Nonlinear Driven Systems

P. S.[*], M. B.[†], K. M.[‡], K. G.[§], D. Ł.[¶]

26 czerwca 2021

Spis treści

[*]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie
[†]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie
[‡]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie
[§]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie
[¶]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie

1 Wprowadzenie

Rozważamy dynamiczny układ nieliniowy opisany układem pierwszego rzędu Równania różniczkowe zwyczajne: $\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}(t, \boldsymbol{u}, r)$ W tym przypadku: $t \in \boldsymbol{R}, \boldsymbol{u} \in \boldsymbol{R^n}, r \in \boldsymbol{R}$,gdzie następująco widać czas, stan n-wymiarowy oraz parametr systemowy. Przyjmujemy więc, że:

$f : \boldsymbol{R} \times \boldsymbol{R^n} \times \boldsymbol{R} \to \boldsymbol{R^n}$, $(t, \boldsymbol{u}, r) \to \boldsymbol{f}(t, \boldsymbol{u}, r)$
Jest to odwzorowanie klasy C^∞ oraz okresowe w t w okresie 2π, skąd wynika:

$\boldsymbol{f}(t + 2\pi, \boldsymbol{u}, r) = (t, \boldsymbol{u}, r)$

Za pomocą hipotezy okresowej podanej powyżej możemy zdefiniować C^∞ dyfeomorfizmem T_r z przestrzeni stanów $\boldsymbol{R^n}$:

$T_r : \boldsymbol{R^n} \to \boldsymbol{R^n}, \boldsymbol{v} \to T_r(\boldsymbol{v}) = \phi(2\pi, \boldsymbol{v}, r)$

Odwzorowanie T_r jest zwykle nazywane odwzorowaiem Poincare i używane jest do badań właściwości jakościowych układu dynamicznego podanego powyżej.

Jeżeli rozwiązanie $\boldsymbol{u}(t) = \phi(t, P_0, r)$ jest okresowe w okresie 2π, wtedy też punkt P_0 jest stałym punktem od T_r.

Wprowadzamy następnie hyperboliczny punkt stały oraz określamy jego klasyfikacje. Niech $P \in \boldsymbol{R^n}$ będzie punktem stałym T_r wtedy okolica P, odwzorowania T_r może być aproksymowana pzez jego pochodną $DT_r(P)$. Jest to możliwe jeżeli punktem stałym jest punkt hiuperboliczny. P nazywamy hiperbolicznym puntkem stałym T_r jeżeli $DT_r(P)$ jest hyperboliczne.

Przykłady stałych punktów hiperbolicznych:

(i) PD-typ jeżeli $\dim E^u$ jest parzysty oraz $\det L^u > 0$
(ii) ND-typ jeżeli $\dim E^u$ jest nieparzysty oraz $\det L^u > 0$
(iii) PI-typ jeżeli $\dim E^u$ jest parzysty oraz $\det L^u < 0$
(iv) NI-typ jeżeli $\dim E^u$ jest nieparzysty oraz $\det L^u < 0$.

2 Sterowane systemy anharmoniczne

2.1 Portret fazowy

Oceniamy portret fazowy dla parametrów. Dla tych wartości parametrów przypuszcza się, że system wykazuje zachowanie chaotycznie. Stosujemy technikę Runge-Kutty do całkowania równania różniczkowego.

Rozważamy system anharmoniczny
$\frac{d^2u}{dt^2} + a\frac{du}{dt} + bu + cu^3 = k_1 + k_2\cos(\Omega t)$
Niech $u_1 := u$, $u_2 := \frac{du}{dt}$, $u_3(t) = \Omega t$
Wtedy można zapisać równanie jako układ pierwszego rzędu
$\frac{du_1}{dt} = u_2, \quad \frac{du_2}{dt} = -au_2 - bu_1 - cu_1^3 + k_1 + k_2\cos(u_3), \quad \frac{du_3}{dt} = \Omega$
W programie posługujemy się wartościami:
a = 1, b = -10, c = 100, $k_1 = 0$, $k_2 = 1.2$ $\Omega = 3.5$ i znajdujemy portret fazowy $(u_1(t), u_2(t))$ za pomocą szeregu Liego

2.2 Przekrój Poincare

Sterowany system anharmoniczny jest nadawany przez system
$\frac{du_1}{dt} = u_2, \quad \frac{du_2}{dt} = -au_2 - bu_1 - cu_1^3 + k_1 + k_2\cos(\Omega t)$
a, b, c, k_1, k_2 *i* Ω są parametrami bifurkacji. Ten system jest niezmienny pod transformacją
$t \to t + 2\pi n/\Omega$,

W ten sposób możemy wprowadzić dyfeomorfizm na płaszczyźnie (u_1, u_2), co może służyć do liczbowego rozstrzygania czy istnieje chaotyczne zachowanie.

Zakładamy, że $(t, u_1(0), u_2(0)) \to u_1(t, u_1(0), u_2(0))$
$(t, u_1(0), u_2(0)) \to u_2(t, u_1(0), u_2(0))$ jest rozwiązaniem napędzanego układu anharmonicznego, zaczynając od punktu $(u_1(0), u_2(0)) = p_0$
kiedy t=0 $(u_1(2\pi/\Omega), u_2(2\pi/\Omega)) = p_1$
W ten sposób zdefiniowaliśmy dyfeomorfizm $T_r : R^2 \to R^2, \quad p_0 \to p_1$ gdzie r jest parametrem bifurkacji.

2.3 Wykładnik Liapunowa

Rozważmy napędzane Równanie van der Pola. Aby znaleźć najwyższy jednowymiarowy wykładnik Liapunova obliczamy system wariacyjny.

Sterowany system anharmoniczny jest podany przez
$\frac{du_1}{dt} = u_2, \quad \frac{du_2}{dt} = -au_2 - bu_1 - cu_1^3 + k_1 + k_2\cos(\Omega t)$ a>o Równanie można zapisać jako układ autonomiczny $\frac{du_1}{dt} = u_2$ $\frac{du_2}{dt} = -au_2 - bu_1 - cu_1^3 + k_1 + k_2\cos(u_3)$ $\frac{du_3}{dt} = \Omega$

$u_3(t = o) = 0$. Równanie wariacyjne
$\frac{dv}{dt} = (D_u f)v$ jest dany przez $\frac{du_1}{dt} = u_2$ $\frac{du_2}{dt} = -au_2 - bu_1 - 3cu_1^3 v_1 - k\sin(u_3)u_3$ $\frac{du_3}{dt} = 0$

Bez utraty ogólności możemy ustawić $u_3(t) = 0$. Wtedy system wariacyjny upraszcza się do $\frac{du_1}{dt} = u_2$ $\frac{du_2}{dt} = -au_2 - bu_1 - 3cu_1^3 v_1$

Jednowymiarowy wykładnik Liapunowa wynika z $\lambda := \lim_{T\to\infty} \frac{1}{T} \ln ||v(T)||$ $||v(T)|| = |v_1(T)| + |v_2(T)|$ W programie obliczamy jednowymiarowy wykładnik λ Liapunowa dla parametrów

k = 1.2, a = 1.0, b = -10.0, c = 100.0, Ω = 3.5.

Stosujemy technikę szeregu Lie. Znajdujemy dla jednowymiarowego wykładnika Liapunowa $\lambda \approx 0,34$.

2.4 Funkcja autokorelacji

Sterowany system anharmoniczny jest podany przez $\frac{d^2u}{dt^2} + a\frac{du}{dt} + bu + cu^3 = k\cos(\Omega t)$

Niech $u_1 := u,\ u_2 := \frac{du}{dt}$. Wtedy system można zapisać jako
$\frac{du_1}{dt} = u_2, \quad \frac{du_2}{dt} = -au_2 - bu_1 - cu_1^3 + k\cos(\Omega t)$ Średni czas u(t) jest określana jako $\langle u\rangle := \lim_{T\to\infty} \frac{1}{T}\int_0^T u(t)dt$. Funkcja autokorelacji u(t) jest zdefiniowana przez $C_{uu}(\tau) := \frac{\lim_{T\to\infty} \frac{1}{T}\int_0^T u(t) - \langle u(t)\rangle)(u(t+r) - \langle u(t+r)\rangle)dt}{\lim_{T\to\infty} \frac{1}{T}\int_0^T u(t) - \langle u(t)\rangle)^2 dt}$ Funkcja autokorelacji mierzy korelację między kolejnymi sygnałami. Pozostaje stała lub oscyluje dla normalnego ruchu i szybko zanika. Jeśli sygnały stają się nieskorelowane w chaotycznym reżimie, pierwszy generujemy szereg czasowy z całkowania równania różniczkowego. Szeregi czasowe służy do obliczania funkcji autokorelacji.

2.5 Gęstość widmowa mocy

Funkcja autokorelacji rzeczywistego sygnału skalarnego u(t) określona jest wzorem: $R_{uu}(\tau) = E[u(t+r)u(t)]\ E[u(t+r)u(t)] := \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\rho_{u(t+r),u(t)}(x,y)xydxdy$ $\rho_{u(t+r),u(t)}(x,y)xydxdy$ reprezentuje łączną funkcję gęstości prawdopodobieństwa $u(t+r)$ i u(t). Jeżeli u(t) jest ergodyczne, to funkcja autokorelacji u(t) może być równa $R_{uu}(\tau) = \lim_{T\to\infty}\frac{1}{T}\int_{-T}^{T}u(t+\tau)u(t)dt$.

Gęstość widmowa mocy $S_{uu}(\omega)$ jest równa:
$S_{uu}(\omega) := \int_{-\infty}^{\infty}R_{uu}(\tau)e^{-i\omega\tau}d\tau$
Co prowadzi do:
$S_{uu}(\omega) := \lim_{T\to\infty}E(\frac{1}{2T}|\int_{-T}^{T}u(t)e^{-i\omega t}dt|^2)$

Gęstość widmowa mocy może wskazywać, czy dynamiczny system jest okresowy lub quasi-periodyczny. Widmowa gęstość mocy układu okresowego o częstotliwości ω ma funkcje delta przy ω i jest to harmoniczne.A układ quasiperiodyczny o częstotliwościach podstawowych $\omega_1, ..., \omega_k$)ma na nich funkcje delta pozycji, a także we wszystkich kombinacjach liniowych ze współczynnikami całkowitymi.

3 Wahadło napędzane parametrycznie i zewnętrznie

Wahadło napędzane parametrycznie i zewnętrznie ma postać:

$$\frac{d^2u}{dt^2} + a\frac{du}{dt} + (1 + k_2\cos(\Omega_2 t))\sin u = k_1\cos(\Omega_1 t).$$

Gdzie za dane przyjmujemy wartość:

$$u_1(t) := u(t), \qquad u_2(t) := \frac{du_1}{dt}, \qquad u_3(t) := \Omega_1 t, \qquad u_4(t) := \Omega_2 t$$

Zatem równanie różniczkowe drugiego rzędu można zapisać jako autonomiczny układ równań różniczkowych zwyczajnych pierwszego rzędu

$$\frac{du_1}{dt} = u_2$$
$$\frac{du_2}{dt} = -au_2 - (1 + k_2\cos u_4)\sin u_1 + k_1\cos u_3$$
$$\frac{du_3}{dt} = \Omega_1$$
$$\frac{du_4}{dt} = \Omega_2$$

gdzie: $u_3(t = 0) = 0$

$u_4(t = 0) = 0.$

4 Kod w Pythonie dla danych przypadków

4.1 Wahadło o napędzie parametrycznym

```
import matplotlib.pyplot as plt
import math

ax = plt.subplot()

def paint():
    a = 0.15
    k = 0.94
    omega = 1.56
    tau = 0.005
    count = 0.0
    u11 = 0.1; u22 = 0.2; u33 = 0.0
    m = []
    n = []
    while(count < 1200.0):
        u1 = u11; u2 = u22; u3 = u33
        v2 = -a*u2 - (1.0 + k*math.cos(u3))*math.sin(u1)
        u11 = u1 + tau*u2 + tau*tau*v2/2.0
        u22 = u2 + tau*v2 + tau*tau*(-u2*(1.0-k*math.cos(u3))*math.cos(u1)
        - a*v2 + omega*k*math.sin(u3)*math.sin(u1))/2.0
        u33 = omega*tau + u3
        if(count> 1000.0):
            if(u11 > math.pi): u11 = u11 - 2.0*math.pi
            if(u11 < math.pi): u11 = u11 + 2.0*math.pi
            m.append((int) (80.0*u11 + 300.0))
            n.append((int) (60.0*u22 + 200.0))
        count += tau
    plt.plot(m,n,m,n, color='r')

#plt.ioff()

paint()

plt.show()
```

4.2 Przekrój Ponicare

```
import matplotlib.pyplot as plt
import math

ax = plt.subplot()

def paint():
    a = 0.15
    k = 0.94
    omega = 1.56
    tau = 0.005
    count = 0.0
    u11 = 0.1; u22 = 0.2; u33 = 0.0
    z = 0
    m = []
    n = []
    while(count < 10000.0):
        u1 = u11; u2 = u22; u3 = u33
        v2 = -a*u2 - (1.0+k*math.cos(u3))*math.sin(u1)
        u11 = u1 + tau*u2 + tau*tau*v2/2.0
        u22 = u2 + tau*v2 + tau*tau*(-u2*(1.0-k*math.cos(u3))*math.cos(u1)
        - a*v2 + omega*k*math.sin(u3)*math.sin(u1))/2.0
        u33 = omega*tau + u3
        if(count> 1000.0):
            if(u11 > math.pi): u11 = u11 - 2.0*math.pi
            if(u11 < math.pi): u11 = u11 + 2.0*math.pi
            if(abs(count - 2.0*math.pi*z/omega) < 0.02):
                m.append((int) (80.0*u11 + 300.0))
                n.append((int) (60.0*u22 + 200.0))
        count += tau
        if(count > 2.0*math.pi*z/omega): z=z+1
    plt.plot(m,n,m,n, color='r')

#plt.ioff()

paint()

plt.show()
```

4.3 Równanie Van der Pol

```
def fsystem(h, t, u, hf):
    a = 5.0
    k = 5.0
    omega = 2.466
    hf[0] = h*u[1]
    hf[1] = h*(a*(1.0-u[0]*u[0])*u[1]-u[0]+k*math.cos(omega*t))
```

```
def map(u, steps, h, t, N):
    a = [ 0.0, 1.0/4.0, 3.0/8.0, 12.0/13.0, 1.0, 1.0/2.0 ]
    c = [ 16.0/135.0, 0.0, 6656.0/12822.0, 28561.0/56430.0, -9.0/50.0, 2.0/55.0 ]
    n = 5; m = 6
    b = [[0 for x in range(n)] for x in range(m)]
    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] = 3680/513.0; b[4][3] = -845.0/4104.0; b[4][4] = 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] = -11.0/40.0
    f = [[0 for x in range(N)] for x in range(m)]
    uk = []
    for i in range(steps):
        fsystem(h, t, u, f[0])
        for k in range(5):
            tk = t + a[k]*h
            for l in range(N):
                uk.append(u[l])
                for j in range(k-1):
                    uk[l] += b[k][j] * f[j][l]
            fsystem(h, tk, uk, f[k])
        for l in range(N):
            for k in range(6):
                u[l] += c[k]*f[k][l]
```

```
def paint():
    steps = 1
    N = 2
    h = 0.005
    t = 0.0
    u = [ 0.8, 0.6 ]
    mx = []
    my = []
    nx = []
    ny = []
    for i in range(1000):
        t += h
        map(u, steps, h, t, N)
    t = 0.0
    for i in range(100000):
        mx.append((int) (150*u[0] + 250.0))
        my.append((int) (150*u[1] + 200.0))
        t += h
        map(u, steps, h, t, N)
        nx.append((int) (100.0*u[0] + 350.0))
        ny.append((int) (15.0*u[1] + 220.0))
    plt.plot(nx, ny, nx, ny, color='r')
```

4.4 Wahadło napędzane zewnętrznie i parametrycznie

```
def fsystem(h, t, u, hf):
    a = 0.1
    k1 = 0.3
    k2 = 0.3
    omega1 = 0.618034
    omega2 = 1.618034
    hf[0] = h*u[1]
    hf[1] = h*(-a*u[1] - (1.0 + k2*math.cos(omega2*t))*math.sin(u[0])
    + k1*math.cos(omega1*t))
```

```
def map(u, steps, h, t, N):
    a = [ 0.0, 1.0/4.0, 3.0/8.0, 12.0/13.0, 1.0, 1.0/2.0 ]
    c = [ 16.0/135.0, 0.0, 6656.0/12822.0, 28561.0/56430.0,
     -9.0/50.0, 2.0/55.0 ]
    n = 5; m = 6; uk = []
    b = [[0 for x in range(n)] for x in range(m)]
    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] = 3680/513.0; b[4][3] = -845.0/4104.0; b[4][4] = 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] = -11.0/40.0
    f = [[0 for x in range(N)] for x in range(m)]
    for i in range(steps):
        fsystem(h, t, u, f[0])
        for k in range(5):
            tk = t + a[k]*h
            for l in range(N):
                uk.append(u[l])
                for j in range(k-1):
                    uk[l] += b[k][j] * f[j][l]
            fsystem(h, tk, uk, f[k])
        for l in range(N):
            for k in range(6):
                u[l] += c[k]*f[k][l]
```

```
def paint():
    steps = 1
    N = 2
    h = 0.005
    t = 0.0
    u = [ 1.0, 0.0 ]
    mx = []
    my = []
    nx = []
    ny = []
    for i in range(2000):
        t += h
        map(u, steps, h, t, N)
    t = 0.0
    for i in range(50000):
        mx.append((int) (150*u[0] + 250.0))
        my.append((int) (150*u[1] + 200.0))
        t += h
        map(u, steps, h, t, N)
        nx.append((int) (150.0*u[0] + 250.0))
        ny.append((int) (150.0*u[1] + 200.0))
    plt.plot(mx, my, nx, ny, color='r')
```

5 Bibliografia

Literatura

[1] Willi-Hans Steeb, in collaboration with Yorick Hardy and Ruedi Stoop