Układy autonomiczne na płaszczyźnie & Rozwiązywanie równań różniczkowych

Emil Śmiech,
Amelia Królczyk,
Michał Palwal,
Sabina Adamska

Czerwiec 2021

Układy autonomiczne na płaszczyźnie

Klasyfikacja punktów stałych

W pierwszej części naszego projektu omówimy autonomiczne układy równań różniczkowych zwyczajnych w płaszczyźnie:

$$\frac{du_1}{dt} = f_1(u_1, u_2), \qquad \frac{du_2}{dt} = f_2(u_1, u_2)$$

gdzie $f_1, f_2 \in C^2$. Punkty stałe $\{u_1^*, u_2^*\}$, są podane jako rozwiązanie równania:

$$f_1(u_1^*, u_2^*) = 0, \qquad f_2(u_1^*, u_2^*) = 0.$$

Dla przykładu układ:

$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = -u_1 + \mu(1 - u_1^2)u_2, \qquad \mu \in \mathbf{R}$$

ma tylko jeden punkt stały $\{u_1^*, u_2^*\} = (0, 0)$. Układ:

$$\frac{du_1}{dt} = u_1(1 - u_1^2 - u_2^2), \qquad \frac{du_2}{dt} = u_2(1 - u_1^2 - u_2^2)$$

ma dwa punkty stałe - kolektor (okrąg) $1 - u_1^{*2} - u_2^{*2} = 0$ i punkt $\{u_1^*, u_2^*\} = (0, 0)$. Równanie zlinearyzowane (zwane także równaniem wariacyjnym) jest dane przez:

$$\begin{aligned} \frac{dv_1}{dt} &= \frac{\partial f_1(\mathbf{u}(t))}{\partial u_1} v_1 + \frac{\partial f_1(\mathbf{u}(t))}{\partial u_2} v_2 \\ \frac{dv_2}{dt} &= \frac{\partial f_2(\mathbf{u}(t))}{\partial u_1} v_1 + \frac{\partial f_2(\mathbf{u}(t))}{\partial u_2} v_2 . \end{aligned}$$

Jeśli założymy, że jedynym punktem stałym jest początek $(0, 0)$, to równanie zlinearyzowane uprości się do:

$$\frac{dv_1}{dt} = av_1 + bv_2, \qquad \frac{dv_2}{dt} = cv_1 + dv_2$$

gdzie:

$$a := \frac{\partial f_1}{\partial u_1}(0,0), \qquad b := \frac{\partial f_1}{\partial u_2}(0,0), \qquad c := \frac{\partial f_2}{\partial u_1}(0,0), \qquad d := \frac{\partial f_2}{\partial u_2}(0,0) .$$

Oczekujemy, że rozwiązania tego równania będą geometrycznie podobne do rozwiązań pierwotnego układu w pobliżu punktu $(0, 0)$, co w większości przypadków zostanie spełnione. Nietrywialne rozwiązania istnieją wtedy i tylko wtedy, gdy:

$$\det \begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix} = 0 .$$

Z tego wynika, że:

$$\lambda^2 - (a + d)\lambda + (ad - bc) = 0$$

co nazywa się równaniem charakterystycznym. Gdy ma ono dwa różne pierwiastki, λ_1, λ_2, generowane są dwie liniowo niezależne rodziny rozwiązań. Definiujemy:

$$p := a + d, \qquad q := ad - bc.$$

Równanie charakterystyczne przyjmuje postać:

$$\lambda^2 - p\lambda + q = 0.$$

Niech $\Delta := p^2 + 4q$ będzie dyskryminatorem. Następnie pierwiastki λ_1, λ_2 są dane przez:

$$\lambda_1 = \frac{1}{2}(p + \Delta^{1/2}), \qquad \lambda_2 = \frac{1}{2}(p - \Delta^{1/2}).$$

W poniższej tabeli wymieniono możliwe przypadki.

(i)	λ_1, λ_2	real, unequal, same sign	$\Delta > 0, q > 0$	Node
(ii)	$\lambda_1 = \lambda_2$	(real) $b \neq 0, c \neq 0$	$\Delta = 0, p \neq 0$	Inflected Node
(iii)	λ_1, λ_2	complex, non-zero real part	$\Delta < 0, p \neq 0$	Spiral
(iv)	$\lambda_1 \neq 0, \lambda_2 = 0$		$q = 0$	Parallel Lines
(v)	λ_1, λ_2	real, different sign	$q < 0$	Saddle Point
(vi)	λ_1, λ_2	pure imaginary	$q > 0, p = 0$	Centre

Klasyfikując punkty stałe przyjęliśmy za pewnik nieudowodnione założenie, że ścieżki fazowe równania pierwotnego i zlinearyzowanego mają ten sam charakter w pobliżu punktu stałego. Dotyczy to ogólnie spiral, węzłów i punktów siodłowych, ale nie środka. Na przykład przybliżenie liniowe może przewidzieć środek, w którym pierwotne równanie miało spiralę. Odwrotnie, układ równań różniczkowych:

$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = -u_1^3$$

ma środek na początku, ale zlinearyzowany układ równań:

$$\frac{dv_1}{dt} = v_2, \qquad \frac{dv_2}{dt} = 0$$

już nie ma.

Orbita homokliniczna

Układy anharmoniczne z orbitą homoklinicznną odgrywają szczególną rolę w badaniu układów chaotycznych. Orbity te dążą do tego samego stałego punktu zarówno dla $t \to \infty$, jak i $t \to -\infty$. Jako przykład rozważymy układ anharmoniczny

$$\frac{d^2u}{dt^2} - \frac{u}{2} + u^2 + u^3 = 0.$$

Wprowadzając $u_1 := u$ i $u_2 := du/dt$ otrzymujemy układ autonomiczny w płaszczyźnie:

$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = \frac{u_1}{2} - u_1^2 - u_1^3.$$

Trzy stałe punkty tego układu są podane przez:

$$(u_1^*, u_2^*) = (0,0), \quad (u_1^*, u_2^*) = (-(\sqrt{3}+1)/2, 0), \quad (u_1^*, u_2^*) = ((\sqrt{3}-1)/2, 0)\,.$$

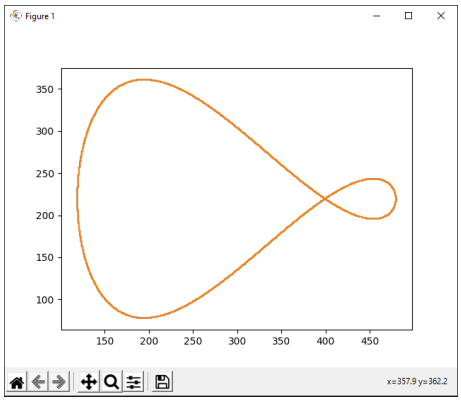
Półpłaszczyzna $u_1 > 0$ zawiera unikalną homoklinniczną orbitę $\Gamma(t) = (u_1(t), u_2(t))$ daną przez:

$$u(t) \equiv u_1(t) = \frac{2e^{t/\sqrt{2}}}{(e^{t/\sqrt{2}} + \frac{2}{3})^2 + 1}.$$

Punkt stały (0,0) jest hiperbolicznym punktem stałym, a pozostałe dwa punkty stałe są eliptyczne. Dla (0,0) wartości własne macierzy funkcjonalnej są rzeczywiste, a dla dwóch pozostałych punktów stałych wartości własne są czysto urojone.

W programie Homoclinic.py obliczono orbitę homoklinniczną podanego powyżej równania różniczkowego.

```python
import matplotlib.pyplot as plt
import math



t = 0.0001
count = 0.0
u11 = -0.001
u22 = 0.001
m1_tab = []
n1_tab = []
m2_tab = []
n2_tab = []

while count < 42.5:
    u1 = u11
    u2 = u22
    V2 = u1 * (0.5 - u1 - u1 * u1)
    u11 = u1 + t * u2 + t * t * V2 / 2.0
    u22 = u2 + t * (u1 / 2.0 - u1 * u1 - u1 * u1 * u1) + \
        t * t * u2 * (0.5 - 2.0 * u1 - 3.0 * u1 * u1) / 2.0
    m1 = math.floor(150 * u1 + 400)
    m1_tab.append(m1)
    n1 = math.floor(150 * u2 + 220)
    n1_tab.append(n1)
    m2 = math.floor(150 * u11 + 400)
    m2_tab.append(m2)
    n2 = math.floor(150 * u22 + 220)
    n2_tab.append(n2)
    count += t

plt.plot(m1_tab, n1_tab)
plt.plot(m2_tab, n2_tab)
plt.show()
```



Wahadło

Równanie dla wahadła jest wyrażone przez:

$$\frac{d^2u}{dt^2} + \omega^2 \sin(u) = 0, \qquad \omega^2 := \frac{g}{L}$$

gdzie **L** jest długością wahadła, **g** jest przyspieszeniem ziemskim, a **u** jest kątowym przemieszczeniem wahadła z jego położenia równowagi. Wprowadzając wielkości $\tilde{u}(\tilde{t}(t)) = u(t)$, $\tilde{t}(t) = \omega t$ możemy zapisać równanie wahadła w postaci bezwymiarowej:

$$\frac{d^2\tilde{u}}{d\tilde{t}^2} + \sin(\tilde{u}) = 0\,.$$

W dalszej części pomijamy tyldę. Zakładając, że $u_1 = u$, oraz $u_2 = du/dt = du_1/dt$ otrzymujemy układ autonomiczny pierwszego rzędu:

$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = -\sin(u_1)\,.$$

W ten sposób otrzymujemy punkty stałe $(n\pi, 0)$, $n \in \mathbf{Z}$. Równanie wariacyjne jest dane przez:

$$\frac{dv_1}{dt} = v_2, \qquad \frac{dv_2}{dt} = -\cos(u_1)v_1\,.$$

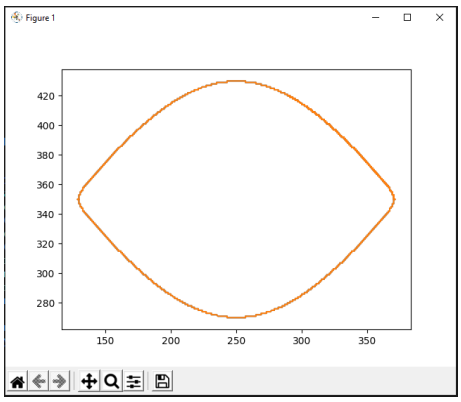
Wstawiając punkty stałe do powyższych równań dostajemy:

$$\frac{dv_1}{dt} = v_2, \qquad \frac{dv_2}{dt} = \begin{cases} -v_1 \text{ if } n \text{ even} \\ v_1 \text{ if } n \text{ odd} \end{cases}$$

W pierwszym przypadku wartości własne to i, $-i$, tak więc mamy środek. W drugim przypadku wartości własne to 1, -1. W ten sposób mamy niestabilny węzeł.

W programie Java Pendulum.py zintegrowano układ dynamiczny za pomocą integratora symplektycznego.

```python
import matplotlib.pyplot as plt
import math


t = 0.001
steps = 20000
q = 1.0
p = 1.75
mx1_tab = []
ny1_tab = []
mx_tab = []
ny_tab = []


for i in range(steps):
    q1 = q
    p1 = p
    q = q1 + t * p1
    p = p1 - t * math.sin(q)
    mx1 = math.floor(40 * q1 + 250 + 0.5)
    mx1_tab.append(mx1)
    ny1 = math.floor(40 * p1 + 350 + 0.5)
    ny1_tab.append(ny1)
    mx = math.floor(40 * q + 250 + 0.5)
    mx_tab.append(mx)
    ny = math.floor(40 * p + 350 + 0.5)
    ny_tab.append(ny)

plt.plot(mx1_tab, ny1_tab)
plt.plot(mx_tab, ny_tab)
plt.show()
```



Układy cyklu granicznego

Dany jest na płaszczyźnie układ autonomiczny równań różniczkowych pierwszego rzędu:

$$\frac{d^2u}{dt^2} + f(u)\frac{du}{dt} + g(u) = 0\,.$$

Niech:

$$F(u) = \int_0^u f(s)ds\,.$$

Załóżmy, że f jest parzystą, a g jest nieparzystą funkcją, obie są ciągłe dla wszystkich u i g spełnia warunek Lipschitza. Załóżmy dalej, że:

$$f(0) < 0, \qquad ug(u) > 0 \quad \text{for} \quad u \neq 0, \qquad F(u) \to \pm\infty \quad \text{if} \quad u \to \infty$$

Oraz, że f ma pojedyncze zero przy $u = b\ (u > 0)$ i rośnie monotonicznie dla $u \geq b$.. Wtedy układ:

$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = -f(u_1)u_2 - g(u_1)$$

ma stabilny cykl graniczny. Jako pierwszy przykład rozważymy słynne równanie Van der Pola:

$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = -u_1 + \mu(1 - u_1^2)u_2\,.$$

Równanie Van der Pola przyjmuje tylko jeden punkt stały, a mianowicie (0,0). Równanie wariacyjne jest podane przez:

$$\frac{dv_1}{dt} = v_2, \qquad \frac{dv_2}{dt} = -(1 + 2\mu u_1 u_2)v_1 + \mu(1 - u_1^2)v_2\,.$$

Wstawienie punktu stałego (0,0) skutkuje:

$$\frac{dv_1}{dt} = v_2, \qquad \frac{dv_2}{dt} = -v_1 + \mu v_2\,.$$

Zatem dla $\mu > 0$ wartości własne mają dodatnią część rzeczywistą. Stąd punkt stały (0,0) jest niestabilny.

W programie vdpol.cpp obliczono portret fazowy równania Van der Pola za pomocą techniki Runge-Kutta. Wartość parametru to $\mu = 5.0$..

```cpp
#include <fstream>
using namespace std;

const int N = 2;

void fsystem(double h, double t, double u[N], double hf[N])
{
    double mu = 5.0;
    hf[0] = h * u[1];
    hf[1] = h * (-u[0] + mu * (1.0 - u[0] * u[0]) * u[1]);
}

void map(double u[N], int steps, double h, double t)
{
    double uk[N];
    double tk;
    double a[6] = {0.0, 1.0 / 4.0, 3.0 / 8.0, 12.0 / 13.0, 1.0, 1.0 / 2.0};
    double c[6] = {16.0 / 135.0, 0.0, 6656.0 / 12825.0, 28561.0 / 56430.0,
                   -9.0 / 50.0, 2.0 / 55.0};
    double b[6][5];
    b[0][0] = b[0][1] = b[0][2] = b[0][3] = b[0][4] = 0.0;
    b[1][0] = 1.0 / 4.0;
    b[1][1] = 0.0;
    b[1][2] = 0.0;
    b[1][3] = 0.0;
    b[1][4] = 0.0;
    b[2][0] = 3.0 / 32.0;
    b[2][1] = 9.0 / 32.0;
    b[2][2] = 0.0;
    b[2][3] = 0.0;
    b[2][4] = 0.0;
    b[3][0] = 1932.0 / 2197.0;
    b[3][1] = -7200.0 / 2197.0;
    b[3][2] = 7296.0 / 2197.0;
    b[3][3] = b[3][4] = 0.0;
    b[4][0] = 439.0 / 216.0;
    b[4][1] = -8.0;
    b[4][2] = 3680.0 / 513.0;
    b[4][3] = -845.0 / 4104.0;
    b[4][4] = 0.0;
    b[5][0] = -8.0 / 27.0;
    b[5][1] = 2.0;
    b[5][2] = -3544.0 / 2565.0;
    b[5][3] = 1859.0 / 4104.0;
    b[5][4] = -11.0 / 4.0;
```

```cpp
    double f[6][N];
    int i, j, l, k;

    for (i = 0; i < steps; i++)
    {
        fsystem(h, t, u, f[0]);
        for (k = 1; k <= 5; k++)
        {
            tk = t + a[k] * h;
            for (l = 0; l < N; l++)
            {
                uk[l] = u[l];
                for (j = 0; j <= k - 1; j++)
                    uk[l] += b[k][j] * f[j][l];
            }
            fsystem(h, tk, uk, f[k]);
        }
        for (l = 0; l < N; l++)
            for (k = 0; k < 6; k++)
                u[l] += c[k] * f[k][l];
    }
}
```

```cpp
int main(void)
{
    ofstream data;
    data.open("phase_data.dat");
    int steps = 1;
    double h = 0.005;
    double t = 0.0;
    double u[N] = {0.1, 0.2}; // initial conditions
    int i;
    // wait for transients to decay
    for (i = 0; i < 1000; i++)
    {
        t += h;
        map(u, steps, h, t);
    }

    t = 0.0;
    for (i = 0; i < 40000; i++)
    {
        t += h;
        map(u, steps, h, t);
        data << u[0] << " " << u[1] << "\n";
    }
    data.close();
    return 0;
}
```

Jako drugi przykład rozważamy układ:

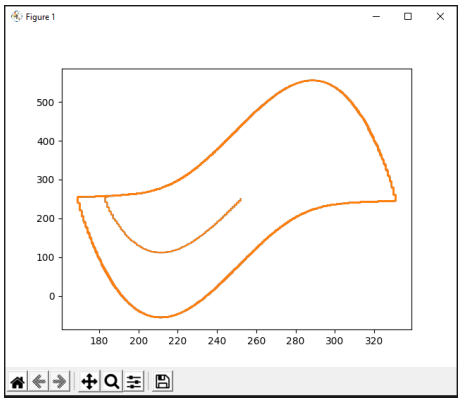
$$\frac{du_1}{dt} = u_2, \qquad \frac{du_2}{dt} = -u_1 - \mu \sin(u_2)\,.$$

Układ przedstawia nieskończenie wiele cykli granicznych. Zakładamy, że $\mu > 0.$. Wtedy (0,0) jest niestabilnym

punktem stałym.

W programie LimitCycles.py obliczono portret fazowy dla różnych wartości początkowych. Rozważaliśmy dwa różne warunki początkowe. Jeden blisko początku, a drugi między pierwszym a drugim cyklem granicznym. Stosujemy integrację symplektyczną.

```python
import matplotlib.pyplot as plt
import math

t = 0.005
T = 10000
x = 0.05
y = 0.01
mu = 5.0
mx1_tab = []
ny1_tab = []
mx_tab = []
ny_tab = []
for i in range(T):
    x1 = x
    y1 = y
    x = x1 + t * y1
    y = y1 * math.exp(-t * mu * (x * x - 1.0)) - t * x
    mx = math.floor(40 * x + 250 + 0.5)
    mx_tab.append(mx)
    ny = math.floor(40 * y + 250 + 0.5)
    ny_tab.append(ny)
    mx1 = math.floor(40 * x1 + 250 + 0.5)
    mx1_tab.append(mx1)
    ny1 = math.floor(40 * y1 + 250 + 0.5)
    ny1_tab.append(ny1)

plt.plot(mx1_tab, ny1_tab)
plt.plot(mx_tab, ny_tab)
plt.show()
```



Układy Lotka-Volterra

Układy Lotka-Volterra opisują interakcję dwóch (lub więcej) konkurujących ze sobą gatunków. Najprostszy model jest dany przez:

$$\frac{du_1}{dt} = au_1 - bu_1u_2, \qquad \frac{du_2}{dt} = -cu_2 + bu_1u_2$$

gdzie a, b, c są dodatnimi stałymi. Dzięki określeniu au_1 gatunek 1 urósłby wykładniczo. Jednak dzięki określeniu —bu_1 u_2 gatunek 1 zmniejszy się. Podobnie dla gatunku 2 termin – cu_2 będzie malał wykładniczo, ale termin bu_1u_2 daje coraz większy udział. Dlatego oczekujemy, że rozwiązanie jest okresowe wokół punktu stałego $u_1^* = c/b$, $u_2^* = a/b$. Jako przykład rozważymy przypadek specjalny ($a = b = c = 1$):

$$\frac{du_1}{dt} = u_1 - u_1u_2, \qquad \frac{du_2}{dt} = -u_2 + u_1u_2$$

gdzie $u_1(t = 0) > 0$ i $u_2(t = 0) > 0$. Ponieważ zakładamy, że $u_2(t) > 0$ i $u_2(t) > 0$, okazuje się, że układ ma jeden stały punkt w $(u_1^*, u_2^*) = (1, 1)$. Punkt stały to środek, więc w sąsiedztwie tego stałego punktu rozwiązania są w przybliżeniu okręgami. Całkowanie równania:

$$\frac{du_1}{u_1 - u_1u_2} = \frac{du_2}{-u_2 + u_1u_2}$$

daje stałą ruchu

$$\ln(u_1) + \ln(u_2) - u_1 - u_2 = C$$

lub:

$$u_1u_2e^{-u_1}e^{-u_2} = C_1\,.$$

Tę stałą można wykorzystać do sprawdzenia dokładności całkowania.

```cpp
#include <fstream>
using namespace std;

const int N = 2;

void fsystem(double h, double t, double u[N], double hf[N])
{
    hf[0] = h * (u[0] - u[0] * u[1]);
    hf[1] = h * (-u[1] + u[0] * u[1]);
}

void map(double u[N], int steps, double h, double t)
{
    double uk[N];
    double tk;
    double a[6] = {0.0, 1.0 / 4.0, 3.0 / 8.0, 12.0 / 13.0, 1.0, 1.0 / 2.0};
    double c[6] = {16.0 / 135.0, 0.0, 6656.0 / 12825.0, 28561.0 / 56430.0, -9.0 / 50.0, 2.0 / 55.0};
    double b[6][5];
    b[0][0] = b[0][1] = b[0][2] = b[0][3] = b[0][4] = 0.0;
    b[1][0] = 1.0 / 4.0;
    b[1][1] = 0.0;
    b[1][2] = 0.0;
    b[1][3] = 0.0;
    b[1][4] = 0.0;
    b[2][0] = 3.0 / 32.0;
    b[2][1] = 9.0 / 32.0;
    b[2][2] = 0.0;
    b[2][3] = 0.0;
    b[2][4] = 0.0;
    b[3][0] = 1932.0 / 2197.0;
    b[3][1] = -7200.0 / 2197.0;
    b[3][2] = 7296.0 / 2197.0;
    b[3][3] = b[3][4] = 0.0;
    b[4][0] = 439.0 / 216.0;
    b[4][1] = -8.0;
    b[4][2] = 3680.0 / 513.0;
    b[4][3] = -845.0 / 4104.0;
    b[4][4] = 0.0;
    b[5][0] = -8.0 / 27.0;
    b[5][1] = 2.0;
    b[5][2] = -3544.0 / 2565.0;
    b[5][3] = 1859.0 / 4104.0;
    b[5][4] = -11.0 / 4.0;
```

```
    double f[6][N];
    int i, j, l, k;
    for (i = 0; i < steps; i++)
    {
        fsystem(h, t, u, f[0]);
        for (k = 1; k <= 5; k++)
        {
            tk = t + a[k] * h;
            for (l = 0; l < N; l++)
            {
                uk[l] = u[l];
                for (j = 0; j <= k - 1; j++)
                    uk[l] += b[k][j] * f[j][l];
            }
            fsystem(h, tk, uk, f[k]);
        }
        for (l = 0, l < N; l++;)
            for (k = 0, k < 6; k++;)
                u[l] += c[k] * f[k][l];
    }
}

int main(void)
{
    ofstream data;
    data.open("phase_data.dat");

    int steps = 1;
    double h = 0.001; // step length
    double t = 0.0;
    double u[N] = {2.0, 1.5}; // initial conditions

    for (int i = 0; i < 20000; i++)
    {
        t += h;
        map(u, steps, h, t);
        data << u[0] << " " << u[1] << "\n";
    }
    data.close();
    return 0;
}
```

W następującym programie pokazano, że

$$\ln(u_1) + \ln(u_2) - u_1 - u_2 = C$$

jest stałą ruchu w układzie Lotka-Volterra.

```
#include <iostream>
#include "symbolicc++.h"

using namespace std;

int main(void)
{
    Sum<double> f("f", 0), u1("u1", 0), u2("u2", 0), t("t", 0), r("r", 0);
    f.depend(u1);
    f.depend(u2);
    u1.depend(t);
    u2.depend(t);
    f = ln(u1) + ln(u2) - u1 - u2;
    r = df(f, t);
    cout << "r = " << r << endl;
    r.put(df(u1, t), u1 - u1 * u2);
    r.put(df(u2, t), -u2 + u1 * u2);
    cout << "r = " << r << endl;
    r.expand();
    cout << "r = " << r;
    return 0;
}
```

Rozwiązywanie równań różniczkowych

W drugiej części projektu poruszymy temat numerycznych metod całkowania nieliniowych równań różniczkowych zwyczajnych. Prawie wszystkich nieliniowych równań różniczkowych nie da się rozwiązać w formie zamkniętej, więc należy zastosować metody numeryczne. Przy rozwiązywaniu równań różniczkowych numerycznie napotykamy dużo problemów. Na przykład, jeśli równania różniczkowe pochodzą z układu Hamiltona, to schemat numeryczny powinien zachować całkowitą energię, która jest stałą ruchu. W przypadku nieliniowego równania różniczkowego może się również zdarzyć, że schemat dyskretyzacji prowadzi do chaotycznego odwzorowania, mimo że pierwotne nieliniowe równanie różniczkowe nie wykazywało zachowań chaotycznych. Stabilne punkty stałe nieliniowych równań różniczkowych mogą stać się niestabilne w ramach schematu dyskretyzacji.

Uważamy za autonomiczne

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u})$$

i nieautonomiczne układy równań różniczkowych zwyczajnych

$$\frac{d\mathbf{u}}{dt} = \mathbf{g}(\mathbf{u}, t)\,.$$

Zakładamy, że mamy problem z wartością początkową $u(t = 0) = u_0$ i że funkcje f i g są analityczne. Omówiono tu metodę Eulera, technikę serii Lie (która jest ściśle związana z techniką szeregów Taylora), metodę piątego rzędu Runge-Kutta-Fehlberga i całkowanie symplektyczne. Przedstawiono również metodę Verleta ważną w dynamice molekularnej. Zbadano również fałszywe rozwiązania i niewidzialny chaos. Na koniec omówiono całkowanie numeryczne układów równań różniczkowych zwyczajnych, które przyjmują pierwsze całki.

Metoda Eulera

Metoda Eulera do numerycznego rozwiązywania układów równań różniczkowych zwyczajnych pierwszego rzędu:

$$\frac{d\mathbf{u}}{dt} = \mathbf{g}(\mathbf{u}, t)$$

polega na obliczeniu dyskretnego zbioru wartości uk, dla argumentów tk, przy użyciu równania różnicowego:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + h\mathbf{g}(t_k, \mathbf{u}_k), \qquad h := t_{k+1} - t_k\,.$$

Tutaj h to długość kroku. Rozszerzając u(t + h) w szeregu Taylora z resztą około t, można zauważyć, że:

$$\frac{\mathbf{u}(t+h) - \mathbf{u}(t)}{h} = \frac{\mathbf{u}(t) + h\mathbf{u}'(t) + \frac{1}{2}h^2\mathbf{u}''(\xi) - \mathbf{u}(t)}{h} = \mathbf{u}'(t) + \frac{1}{2}h\mathbf{u}''(\xi)$$

$(t \leq \xi \leq t+h)$ więc lewa strona metody Eulera jest przybliżeniem $O(h)$ do pochodnej, którą zastępuje, pod warunkiem, że druga pochodna rozwiązania u jest ograniczona w interesującym przedziale czasu. Gdy h dąży do zera, przybliżenie Eulera coraz lepiej przedstawia oryginalne równanie różniczkowe; to znaczy jest zgodny z równaniem różniczkowym.

Jako przykład rozważmy nieliniowe równanie różniczkowe

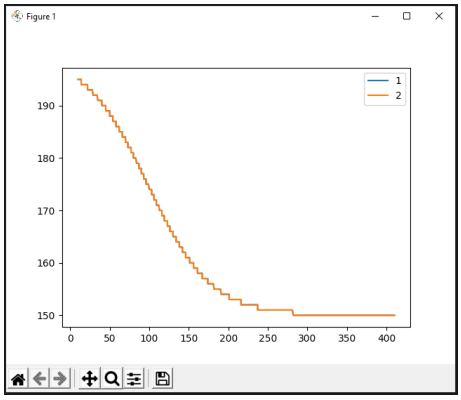
$$\frac{du}{dt} = u(1 - u)$$

z warunkiem początkowym $u(t = 0) = u_0 > 0.$. To nieliniowe równanie różniczkowe przyjmuje punkty stałe $u_1^\star = 0,\ u_2^\star = 1.$. Punkt stały $u_1^\star = 0$. jest niestabilny, natomiast punkt stały $u_2^\star = 1$ jest stabilny. Metoda Eulera prowadzi do równania różnicowego

$$u_{k+1} = u_k + hf(u_k) = u_k + hu_k(1 - u_k), \qquad k = 0, 1, 2, \ldots$$

gdzie h jest długością kroku. Zakładamy, że $h = 0,005$ i $u_0 = 0,1$. Okazuje się, że u_k dąży do stabilnego stałego punktu $u_2^* = 1$, jako $k \to \infty$.

```python
import matplotlib.pyplot as plt

def f(u):
    return u*(1.0 - u)

h = 0.005
u = 0.1
ul = 0.0
t = 0.0
m_tab = []
n_tab = []
p_tab = []
q_tab = []
for i in range(2000):
    ul = u
    m = (int)(40.0*t+10.0+0.5)
    m_tab.append(m)
    n = (int)(200.0-50.0*ul+0.5)
    n_tab.append(n)
    u = ul + h*f(ul)
    p = (int) (40.0*(t+h)+10+0.5)
    p_tab.append(p)
    q = (int)(200.0-50.0*u+0.5)
    q_tab.append(q)
    t += h

plt.plot(m_tab, n_tab, label="1")
plt.plot(p_tab, q_tab, label="2")
plt.legend()
plt.show()
```



Technika serii kłamstw

Niech V będzie liniowym operatorem różniczkowym

$$V := V_1(\mathbf{z})\frac{\partial}{\partial z_1} + V_2(\mathbf{z})\frac{\partial}{\partial z_2} + \cdots + V_n(\mathbf{z})\frac{\partial}{\partial z_n}$$

gdzie V_i są funkcjami zmiennych zespolonych $z_1, z_2, ..., z_n$, z których zakłada się, że są holomorficzne w sąsiedztwie tego samego punktu. Jeśli f jest funkcją, która jest holomorficzna w sąsiedztwie tego samego punktu, możemy zastosować operator V do f

$$Vf := V_1(\mathbf{z})\frac{\partial f}{\partial z_1} + V_2(\mathbf{z})\frac{\partial f}{\partial z_2} + \cdots + V_n(\mathbf{z})\frac{\partial f}{\partial z_n}\,.$$

Ponieważ pochodne funkcji holomorficznej są holomorficzne, ponownie otrzymujemy funkcję, która jest holomorficzna w tym samym punkcie. Dotyczy to wszystkich operacji iterowanych. Oznacza to, że wszystkie funkcje

$$V^2 f := V(Vf), \quad \ldots, \quad V^n f := V(V^{n-1} f), \ldots$$

są holomorficzne w badanym punkcie i mogą być rozszerzane w regularnych zbieżnych seriach potęg. Serie:

$$\exp(tV)f(\mathbf{z}) \equiv \sum_{k=0}^{\infty} \frac{t^k}{k!} V^k f(\mathbf{z}) = f(\mathbf{z}) + tVf(\mathbf{z}) + \frac{t^2}{2!}V^2 f(\mathbf{z}) + \cdots$$

są nazywane seriami Lie.

Formalnie wyjaśniają to symbole spisanych serii, ponieważ każdy termin składa się z czynnika $t_n, n!$ oraz holomorficzna funkcja $V_n\ f$ zmiennych zespolonych $z_1, ..., z_n$. Uważamy, że t jest nową zmienną zespoloną, która jest niezależna od zmiennych $z_1, ..., z_n$. Szereg ten ma nie tylko znaczenie formalne, ale jest również zbieżny jako szereg potęgowy t, a zatem jest funkcją holomorficzną $n+1$ zmiennych zespolonych $z_1, ..., z_n, t$. O zbieżności szeregu Liego świadczy metoda majorantów Cauchy'ego.

Jeśli C jest skończoną, zamkniętą domeną przestrzeni z, w której operator różniczkowy V i funkcja f są holomorficzne, istnieje taka liczba dodatnia T, że szereg Lie jest zbieżny absolutnie i jednostajnie dla $|t| \leq T$ w całej dziedzinie G, gdzie reprezentuje holomorficzną funkcję $n+1$ zmiennych zespolonych $z_1, ..., z_n, t$.

Szereg Lie można rozróżnić na podstawie tych zmiennych wyraz po wyrazie i dowolnej liczbie czasów we wnętrzu G i $|t| \leq T$

$$\frac{\partial}{\partial t}\left(\sum_{k=0}^{\infty}\frac{t^k}{k!}V^k f(\mathbf{z})\right) = \sum_{k=0}^{\infty}\frac{t^k}{k!}V^{k+1} f(\mathbf{z})$$
$$\frac{\partial}{\partial z_j}\left(\sum_{k=0}^{\infty}\frac{t^k}{k!}V^k f(\mathbf{z})\right) = \sum_{k=0}^{\infty}\frac{t^k}{k!}\frac{\partial}{\partial z_j}(V^k f(\mathbf{z}))$$

Dla sum i iloczynów serii Lie mamy:

$$e^{tV}(c_1 f_1(\mathbf{z}) + c_2 f_2(\mathbf{z})) = c_1 e^{tV} f_1(\mathbf{z}) + c_2 e^{tV} f_2(\mathbf{z})$$
$$e^{tV}(f_1(\mathbf{z}) f_2(\mathbf{z})) = (e^{tV} f_1(\mathbf{z}))(e^{tV} f_2(\mathbf{z}))\,.$$

Jeśli $P(Z_1, Z_2)$ oznacza wielomian w Z_1, Z_2, to jest tak:

$$e^{tV} P(f_1(\mathbf{z}), f_2(\mathbf{z})) = P(e^{tV} f_1(\mathbf{z}), e^{tV} f_2(\mathbf{z}))\,.$$

Jeśli $F(z)$ oznacza dowolną funkcję, która jest holomorficzna w sąsiedztwie $\{z_1, ..., z_n\}$, której odpowiednie rozwinięcie szeregu potęgowego nadal zbiega się w punkcie $\{Z_1, ..., Z_n\}$, to prawdą jest, że:

$$F(\mathbf{Z}) = \sum_{k=0}^{\infty}\frac{t^k}{k!}V^k F(\mathbf{z})$$

lub napisane w inny sposób:

$$F(e^{tV}\mathbf{z}) = e^{tV} F(\mathbf{z})\,.$$

Inny ważny wynik uzyskuje się przez różniczkowanie funkcji Z_j względem t. Niech:

$$Z_j := \exp(tV) z_j, \qquad j = 1, 2, \ldots, n\,.$$

Różniczkowanie szeregu Lie termin po członie daje:

$$\frac{\partial Z_j}{\partial t} = e^{tV}(V z_j), \qquad j = 1, 2, \ldots, n\,.$$

Ponieważ $V_{zj} = V_j(z)$ to mamy

$$e^{tV} V_j(\mathbf{z}) = V_j(\mathbf{Z})\,.$$

Razem powoduje to

$$\frac{\partial Z_j}{\partial t} = V_j(\mathbf{Z}), \qquad j = 1, 2, \ldots, n\,.$$

Jeśli $\{z_1, ..., z_n\}$ jest punktem domeny holomorficznej V, funkcje $exp(tV)z_j$ w wystarczająco małym środowisku $t = 0$ spełniają układ równań różniczkowych

$$\frac{\partial Z_j}{\partial t} = V_j(\mathbf{Z})$$

przy warunkach początkowych $Z_j(t = 0) = z_j$. Zatem $exp(tV)z_j$ są rozwiązaniami układu równań różniczkowych, które są jednoznacznie określone przez te warunki początkowe.

Omawianą powyżej serię Lie można ograniczyć do domeny rzeczywistej. W naszych badaniach numerycznych rozważamy autonomiczny układ równań różniczkowych zwyczajnych:

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u})$$

gdzie $\mathbf{u} = (u_1, \ldots, u_n)^T$. Niech V_j będą funkcjami analitycznymi zdefiniowanymi na R_n. Rozważmy pole wektorowe:

$$V := \sum_{j=1}^{n} V_j(\mathbf{u}) \frac{\partial}{\partial u_j}.$$

Wtedy rozwiązanie problemu z wartością początkową systemu autonomicznego dla wystarczająco małego t można podać jako szereg Liego:

$$u_j(t) = \exp(tV) u_j|_{\mathbf{u}=\mathbf{u}(\mathbf{0})}$$

gdzie $j = 1, 2, ..., n$. Rozszerzanie funkcji wykładniczej daje:

$$u_j(t) = u_j(0) + tV(u_j)|_{\mathbf{u}=\mathbf{u}(0)} + \frac{t^2}{2} V(V(u_j))|_{\mathbf{u}=\mathbf{u}(0)} + \cdots$$

gdzie $j = 1, 2, ..., n$. W większości praktycznych przypadków do numerycznego całkowania równania różniczkowego można przyjąć tylko skończoną liczbę wyrażeń w tym rozwinięciu. Następnym przybliżeniem jest rozważenie:

$$\exp(t(V_1 + V_2)) = \prod_{j=1}^{n} \exp(c_j t V_1) \exp(d_j t V_2) + O(t^{n+1})$$

jeśli pole wektorowe V można zapisać jako $V = V_1 + V_2$ Aż do drugiego rzędu mamy:

$$\exp(t(V_1 + V_2)) = \exp(tV_1)\exp(tV_2) + O(t^2)$$

gdzie $k = 1$ z $c_1 = d_1 = 1$. Mamy dla trzeciego rzędu:

$$\exp(t(V_1 + V_2)) = \exp(\frac{1}{2}tV_1)\exp(tV_2)\exp(\frac{1}{2}tV_1) + O(t^3)\,.$$

Są to integratory symplektyczne.

Badanie struktury osobliwości w złożonej płaszczyźnie czasowej układu dynamicznego jest przedmiotem zainteresowania systemów o zachowaniu chaotycznym. Technikę serii Lie zaimplementowano w SymbolicC ++ z klasą Complex

```cpp
#include <iostream>
#include<cmath>
#include <complex>
using namespace std;

/* 10.3 Lie Series Method */

double magnitude(double a, double b) {
    return sqrt(a*a+b*b);
}

int main(void) {


    complex<double> r(40.0,0.0);
    complex<double> s(16.0,0.0);
    complex<double> b(4.0,0.0);
    complex<double> count(0.0,0.0);
    complex<double> eps(0.0,0.01);
    complex<double> half(0.5,0.0);
    complex<double> usl(0.8,0.0);
    complex<double> us2(0.8,0.0);
    complex<double> us3(0.8,0.0);
    complex<double> ul, u2, u3;

    while(magnitude(count.real(), count.imag()) < 2.0) {
        ul = usl;
        u2 = us2;
        u3 = us3;
        complex<double> V1 = s*(u2-ul);
        complex<double> V2 = -u2-ul*u3+r*ul;
        complex<double> V3 = ul*u2-b*u3;
        complex<double> Wl = s*s*(ul-u2) + s*(-u2-ul*u3+r*ul);
        complex<double> W2 = u2+ul*u3-r*ul - s*(u2-ul)*u3
                             - (ul*u2-b*u3)*ul + r*s*(u2-ul);
        complex<double> W3 = s*u2*(u2-ul) + (-u2-ul*u3+r*ul)*ul
                             -b*(ul*u2-b*u3);
        usl = ul + eps*V1 + half*eps*eps*Wl;
        us2 = u2 + eps*V2 + half*eps*eps*W2;
        us3 = u3 + eps*V3 + half*eps*eps*W3;
        count += eps;
    }
    cout << usl << endl;
    cout << us2 << endl;
    cout << us3 << endl;
    return 0;
}
```

Technika Runge-Kutta-Fehlberga

Metody Runge-Kutta zostały opracowane w celu uniknięcia obliczeń pochodnych wysokiego rzędu, które mogą obejmować metodę Taylora. W miejsce tych pochodnych używane są dodatkowe wartości danej funkcji $g(u,t)$ w sposób, który zasadniczo powiela dokładność wielomianu Taylora. Załóżmy, że problem z wartością początkową:

$$\frac{d\mathbf{u}}{dt} = \mathbf{g}(\mathbf{u}, t), \qquad \mathbf{u}(t=0) = \mathbf{u}_0$$

dla autonomicznego układu równań różniczkowych zwyczajnych pierwszego rzędu należy całkować, gdzie $u(t_0) = u_0$. Typowy krok całkowania aproksymuje u przy $t = t_0 + h$, gdzie h jest długością kroku. Formuły to:

$$\mathbf{u}(t) = \mathbf{u}_0 + h\sum_{k=0}^{5} c_k\mathbf{g}^{(k)}, \qquad t = t_0 + h$$

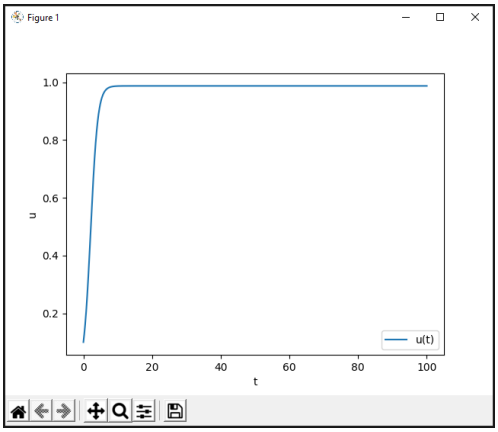
oraz

$$\mathbf{g}^{(0)} = \mathbf{g}(\mathbf{u}_0), \qquad \mathbf{g}^{(k)} = \mathbf{g}(\mathbf{u}_0 + h\sum_{j=0}^{k-1} b_{kj}\mathbf{g}^{(j)}).$$

Zatem $u(t)$ przybliża dokładne rozwiązanie. Współczynniki $c_k (k = 0, 1, .., 5)$ wynoszą:

$$c[0] = \frac{16}{135}, \quad c[1] = 0, \quad c[2] = \frac{6656}{12825},$$
$$c[3] = \frac{28561}{56430}, \quad c[4] = -\frac{9}{50}, \quad c[5] = \frac{2}{55}.$$

```python
import matplotlib.pyplot as plt

h = 0.005
u0 = 0.1
u = u0
t0 = 0.0
u_tab = []
t_tab = []
t = t0

C = [16/135, 0, 6656/12825, 28561/56430, -9/50, 2/55]

B = [ [0,0,0,0,0,0], [1/4, 0, 0, 0, 0], [3/32, 9/32, 0, 0, 0],
      [1932/2197, -7200/2197, 7296/2197, 0, 0],
      [439/216, -8, 3680/513, -845/4104, 0],
      [-8/27, 2, -3544/2565, 1859/4104, -11/40]]

def g(u):
  return u*(1.0 - u)

def gK(k):
  if k==0:
    return g(u0)
  else:
    sum = 0.0
    for j in range (k-1):
      sum += B[k][j] * g(j)
    return g(u0 + h*sum)

def sumCG():
  sum = 0.0
  for k in range(5):
    sum += C[k] * gK(k)
  return sum

for i in range(20000):
  u0 = u
  t_tab.append(t)
  u = u0 + h * sumCG()
  t += h
  u_tab.append(u)

plt.plot(t_tab, u_tab, label="u(t)")
plt.xlabel('t')
plt.ylabel('u')
plt.legend()
plt.show()
```



Powyżej przedstawiono kod obliczeń przeprowadzonych na podstawie wzorów, w celu sprawdzenia metody. Wykres przedstawia przybliżone rozwiązanie w czasie.

Rozwiązania widmowe

Kiedy dyskretyzujemy równania różniczkowe, może się zdarzyć, że wynikowe równanie różnicowe wykazuje chaotyczne zachowanie, nawet jeśli pierwotne równania różniczkowe zmierzają do punktu stałego. Rozważmy zwykłe

równanie różniczkowe

$$\frac{du}{dt} = u(1-u)$$

z warunkiem początkowym $u(0) = u_0 > 0$. Punkty stałe są podane przez

$$u^* = 0, \qquad u^* = 1.$$

Punkt stały $u^* = 1$ jest asymptotycznie stabilny. Dokładne rozwiązanie równania różniczkowego podaje wzór

$$u(t) = \frac{u_0 e^t}{1 - u_0 + u_0 e^t}.$$

Dokładne rozwiązanie, zaczynając od wartości początkowej $u_0 = 0,5$, rośnie monotonicznie i zmierza do 1, gdy t zmierza do nieskończoności. Dla $t = ln9999 = 9.21024$ mamy, że

$$u(t) = \frac{e^t}{1 + e^t} = \frac{9999}{10000} = 0.9999$$

tak, że $u(t)$ jest już dość blisko asymptotycznie stabilnego punktu stałego $u^* = 1$.

Aby zintegrować to równanie za pomocą schematu różnic skończonych, stosujemy schemat różnic centralnych:

$$\frac{du}{dt} \to \frac{u_{n+1} - u_{n-1}}{2h}.$$

Zatem równanie różniczkowe przyjmuje postać:

$$\frac{u_{n+1} - u_{n-1}}{2h} = u_n(1 - u_n)$$

z warunkami początkowymi:

$$u_0 = u_0, \qquad u_1 = u_0 + hu_0(1 - u_0).$$

Otrzymujemy:

$$u_{n+1} = u_{n-1} + 2hu_n(1 - u_n)\,.$$

Wprowadzając $v_n = u_n - 1$ otrzymujemy układ równań różnicowych pierwszego rzędu:

$$u_{n+1} = v_n + 2hu_n(1 - u_n), \qquad v_{n+1} = u_n\,.$$

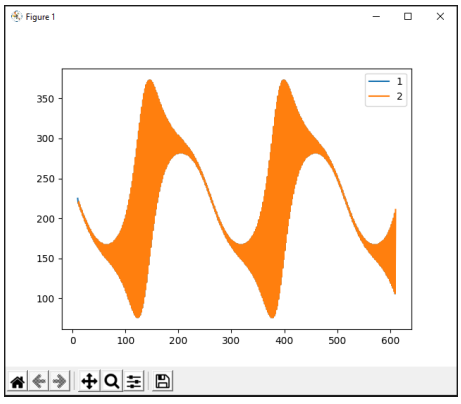
Obliczamy rozwiązanie numeryczne za pomocą równania różnicowego, zaczynając od wartości początkowej $u_0 = 0,5$ i używając długości siatki czasu $h = 0,05$. Równanie różnicowe nie jest stabilne w punkcie stałym $u^* = 1$ i znajdujemy zachowanie oscylacyjne. Takie zjawisko nazywa się rozwiązaniem widmowym lub rozwiązaniem fałszywym.

Dla $0 < t < 8,5$ rozwiązanie numeryczne daje dobre przybliżenie rzeczywistego rozwiązania. Rozwiązanie u_n rośnie monotonicznie i zbliża się do 1.000. Po $t = 8,6$ rozwiązanie numeryczne nie jest już monotonne. W

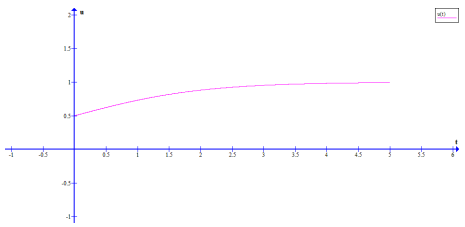
momencie $t = 9,7$ wartość u_n przyjmuje po raz pierwszy wartość nieco większą niż 1. Amplituda oscylacji rośnie i rośnie. Wzrost tej amplitudy jest geometryczny, a tempo wzrostu jest takie, że amplituda jest mnożona przez około $e = 2,71$, podczas gdy t wzrasta o jeden, aż do około$t = 17,0$, kiedy oscylacja traci symetrię względem $u^* = 1$. Powtarzanie takich cykli wydaje się być prawie okresowe. Rozwiązania widmowe pojawiają się również, nawet jeśli h jest dość małe. Jedną z przyczyn tego zjawiska jest to, że centralny schemat różnicowy jest schematem różnicowym drugiego rzędu i że niestabilność pojawia się przy $u^* = 1$ i $u^* = 0$.

Globalne zachowanie rozwiązań numerycznych obliczonych za pomocą równania różnicowego jest bardzo wrażliwe na warunek początkowy, długość siatki czasowej i dokładność zastosowanych obliczeń. Zależy również drastycznie od kalkulatorów. Zjawisko to jest spowodowane błędami zaokrągleń. Dokładne rozwiązanie u_n schematu centralnej różnicy przyjmuje wartości dość bliskie 1, ale rozwiązanie numeryczne obliczane przez komputer cyfrowy może przyjmować tylko wartości o skończonych cyfrach, tak że prawie wszystkie efektywne cyfry są tracone w obliczeniach, gdy u_n pozostaje w pobliżu 1. Jeżeli wybierzemy przedział czasowy na tyle mały, że dokładne rozwiązanie schematu różnicowego będzie zbliżone do prawdziwego rozwiązania danego przedziału czasu $0 < t < T$ w ramach zadanego błędu, to numeryczne rozwiązanie otrzymane przez komputer może nie przybliżać prawdziwego rozwiązanie w danym przedziale czasu z powodu błędów zaokrąglenia, jeśli zastosowana precyzja nie jest wystarczająco duża. Oscylacyjne zachowanie nie jest spowodowane skończoną precyzją lub błędami zaokrągleń.

```python
import matplotlib.pyplot as plt
import math

h = 0.05
u = 0.5
v = u + h * u * (1.0 - u)
t0 = 0.0
tb_tab = []
te_tab = []
m_tab = []
n_tab = []

for i in range(600):
  ul = u
  vl = v
  u = vl + 2.0 * h * ul * (1.0 - ul)
  v = ul
  tb = math.floor(20.0 * t0 + 10.0)
  tb_tab.append(tb)
  m = math.floor(300.0 - 150.0 * ul)
  m_tab.append(m)
  te = math.floor(20.0 * (t0 + h) + 10.0)
  te_tab.append(te)
  n = math.floor(300.0 - 150.0 * u)
  n_tab.append(n)
  t0 += h

plt.plot(tb_tab, m_tab, label="1")
plt.plot(te_tab, n_tab, label="2")
plt.legend()
plt.show()
```



```cpp
#include <iostream>
#include<fstream>
using namespace std;

int main(void) {
    ofstream resultFile("ghostLieResult.dat");
    double h = 0.001;
    double t = 0.0;
    double u = 0.5;
    double ul;
    while(t <= 5.0) {
        t += h;
        ul = u + h*u*(1.0-u) + h*h*u*(1.0-u)*(1.0-2.0*u)/2.0;
        cout << "t = " << t << " " << "u = " << ul << endl;
        resultFile << t << " " << ul << endl;
        u = ul;
    }
    resultFile.close();
    return 0;
}
```



Integracja symplektyczna

Równania ruchu Hamiltona w postaci standardowej lub kanonicznej są podane przez

$$\frac{dq_j}{dt} = \frac{\partial H}{\partial p_j}, \qquad \frac{dp_j}{dt} = -\frac{\partial H}{\partial q_j}, \qquad j = 1, 2, \ldots, N.$$

W hamiltonowskim opisie mechaniki ewolucja układu jest opisana za pomocą równań różniczkowych pierwszego rzędu $2N$. Zmienne $2n$ $q_1, ..., q_N, P_1, ..., P_N$ są często nazywane zmiennymi kanonicznymi. Definiują przestrzeń fazową 2_N - wymiarową. Rozwiązanie równań Hamiltona

$$q_j(t) = q_j(\mathbf{q}_0, \mathbf{p}_0, t), \qquad p_j(t) = p_j(\mathbf{q}_0, \mathbf{p}_0, t)$$

$$\mathbf{q}_0 = (q_1(0), \ldots, q_N(0)) \text{ and } \mathbf{p}_0 = (p_1(0), \ldots, p_N(0))$$

są zbiorem warunków początkowych, określają stan systemu w czasie t. W swojej ewolucji czasowej $(q(t), p(t))$ odwzorowują trajektorię eksplorującą różne regiony w przestrzeni fazowej. Ze względu na symetrię równań Hamiltona naturalne jest rozważanie zmiennych q_j i p_j na bardzo równych zasadach. Współrzędne N i n momentów można zatem uznać za pojedynczy zestaw współrzędnych $2N$, gdzie z_j

$$\mathbf{z} = (q_1, \ldots, q_N, p_1, \ldots, p_N).$$

Korzystając z tego zapisu, równania Hamiltona można zapisać w zwięzłej formie jako

$$\frac{d\mathbf{z}}{dt} = J_{2N} \cdot \nabla H(\mathbf{z})$$

gdzie:

$$\nabla := (\partial/\partial z_1, \ldots, \partial/\partial z_{2n}).$$

Macierz $2n \times 2n$, czyli J_{2N}, jest nazywana symplektyczną

$$J_{2N} := \begin{pmatrix} \mathbf{0} & I_N \\ -I_N & \mathbf{0} \end{pmatrix}$$

$$\sum_{j=1}^{N}\left(\frac{\partial \dot{q}_j}{\partial q_j}+\frac{\partial \dot{p}_j}{\partial p_j}\right)=0\,.$$

gdzie I_N to macierz jednostek $N \times N$. Spełniają warunek nieściśliwości

Oznacza to, że dywergencja systemu Hamiltona wynosi zero. W ten sposób element objętości w przestrzeni fazowej jest zachowany pod przepływem Hamiltona. Wynik ten jest znany jako twierdzenie Liouville'a i jest jedną z podstawowych właściwości układów dynamicznych Hamiltona.

Jeśli system dynamiczny ewoluuje pod wpływem przepływu Hamiltona, wiele ważnych wielkości pozostaje niezmiennych. Najbardziej fundamentalnym z nich jest byt geometryczny, różniczkowa forma dwustopniowa.

$$\omega_2 := \sum_{j=1}^{N} dp_j \wedge dq_j\,.$$

Oznacza to, że jeśli oznaczymy przepływ fazy Hamiltona przez ϕ_t gdzie ϕ_t odwzorowuje warunki początkowe do rozwiązania w czasie t, otrzymamy

$$(\Phi_t)^*\omega_2 = \omega_2\,.$$

Mamy więc definicję:
Transformacja $(q,p) \to (Q,P)$ nazywana jest symplektyczną, jeśli zachowuje 2-formę $_2$.

Każda transformacja zachowująca $_2$ również zachowuje postać równań Hamiltona. Odwrotna sytuacja nie jest prawdą. Przepływ mający tę właściwość jest nazywany symplektycznym. W przypadku jednego stopnia swobody $(N = 1)$ oznacza to po prostu zachowanie (zorientowanego) obszaru fazowego.

To zachowanie 2-formy jest podstawową właściwością systemów Hamiltona. W rzeczywistości przepływ Hamiltona można scharakteryzować wyłącznie za pomocą 2-formy. Jeśli domena D w R^{2N} jest po prostu podłączona (tj. Nie ma dziur) i

$$\frac{d\mathbf{p}}{dt} = \mathbf{f}(\mathbf{q},\mathbf{p}), \qquad \frac{d\mathbf{q}}{dt} = \mathbf{g}(\mathbf{q},\mathbf{p})$$

jest gładkim układem różniczkowym, którego przepływ zachowuje postać 2, to ten układ równań różniczkowych jest układem Hamiltona dla pewnej funkcji Hamiltona H.

Zachowanie 2-formy $_2$ jest jedną z całej hierarchii wielkości zachowanych przez przepływ hamiltonianu, który po raz pierwszy zbadał Poincare, który nazwał niezmiennikami całkowymi.

Twierdzenie Liouville'a. Przepływ Hamiltona zachowuje element objętości w przestrzeni fazowej

$$\int \prod_{j=1}^{N} dp_j \wedge dq_j = \int \prod_{j=1}^{N} dP_j \wedge dQ_j$$

w którym znak całki reprezentuje 2N-wymiarową integrację po zadanej objętości w przestrzeni fazowej.

Inną właściwością konserwatywnych systemów Hamiltona jest to, że H jest stałą ruchu, tj. $dH/dt = 0$. Jesteśmy szczególnie zainteresowani numerycznymi dyskretyzacjami systemów Hamiltona i intuicyjnie wydaje się, że dobrym pomysłem jest pozwolić, aby dyskretyzacja uchwyciła jak najwięcej jak to możliwe z oryginalnej struktury Hamiltona. To prowadzi do badania przemian zachowujących różniczkową dwie formy. Ponieważ nie można oczekiwać, że numeryczna dyskretyzacja systemu ciągłego będzie dokładna, nie możemy zachować wszystkich właściwości pierwotnego przepływu. Na przykład nie możemy zachować pierwotnej postaci funkcji Hamiltona i właściwości zachowania powierzchni pierwotnego przepływu - zachowanie obu tych wielkości sprowadza się do dokładnego rozwiązania układu. Odkąd zachowanie symplektyczności jest tak podstawową własnością systemów Hamiltona, wydaje się naturalne, że próba zachowania tej właściwości w systemie dyskretyzowanym wydaje się naturalna.

Kiedy ciągły przepływ jest dyskretyzowany, dyskretny przepływ w zasadzie staje się następstwem przekształceń z jednego etapu w inny. Dlatego zachowanie formy 2 można zachować, zapewniając, że transformacje w systemie dyskretnym są symplektyczne. Innym powodem, dla którego transformacje symplektyczne są przydatne, jest to, że często można uprościć całkowanie równań ruchu układu, przekształcając je do innego zestawu współrzędnych. W hamiltonowskim opisie mechaniki mamy dwa zbiory niezależnych zmiennych p i q, które są bardzo równorzędne.

Dlatego możemy przekształcić z jednego zestawu zmiennych w przestrzeni fazowej (q, p) do nowego zbioru (Q, P). Ta transformacja może być symbolicznie zapisana jako

$$P_i = P_i(q_1, \ldots, q_N, p_1, \ldots, p_N), \qquad Q_i = Q_i(q_1, \ldots, q_N, p_1, \ldots, p_N).$$

Istnieją różne sposoby konstruowania transformacji symplektycznych. Najłatwiejsza z nich to metoda generowania funkcji. Rozważmy system jednego stopnia swobody

$$\frac{dq}{dt} = \frac{\partial H(p,q)}{\partial p}, \qquad \frac{dp}{dt} = -\frac{\partial H(p,q)}{\partial q}.$$

Układ ten można zdyskretyzować metodą Eulera, otrzymując

$$Q = q + \tau \frac{\partial H(p,q)}{\partial p}, \qquad P = p - \tau \frac{\partial H(p,q)}{\partial q}.$$

System dyskretny definiuje mapę od (q, p) (przybliżone rozwiązanie w czasie t) do (Q, P) (przybliżone rozwiązanie w czasie $t + r$). Jakobian dla tej transformacji można zapisać jako

$$\frac{\partial(Q,P)}{\partial(q,p)} = 1 + O(\tau).$$

Zatem ta przemiana nie zachowuje obszaru. Dlatego metoda Eulera nie jest symplektyczna. Zmodyfikujmy nieco metodę

$$P = p - \tau \frac{\partial H(q,P)}{\partial q}, \qquad Q = q + \tau \frac{\partial H(q,P)}{\partial p}.$$

W tym przypadku jakobian spełnia poniższą równość:

$$\frac{\partial(Q,P)}{\partial(q,p)} = 1.$$

Zmodyfikowana metoda jest zatem symplektyczna.

Aby system wyższego wymiaru był symplektyczny, zachowanie objętości jest właściwością konieczną, ale niewystarczającą.

Załóżmy, że mamy do czynienia z rozłącznym planarnym $(n = 1)$ układem Hamiltona,

$$H(q,p) = \frac{1}{2}p^2 + V(q).$$

Wybierając funkcję generującą drugiego rodzaju,

$$F^2(q,P,t) = qP + tH(q,P) = qP + \frac{1}{2}tP^2 + tV(q)$$

$$S^t : (q,p) \to (Q,P)$$

Powyższe odwzorowanie, to odwzorowanie symplektyczne

Metody symplektyczne wyższego rzędu są skonstruowane w następujący sposób. Zakładamy, że pole wektorowe V można zapisać jako $V = V_1 + V_2$. Jeśli $[V_1, V_2] = 0$, to dla t mamy dostatecznie małe

$$\exp(tV) = \exp(tV_1 + tV_2) = \exp(tV_1)\exp(tV_2)$$

gdzie [,] oznacza komutator. Ogólnie mamy $[V1, V2] \neq 0$

Formuła jest następująca. Dla dowolnych nieprzemiennych operatorów X i Y, iloczyn dwóch funkcji wykładniczych, $exp(X)$ $exp(Y)$, można wyrazić w postaci pojedynczej funkcji wykładniczej jako

$$\exp(X)\exp(Y) = \exp(Z)$$

gdzie

$$Z = X + Y + \frac{1}{2}[X,Y] + \frac{1}{12}([X,X,Y] + [Y,Y,X]) + \dots .$$

Tutaj [,] oznacza komutator i komutatory wyższego rzędu, takie jak uwzględniono powyżej

$$[X,X,Y] := [X,[X,Y]] .$$

Cechą wzoru Bakera-Campbella-Hausdorffa jest to, że pojawiają się tylko komutatory X i Y z wyjątkiem liniowych wyrażeń w szeregu.

Inną techniką, którą można zastosować do integracji, jest formuła Trottera. Niech A będzie generatorem kontrgrupy Co - półgrupy $exp(tA)_{t\geqslant 0}$ w przestrzeni Banacha E i niech $B \in L(E)$ będzie liniowym operatorem dyssypatywnym, gdzie $L(E)$ oznacza przestrzeń wektorową wszystkich liniowych mapy ograniczone $E \to E$. Następnie $A + B$ generuje C_0 - półgrupę, która jest określona wzorem Trottera

$$\exp(t(A+B)) = \lim_{n\to\infty}\left(\exp\left(\frac{t}{n}A\right)\exp\left(\frac{t}{n}B\right)\right)^n$$

gdzie limit jest brany pod uwagę w silnej topologii operatora.

Metoda Verleta

Algorytm Verleta jest metodą całkowania równań różniczkowych zwyczajnych drugiego rzędu:

$$\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}(\mathbf{x}(t), t) .$$

Jest on również używany w symulacjach dynamiki molekularnej. Ma stały przedział czasowy dyskretyzacji h i wymaga tylko jednej oceny siły F na krok. Algorytm jest wyprowadzany przez dodanie rozwinięć Taylora dla współrzędnych x przy $t = h$ wokół 0:

$$\mathbf{x}(h) = \mathbf{x}(0) + h\frac{d\mathbf{x}(0)}{dt} + \frac{h^2}{2}\mathbf{F}(\mathbf{x}(0),0) + \frac{h^3}{6}\frac{d^3\mathbf{x}(0)}{dt^3} + O(h^4)$$
$$\mathbf{x}(-h) = \mathbf{x}(0) - h\frac{d\mathbf{x}(0)}{dt} + \frac{h^2}{2}\mathbf{F}(\mathbf{x}(0),0) - \frac{h^3}{6}\frac{d^3\mathbf{x}(0)}{dt^3} + O(h^4)$$

co prowadzi do:

$$\mathbf{x}(h) = 2\mathbf{x}(0) - \mathbf{x}(-h) + h^2\mathbf{F}(\mathbf{x}(0),0) + O(h^4) .$$

Znając wartości x w czasie 0 i $-h$, algorytm ten przewiduje wartość $x(h)$. Zatem potrzebujemy dwóch ostatnich wartości x, aby otrzymać następną. Jeśli mamy do dyspozycji tylko pozycję początkową $x(0)$ i prędkość początkową $v(0)$, przybliżymy $x(h)$ przez:

$$\mathbf{x}(h) \approx \mathbf{x}(0) + h\mathbf{v}(0) + \frac{h^2}{2}\mathbf{F}(\mathbf{x}(0), 0)$$

czyli ustawiliśmy

$$\mathbf{v}(0) = \frac{\mathbf{x}(0) - \mathbf{x}(-h)}{h}.$$

Rozważmy jednowymiarowy oscylator harmoniczny

$$\frac{d^2x}{dt^2} + \Omega^2 x = 0$$

gdzie Ω jest stałą częstotliwością.

Analityczne rozwiązanie tego równania różnicowego można zapisać w postaci

$$x(t) = \exp(i\omega t)$$

z spełniającym warunek

$$2 - 2\cos(\omega h) = h^2\Omega^2.$$

Jeśli $h^2\Omega^2 \equiv (h\Omega)^2 > 4$, częstotliwość staje się urojona, a rozwiązanie analityczne staje się niestabilne. Dlatego warto wprowadzić bezwymiarowy czas T przez

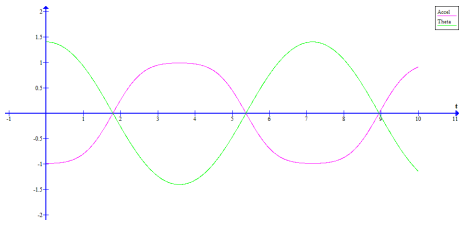
$$\tilde{x}(\tau(t)) = x(t), \qquad \tau(t) = \Omega t.$$

Następnie liniowe równanie różniczkowe dla oscylatora harmonicznego przyjmuje postać

$$\frac{d^2\tilde{x}}{d\tau^2} + \tilde{x} = 0.$$

W poniższym programie podano implementację metody Verleta dla jednowymiarowego wahadła $d^2\tilde{\theta}/dt^2 = -(g/L)\sin(\theta)$.

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

int main(void) {
    ofstream resultFileTheta("verletResultTheta.dat");
    ofstream resultFileAccel("verletResultAccel.dat");
    const double pi = 3.141592654;
    double theta =1.4 ; // initial angle (in radians)
    double omega =0.0 ; // initial velocity
    double g_over_L = 1.0; // the constant g/L
    double time =0.0 ; // initial time
    double time_old; // time of previous reversal
    double tau = 0.005; // time step size
    // take one backward step to start Verlet
    double accel = -g_over_L*sin(theta); // gravitational acceleration
    double theta_old = theta - omega*tau + 0.5*tau*tau*accel;
    int nStep = 2000; // number of steps
    for(int i=1; i<=nStep; i++) {
        time += tau;
        accel = - g_over_L*sin(theta);
        double theta_new = 2.0*theta - theta_old + tau*tau*accel;
        cout << "time = " << time << " "
             << "theta = " << theta_new << " "
             << "accel = " << accel;
        cout << endl;
        resultFileTheta << time << " " << theta << endl;
        resultFileAccel << time << " " << accel << endl;
        theta_old = theta;
        theta = theta_new;
    }
    resultFileTheta.close();
    resultFileAccel.close();
    return 0;
}
```



Metoda Stormera

Rozważamy układ równań różniczkowych zwyczajnych drugiego rzędu:

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}(\mathbf{x})$$

gdzie x jest zbiorczym wektorem położenia, m jest ukośną macierzą mas, a F jest zbiorczym wektorem sił. Dyskretyzację znaną jako metoda Stormera drugiego rzędu podaje:

$$\frac{1}{\Delta t^2}m(\mathbf{X}^{n+1} - 2\mathbf{X}^n + \mathbf{X}^{n-1}) = \mathbf{F}(\mathbf{X}^n)$$

gdzie Δt jest krokiem w czasie, a X^n oznacza przybliżenie różnicy do x w czasie $n\Delta t$. Ta metoda może być używana jako integrator dynamiki molekularnej wraz ze wzorem:

$$\mathbf{V}^n = \frac{1}{2\Delta t}(\mathbf{X}^{n+1} - \mathbf{X}^{n-1})$$

do obliczenia prędkości $v = dx/dt$. Ta kombinacja jest równoważna metodzie leapfrog, zdefiniowanej jako:

$$\begin{aligned}\mathbf{V}^{n+1/2} &= \mathbf{V}^{n-1/2} + \Delta t m^{-1}\mathbf{F}(\mathbf{X}^n)\\ \mathbf{X}^{n+1} &= \mathbf{X}^n + \Delta t\mathbf{V}^{n+1/2}.\end{aligned}$$

Ukryty schemat dyskretyzacji z inną prawą stroną:

$$\frac{1}{\Delta t^2}m(\mathbf{X}^{n+1} - 2\mathbf{X}^n + \mathbf{X}^{n-1}) = \frac{1}{12}\mathbf{F}(\mathbf{X}^{n-1}) + \frac{5}{6}\mathbf{F}(\mathbf{X}^n) + \frac{1}{12}\mathbf{F}(\mathbf{X}^{n+1})$$

Inną metodą, którą można zastosować do całkowania równania różniczkowego, jest:

$$\frac{1}{\Delta t^2}m(\mathbf{X}^{n+1} - 2\mathbf{X}^n + \mathbf{X}^{n-1}) = \mathbf{F}\left(\frac{1}{2}(\mathbf{X}^{n-1} + \mathbf{X}^{n+1})\right).$$

Niewidzialny chaos

Dyskretyzacja nieliniowych równań różniczkowych może prowadzić do odwzorowań, które wykazują chaotyczne zachowanie. Rozważamy system anharmoniczny:

$$\frac{d^2u}{dt^2} + u^3 = 0.$$

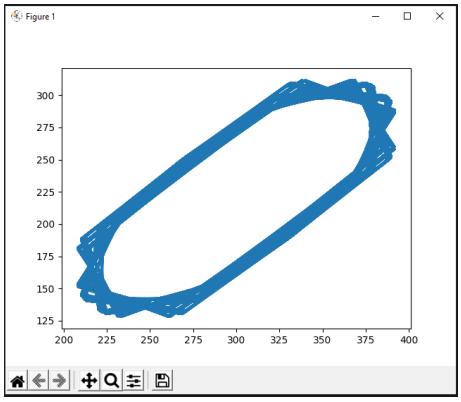
Korzystając ze schematu różnic skończonych, to równanie różniczkowe można zapisać jako:

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{(\Delta t)^2} = -u_n^3$$

By uzyskać przybliżone rozwiązania równania różniczkowego. Dla ustalonego kroku czasowego $\Delta t, \Delta t > 0$, ten schemat jest równoważny ze schematem: $U_{n+1} - 2U_n + U_{n-1} = -U_n^3$
przez zamiennik $u_n = U_n/\Delta t$. Definiujemy $V_n := U_{n-1}$. W rezultacie otrzymujemy odwzorowanie: $U_{n+1} = -V_n + 2U_n - U_n^3, V_{n+1} = U_n$

Te odwzorowanie jest odwracalne i zachowuje obszar. Numeryczne rozwiązania schematu różnicowego dla małego Δt odpowiadają orbitom w pobliżu początku (0,0) mapy poprzez transformację $u_n = U_n/\Delta t$. Eksperymenty numeryczne pokazują, że wokół początku mapy istnieją niezmienne okręgi (0,0).

```python
import matplotlib.pyplot as plt
import math

u = 0.9
v = 0.75
T = 3000
mx_tab = []
ny_tab = []
for t in range(T):
  ul = u
  vl = v
  u = -vl + 2.0 * ul - ul * ul * ul
  v = ul
  mx = math.floor(100 * u + 300 + 0.5)
  mx_tab.append(mx)
  ny = math.floor(100 * v + 220 + 0.5)
  ny_tab.append(ny)

plt.plot(mx_tab, ny_tab)
plt.show()
```



Pierwsze całki i całkowanie numeryczne

Rozważmy autonomiczny układ równań różniczkowych zwyczajnych pierwszego rzędu:

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u})$$

gdzie pole wektorowe $f : R_n \to R_n$ jest analityczne. Niektóre z tych systemów przyjmują pierwszą całkę, tj. Istnieje funkcja skalarna I taka, że

$$\frac{dI(\mathbf{u})}{dt} = 0\,.$$

A zatem

$$\sum_{j=1}^{n} \frac{\partial I}{\partial u_j}\frac{du_j}{dt} \equiv \sum_{j=1}^{n} \frac{\partial I}{\partial u_j} f_j(\mathbf{u}) = 0\,.$$

Zachowanie pierwszych całek w całkowaniu numerycznym jest ważne ze względu na ich fizyczne znaczenie, np. w mechanice i astronomii, ale także dlatego, że mogą zapewnić długoterminowe efekty stabilizujące. Dlatego chcemy znaleźć dyskretne przybliżenie układu równań różniczkowych.

$$\frac{\mathbf{u}' - \mathbf{u}}{\tau} = \mathbf{g}(\mathbf{u}, \mathbf{u}', \tau)$$

$(\mathbf{u} \equiv \mathbf{u}(n\tau),\ \mathbf{u}' \equiv \mathbf{u}((n+1)\tau))$ takie, że pierwsza całka jest zachowana dokładnie, tj. $I(u') = I(u)$. Na przykład niejawna reguła punktu środkowego

$$\frac{\mathbf{u}' - \mathbf{u}}{\tau} = \mathbf{f}\left(\frac{\mathbf{u}' + \mathbf{u}}{2}\right)$$

zachowuje całki kwadratowe.

McLaren i Quispel przedstawili bardziej ogólny przypadek. Układ równań różniczkowych zwyczajnych można zapisać w postaci:

$$\frac{d\mathbf{u}}{dt} = S \cdot \nabla I(\mathbf{u})$$

gdzie S jest antysymetryczną macierzą $n \times n$ nad R, tj. $S^T = -S$, a ∇ oznacza gradient. Dyskretną wersją z zachowaniem integralności jest:

$$\frac{\mathbf{u}' - \mathbf{u}}{\tau} = \tilde{S}(\mathbf{u}, \mathbf{u}', \tau)\overline{\nabla} I(\mathbf{u}, \mathbf{u}')$$

gdzie u, u'oznacza u_n odpowiednio u_{n+1}, a $\overline{S}$ jest macierzą antysymetryczną spełniającą (dla spójności):

$$\tilde{S}(\mathbf{u}, \mathbf{u}', \tau) = S(\mathbf{u}) + O(\tau)\,.$$

Rząd dokładności integratora zachowującego całkę w oparciu o dyskretyzację jest określony przez $\overline{S}$ i przez wybór dyskretnego gradientu $\overline{I}(u, u')$, tj. Przez $\overline{S}$ i macierz B, tensor M i wyższego rzędu części $\overline{I}$.