phi} = -g\phi$ (das ist die Differentialgleichung eines sogenannten „harmonischen Oszillators“). Diese Differentialgleichung könnte man analytisch oder numerisch lösen und die Lösung mit der Lösung von $(*)$ vergleichen - zum Beispiel indem man die Trajektorie beider Lösungen im Phasenraum zeichnet oder zusätzlich zum ursprünglichen Pendel ein weiteres Pendel in die Animation einzeichnet, dessen Auslenkungswinkel aber der Differentialgleichung $l\ddot{\phi} = -g\phi$ gehorcht.

- Wenn man in die Differentialgleichung $(*)$ zusätzlich einen geschwindigkeitsabhängigen Reibungsterm mit aufnimmt, nimmt die Amplitude der Schwingung langsam ab. Würde man ein solches Pendel simulieren, dann könnte man bei der Pendel-Animation erkennen, dass das Pendel im Laufe der Zeit immer weniger ausschlägt; im Phasenraum würde man sehen, dass die Trajektorie spiralförmig gegen den Nullpunkt strebt.

Genauso sollte man auch für andere Anwendungen / Differentialgleichungen versuchen, Animationen und Graphiken zu erzeugen, die die Aspekte deutlich machen, die einen im einzelnen Fall interessieren.

Beispiele dafür, wie man Videos in Matlab erzeugen kann, finden Sie in den Quellcodes der beiden Funktionen *pendelAnimation* und *phasenAnimation*. Die folgenden Hinweise für die Erstellung von Videos in Matlab könnten nützlich sein:

- In den beiden Beispiel-Funktionen wird mit Hilfe der Matlab-Funktion *avifile* ein avi-Video erzeugt.

- Ab der Matlab-Version 2010b gibt es auch eine Funktion *VideoWriter*, mit der sich Videos aufzeichnen lassen. Eine Übersicht über weitere Möglichkeiten, um Videos und Animationen in Matlab zu erzeugen finden Sie zum Beispiel unter „http://stackoverflow.com/questions/11051307/approaches-to-create-a-video-in-matlab“.

- Bitte beachten Sie, dass der Kompressionsmodus, den Sie verwenden können, um die avi-Datei zu erzeugen, möglicherweise von Ihrem System abhängt; in den beiden Beispielfunktionen wird der Kompressionsmodus „Cinepak“ verwendet. Beachten Sie auch, dass unkomprimierte avi-Videos schnell sehr groß werden können.

- Wenn während der Erstellung eines Videos ein Fehler auftritt (z.B. weil Sie sich im Quellcode vertippt haben), bricht Matlab die Ausführung des Codes ab, was dazu führt, dass die bereits geöffnete avi-Datei nicht mehr geschlossen wird. Sie können die Datei dann nicht überschreiben und auch nicht löschen, bevor sie wieder geschlossen wurde. Um die Datei von der Matlab-Konsole aus zu schließen, können Sie den Befehl *clear mex* verwenden.

## Weitere Hinweise

Hier noch einige weitere Hinweise zur Lösung von ODEs in Matlab:

- Den Matlab-ODE-Solvern kann man verschiedene Fehlerschranken vorgeben. In der Beispiel-Funktion *berechneWinkel* wurde die relative Fehler-Toleranz auf $10^{-5}$ gesetzt. Eine geringere Fehlertoleranz erhöht natürlich die Rechenzeit.

- Es gibt mehrere verschiedene ODE-Solver in Matlab, die sich unterschiedlich gut für verschiedene Problemtypen eignen. Wenn der von Ihnen verwendete Solver sehr lange für die Berechnung einer Lösung benötigt, kann es sich lohnen, einen anderen Solver auszuprobieren. Alle Matlab-ODE-Solver werden in der Matlab-Hilfe beschrieben (eine Übersicht finden Sie unter dem Stichwort „ODEs“).

- Bei manchen dynamischen Systemen kennt man Erhaltungsgrößen, d.h. Größen, die sich im Laufe der Zeit nicht ändern. Beim ungedämpften Pendel ist zum Beispiel die Gesamtenergie eine Erhaltungsgröße. Indem man die Gesamtenergie gegen die Zeit plotten lässt, erhält man einen Eindruck davon, wie plausibel die Lösung physikalisch ist. Wenn die Gesamtenergie der berechneten Lösung sich im Laufe der Zeit stark ändern würde, würde dies auf eine numerisch schlechte Lösung hindeuten (man beachte, dass die vorgegebene Fehlertoleranz nicht immer sicher eingehalten wird). In unserem Beispiel wurde die Funktion *energiePlot* implementiert, um die Gesamtenergie gegen die Zeit aufzutragen; somit kann man erkennen, inwieweit die Gesamtenergie der numerischen Lösung tatsächlich zeitlich konstant ist.

- Bei der Erstellung eines Videos mit einer der beiden Beispiel-Funktionen *pendelAnimation* und *phasenAnimation* werden die einzelnen Frames während der Video-Erstellung am Bildschirm angezeigt. Die Geschwindigkeit, in der die Frames hierbei angezeigt werden, hängt aber lediglich von der Geschwindigkeit ihres Computers bei der Darstellung der Bilder ab (beachten Sie, dass die numerische Lösung der Differentialgleichung für das gegebene Beispiel sehr schnell funktioniert und das erste Bild erst angezeigt wird, nachdem die Differentialgleichung für das komplette Zeitintervall schon gelöst wurde). Die Geschwindigkeit, in der Sie die Bilder beim Erzeugen des Videos sehen, hängt auch nicht mit der späteren Abspielgeschwindigkeit des erzeugten Video zusammen (diese wird festgelegt, indem man beim *avifile*-Objekt die Eigenschaft *fps* setzt, die für „frames per second“ steht).