ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní – Ústav mechaniky, biomechaniky a mechatroniky

DIPLOMOVÁ PRÁCE

Adaptivní řízení nelineárního systému s referenčním modelem

Adaptive control of a nonlinear system with a reference model

Jaroslav Průcha

2020

Prohlašuji, že jsem tuto práci vypracoval samostatně s použitím literárních zdrojů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů.

Datum: ....

Podpis

Poděkování

Děkuji vedoucímu své diplomové práce doc. Ing. Ivo Bukovskému, Ph.D. za zajímavé téma, odborné vedení práce, trpělivost a věnovaný čas.
Děkuji své mamince Iloně za její obětavost a za obrovské množství času, prostředků i podpory, které mi během mých studií věnovala a umožnila mi tak plně se soustředit na své akademické i osobní cíle.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Průcha** Jméno: **Jaroslav** Osobní číslo: **422807**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Strojní inženýrství**

Studijní obor: **Mechatronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Adaptivní řízení nelineárního systému s referenčním modelem

Název diplomové práce anglicky:

Adaptive control of a nonlinear system with a reference model

Pokyny pro vypracování:

Navrhněte řízení se strojovým učením pro nelineární dynamický systém s nelineární funkcí útlumu a nelineárním třením na základě měřených nebo simulovaných dat s pomocí dávkových algoritmů strojového učení. Proveďte případovou studii dávkových učících algoritmů a vhodných neuronových architektur pro aproximaci a řízení systému s ohledem na jeho nelinearity; experimentálně ověřte několik konfigurací parametrů a nastudujte vliv nelinearit na kvalitu aproximace a naučení regulátoru. Pro řízení zvolte např. princip s referenčním modelem(MRAC), funkčnost a kvalitu naučení neuro-regulátoru ověřujte simulací se spojitým modelem soustavy. Odvozené a implementované algoritmy nejprve zdokumentujte a jejich správnost současně demonstrujte na jiném dynamickém systému s konstantními parametry.Na základě vaší analýzy a experimentů navrhněte nejvhodnější architektury pro soustavu a regulátor, jejich konfigurace a učící algoritmy.

Seznam doporučené literatury:

[1] David B. Fogel, Derong Liu ,. James M. Keller. Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation

[2] BUKOVSKY, Ivo, Peter M. BENES a Martin VESELY. Introduction and Application Aspects of Machine Learning for Model Reference Adaptive Control with Polynomial Neurons. Artificial Intelligence and Machine Learning Applications in Civil, Mechanical, and Industrial Engineering [online]. 2020 [vid. 2020-03-07]. Dostupné z: doi:10.4018/978-1-7998-0301-0.ch004

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Ivo Bukovský, Ph.D., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **30.04.2020** Termín odevzdání diplomové práce: **07.08.2020**

Platnost zadání diplomové práce: _____

doc. Ing. Ivo Bukovský, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Miroslav Španiel, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

28.7.2020
Datum převzetí zadání

Podpis studenta

Anotační list

Jméno autora:	Jaroslav Průcha
Název DP:	Adaptivní řízení nelineárního systému s referenčním modelem
Anglický název:	Adaptive control of a nonlinear system with a reference model
Rok:	2020
Obor:	Mechatronika
Ústav:	Ústav mechaniky, biomechaniky a mechatroniky
Vedoucí práce:	doc. Ing. Ivo Bukovský, Ph.D.
Bibliografické údaje:	Počet stran 91 Počet obrázků 42 Počet tabulek 13
Klíčová slova:	adaptivní řízení, torzní kmitání, MRAC, HONU, neuronové jednotky
Keywords:	adaptive control, torsic oscillation, MRAC, HONU, neural units

Abstrakt

Tématem této práce je adaptivní identifikace a řízení systému s nelineárním tlumením pomocí Model Reference Adaptive Control (MRAC). Metody odvozené a implementované v této práci se odvíjejí od současných trendů výzkumu v oblasti HONU (Neuronové jednotky vyššího řádu) na FS ČVUT. K řešení byl zvolen systematický postup.
Nejprve byly implementovány identifikační algoritmy pro lineární (LNU) i nelineární (QNU,CNU) HONU. Správnost těchto algoritmů byla ověřena na několika SISO systémech vykazujících různé stupně nelinearity. Bylo provedeno jejich základní srovnání pro jednotlivé aplikace.
V další části byly implementovány řídící algoritmy pro lineární (LNU) i nelineární (QNU) HONU, včetně nelineárního regulátoru. Tyto algoritmy byly vyzkoušeny, především na lineárním oscilátoru, ale částečně i mj., na modelu tlumeného matematického kyvadla.
Poté byla provedena studie související problematiky optimalizace se zaměřením na učení HONU pro identifikaci a řízení dynamických systémů. Bylo provedeno vyhodnocení optimalizačních algoritmů optimálních pro řešenou problematiku.
Na závěr byla provedena analýza systému torzně kmitajících hmot s nelineárním tlumením. Byly sestaveny pohybové rovnice a bylo implementováno jejich řešení pomocí řešiče diferenciálních rovnic v Pythonu. Tento systém byl identifikován a následně řízen pomocí HONU. Byl zaznamenán vliv koeficientů u nelinearit na úspěšnost zmíněné identifikace a řízení.

Abstract

The topic of this thesis is adaptive identification and control of a system with nonlinear damping using Model Reference Adaptive Control (MRAC). The methods derived and implemented in this work revolve around current trends in research in the area of HONU (Higher Order Neural Units) at the FME CTU. A systematic approach has been chosen in tackling this task
First, identification algorithms have been implemented for linear (LNU) and nonlinear (QNU,CNU) HONU. The correct function of these algorithms has been verified on several SISO systems with various degree of nonlinearity. Their basic comparison for individual applications has been made.
In the next part, control algorithms have been implement for linear (LNU) and nonlinear (QNU) HONU, including nonlinear regulator. These algorithms have been tested, primarily on a damped linear oscillator, but partially also on a model of damped mathematical pendulum.
Furthermore, a study of optimization with emphasis on its use in HONU learning in identification and control has been done. Assessment of suitable optimization algorithms has been done.
Finally, an analysis of a system of torsically oscillating masses with nonlinear damping has been done. Dynamic equations have been written and their solution has been implemented with the use of a differential equation solver in Python. This system has been identified and then controlled with use of HONU. The effect of coefficients in the nonlinearities on aforementioned identification and control has been analysed.

Obsah

1 Zkratky

NU	Neuronová jednotka (Neural Unit)
LNU, QNU, CNU	Lineární, kvadratická, kubická neuronová jednotka
SNU	Statická neuronová jednotka (Static Neural Unit)
DNU	Dynamická neuronová jednotka (Dynamic Neural Unit)
DLNU, DQNU DCNU	Dynamická L/Q/C neuronová jednotka
SLNU, SQNU,SCNU	Statická L/Q/C neuronová jednotka
G-D	Algoritmus Gradient-Descent
L-M	Levenberg-Marquardtův algoritmus
CG	Metoda konjugovaných gradientů
BFGS	Broyden–Fletcher–Goldfarb–Shannonův algoritmus
L-BFGS-B	Paměťově omezený a ohraničený BFGS algoritmus
B-H	Algoritmus Basin Hopping
LSQ	Metoda nejmenších čtverců (Least Square)
DOF	Stupeň, resp. stupně, volnosti (Degrees Of Freedom)
MRAC	Řízení s referenčním modelem
MAE	Užitá metrika střední hodnoty absolutní hodnoty chyby modelu
qErr	Užitá metrika součtu druhých mocnin prvků v chybovém vektoru
maxDiff	Užitá metrika maxima odchylky modelu
mj.	Mimo jiné

2 Úvod

Řízení systémů je integrální součástí mechatroniky a s rostoucí výpočetní silou a dostupností hardwaru také jedním z pilířů současného technického rozvoje - od tradičních strojařských aplikací jako jsou obráběcí stroje, až po moderní aplikace jako elektrická, či hybridní auta, nebo vojenské drony.
Zmíněná dostupnost a síla výpočetních prostředků, spolu s neustálým rozvojem a inovacemi v teorii řízení, otevírá cestu ke stále většímu počtu sofistikovanějších metod řízení. Podmnožinou těchto přístupů jsou metody datové, mezi které patří i aplikace neuronových sítí a jednotek na problematiku identifikace a řízení dynamických systémů.

2.1 Neuronové jednotky vyššího řádu

Neuronové jednotky vyššího řádu [1], zkráceně HONU, patří právě do této skupiny. Jak bude v této práci dále podrobněji rozebráno, základní rovnicí HONU je skalární součin vahového vektoru vektoru $\mathbb{W}$ a vektoru $\mathbb{X}$, který nějakým dále definovaným způsobem obsahuje předchozí stavy a vstupy systému. To vše v diskrétních časových krocích. Matematicky tedy můžeme výstup HONU pro identifikaci zapsat jako $y_n = \mathbb{W} \cdot \mathbb{X} = w_0x_0 + w_1x_1 + ... + w_nx_n$. Analogicky pak pro HONU-regulátor $q = \mathbb{V} \cdot \xi$.[5]
Samotným cílem učení je pak sestavení a řešení optimalizační úlohy pro cílovou funkci, kterou je zpravidla součin druhých mocnin chybového vektoru e, algoritmicky dot(e,e).[5] Existuje řada metod, od různých variant LSQ až po stochastické, které lze ke splnění tohoto cíle použít - z dávkových metod například Levenberg-Marquardtův, či Broyden–Fletcher–Goldfarb–Shannonův algoritmus; z krokových metod pak Gradient-Descent nebo ADAM. [3][4]
Některé open-sourcově implementované metody, např BFGS, umožňují sestavení složitějších kriteriálních funkcí, kde lze mj. penalizovat řízení pomocí HONU za překmit oproti žádané hodnotě, případně při identifikační úloze nelineárního systému sčítat odchylky více různých simulací s různými vstupy, což umožňuje robustnější vystihnutí skutečné dynamiky systému.[3]

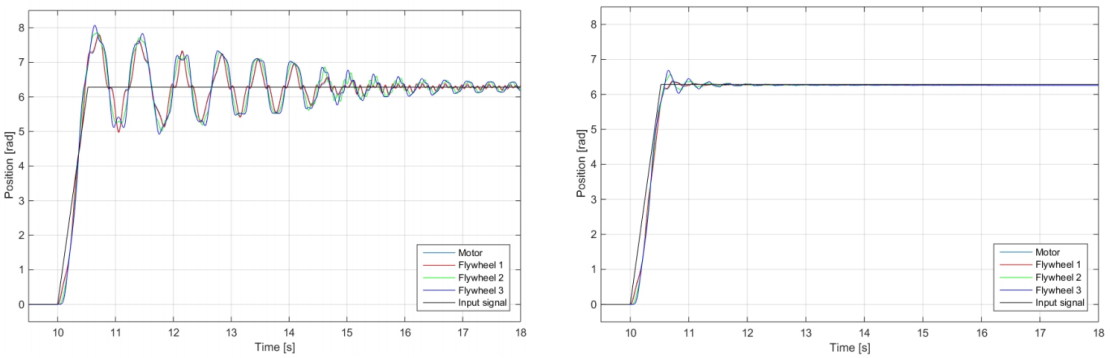
2.2 Další pokročilé metody řízení

HONU není jediný moderní přístup k řízení systémů a na Fakultě Strojní ČVUT probíhá výzkum dalších principů řízení zaměřených na obecné principy i konkrétnější případy.

[1] Angl. Higher Order Neural Units

2.2.1 Vlnové řízení

Jedním z moderních principů řízení zkoumaných na FS ČVUT je vlnové řízení (angl. Wave Based Control). Tato metoda byla na Fakultě Strojní úspěšně použita k řízení obdobného systému k tomu, jaký je analyzován v této práci.[1] Jak



(a) Traditional feedback loop control

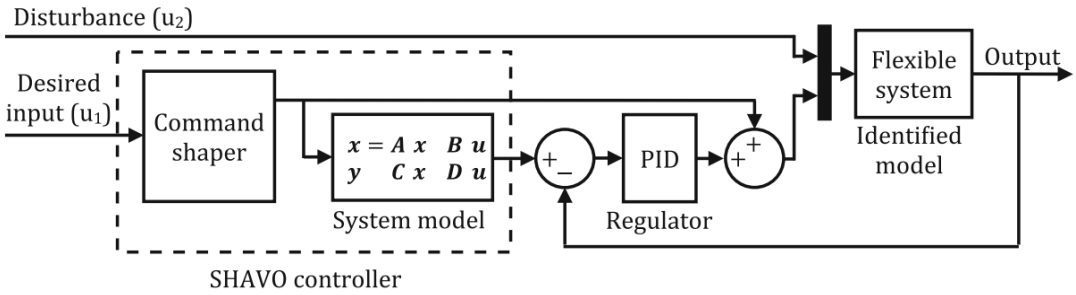
(b) Wave based control (WBC)

Obrázek 1: Úspěšné řízení pomocí Wave Based Control[1]

je patrné z obrázku 1, dokáže tato metoda dobře eliminovat nežádoucí chování, které vzniká při řízení pomocí tradiční zpětnovazební smyčky. Přesněji došlo k výraznému snížení překmitu a podstatnému snížení času ustálení oscilací kolem žádané polohy.

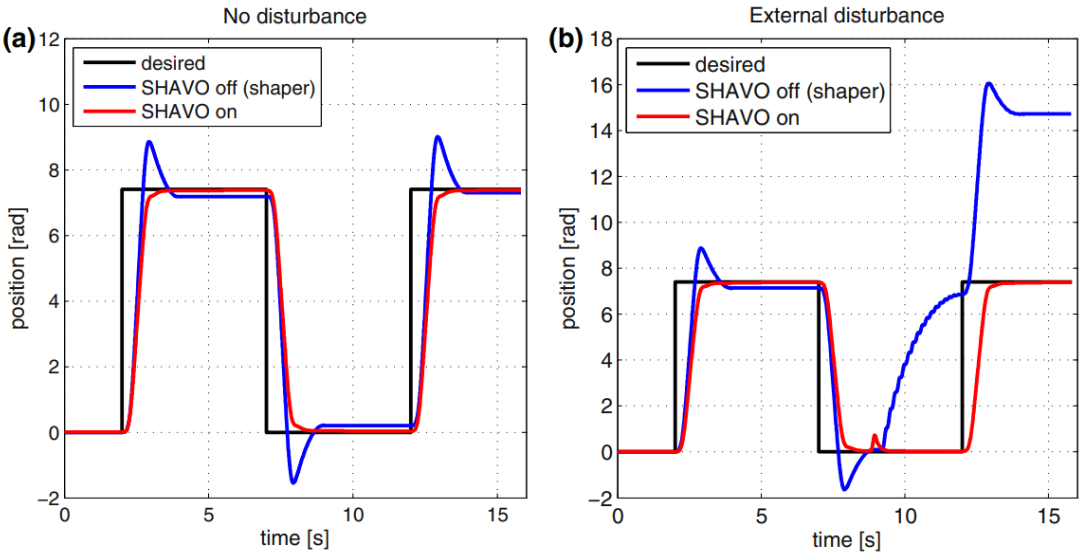
2.2.2 SHAVO řízení

Další metodou hodnou pozornosti vyvíjenou na Fakultě Strojní je SHAVO. Metoda bere svůj název z kombinace slov SHAper + serVO control. Jedná se o metodu kombinující vlastnosti dopředného a zpětnovazebního řízení, jak je znázorněno na následujícím schématu.[2] Výsledky řízení dynamického systému řetězcového



Obrázek 2: Schéma řízení SHAVO[2]

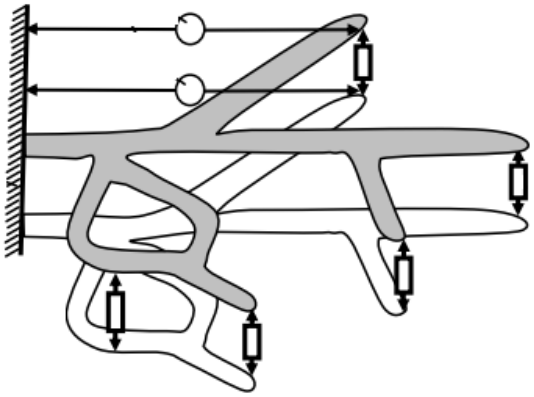
charakteru, podobném systému rozebíranému dále v této práci, byly na FS publikovány v publikaci [2] a vykazují velmi dobré výsledky. Při užití této metody došlo k odstranění překmitů a chování se velmi výrazně zlepšilo při přítomnosti vnějších vzruchů.



Obrázek 3: Výsledek řízení SHAVO[2]

2.2.3 Další zkoumané metody řízení

Rozvoj řízení také probíhá prostřednictvím dalších metod, mezi něž patří následující. *Mechatronická tuhost* je metoda, která využívá sekundární hmotu spojenou s hmotou primární pomocí aktuátorů. Jejím cílem je pomocí zpětnovazebního řízení navýšit tuhost primární hmoty. Oproti aktivním hltičům je sekundární hmota připojena k rámu a může tedy ovlivňovat i statickou výchylku.[6]



Obrázek 4: Schéma aplikace mechatronické tuhosti[6]

Aktivní integrální řízení vibrací je metoda popsaná v článku Active Integral Vibration Control of Elastic Bodies [7], zaměřující se na aktivní tlumení vibrací s využitím piezo-pásku. Metoda je spíše zaměřená na problematiku MIMO řízení, s jeho redukcí na SISO případ. [7] Jako taková však není příliš relevantní ke konkrétní problematice, kterou si klade za cíl tato práce prozkoumat.

3 Simulace systémů

Vzledem k simulační povaze této práce je vhodné věnovat pozornost kromě rozebírané problematiky identifikace a MRAC i uvést základní metody použité pro simulaci systémů.
V práci byly použity dvě hlavní metody - metoda konečných diferencí a převod do stavového popisu s následnou integrací.
i) *Metoda konečných diferencí*
Metody konečné diference se používají k numerickému řešení diferenciálních rovnic. Metoda diskretizuje rovnice a převádí je na soustavy rovnic, které jsou řešitelné algebraickými technikami.[8]
Metoda má základ v Taylorově rozvoji

$$f(x_0+h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f(x_0)}{2!}h^2 + \frac{f'(x_0)}{3!}h^3 + \cdots = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}h^k. \quad (1)$$

Rozvoj lze přepsat jako

$$f(a+h) = f(a) + f'(a)h + R_1(x), \quad (2)$$

kde $R_1(x)$ je chyba aproximace. Vyjádříme derivaci $f'(a)$ a upravíme zápis na

$$\frac{f(x_0 + ih) - f(x_0)}{ih} = f'(x_0) + \mathcal{O}(h). \tag{3}$$

Obdobný přístup lze aplikovat na vyšší derivace. Tato metoda je zatížena numerickými chybami a je potřeba dbát na její stabilitu.[8] Proto byl primárně používán druhý přístup a v několika zbývajících případech byla ověřena stabilita.

ii) *Stavový popis a následná integrace*

Metodu ilustrujme na jednoduché rovnici druhého řádu $m\ddot{v} = kx + b\dot{x}$. Přepíšeme ji do stavového popisu tak, aby se v rovnici vyskytovaly nejvýše derivace prvního řádu:

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= \frac{1}{m}(kx + b\dot{x}). \end{aligned} \tag{4}$$

V takové formě lze rovnice řešit pomocí nástrojů již implementovaných v Pythonu. Jedním z nich je knihovna SciPy, přesněji scipy.integrate.odeint [1]. Samotná implementace takového řešení je popsána dále na konkrétních příkladech.[3][8]

3.1 Model tlumeného kyvadla

Kyvadlo jako systém vykazuje značnou nelinearitu a tedy se jedná o dobrý model, na kterém lze trénovat a testovat metody predikce a řízení systémů vykazujících silné nelineární chování.[9]

Napíšeme nejprve pohybovou rovnici tlumeného kyvadla

$$\frac{d^2\theta(t)}{dt^2} + f_d\frac{d\theta}{dt} + \frac{g}{l}sin\theta(t) = u(t). \tag{5}$$

Rovnici můžeme řešit spojitě pomocí ODE solveru nebo diskrétně například metodou konečných diferencí.

Diferenční schéma získáme dosazením za užití Eulerovy dopředné metody,

$$\ddot{\theta}(t) \simeq \frac{\theta[k+1] - 2\theta[k] + \theta[k-1]}{\Delta t^2} \ , \ \dot{\theta}(t) \simeq \frac{\theta[k+1] - \theta[k]}{\Delta t} \ , \ \theta(t) = \theta[k]. \tag{6}$$

Po dosazení a upravě dostaneme rovnici

$$\theta[k+1] = \frac{u[k] - Mglsin(\theta[k]) + f_d\frac{\theta[k]}{\Delta t} + J\frac{(2\theta[k] - \theta[k-1])}{\Delta t^2}}{\frac{J}{\Delta t^2} + \frac{f_d}{\Delta t}}. \tag{7}$$

Další metodou je úprava diferenciální rovnice do tvaru soustavy diferenciálních rovnic prvního řádu a použití ODE solveru k jejich výpočtu. V jazyku Python takovou funkci poskytuje například knihovna SciPy.

$$\frac{d\theta(t)}{dt} = \omega(t) \tag{8}$$

$$\frac{d\omega(t)}{dt} = \frac{1}{J}(u(t) - Mglsin(\theta(t)) - f_d\frac{d\theta}{dt}) \tag{9}$$

Tuto soustavu lze řešit pomocí ODE solveru, který poskytuje knihovna SciPy.[3] Základním požadavkem tohoto solveru je převedení této soustavy do formy funkce:

Kód 1: Funkce inversePendulum pro simulaci tlumeného matematického kyvadla

```
1 def inversePendulum(angle,t, u, d, regulator):
2     w = angle[0]
3     phi = angle[1]
4     if regulator:
5         u = r0*(d-phi)
6     dphidt = w
7     dwdt = (u - M*g*l*np.sin(phi) - fd * w)/J
8     return [dwdt,dphidt]
```

Funkce má následující parametry

(i) angle - stavový vektor s $\frac{d\theta(t)}{dt}$ a $\frac{d\omega(t)}{dt}$

(ii) t - vektor časových hodnot pro které se soustava řeší

(iii) u - parametr určující hodnotu vstupu u pro daný krok

(iv) d - žádaná hodnota úhlu, použitá v případě, že používáme i regulátor

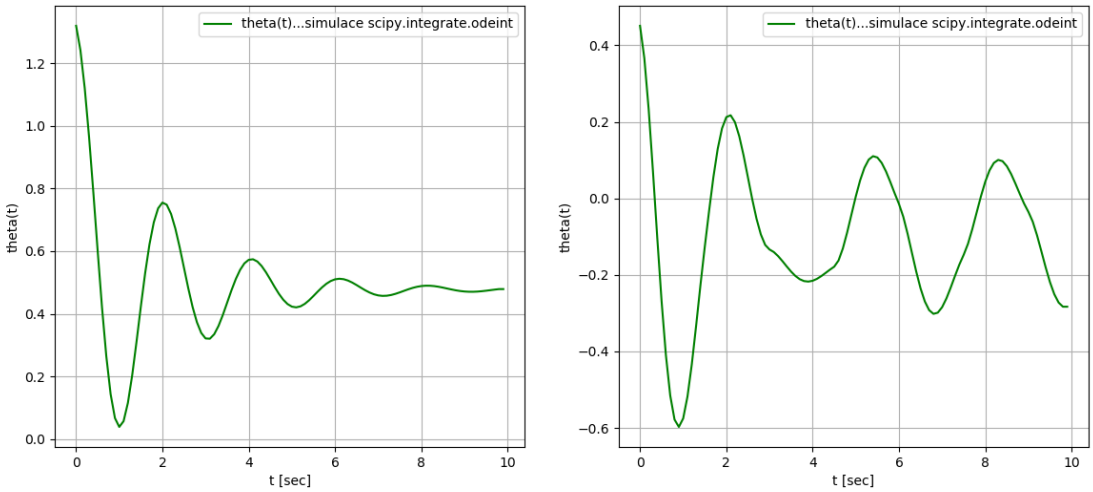
(v) regulator - enable/disable regulatoru, jehož forma se může ve skriptu měnit.

Samotné řešení průběhu pohybu kyvadla probíhá ve smyčce,

Kód 2: Simulace tlumeného kyvadla s využitím funkce inversePendulum a funkce odeint

```
1 for i in range(0, N ):
2     if i==N-1 or i==N:
3         tt = [t[i], t[i]]
4     else:
5         tt = [t[i], t[i+1]]
6     x=odeint(inversePendulum,x0,tt,args=(u[i],d[i],
7                                          regulator))
8     y[i] = x[1, 1]
9     x0 = x[1, :] # nove poc. podm pro dalsi integraci
```

Povšimnutíhodné je především, že řešení v každém kroku integruje rovnice do dalšího kroku $i + 1$. Toto řešení budeme dále považovat za exaktní a bude dále v této práci využito při verifikaci správnosti a přesnosti predikce, resp. simulace, stavu soustavy pomocí HONU.



Obrázek 5: Simulace kyvadla pro různé hodnoty vstupů a počátečních podmínek

Byla provedena verifikace obou metod pro různé počáteční podmínky a průběhy vstupů u. Nebyla v žádném průběhu zjištěna odchylka mezi metodami řešení i) a ii) z předchozí kapitoly.

4 Simulace systému pomocí HONU

4.1 Úvod do základního rozdělení a implementace neuronových jednotek

4.1.1 Lineární neuronová jednotka LNU

Lineární neuronová jednotka pracuje pouze s lineárními verzemi vektoru $\mathbb{X}$ a predikci následujícího kroku pomocí této jednotky lze napsat zapsat jako

$$y_n = \mathbb{W} \cdot \mathbb{X} = w_0 x_0 + w_1 x_1 + ... + w_n x_n. \tag{10}$$

Matice $\mathbb{X}$ přitom odpovídá

$$\mathbb{X} = \begin{bmatrix} 1 \\ y_n[k-1] \\ \dots \\ y_n[k-n_y] \\ u[k-1] \\ \dots \\ u[k-n_u] \end{bmatrix}, \tag{11}$$

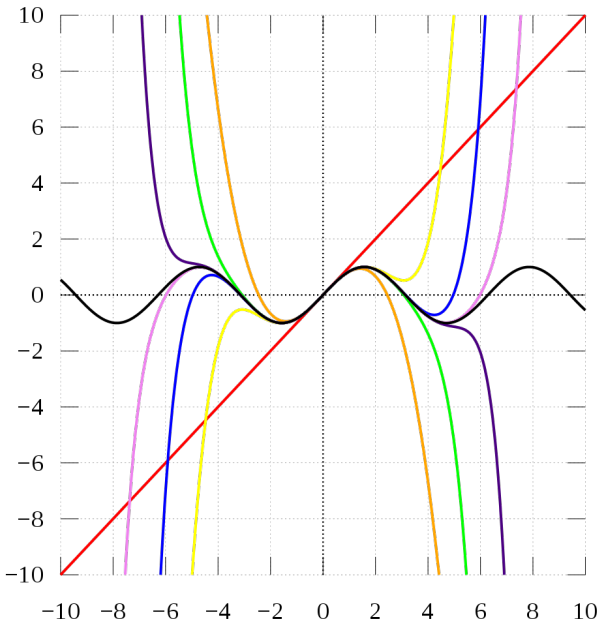
kde n_y označíme jako hloubku neuronu ve výstupu a n_u jako hloubku neuronu ve vstupu.[5]
Je zřejmé, že velikost vektoru je $[n_x \times 1]$, kde $n_x = n_y + n_u + 1$. Platí, že $Col\mathbb{X}^1$ je identické pro LNU s $\mathbb{X}$ a není v tomto případě třeba nic dalšího řešit, ale je vhodné poznamenat, že v samotné implementaci z důvodů přehlednosti programu provedeme operaci $nw = nx$.
Kód pro vygenerování potřebného $\mathbb{X}$ pak vypadá následovně. Za zmínění stojí dosud neuvedená proměnná colx,která v tomto případě odpovídá přímo $\mathbb{X}$, ale dále bude zobecněna pro případy vyššího stupňů HONU.
Při inicializaci vektoru $x[k, kk]$ v kódu si lze povšimnou použití počátečních podmínek. Tento postup zlepšuje, oproti alternativě, kterou je započetí vnější smyčky **for** na hodnotě **max**(nu,ny), přesnost modelu v prvních krocích. To je výhodné mj. při porovnávání přesností modelů.

Kód 3: Kód funkce assembleColX pro LNU

```
print('Learning – LNU')
for k in range(1, N):
    for kk in range(1, ny):
        if k − kk < 0: # add initial conditions
            x[k, kk + 1] = phiPP
        else:
            x[k, kk + 1] = yn[k − kk]
    for kk in range(1, nu):
        if k − kk < 0: # add initial conditions
            x[k, kk + ny + 1] = uPP
        else:
            x[k, kk + ny + 1] = u[k − kk]
        x[k, 0] = 1
        idx = 0
    for i in range(0, nx): # assemble linear colx
        colx[k, i] = x[k, i]
```

4.1.2 Kvadratická neuronová jednotka QNU

Lineární neuronová jednotka poskytuje ze své podstaty pouze lineární aproximace systému. To bude nevyhnutelně nedostačující pro systémy, které vykazují výraznou nelinearitu.



Obrázek 6: Taylorův rozvoj pro r = 1, 3, 5,... [10]

Analogicky k Taylorově řadě, která je ilustrovaná na obr. 6, dosáhneme lepší aproximace užitím vyššího stupně HONU.
Pro značení vyšších stupňů HONU bylo zavedeno značení

$$Col\mathbb{X}^n, \tag{12}$$

kde n značí stupeň HONU. QNU bude v této práci značen jako $Col\mathbb{X}^2$. [5][11][2]
Pro QNU platí rovnice

$$y_n = \sum_i \sum_{j=i} w_{ij} x_i x_j \tag{13}$$

Toto můžeme opět přepsat jako

$$y_n = \mathbb{W} \cdot Col\mathbb{X}^2. \tag{14}$$

[2]Značení v této práci bylo mírně upraveno oproti některým publikacím kvůli usnadnění přechodu mezi implementací v kódu a zde zapsanými rovnicemi. Princip značení byl však zachován.

Správné určení velikosti $Col\mathbb{X}^2$ je jednoduchou kombinatorickou úlohou, jejíž výsledek je $n_w = \frac{n_x(n_x+1)}{2}$. Implementace v Pythonu vypadá následovně:

Kód 4: Kód funkce assembleColX pro QNU

```
if QNU == 1: # CASE QNU
    for k in range(1, N):
        for kk in range(1, ny):
            if k − kk < 0:
                x[k, kk + 1] = phiPP
                # phiPP je pocatecni podminka stavu
            else:
                x[k, kk + 1] = yn[k − kk]
        for kk in range(1, nu):
            if k − kk < 0:
                x[k, kk + ny + 1] = uPP
                # uPP je pocatecni podminka vstupu
            else:
                x[k, kk + ny + 1] = u[k − kk]
        idx = 0
        for i in range(0, nx):
            for j in range(i, nx):
                colx[k, idx] = x[k, i] * x[k, j]
                idx += 1
```

Můžeme uvažovat například soustavu, kde $n_y = 2$ a $n_u = 2$. Bude se tedy jednat o všechny unikátní permutace vzniklé vzájemným pronásobením dvou vektorů $\mathbb{X}$, symbolicky můžeme tuto operaci označit například jako

$$Col\mathbb{X}^2 = \mathbb{X} \times \mathbb{X}. \tag{15}$$

Rozepsané pro zmíněný případ bude, tedy
$Col\mathbb{X}^2 = [1, y_{k-1}, y_{k-2}, u_{k-1}, u_{k-2}, y_{k-1}^2, y_{k-2}^2, y_{k-1}y_{k-2}, u_{k-1}^2,$

$$u_{k-2}^2, u_{k-1}u_{k-2}, y_{k-1}u_{k-1}, y_{k-2}y_{k-2}, y_{k-1}u_{k-2}, u_{k-1}y_{k-2}]$$

4.1.3 Kubická neuronová jednotka CNU

Analogickým rozšířením ke QNU je Kubická neuronová jednotka, CNU (Cubic Neural Unit). Tímto způsobem opět dospějeme k obdobnému popisu

$$Col\mathbb{X}^3 = \mathbb{X} \times \mathbb{X} \times \mathbb{X}, \tag{16}$$

který symbolizuje všechny možné permutace pronásobení těchto tří vektorů. [5],[12] Analogicky pak platí

$$\tilde{y} = \sum_i \sum_j \sum_k w_{ijk} x_i x_j x_k \tag{17}$$

Což opět můžeme opět přepsat jako

$$y_n = \mathbb{W} \cdot Col\mathbb{X}^3. \tag{18}$$

Správné určení velikosti $Col\mathbb{X}^3$ je již mírně náročnější, ale nabízí se představa, dle postupu u QNU, analogicky rozšířit vzorec na $n_w = \frac{n_x(n_x+1)(n_x+2)}{6}$. Jednoduchým ověřovacím skriptem v MATLABu byla správnost této hypotézy ověřena. Implementace v Pythonu vypadá opět analogicky:

Kód 5: Kód funkce assembleColX pro CNU

```
if CNU: # CASE CNU
    for k in range(1, N):
        for kk in range(0, ny):
            if k − kk < 0:
                x[k, kk+1] = phiPP # phiPP je pocatecni podminka stavu
            else:
                x[k, kk+1] = yn[k − kk]
        for kk in range(0, nu):
            if k − kk < 0:
                x[k, kk + ny + 1] = uPP # uPP je pocatecni podminka vstupu
            else:
                x[k, kk + ny + 1] = u[k − kk]
        idx = 0
        for i in range(0, nx):
            for j in range(i, nx):
                for l in range(j, nx):
                    colx[k, idx] = x[k, i]∗x[k, j]∗x[k,l]
                    idx += 1
```

Tyto skripty byly implementovány formou funkce [3] assembleColX(X_, u, phiPP, uPP). Tato funkce představuje jeden krok cyklu **for** k, což umožňuje její použitelnost jak pro dynamické, tak i statické učení.

4.1.4 Vyšší neuronové jednotky HONU

Obecně lze zvyšovat stupeň použitých jednotek analogicky výše.[5] Většina systémů však nevykazuje chování, které by bylo možné vhodně aproximovat pomocí např. HONU pátého stupně, a zároveň udržet počet parametrů v přijatelném rozsahu. Také je vhodné povšimnout si, že velikost $Col\mathbb{X}^n$ stoupá exponenciálně a platí $size(Col\mathbb{X}^n) \sim n_x^n$. Je tedy otázkou, zda není lepší využít výpočetní prostředky raději k rozšíření HONU zvýšením hloubky n_u či n_y.

Tabulka 1: Velikost vektoru $\mathbb{W}$ v závislosti na stupni HONU

[3] Proměnná X_ zde představuje stavy použité ve vektoru x, tedy zpravidla yn nebo yref.

Užitý model HONU	LNU	QNU	CNU	Vyšší řád
Počet prvků vektoru $\mathbb{W}$	n_x	$\frac{n_x(n_x+1)}{2}$	$\frac{n_x(n_x+1)(n_x+2)}{6}$	$\sim n_x^r$

To jednak může způsobit přehnaně vysoké výpočetní nároky, které vedou k nevhodnosti algoritmu pro reálné aplikace v řízení dynamických systémů, ale také bude často dosáhnuto vyšší přesnosti a spolehlivosti predikce použitím stejného počtu prvků, ale ve formě LNU, či QNU nebo maximálně CNU.
Ilustrujme tento růst potřebných parametrů na případu s hloubkou $n_y = 10$ a $n_u = 10$. Zapíšeme délku prostého vektoru $\mathbb{X}$ jako $n_x = n_y + n_u + 1 = 21$.

Tabulka 2: Velikost vektoru $\mathbb{W}$ pro konkrétní případ $n_y = 10$ a $n_u = 10$

Užitý model HONU	LNU	QNU	CNU
Počet prvků vektoru $\mathbb{W}$	21	231	1771

Vidíme, že počet parametrů velmi rychle narůstá nad přijatelnou mez. Je tedy důležité najít vhodnou kombinaci lepší aproximace nelineárního chování pomocí HONU vyššího řádu a hloubky modelu.

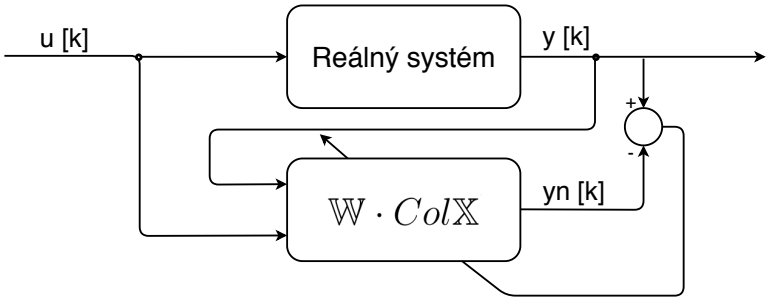
4.2 Modelování systémů pomocí HONU

Při modelování a predikci následujících hodnot je potřeba rozlišit dva typy simulace a učení HONU, které určují, zda se jedná o statický a dynamický model. Rozdíl je zásadní a ovlivňuje samotný způsob učení i použitelnost v predikci dalšího chování systému. V učení rozdíl spočívá v tom, které hodnoty se v průběhu učení používají - tedy předchozí hodnoty vypočtené neuronem, dynamický, nebo hodnoty naměřené z reálného [4] systému, statický.

4.2.1 Statický neuron - predikce

Statické učení spočívá v učení pomocí dat naměřených, resp. odsimulovaných, na systému, který se snažíme takto identifikovat. Metoda se tedy učí predikovat z naměřených hodnot následující hodnotu, či několik hodnot, ale při delší simulaci zpravidla selhává.
Nevýhodou pro reálné aplikace pak je nutnost real-time měření a rychlého zpracování měřených dat. Výhodou je naopak mnohem vyšší přesnost, stabilita a jednodušší učení.

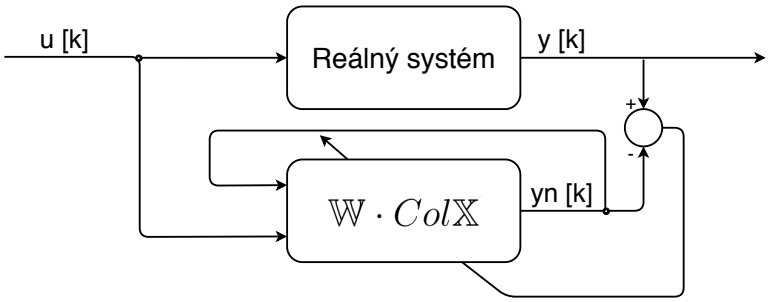
[4] V tomto kontextu se reálným systémem myslí i simulovaný, v případě simulační úlohy.



Obrázek 7: Princip simulace statickým neuronem

4.2.2 Dynamický neuron - simulace

Dynamické simulace ve svém průběhu používají odsimulovaná data, jak je znázorněno na obr. 8, a tedy je při dynamickém učení kladen důraz i na stabilitu, tj. správně naučený systém je schopen držet se průběhu systému aniž by z něj dostával informace. Matematicky lépe tedy můžeme tvrdit, že prvotní chyba, která nevyhnutelně bude do diskrétního numerického systému vnesena, se nezvětšuje.



Obrázek 8: Princip dynamického neuronu

4.2.3 Simulace pomocí dynamického LNU, QNU a CNU s krokovým učením pomocí Gradient Descent algoritmu

i) Mějme nejprve jednoduchý systém popsaný rovnicí[5]:

$$phi[k] = phi[k-1]^2 + \frac{S_u \cdot u[k-1] - phi[k-1]}{\tau} \cdot dt + 0.1 \cdot u[k-1] \cdot phi[k-1] + 0.01 \cdot u[k-1]^2 \tag{19}$$

[5]Pro zjednodušení přechodu mezi skriptem a rovnicí byl použit podobný zápis s indexy a ϕ vyjádřeným písemně.

Rovnice je patrně nelineární s nejvyšším stupněm polynomiární nelinearity $r = 2$. Každá simulace měla tři hlavní fáze:

(1) Ve vnější smyčce **for** epoch **in range**(epochs) je vnořena smyčka **for** k **in range** (1,N)

(2) Ve smyčce **for** k **in range**(1,N) se nejprve sestaví k-tý prvek vektoru colx pomocí funkce colx[k, :] = assembleColX(yn,u,phiPP,uPP). S touto hodnotou se vypočte hodnota výstupu neuronu, chyba e a pomocí updatové funkce dw se upraví hodnota vektoru w.

(3) V další smyčce **for** k **in range**(1,N) simulace pomocí získaného modelu je simulován samotný systém z počátečních podmínek. Vstupy simulace jsou PP a vstup u[k] , výstupem je průběh simulované veličiny yn[k].

Jelikož se jedná o dynamickou simulaci, NU se učí simulovat celý průběh chování systému. Pro statické učení toto neplatí a důraz se klade pouze na odhad následujícího kroku, či pouze několik kroků dopředu a výsledky při použití staticky učeného neuronu na simulaci celého průběhu bez dodávání dat naměřených ze systému v předchozích krocích jsou horší.
Samotné učení pomocí algoritmu Gradient Descent vychází ze vztahu[5]:

$$w[k+1] = w[k] - \Delta w = w[k] - \mu \cdot \frac{\partial Q[k]}{\partial w}, \tag{20}$$

kde Q je chybové kritérium, zpravidla $Q[k] = e^2[k]$, kde $e[k] = y[k] - yn[k]$.
S touto znalostí za $Q[k]$ dosadíme a zderivujeme,

$$\mu \cdot \frac{\partial e[k]^2}{\partial w} = 2\mu \cdot e[k] \cdot \frac{\partial e[k]}{\partial w} = 2\mu \cdot e[k] \cdot \frac{\partial (y[k] - yn[k])}{\partial w}. \tag{21}$$

Parciální derivace přepíšeme jako

$$\frac{\partial (y[k] - yn[k])}{\partial w} = \frac{\partial (y[k])}{\partial w} + \frac{\partial (-yn[k])}{\partial w} = 0 + \frac{\partial (-w \cdot colx[k])}{\partial w} \approx -colx[k]. \tag{22}$$

Algoritmus G-D byl poté implementován v Pythonu:

Kód 6: Algoritmus Gradient Descent

```
1 for epoch in range(epochs):
2     for k in range(1,N):
3         colx[k, :] = assembleColX(yn,u,phiPP,uPP)
4         yn[k] = dot(colx[k,:], w)
5         e = (phi − yn)
6         J = colx[k,:]
7         dw = mu∗e[k]∗colx[k,:]
```

```
8            w = w + dw
9            wall[epoch] = w
10           MAE[epoch]=mean(abs(e))
11       pbar.update(epoch) # progress bar
12   pbar.finish()
```

Tato funkce má především následující výstupy:

(i) wall - matice ukazující průběh jednotlivých vah v průběhu epoch učení

(ii) MAE - střední chyba (mean absolute error) v průběhu epoch

(iii) w - samotný váhový vektor pro neuronovou jednotku

Při používání Pythonu autor narazil na problém s nepřepisováním hodnot v poli yn, který vedl k chybné verifikaci, která se jevila přesnější, než simulace pomocí NU ve skutečnosti je.
Z tohoto důvodu při znovu-používání proměnných je vhodné proměnnou smazat a znovu zadefinovat: **del** yn; yn = ones(N)∗phiPP. V opačném případě mohou nastávat potíže vzniklé pravděpodobně optimalizací na straně použitého vývojového prostředí.[6]

Kód 7: Ukázka principu dynamické simulace v kódu

```
1 del yn; yn = ones(N)*phiPP
2 for k in range(0 , N):
3     colx[k,:] = assemblexColX(yn,u,phiPP,uPP)
4     yn[k] = dot(colx[k,:],w)
```

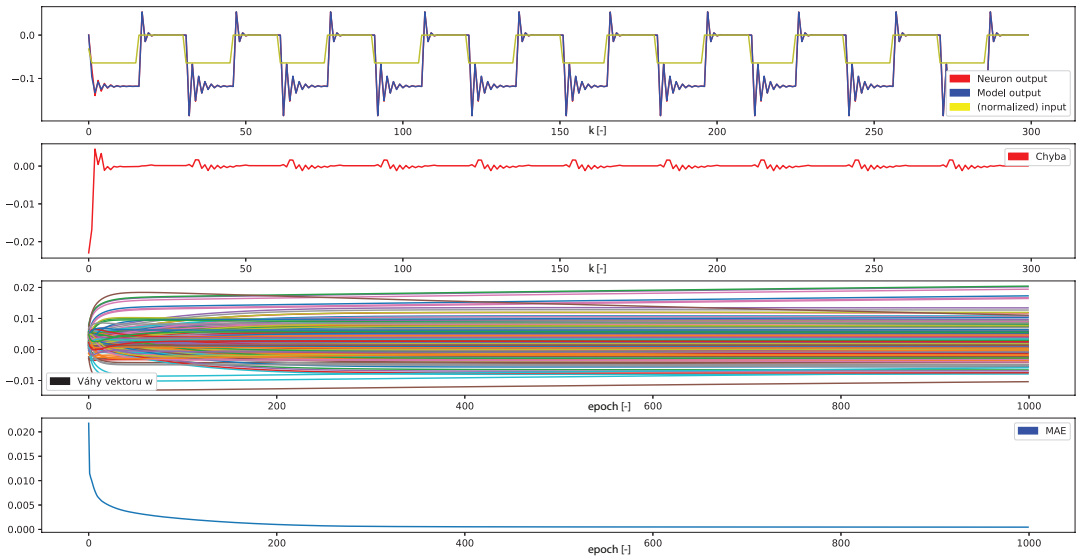
V tomto případě se tedy jedná o dynamickou simulaci. Vypočtený vektor yn[0:k−1] je dále používán pro výpočet yn[k]
Výstup vypadá na pohled obdobně pro LNU, QNU i CNU.
Jelikož budeme porovnávat modely různých parametrů jako LNU, QNU a jejich různé řády,je vhodné zavést nějakou metriku, která bude ukazovat přesnost modelu.
Dobře vypovídající normou je například suma absolutních hodnot prvků chybového vektoru:

Kód 8: Metrika použitá k určování odchylky neuronového modelu od modelovaného systému

```
1 modelErr = zeros(N)
2 modelErr = yn-phi
3 print('Total error is ', sum(abs(modelErr[5:])))
```

[6]Tento závěr vyplynul z faktu, že přidáním řádku mazajícího proměnnou byl tento problém odstraněn.

[7]Bude používáno značení ve formátu XNU_ny_nu, kde X odpovídá stupni HONU, tedy L,Q a C, ny a nu jsou už zavedené stupně neuronu.



Obrázek 9: Průběhy simulace nelineárního systému (19) z této podkapitoly pomocí CNU_4_4 [7]

Vektor modelErr[ ] se začíná sčítat od pátého indexu, vzhledem k peaku nepřesnosti na začátku.
Užitím této metriky můžeme porovnat celkovou odchylku mezi jednotlivými modely.

Tabulka 3: Chyby G-D HONU při simulaci jednoduchého nelineárního systému

Užitý model HONU	LNU_8_8	QNU_8_8	CNU_8_8
Celková chyba	0.435	0.111	0.091

Je tedy vidět, že vyšší stupně HONU aproximují nelinearitu lépe. Je vhodné si povšimnout, že v tomto případě odpovídá modelovaný systém svou mírou nelinearity QNU, ale i tak je CNU o něco přesnější. To může být dáno mj. použitím primitivních optimalizačních algoritmů v této fázi. V další analýze a implementaci problematiky indentifikace a řízení pomocí HONU budou použity sofistikovanější metody optimalizace.

ii) Druhým modelem pro testování identifikace pomocí HONU je model tlu-

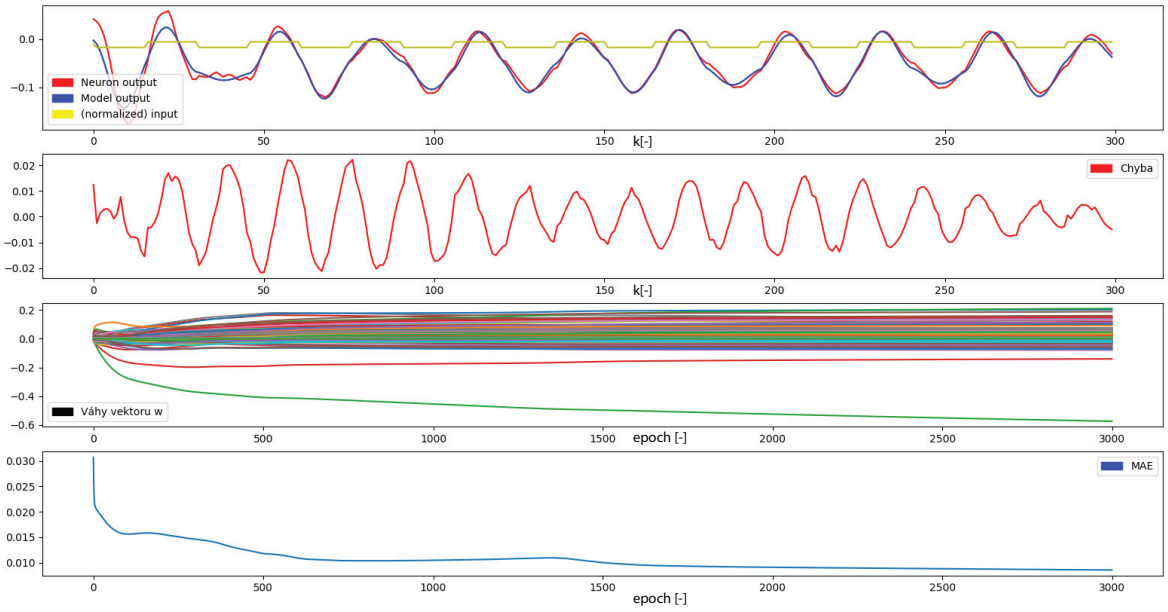
meného kyvadla popsaného rovnicí

$$phi[k] = (u[k-1] - Mglsin(phi[k-1]) + \frac{f_d phi[k-1]}{dt} + \\ + J\frac{(2phi[k-1] - phi[k-2]}{dt^2})\frac{1}{\frac{J}{dt^2} + \frac{f_d}{dt}} \quad (23)$$

Funkce *sin* lze rozepsat pomocí mocninného rozvoje jako

$$sinx = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}. \quad (24)$$

Z toho je zřejmé, že pro přesnou identikaci systému se sinovou nelinearitou bychom potřebovali nekonečný stupeň polynomu. Také platí, že čím vyšší stupeň HONU, tím lépe bychom měli být schopni aproximovat systém ve větším rozsahu.[5] Tato úvaha je analogická k aproximaci funkcí Taylorovým rozvojem[10], který je nám známý ze základních kurzů matematiky.



Obrázek 10: Průběhy simulace tlumeného kyvadla (23) z této podkapitoly pomocí QNU_8_8 s krokovým učením pomocí G-D

Tabulka 4: Chyby HONU při simulaci tlumeného kyvadla

Užitý model HONU	LNU_8_8	QNU_8_8	CNU_8_8
Celková chyba	5.192	2.174	2.223

Dle očekávání, LNU nedokáže aproximovat kyvadlo mimo velmi malé výchylky. Překvapivě QNU dosáhlo lepší přesnosti než CNU. To lze přisoudit jednoduchosti optimalizačních algoritmů, které nedokází spolehlivě najít globální, či alespoň dobré lokální, minimum funkce e^2. V další analýze budou použity sofistikovanější metody.

4.2.4 Simulace pomocí dynamického LNU, QNU a CNU s dávkovým učením pomocí Levenberg-Marquardt algoritmu

Narozdíl od algoritmu Gradient Descent, Levenberg-Marquardtův algoritmus [5][9][12] (dále L-M) funguje na principu tzv. dávkového učení. Tento způsob spočívá ve vytvoření vektoru, který obsahuje hodnoty vektoru $\mathbb{X}$, resp. $Col\mathbb{X}$ pro $k = 1, .., N$. V souladu s již zavedeným značením v předchozích kapitolách tedy platí

$$\mathbb{X}_{LM} = \begin{bmatrix} \mathbb{X}(k=1) \\ \cdots \\ \mathbb{X}(k=N) \end{bmatrix}. \tag{25}$$

Přírůstková funkce[5][12] u L-M algoritmu je

$$d\vec{w} = (((J^T \cdot J)^{-1} + \frac{1}{\mu} I) \cdot J^T) \cdot \vec{e}, \tag{26}$$

kde

$$J = \frac{\partial y_n}{\partial \mathbb{W}} = \frac{\partial(\mathbb{W} \cdot Col\mathbb{X})}{\partial \mathbb{W}} \approx Col\mathbb{X}, \tag{27}$$

a pro rovnici tedy platí

(i) J je jakobián, v tomto případě matice derivací výstupu dle vektoru vah $J = \mathbb{X}_{LM}$

(ii) e je chybový vektor velikosti $N \times 1$, tedy rozdíl výstupu simulovaného systému a HONU

(iii) $d\vec{w}$ je vektor upravující hodnoty vektoru vah $\vec{w}$

(iv μ je koeficient rychlosti učení (angl. learning rate)

Samotná implementace učení v Pythonu pak vypadá následovně.

Kód 9: Učení statického L-M

```
for k in range(1,N):
    colx_static[k,:] = assembleColX(yr)

for epoch in range(epochs):
    yn[:] = dot(colx, w)
    e = (phi − yn)
    J = colx_static
    dw = dot(dot(inv(dot(J.T, J) + 1 / mu ∗ I), J.T), e)
    w = w + dw
```

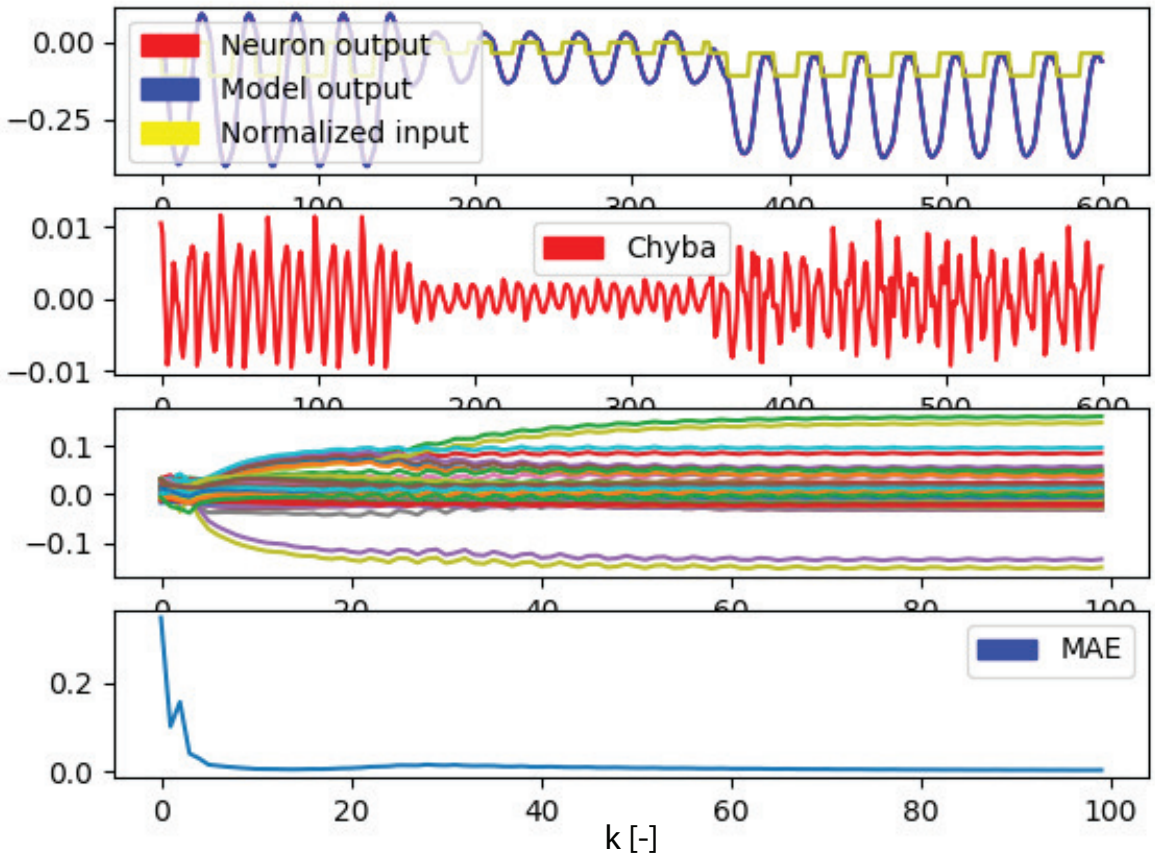
Kód 10: Učení dynamického L-M

```
for epoch in range(epochs):
    for k in range(1,N):
        colx[k,:] = assembleColX(yn)
        yn[k] = dot(colx[k,:], w)
    e = (phi − yn)
    J = colx
    dw = dot(dot(inv(dot(J.T, J) + 1 / mu ∗ I), J.T), e)
    w = w + dw
```

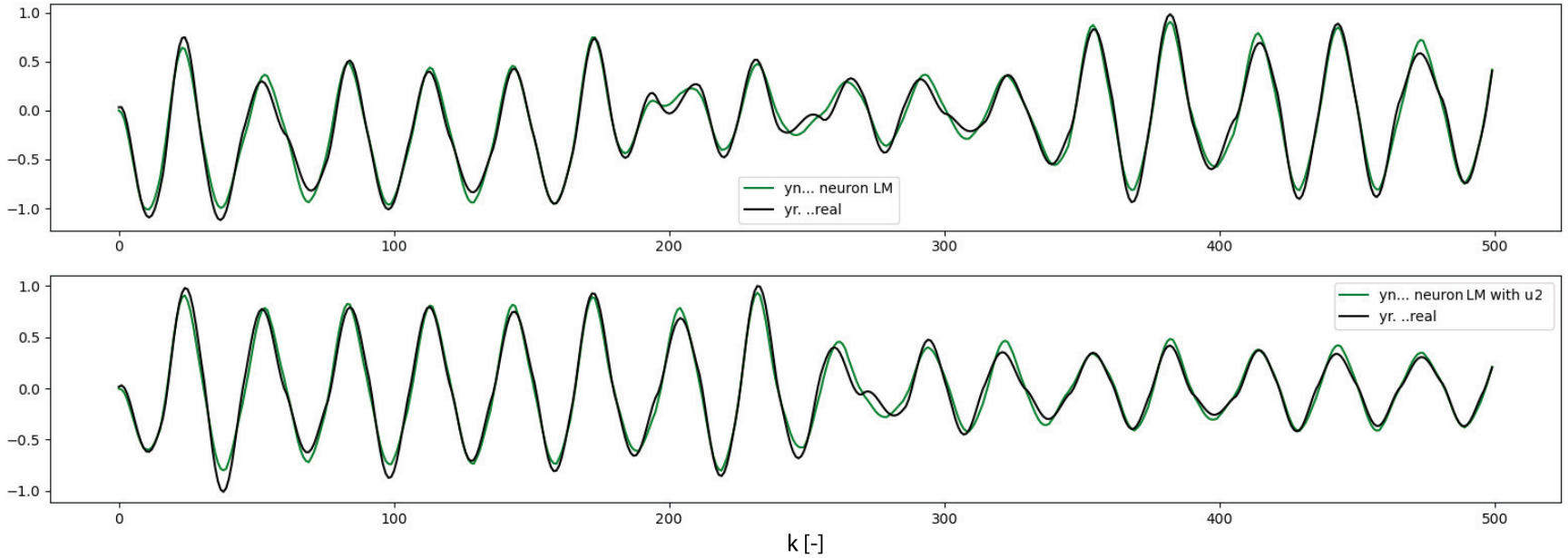
Tabulka 5: Chyby L-M HONU při predikci kyvadla v čase 60s.

Užitý model HONU	LNU_8_8	QNU_8_8	CNU_8_8
Celková chyba	1.708	1.073	0.741

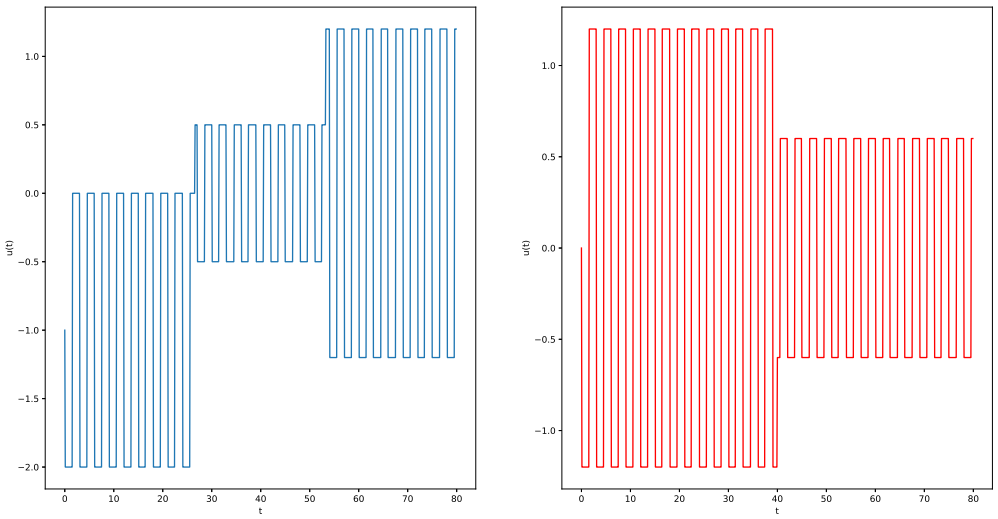
Z průběhu je vidět, že váhy vektoru $\mathbb{W}$ se poměrně rychle ustálí a po asi 100 epochách se již nemění.



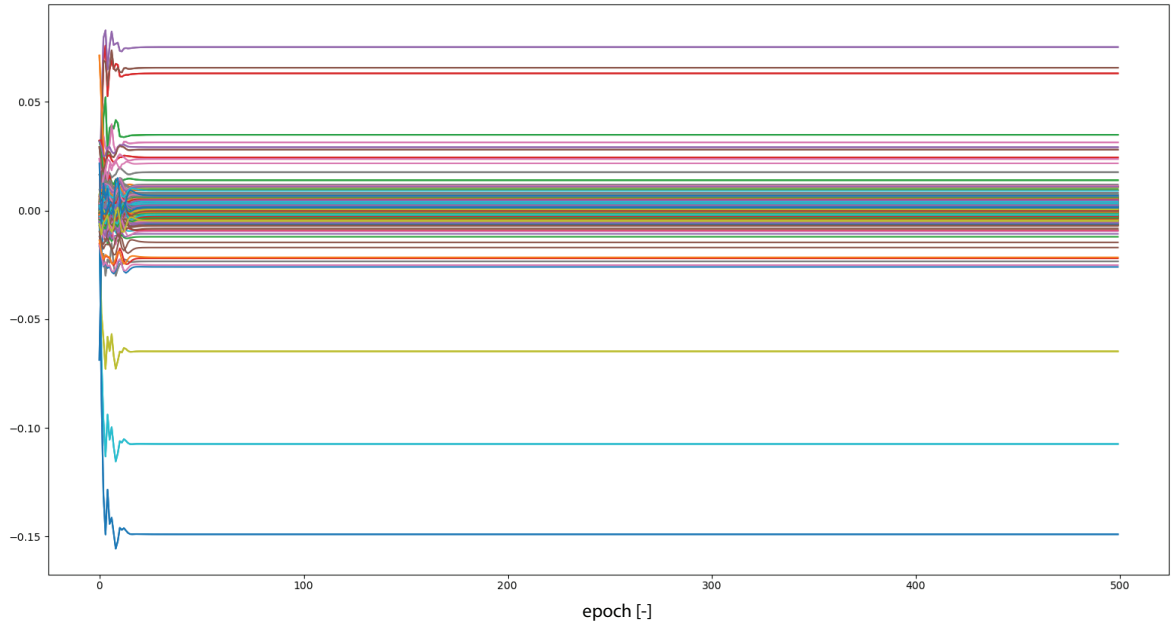
Obrázek 11: Průběhy predikce tlumeného kyvadla (23) z této podkapitoly pomocí statického QNU_8_8 s dávkovým učením



Obrázek 12: Průběhy simulace tlumeného kyvadla (23) z této podkapitoly pomocí dynamického QNU_10_10 s dávkovým učením



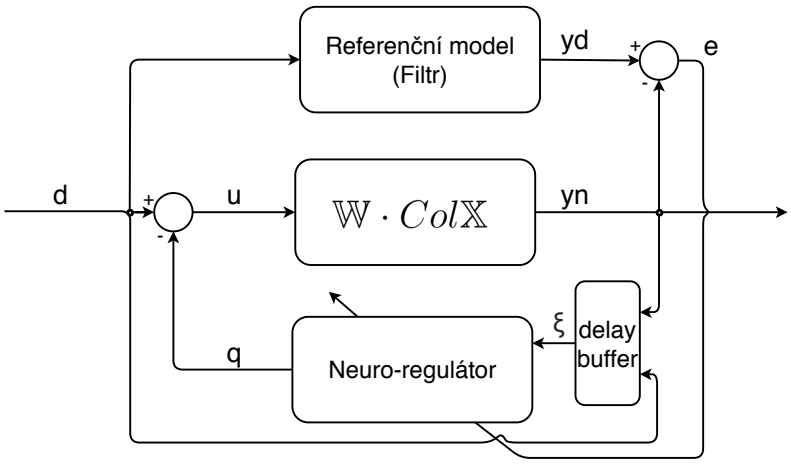
Obrázek 13: Učicí (vlevo) a testovací vstup pro simulaci tlumeného kyvadla



Obrázek 14: Průběh vah učení HONU pomocí L-M k simulaci tlumeného kyvadla

5 Řízení pomocí HONU neuroregulátoru

Kromě identifikace systémů lze HONU přístup použít i pro regulaci veličin. Výhodou oproti konvenčním metodám je, že se jedná o datový přístup, a tedy není nutné ručně nastavovat regulátor např. Ziegler-Nicholsovou metodou. Celkové možnosti regulace také díky univerzálnosti HONU mohou dosahovat dobrých výsledků. Nevýhodou je mj. výpočetní náročnost HONU a otázka stability zpětnovazebního systému.



Obrázek 15: Schéma funkce neuroregulátoru.

5.1 Neuroregulátor

Neuroregulátor je HONU, stejně jako již představené NU, pomocí kterých jsme simulovali systémy.[5][12] Tedy platí

$$q = \mathbb{V} \cdot \xi \tag{28}$$

Opět je otázkou, jak budeme definovat vektor ξ.
i) Výstup neuronu y_n
První možností je používat ve vektoru ξ hodnoty výstupu simulační HONU, tedy y_n

$$\xi = \begin{bmatrix} 1 \\ y_n[k-1] \\ \dots \\ y_n[k-n_v] \end{bmatrix}, \tag{29}$$

kde n_v je stupeň neuroregulátoru.[5]
ii) Výstup referenčního modelu y_{ref}
Druhou zkoumanou možností je použití ve vektoru ξ hodnot z referenčního modelu y_{ref}.[12]

$$\xi = \begin{bmatrix} 1 \\ y_{ref}[k-1] \\ \cdots \\ y_{ref}[k-n_v] \end{bmatrix}, \tag{30}$$

kde n_v je stupeň neuroregulátoru.
Dále při učení lze rozlišit dva postupy z hlediska použitého y_n. Buďto se standartně počítá výstup HONU $y_n = \mathbb{W} \cdot colX$, nebo se může použít metoda, kde se předpokládá,že systém je dobře řízen a tedy všechny předchozí hodnoty y_n odpovídají refernčnímu y_{ref}. Tedy, přesnější zapsáno, $y_n[r] = y_{ref}[r]$ pro $r \in \{k-1, ..., k-ny\}$. Takto se model učí držet se neustále na požadované hodnotě. Metoda se však jeví méně rigorózní, jelikož není nijak zaručena stabilita v numerickém slova smyslu, tedy, že drobné odchylky nezačnou narůstat nad přípustnou mez. Pro některé dynamické systémy se však může jednat o zajímavý přístup.

5.2 Učení neuroregulátoru pomocí algoritmu Gradient Descent

Minimalizujeme funkci $e^2 \to min$ v parametrech $\mathbb{V}$ a tedy, mj. dle [5],

$$\Delta\mathbb{V} = -\frac{\mu}{2}\frac{\delta e_{ref}^2(k)}{\delta\mathbb{V}} = \mu e_{ref}(k)\frac{\delta y_n(k)}{\delta\mathbb{V}}. \tag{31}$$

Je tedy potřeba zejména určit hodnotu $\frac{\delta y_n(k)}{\delta\mathbb{V}}$.

$$\frac{\delta y_n(k)}{\delta\mathbb{V}} = \frac{\delta}{\delta\mathbb{V}}\mathbb{W}\mathbb{X} \tag{32}$$

Vektor $\mathbb{X}$ bude obsahovat členy předešlých stavů $y(k)$ a vstupů $u(k)$. U vstupů je závislost na $\mathbb{V}$ snadno zjistitelná, u určení derivace $y(k-p)$, kde $p \in \mathbb{Z}, p \in \{1, .., n_y\}$, je potřeba hlubší analýza. Je zřejmé, že i předchozí vztahy budou nějak závislé na hodnotách parametrů regulátoru. Rozepsáním této derivace dostaneme

$$\mathbb{W}\frac{\partial\mathbb{X}}{\partial\mathbb{V}} = \mathbb{W}\frac{\partial}{\partial\mathbb{V}}\begin{bmatrix} 1 \\ y_n[k-1] \\ \cdots \\ y_n[k-n_y] \\ u[k-1] \\ \cdots \\ u[k-n_u] \end{bmatrix}. \tag{33}$$

Nejprve provedeme derivaci prvku vstupu $\frac{\partial u[k]}{\partial \mathbb{V}}$.

$$\frac{\partial u[k]}{\partial \mathbb{V}} = \frac{\partial (d[k] - q[k])}{\partial \mathbb{V}} = -\frac{\partial (q[k])}{\partial \mathbb{V}} = -\frac{\partial (\mathbb{V} \cdot \xi)}{\partial \mathbb{V}} = -\xi^T - \mathbb{V} \cdot \frac{\partial (\xi)}{\partial \mathbb{V}} \tag{34}$$

Člen derivace $\mathbb{V} \cdot \frac{\partial (\xi)}{\partial \mathbb{V}}$ závisí na použitém vektoru ξ. V případě použití hodnot y_{ref} bude nulový, v případě použití y_n bude nenulový. Ve druhém případu lze budto odvodit tyto hodnoty pomocí rekurentní backpropagace, či, v závislosti na dynamice konkrétního systému, v některých případech hodnoty zanedbat a položit je rovny nule.[5][12] Tato problematika bude o něco více rozvinuta ještě v kapitole věnované L-M algoritmu.

$$\frac{\partial y[k]}{\partial \mathbb{V}} = \mathbb{W}\frac{\partial \mathbb{X}[k]}{\partial \mathbb{V}} = \mathbb{W}\frac{\partial}{\partial \mathbb{V}} \begin{bmatrix} 1 \\ y_n[k-1] \\ \cdots \\ y_n[k-n_y] \\ u[k-1] \\ \cdots \\ u[k-n_u] \end{bmatrix}. \tag{35}$$

Zde opět vystupují již vyřešené derivace $u[k-p]$, kde $p \in \mathbb{Z}, p \in \{1, .., n_u\}$ a rekurzivně se zde objevují derivace $y[k-r]$, kde $r \in \mathbb{Z}, r \in \{1, .., n_y\}$. Využijeme-li této rekurzivity, můžeme se dostat až na počátek časového intervalu. Zde zřejmě regulátor roli v hodnotě stavu y_n nehrál, a tedy i $\frac{\partial y[k]}{\partial \mathbb{V}} = 0$ pro $k \leq 0$.
Algoritmicky z této analýzy vyplývá, že hodnotu $\frac{\partial y[k_{start}]}{\partial \mathbb{V}}$ určit dokážeme. Tuto hodnotu uložíme do pole a budeme ji dále dosazovat ve výpočtu dalších derivací.

5.3 QNU Regulátor u QNU systému

Při odvozování budeme postupovat analogicky k LNU-LNU systému. Rozdíl spočívá v tom, že kromě $u[k-p]$, kde $p \in \mathbb{Z}, p \in \{1, .., n_u\}$ a $y[k-r]$, kde $r \in \mathbb{Z}, r \in \{1, .., n_y\}$, se objeví i kvadratické prvky představené již v předchozích kapitolách. V principu se na postupu nic nemění. Pro součin dvou prvků použijeme pravidlo o derivaci součinu, tedy

$$\frac{\partial (f \cdot g)}{\partial \mathbb{V}} = g\frac{\partial (f)}{\partial \mathbb{V}} + f\frac{\partial (g)}{\partial \mathbb{V}} \tag{36}$$

Pro názornost uvedeme příklad kvadratického vektoru $Col\mathbb{X}^2$

$$Col\mathbb{X} = \begin{bmatrix} 1 \\ y_n[k-1] \\ u[k-1] \\ u[k-1]^2 \\ y_n[k-1]^2 \\ u[k-1]y_n[k-1] \end{bmatrix}. \tag{37}$$

Z výše uvedené derivace vezmeme prvek $u[k-1]y_n[k-1]$. Při dalším postupu použijeme pravidlo o derivaci součinu a dostaneme

$$\frac{\partial(u[k-1]\, y_n[k-1])}{\partial \mathbb{V}} = u[k-1]\frac{\partial y_n[k-1]}{\partial \mathbb{V}} + y_n[k-1]\frac{\partial u[k-1]}{\partial \mathbb{V}} \tag{38}$$

Derivace $\frac{\partial y_n}{\partial \mathbb{V}}$ jsou pro nás známé z předchozích kroků. V počátečních krocích je naopak $y_n[k_{start}]$ nezávislé na $\mathbb{V}$ a tedy je derivace rovna nule. Derivace $\frac{\partial u[k]}{\partial \mathbb{V}}$ však nulová nebude a skrze ji se zaplní i derivace y_n. Pro QNU regulátor tedy bude platit

$$\begin{aligned} \frac{\partial y_n[k]}{\partial \mathbb{V}} = \mathbb{W}Col\mathbb{X}^2[k-1] &= \mathbb{W}\begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]^2}{\partial \mathbb{V}} \\ \frac{\partial y_n[k-1]^2}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]y_n[k-1]}{\partial \mathbb{V}} \end{bmatrix} \\ &= \mathbb{W}\begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ -\xi^T[k-1] \\ 2u[k-1](-\xi^T[k-1]) \\ 2y_n[k-1]\frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ (-\xi^T)y_n[k-1] + u[k-1]\frac{\partial y_n[k-1]}{\partial \mathbb{V}} \end{bmatrix}. \end{aligned} \tag{39}$$

Pro případ kvadratického regulátoru QNU se změní také vektor ξ. Ten bude vypadat obdobně k $Col\mathbb{X}^2$, tedy pro příklad $n_v = 2$

$$\xi^2 = \begin{bmatrix} 1 \\ y_n[k-1] \\ y_n[k-2] \\ y_n[k-1]^2 \\ y_n[k-1]y_n[k-2] \\ y_n[k-2]^2 \end{bmatrix}. \tag{40}$$

U derivace vstupu $\frac{\partial u[k]}{\partial \mathbb{V}}$ se toto dále neprojeví a platí opět

$$\frac{\partial u[k]}{\partial \mathbb{V}} = \frac{\partial (d[k] - q[k])}{\partial \mathbb{V}} = -\frac{\partial (q[k])}{\partial \mathbb{V}} = -\frac{\partial (\mathbb{V} \cdot \xi)}{\partial \mathbb{V}} = -\xi^T - \mathbb{V} \cdot \frac{\partial (\xi)}{\partial \mathbb{V}}. \tag{41}$$

5.4 Neuronový regulátor s dávkovým učením

Pracujeme v diskrétním čase s dynamickým systémem, který se snažíme identifikovat a řídit pomocí HONU. Oproti krokovému učení, kde v každém časovém kroku k přepočteme vahové vektory $\mathbb{W}$ a $\mathbb{V}$, přepočítávají se při dávkovém učení tyto vektory pro všechny časové kroky $k \in 1, .., N$. [5][12]
Zřejmou výhodou takového přístupu oproti krokovému učení je, že přírůstková funkce vektoru vah bere v potaz celý průběh systému v nějakém rozmezí. To umožňuje lépe identifikovat systémy, které vykazují různé chování v různých oblastech. Zatímco krokové učení bude vahový vektor postupně přeučovat, dávkové učení se bude snažit nafitovat průběh identifikovaného modelu na celý zkoumaný průběh.
Z implementačního hlediska je potřeba rozlišit jednotlivé fáze učení. Nejprve je zapotřebí získat data systému, který budeme identifikovat a řídit.
Schematicky batchové učení vypadá následovně.

Kód 11: Obecný průběh identifikace a řízení systému pomocí HONU

```
# 1. system simulation
for k in range(N):
    yr[k] = simulateSystem(u);
# 2. identification learning
for epoch in range(epochs_identification):
    calculate y, colX, e, Jw
    calculate dw
    w ← w + dw
# 3. regulator learning
for epoch in range(epochs_regulator):
    for k in range(N):
        calculate xi[k], q[k], u[k], yref[k], y[k], yr[k]
        calculate colX[k], e[k], dydv[k,:], dxdro[k]
        calculate Jro[k,:], Jv[k,:]
    calculate dv, dro
    v ← v + dv
    ro ← ro + dro
# 4. regulator testing
for k in range(N):
    calculate yn[k], xi[k], q[k], d[k]
    calculate u[k]
    yr_test[k] = simulateSystem(u)
```

5.4.1 LNU-LNU regulátor s učením pomocí Levenberg-Marquardtova algoritmu

Updatová funkce pro L-M algoritmus[5] je obecně

$$d\vec{v} = (((J_v^T \cdot J_v)^{-1} + \frac{1}{\mu} I) \cdot J_v^T) \cdot \vec{e}_{ref}, \tag{42}$$

kde J_v je Jacobimo matice a její k-tý řádek lze zapsat jako

$$J_v[k,:] = \frac{\partial y_n[k]}{\partial \mathbb{V}} \tag{43}$$

i) Odvození s využitím řetízkového pravidla

Tedy s využitím řetězového pravidla pro derivace lze obecně zapsat

$$J_v[k,:] = \frac{\partial y_n[k]}{\partial \mathbb{V}} = \frac{\partial y_n}{\partial u} \frac{\partial u}{\partial q} \frac{\partial q}{\partial \mathbb{V}}. \tag{44}$$

Po částech vyjádříme členy této rovnice. Nejprve rozepíšeme derivaci $\frac{\partial y_n[k]}{\partial u}$, tedy

$$\frac{\partial y_n[k]}{\partial u} = \mathbb{W} \begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ \vdots \\ \frac{\partial y_n[k-n_y]}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]}{\partial \mathbb{V}} \\ \vdots \\ \frac{\partial u[k-n_u]}{\partial \mathbb{V}} \end{bmatrix}, \frac{\partial u[k-1]}{\partial \mathbb{V}} = -r_0 \xi[k-1]^T \tag{45}$$

Rozměrově operace odpovídá $[1, n_w] \times [n_w, n_v] = [1, n_v]$.
Derivace $\frac{\partial u}{\partial q} = -1$.
Třetí derivací v součinu dle chain rule je $\frac{\partial q(k)}{\partial \mathbb{V}}$. Podle pravidla o derivaci součinu funkcí můžeme napsat

$$\frac{\partial q(k)}{\partial \mathbb{V}} = \frac{\partial(\mathbb{V} \cdot \xi)}{\partial \mathbb{V}} = \xi^T + \mathbb{V} \cdot \begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ \vdots \\ \frac{\partial y_n[k-n_v]}{\partial \mathbb{V}} \end{bmatrix} \tag{46}$$

Rozměrově se jedná o $[1 \times n_v] + [1 \times n_v] \cdot [n_V \times n_v] = [1 \times n_v]$
Takže platí $J_v[k,:] = \frac{\partial y_n[k]}{\partial \mathbb{V}} = \frac{\partial y_n}{\partial u} \frac{\partial u}{\partial q} \frac{\partial q}{\partial \mathbb{V}}$ a rozměrově $dim(J_v[k,:]) =$

$-r_0[1 \times n_v] \cdot [1 \times n_v]$.
Jelikož je rozměr $J_v[k,;]$ v časovém kroku k $[1 \times n_v]$, bude celkový rozměr J_v roven $[N \times n_v]$, což je očekávaný rozměr matice.

ii) Odvození v maticovém tvaru

Odvození tvaru jakobiánu provedeme analogicky k postupu u Gradient Descentu

$$\frac{\partial y[k]}{\partial \mathbb{V}} = \mathbb{W}\frac{\partial \mathbb{X}[k]}{\partial \mathbb{V}} = \mathbb{W}\frac{\partial}{\partial \mathbb{V}}\begin{bmatrix} 1 \\ y_n[k-1] \\ \cdots \\ y_n[k-n_y] \\ u[k-1] \\ \cdots \\ u[k-n_u] \end{bmatrix}. \tag{47}$$

A vtáhnutím derivací dovnitř dostaneme

$$J_v[k,:] = \frac{\partial y_n[k]}{\partial \mathbb{V}} = \mathbb{W}\mathbb{X} = \mathbb{W}\begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ \vdots \\ \frac{\partial y_n[k-n_y]}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]}{\partial \mathbb{V}} \\ \vdots \\ \frac{\partial u[k-n_u]}{\partial \mathbb{V}} \end{bmatrix}. \tag{48}$$

Stejně jako při odvození G-D algoritmu pro regulátor si všimneme, že derivace $\frac{\partial y_n[k-r]}{\partial \mathbb{V}}$, kde $r \in \mathbb{Z}, r \in \{1,..,n_y\}$ se rekurzivně objevují ve výpočtu dalších derivací. Dojdeme-li na začátek regulovaného intervalu, derivace jsou zřejmě rovny nule, jelikož regulátor ještě neběžel a nemohl tedy ovlivnit výstupní hodnotu neuronu. Stačí tedy od začátku ukládat derivace a používat je pro výpočet dalších derivací. Tento přístup se nazývá *rekurentní backpropagace*[8].[5][12]
Při samotné implementaci systému je také možné provést zanedbání[5]

$$\frac{\partial y_n[k-r]}{\partial \mathbb{V}} \approx 0 \ , \ \ r \in \mathbb{Z}, r \in \{1,..,n_y\}. \tag{49}$$

Jedná se o zjednodušení, jehož vhodnost závisí na konkrétní dynamice systému. Obecně lze tento postup zanedbání vyzkoušet a pokud učení probíhá dobře, je zanedbání přípustné. Pokud průběh není ideální, je vhodné derivace nezanedbávat a využít již zmíněné backpropagace.

[8] Angl. recurrent backpropagation

5.4.2 QNU-QNU regulátor s učením pomocí Levenberg-Marquardtova algoritmu

Pro odvození L-M algoritmu pro regulátor použijeme druhý přístup uvedený v předchozí kapitole vzhledem k jeho značné podobnosti k odvození pro případ odvození identifikace pomocí Gradient Descentu.

$$J_v[k,:] = \frac{\partial y_n[k]}{\partial \mathbb{V}} = \mathbb{W}\frac{\partial \mathbb{X}}{\partial \mathbb{V}} \tag{50}$$

Můžeme tedy v analogickém zápisu k (30) zapsat k-tý řádek Jakobiho matice jako

$$J_v[k,:] = \frac{\partial y_n[k]}{\partial \mathbb{V}} = \mathbb{W}Col\mathbb{X}^2[k-1] = \mathbb{W}\begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]^2}{\partial \mathbb{V}} \\ \frac{\partial y_n[k-1]^2}{\partial \mathbb{V}} \\ \frac{\partial u[k-1]y_n[k-1]}{\partial \mathbb{V}} \end{bmatrix}$$

$$= \mathbb{W}\begin{bmatrix} 0 \\ \frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ -\xi^T[k-1] \\ 2u[k-1](-\xi^T[k-1]) \\ 2y_n[k-1]\frac{\partial y_n[k-1]}{\partial \mathbb{V}} \\ (-\xi[\urcorner - \nVdash]^T)y_n[k-1] + u[k-1]\frac{\partial y_n[k-1]}{\partial \mathbb{V}} \end{bmatrix}. \tag{51}$$

Pro problematickou derivaci $\frac{\partial y_n}{\partial \mathbb{V}}$ použijeme stejný postup jako v metodě Gradient Descent, resp. LNU L-M. Jedná se opět o tzv. rekurentní backpropagation, kde derivace v předešlých krocích využijeme k výpočtu dalších kroků.

5.5 Řízení 1 DOF lineárního tlumeného oscilátoru

Algoritmy identifikace spolu s řízením byly, mimo jiné, zkoušeny na lineárním tlumeném oscilátoru, popsaném rovnicí

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = u(t). \tag{52}$$

Tento model byl aproximován rovnicí

$$yr[k] = (u[k-1]+yr[k-1]\cdot(\frac{2m}{dt^2}-k_{param})+m\cdot yr[k-2]\cdot(\frac{b}{2dt}-\frac{m}{dt^2}))\cdot\frac{1}{\frac{m}{dt^2}+\frac{b}{2dt}}. \tag{53}$$

s využitím metody konečných diferencí. Stabilita tohoto schématu byla úspěšně ověřena i pomocí odesolveru z knihovny SciPy na řadě různých vstupů.

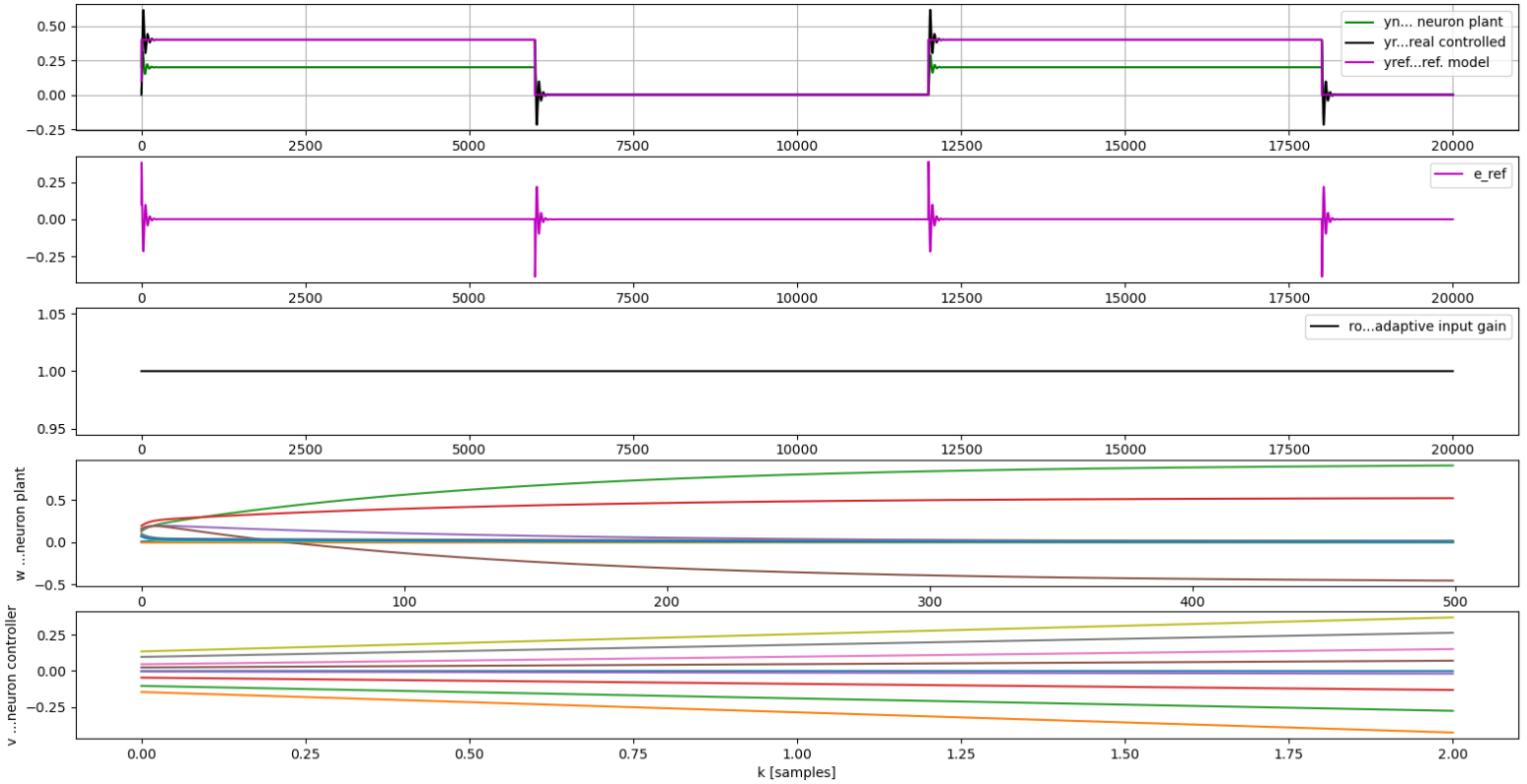
Kód 12: Implementace řízení pomocí HONU

```
for k in range(N):
    if k<10:
        yref[k] = mean(d[1:k + 10])
    else:
        yref[k] = mean(d[k−10:k+10])

for epoch in range(epochs_reg):
    for k in range(nvy,N):
        xi[k,:] = assembleXi(yref,phiPP,nvy,nvx)
        xi[k, 0] = 1
        q[k] = dot(v,xi[k,:])
        u[k] = (d[k] − q[k]) ∗ ro
        colx[k, :] = assembleColX(yn,u,uPP,phiPP)
        yn[k] = dot(w, colx[k,:])
        eref[k] = yref[k] − yn[k]
        dxdv[1+ny+1:,:]= dxdv[1+ny:−1,:]
        dxdv[1+ny,:]=−ro∗xi[k,:]
        dydv=−dot(w,dxdv)
        Jv[k,:]=dydv
        dxdro[2:] = dxdro[1:−1]
        dxdro[1] = d[k − 1] − q[k − 1]
        Jro[k,:] = dot(dxdro,w)
    Jv = dydv_ar
    dv = dot(dot(inv(dot(Jv.T, Jv) + 1 / muv ∗ Iv), Jv.T), eref)
    dro = dot(dot(inv(dot(Jro.T, Jro) + 1 / muro), Jro.T), eref)
    v = v + dv; ro = ro + dro
    vall[epoch] = v; rall[epoch] = ro
    MAE[epoch]=mean(abs(eref))
    totErr[epoch] = sum(abs(eref[20:]))
    if(epoch>3 and totErr[epoch−1]<totErr[epoch] and
                                    totErr[epoch−2]<totErr[epoch]):
        muv = 0.95∗muv; muro = 0.95∗muro
        decreased+=1
        print('Learning rate decreased #', decreased,' times')
    if epoch > 35 and mean(totErr[epoch−25:epoch])
                                    >mean(totErr[epoch−35:epoch−10]):
        print('Stopping the regulator learning at epoch #', epoch +1,'
                                    due to rising mean of regulator error.')
        break
    print('Total error is ', sum(abs(eref)))
```

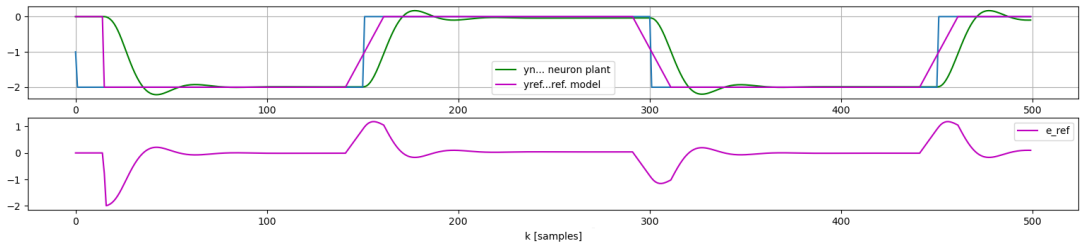
Jedná se o implementaci rovnic odvozených v předchozí kapitole. Pro přehlednost byly vyřazeny deklarace polí a proměnných.

Učení regulátoru je náchylné k tzv. přeučení, kde dochází ke kontinuálnímu nárůstu překmitu, což vede i k postupnému zvyšování celkové chyby. Z toho důvodu byla podmínka if, která při detekci kontinuálně se zhoršujícího průměru odchylky učení ukončí.



Obrázek 16: Průběh řízeného lineárního tlumeného oscilátoru s hloubkou 10 s využitím L-M

Lze si povšimnout, že dochází k poměrně vysokému překmitu. Tomu se dá zamezit buďto dřívějším ukončením učení nebo použitím optimalizačních metod, které budou dále představeny. V takovém případě lze do kriteriální funkce zahrnout penalizaci za vysoký překmit, například, schematicky zapsáno, ve tvaru Q = dot(e,e) + 50∗**max**(**abs**(yref−yn)). Jelikož se optimalizační algoritmy snaží Q minimalizovat, vytvoří takový přístup tendenci maximální překmit držet v přijatelných mezích. Na obr. 17 je vidět průběh systému s regulátorem. V tomto případě byl použit



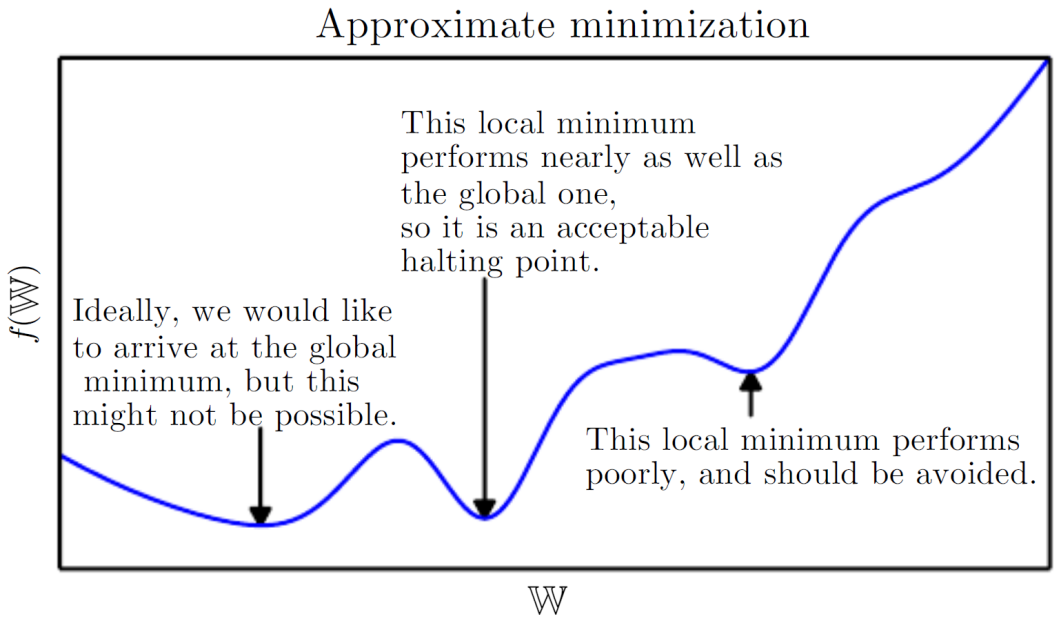
Obrázek 17: Regulovaný lineární tlumený oscilátor s optimalizací pomocí algoritmu Basin hopping

globální [9] optimalizační algoritmus Basin hopping[3][4], který bude lépe představen v následujících kapitolách. V kriteriální funkci byla zohledněna i velikost překmitu řízené soustavy.

6 Metody optimalizace pro HONU

Především u složitějších NU jako QNU a CNU při identifikaci nelineárních systémů naráží jednoduché optimalizační metody na problém hledání minima. Ilustrujme problematiku na 1-D příklad.[13] Je tedy důležité vybrat vhodnou metodu, která dokáže nalézt alespoň dobré lokální minimum, či, v ideálním případě, globální minimum, kriteriální funkce.

[9] Přesnější označení by bylo pseudo-globální. Princip algoritmu bude popsán v jemu věnované podkapitole.



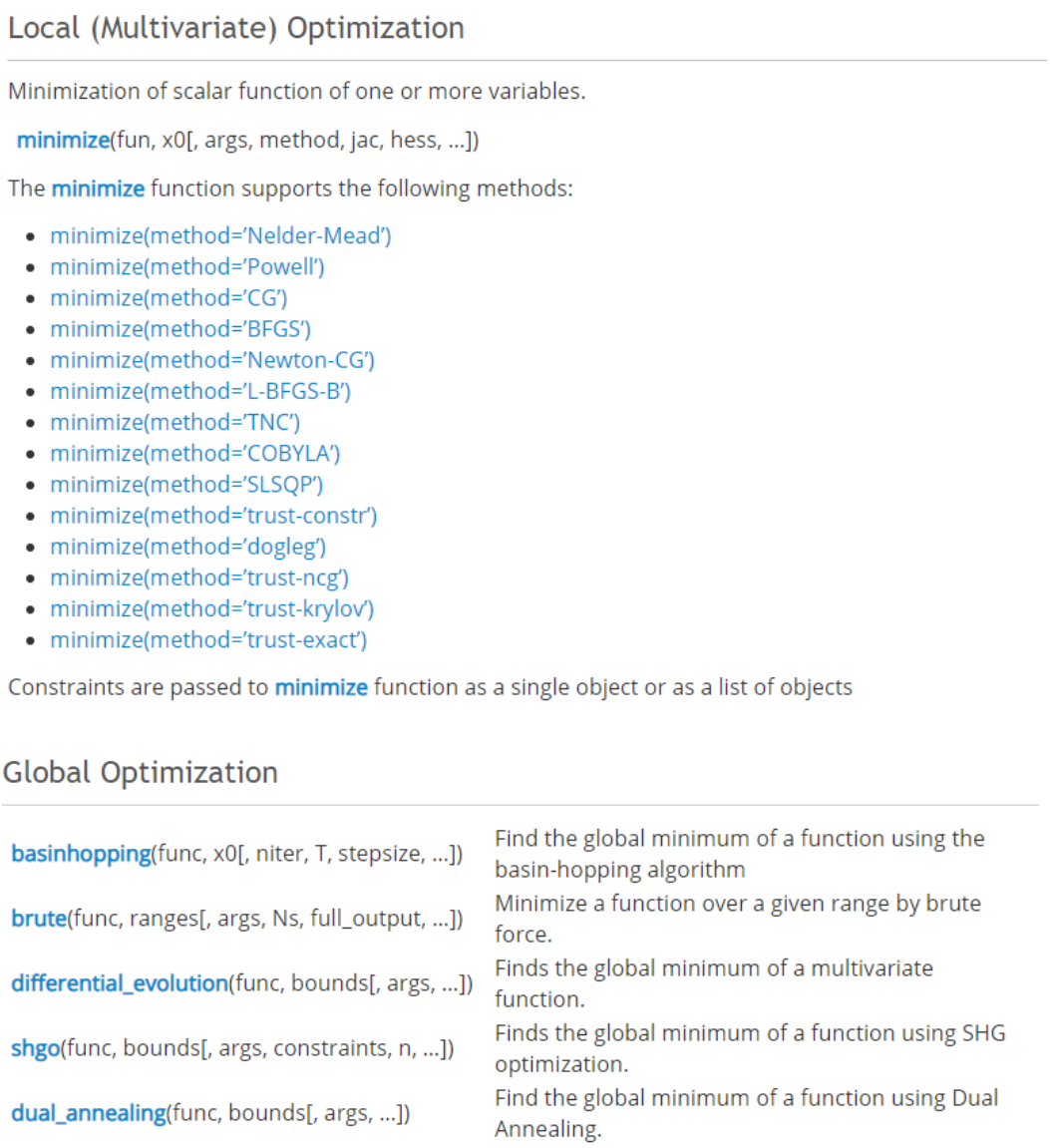
Obrázek 18: Ilustrace extrémů funkce [13]

Pokud bychom začínali v pravé polovině grafu a použili pouze jednoduchou gradientovou metodu, nedostaneme se do žádaného minima. Jak je na obr. 18 znázorněno, mohou se objevit i neglobální extrémy s podobnou hodnotou minima, mezi kterými již mohou mít i pokročilejší globální optimalizační metody jako jsou například genetické algoritmy problém rozlišovat. V případě malého rozdílu bude však v principu vhodné zlepšovat přesnost např. zvýšením n_y či n_u neuronu než hledáním o zlomek vhodnější minimum. Je však vhodné podotknout, že tento jev u některých aplikací omezí možnost přesného opakování výsledků, protože nemusí vždy být nalezeno stejné minimum.[3][4]

6.0.1 Knihovna SciPy.optimize pro Python

Existuje množství open-source projektů poskytujících implementaci řady optimalizačních algoritmů. Jednou z takových implementací je SciPy pro Python. Jedná se o rozsáhlou knihovnu poskytující řadu nástrojů pro vědecké výpočty.
Pro potřeby učení HONU využijeme balíček scipy.optimize[10], který poskytuje řadu optimalizačních funkcí. Jejich neúplný přehled je znázorněn níže.

[10]https://docs.scipy.org/doc/scipy/reference/optimize.html



Obrázek 19: Přehled některých metod dostupných z balíčku scipy.optimize [3]

Balíček obsahuje řadu optimalizačních metod, hledajících lokální, čí globální minimum kriteriální funkce.[3]
V případě použití funkce hledající lokální minimum je vhodné provést odhad koeficientů, například pomocí Levenberg-Marquardtova algoritmu. Samotný odhad se poté použije na začátku lokální optimalizace pomocí knihovny.
V případě hledání globálního minima zpravidla algoritmy nemají výchozí bod a funkci se jako argument poskytnou hranice intervalů, ve kterých se parametry

mohou vyskytovat. Obecně tedy volání optimalizační metody vypadá následovně.

Kód 13: Volání funkce minimize z knihovny SciPy

```
1 from scipy.optimize import minimize
2 bounds = [ (−1,1) for i in range(nw) ]
3 result=minimize(fQ, w, method=opt_method, bounds = bounds ,
4                                   options={'gtol': 1e−12, 'disp': True})
```

(1) fQ - kriteriální funkce fQ(w), která má argument vektor vah neuronu w a výstup je norma chyby $\|e\|_1$ nebo $e \cdot e = (\|e\|_2)^2$.

(2) w - vahový vektor, který optimalizujeme

(3) opt_method - zvolená optimalizační metoda, např. BFGS

(4) bounds - pole dvojic[11] určujících hranice jednotlivých parametrů ve vektoru w

(5) options - další volitelné parametry měnící nastavení jako podmínku normy gradientů pro ukončení výpočtu nebo průběžné zobrazování stavu výpočtu při iteracích

Globální optimalizační metody mají své vlastní dedikované funkce s podobnou syntaxí popsanou v dokumentaci SciPy.[3]

6.0.2 Broyden–Fletcher–Goldfarb–Shannonova metoda

Jedná se o kvazi-Newtonovskou metodu používanou ve strojovém učení, využívající aproximace Hessovy matice pomocí gradientů. Patří do rodiny *Hill-Climbing* algoritmů[12], které jsou schopny nalézt bod lokálního extrému, nikoli však globální extrém.[14]
Kromě základního algoritmu se používají také verze L-BFGS, kde L indikuje *Limited-memory*, tedy variantu využívající méně počítačové paměti, což může u rozsáhlých úloh hrát důležitou roli. A BFGS-B, resp. L-BFGS-B, kde B indikuje, že parametry jsou Bounded, tedy ohraničené intervalem, jak již bylo uvedeno v předchozí podkapitole.[3]

Kód 14: Implementace optimalizační funkce algoritmu L-BFGS-B z knihovny SciPy

```
1 # definice kriterialni funkce
2 def fQ(w):
```

[11] V Pythonu proměnná typu tuple
[12] česky - gradientní algoritmus

```
    yn = ones(N)*phiPP
    for k in range(1,N):
        colx[k, :] = assembleColX(yn,u,uPP,phiPP)
    yn[k] = dot(colx[k,:], w)
    e = (yn - yr)
    Q = dot(e,e)
return Q

# 1. LM - prvotni odhad
yn = ones(N)*phiPP
epochs = 3 ; mu = 0.1
for epoch in range(epochs):
    for k in range(1,N):
        colx[k,:] = assembleColX(u,uPP,phiPP)
    yn[:] = dot(colx, w)
 y e = (yr - yn)
    J = colx
    dw = dot(dot(inv(dot(J.T, J) + 1 / mu * I), J.T), e)
    w = w + dw

# 2. optimalizace parametru metodou L-BFGS-B
bounds = [ (-1,1) for i in range(nw) ]
result=minimize(fQ, w, method='L-BFGS-B', bounds = bounds ,
                                    options={'gtol': 1e-12, 'disp': True})
```

Funkčnost implementace metody byla ověřena na simulační úloze kvadraticky nelineárního systému

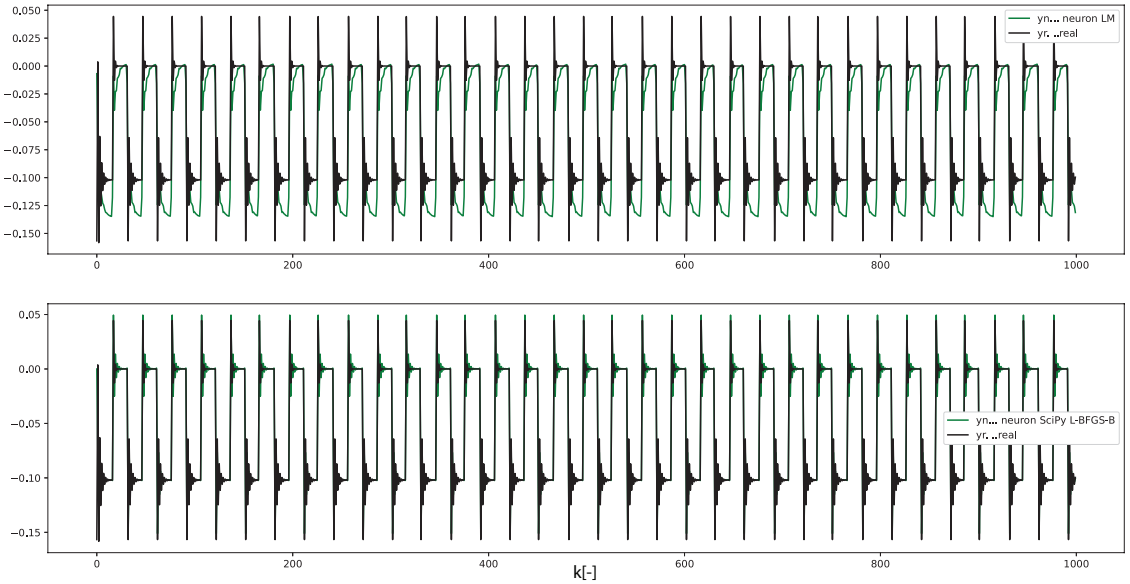
$$yr[k] = yr[k-1]^2 + (S_u \cdot u[k-1] - yr[k-1])\frac{dt}{\tau} + 0.1 \cdot u[k-1] \cdot yr[k-1] + 0.01 \cdot u[k-1]^2 \tag{54}$$

a lineárního oscilujicího systému

$$yr[k] = (u[k-1] + yr[k-1] \cdot (\frac{2m}{dt^2} - k_{param}) + m \cdot yr[k-2] \cdot (\frac{b}{2dt} - \frac{m}{dt^2})) \cdot \frac{1}{\frac{m}{dt^2} + \frac{b}{2dt}}. \tag{55}$$

i) Dynamická identifikace

Nejprve byla optimalizace vyzkoušena na případ dynamické identifikace systému, tedy případ, kde pro výpočet následujícího časového kroku jsou použity předchozí simulované hodnoty.



Obrázek 20: Simulace kvadraticky nelineárního systému - nahoře: systém odhadnutý pomocí L-M, dole: systém naučený pomocí L-BFGS-B

Tabulka 6: Srovnání identifikace pomocí LM a L-BFGS u DLNU

Metoda	LM-předučení	LM 50 epoch	L-BFGS
Celková chyba	4.292	0.449	0.467
Výpočetní čas	-	0.56s	18.93s

Pro tento systém se ukazuje L-M algoritmus podstatně efektivnější. Dosahující o něco vyšší přesnosti a především podstatně vyšší časové efektivity.
Tato vyšší efektivita L-M je především dána tím, že algoritmus jako takový zná jakobián J a naopak vyžaduje jeho naprogramování. BFGS takový požadavek nemá a gradientní vektory počítá numericky. Tedy pro výpočet gradientu podle prvku z vektoru $\mathbb{W}$, w_i, zavolá funkci fQ(w) a poté fQ(w_), pro w_ je prvek $w_i' = w_i + \Delta w_i$.[3] S rostoucím počtem prvků tedy algoritmus vyžaduje rychle zvyšující se počet volání funkce fQ, což proces zpomaluje. To, mimo jiné, vyžaduje důraz na co nejoptimálnější implementaci této funkce.
Při identifikaci nelineární DNU (DQNU,DCNU) vzniká problém s ukončením

smyčky optimalizačních metod. Je tedy vhodné zajistit, že proces bude dokončen v přiměřeném čase. To u funkcí z knihovny scipy.optimize jde například pomocí argumentu maxiter. V kódu implementace omezující počet iterací u BFGS algoritmu na **iter** = 100 vypadá následovně:

Kód 15: Kód volání minimalizační metody BFGS

```
1    result=minimize(fQ, w, method='BFGS' ,
2                    options={'gtol': 1e-5, 'disp': True, 'maxiter': 100})
```

ii) Statická identifikace

Dále byla optimalizace provedena pro statickou variantu identifikace. Tedy pro variantu, kde pro výpočet následujícího kroku používá systém hodnoty naměřené, resp. nasimulované na identifikovaném systému.

Problémy se ztrátou stability v tomto případě nenastaly a optimalizace probíhala v pořádku při použití SLNU i SQNU. Optimalizace byla vyzkoušena na několika modelech. Výstup v tabulce níže je pro model tlumeného kyvadla (3).

Tabulka 7: Srovnání identifikace kyvadla pomocí LM a L-BFGS u SQNU

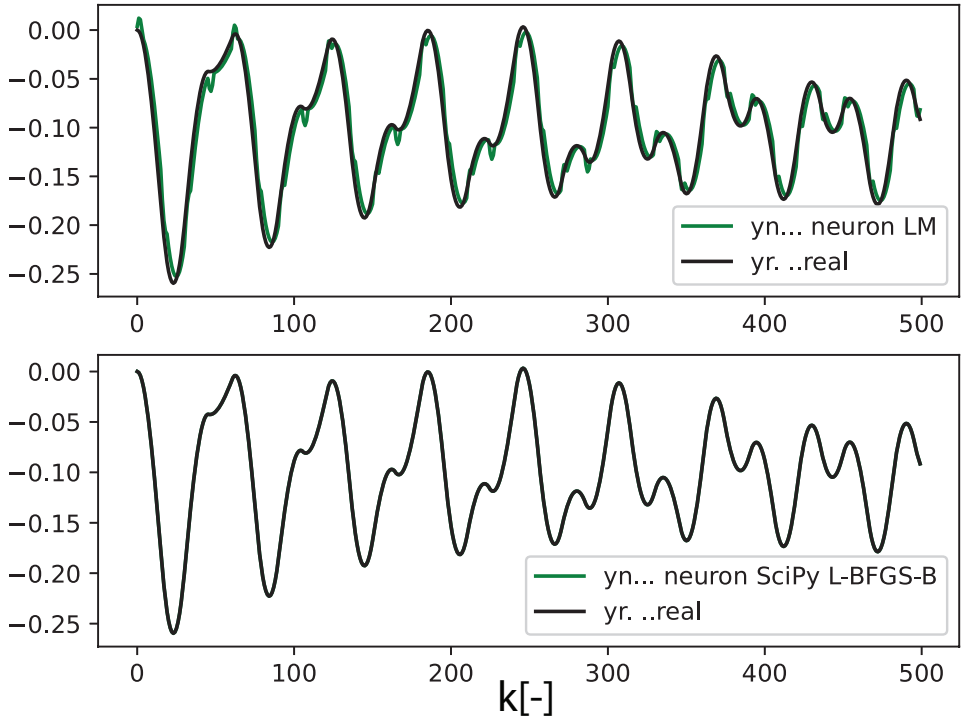
Metoda	LM-předučení	LM 350 [13]	L-BFGS 50	LM 350 + L-BFGS
Celková chyba	3.60	0.0342	0.0141	0.0327
Výpočetní čas	-	1.604s	1.385s	1.632s

V případě statické identifikace se vygenerovaná matice colx nemění se změnou optimalizovaného parametru $\mathbb{W}$, ev. $\mathbb{V}$, a algoritmus je velmi rychlý a přesný. Překonává L-M s 350 epochami.

Tabulka 8: Srovnání identifikace lineárního oscilujícího systému u LNU_4_4 [14]

Metoda	LM-předučení	LM 350	L-BFGS
Celková chyba	4.972	0.013	3.76e10-4
Výpočetní čas	-	0.055s	0.014s

[14] Připomeňme zavedené značení pro HONU. Obecně, u značení XNU_ny_nu, X indikuje řád HONU, typicky **L**inear, **Q**uadratic nebo **C**ubic. Hodnoty n_y a n_u určují hloubku z hlediska použitých předešlých časových kroků, jak bylo definováno v kapitole 3.



Obrázek 21: Statická identifikace lineárního oscilujícího systému

6.0.3 Nelder-Mead

Jedná se o optimalizační metodu, která využívá simplexový přístup.[3] Přístup byl vyzkoušen na několika typech systémů a porovnán s ostatními metodami.
Výhodou této metody je značná stabilita i při dynamickém učení. Nevýhodou je lokálnost optimalizace a zpravidla velmi dlouhý průběh.
Metoda se ukázala být pro danou problematiku méně vhodnou a nebude tedy dále rozebírána.

6.0.4 Basin hopping

Basin hopping je optimalizační algoritmus nejlépe popsatelný jako pseudo-globální algoritmus. K hledání minima kriteriální funkce využívá dva základní principy[3][4][14]:

(i) Lokální optimalizace v bodu pomocí dané metody (např. BFGS)

(ii) Pokus o úkrok[15] nějakým směrem v prostoru optimalizovaného vektoru

V této práci byla použita implementace algoritmu v knihovně SciPy dostupné pro Python.[3]

Kód 16: Použití algoritmu Basin Hopping z knihovny SciPy.optimize

```
from scipy.optimize import basinhopping
maxIter = 150
minimizer_kwargs = {"method": "BFGS"}
result = basinhopping(fQ, w, minimizer_kwargs=minimizer_kwargs,
                      niter=maxIter, callback = basinhopping_Callback)
w = result.x
```

Významy argumentů funkce basinhopping jsou stejné jako v již popsaných funkcích, nový argument minimizer_kwargs určuje, která lokální metoda bude použita pro lokální etapu optimalizace (i).
Možnost využití lokálního gradientního algoritmu se zdá být výhodné ve srovnání s negradientními globálními metodami, jelikož šance, že se algoritmus při úkroku trefí přímo do minima je velmi nízká, a tedy jsou řešení získané pomocí basin hopping obecně robustnější, jelikož metoda inherentně bude hledat řešení, která se přirozeně nachází v sedlovém bodě s dynamikou, která nějakým způsobem reprezentuje dynamiku systému. Ve srovnání, genetické algoritmy jsou sice schopny nalézt přesné řešení, které ale mnohdy odpovídá pouze přesným podmínkám při řešení optimalizační úlohy a i s malou změnou vstupu dojde k velkému nárůstu odchylky.

(1) $i \leftarrow 0$
(2) $X_i \leftarrow$ random initial point in variable space
(3) $Y_i \leftarrow$ LOCALSEARCH (X_i)
(4) **while** STOP not satisfied **do**
(5) $\quad X_{i+1} \leftarrow$ PERTURB (Y_i)
(6) $\quad Y_{i+1} \leftarrow$ LOCALSEARCH (X_{i+1})
(7) $\quad$ **if** $f(Y_{i+1}) < f(Y_i)$ **then**
(8) $\quad\quad i \leftarrow i + 1$

Obrázek 22: Pseudokódový zápis optimalizačního diagramu Basin Hopping[4]

[15] V angl. literatuře se používá termín *perturbance*.

source https://www.hindawi.com/journals/aai/2012/674832/

6.0.5 Genetické algoritmy

Genetické algoritmy jsou rodina algoritmů založených na analogii s přirozenou selekcí v přírodě. Algoritmus vytvoří řadu jedinců, kteří jsou v našem případě náhodné volby vektoru $\mathbb{W}$ nebo $\mathbb{V}$. Dle typu algoritmu se, zjednodušeně řečeno, tyto vektory kombinují, analogicky k přenosu genů z rodičů na potomka, a dávají vzniknout novým jedincům. Samotná pravděpodobnost předání vlastností rodiče na některého z jedinců další generace je řízena *fitness function*, tedy funkcí, která poskytuje nějakou metriku vhodnosti jedince. V našem případě se jedná identicky o fQ(w), resp. fQ(v) z předchozí kapitoly. Funkce má argument vahového vektoru, který mění a postupně optimalizuje algoritmus, počítá průběh chování systému a vrací celkovou chybu, typicky součet absolutních hodnot chyby ve všech krocích, nebo sumu jejich kvadrátů.[3][14]
Hlavně při použití u dynamického učení dochází obzvláště u této rodiny algoritmů k výrazným problémům již při identifikaci, kde během výpočtu dojde ke ztrátě stability a výpočet je ukončen s chybou způsobenou přetečením některých z proměnných.
Pro samotnou implementaci byla opět využita knihovna Scipy[3], konkrétně její balíček scipy.optimize, který obsahuje funkci differential_evolution. V Pythonu byla funkce zavolána následovně:

Kód 17: Syntaxe zavolání funkce genetického algoritmu

```
1    result=differential_evolution(fQ,bounds=bounds, maxiter = 500 )
```

Metoda byla úspěšně vykoušena na dynamické identifikaci kvadraticky nelineárního systému (53). Pokusy identifikovat kyvadlo byly méně neúspěšné. Podařilo se najít $\mathbb{W}$, které dobře odpovídalo konkrétnímu vstupu, ale řešení bylo velmi nerobustní pro i mírně odchýlené hodnoty počátečních podmínek, resp. vstupů.

Tabulka 9: Srovnání identifikace pomocí LM a genetického algoritmu u systému (53)

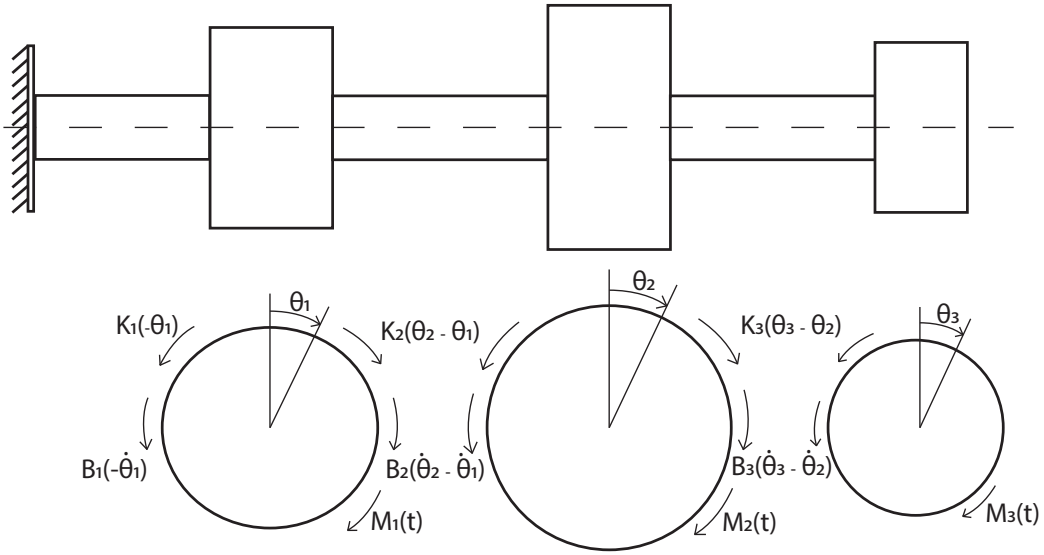
Metoda	LM	differential evolution
Celková chyba	0.4496	0.4491

Je tedy patrné, že v některých případech může i genetický algoritmus dospět k velmi dobrému řešení, ale z důvodu problémů se stabilitou a především velmi vysoké časové náročnosti se nejeví jako vhodný způsob učení. Jeho předností

zůstává fakt, že se jedná o globální algoritmus. Metody založené na výpočtu gradientu nebo výpočtu či odhadu Hessovy matice jsou obzvláště u vysoce nelineárních problémů odkázané na vhodný prvotní odhadu optimalizovaného vektoru.

7 Model torzně kmitavé soustavy s nelineárním tlumením

Dalším předmětem rozboru v této práci je řetězec torzně kmitajících hmot s nelineárním tlumením.
Pro analýzu byl zvolen systém znázorněný na diagramu níže.



Obrázek 23: Diagram řetězce tří torzně kmitajících hmot

Na diagramu jsou vyznačené momenty působící na jednotlivé elementy, pro i-tý element zřejmě platí

$$I_i\ddot{\phi}_i = \sum_k M_k, \tag{56}$$

kde I_i je moment setrvačnosti i-té hmoty, ϕ_i je úhel natočení i-té hmoty a M_k jsou jednotlivé momenty působící na element. Zřejmě můžeme, s využitím diagramu v

obrázku 12 obecně pro m hmot zapsat pohybové rovnice jako

$$\begin{aligned} I_1\ddot{\phi}_1 &= K_1(-\phi_1) + B_1(-\dot{\phi}_1) + K_2(\phi_2 - \phi_1) + B_2(\dot{\phi}_2 - \dot{\phi}_1) \\ I_2\ddot{\phi}_2 &= -K_2(\phi_2 - \phi_1) + K_3(\phi_3 - \phi_2) - B_2(\dot{\phi}_2 - \dot{\phi}_1) + B_3(\dot{\phi}_3 - \dot{\phi}_2) \\ &\vdots \\ I_m\ddot{\phi}_m &= -K_m(\phi_m - \phi_{m-1}) - B_m(\dot{\phi}_m - \dot{\phi}_{m-1}). \end{aligned} \tag{57}$$

Takový popis je pouze aproximací reálného mechanického chování soustav. Pro další přiblížení reálnému světu do popisu dále přidáme další zdroje útlumu f_i.

$$\begin{aligned} I_1\ddot{\phi}_1 &= K_1(-\phi_1) + B_1(-\dot{\phi}_1) + K_2(\phi_2 - \phi_1) + B_2(\dot{\phi}_2 - \dot{\phi}_1) - f_1 \\ I_2\ddot{\phi}_2 &= -K_2(\phi_2 - \phi_1) + K_3(\phi_3 - \phi_2) - B_2(\dot{\phi}_2 - \dot{\phi}_1) + B_3(\dot{\phi}_3 - \dot{\phi}_2) - f_2 \\ &\vdots \\ I_m\ddot{\phi}_m &= -K_m(\phi_m - \phi_{m-1}) - B_m(\dot{\phi}_m - \dot{\phi}_{m-1}) - f_m \end{aligned} \tag{58}$$

Obecně pisujeme řetězec m torzně kmitajících hmot. Pro $f_i = 0, \forall i \in N$ se jedná o standartní problém lineárních oscilátorů, který v této práci již byl popsán pro případ SISO a relativně jednoduše identifikován i řízen v 1DOF případě.
Byly zavedeny dva typy nelineárního tlumení. První typ, označovaný f_i^I je

$$f_i^I = f_i^I(a_i, b_i, \dot{\phi}_i) = a_i \cdot (\frac{2}{1 + e^{-b_i \cdot \dot{\phi}_i}} - 1). \tag{59}$$

Funkce $f_i^I(a_i, b_i, \dot{\phi}_i)$[16] nazveme třením typu 1. Toto tření odpovídá tření kuličkových ložisek.
Zavedeme také druhou funkci $f_i^{II}(a_i, b_i, \Delta\dot{\phi}_i)$[17]

$$f_i^{II} = f_i^{II}(\alpha_i, \beta_i, \Delta\dot{\phi}_i) = \alpha_i \cdot (\frac{2}{1 + e^{-\beta_i \cdot \Delta\dot{\phi}_i}} - 1). \tag{60}$$

Tlumení typu 2 odpovídá vazkému tření ve hřídeli. Je tedy přítomno v pohybových rovnicích obou sousedních hmot, narozdíl od tření ložisek.
A pro přidané nelineární tlumení použité v soustavě rovic [46] tedy platí [18]

$$f_i = f_i^I + f_i^{II} - f_{i+1}^{II}. \tag{61}$$

[16] Povšimněme si, že se jedná o rovnici hladké sigmoidy.
[17] Opět má funkce tvar rovnice hladké sigmoidy.
[18] Poslední hmota v této formulaci obsahuje fiktivní člen $f_{n+1}^{II} = 0dw$

7.1 Příklad torzně kmitavé soustavy se 3 stupni volnosti a nelineárním tlumením

Vezměme soustavu 46 a přepišme ji pro soustavu tří torzně kmitavých závaží.

$$\begin{aligned} I_1\ddot{\phi}_1 &= K_1(-\phi_1) + B_1(-\dot{\phi}_2) + K_2(\phi_2 - \phi_1) + B_2(\dot{\phi}_2 - \dot{\phi}_1) - f_1(\dot{\phi}_1, \Delta\dot{\phi}_1) \\ I_2\ddot{\phi}_2 &= -K_2(\phi_2 - \phi_1) + K_3(\phi_3 - \phi_2) - B_2(\dot{\phi}_2 - \dot{\phi}_1) + B_3(\dot{\phi}_3 - \dot{\phi}_2) - f_2(\dot{\phi}_2, \Delta\dot{\phi}_2) \\ I_3\ddot{\phi}_3 &= -K_3(\phi_3 - \phi_2) - B_3(\dot{\phi}_3 - \dot{\phi}_2) - f_3(\dot{\phi}_3, \Delta\dot{\phi}_3) \end{aligned} \tag{62}$$

V souladu s formou soustavy rovnic se kterou jsme pracovali například v předmětu Kmitání mechanických soustav můžeme napsat

$$\begin{aligned} &\begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \begin{bmatrix} \ddot{\phi}_1 \\ \ddot{\phi}_2 \\ \ddot{\phi}_3 \end{bmatrix} + \begin{bmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} + \\ &\qquad + \begin{bmatrix} b_1 + b_2 & -b_2 & 0 \\ -b_2 & b_1 + b_2 & -b_3 \\ 0 & -b_3 & b_3 \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{bmatrix} + \begin{bmatrix} f_1(\dot{\phi}_1, \Delta\dot{\phi}_1) \\ f_2(\dot{\phi}_2, \Delta\dot{\phi}_2) \\ f_3(\dot{\phi}_3, \Delta\dot{\phi}_3) \end{bmatrix} = \begin{bmatrix} M_1(t) \\ M_2(t) \\ M_3(t) \end{bmatrix}. \end{aligned} \tag{63}$$

Tento maticový zápis převedeme do stavového popisu:

$$\begin{aligned} \dot{\phi}_1 &= \omega_1 \\ I_1\dot{\omega}_1 &= K_1(\phi_2 - \phi_1) + B_1(\omega_2 - \omega_1) - f_1(\omega_1, \Delta\omega_1) + M_1(t) \\ \dot{\phi}_2 &= \omega_2 \\ I_2\dot{\omega}_2 &= -K_1(\phi_2 - \phi_1) + K_2(\phi_3 - \phi_2) - B_1(\omega_2 - \omega_1) + B_2(\omega_3 - \omega_2) \\ &\quad -f_2(\omega_2, \Delta\omega_2) + M_2(t) \\ \dot{\phi}_3 &= \omega_3 \\ I_3\dot{\omega}_3 &= -K_2(\phi_3 - \phi_2) - B_2(\omega_3 - \omega_2) - f_3(\omega_3, \Delta\omega_3) + M_3(t) \end{aligned} \tag{64}$$

Tento systém byl implementován v Pythonu v souboru simTorznisyst.py. Tento baliček obsahuje funkce

(i) runSim3(u,x0_3dof,dt,T) - simuluje průběh systému se 3 stupni volnosti pro interval $t \in (0, T)$ s časovým krokem dt.

(ii) runSim2(u,x0_2dof,dt,T) simuluje průběh systému se 2 stupni volnosti pro interval $t \in (0, T)$ s časovým krokem dt.

(iii) runSim1(u,x0_1dof,dt,T) simuluje průběh systému se 1 stupněm volnosti pro interval $t \in (0, T)$ s časovým krokem dt.

(iv) ode_3DOF_oscillator(state,t, u) jedná se o implementaci soustavy diferenciálních rovnic se 3 stupni volnosti volanou solverem diferenciálních rovnic v knihovně SciPy.

(v) ode_2DOF_oscillator(state,t, u) jedná se o implementaci soustavy diferenciálních rovnic se 2 stupni volnosti volanou solverem diferenciálních rovnic v knihovně SciPy.

(vi) ode_1DOF_oscillator(state,t, u) jedná se o implementaci soustavy diferenciálních rovnic se 1 stupněm volnosti volanou solverem diferenciálních rovnic v knihovně SciPy.

(vii) plotOutput(t,selectPlot,u,*args) zobrazí grafy dat v argumentu *args. Struktura selectPlot umožňuje změnit některá základní nastavení - zobrazit také průběhy úhlových rychlostí, zobrazit pouze průběh úhlu poslední hmoty, aj.

Příkladem implementace rovnic systému se třemi stupni volnosti je funkce ode_3DOF_oscillator napsaná tak, aby byla řešitelná s pomocí funkce odeint z balíčku scipy.integrate.

Kód 18: Funkce ode_3DOF_oscillator z balíčku simTorznisyst.py

```
1   def ode_3DOF_oscillator(state,t, u):
2   phi, omega = state[:3], state[3:]
3   M = u
4   K1, K2, K3, B1, B2, B3, a1, a2, a3, b1, b2, b3, \
5   alpha1, alpha2, alpha3, beta1, beta2, beta3, I1, I2, I3 = oscillatorParams()
6   if −b1 ∗ omega[0]>100:
7       f1_I = −a1
8   elif −b1 ∗ omega[0]<−100:
9       f1_I = a1
10  else:
11      f1_I = a1 ∗ (2 / (1 + np.exp(−b1 ∗ omega[0])) − 1)
12  if −b2 ∗ omega[1] > 100:
13      f2_I = −a2
14  elif −b1 ∗ omega[1] < −100:
15      f2_I = a2
16  else:
17      f2_I = a2 ∗ (2 / (1 + np.exp(−b2 ∗ omega[1])) − 1)
18  if −b3 ∗ omega[2] > 100:
19      f3_I = −a3
20  elif −b1 ∗ omega[2] < −100:
21      f3_I = a3
22  else:
23      f3_I = a3 ∗ (2 / (1 + np.exp(−b3 ∗ omega[2])) − 1)
24
25  if −beta1 ∗ (omega[0])>100:
26      f1_II = −alpha1
```

```
    elif −beta1 ∗ (omega[0])<−100:
        f1_II = alpha1
    else:
        f1_II = alpha1 ∗ (2 / (1 + np.exp(−b1 ∗ (omega[0]))) − 1)
    if −beta2 ∗ (omega[1]−omega[0])>100:
        f2_II = −alpha2
    elif −beta2 ∗ (omega[1]−omega[0])<−100:
        f2_II = alpha2
    else:
        f2_II = alpha2 ∗ (2 / (1 + np.exp(−beta2 ∗ (omega[1]−omega[0]))) − 1)
    if −beta3 ∗ (omega[2] − omega[1]) > 100:
        f3_II = −alpha3
    elif −beta3 ∗ (omega[2] − omega[1]) < −100:
        f3_II = alpha3
    else:
        f3_II = alpha3 ∗ (2 / (1 + np.exp(−beta3 ∗ (omega[2] − omega[1]))) − 1)

    dw1dt = (K1 ∗ (− phi[0]) + B1∗(− omega[0]) + K2 ∗ (phi[1] − phi[0]) +
        B2 ∗ (omega[1] − omega[0]) − f1_I− f1_II + f2_II+ M[0])/I1
    dw2dt = (−K2 ∗ (phi[1] − phi[0]) + K3 ∗ (phi[2] − phi[1]) −
        B2 ∗ (omega[1] − omega[0]) + B3 ∗ (omega[2] − omega[1]) − f2_I − f2_II +
        f3_II + M[1]) / I2
    dw3dt = (−K3 ∗ (phi[2] − phi[1]) − B3 ∗ (omega[2] − omega[1]) − f3_I − f3_II +
        M[2]) / I3
    return [omega[0],omega[1],omega[2],dw1dt,dw2dt,dw3dt]
```

Tato funkce se poté volá ve funkci runSim3, která používá k řešení knihovnu SciPy, ale lze ji použít i při dalších operacích jako MRAC, kde je nutné jen vypočítat další krok.

Kód 19: Funkce runSim3 z balíčku simTorznisyst.py

```
def runSim3(u,x0_3dof,dt,T):
    t=np.arange(0,T,dt) ; N=len(t)
    DOF = 3
    x0_3dof = x0_3dof[0]
    M = np.zeros((N,3))
    M[:,0],M[:,1] = np.zeros(N), np.zeros(N)
    M[:,2] = u
    y1 = np.zeros(N)
    y2 = np.zeros(N)
    y3 = np.zeros(N)
    for i in range(0, N ):
        if i==N−1 or i==N:
            tt = [t[i], t[i]] # [t1 t2]
        else:
            tt = [t[i], t[i+1]] # [t1 t2]
        x=odeint(ode_3DOF_oscillator,x0_3dof,tt,args=(M[i,:],))
        y1[i] = x[1, 0]
        y2[i] = x[1, 1]
```

```
        y3[i] = x[1, 2]
        w1[i] = x[1, 3]
        w2[i] = x[1, 4]
        w2[i] = x[1, 5]
        x0_3dof = x[1, :] # jako nove poc. podm pro dalsi integraci
    return t,y1,y2,y3,w1,w2,w3
```

Samotný balíček je v práci importován **import** simTorzniSyst1DOF as sim a dále používán, např. t, y1, y2, yr = sim.runSim3(u2,x0 , dt, T).
V této práci byly použity primárny následující hodnoty

Tabulka 10: Hodnoty konstant první konfigurace systému

Hmota	①	②	③
I $[\frac{kg}{m^2}]$	0.0174	0.160	0.160
K $[\frac{kg}{s}]$	132.5	14.24	21.85
B [-]	$\frac{K_1}{1000}$	$\frac{K_2}{1000}$	$\frac{K_3}{1000}$
a [-]	0.0001	0.0001	0.0001
b [-]	1000	1000	1000
α [-]	0.0001	0.0001	0.0001
β [-]	1000	1000	1000

Použité časové kroky v simulaci byly konstantní a jejich hodnoty byly:

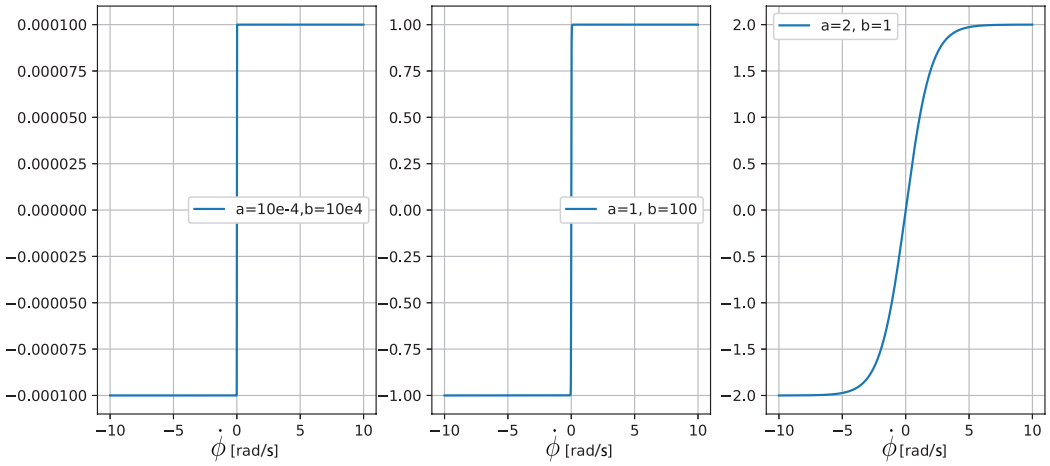
(i) dt = 0.1s pro úvodní studii 1DOF kmitání

(ii) dt = 0.02s pro 2DOF, 3DOF a další 1DOF

Délky simulace byly zpravidla 40s a 80s. U některých úvodních simulací pouze 10s, či 20s.
Jednou ze zajímavých možností adaptivní identifikace a řízení je možnost přizpůsobení se změně některých vlastností systému - tedy například postupnému posuvu hodnot konstant tlumení. To je ovšem pokročilejší problematika, která je bohužel nad časové možnosti tohoto rozboru s časovou dotací Diplomové práce.

7.2 Problematika simulace nelineárně tlumených systémů pomocí HONU

V této práci již bylo ukázáno, že lineární tlumené oscilátory lze velmi přesně identifikovat a simulovat pomocí HONU. Také bylo ukázáno, že takový systém lze řídit. Byla ukázána i možnost alespoň přibližné simulace systému s nelineritou sinového typu a lineárním tlumením. Dále si tato práce klade za cíl rozšířit tuto analýzu na nelineárně tlumený systém, který obsahuje nelinerity typu $\frac{a_i}{1+e^{-b_i \cdot \dot{\phi}_i}}$.

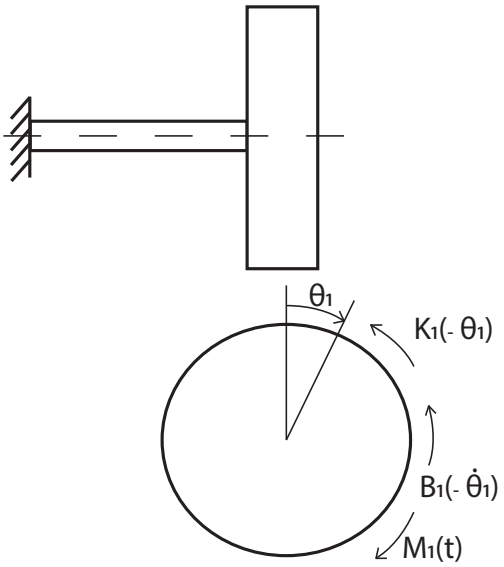


Obrázek 24: Průběh funkcí nelineárního tlumení v závislosti na parametrech a, b.

Hodnoty a a b nejprve zvolené odpovídají řádově průběhům i) a ii) zleva. Povaha tlumení je tedy spíše skoková.
V první části této kapitoly se tedy zaměříme na tuto variantu. takový postup je vhodný především vzhledem k tomu, že i v této variantě je úloha řízení 2DOF a především 3DOF systému náročná. Na závěr této práce budou poté zvoleny koeficienty, u kterých se sigmoidní nelinerita projeví více a bude proveden pokus o identifikaci a řízení takového nelineárního systému.

7.3 Identifikace nelineárně tlumeného 1DOF systému

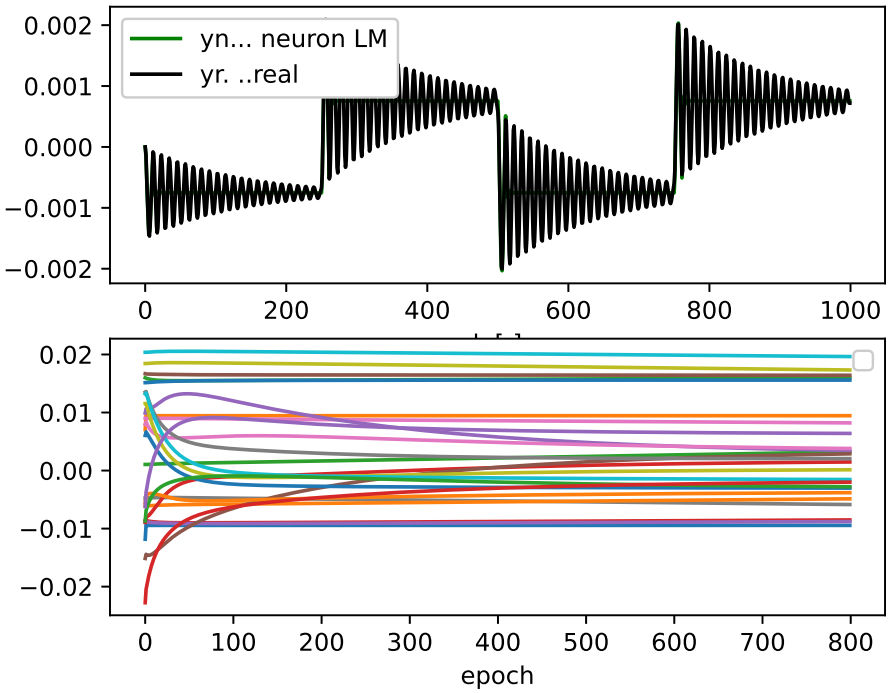
Vzhledem k silně nelineární povaze systému torzně kmitajících hmotiček byla nejprve řešena úloha identifikace a řízení torzně kmitajícího systému s 1DOF.



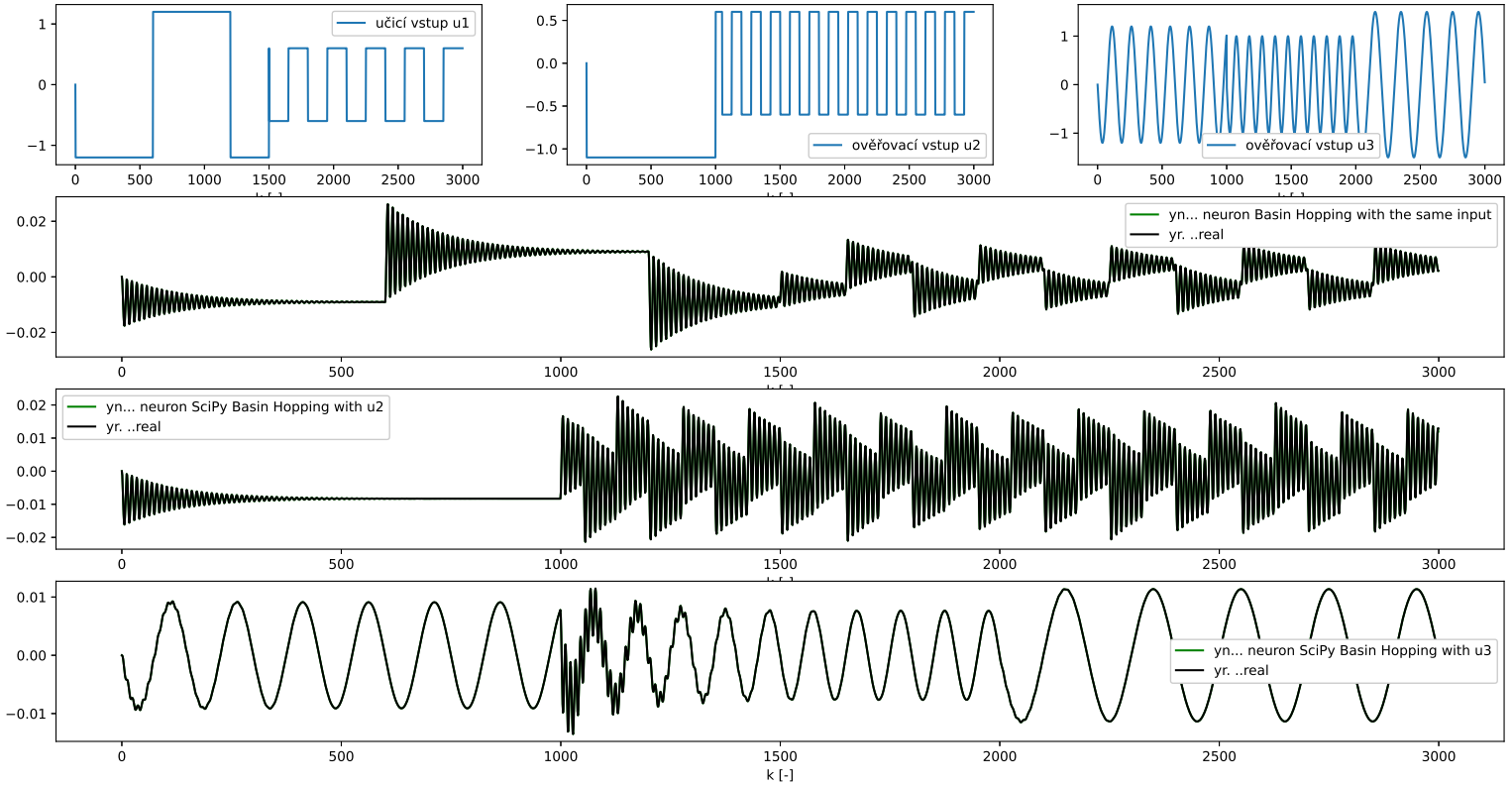
Obrázek 25: Schéma 1DOF oscilátoru

Tabulka 11: Chyby HONU při simulaci 1DOF tlumeného oscilátoru

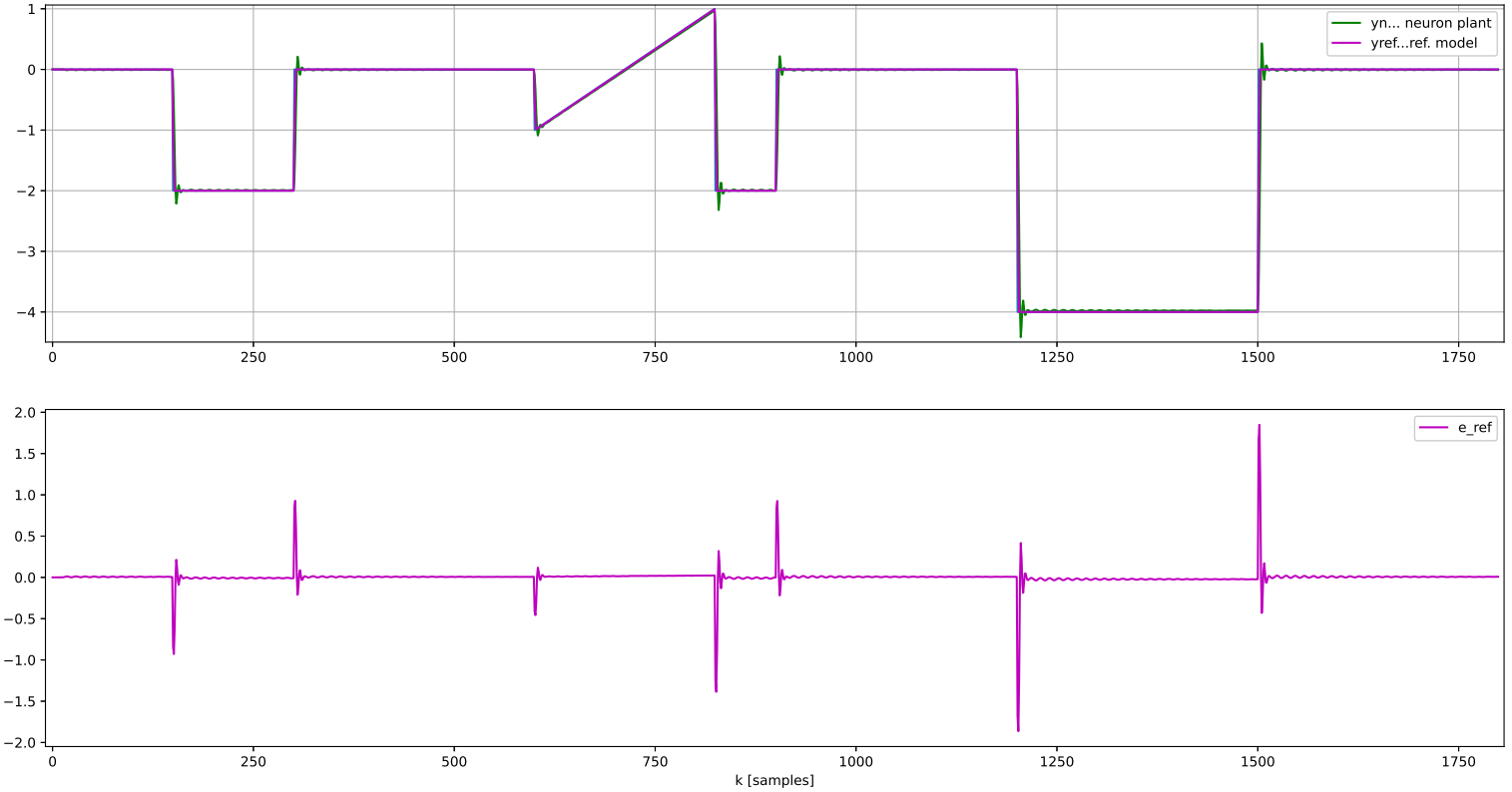
LNU_8_8	u_1	u_2	u_3
MAE	2.97e-06	3.76e-04	4.36e-05
qErr	3.98e-08	4.2e-04	1.04e-05
maxDiff	1.52e-05	9.4e-04	2.67e-04
LNU_12_12	u_1	u_2	u_3
MAE	1.08e-05	2.32e-05	4.69e-06
qErr	6.34e-06	3.27e-06	3.10e-07
maxDiff	4.95e-05	6.38e-05	3.81e-05
LNU_20_20	u_1	u_2	u_3
MAE	1.25e-05	2.96e-05	4.36e-06
qErr	6.68e-06	4.83e-06	2.73e-07
maxDiff	5.69e-05	8.14e-05	3.52e-05



Obrázek 26: Předtrénování 1DOF systému pomocí L-M algoritmu



Obrázek 27: HONU natrénované k predikci nelineárně tlumeného kyvadla



Obrázek 28: Řízený 1DOF nelineárně tlumený oscilátor s hloubkou 19

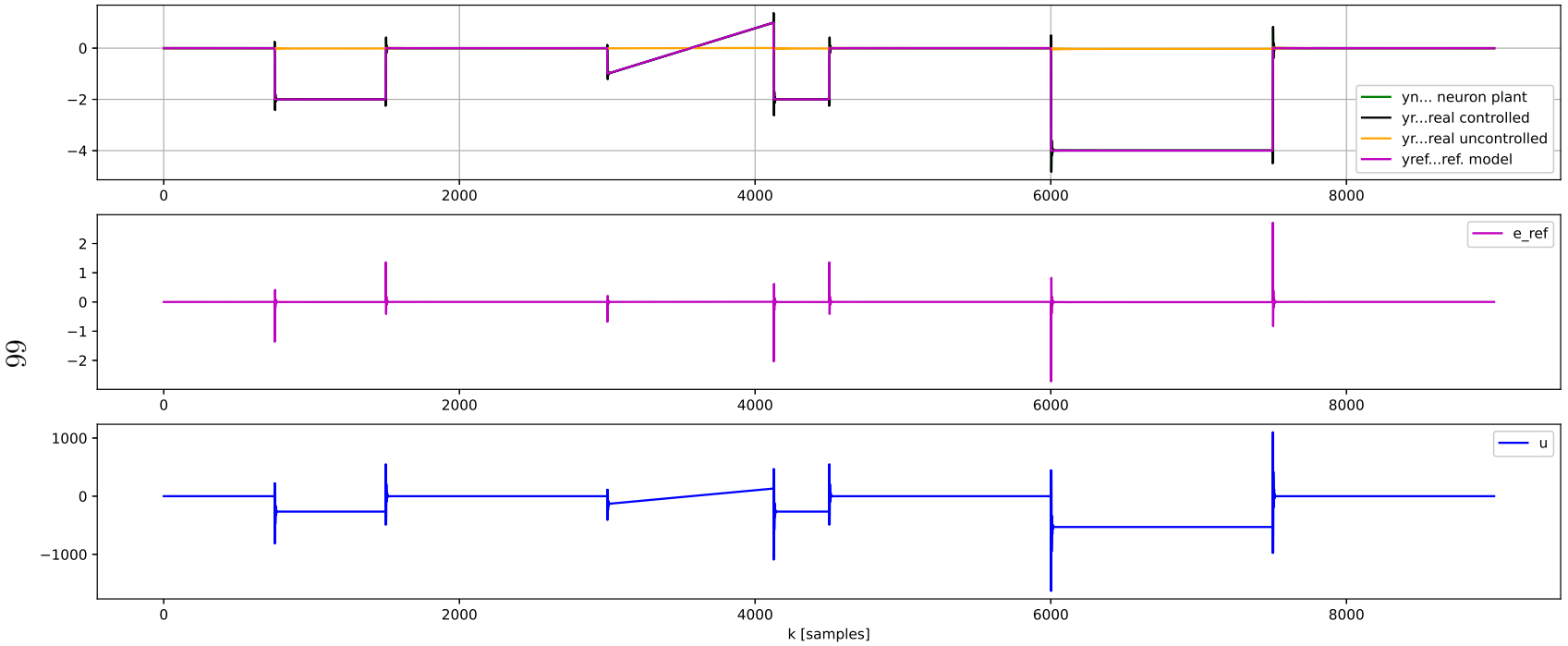
Z obrázku 27 a tabulky 11 je patrné, že identifikace proběhla pro testovací signály úspěšně.
Z obrázku 28 je zřejmé, že i řízení proběhlo úspěšně - otázkou je, zda i reálný, resp. zde simulovaný, model je stále dobře aproximován neuronem. V tomto případě bylo sice řízení neuronového výstupu přesné, ale ukázalo se, že nebylo dostatečně postihnuto kmitání systému - resp. bylo pravděpodobně v modelu přetlumeno - a výsledek byl značně rozkmitaný [19].
Po dokončení učení byla do smyčky implementována simulace reálného systému

Kód 20: Funkce runSim3 z balíčku simTorznisyst.py

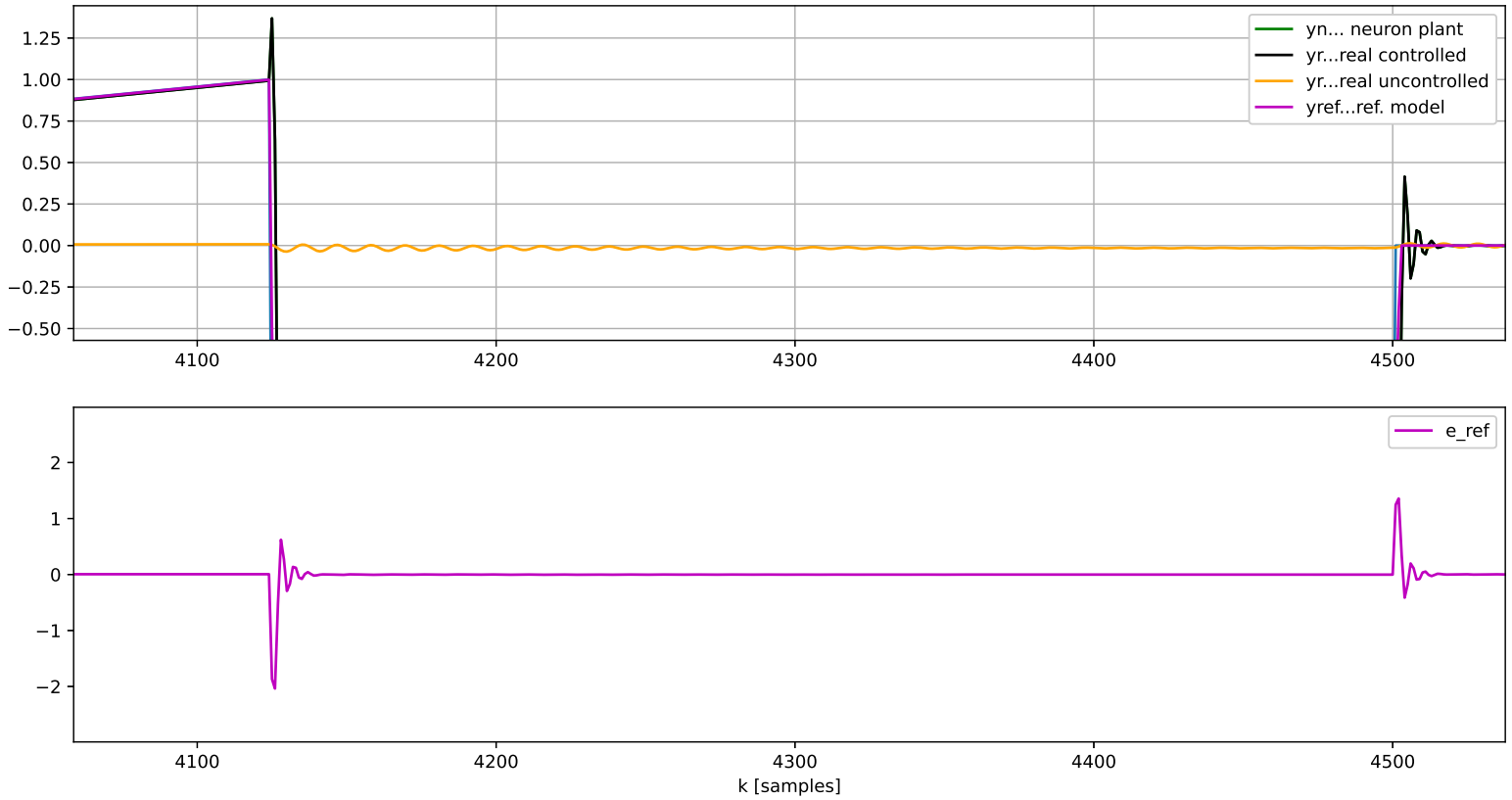
```
1 if k == N − 1 or k == N:
2     tt = [t[k], t[k]] # [t1 t2]
3 else:
4     tt = [t[k], t[k + 1]] # [t1 t2]
5 intStep = odeint(sim.ode_1DOF_oscillator, x0_1dof, tt, args=(u[k],))
6 yr[k] = intStep[1, 0]
7 w1[k] = x[1, 1]
8 x0_1dof = intStep[1, :] # jako nove poc. podm pro dalsi integraci
```

Toto je lépe pozorovatelné na detailnějším výřezu níže: Tento regulátor byl dále ověřen na jiném než trénovacím průběhu hodnoty žádané veličin d, reps. y_ref, jak je znázorněno na obrázku 31. Regulátor si udržuje své vlastnosti, tj. rychlá reakce na skokovou změnu y_ref, ale také nežádoucí překmit.
V grafech jsou také vykresleny neregulované průběhy, tj. průběhy, kde $u[k] = d[k]$. Jak je vidět, bez regulátoru je průběh nežádoucí. Také bylo provedeno zesílení, např. $u[k] = 10d[k]$, to vedlo k velkým nežádoucím oscilacím systému.

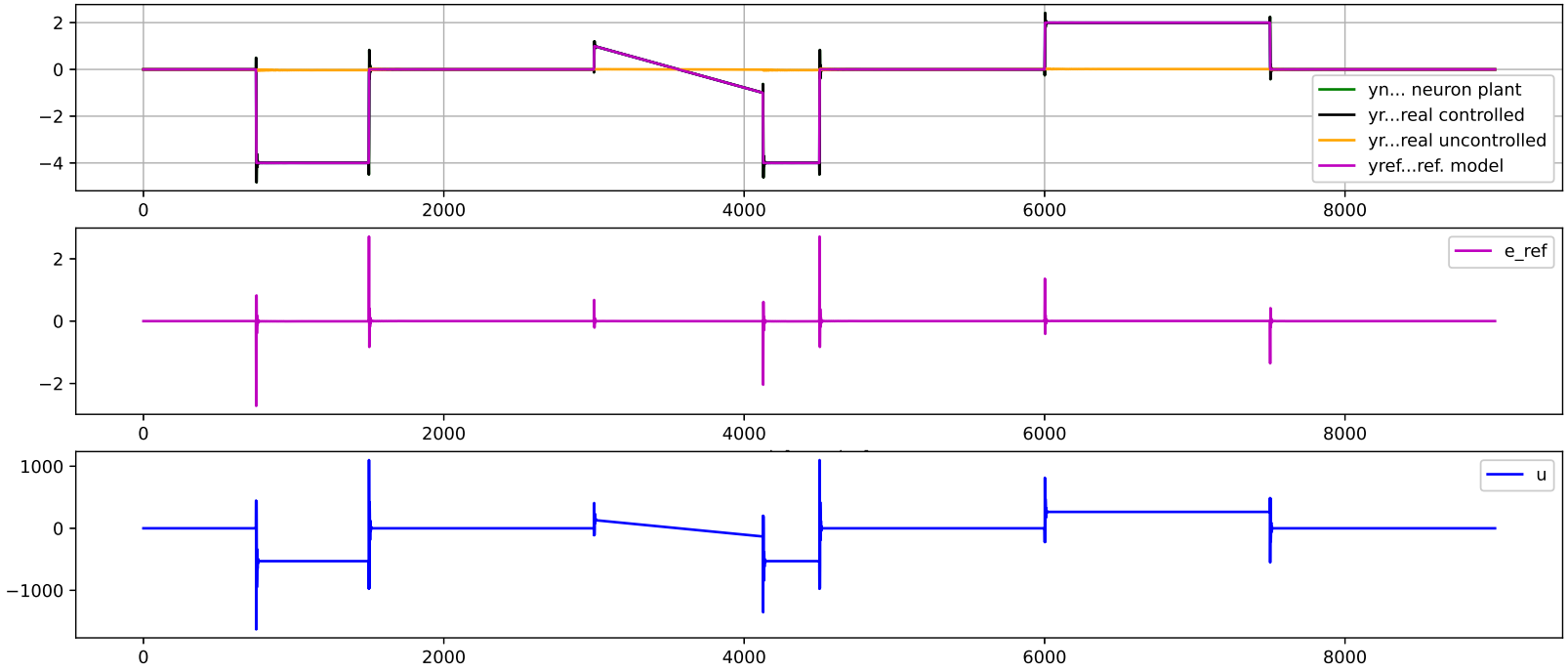
[19] Vizualizace není uváděna



Obrázek 29: Řízený 1DOF nelineárně tlumený oscilátor s y_r a značným překmitem a hloubou neuroregulátoru nvx = 19



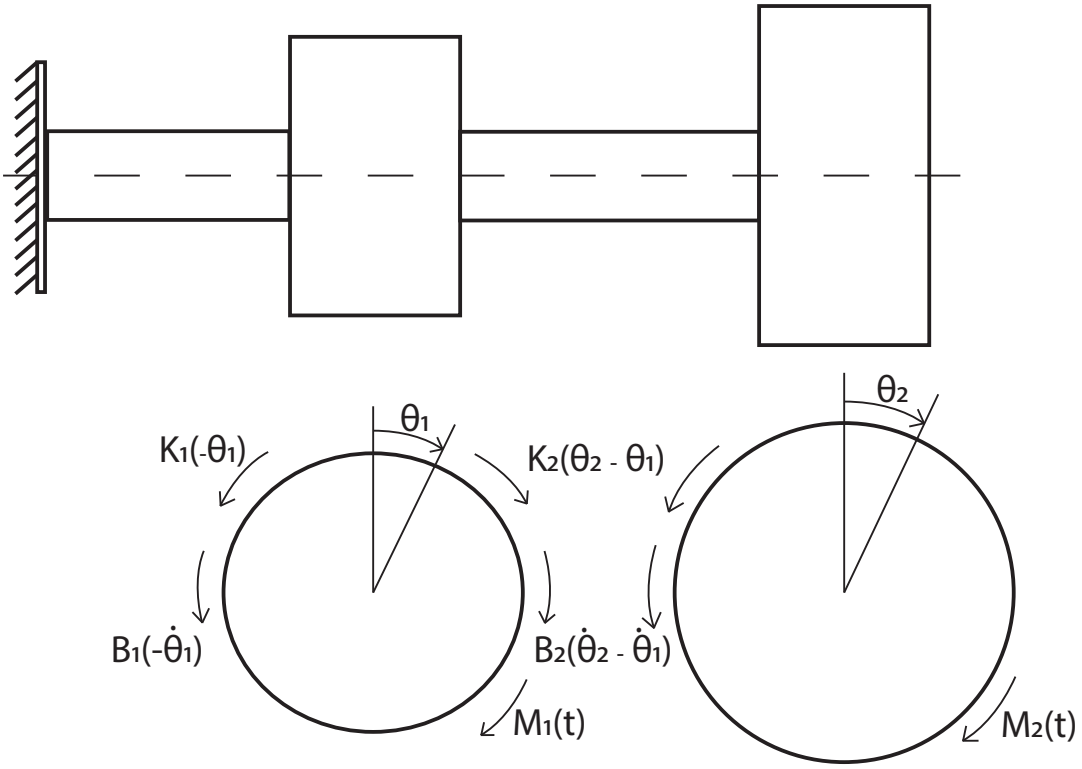
Obrázek 30: Detail: Řízený 1DOF nelineárně tlumený oscilátor s mírně překmitávajícím y_r



Obrázek 31: Řízený 1DOF nelineárně tlumený oscilátor s y_r pro testovací průběh veličiny d s hloubou neuroregulátoru nvx = 19

7.4 Identifikace a řízení nelineárně tlumeného 2DOF systému

Dále byla provedena identifikace 2DOF nelineárně tlumeného systému.



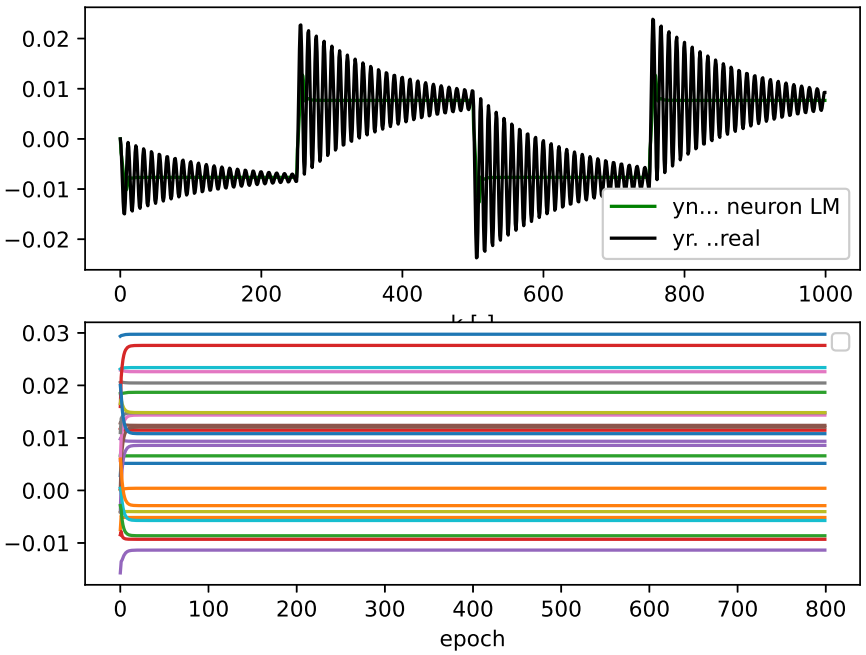
Obrázek 32: Diagram řetězce dvou torzně kmitajících hmot

Sestavené rovnice jsou zcela analogické ve srovnání s 3DOF řetězcem a nejsou tedy znovu opakovány.
Několika počátečními výpočty bylo ověřeno, že LNU, tedy HONU řádu $r = 1$, je dostačující k dobrému popisu dynamiky tohoto systému. Bylo tedy provedeno několik identifikací pro LNU_8_8, LNU_12_12 a LNU_20_20.
Kvantitativní porovnání bylo zajištěno pomocí tří metrik[20]:

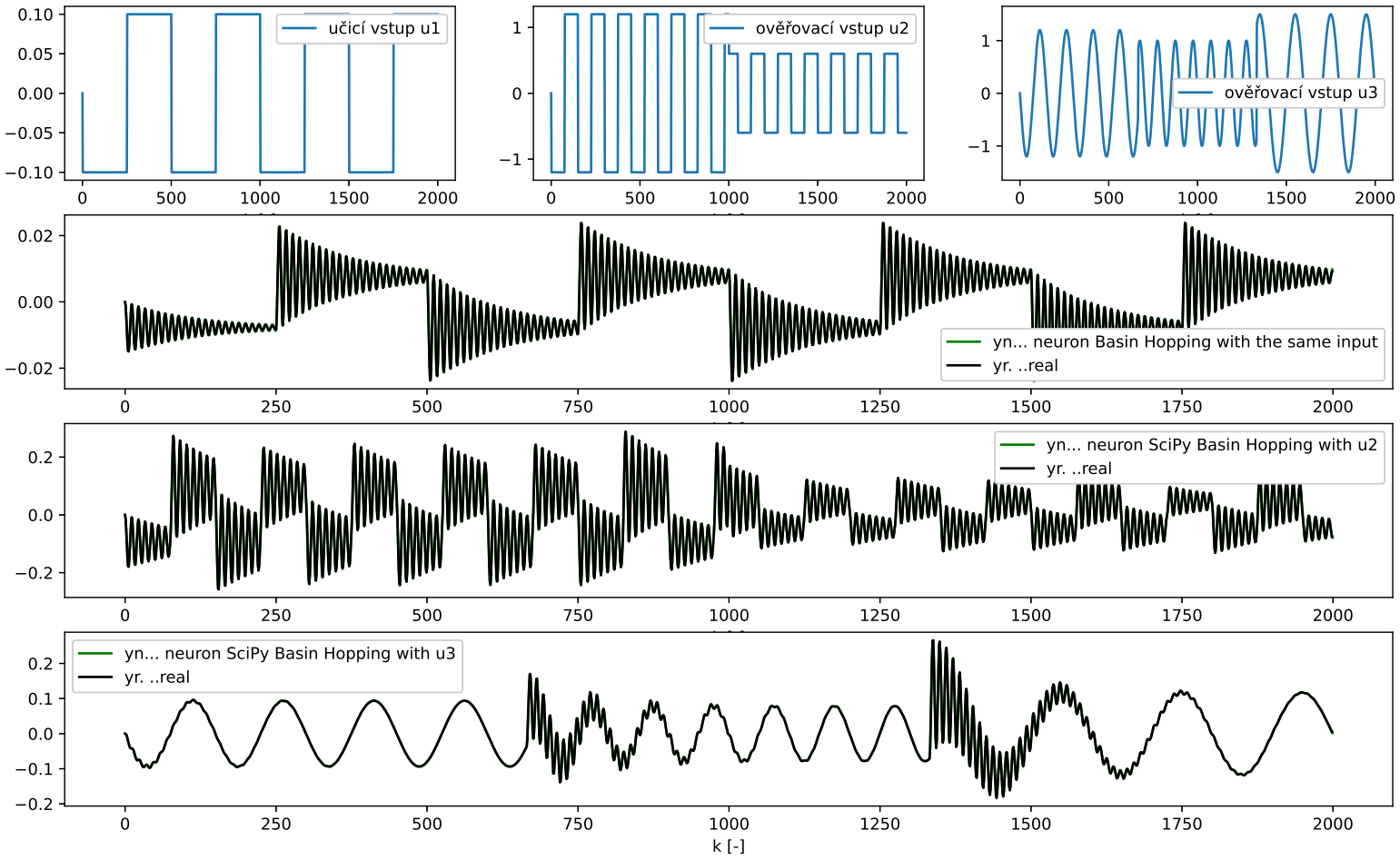
- MAE - průměrná chyba modelu v absolutní hodnotě mean(**abs**(yr−yn))
- qErr - suma druhých mocnin hodnot odchylek v chybovém vektoru dot(e,e)

[20] V praktické implementaci nebyly uvažovány kraje vektorů, tedy použité vektory byly například yn[10:−10].

- maxDiff - maximální odchylka modelu **max**(**abs**(yr−yn))



Obrázek 33: Předtrénování 2DOF kmitavého systému pomocí L-M



Obrázek 34: Identifikace a testování modelu 2DOF nelineárně tlumeného oscilátoru LNU_12_12

Z výstupu je tedy patrné, že model dosahuje vysoké přesnosti.

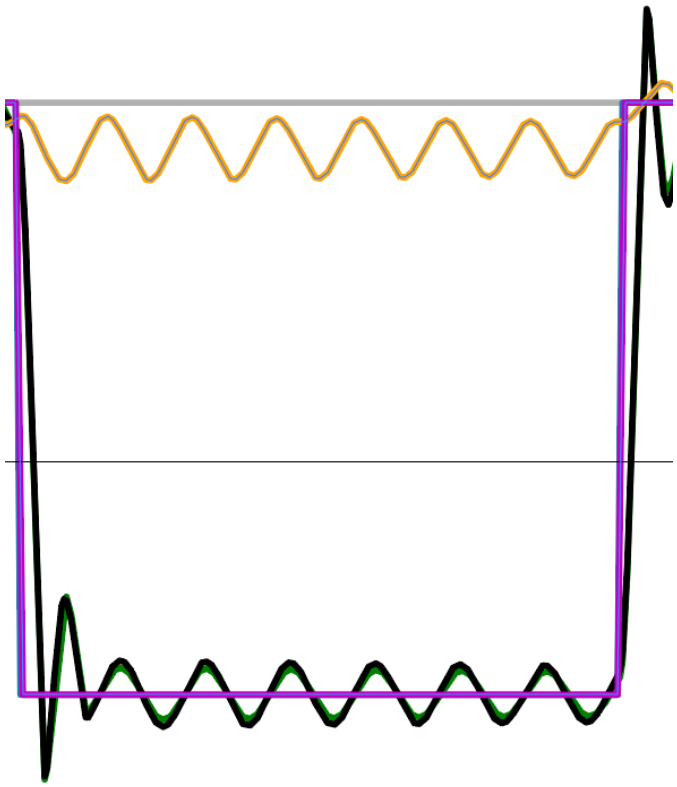
Tabulka 12: Chyby HONU při simulaci 2DOF tlumeného oscilátoru

LNU_8_8	u_1	u_2	u_3
MAE	7.01e-05	2.1e-03	8.6e-04
qErr	1.88e-05	0.013	4.6e-03
maxDiff	4.1e-04	7.2e-03	5.8e-03
LNU_12_12	u_1	u_2	u_3
MAE	4.53e-05	1.3e-03	5.5e-04
qErr	7.70e-06	4.7e-03	1.7e-03
maxDiff	2.6e-04	4.3e-03	3.5e-03
LNU_20_20	u_1	u_2	u_3
MAE	6.95e05	2.1e-03	4.1e-04
qErr	1.86e-05	1.3e-02	6.9e-03
maxDiff	2.6e-04	4.6e-03	5.8e-03

Tento systém byl dále řízen pomocí MRAC, s užitím teorie již odvozené a implementované v rešeršní části.
Jak je patrno z obrázku 34, 2DOF systém byl opět dobře identifikován a průběh y_r velmi přesně koresponduje s průběhem y_n. Tato přesnost již není tak vysoká jako u 1DOF systému, jak lze vidět na detailu na obrázku 35 upraveném tak, aby oba průběhy byly vidět zároveň:.

7.5 Diskuze výsledků identifikace a řízení 1DOF a 2DOF nelineárně tlumeného torzně kmitavého systému

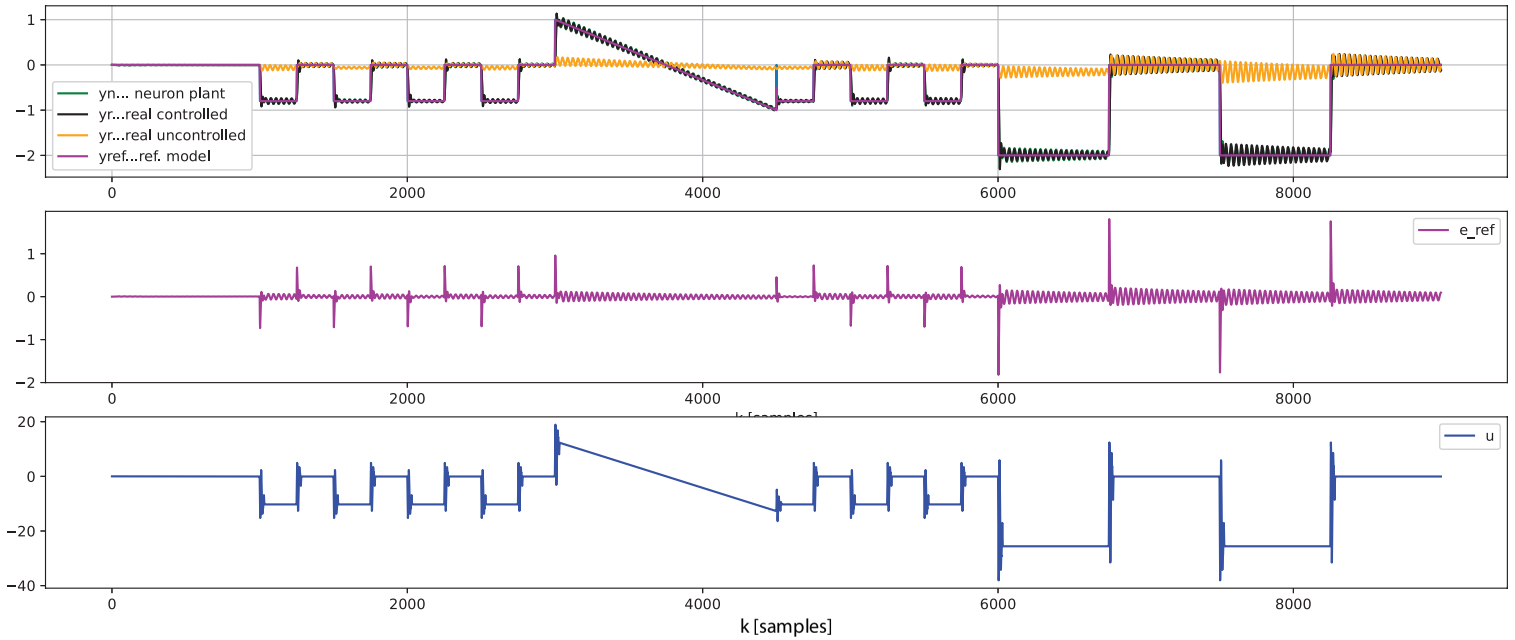
Byla provedena identifikace a řízení 1DOF a 2DOF torzního systému pomocí HONU.
i) V rámci identifikace, která byla implementována primárně pomocí pseudoglobální verze algoritmu BFGS v rámci algoritmu Basin Hopping, které byly popsány v kapitole věnované optimalizaci, bylo provedeno porovnání pro LNU_8_8, LNU_12_12 a LNU_20_20. Výsledky byly překvapivě dobré, jak ukazují tabulky 11 a 12. Tato přesnost byla potvrzena i při průběhu řízení, kde, jak je na obrázku 30 a 35. patrné, se průběh výstupu neuronu y_n odchyluje od reálného, resp. simulovaného, systému y_r jen málo a stabilně se drží kolem jeho hodnoty.
Z důvodu jak přesnosti, tak i stability, aproximace pomocí LNU_12_12 a LNU_20_20 nebyly provedeny optimalizace s využitím HONU vyššího řádu, jelikož jejich inherentní nevýhoda ve formě jejich mnohem vyššího počtu parametrů je pro



Obrázek 35: Detail: Řízení 2DOF kmitavého systému; zobrazení upraveno pro možnost současného pozorování yn a yr

aplikace real-time řízení staví do pozice spíše nutného zla v případě nedostatečnosti LNU, než plnohodnotného konkurenta. Toto je krátce okomentování a kompromis mezi metodami je navržen i v kapitole 8.2.
ii) Řízení bylo primárně realizováno pomocí L-M algoritmu. Problematickou se ukázala být otázka počateční hodnoty vahového vektoru neuroregulátoru v. Tento problém byl řešen implementováním smyčky, ve které se generovaly částečně náhodné vektory v, přičemž zároveň byla ukládána hodnota odchylky od y_ref. Nejlepší regulátor byl po ukončení smyčky vybrán podle velikosti této odchyky.
Naučené regulátory vykazují u 1DOF značný překmit, ale také rychlé ustálení - řádově 80-200 ms. V budoucích aplikacích by bylo vhodné toto učení implementovat pokročilejší metodou, například pomocí algoritmu Basin Hopping, který umožní penalizovat překmit a dostaneme tak sice pomalejší regulátor, ale také nižší hodnoty překmitu.

U 2DOF verze nedocházelo k takovému překmitu, ale průběh regulace je více rozkmitaný, jak je patrné z obrázku 36, který zobrazuje průběh regulace na testovací, tj. jiném než trénovacím, průběhu yref.
Regulátory tedy byly také úspěšně ověřeny na jiných průbězích žádané hodnoty d, jak je ukázáno např. na obrázku 31.



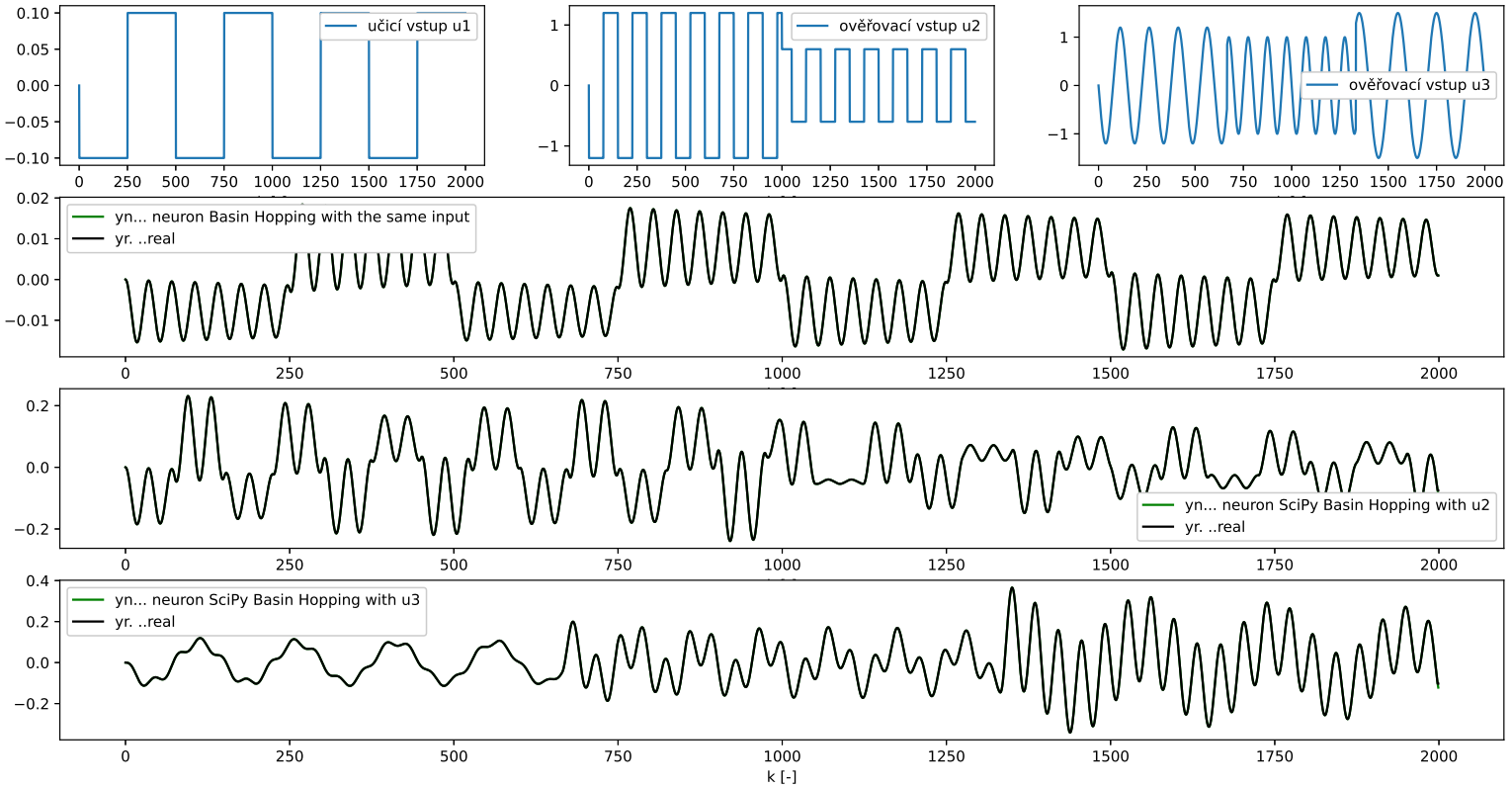
Obrázek 36: Řízení modelu 2DOF nelineárně tlumeného oscilátoru LNU_20_20 pomocí neuro-regulátoru s hloubkou 30 s použitím testovacího průběhu yref

7.6 Identifikace a řízení nelineárně tlumených torzně kmitavých systémů s jinými parametry nelinearit

Dále byla provedena analýza stejných systémů se změnou parametrů:
$b_i^I = 1000 \leftarrow b_i^{II} = 1$ a $\beta_i^I = 1000 \leftarrow \beta_i^{II} = 1$.
Způsobená změna chování nelinearity již byla znázorněna na obrázku 24.
Tato změna podstatně zhoršila chování použitých optimalizačních algoritmů BFGS a Basin Hopping, které pro většinu parametrů vyústily v chybu RuntimeWarning: invalid value encountered **in** double_scalars alpha1 = **min**(1.0, 1.01∗2∗(phi0 − old_phi0)/derphi0).
Je obtížné udělat jednoznačný závěr o schopnosti HONU aproximovat tento typ nelinearit, jelikož velká část problémů, na které jsem při řešení této části narazil, souvisí s nedokonalou optimalizací. Vyvstává zde tedy možnost implementace algoritmů podobných Basin Hopping, které by však pro aplikace v HONU byly robustnější, jak je o něco více rozvinuto v podkapitole 8.3.
Z důvodu problematické implementace bylo provedeno jen srovnání pro 2DOF systém, které je zobrazeno v tabulce 13. Srovnání se systémem s původním tlumením ukazuje, že tato identifikace poskytuje o něco lepší výsledky. To je pravděpodobně dáno méně skokovým charakterem tohoto tlumení, ale pravděpodobně bylo způsobeno i větší pozorností věnované správnému průběhu optimalizace tak, aby bylo zamezeno výše zmíněné chybě.

Tabulka 13: Chyby HONU při simulaci 2DOF tlumeného oscilátoru s tlumením s jinými koeficienty

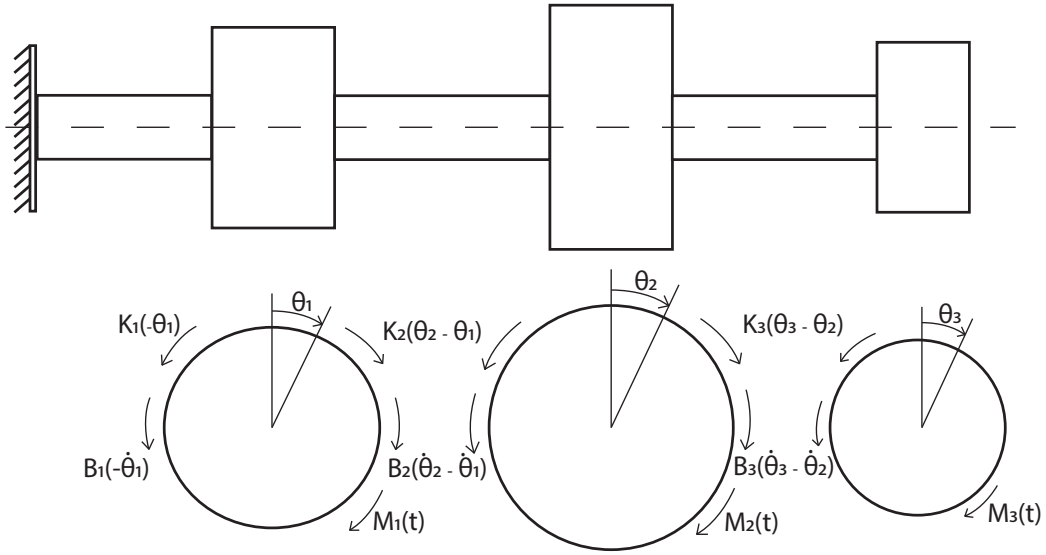
LNU_8_8	u_1	u_2	u_3
MAE	1.29e-06	2.08e-04	8.47e-05
qErr	5.82e-06	1.06e-04	1.29e-4
maxDiff	5.02e-05	7.18e-04	8.47e0-4
LNU_12_12	u_1	u_2	u_3
MAE	1.29e-06	2.09e-05	8.66e-06
qErr	5.82e-06	1.05e-05	1.29e-05
maxDiff	5.13e-05	7.23e-04	2.71e-05
LNU_20_20	u_1	u_2	u_3
MAE	1.29e-06	2.1e-06	8.71e-06
qErr	5.82e-05	1.06e-05	1.29e-05
maxDiff	5.09e-06	7.46e-05	2.75e-05



Obrázek 37: Identifikace 2DOF nelineárně tlumeného oscilátoru s jinými koeficienty nelineárního tlumení pomocí LNU_12_12

7.7 Identifikace nelineárně tlumeného 3DOF systému

Dále byla provedena identifikace 3DOF nelineárně tlumeného systému.



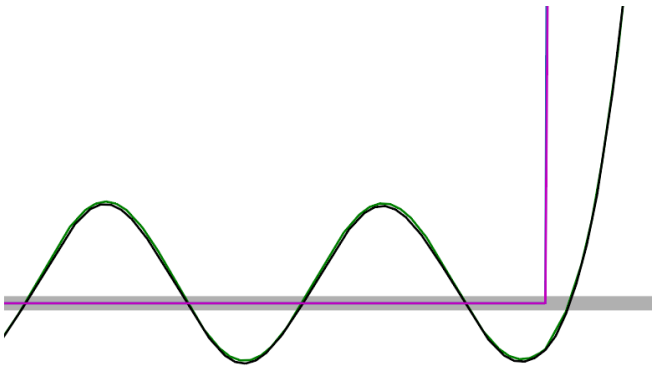
Obrázek 38: Diagram řetězce tří torzně kmitajících hmot

i) *Identifikace:* V této části již byla identifikace náročnější - použité optimalizační algoritmy, tj. především Basin Hopping, často divergovaly[21], a ani samotný model není zcela přesný - zůstává však dostatečně stabilní při různých průbězích řízení. Z toho důvodu v této části již není uvedeno srovnání, ale pouze je zde ukázán úspěšný model, který byl identifikován s využitím LNU_12_12, znázorněn na obrázku 39. V tomto případě nebyly měřené jednotlivé metriky jako u předchozích případů. I při přesném přiblížení vektorového obrázku je však patrné, že simulace neuronem je stabilní a poskytuje poměrně přesný odhad. Toto je dále ukázáno na detailu v obrázku 39. V budoucnu by bylo vhodné tuto analýzu rozšířit a navrhnout lepší způsoby identifikace, buďto lepšími algoritmy nebo například navrhovaným hybridním HONU - viz. podkapitola 8.2.

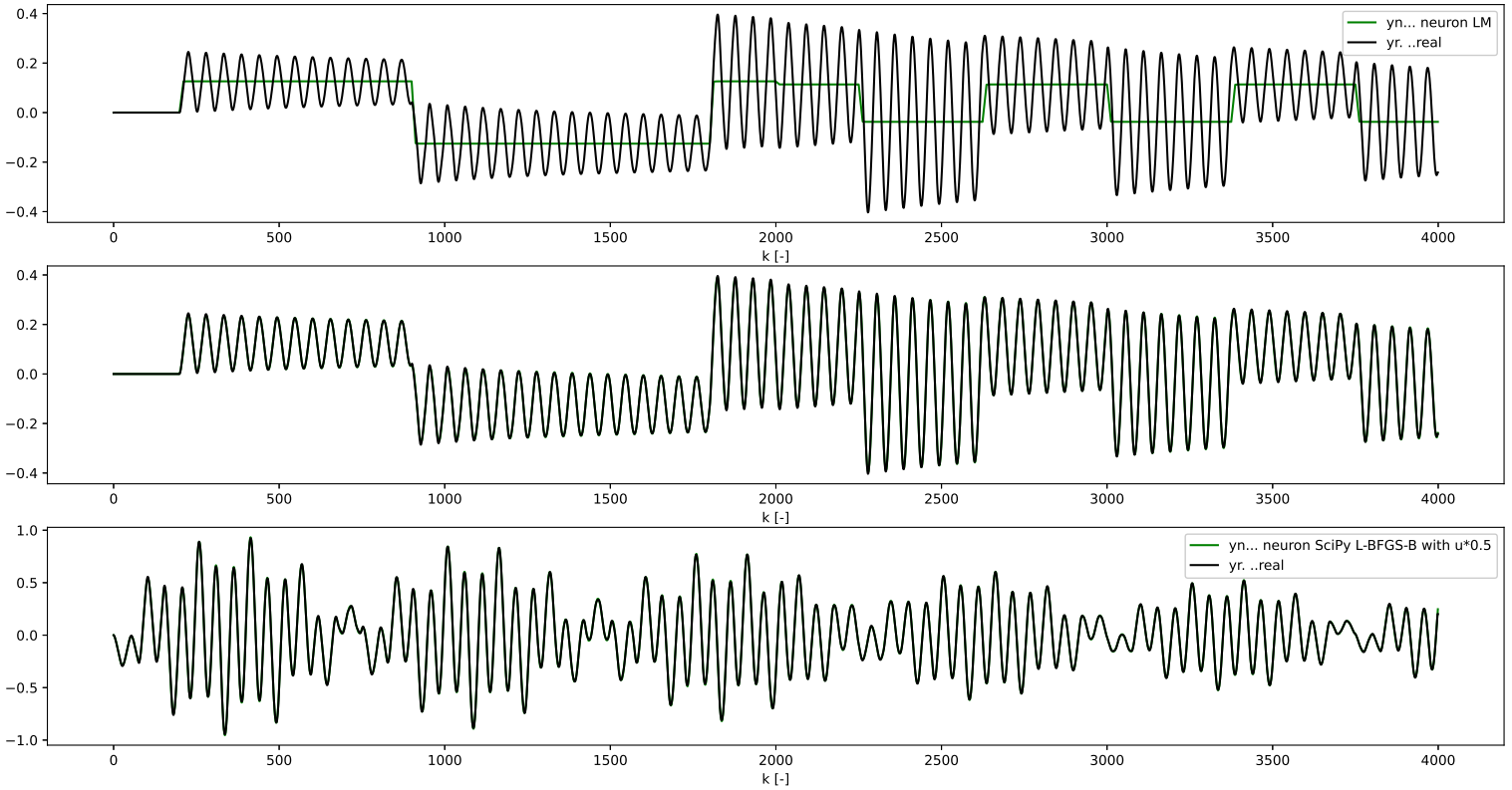
ii) *Řízení:* Po relativně úspěšné identifikaci byl proveden i pokus o řízení 3DOF systému.
Identifikovaný systém, jak je dále patrné z výstupů řízení je poměrně kmitavý. To lze přisoudit nedokonalé identifikaci, ale také dynamickým vlastnostem 3DOF

[21] Zde vyvstává již několikrát zmíněná problematika nedokonalosti tohoto algoritmu při použití na optimalizaci HONU a navrhovaná zlepšená implementace v rámci např. budoucí diplomové práce.

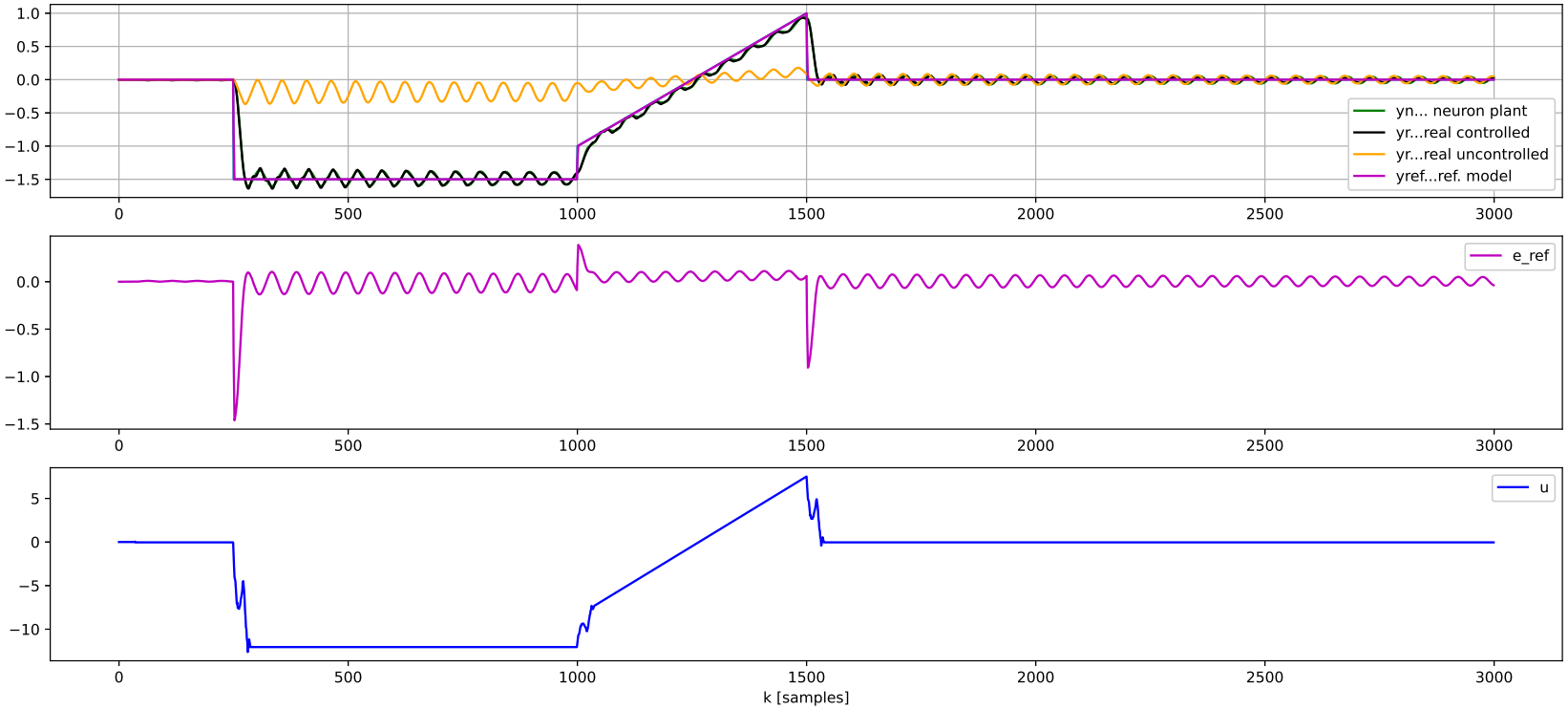
systému. Byla prozkoumána řada variant hloubky neuroregulátoru a nejlepší výsledek byl dosažen pro hloubku $nvx = 40$. Na obrázku 41 a 42 je vidět, že chování regulovaného systému je podstatně lepší, než chování jeho neregulované verze.
Natrénovaný regulátor byl následně vyzkoušen i na testovacím průběhu veličiny d, resp. yref, výsledek, na obrázku 42, je opět podobné kvality jako výsledek na trénovacím průběhu.



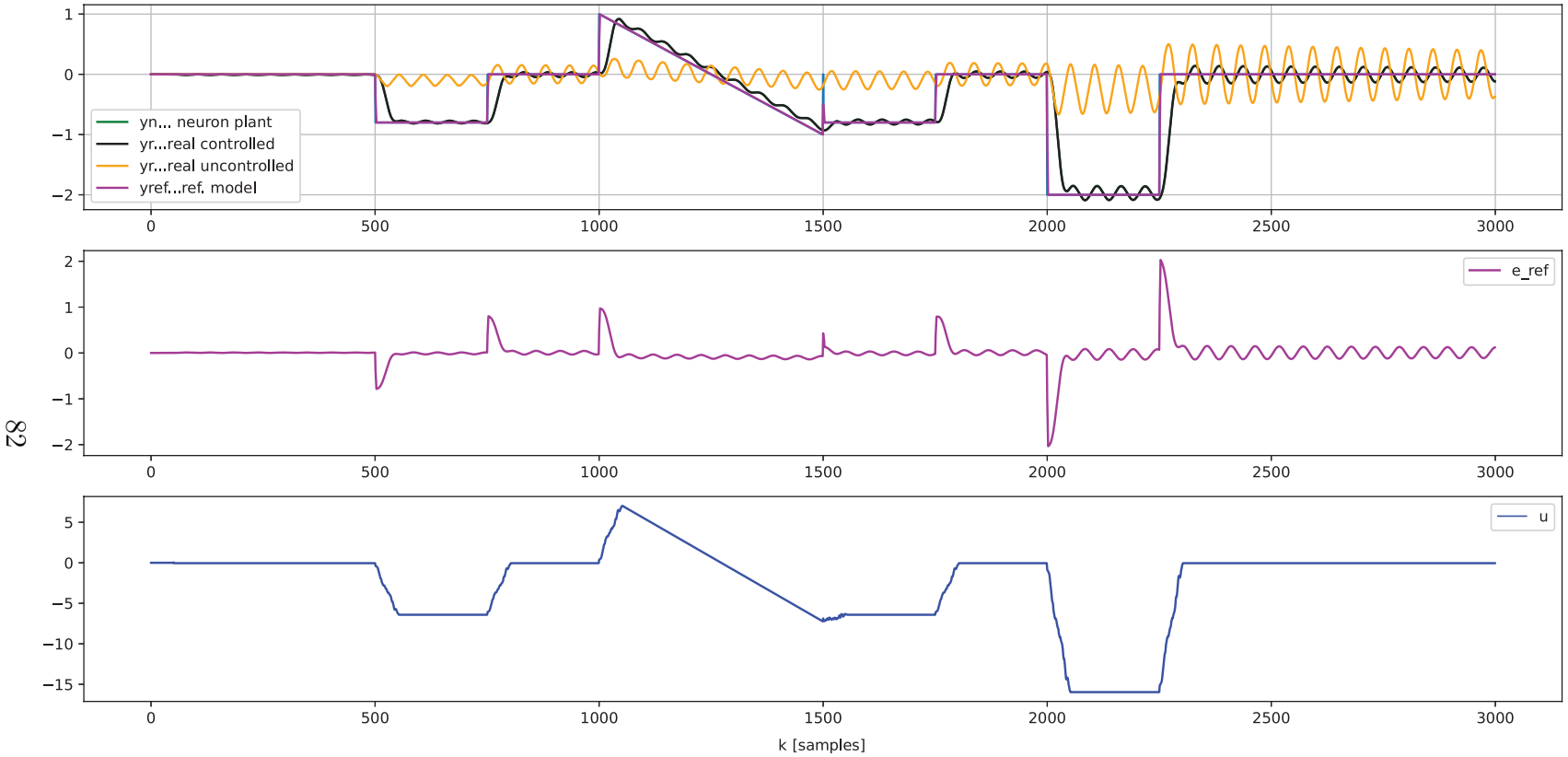
Obrázek 39: Detail obr. 42 v oblasti (k = 2200-2300): Řízení 3DOF nelineárně tlumeného oscilátoru s hloubkou neuroregulátoru 40 na testovacím průběhu



Obrázek 40: Identifikace 3DOF nelineárně tlumeného oscilátoru pomocí LNU_12_12



Obrázek 41: Řízení 3DOF nelineárně tlumeného oscilátoru s hloubkou neuroregulátoru 40 na testovacím průběhu yref



Obrázek 42: Řízení 3DOF nelineárně tlumeného oscilátoru s hloubkou neuroregulátoru 40 na testovacím průběhu yref

8 Možná témata dalších prací a dalšího vývoje problematiky

8.1 Stabilita HONU

Důležitou otázkou pokládanou v jakékoli úloze řízení je jeho stabilita. Jako potenciální řešení této otázky se jeví datový přístup k BIBS stabilitě[15] a obdobná problematika zkoumaná na FS, například viz. Striktní Stabilita Adaptivních Dynamických Polynomiálních Systémů[11].

8.2 Řádově-hybridní HONU

V této práci byla ukázána značná nevýhoda HONU vyššího řádu - jeho velmi rychle rostoucí počet parametrů [22].
Jako potenciální řešení tohoto problému se jeví kombinace LNU a QNU. Při zkoumání HONU vyššího řádu, např. QNU_10_10, se ukazuje, že kombinace jako $yn[k] \cdot u[k-10]$ mají obvykle velmi nízký, téměř vynulovaný, odpovídající prvek vektoru $\mathbb{W}$. Nabízí se zde možnost zcela zahodit takové prvky a použít například vektor $colX^{hybrid2}$ odpovídajíc vektoru $colX^{hybrid2} = [1 \quad \mathbb{X} \quad \mathbb{X}_q]^T$, kde $\mathbb{X}_q$ obsahuje prvky vektoru $\mathbb{X}$ na druhou, tj $\mathbb{X}_q = [1 \quad y_n[k-1]^2 \quad y_n[k-2]^2 \quad \dots \quad]^T$.

8.3 Hlubší prozkoumání problematiky optimalizace v rámci učení HONU

V této práci byly prozkoumány některé globální metody vhodné například pro offline předučení. Především se jedná o verzi BFGS v rámci algoritmu Basin Hopping. Tento algoritmus občas vykazuje neideální chování, kde současný odhad diverguje, tj. volá funkci fQ(w) s takovým vektorem w, že funkce vrací ,,NaN". Další prozkoumání a případná úprava tohoto algoritmu by mohla být přínosná. Také se naskytuje možnost vlastní implementace algoritmů L-M, či CG v podobném stylu jako je algoritmus Basin Hopping, tj. s prohledáváním prostoru počátečních odhadů w_0. Algoritmus Basin Hopping implementovaný v knihovně SciPy v Pythonu je open-source, což by mohlo tuto úlohu ulehčit a převést ji spíše do spektra problémů reverzního inženýrství.

[22] Patrné mj. z tabulek 1 a 2

9 Závěr

Cílem této práce byla implementace identifikace a MRAC řízení na torzně kmitající systém s nelineárním tlumením.

K řešení problému byl zvolen systematický přístup a jednotlivé postupy byly implementovány a testovány v programovacím jazyku Python, především s využitím knihoven SciPy a Numpy. Tyto postupy byly vyzkoušeny a nejvhodnější z nich byly aplikovány na řešenou problematiku nelineárně tlumeného torzně kmitavého systému pro 1DOF,2DOF a 3DOF.

Nejprve byly implementovány základní algoritmy pro identifikaci, včetně testovacích systémů s různým stupněm nelinearity, na kterých tyto algoritmy byly otestovány. Tyto algoritmy byly implementovány jak pro lineární HONU, LNU, tak i pro HONU vyššího řádu, tj. QNU a CNU. Testovací systémy byly řešeny buďto pomocí ode solveru z knihovny SciPy, případně numerickým schématem pomocí metody konečných diferencí - v tomto případě byly ale i tak později otestovány pomocí ode solveru.

Následovně byly implementovány lineární a kvadratické algoritmy řídícího neuroregulátoru. Tyto algoritmy byly úspěšně otestovány na lineárním tlumeném kmitavém 1DOF systému a na modelu tlumeného matematického kyvadla. Bylo provedeno předběžné vyhodnocení vhodných postupů, mj. fakt, že lineární regulátor zpravidla dosahuje lepších výsledků než kvadratický, což je pravděpodobně způsobeno obtížnější optimalizací.

Poté byla pozornost věnována problematice optimalizace se zaměřením na aplikaci v problematice HONU. Vhodné učení, tedy optimalizace vektorů $\mathbb{W}$, reps. $\mathbb{V}$, je klíčovou částí každé aplikace adaptivní identifikace a řízení s využitím neuronových jednotek i sítí. Byla prozkoumána řada algoritmů a bylo provedeno základní srovnání kvality jejich výsledků a časové náročnosti. Jako základní dobrá metoda se jeví často používaný L-M. Z ostatních prozkoumaných metod se obecně ukázaly být vhodnější metody gradientní. Ze zkoumaných algoritmů se ukázal být nejvhodnější algoritmus BFGS a dále jeho globální (pseudoglobální) verze v Basin Hopping, která byla použita k identifikaci v části zaměřené na torzní systém. Optimalizační algoritmy byly aplikovány i na řízení, zde se ale ukázalo správné nastavení parametrů tak, aby optimalizace proběhla dobře, náročnější a zároveň často nebylo dosaženo výrazně lepších výsledků než při použití L-M k trénování regulátoru.

V další části byl analyzován systém torzně kmitajících hmot s nelineárním

tlumením, byly sestaveny jeho pohybové rovnice a byla provedena implementace v prostředí Python tak, aby systém bylo možné numericky integrovat pomocí odeint řešiče diferenciálních rovnic z knihovny SciPy.
Postupně byly řešeny případy 1DOF, 2DOF a 3DOF. Bylo provedeno srovnání úspěšnosti identifikace pro různé hloubky HONU a jednotlivé výsledky byly zaznamenány, kromě slovního shrnutí, také do tabulek 11, 12 a 13. Byla provedena i analýza vlivu změny parametrů nelineárního tlumení, tu však komplikovaly problémy s optimalizací. Možné řešení tohoto problému je krátce naznačeno v podkapitole 8.3. I přes tento problém se možnosti identifikace systému s jinými parametry jeví dobré.
Pro tyto torzní systémy byly také navrhnuty a natrénovány lineární neuroregulátory. Pro 1DOF byly výsledky dobré. Při řízení vznikají poměrně velké překmity, ale zároveň se systém rychle ustální na nové žádané hodnotě. Regulační odchylka je nulová. Problém vysokého překmitu bude možné pravděpodobně vyřešit implementací pokročilejších optimalizačních metod s možností vytvoření vlastní kriteriální funkce, ve které se překmit může penalizovat. Regulátory byly úspěšně testovány i na jiném průběhu yref. Vlastnosti regulátoru zůstaly v tomto případě velmi podobné.
V případě 2DOF systému již byla identifikace i řízení mírně náročnější. Nepodařilo se dosáhnout rychlého ustálení na nové hodnotě jako v případě 1DOF systému a systém dále mírně osciloval. I tak bylo dosažené chování výrazně lepší než u neregulovaného systému. Stejně jako u ostatních stupňů volnosti byly výsledky úspěšně vyzkoušeny na jiném průběhu žádané veličiny d. Lze předpokládat, že podrobnějším zaměřením-se na optimalizační metody a dalším zlepšením identifikace, např. ve shodě s metodou navrženou v podkapitole 8.2, bude možné dosáhnout lepšího chování regulátoru.
Identifikace v případě 3DOF systému byla náročnější a výsledky jsou méně přesné. Při testování identifikace i při samotném řízení se však jeví stabilní z numerického hlediska. Regulovaný systém má stále značně kmitavý charakter, ale dokáže systém alespoň přibližně uřídit kolem žádaných veličin, a to jak na trénovacím, tak i na verifikačním průběhu yref.

Celkově hodnotím výsledek této práce jako úspěšný. Podařilo se ukázat, že HONU jsou schopné identifikace torzně kmitajícího systému s nelineárním tlumením a také byla ukázána možnost jejich řízení. Samotná rešerše a implementace jednotlivých algoritmů v Pythonu zabrala velké množství času, což znemožnilo vyřešit některé problémy, mj. s optimalizací. U těchto problémů byl alespoň navrhnut možný postup jejich řešení.
V této práci byl položen robustní základ pro řešení dané problematiky, spolu s implementací řady algoritmů, na který lze nyní dále navázat v řadě směrů. Mimo

jiné bude na zde uvedené postupy plynule navázáno v mém doktorském studiu na FS ČVUT.

Seznam obrázků

Seznam tabulek

Ukázky kódu

References

[1] VALÁŠEK, Michael, Filip ŠÁFR, Zdenek NEUSSER a Jan PELIKÁN. Position Control of Flexible Chain Using Wave Based Control. *ECCOMAS Thematic Conference on Multibody Dynamics*. Červen 19-22, 2017. Praha, Česká Republika, 2017.

[2] BENEŠ, Petr, Michael VALÁŠEK, Zbyněk ŠIKA, Jan ZAVŘEL a Jan PELIKÁN. SHAVO Control — the Combination of the Adjusted Command Shaping and Feedback Control for Vibration Suppression. *Acta Mechanica*. Department of Mechanics, Biomechanics and Mechatronics, Faculty of Mechanical Engineering, Czech Technical University in Prague, **2019**, 15.

[3] Optimization and Root Finding (scipy.optimize). *Https://docs.scipy.org/* [online]. [cit. 2020-04-13]. Dostupné z: https://docs.scipy.org/doc/scipy/reference/optimize.html

[4] Basin Hopping as a General and Versatile Optimization Framework for the Characterization of Biological Macromolecules. Artificial Intelligence Applications in Biomedicine [online]. 2012, 04 Dec 2012, 2012, 19 [cit. 2020-06-30]. DOI: 674832. Dostupné z: https://www.hindawi.com/journals/aai/2012/674832/

[5] BEKDAS, Gebrail, Sinan Melih NIGDELI a Melda YUCEL, ed. *Artificial Intelligence and Machine Learning Applications in Civil, Mechanical, and Industrial Engineering*. USA: IGI Global, 2019. ISBN 1799803015. ISSN 2327-0411.

[6] VALÁŠEK, M., J. PELIKÁN a M. NEČAS. Mechatronics Stiffness with Disturbance Force Rejection. *Bulletin of Applied Mechanics*. **2010**, 89-93. ISSN 1801-1217.

[7] VALÁŠEK, M. a M. SMRŽ. Active Integral Vibration Control of Elastic Bodies. *Applied and Computational Mechanics*. University of Western Bohemia, 2008, **2**.

[8] HELLEVIK, Leif Rune. *Numerical Methods for Engineers* [online]. Department of Structural Engineering, NTNU, 2020 [cit. 2020-06-20]. Dostupné z: http://folk.ntnu.no/leifh/teaching/tkt4140/

[9] WEI, Qinglai a Derong LIU. Policy Iteration Adaptive Dynamic Programming Algorithm for Discrete-Time Nonlinear Systems. In: LIU, Derong a Qinglai WEI. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*. 25. IEEE, 2014, s. 621-634.

DOI: 10.1109/TNNLS.2013.2281663. ISSN 2162-2388. Dostupné také z: https://ieeexplore.ieee.org/document/6609085

[10] Taylor series. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-07-30]. Dostupné z: https://en.wikipedia.org/wiki/Taylor_series

[11] BENEŠ, Peter, Ivo BUKOVSKÝ a Ondřej BUDÍK. Striktní Stabilita Adaptivních Dynamických Polynomiálních Systémů. *AUTOMATIZÁCIA A RIADENIE V TE-RII A PRAXI*. STARÁ LESNÁ, SR, 2019, **13**.

[12] BUKOVSKY, Ivo, Jan VORACEK, Kei ICHIJI a Homma NORIYASU. Higher Order Neural Units for Efficient Adaptive Control of Weakly Nonlinear Systems. *International Joint Conference on Computational Intelligence*. 2017-, **9th**. DOI: 10.5220/0006557301490157 ·.

[13] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep Learning* [online]. USA: MIT Press, 2016 [cit. 2020-07-31]. Dostupné z: http://www.deeplearningbook.org

[14] KOČVARA, Michal. *Algoritmy numerické optimalizace* [online]. AV ČR, 2004 [cit. 2020-07-31]. Dostupné z: http://staff.utia.cas.cz/kocvara/numopt.pdf

[15] WANG, Zhuo a Derong LIU. Stability Analysis for a Class of Systems: From Model-Based Methods to Data-Driven Methods. *IEEE Transactions on Industrial Electronics*. IEEE, 2014, **13**(11), 6463 - 6471. DOI: 10.1109/TIE.2014.2308146.