Harmonikus oszcillátor

Számítógépes szimulációk szamszimf17la

Csabai István, Stéger József

ELTE
Komplex Rendszerek Fizikája Tanszék
Email: csabai@complex.elte.hu, steger@complex.elte.hu

Mottó

Harmonikus oszcillátor,
Hidrogén atom,
Van-e más is a világon,
Én nem tudhatom.

- *Megfelelően egyszerű modell:* A legtöbb valódi jelenség túl bonyolult ahhoz, hogy egzaktul leírjuk. Kellően egyszerű modellt kell választani, ami matematikai formába önthető, és a lényeget leírja. Tisztában kell lenni a modell korlátaival, érvényességi határaival.
- *Algoritmusok:* A matematikai modell megoldásához hatékony algoritmusokat kell találni. A számítógépek pontosságban, memóriában és futási időben jelenthetnek korlátokat. Számos ötletes algoritmus (pl. FFT) olyan problémák kezelését is lehetővé tette, amelyek elsőre kezelhetetlennek tűntek.
- *Implementáció:* A modell számítógépes implementálása sok absztrakciós szinten lehetséges (pl. C, C++, C#, Fortran, Java, python, Matlab, Mathematica), melyek különböznek a fejlesztésre fordítandó időben és az alkalmazás rugalmasságában.
- *Vizualizáció és interpretáció:* Az eredményeket meg kell jeleníteni akár a program által, akár utólag más grafikus eszközökkel. Nagy adatmennyiségnél maga a szemléletes vizualizáció is kihívás. A szimuláció eredményeit össze kell vetni az elvárt viselkedéssel, feltárni az esetleges numerikus műtermékeket.

Kevés rendszer van a fizikában, amit analitikusan meg lehet oldani. A harmonikus oszcillátor egy közülük. Mozgásegyenlete:

$$m\ddot{x} = -m\omega^2 x,$$

amely analitikusan integrálható, és a

$$x(t) = x_0 \cos(\omega t) + \frac{v_0}{\omega} \sin(\omega t)$$

megoldást adja.

A harmonikus oszcillátor mozgásegyenlete másodrendű:

$$\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = -\omega^2 x,$$

amely átírható két elsőrendű csatolt egyenletre:

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= v,\\ \frac{\mathrm{d}v}{\mathrm{d}t} &= -\omega^2 x = a.\end{aligned}$$

Ez a rendszer analitikusan megoldható. A gyakorlás kedvéért oldjuk meg a numerikusan a differenciál egyenleteket, és vessük össze az eredményt az ismert formulával. Induljunk ki az $x(0), v(0)$ kezdőállapotból, és $\mathrm{d}t$ időközökkel léptessük a rendszert:

$$\begin{aligned}v(t+\mathrm{d}t) &= v(t) + a(t)\mathrm{d}t,\\ x(t+\mathrm{d}t) &= x(t) + v(t+\mathrm{d}t)\mathrm{d}t.\end{aligned}$$

Az itt alkalmazott *Euler-Cromer-szabály* annyiban más mint eredeti *Euler-algoritmus*, hogy a második egyenletben a frissített $v(t+\mathrm{d}t)$ érték szerepel a $v(t)$ helyett. Az eredeti algoritmus ezen a problémán instabil megoldást ad, és nem őrzi meg az energiát:

$$E = \frac{1}{2}mv^2 + \frac{1}{2}m\omega^2 x^2.$$

Betöltjük a standard headereket

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
```

Deklaráljuk a változókat

```
double omega;            // the natural frequency
double x, v;             // position and velocity at time t
int periods;             // number of periods to integrate
int stepsPerPeriod;      // number of time steps dt per period
string fileName;         // name of output file
```

Deklaráljuk a függvényeket

```
void getInput();                  // for user to input parameters
void EulerCromer(double dt);      // takes an Euler-Cromer step
double energy();                  // computes the energy
```

Függvények kifejtése

```
void getInput ( ) {
    cout << "Enter␣omega:␣";
    cin >> omega;
    cout << "Enter␣x(0)␣and␣v(0):␣";
    cin >> x >> v;
    cout << "Enter␣number␣of␣periods:␣";
    cin >> periods;
    cout << "Enter␣steps␣per␣period:␣";
    cin >> stepsPerPeriod;
    cout << "Enter␣output␣file␣name:␣";
    cin >> fileName;
}

void EulerCromer (double dt) {
    double a = - omega * omega * x;
    v += a * dt;
    x += v * dt;
}

double energy ( ) {
    return 0.5 * (v * v + omega * omega * x * x);
}
```

```cpp
int main ( ) {
    getInput();
    ofstream file(fileName.c_str());
    if (!file) {
        cerr << "Cannot open " << fileName << "\nExiting ...\n";
        return 1;
    }
    const double pi = 4 * atan(1.0);
    double T = 2 * pi / omega;
    double dt = T / stepsPerPeriod;
    double t = 0;
    file << t << '\t' << x << '\t' << v << '\n';
    for (int p = 1; p <= periods; p++) {
        for (int s = 0; s < stepsPerPeriod; s++) {
            EulerCromer(dt);
            t += dt;
            file << t << '\t' << x << '\t' << v << '\n';
        }
        cout << "Period = " << p << "\tt = " << t
             << "\tx = " << x << "\tv = " << v
             << "\tenergy = " << energy() << endl;
    }
    file.close();
}
```

Frissítsük fel a feladat megoldásához szükséges ismereteinket! (g++, `gnuplot`, `pdflatex` használata.) Próbaként nézzük meg a bevezető anyagban lévő harmonikus oszcillátor példakódját! A leírás alapján értelmezzük a példaprogram működését, fordítsuk le a kódot, és futtassuk a szimulációs programot. Ábrázoljuk és elemezzük a kimenő adatokat. Írjunk rövid jegyzőkönyvet, amely bevezetést, összekötő szöveget és ábrákat tartalmaz, legalább a következők tanulmányozásával:

1. A harmonikus oszcillátor kitérés-idő diagramja. (Figyeljünk az ábrán használt skálára.)
2. Ábrázoljuk hosszabb időre a kitérés-sebesség diagramot. Milyen ábrát várunk és mit kapunk e helyett?
3. Nézzük meg, hogy megmarad-e az energia! Teszteljük a sima *Euler-algoritmust* is!
4. Elemezzük, hogyan függ a futási idő a lépések számától. (Használható a `time` parancs.)