Simulace problému N těles

Michal Gregor

ČVUT–FIT

gregormi6@cvut.cz

23. května 2024

1 Úvod

V této semestrální práci jsem se věnoval matematickému problému N-těles. Jde o systém N těles(hmotných bodů), které se navzájem přitahují gravitační silou popsanou Newtonovým gravitačním zákonem. Každá dvojice těles se přitahuje silou

$$F = G\frac{m_1 m_2}{r^2}, \tag{1}$$

kde G je gravitační konstanta, m_1 a m_2 jsou hmotnosti těles a r je jejich vzdálenost. Spojením s 2. Newtonovým zákonem je tedy systém popsán soustavou obyčejných diferenciálních rovnic 2. řádu ve tvaru

$$\ddot{x_i} = G\sum_{j=1,i\neq j}^{N}\frac{m_j}{r_{ij}^2}, \quad i = 1,\ldots,N, \tag{2}$$

$$\dot{x_i}(0) = v_i^0, \quad i = 1,\ldots,N, \tag{3}$$

$$x_i(0) = x_i^0, \quad i = 1,\ldots,N, \tag{4}$$

kde x_i je polohový vektor i-tého tělesa, m_j je hmotnost j-tého tělesa a r_{ij} je vzdálenost i-tého a j-tého tělesa. Musíme zadat počáteční podmínky, a to počáteční polohu a rychlost každého tělesa.

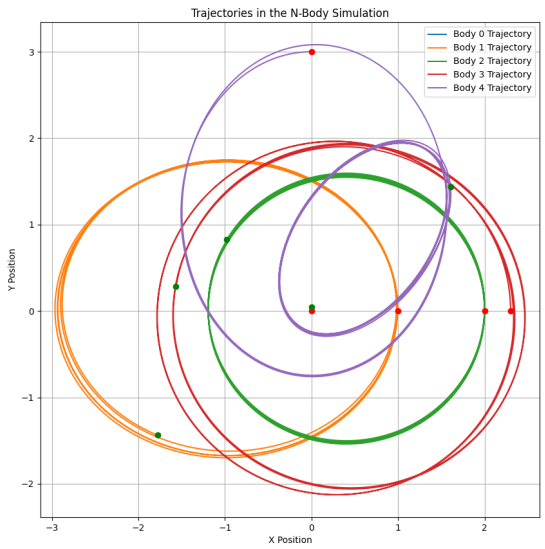
V této práci jsem v pythonu implementoval numerický řešič takovéto soustavy pro obecný počet těles N, libovolné počáteční podmínky a dimenzi prostoru 2 nebo 3, a věnoval jsem se tomu, jak výsledky vykreslovat.

2 Metody/postupy/algoritmy

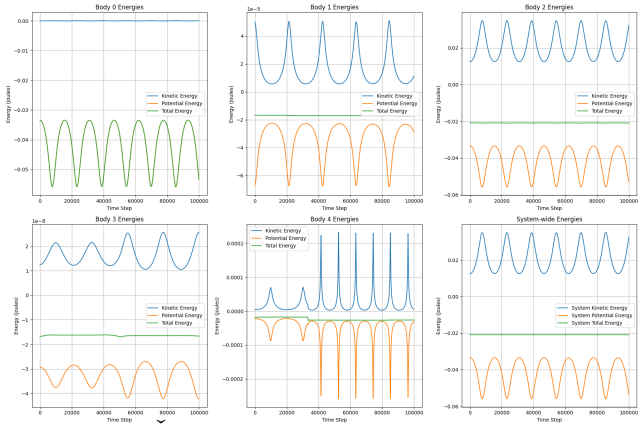
Numerické řešení diferenciálních rovnic spočívá v tom, že si spojitý čas rozsekáme na malé časové dílky a iterativně počítáme hodnotu v dalším časovém bodě z hodnoty v předchozích časech. Nejjednodušší metoda je Eulerova metoda, kdy hodnotu v následujícím časovém kroku spočítáme jako hodnotu v současném kroku plus derivace funkce v současném kroku krát časový přírůstek. Lepší metodou pro tento problém je tzv. leapfrog metoda, kdy derivace funkce (rychlosti) počítáme vždy uprostřed intervalů. Implementoval jsem obě tyto metody. Leapfrog je však pro tento problém mnohem vhodnější, protože mnohem lépe zachovává celkovou energii. Vykreslování výsledků jsem dělal pomocí knihovny matplotlib. Naprogramoval jsem vykreslování trajektorií a časových závislostí kinetických a potenciálních energií těles a celkového systému, a dále jsem také pomocí matplotlib vytvořil skript, který z nasimulovaných dat vytvoří krátké video.

3 Výsledky

Výsledkem mé práce jsou dva Python moduly. Jeden, který dostane počáteční podmínky N těles a další volitelné argumenty simulace, provede simulaci numerickým řešením výše uvedených rovnic a vykreslí časovou závislost energií těles a jejich trajektorie. Druhý modul pak vezme výsledná data z prvního a podle dalších parametrů vytvoří video simulace. Dále jsem připravil několik ukázkových počátečních podmínek pro dimenze 2 i 3 a ve složce example-outputs jsou ukázkové výsledky pro tyto počáteční podmínky. Níže je pro ukázku výsledek trajektorií a časových závislostí energií pro jednu vybranou počáteční podmínku.



Obrázek 1: Trajektorie N-těles



Obrázek 2: Časová závislost kinetických a potenciálních energií

4 Závěr

Problém N-těles je starý a velmi známý problém, takže naprogramovat numerický řešič diferenciálních rovnic, které ho popisují, bylo určitě pěkné programovací cvičení, ale není to nic přelomového. Myslím, že moje práce je ale docela zajímavá hlavně ohledně vykreslování výsledků. Co jsem koukal na implementace dostupné na internetu, tak typicky byl naimplementovaný výpočet simulace a vykreslení trajektorií. Nikde jsem ale neviděl také vykreslování časových závislostí energií, jako jsem to udělal já. Tyto grafy jsou myslím velmi zajímavé a ilustrativní. Je v nich hezky vidět dynamika systému, jak se kinetická energie přelévá do potenciální a naopak, nebo jak tělesa přeskakují na jiné stabilní orbity při blízkém průletu kolem hmotného tělesa. Také je zde dobře vidět, když se nějaké těleso osvobodí, jeho celková energie se stane kladnou a následně těleso odlétá do nekonečna (jako například pro počáteční podmínku 2d-3). Dále si myslím, že se mi docela povedlo vytváření animací, a to i ve třech dimenzích. Animace má několik možných parametrů, jako je třeba její délka a délka čar, které za sebou nechávají tělesa. Ve třech dimenzích je také možné si nastavit, zda-li chcete pro každé těleso vidět i jeho projekci na tři roviny v třírozměrném prostoru. Díky tomu je možné získat mnohem lepší prostorovou představu o tom, jak se tělesa pohybují.

Nevýhodou mé implementace v Pythonu je rychlost programu. Python není zrovna nejrychlejší jazyk a celý výpočet probíhá sériově na CPU. Pro trochu větší množství těles už je to docela pomalé. Jedna simulace pro 13 těles mi běžela asi půl hodiny. Pokud bych chtěl mít těles třeba sto, tak by bylo zapotřebí výpočet paralelizovat a ideálně využívat i GPU. To by se určitě dalo dělat dál.

Původně jsem myslel, že bych ještě mohl udělat tělesa konečně velká a umožnit, aby se srážela a spojovala do větších těles. To by principielně mělo být možné, ale hodně by se to tím celé zkomplikovalo. Teď si myslím, že by to bylo spíš téma na bakalářskou práci.

Výsledky mé práce by se podle mě daly dobře využít jako výuková pomůcka do hodin fyziky na středních školách, když se probírá Newtonův gravitační zákon.

Reference

[1] Matplotlib development team. *Matplotlib Documentation*. 2024. [cit. 2024-05-23] `https://matplotlib.org/stable/index.html`.

[2] Richard Feynmann. The theory of gravitation. online, 1963. [cit. 2024–23–05] `/https://www.feynmanlectures.caltech.edu/I_07.html`.

[3] OpenAI. *Chat GPT*. 2024. [cit. 2024-05-23] `https://chatgpt.com/`. Pomáhal jsem si s GPT pro brainstormování nápadů a uhlazování kódu.