Leapfrog és Runge Kutta módszer

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020 október 12.

A leapfrog[1] módszer

Másodrendű dinamikai egyenleteknél működik (molekuladinamikában hasonló az ún. Verlet módszer)

- a sebességeket és a koordinátákat külön-külön lépésben frissítjük
- másodrendű módszer, a hiba $O(h^4)$

$$\begin{aligned} a_n &= F(x_n)/m \\ v_{n+1/2} &= v_{n-1/2} + \Delta t \cdot a_n \\ x_{n+1} &= x_n + \Delta t \cdot v_{n+1/2} \end{aligned}$$

[1] bakugrás

A leapfrog[1] módszer

Másodrendű dinamikai egyenleteknél működik (molekuladinamikában hasonló az ún. Verlet módszer)

- a sebességeket és a koordinátákat külön-külön lépésben frissítjük
- másodrendű módszer, a hiba $O(h^4)$

$$\begin{aligned} a_n &= F(x_n)/m \\ v_{n+1/2} &= v_{n-1/2} + \Delta t \cdot a_n \\ x_{n+1} &= x_n + \Delta t \cdot v_{n+1/2} \end{aligned}$$

Ez még átírható így is:

$$\begin{aligned} a_n &= F(x_n)/m \\ x_{n+1} &= x_n + v_n \cdot \Delta t + \frac{1}{2} a_n (\Delta t)^2 \\ v_{n+1} &= v_n + \frac{1}{2}(a_n + a_{n+1}) \cdot \Delta t \end{aligned}$$

A hiba lecsökkent, mégis ugyanannyi számolást kell csak végezni!

[1] bakugrás

Az integrálási idő invertálhatósága

Visszafelé időfejlesztve a rendszert visszajutunk-e a kezdeti állapotba?

- ha nem disszipatív (pl nincs súrlódás), akkor elvileg igen
- megmarad az energia

Visszafelé léptetve egy diszkrét integrátort, visszajutunk-e az eredeti kiindulási pontba?

- egyszerű integrátorral általában nem
- a numerikus hibák összeadódnak
- kaotikus egyenleteknél pedig fel is erősödnek

Az integrálási idő invertálhatósága

Visszafelé időfejlesztve a rendszert visszajutunk-e a kezdeti állapotba?

- ha nem disszipatív (pl nincs súrlódás), akkor elvileg igen
- megmarad az energia

Visszafelé léptetve egy diszkrét integrátort, visszajutunk-e az eredeti kiindulási pontba?

- egyszerű integrátorral általában nem
- a numerikus hibák összeadódnak
- kaotikus egyenleteknél pedig fel is erősödnek

A leapfrog módszerről megmutatható:

- az időintegrálás szempontjából invertálható
- az energiamegmaradás is teljesül ("symplectic integrator")

Az Euler-módszer javítása: középponti módszer

Az Euler-módszer aszimmetrikus:

$$\begin{aligned} y_{n+1} &= y_n + h \cdot f(x_n, y_n) \\ x_{n+1} &= x_n + h \end{aligned}$$

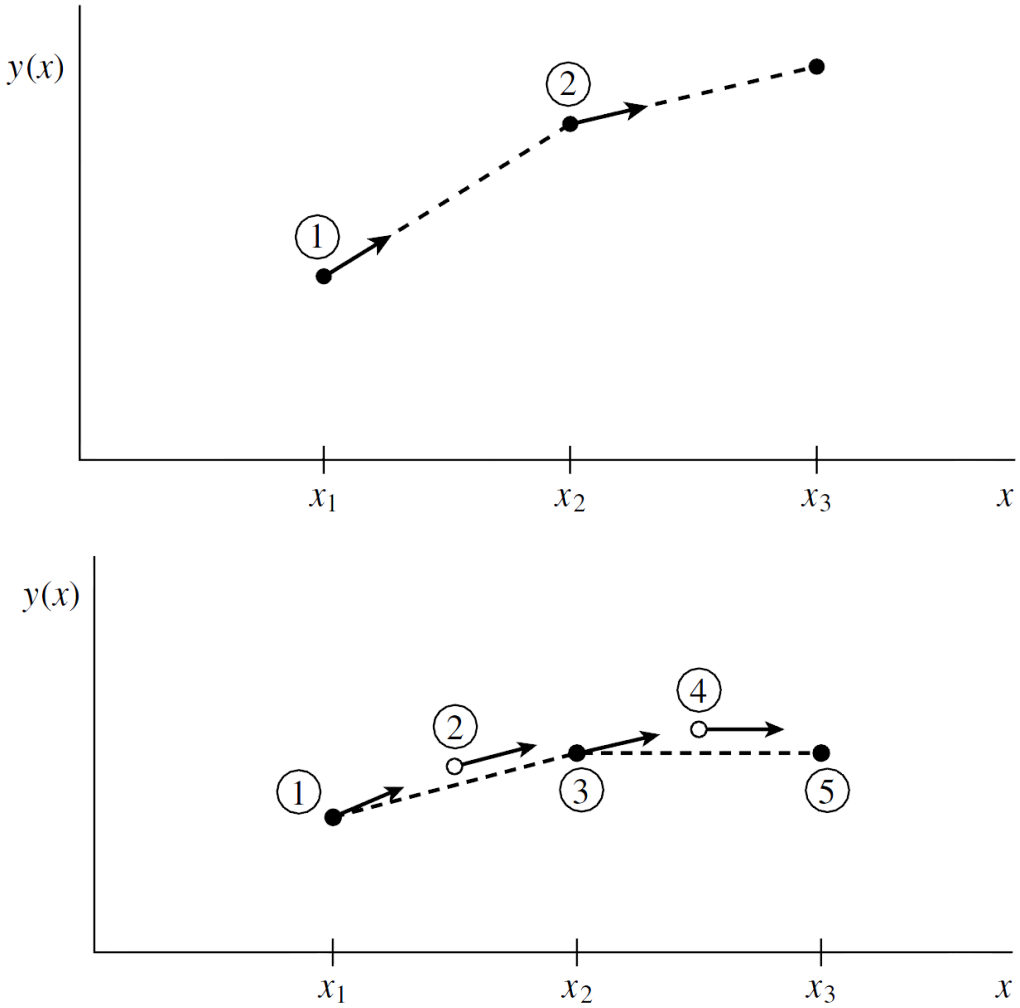
Az aszimmetria ott jelenik meg, hogy megoldást egy h-val léptetjük, de a szükséges deriváltat mindig az x_n helyen, vagyis az intervallum elején számítjuk ki.

Javítsunk a módszeren:

- kiszámítjuk a deriváltat az intervallum elején
- teszünk egy fél lépést és az előző lépést használva kiszámoljuk a deriváltat egy középső pontban
- ezt használjuk a teljes lépésben

$$\begin{aligned} k_1 &= h \cdot f(x_n, y_n) \\ k_2 &= h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\ y_{n+1} &= y_n + k_2 + O(h^3) \end{aligned}$$

Egyszerű Euler-lépés és a középponti módszer



A hiba további csökkentése: negyedrendű Runge–Kutta-módszer

A lépést több részlépésből előállítva bízhatunk abban, hogy a hiba tovább csökken.

Negyedrendű Runge–Kutta (RK4):

$$
\begin{aligned}
k_1 &= h \cdot f(x_n, y_n) \\
k_2 &= h \cdot f(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1) \\
k_3 &= h \cdot f(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2) \\
k_4 &= h \cdot f(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4 + O(h^5)
\end{aligned}
$$

A negyedrendű Runge–Kutta-módszer

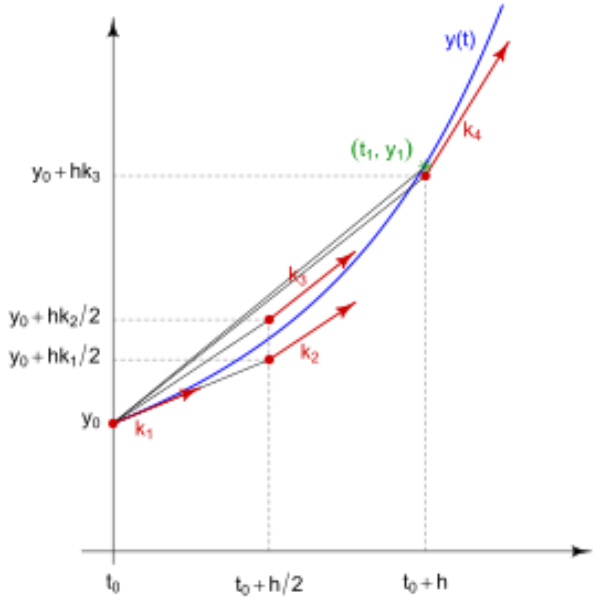


Figure: A függvény deriváltját négy pontban számoljuk ki: a kezdetiben, kétszer a lépéstávolság felénél és egyszer még a lépés végén. Wikipedia.

A Runge–Kutta-együtthatók meghatározása

Az együtthatók meghatározása nehéz és hosszadalmas

- az RK4-re a levezetést lásd pl Wikipedia-n
- a RK módszer magasabb rendekre is általánosítható
- általában a különböző (rendű) RK-ra az együtthatókat egy ún. Butcher-táblában adják meg
- RK4 táblázata:

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Módszerek összehasonlítása

	f kiértékeléseinek száma	hiba
Euler-módszer	1	$O(h^2)$
középponti módszer	2	$O(h^3)$
4-ed rendű Runge–Kutta	4	$O(h^5)$

A Runge–Kutta-módszer magasabb rendekre is általánosítható

- meddig érdemes elmenni?
- a több köztes pont mindig nagyobb pontosságot jelent? $\Rightarrow$ nem feltétlenül
- viszont több függvénykiértékeléssel jár

Kitekintés: numerikus integrátorok

- az RK módszer egy általános, sokféle problémára alkalmazható módszer
- speciális feladatok esetén illetve ha a megoldásról van valami előtudás, akkor található ennél optimálisabb módszer is

Hogyan struktúráljuk az integrátort programot?

Program írásakor

- A koordinátákat nem érdemes külön-külön változóban tárolni, tegyük be mindet egy vektorba
- ugyanez a helyzet az egyenlet paramétereivel is
- az f_i deriváltakat kell megírni, ezt tegyük egy külön függvénybe
- érdemes az integrátor függvényt úgy megírni, hogy paraméterként (többek között) azt a függvényt várja, amely ki tudja számolni az f_i deriváltakat $\Rightarrow$ **függvénypointerek** használata
- ezáltal az integrátor függvényt bármilyen mozgásegyenlet megoldására fel lehet használni, csak a deriváltakat számoló függvényt kell cserélni az egyes problémákra

Hogyan struktúráljuk az integrátort programot?

Függvény a deriváltak kiszámolására: ha bonyolult, érdemes `typedef`-t használni:

```
1 typedef void ODE(
2     double *,         // parameterek
3     double,          // fuggetlen valtozo
4     double *,         // fuggo valtozok
5     double *,         // derivaltak (kimeno)
6     int);             // egyenletek szama
```

Ilyen `ODE` függvényre volt példa a korábban már látott `HarmonicOscillator`

Hogyan struktúráljuk az integrátort programot?

Függvény a deriváltak kiszámolására: ha bonyolult, érdemes `typedef`-t használni:

```
1  typedef void ODE(
2      double *,          // parameterek
3      double,           // fuggetlen valtozo
4      double *,          // fuggo valtozok
5      double *,          // derivaltak (kimeno)
6      int);             // egyenletek szama
```

Ilyen `ODE` függvényre volt példa a korábban már látott `HarmonicOscillator`

Az előbbiek segítségével pl az integrátor függvény prototípusa

```
1  void eulerLepes(
2      double t_init,        // fuggetlen valtozo kezdoertek
3      double dt,        // lepeshossz
4
5      double *p,        // parameterek
6      double *y,        // valtozok
7      double *dy,       // derivaltak
8      int N,            // egyenletek szama
9      ODE *);           // egyenletek
```