Nonlinear Hamilton Systems

Modelowanie Komputerowe

Gabriela Godek, Gabriela Białoskórska, Wojciech Kura, Ignacy Tekieli
Fizyka Techniczna III.

Czerwiec 2021

Spis treści

1 Wprowadzenie

1.1 Cel pracy

Celem niniejszego projektu była implementacja kodu w wybranym języku programowania (Python/Cpp), na podstawie przykładowych programów zamieszczonych w książce *"The nonlinear workbook"* (Willi-Hans Steeb, 3rd edition). Wybrany przez nas rozdział książki dotyczył nieliniowych systemów Hamiltona. Na potrzeby projektu napisaliśmy dwa programy: `Hamilton Equations of Motion` (zainspirowany kodem w języku cpp) oraz `Heiles - Poincare` (oparty na programie napisanym w języku Java). Pierwszy z nich zaimplementowaliśmy, podobnie jak autor książki, w Cpp, natomiast w przypadku drugiego posłużyliśmy się językiem Python.

1.2 Wstęp teoretyczny

1.2.1 Równania Ruchu Hamiltona

Rozważmy klasyczny, zamknięty układ fizyczny z trzema stopniami swobody (na przykład N cząstek w trójwymiarowym pudle.) Stan takiego układu w pełni określa zbiór 6N niezależnych zmiennych rzeczywistych (gdzie: $p^N = (p_1, p_2, ..., p^N), q^N = (q_1, q_2, ..., q^N)$, odpowiednio p_j i q_j oznaczają pęd i położenie i-tej cząstki.) Jeśli wektor stanu $X^N = X^N(p^N, q^N)$ jest znany w danej chwili czasu, wówczas jest również określony dla innego, dowolnego momentu (zgodnie z Prawami Newtona.) Jeśli dla powyższego układu możemy zdefiniować funkcję Hamiltona $H(X, t)$, ewolucja wartości p_j oraz q_j w czasie wyraża się **Równaniami Ruchu Hamiltona**:

$$\frac{dp_j}{dt} = -\frac{\partial H}{\partial q_j}$$

$$\frac{dq_j}{dt} = -\frac{\partial H}{\partial p_j}$$

Jeżeli funkcja Hamiltona nie zależy wyraźnie od czasu, wówczas jest stałą ruchu:

$$H(X^N) = E$$

gdzie E jest całkowitą energią układu (nazywanym w tym przypadku *zachowawczym.*) Jeśli powiążemy ten układ z przestrzenią fazową, wówczas wektor stanu $X^N = X^N(p^N, q^N)$ określa punkt w przestrzeni fazowej, a w miarę upływu czasu wyznacza w niej trajektorię. W przestrzeni fazowej nie może dojść do przecięcia się dwóch trajektorii; w przeciwnym wypadku nie można byłoby jednoznacznie określić późniejszego ruchu trajektorii. Wobec tego:

$$\frac{dq_{11}}{\frac{\partial H}{\partial p_{11}}} = \frac{dq_{12}}{\frac{\partial H}{\partial p_{12}}} = \frac{dq_{13}}{\frac{\partial H}{\partial p_{13}}} = \frac{dq_{21}}{\frac{\partial H}{\partial p_{21}}} = ... = \frac{dp_{11}}{-\frac{\partial H}{\partial q_{11}}} = ... = \frac{dp_{N3}}{-\frac{\partial H}{\partial q_{N3}}} = dt$$

Ten wzór dostarcza nam 6N - 1 równań, zachodzących pomiędzy współrzędnymi przestrzeni fazowej, które po rozwiązaniu dają nam 6N - 1 całek ruchu:

$$f_j(X^N) = C_j$$

gdzie C_j jest stałą. Powyższe całki ruchu można podzielić na dwa rodzaje: *isolating* i *nonisolating*. 'Isolating integrals' definiują całą powierzchnię w przestrzeni fazowej i są szczególnie ważne w teorii ergodycznej, podczas gdy 'Nonisolating integrals' nie definiują powierzchni i są aż tak nieistotne. Jednym z głównych problemów ergodikoteorii jest określenie, ile 'Isolating integrals' charakteryzuje dany układ. Przykładem takiej całki jest energia całkowita. Dla cząstek N w pudełku jest to prawdopodobnie jedyna taka całka.

Rozważmy szczególny przypadek funkcji Hamiltona ($H : R^4 \longrightarrow R$) z dwoma stopniami swobody:

$$H(p,q) = \frac{1}{2}(p_1^2 + p_2^2) + V(q_1, q_2)$$

Równania ruchu Hamiltona dane są wzorami:

$$\frac{dq_j}{dt} = \frac{\partial H}{\partial p_j} = p_j$$

$$\frac{dp_j}{dt} = -\frac{\partial H}{\partial q_j} = -\frac{\partial V}{\partial q_j}$$

gdzie j = 1, 2. Wykorzystując model Henona-Heilesa mamy:

$$H(p,q) = \frac{1}{2}(p_1^2 + p_2^2 + q_1^2 + q_2^2)_{q^1}^2 q_2 - \frac{1}{3}q_2^3$$

Wobec tego znajdujemy:

$$\frac{dq_1}{dt} = p_1$$

$$\frac{dq_2}{dt} = p_2$$

$$\frac{dp_1}{dt} = -q_1 - 2q_1q_2$$

$$\frac{dp_2}{dt} = -q_2 - q_1^2 + q_2^2$$

1.2.2 Poincare section method

Poincare section method, inaczej zwane *Sufrace-of-section method* ma szczególne zastosowanie w układach Hamiltona z dwoma stopniami swobody. W metodzie tej dochodzi do przechodzenia trajektorii przez powierzchnię przecinającą powłokę energetyczną, np. (p_2, q_2) w punkcie $q_1 = 0$.

2 Materiały i Metody

2.1 Wiadomości wstępne

Poniższe projekty, w oparciu o opisane zjawiska, powstały na podstawie programów umieszczonych w książce *"The nonlinear workbook"* (Willi-Hans Steeb, 3rd edition). Pierwszy z nich (Hamilton Equations of Motion) został napisany w języku Cpp, z tą jednak różnicą, że w oryginalnym programie zawartym w książce błędnie opisane zostały ścieżki plików, co naprawiono. Drugi program (Heiles - Poincare) zaimplementowano w języku Python, na podstawie zamieszczonego w książce kodu w Java. W tym celu wykorzystano bibliotekę `Matplotlib`.

2.2 Implementacja

2.2.1 Hamilton Equations of Motion

Poniższy kod umożliwia obliczenie pochodnych, wyprowadzonych we wstępie teoretycznym (dq_1/dt, dq_2/dt, dp_1/dt, dp_2/dt).

Implementacja:

```
// hamiltoneq.cpp

#include <iostream>
#include "headers/symbolicc++2.h"
using namespace std;

int main(void)
{
  Symbolic h("h"), q1("q1"), q2("q2"), p1("p1"), p2("p2"),
  pt1, pt2, qt1, qt2;

  // Hamiltonian
  h = (p1*p1+p2*p2+q1*q1+q2*q2)/2+q1*q1*q2-q2*q2*q2/3;
  // R wnania ruchu Hamiltona
  pt1 = -df(h,q1); pt2 = -df(h,q2);
  qt1 = df(h,p1);  qt2 = df(h,p2);

  cout << "dp1/dt = " << pt1 <<endl;
  cout << "dp2/dt = " << pt2 <<endl;
  cout << "dq1/dt = " << qt1 <<endl;
  cout << "dq2/dt = " << qt2 <<endl;

  return 0;
}
```

2.2.2 Heiles - Poicare

Poniższy kod umożliwia oszacowanie powierzchni przekroju dla energii E = 1/12, dla dwóch różnych warunków początkowych. Warunki początkowe mają następującą postać:

$$q_1(t=0) = \frac{1}{\sqrt{8}} \quad (1) \qquad q_2(t=0) = 0 \quad (2)$$

$$p_1(t=0) = 0 \quad (3) \qquad p_2(t=0) = \frac{1}{\sqrt{24}} \quad (4)$$

oraz

$$q_1(t=0) = 0 \quad (5) \qquad q_2(t=0) = 0 \quad (6)$$

$$p_1(t=0) = \frac{1}{\sqrt{6}} \quad (7) \qquad q_2(t=0) = 0 \quad (8)$$

Znajdujemy dwie niezmiennicze krzywe. Program `mainHHH.py` jest głównie odpowiedzialny za rysowanie wykresów, natomiast `HenonHeiles.py` oblicza opisane we wstępie funkcje.

Implementacja:

`mainHHH.py`

```
from matplotlib import pyplot
from scipy import *
from numpy import *

from HenonHeiles import *

t = linspace(0, 100.0, 1000)
print ("Wielkosc kroku " + str(t[1]-t[0]))

# Cztery warunki poczatkowe dla E = 1/12 w ukladzie Henona-Heilesa
init_cons = [array([0.0, 0.5, 0.0, 0.0]),\
             array([0.408248, 0.0, 0.0, 0.0]),\
             array([0.0, 0.0, 0.408248, 0.0]),\
             array([0.0, 0.0, 0.0, 0.408248])\
             ]

outs = list()
```

```
for con in init_cons:
    outs.append(rungekutta4(punkt, t, con))

# Rysowanie wynikow
fig1 = figure(1)
for ii in xrange(4):
    subplot(2, 2, ii+1)
    plot(outs[ii][:,1],outs[ii][:,3])
    ylabel("py")
    xlabel("y")
    title("Pelna trajektoria")

fig1.suptitle('Pelna trajektoria przy E = 1/12', fontsize=20)


fig2 = figure(2)
for ii in xrange(4):
    subplot(2, 2, ii+1)
    xcrossings = punkt0(outs[ii][:,0])
    yints = [.5*(outs[ii][cross, 1] + outs[ii][cross+1, 1])
    for cross in xcrossings]
    pyints = [.5*(outs[ii][cross, 3] + outs[ii][cross+1, 3])
    for cross in xcrossings]
    plot(yints, pyints,'.')
    ylabel("py")
    xlabel("y")
    title("Poincare")

fig2.suptitle('Poincare E = 1/12', fontsize=20)

show()
```

HenonHeiles.py

```
from matplotlib.pyplot import *
from scipy import *
from numpy import *

# Wykorzystanie algorytmu Rungego–Kutty 4. rzedu
def rungekutta4 (yprime, time, y0):
    # yprime to lista funkcji, y0 to lista wartosci dla y
    # time to lista wartosci t z generowanymi wartosciami

    N = len(time)

    y = array([thing * ones(N) for thing in y0]).T

    for ii in xrange(N - 1):
        dt = time[ii + 1] - time[ii]
        k1 = dt * yprime(y[ii], time[ii])
        k2 = dt * yprime(y[ii] + 0.5 * k1, time[ii] + 0.5 * dt)
        k3 = dt * yprime(y[ii] + 0.5 * k2, time[ii] + 0.5 * dt)
        k4 = dt * yprime(y[ii] + k3, time[ii + 1])
        y[ii + 1] = y[ii] + (k1 + 2.0 * (k2 + k3) + k4) / 6.0

    return y

# warpar - wartosci par
# twar = wartosci czasu
def enrgia(warpar):
    (x, y, px, py) = tuple(warpar)
    return .5 * (px ** 2 + py ** 2) + .5 * (x ** 2 + y ** 2 + 2.
    * x ** 2. * y - (2.0 / 3) * y ** 3)

def punkt(warpar,twar):
    # zwraca pary (x,y) dla Henona–Heilesa Hamiltonianu
    (x, y, px, py) = tuple(warpar)
    return array([px, py, -x - 2 * x * y, -y - x * x + y * y]).T

def punkt0(data):
    # znajduje indeksy gdzie punkty przecinaja sie w zerze
    p0 = list()
    for ii in xrange(len(data) - 1):
        if (data[ii] > 0) & (data[ii + 1] < 0):
            p0.append(ii)
        if (data[ii] < 0) & (data[ii + 1] > 0):
            p0.append(ii)
    return array(p0)
```

3 Wyniki

3.1 Hamilton Equations of Motion

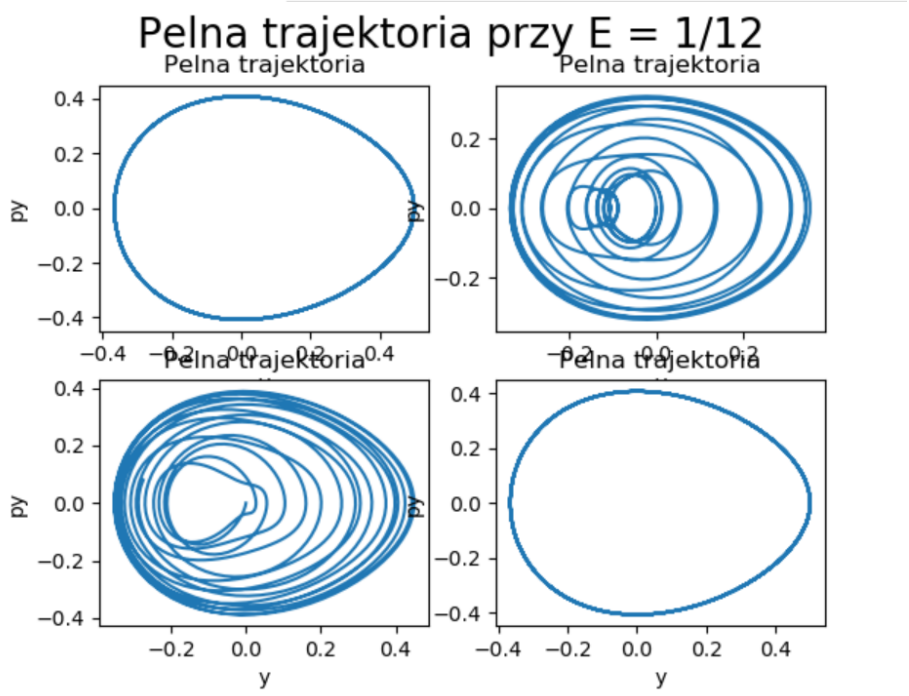
Otrzymane wartości pochodnych wyglądają następująco:

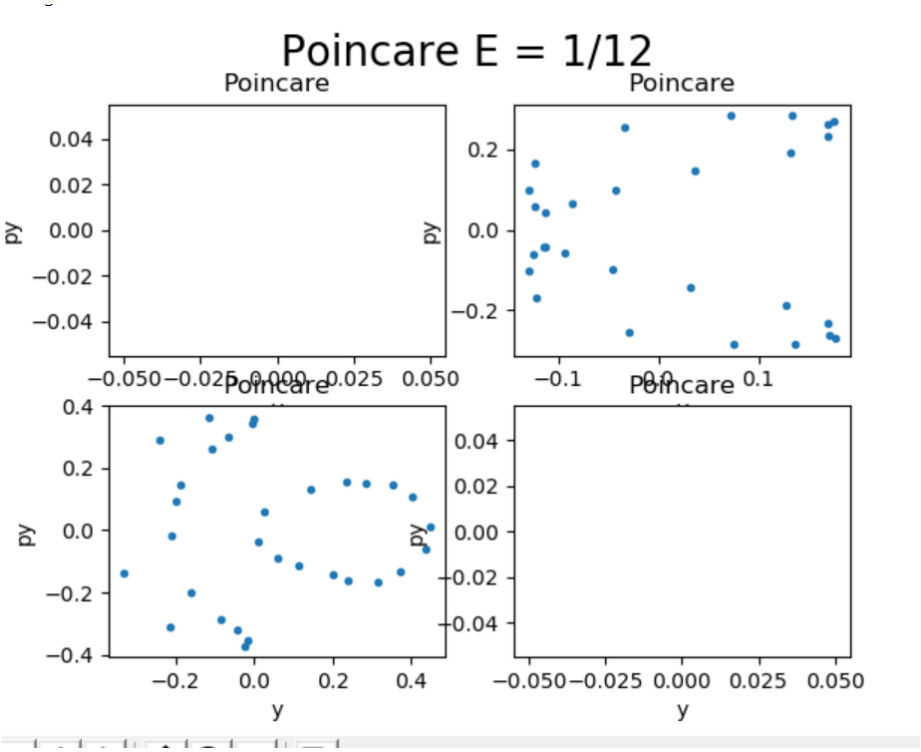
```
[Running] cd "c:\Users\straz\Documents\cpp\" && g++ hamiltoneq.cpp -o h
dp1/dt = -q1-2*q1*q2
dp2/dt = -q2-q1^(2)+q2^(2)
dq1/dt = p1
dq2/dt = p2

[Done] exited with code=0 in 9.323 seconds
```

3.2 Heiles - Poincare

Otrzymane wizualizacje wyników wyglądają następująco:





4 Podsumowanie

W projekcie wykonano dwa programy, oparte na kodach źródłowych zamieszczonych w literaturze. Pierwszy z nich (Hamilton Equations of Motion), napisany w języku Cpp, prawidłowo oblicza wartości pochodnych, opisanych we wstępie teoretycznym. Drugi natomiast (Heiles - Poincare), napisany w języku Python, zawiera dwa osobne programy. `mainHHH.py` umożliwia użytkownikowi otrzymanie wartości samych funkcji, `HenonHeiles.py` natomiast narysowanie trajektorii. Również działają one w sposób prawidłowy. Kod został czytelnie opisany, aby krok po kroku można było śledzić jego działanie.

5 Bibliografia

Literatura wykorzystana do napisania programów oraz wstępu:

- *"The nonlinear workbook"* - Willi-Hans Steeb, 3rd edition
  Chapter 4 - *Nonlinear Hamilton Systems*