Technische Universität München
Institut für Informatik
Prof. Dr. Hans-Joachim Bungartz
Dipl.-Inf. Christoph Riesinger
Dipl.-Math. Jürgen Bräckle

WS 2013/2014

Numerisches Programmieren

4. Programmieraufgabe: Freier Fall, Planetensystem & ODEs

Dieses Aufgabenblatt steht wieder ganz unter dem Motto "Spielerisch zur numerischen Simulation". Wir werden uns hierbei ODEs im Zusammenhang mit punktförmigen Objekten anschauen, z. B. einer kleinen Kugel. Ein solches Objekt betrachten wir uns im freien Fall und unter der Beeinflussung von diversen anderen Kräften. Die daraus entstehenden ODEs sollen mit einfachen Zeitschrittverfahren gelöst werden.

Einführung: Freier Fall

Physikalische Grundlagen

Betrachten wir zuerst ein stark vereinfachtes Modell des freien Falls von einer kleinen Kugel in 2D. Der Zustand unserer vereinfachten Kugel ist für einen Zeitpunkt t eindeutig über die Position $\vec{p}(t)$ und die momentane Geschwindigkeit $\vec{v}(t)$ bestimmt.

Nun betrachten wir die Änderung der Position und Geschwindigkeit über einen kleinen Zeitschritt Δt. Hierbei gilt, dass die Positionsänderung dem Integral der Geschwindigkeit $\vec{v}$ während des nächsten Zeitschritts entspricht:

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \int_t^{t+\Delta t} \vec{v}(T)dT$$

Des Weiteren gilt auch, dass sich die Geschwindigkeitsänderung über das bestimmte Integral der Beschleunigung $a(t)$ berechnen läßt:

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \int_t^{t+\Delta t} \vec{a}(T)dT$$

Zur Vereinfachung betrachten wir nun den freien Fall mit konstanter Beschleunigung durch die Gravitation mit $g = 9.81\frac{m}{s^2}$ und damit $\vec{a} = (0, -g)^T$:

$$\Delta\vec{v}(t + \Delta t) = \int_t^{t+\Delta t} \vec{a}(T)dT = \int_t^{t+\Delta t} \begin{pmatrix} 0 \\ -g \end{pmatrix} dT$$

Nun können wir uns die Geschwindigkeit zu jedem beliebigen Zeitpunkt für gegebene Anfangswerte $\vec{v}(0)$ exakt ausrechnen indem wir die Integrale weiter auswerten:

$$\vec{v}(t) = \vec{v}(0) + \Delta\vec{v}(t) = \int_0^t \begin{pmatrix} 0 \\ -g \end{pmatrix} dT = \vec{v}(0) + \left[\begin{pmatrix} 0 \\ -g \cdot T \end{pmatrix} \right]_0^t = \vec{v}(0) + \begin{pmatrix} 0 \\ -g\Delta t \end{pmatrix}$$

Mit der Auswertung des Integrals für die Positionsänderung können wir letztendlich auch die Position in Abhängigkeit von der Zeit t berechnen:

$$\vec{p}(t) = \vec{p}(0) + \left[\begin{pmatrix} \vec{v}(0)T \\ \vec{v}(0)T - \frac{1}{2}gT^2 \end{pmatrix} \right]_0^t = \vec{p}(0) + \vec{v}(0)\Delta t - \begin{pmatrix} 0 \\ \frac{1}{2}g\Delta t^2 \end{pmatrix}$$

Damit liegt uns eine analytische Lösung vor, die uns für gegebene Anfangswerte zu einem beliebigen Zeitpunkt t die Position zurückgibt. Nachdem wir den freien Fall nun analytisch betrachtet haben, wollen wir im Weiteren betrachten, wie man eine solche Simulation durch numerische Verfahren approximieren kann. Leider hängt z. B. in Spielen die Simulation nicht nur von der Gravitation ab, sondern auch noch von vielen anderen Einflussgrößen (Federn, Stöße durch andere Objekte, Kollisionsimpulse, etc.) die uns eine analytische Lösung erschweren bzw. unmöglich machen.

ODE Beispielfunktionen

Die erste ODE im Rahmen dieses Aufgabenblattes ist die des freien Falls:

$$\vec{f}(p, t) = \begin{pmatrix} \vec{f}_x(0) \\ \vec{f}_y(0) - g \cdot t \end{pmatrix}$$

$\vec{f}$ entspricht hierbei der Geschwindigkeit $\vec{v}$. Des Weiteren betrachten wir auch noch drei weitere, etwas komplexere Funktionen, um die Unterschiede der Verfahren besser verstehen zu können.

Die zweite Funktion in unserer Testreihe ist mit

$$\vec{g}(p, t) = \begin{pmatrix} \vec{g}_x(0) + p_x(t)^2 \cdot 0.005 \\ \vec{g}_y(0) - gt - p_x(t)^2 t^2 \cdot 0.001 \end{pmatrix}$$

gegeben, wobei $p(0)$ die Anfangsbedingung beschreibt, $\vec{g}$ die Funktion und g die Gravitationskonstante. Sie ist 2. Ordnung in $(p_x(t))$ und (t)!

Die dritte ist 4. Grades in t und gegeben mit:

$$\vec{h}(p,t) = \begin{pmatrix} \vec{h}_x(0) + 4.0 \cdot t \\ \vec{h}_y(0) + 0.1 \cdot t^4 - 8.0 \cdot t^2 \end{pmatrix}$$

Die vierte Funktion ist ein steifes Beispiel, und daher sehr effizient mit impliziten Verfahren lösbar:

$$\vec{l}(p,t) = \begin{pmatrix} p_y(t) - 1 \\ -100 * (p_x(t) - 10) - 101 * p_y(t) + 1 \end{pmatrix}$$

Programmoberfläche

Als Hilfestellung und zur Veranschaulichung der Funktionen gibt es eine GUI, die in Abbildung 1 genauer beschrieben ist.

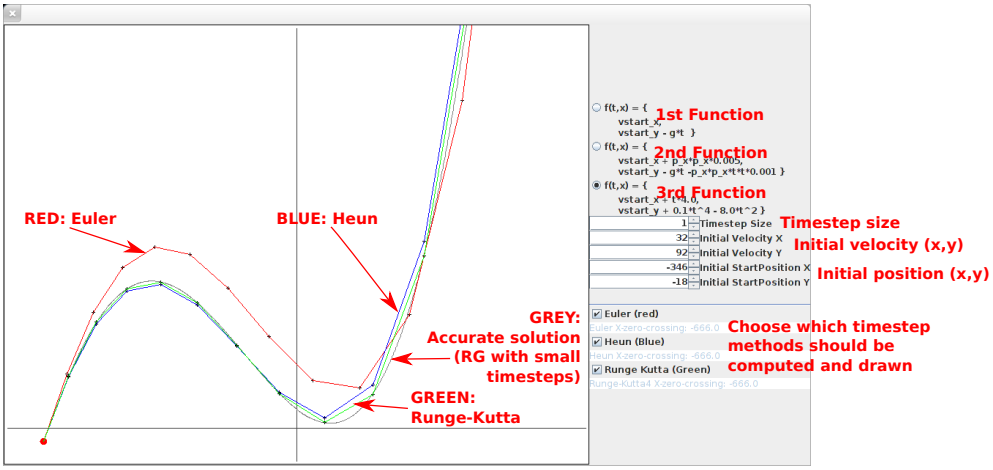

Abbildung 1: GUI for timestep methods

Auf der rechten Seite kann eine der im obigen Abschnitt beschriebenen Funktionen gewählt werden. Die Zeitschrittweite sowie Anfangswerte für Position und Geschwindigkeit lassen sich direkt darunter einstellen.

Im Zeichenfenster auf der linken Seite markiert der rote Punkt den im rechten Bereich angegebenen Startpunkt der ODE. Die Zeitschrittverfahren sind farbkodiert (siehe Legende rechts), die Kurvenwerte an den diskreten Zeitpunkten jeweils mit einem schwarzen Kreuz markiert.

Planetensystem: Physikalische Grundlagen

Dieser Abschnitt schildert eine Weiterführung des freien Falls. Verschiedene Objekte, in unserem Fall Planeten, sollen sich unter gegenseitigem Einfluss durch die Gravitation im Raum bewegen. Ausgehend von der Wirkung der Gravitationskraft F zwischen zwei Objekten

$$F = Gm_1m_2\frac{\vec{p_2} - \vec{p_1}}{|\vec{p_2} - \vec{p_1}|^3}$$

mit G der Gravitationskonstante, den Massen m_1 und m_2 und den Positionen p_1 und p_2 der beiden Objekte, lässt sich ein System gewöhnlicher Differentialgleichungen aufstellen. Die Summe aller Kräfte auf ein Objekt wird über den Zusammenhang $F = ma$ in eine Beschleunigung umgerechnet, welche der 2. Ableitung des Ortes p enspricht. Durch Einbeziehen der Geschwindigkeit v kann man nun diese Differentialgleichung 2. Ordnung in eine Differentialgleichung 1. Ordnung umwandeln:

$$\begin{pmatrix}\dot{p}_i(t)\\ \dot{v}_i(t)\end{pmatrix} = \begin{pmatrix}v_i(t)\\ a_i(t)\end{pmatrix} = \begin{pmatrix}v_i(t)\\ -G\sum\limits_{j\neq i} m_j \frac{p_i(t)-p_j(t)}{|p_i(t)-p_j(t)|^3}\end{pmatrix} \text{ für ein Objekt } i.$$

Ab drei Körpern ist dieses Problem nicht mehr analytisch lösbar, und wir sind auf numerische Mittel wie den uns bekannten Einschrittverfahren angewiesen.

Zeitschrittverfahren

Nun werden die in dieser Programmieraufgabe verwendeten Zeitschrittverfahren vorgestellt, welche sich alle auf das $\mathbb{R}^n$ erweitern lassen. Folgende Informationen finden sich auch in den Vorlesungsunterlagen.

Expliziter Euler

Mit Genauigkeitsordnung 1 ist der explizite Euler das einfachste Zeitintegrationsverfahren. Aus der Ordnung des Verfahrens können wir schließen, dass es nur für lineare Funktionen exakt ist. Ausgehend von einem Zeitpunkt t wird angenommen, dass die zu integrierenden Funktion als Gerade beschrieben werden kann. Die Position zum nächsten Zeitpunkt $t + \Delta t$ ergibt sich damit wie folgt:

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \Delta t \vec{f}(t, \vec{p}(t))$$

Die Funktion $\vec{f}$ gibt in dieser Aufgabe die Geschwindigkeit zum gegebenen Zeitpunkt t und einer Position $\vec{p}(t)$ zurück.

Heun

Das Verfahren von Heun ist ein Verfahren zweiter Ordnung und mit eine Ordnungsstufe genauer als das Eulerverfahren. Die erste Gleichung, die hierbei benötigt wird, approximiert den nächsten Punkt analog dem Eulerverfahren:

$$\overline{\vec{p}(t + \Delta t)} = \vec{p}(t) + \Delta t \vec{f}(t, \vec{p}(t))$$

Dieser Punkt wird aber nun nicht als neuer Punkt, sondern in einer weiteren Formel verwendet, um den neuen Punkt mit einem Genauigkeitsgrad 2 zu berechnen:

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \frac{\Delta t}{2} \left(\vec{f}(t, \vec{p}(t)) + \vec{f}(t + \Delta t, \overline{\vec{p}(t + \Delta t)}) \right)$$

Runge-Kutta

Die ersten nach ihren Erfindern Carl Runge und Martin Willhelm Kutta benannten Methoden wurden zu Beginn des 20. Jahrhunderts beschrieben. Die grundlegende Idee hinter den Verfahren ist es, durch das Berechnen von Zwischenwerten innerhalb eines Zeitschrittes ein Verfahren höherer Ordnung zu erhalten. Sowohl das Euler- als auch das Heun-Verfahren entsprechen im übrigen Runge-Kutta-Verfahren der jeweiligen Genauigkeitsordnung.

Das hier beschriebene Verfahren stellt das sogenannte "klassische Runge-Kutta-Verfahren" dar und ist von Genauigkeitsordnung 4.

Zuerst werden die Koeffizienten $\vec{k_i}$ bestimmt:

$$\vec{k_1} = \Delta t \vec{f}(t, \vec{p}(t))$$
$$\vec{k_2} = \Delta t \vec{f}(t + \frac{1}{2}\Delta t, \vec{p}(t) + \frac{1}{2}\vec{k_1})$$
$$\vec{k_3} = \Delta t \vec{f}(t + \frac{1}{2}\Delta t, \vec{p}(t) + \frac{1}{2}\vec{k_2})$$
$$\vec{k_4} = \Delta t \vec{f}(t + \Delta t, \vec{p}(t) + \vec{k_3})$$

Die neue Position können wir dann mit folgender Formel berechnen:

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \frac{1}{6}(\vec{k_1} + 2\vec{k_2} + 2\vec{k_3} + \vec{k_4})$$

Impliziter Euler

Die implizite Variante des Euler-Verfahrens

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \Delta t \vec{f}(t + \Delta t, \vec{p}(t + \Delta t))$$

unterscheidet sich nur durch das Auftauchen des nächsten Zeitpunktes $t + \Delta t$ auf der rechten Seite vom expliziten Euler, wodurch sich der nächste Wert $\vec{p}(t + \Delta t)$ nicht mehr explizit berechnen lässt. Methoden zur Lösung von Gleichungssystem, wie z.B. dem Newton-Verfahren, müssen angewendet werden. Diesen zusätzlichen Aufwand können wir durch die Effektivität bei steifen ODEs ausgleichen.

Newton-Verfahren

Das Newton-Verfahren ist eine Methode zur Lösung von nicht-linearen Systemen

$$f(\vec{x}) = \vec{0} \text{ mit } f : \mathbb{R}^n \to \mathbb{R}^n.$$

Ausgehend von einem Startvektor x_0 wird mit der Formel

$$x_{k+1} = x_k - J(x_k)^{-1} f(x_k)$$

und der Jacobi-Matrix $J(x_k)$ von f eine Folge von immer besseren Näherungen an eine Nullstelle berechnet. Die Newton-Folge wird abgebrochen, sobald ein Stelle x_k mit $||f(x_k)|| < \epsilon$ gefunden oder eine maximale Iterationszahl erreicht wurde. Will man ein System von Gleichungen der Form $g(x) = h(x)$ lösen, wendet man einfach das Newton-Verfahren auf die Funktion $f(x) = g(x) - h(x)$ an.

Beschreibung des Programmgerüsts

Das auf der Vorlesungsseite zur Verfügung gestellte Programmgerüst enthält:

- `freierfall`:
  Dieses Package enthält alle Klassen zur Berechnung des freien Falls und der ensprechenden Benutzeroberfläche. Zur Bearbeitung der Programmieraufgabe ist ein Verständnis dieser Klassen nicht erforderlich.
- `planeten`:
  Dieses Package enthält alle Klassen zur Simulation von Planetensystemen. Zur Bearbeitung der Programmieraufgabe ist ein Verständnis dieser Klassen nicht erforderlich.
- `ODE`:
  Dieses Interface beschreibt eine ODE der Form $\dot{\vec{y}}(t) = f(t, \vec{y}(t))$ und ermöglicht das Auswerten der rechten Seite f.
- `ExpliziterEuler`, `Heun` und `RungeKutta4`:
  Diese Klassen ermöglichen die Berechnung eines Schrittes des entsprechenden Einschrittverfahrens.
- `ImpliziterEuler`:
  Neben der Berechnung eines Zeitschrittes des impliziten Eulers enthält diese Klasse die Newton-Methode, eine Approximation der Jacobi-Matrix, und die Gauss-Elimination.
- `Funktion`:
  Diese Beschreibung einer Funktion $f : \mathbb{R}^n \to \mathbb{R}^m$ lässt sich an einer Stelle x auswerten, und wird für den Einsatz der Newton-Methode benötigt.
- `Test`:
  Die Test-Klasse startet die beiden Benutzeroberflächen für den freien Fall und das Planetensystem, und enthält einige aus der Übung bekannte Testbeispiele.

Aufgaben

Im Folgenden werden die zu implementierenden Methoden aufgelistet. Details finden Sie jeweils in den Kommentaren zu den einzelnen Methoden. Überlegen Sie sich zu allen Methoden Testbeispiele und testen Sie unbedingt jede einzelne Methode direkt nach der Erstellung. Sie können zusätzlich die zur Verfügung gestellte Testklasse `Test` verwenden. Es ersetzt aber nicht das Verwenden eigener Testbeispiele.

- Programmieren Sie in den Klassen `ExpliziterEuler`, `Heun`, `RungeKutta4` und `ImpliziterEuler` den gegebenen Methodenrumpf `nextStep`.
- Programmieren Sie in der Klasse `ImpliziterEuler` den gegebenen Methodenrumpf `newtonMethod`.

Formalien und Hinweise

- Das Programmgerüst erhalten Sie auf den Webseiten zur Vorlesung.
- Ergänzen Sie das Programmgerüst bitte **nur an den dafür vorgegebenen Stellen!** Falls Sie die Struktur der Programme an anderer Stelle verändern, können wir sie evtl. nicht mehr testen.
- Beseitigen Sie vor Abgabe Ihres Programms alle Ausgaben an die Konsole und reichen Sie bis zum **03. Februar 2014, 12:00 Uhr** über Moodle ein.
- Reichen Sie nur die Dateien `ExpliziterEuler.java`, `Heun.java`, `RungeKutta4.java` und `ImpliziterEuler.java` ein. Deren Zuordnung zum Package `ode` muss nicht entfernt werden.
- Bitte laden Sie Ihre java-Dateien als flaches tgz-Archiv hoch. Der Dateiname ist beliebig wählbar, bei der Erweiterung muss es sich jedoch um .tgz oder .tar.gz handeln.

  Ein solches Archiv können Sie beispielsweise mit dem Linux-Tool tar erstellen, indem Sie die laut Aufgabenstellung zu bearbeitenden java-Dateien in ein sonst leeres Verzeichnis legen und dort anschließend den Befehl

  ```
  > tar cvvzf numpro_aufg4.tgz *.java
  ```

  ausführen.
- Wir empfehlen Ihnen, die Programme unter Linux (Rechnerhalle) zu testen.

Bitte beachten Sie: *Alle Abgaben, die nicht den formalen Kriterien genügen, werden grundsätzlich mit 0 Punkten bewertet!*