Einführung in die Numerische Mathematik

Sommersemester 2015
Prof. Dr. Jochen Garcke
Patrick Diehl

universitätbonn

Übungsblatt 9.

Abgabe am **16.06.2015 vor der Vorlesung**.

Aufgabe 1. (Eulerverfahren für Anfangswertprobleme)

Für das Anfangswertproblem

$$\dot{x}(t) = t - t^3, \qquad x(0) = 0.$$

sollen mit Hilfe des Euler–Verfahrens Näherungswerte $y^{(k)}$ berechnet werden. Man gebe $x(t)$ und $y^{(k)}$ explizit an und zeige, dass an jeder Stelle $t = k \cdot h = const.$ der Fehler für $h \to 0$ gegen Null konvergiert.

Aufgabe 2. (Explizites und implizites Euler-Verfahren)

Das Anfangswertproblem

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -1000 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}, \qquad \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

soll mit dem expliziten und dem impliziten Euler–Verfahren mit fester Schrittweite h gelöst werden. Bestimmen Sie jeweils explizit die Iterierten (y_1^k, y_2^k) und vergleichen Sie diese mit der exakten Lösung an der Stelle $t = k \cdot h$. Was stellen Sie fest?

Aufgabe 3. (Heun-Verfahren)

Gegeben sei das Anfangswertproblem $\dot{x}(t) = f(t, x(t))$, $x(t_0) = x_0$. Beweisen Sie mithilfe der Taylor-Entwicklung, dass das Verfahren von Heun,

$$y^{(k+1)} := y^{(k)} + \frac{\tau_k}{2}\Big(f(t_k, y^{(k)}) + f\big(t_{k+1}, y^{(k)} + \tau_k f(t_k, y^{(k)})\big)\Big),$$

die Konsistenzordnung zwei hat, falls f genügend glatt ist.

Programmieraufgabe 1. (Planetenbewegung)

Ziel dieser Aufgabe ist es, in einem vereinfachten Modell unseres Sonnensystems die Bewegung von Sonne, Erde, dem Planeten Jupiter und einem Kometen ähnlich dem Halleyschen Kometen zu simulieren. Die Bahnen verlegen wir in den zweidimensionalen Raum und die Sonne in den Ursprung. Die Körper werden als jeweils ein Partikel dargestellt, zwischen den Partikeln wirkt das Gravitationspotential.

Algorithmus

Wir verwenden die Geschwindigkeitsform des Verlet-Algorithmus verwenden. Alle vektoriellen Größen x, F, v sind in R^N (hier $N = 2$). Wir erhalten

$$x_i^{n+1} = x_i^n + \delta t v_i^n + \frac{F_i^n \cdot \delta t^2}{2m_i}$$

Für die Geschwindigkeit ergibt sich

$$v_i^{n+1} = v_i^n + \frac{(F_i^n + F_i^{n+1})\delta t}{2m_i}$$

Thermodynamische Größen, wie zum Beispiel die kinetische oder die potentielle Energie, können alle zum gleichen Zeitpunkt t berechnet werden, da die Orte wie auch die Geschwindigkeiten der Partikel zum gleichen Zeitpunkt vorliegen. So läßt sich die kinetische Energie

$$E_{kin}^n = \frac{1}{2}\sum_{i=1}^{N} m_i |v_i^n|^2$$

zum Zeitpunkt t_n direkt nach den Geschwindigkeiten v_i berechnen. Ebenso erhält man die potentielle Energie

$$V^n = V(x_i^n, ..., x_N^n)$$

zum Zeitpunkt t_n aus den Orten $x_1^n, \ldots, x_N^m$ der Partikel zum Zeitpunkt t_n. Alles zusammen ergibt sich der Algorithmus 1.

Algorithm 1: Geschwindigkeits-Verlet-Algorithmus

Start mit Anfangsdaten x, v, t;
Hilfsvektor F^{old};
berechne Kräfte F;
while $t < t\_end$ **do**
  t = t + delta_t;
  for *Schleife über all i* **do** ▷ update x
    ;
    $x_i = x_i$ + delta_t $*$ $(v_i + .5/m_i * F_i *$ delta_t);
    $F_i^{old} = F_i$;
  berechne Kräfte F;
  for *Schleife über all i* **do** ▷ update v
    ;
    $v_i = v_i$ + delta_t $*$ .5 / $m_i * (F_i + F_i^{old})$;
  berechne abgeleitete Größen wie z.B. kinetische oder potentielle Energie;
  gebe Werte t, x, v sowie abgeleitete Größen aus

Implementierung

Dafür könnt (und sollt!) ihr folgendes Gerüst für die Implementierung benutzen und die fehlenden Methoden implementieren. Wir nutzen für die Partikel eine Klasse, in der die Daten für jedes Partikel gespeichert sind, hier können wir die Variablen für Masse, Ort, Geschwindigkeit und Kraft der Zeitintegration direkt übernehmen.

```
1  class Particle:
2      def __init__(self, m, x, v, F=((0.,0.)), F_old=((0.,0.)), number=None ):
3          self.mass = float(m) # Masse
4          self.x = array(x, float) # Ort
5          self.v = array(v, float) # Geschwindigkeit
6          self.F = array(F, float) # Kraft zum aktuellen Zeitschritt
7          self.F_old = array(F_old, float) # Kraft zum vorherigen Zeitschritt
8          if not number is None:
9              self.number = number
10
11     def update_x(self, delt):
12         '''bewegt das Partikel'''
13
14     def update_v(self, delt):
```

```
15          '''update der Geschwindigkeiten,
16          Rueckgabe der kinetischen Energie'''
```

Die Klasse Problem enthält problemspezifische Variablen wie z.B. den Namen, Zeitschrittweite, usw. Außerdem wird hier die Routine für das verwendete Potential, usw.

```
1  class Problem:
2      def __init__(self, filename):
3
4      def read_particles(self, filename):
5
6      def write_particles(self, file, time):
7          output = '%d\ntime␣=␣%f\n' % (len(self.particles), time)
8          file.write(output)
9          for particle in self.particles:
10             output = '%s␣' % particle.number
11             for i in particle.x:
12                 output += '%g␣' % i
13             output += '\n'
14             file.write(output)
15
16     def comp_forces(self):
17         '''Berechnung der Kraefte, ruft fuer jedes Partikelpaar force auf
18         Rueckgabe der potentiellen Energie'''
19
20     def force(p1, p2):
21         '''Kraftberechnung zwischen zwei Partikeln beim stellaren Beispiel
22         Rueckgabe der potentiellen Energie'''
```

Die Kraftberechnung soll zunächst ganz elementar ausgeführt werden, indem wir für die Kraft auf das Partikel i seine Wechselwirkung mit jedem anderen Partikel j bestimmen. Die Kraft auf ein Partikel i erhalten wir wegen

$$V(x_1, \ldots, x_N) = \sum_{i=1}^{N} \sum_{j=1, j<i}^{N} U(p_i, p_j)$$

mit

$$F_i = -\nabla_{x_i} V(x_1, \ldots, x_N) = \sum_{j=1, j\neq i}^{N} -\nabla_{x_i} U(p_i, p_j) = \sum_{j=1, j\neq i}^{N} F_{ij}$$

gerade als Summe über die Einzelkräfte $F_{ij} = -\nabla_{x_i} U(p_i, p_j)$. Die Berechnung der Kräfte kann dabei (vorerst) als zweifache Schleife über alle Partikel geschrieben werden.

Der Halleysche Komet

Unser Zeitintegrationsverfahren kommt ursprünglich aus der Astromomie. Es wurde bereits 1790 von Delambre und später von Störmer und anderen zur Berechnung von Planeten- und Kometen- und Teilchenbahnen benutzt. In Anlehnung daran betrachten wir ein einfaches astronomisches Problem. In einem vereinfachten Modell unseres Sonnensystems simulieren wir die Bewegung von Sonne, Erde, dem Planeten Jupiter und einem Kometen ähnlich dem Halleyschen Kometen. Die Bahnen verlegen wir in den zweidimensionalen Raum und die Sonne in den Ursprung. Die Körper werden als jeweils ein Partikel dargestellt. Zwischen den Partikeln wirkt das Gravitationspotential. Ein Satz von Anfangswerten ist in Tabelle 1 aufgeführt. Die resultierenden Bahnen der Himmelskörper sollen ausgerechnet und dargestellt werden.
Im folgenden Beispiel verwenden wir das (skalierte) Gravitationspotential

$$U(p_i, p_j) = -m_i m_j / r_{ij}$$

als paarweise Wechselwirkung. Aus dem Potential erhalten wir die Kraft

$$F_{ij} = -\nabla_{x_i} U(p_i, p_j) = \frac{m_i m_j}{r_{ij}^3} \vec{r}_{ij}$$

die von Partikel j auf Partikel i ausgeübt wird. Dabei bezeichnet $\vec{r}_{ij} := x_j - x_i$ den Richtungsvektor zwischen den Partikeln i und j an den Orten x_i beziehungsweise x_j und r_{ij} dessen Norm.

m_{Sonne}	$= 1,$	x_{Sonne}	$= (0,0),$	v_{Sonne}	$= (0,0),$
m_{Erde}	$= 3.0 \cdot 10^{-6},$	x_{Erde}	$= (0,1),$	v_{Erde}	$= (-1,0),$
$m_{Jupiter}$	$= 9.55 \cdot 10^{-4},$	$x_{Jupiter}$	$= (0,5.36),$	$v_{Jupiter}$	$= (-0.425,0),$
m_{Halley}	$= 1 \cdot 10^{-14},$	x_{Halley}	$= (34.75,0),$	v_{Halley}	$= (0,0.0296),$
δ_t	$= 0.015,$	t_{end}	$= 468.5$		

Tabelle 1: Parameterwerte für eine vereinfachte Simulation der Bahn des Halleyschen Kometen

Gibt man die Orte der Partikel zu jedem Zeitschritt so aus, wie es im Beispielprogramm angegeben wird, so kann man eine einfache Visualisierung durchführen.

```
1  import numpy
2  import pylab
3
4  file=open('partikel.out')
5
6  x = []
7  y = []
8
9  for i in file:
10     num_particles = int(i)
11     a=file.next()
12     x_step = []
13     y_step = []
14     for j in range(num_particles):
15         line=file.next()
16         fields = line.split()
17         x_step.append(fields[1])
18         y_step.append(fields[2])
19     x.append(x_step)
20     y.append(y_step)
21
22 x = numpy.array(x)
23 y = numpy.array(y)
24
25 pylab.plot(x,y)
26 pylab.show()
```

Zur Kontrolle der Energieerhaltung werden diese zu jedem Zeitschritt rausgeschrieben und können dann einfach visualisiert werden. Die folgende Routine nimmt an, dass die Energien eines Zeitschritts in einer Zeile geschrieben sind.

```
1  import numpy
2  import pylab
3
4  file=open('energies.dat')
5
6  energies=[]
7
8  for line in file:
9      energies.append(line.split())
10
11 energies = numpy.array(energies)
12
13 pylab.plot(energies)
14 pylab.show()
```

(16 Punkte)

Abgabe am 15.06.2015 oder 16.06.2015 im CIP-Pool. Weitere Hinweise finden Sie auf der Webseite.