ČVUT
ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ
ČVUT V PRAZE

Ústav Mechaniky, Biomechaniky a Mechatroniky

Diplomová práce: Detekce poškození konstrukcí pomocí měření vibrací a výpočetní inteligence

Bc. Vojtěch Kopecký

Praha, 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kopecký** Jméno: **Vojtěch** Osobní číslo: **476088**

Fakulta/ústav: **Fakulta strojní**

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

Studijní program: **Aplikované vědy ve strojním inženýrství**

Specializace: **Mechatronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Detekce poškození konstrukcí pomocí měření vibrací a výpočetní inteligence

Název diplomové práce anglicky:

Structural damage detection using vibration measurement and computational intelligence

Pokyny pro vypracování:

1. Seznamte se s prostředky a metodami experimenální modální analýzy (EMA), modelováním konstrukcí metodou konečných prvků (MKP), nejčastějšími druhy porušení konstrukce a metodami detekce a trénovaní neuronových sítí.
2. Proveďte rešerši existujících metod detekce poruchy konstrukce na základě měření vibrací
3. Zvolte jednu z typických poruch konstrukcí a vhodný model tohoto poškození a vytvořte parametrizovaný model konstrukce a poruchy
4. Porovnejte výsledky prostředky experimentální modální analýzy a ověřte experimentálně
5. Vytvořte serie modelů a proveďte simulační experimenty. Ty použijte jako bázi dat pro trénování neuronových sítí na rozpoznání poruchy nosníku.

Seznam doporučené literatury:

BILOŠOVÁ, ALENA. Experimental modal analysis. VŠB TU Ostrava, 2006.
ŠPANIEL, Miroslav; HORÁK, Zdeněk. Úvod do metody konečných prvků. České vysoké učení technické, 2011.
Gupta, M. M., Homma, N., Hou, Z. G., Solo, A. M., & Bukovsky, I. (2010). Higher order neural networks: fundamental theory and applications. In Artificial higher order neural networks for computer science and engineering: trends for emerging applications (pp. 397-422). IGI Global.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Pavel Steinbauer, Ph.D. odbor mechaniky a mechatroniky FS

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

Datum zadání diplomové práce: **22.04.2022** Termín odevzdání diplomové práce: **15.08.2022**

Platnost zadání diplomové práce:

Ing. Pavel Steinbauer, Ph.D.	doc. Ing. Miroslav Španiel, CSc.	doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	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.

20.7.2022

Datum převzetí zadání Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Prohlášení

Prohlašuji, že jsem napsal tuto diplomovou práci s názvem "Detekce poškození konstrukcí pomocí měření vibrací a výpočetní inteligence" sám a všechna literatura a externí zdroje jsou vypsány v seznamu zdrojů.

V Praze dne: 6. 8. 2022

Vojtěch Kopecký

Anotační list/Annotation list

Jméno autora/Author's name:	*Bc. Vojtěch Kopecký*
Vedoucí práce/Supervisor:	*Ing. Pavel Steinbauer, Ph.D.*
Název práce/Title of thesis:	*Detekce poškození konstrukcí pomocí měření vibrací a výpočetní inteligence* *Structural damage detection using vibration measurement and computational intelligence*
Rok/Year:	*2022*
Škola/University:	*České vysoké učení technické v Praze* *Czech Technical University in Prague*
Obor studia/Field of study:	*Aplikované vědy ve strojním inženýrství, specializace mechatronika* *Applied sciences in mechanical engineering, specialization mechatronics*
Ústav/Department:	*12105 Ústav mechaniky, biomechaniky a mechatroniky* *12105 Department of Mechanics, Biomechanics and Mechatronics*
Počet stran/Number of pages:	*126*
Počet tabulek/Number of tables:	*0*
Počet příloh/Number of attachments:	*55 stran/pages*
Počet obrázků/Number of figures:	*105*
Klíčová slova/Keywords:	*detekce poškození konstrukcí, detekce nelineární poruchy, detekce prasklin, strojové učení, hluboké neuronové sítě, konvoluční neuronové sítě, umělá inteligence, output error model, identifikace nelineárního filtru, metoda konečných prvků, experimentální modální analýza* *structural damage detection, nonlinear failure detection, crack detection, machine learning, deep neural network, convolutional neural network, artificial intelligence, output error model, nonlinear filter identification, finite element method, experimental modal analysis*

Abstrakt/Abstract

Globální metody detekce poruchy konstrukcí slouží ke zjištění přítomnosti, případně polohy poruchy v konstrukci. Jsou založeny na měření vibrací. Jelikož vibrace jsou projevem dynamických vlastností a dynamické vlastnosti jsou společné pro celou konstrukci, nemusí být měření prováděno v místě poruchy, aby byla odhalena. Některé z poruch je obtížné odhalit, přitom jejich neodhalení může mít vážné následky. Dosavadní práce se zaměřovali především na poruchu projevující se lokálním poklesem tuhosti. Taková porucha se projevuje lineárně a k její detekci je obvykle třeba mít k dispozici měření totožné konstrukce v neporušeném stavu (referenční měření). Tato práce se zaměřuje na nelineární chování poruchy, konkrétně skokové změny tuhosti při nárazech v prasklinách, uvolněných šroubových spojích a delaminovaných kompozitech. Výhodou tohoto přístupu je, že teoreticky nepotřebuje referenční měření. V rámci práce byl vytvořen simulační model této poruchy a provedeny experimenty na reálném nosníku. Z obojího vyplývá, že tento typ poruchy se na datech ve frekvenční oblasti projevuje peaky, které nejsou projevem vlastní frekvence. Pro detekci poruchy byly vyvinuty dvě metody, jedna založená na identifikaci parametrů nelineárního filtru, druhá používá hluboké neuronové sítě. Úkolem metod bylo provést klasifikaci dat na skupiny s poruchou a bez poruchy. Identifikací filtru byla úspěšně klasifikována data obsahující dvě dominantní vlastní frekvence, kdy parametr vyjadřující nelinearitu byl přibližně 1000krát větší pro data s poruchou než bez poruchy. Pomocí konvolučních neuronových sítí s vrstvou separable convolution bylo dosaženo 97 % metriky accuracy (přesnost) na simulovaných testovacích datech a 100 % accuracy na měřených testovacích datech. Tato data ovšem představovala malý vzorek a byla upravena.

Global methods of structural damage detection are used to detect the presence or location of a failure in the structure. They are based on vibration measurements. Since vibration is caused by dynamic properties and dynamic properties are global properties of the entire structure, measurements do not need to be made at the location of the failure to detect it. Some of the failures are difficult to detect, however not detecting them would have serious consequences. Previous work has focused mainly on the failure formed by a local decrease in stiffness. Such a failure behaves linearly, and to detect it, it is usually necessary to have a measurement of an identical structure in an intact state available (reference measurement). This work focuses on the non-linear failure behaviour, specifically, the step changes in stiffness caused by impacts in cracks, loosely bolted joints and delaminated composites. The advantage of this approach is that it theoretically does not need a reference measurement. In this work, a simulation model of this failure was created, and measurement experiments were carried out. It follows from both that this type of failure manifests itself in the data in the frequency domain by peaks that are not a manifestation of the natural frequency. Two methods were developed for fault detection, one based on the identification of nonlinear filter parameters, the other using deep neural networks. The task of the methods was to classify the data into groups with and without failure. Data containing two dominant natural frequencies were successfully classified by filter identification, where the parameter expressing nonlinearity was 3 orders of magnitude different for data with and without a disorder. Using convolutional neural networks with a separable convolution layer, 97% accuracy was achieved on simulated test data and 100% accuracy on measured test data. However, measured data represented just a small sample and has been edited.

Poděkování

Je mi velkým potěšením, že mohu na tomto místě poděkovat své přítelkyni a rodičům za podporu, bez které by nejenom tato práce nemohla nikdy vzniknout.

Vděčný jsem také Dr. Steinbauerovi, vedoucímu této diplomové práce, za ochotnou pomoc i dostatečnou volnost při jejím vypracovávání.

Obsah

Seznam obrázků 4
Seznam zkratek a symbolů 7
Úvod 12
Rešerše 13
1 Rešerše existujících metod detekce poruchy 13
1.1 Poruchy ocelových konstrukcí 13
1.2 Lokální (klasické) metody detekce poruchy konstrukcí 14
1.3 Globální (vibrační) metody detekce poruchy konstrukcí 14
2 Úvod do použitých metod a teorie 17
2.1 FEM model 17
2.2 Energie ve FEM modelu 20
2.3 Řešení ODE v Python SciPy 20
2.4 Aliasing 21
2.5 Redukce modelu 21
2.6 Tlakový kužel pod šroubem 22
2.7 Modální vlastnosti konstrukce 23
2.8 Postup provedení měření pro EMA 25
2.9 Metody zpracování EMA 27
2.10 Převod spojitého stavového popisu na diskrétní 28
2.11 Output error model 28
2.12 Levenberg-Marquardtova optimalizace 29
2.13 Hankelova singulární čísla 30
2.14 Genetický algoritmus 30
2.15 Hluboké neuronové sítě (DNN) 31
2.16 Confusion matrix a ROC křivka 37
Vlastní řešení 38
3 Cíle práce 38
4 Tvorba dat 40
4.1 Model poruchy 40
4.2 FEM model 41
4.3 Časová simulace 44
4.4 Tvorba dat pro neuronové sítě 53
5 Experimentální modální analýza 58
5.1 Napodobení poruchy v laboratorním prostředí 58
5.2 Průběh měření 59

5.3 Vyhodnocení výsledků ... 61
5.4 Kontrola simulované poruchy ... 69
6 Zpracování identifikací filtru ... 72
6.1 Lineární OE model podle flexibilního mech. systému ... 72
6.2 Přidání nelinearity do modelu ... 74
6.3 Hledání parametrů lineárního modelu ... 75
6.4 Hledání prvotního odhadu ... 77
6.5 Implementace L-M algoritmu pro nelineární filtr ... 79
6.6 Hledání parametrů nelinearity ... 80
7 Zpracování dat neuronovou sítí ... 86
7.1 Konstrukce příznaků (feature engineering) ... 87
7.2 Tvorba datového toku ... 88
7.3 Postup trénování a hodnocení výsledků ... 90
7.4 Dense sítě s jedním vstupem ... 91
7.5 Dense sítě se dvěma vstupy ... 94
7.6 Konvoluční modely single input ... 96
7.7 Konvoluční sítě s více vstupy ... 101
8 Vyhodnocení vlastního řešení problému ... 104
8.1 Vyhodnocení výsledků DNN ... 104
8.2 Vyhodnocení výsledků identifikace filtru ... 104
8.3 Testování na reálných datech z EMA ... 105
Závěr ... 109
Použité zdroje ... 112
Přílohy ... 115
9 Metoda konečných prvků ... 115
9.1 main (Python) ... 115
9.2 shapes_csv (Python) ... 115
9.3 FEM_computations (Python) ... 127
9.4 constants (Python) ... 138
10 Identifikace filtrem ... 139
10.1 main (Python) ... 139
10.2 L_M_Optimization (Python) ... 141
10.3 simulated_data_processing (Matlab) ... 146
11 Strojové učení ... 148
11.1 single_input_model (Python) ... 148
11.2 multiple_input_model (Python) ... 151
11.3 statistics_tools (Python) ... 155

12 Zpracování dat EMA 158
12.1 EMA_Zpracovani (Matlab) 158
12.2 EMA_for_AI (Matlab) 160
13 Grafy DNN 163
13.1 Síť s mnoha dense vrstvami 163
13.2 Síť s vrstvami dense a layer_normalization 164
13.3 Síť dvěma vstupy, vrstvami dense a layer_normalization 165
13.4 Síť s vrstvou convolution a batch_normalization 166
13.5 Síť s vrstvou convolution a layer_normalization 167
13.6 Síť s vrstvou separable_convolution 168
13.7 Síť s více vstupy a vrstvou separable_convolution 169

Seznam obrázků

Obr. 2.1 Nosníkový element pro rovinnou úlohu ... 18
Obr. 2.2 Spektrální výkonová hustota vzorkovaného signálu s aliasingem (výše) a bez (níže) [20] ... 21
Obr. 2.3 Tlakový kužel ve svěrném šroubovém spoji [22] ... 22
Obr. 2.4 Frekvenční přenos složen z jednotlivých modálních příspěvků [24] ... 25
Obr. 2.5 Schéma měřícího řetězce při impaktně buzené EMA [25] ... 26
Obr. 2.6 Vliv tuhosti kladívka na vybuzené frekvence soustavy [24] ... 26
Obr. 2.7 Schéma měřícího řetězce pro automatické měření [26] ... 27
Obr. 2.8 Flowchart principu genetického algoritmu [36] ... 31
Obr. 2.9 Hluboké reprezentace naučené modelem pro klasifikaci číslic [37] ... 32
Obr. 2.10 Ztrátová funkce (loss function), optimalizátor upravující váhy sítě [37] ... 33
Obr. 2.11 Pravděpodobnosti správné klasifikace bodů do tříd 0 (červená) a 1 (zelená), osa x - predikce [38] ... 33
Obr. 2.12 Princip 2D konvoluční vrstvy [37] ... 35
Obr. 2.13 Funkce relu (vlevo) a funkce sigmoid (vpravo) [37] ... 36
Obr. 2.14 Modul inception složený z více paralelních větví [37] ... 37
Obr. 2.15 Confusion matrix s dalšími charakteristikami a jejich vzorci [39] ... 37
Obr. 4.1 Motivační příklad modelu poruchy – nosník s prasklinou ... 40
Obr. 4.2 Mechanický model struktury s nelineární poruchou – pružina pracující v jednom směru ... 41
Obr. 4.3 Průběh síly F_p v závislosti na výchylce u_p ... 41
Obr. 4.4 Model nosníku tvaru L rozdělený na elementy, čísla elementů v rámečcích, globální čísla uzlů v kroužcích ... 42
Obr. 4.5 Model nosníku tvaru L s globálními posuvy, šikmá dvojitá šipka označuje natočení ... 42
Obr. 4.6 Průběh síly ft v závislosti na čase ... 44
Obr. 4.7 Průběh celkové energie soustavy bez tlumení, Diracův impuls, integrace pomocí BDF, frekvence 20 000 Hz ... 47
Obr. 4.8 Průběh celkové energie soustavy bez tlumení, Diracův impuls, integrace pomocí RK45, frekvence 20 000 Hz ... 48
Obr. 4.9 Průběh celkové energie soustavy bez tlumení, integrace pomocí BDF, frekvence 160 000 Hz ... 49
Obr. 4.10 Průběh celkové energie soustavy s tlumením, integrace pomocí BDF, frekvence 160 000 Hz ... 49
Obr. 4.11 Průběh celkové energie soustavy s tlumením, integrace pomocí BDF, frekvence 20 000 Hz ... 50
Obr. 4.12 Výsledky modelu bez poruchy (odshora): měřená rychlost uzlu, silový budící impuls, výchylka souřadnice v místě, kde by byla porucha ... 52
Obr. 4.13 Výsledky modelu s poruchou (odshora): měřená rychlost uzlu, silový budící impuls, výchylka v místě poruchy ... 52
Obr. 4.14 Průběh celkové energie soustavy bez poruchy, integrace pomocí RK45, frekvence 2 000 Hz ... 53
Obr. 4.15 Průběh celkové energie soustavy s poruchou, integrace pomocí RK45, frekvence 2 000 Hz ... 53
Obr. 4.16 FEM model, tvar 1 (vlevo) a tvar 2 (vpravo) ... 54
Obr. 4.17 FEM model, tvar 3 (vlevo) a tvar 4 (vpravo) ... 54
Obr. 4.18 FEM model, tvar 5 (vlevo) a tvar 6 (vpravo) ... 54
Obr. 4.19 FEM model, tvar 7 (vlevo) a tvar 8 (vpravo) ... 55
Obr. 4.20 FEM model, tvar 9 (vlevo) a tvar 10 (vpravo) ... 55
Obr. 4.21 FEM model, tvar 11 (vlevo) a tvar 12 (vpravo) ... 55
Obr. 4.22 FEM model, tvar 13 (vlevo) a tvar 14 (vpravo) ... 55
Obr. 4.23 FEM model, tvar 15 ... 56

Obr. 4.24 Schéma tvorby modelů s různými parametry 57
Obr. 5.1 Tlakový kužel pod šroubem spojujícím části 1 a 2 58
Obr. 5.2 „Odléhání“ dílu 1 od dílu 2 v oblasti mimo tlakový kužel 59
Obr. 5.3 Náčrtek uspořádání experimentu, červeně jsou označeny měřené body, v kruhu čísla dílů 59
Obr. 5.4 Fotografie uspořádání experimentu, 1 – nosník, 4 – seismická hmota, 5 – automatické kladívko 60
Obr. 5.5 Fotografie uspořádání experimentu, 1 – nosník, 2 – podkladový díl, 5 – automatické kladívko 60
Obr. 5.6 Fotografie uspořádání experimentu, 6 – laserový vibrometr 61
Obr. 5.7 Síly impaktů v čase – oranžové body označují nalezená maxima 62
Obr. 5.8 Časový záznam sil rozdělený na jednotlivé impakty (vpravo detail) 62
Obr. 5.9 Časový záznam rychlosti vibrace bodu 4 rozdělený na odezvy jednotlivých impaktů (vpravo detail) 63
Obr. 5.10 Okénko pro přípravu signálů (vpravo detail na počátek okénka) 63
Obr. 5.11 Rychlosti kmitání bodu 4 zpracované okénkem (detail na počátek záznamu) 64
Obr. 5.12 Budící silové impulzy zpracované okénkem (detail počátku záznamu) 64
Obr. 5.13 PSD impaktů, konstrukce bez poruchy, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka 65
Obr. 5.14 PSD kmitání, konstrukce bez poruchy, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka 65
Obr. 5.15 FRF mobility, konstrukce bez poruchy, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka 66
Obr. 5.16 FRF compliance konstrukce bez poruchy z Matlab identification toolbox 66
Obr. 5.17 PSD impaktů, konstrukce s poruchou, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka 67
Obr. 5.18 PSD kmitání, konstrukce s poruchou, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka 67
Obr. 5.19 FRF mobility, konstrukce s poruchou, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka 68
Obr. 5.20 FRF compliance konstrukce s poruchou z Matlab identification toolbox 68
Obr. 5.21 FRF compliance bodu 8 69
Obr. 5.22 Porovnání PSD simulovaných kmitání (vlevo) bez poruchy, (vpravo) s poruchou 69
Obr. 5.23 Porovnání PSD měřených kmitání (vlevo) bez poruchy, (vpravo) s poruchou 70
Obr. 5.24 Detail (Obr. 5.18) na peak vlastní frekvence (vlevo) a „falešnou vlastní frekvenci“ (vpravo) 70
Obr. 6.1 Graf OE modelu pomocí matic stavového popisu (vlevo) a pomocí koeficientů čitatele a jmenovatele přenosu (vpravo) 74
Obr. 6.2 Schéma nelineárního OE modelu 75
Obr. 6.3 (vlevo) Hankelova singulární čísla pro modely daných řádů, (vpravo) FRF ze simulace bez poruchy – modře a rekonstruované pomocí parametrů nalezených LSCE 78
Obr. 6.4 (vlevo) Hankelova singulární čísla pro modely daných řádů, (vpravo) FRF ze simulace s poruchou – modře a rekonstruované pomocí parametrů nalezených LSCE 78
Obr. 6.5 (vlevo) FRF ze simulace bez poruchy – modře a rekonstruované pomocí parametrů nalezených LSCE, (vpravo) FRF ze simulace s poruchou – modře a rekonstruované pomocí parametrů nalezených LSCE 78
Obr. 6.6 Schéma vnitřního a vnějšího optimalizačního cyklu a sdílení dat mezi nimi. 80
Obr. 6.7 Hodnoty tří genů ga při identifikaci dat bez poruchy, tečky odpovídají jednotlivým řešením 81
Obr. 6.8 Hodnoty genů při identifikaci dat bez poruchy v průběhu optimalizace pro všechna řešení 82
Obr. 6.9 Záporná hodnota součtu čtverců odchylky od dat bez poruchy v průběhu optimalizace pro všechna řešení 82
Obr. 6.10 Hodnoty tří genů ga při identifikaci dat s poruchou, tečky odpovídají jednotlivým řešením 83
Obr. 6.11 Hodnoty genů při identifikaci dat bez poruchy v průběhu optimalizace pro všechna řešení 83

Obr. 6.12 Záporná hodnota součtu čtverců odchylky od dat s poruchou v průběhu optimalizace pro všechna řešení... 83
Obr. 6.13 Tabulka s výsledky identifikace modelu na testovacích datech ... 84
Obr. 6.14 ROC křivka pro klasifikaci na základě sumy čtverců G ... 85
Obr. 7.1 Kontrolní zobrazení bezrozměrných vektorů trénovacích dat, označeny příslušnou hodnotou label ... 89
Obr. 7.2 Kontrolní zobrazení bezrozměrných vektorů trénovacích dat (modře) testovaná a (oranžově) referenční, označeny příslušnou hodnotou label ... 90
Obr. 7.3 Průběh loss funkce a accuracy metriky sítě s mnoha dense vrstvami, použitá nevhodná testovací data ... 92
Obr. 7.4 Průběh loss funkce a accuracy metriky pro síť s vrstvami dense a layer_normalization ... 93
Obr. 7.5 ROC křivka pro síť s vrstvami dense a layer_normalization ... 93
Obr. 7.6 Confusion matrix pro síť s vrstvami dense a layer_normalization ... 94
Obr. 7.7 Průběh loss funkce a accuracy metriky pro síť s dvěma vstupy, vrstvami dense a layer_normalization ... 95
Obr. 7.8 ROC křivka pro síť s dvěma vstupy, vrstvami dense a layer_normalization ... 95
Obr. 7.9 Confusion matrix pro model s více vstupy, vrstvou dense a layer_normalization ... 96
Obr. 7.10 Průběh loss funkce a accuracy metriky pro síť s convolution a batch_normalization, nevhodná testovací data ... 97
Obr. 7.11 Průběh loss funkce a accuracy metriky pro síť s vrstvou convolution a layer_normalization ... 98
Obr. 7.12 ROC křivka pro síť s vrstvou convolution a layer_normalization ... 98
Obr. 7.13 Confusion matrix pro síť s vrstvou convolution a layer_normalization ... 99
Obr. 7.14 Confusion matrix pro síť s vrstvou convolution a layer_normalization na trénovacích datech ... 99
Obr. 7.15 Průběh loss funkce a accuracy metriky pro síť s vrstvou separable convolution ... 100
Obr. 7.16 ROC křivka pro síť s vrstvou separable convolution ... 100
Obr. 7.17 Confusion matrix pro síť s vrstvou separable convolution ... 101
Obr. 7.18 Průběh loss funkce a accuracy metriky pro síť se dvěma vstupy a vrstvou separable_convolution ... 102
Obr. 7.19 ROC křivka pro síť se dvěma vstupy a vrstvou separable_convolution ... 102
Obr. 7.20 Confusion matrix pro síť se dvěma vstupy a vrstvou separable_convolution ... 103
Obr. 7.21 Confusion matrix pro síť se dvěma vstupy a vrstvou separable_convolution na trénovacích datech ... 103
Obr. 8.1 Kontrolní zobrazení bezrozměrných vektorů testovacích dat z EMA, označeny příslušnou hodnotou label .. 105
Obr. 8.2 Výsledky pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení ... 106
Obr. 8.3 ROC křivka pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení ... 106
Obr. 8.4 Confusion matrix pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení ... 107
Obr. 8.5 Výsledky pro síť s vrstvou separable convolution na reálných datech z EMA vyhlazených oknem délky 10 . 107
Obr. 8.6 ROC křivka pro síť s vrstvou separable convolution na reálných datech z EMA vyhlazených oknem délky 10 ... 108
Obr. 8.7 Confusion matrix pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení ... 108

Seznam zkratek a symbolů

1D	1 dimenze
2D	2 dimenze
ANN	Neuronové sítě (Artificial neural network)
AR	Autoregresní model (Autoregresive model)
ARX	Autoregresní model (Autoregresive exogenous model)
BDF	Backward differentiation formula – řešič ODE
CE	Complex exponential, metoda zpracování EMA
CNN	Konvoluční neuronová síť (Convolutional neural network)
DNN	Hluboká neuronová síť (Deep neural network)
DOP853	Explicitní Runge-Kuttova metoda řešení ODE řádu 8
EMA	Experimentální modální analýza
FEM	Metoda konečných prvků (Finite element model)
FFT	Rychlá Fourierova transformace (Fast Fourier transform)
FN	Falešně negativní (False negative)
FOH	Tvarovač prvního řádu (First order hold)
FP	Falešně pozitivní (False positive)
FRF	Frekvenční přenosová funkce (Frequency response function)
GA	Genetický algoritmus (Genetic algorithm)
GCN	Grafová konvoluční síť (Graph convolutional network)
IIR	Filtr s nekonečnou odezvou (Infinite impulse response filter)
L-M	Levenberg-Marquardtův algoritmus
LSCE	Least-squares complex exponential
MDOF	Soustava s více stupni volnosti
ODE	Obyčejné diferenciální rovnice (Ordinary differential equation)
OE	Model OE (Output error model)
PCA	Metoda získání příznaků (Principal component analysis)
PSD	Spektrální výkonová hustota (Power spectral density)
RK23	Explicitní Runge-Kuttova metoda řešení ODE řádu 3(2)
RK45	Explicitní Runge-Kutta-Fehlbergova metoda řešení ODE řádu 4(5)
RNN	Rekurentní neuronová síť (Recurent neural network)
ROC	ROC křivka (Receiver operating characteristics)
SDD	Detekce poruchy konstrukce (Structural damage detection)
SDOF	Systém s jedním stupněm volnosti (single degree of freedom)
SIMO	Systém s jendním vstupem a mnoha výstupy (Single input, multiple output)
SISO	Soustava s jedním vstupem a jedním výstupem (Single input, single output)
TN	Skutečně negativní (True negative)
TP	Skutečně pozitivní (True positive)
ZOH	Tvarovač nultého řádu (Zero order hold)

A	[m^2]	Průřez nosníkového elementu
$\underline{\underline{\hat{A}}}$	[různé]	Matice stavového popisu
a_i	[různé]	Konstanta modelu
$\underline{\underline{B}}$	[kg.s^{-1}]	Matice tlumení
$\underline{\underline{\hat{B}}}$	[různé]	Matice stavového popisu
b_{ri}	[-]	Modální tlumení i-té vlastní frekvence
$\underline{\underline{B_p}}$	[s^{-1}]	Matice modálního tlumení
b_i	[různé]	Konstanta modelu
$\underline{b}$	[-]	Bias DNN
$\underline{\underline{\hat{C}}}$	[různé]	Matice stavového popisu
$\underline{\underline{\hat{D}}}$	[různé]	Matice stavového popisu
$\underline{\underline{D_{(\omega)}}}$	[s^2]	Diagonální matice definovaná v textu
d	[-]	Náhodná hodnota násobku rozměrových parametrů
$\underline{\underline{E}}$	[-]	Jednotková matice
E	[Pa]	Youngův modul pružnosti v tahu
E_k	[J]	Kinetická energie
E_p	[J]	Deformační, tedy potenciální energie
E_r	[J]	Energie disipovaná tlumením
$\underline{F}(t,s)$	[různé]	Pravá strana diferenciální rovnice
$\boldsymbol{f}_i^{\,B}$	[N.m^{-3}]	Fyzikální vektor objemové intenzity vnějších sil
$\boldsymbol{f}_i^{\,S}$	[kg.m^{-2}]	Fyzikální vektor plošné intenzity vnějších sil
$\underline{f}$	[N]	Vektor sil, odpovídá vektoru vstupů stavového systému
F	[N.s]	Fourierův obraz f
F_p	[N]	Síla pružiny
f_t	[N]	Časově závislá silová funkce
$\underline{F_r}$	[N]	Síla tlumení
$\hat{g}$	[-]	Číslo globálního uzlu
$\underline{\underline{G_{(\omega)}}}$	[m.N^{-1}]	Frekvenční přenos – dynamická poddajnost
$\underline{\underline{G}}$	[-]	Matice vyjadřující rozdíl mezi dvěma lineárními systémy v parametrech
$G(q)$	[různé]	Přenos diskrétního filtru
$\underline{g}$	[různé]	Gradient
h	[-]	Množina globálních uzlů, na které jsou aplikovány homogenní okrajové podmínky
$\underline{\underline{H}}$	[různé]	Hessián
$\underline{\underline{\hat{H}}}$	[-]	Matice přírůstku tuhosti přidáním pružiny
I	[m^4]	Moment setrvačnosti průřezu

i	[-]	Imaginární jednotka
$i, j, k \ldots$	[-]	V dolním indexu obecné indexy Einsteinovy sumační notace
$\underline{\underline{J}}$	[různé]	Jacobiho matice
k	[-]	Krok diskrétních popisů
k, l	[-]	Index rozlišující systémy bez přidané pružiny (k) s ní (l)
$\underline{\underline{K}}$	[N.m^{-1}]	Matice tuhosti
$\underline{\underline{K^e}}$	[N.m^{-1}]	Matice tuhosti elementu v lokálních souřadnicích
$\underline{\underline{K^e{}_g}}$	[N.m^{-1}]	Matice tuhosti elementu v globálních souřadnicích
k_p	[N.m^{-1}]	Tuhost pružiny
$\underline{\underline{K_l}}$	[N.m^{-1}]	Matice tuhosti s přidanou pružinou
L	[různé]	Ztrátová funkce
l	[m]	Délka nosníkového elementu
$\underline{\underline{M}}$	[kg]	Matice hmotnosti
$\underline{\underline{\hat{M}}}$	[různé]	Matice diskrétního stavového popisu
$\underline{\underline{M^e}}$	[kg]	Matice hmotnosti elementu v lokálních souřadnicích
m	[-]	Řád modelu
n_j	[-]	Vektor jednotkové normály
$\underline{\underline{\hat{N}}}$	[různé]	Matice diskrétního stavového popisu
N_in	[-]	Normalizovaný vstup
N	[-]	Množství vstupních dat
n	[-]	Počet stupňů volnosti
$\underline{\underline{\hat{O}}}$	[různé]	Matice diskrétního stavového popisu
o_i	[-]	Třída dat
$\underline{\underline{\hat{P}}}$	[různé]	Matice diskrétního stavového popisu
$\underline{p}$	[různé]	Vektor parametrů
P_r	[W]	Výkon disipovaný v tlumení
$\underline{p_M}$	[různé]	Parametry příslušející matici M stavového popisu
$\underline{p_N}$	[různé]	Parametry příslušející matici N stavového popisu
$\underline{p_O}$	[různé]	Parametry příslušející matici O stavového popisu
$\underline{q}$	[m.kg$^{1/2}$]	Modální souřadnice
q	[-]	index buzeného směru modelu
q^{-i}	[-]	Operátor posunutí zpět v čase o i diskrétních časových kroků
Q	[s.m.kg$^{1/2}$]	Fourierův obraz modální souřadnice q
$\underline{r}$	[různé]	Vektor odchylek výstupů
$\underline{\underline{R}}$	[-]	Matice transformace gradientu

$\underline{s0}$	[různé]	Vektor počátečních podmínek soustavy diferenciálních rovnic
$\underline{s}$	[různé]	Vektor stavů ve stavovém popisu
$\underline{s_a}$	[různé]	Vektor stavů, které mají být zachovány
$\underline{s_b}$	[různé]	Vektor stavů, které mají být zanedbány
$S\_\_$	[různé]	Spektrální výkonová hustota signálu
$\underline{\underline{T(\varphi)}}$	[-]	Transformační matice z lokálních do globálních souřadnic
$t0$	[s]	Počáteční čas
t	[s]	Čas
t_{end}	[s]	Koncový čas
$\underline{u}$	[m]	Vektor posuvů
u_{il}	[m]	Lokální posuv
$\underline{\boldsymbol{u}}$	[m]	Fyzikální vektor pole posuvů
$\underline{\boldsymbol{u_0}}$	[m]	Počáteční vektor pole posuvů
$\underline{\hat{u}}$	[m]	Amplituda kmitů
u	[-]	Vstup modelu
$\overline{u}$	[m]	Výchylka nemodifikovaného modelu
U	[m.s]	Fourierův obraz u
u_p	[m]	Výchylka v místě pružiny
$\underline{\dot{u}_m}$	[m.s^{-1}]	Rychlost výchylky jednoho místa v konstrukci
$\underline{v}$	[kg$^{-1/2}$]	Vlastní vektor poddajného systému
$\underline{v_{fu}}$	[-]	Vektor označující polohu pružiny pro model s poruchou
$\underline{v_{ft}}$	[-]	Vektor označující polohu budící síly
$\underline{\underline{V}}$	[kg$^{-1/2}$]	Matice vlastních vektorů
$\underline{v_k}$	[-]	Vektor určující polohu pružiny
$\underline{\underline{\mathrm{W}}}$	[-]	Matice jádra DNN
$\underline{w}$	[různé]	Vektor provádějící transformaci ze stavů do u_p
$\underline{\boldsymbol{x}}$	[m]	Fyzikální vektor polohy
x_l, y_l	[m]	Lokální souřadnice
x_G, y_G	[m]	Globální souřadnice
$\underline{y}$	[různé]	Vektor výstupu stavového popisu
$\hat{y}$	[různé]	Predikovaný výstup
$\underline{\underline{Z}}$	[-]	Nulová matice
α	[s^{-1}]	Koeficient Rayleigho tlumení u matice hmotnosti
β	[s]	Koeficient Rayleigho tlumení u matice tuhosti
Δt	[s]	Časový krok
Δ_{ijkl}	[-]	Tenzor 4 řádu definovaný v rovnici

$\underline{\delta_l}$	[m]	Vektor uzlových posuvů elementu v lokálních souřadnicích
$\underline{\delta_g}$	[m]	Vektor uzlových posuvů elementu v globálních souřadnicích
δ_{ij}	[-]	Kroneckerovo delta
η	[-]	Parametr velikosti kroků
Θ	[-]	Oblast reprezentující těleso v kontinuu
λ	[s^{-2}]	Vlastní číslo
$\underline{\underline{\Lambda}}$	[s^{-2}]	Matice vlastních čísel
ρ	[$kg.m^{-3}$]	Hustota
σ_{ij}	[Pa]	Tenzor napětí
φ	[°]	Orientace elementu
$\partial\Theta_\sigma$	[-]	Hranice oblasti Θ, na kterou je aplikována silová okrajová podmínka
$\partial\Theta_u$	[-]	Hranice oblasti Θ, na kterou je aplikována kinematická okrajová podmínka
Ω_c	[Hz]	Maximální frekvence obsažená v signálu
Ω_s	[Hz]	Vzorkovací frekvence
Ω	[s^{-1}]	Vlastní frekvence
ω	[s^{-1}]	Frekvence

Tečka nad symbolem značí derivaci podle času.

V rámci textu jsou pro přehlednost uváděny odkazy na kód vytištěný v příloze, odkazy mají podobu například (ř.123). Uvedený řádek odpovídá číslu řádků v tištěné příloze. Každý přiložený soubor začíná řádkování od 1.

Tato práce mimo jiné využívá terminologii z oblasti strojového učení a knihovny TensorFlow, kde je pojmu tenzor využíváno často jako zobecnění matice pro všechny řády, bez ohledu na matematickou definici tenzoru. Proto kdykoliv je v této práci zmiňován tenzor, může jít o tento „širší“ význam tohoto slova z důvodu nedostatku lepšího obecně uznávaného termínu.

Úvod

Koncept zjišťování poruchy v konstrukci pomocí vibračních vlastností se poprvé vyskytl v 19. století, kdy pracovníci železnice zjišťovali kondici vlakových kol poklepáním kladivem na kolo a poslechem výsledného zvuku [1]. Případná prasklina kola zřejmě ovlivnila „zvonění" kola natolik, že bylo možné detekovat poruchu pouhým uchem. Pokud tuto metodu porovnáme nejenom s těmi, které byly v té době k dispozici, ale i s dnešními běžně užívanými metodami, vyvstane jedna její nesporná výhoda. Jde totiž o metodu globální, která poskytne informaci o poruše kdekoli v součásti. Poruchu sice není možné lokalizovat, ale je možné vyřadit celou součást a nahradit ji. Aby byla tato metoda proveditelná, je zřejmé, že se pracovník musel naučit na více vzorcích, jak zní poškozené a jak nepoškozené kolo. Lze také říci, že je to lidská inteligence, lidské učení, které v tomto případě umožňují úspěšnou identifikaci poruchy.

Přestože od té doby vývoj v dané oblasti značně pokročil, globální metody zjišťování poruch konstrukcí se stále ve velké míře spoléhají na vibrační vlastnosti zkoumaných součástí. Pokud považujeme součástku, konstrukci, či mechanismus za lineární systém, můžeme je popsat pomocí jejich modálních vlastností. Modálními vlastnostmi jsou: vlastní (modální) frekvence, vlastní (modální) tlumení a vlastní tvary kmitu. První dvě uvedené jsou přitom globálními vlastnostmi systému. To například znamená, že stejnou vlastní frekvenci lze zjistit z měření v různých bodech součásti, přitom přidání hmoty v jakémkoli jiném bodě součásti tuto frekvenci může změnit. O vlastnostech celého systému se tak můžeme dozvědět z měření v jednom místě. Tento princip má svá omezení, ale vysvětluje, proč bývají globální metody založeny na frekvenčních vlastnostech.

Na rozdíl od uvedeného příkladu z minulosti dnes umíme velmi přesně zaznamenat vibrace na různých místech součásti v čase, stejně tak jako naměřit silový impuls, kterým jsou vibrace vybuzeny. Umíme tato data zpracovat do formy frekvenční přenosové funkce, která je vysvětlena níže. Ta, s určitou nadsázkou, ukazuje, jak moc bude součást kmitat pro jednotlivé frekvence, které v ní vybudíme. Z této funkce lze odhadovat uvedené modální vlastnosti měřené součásti či konstrukce. Jak již bylo uvedeno, každá změna provedená na součásti může tyto modální vlastnosti ovlivnit, touto změnou může být i porucha na nějakém místě. Pracovník zjišťující, zda je struktura v pořádku, tak může porovnávat současnou podobu frekvenční přenosové funkce, či hodnoty modálních parametrů, s dřívějšími. Pokud zaznamená odlišnost, tato odlišnost může být způsobena poruchou.

Strojové učení, o kterém se zmiňuje již roku 1959 A. L. Samuel [2], se snaží v určitém smyslu napodobit proces učení člověka. Při poskytnutí jistého množství příkladů a správných řešení se naučí určité vzory či pravidla, podle kterých se může rozhodovat příště. Paralelou může být dítě, které se postupným saháním na horká kamna nakonec naučí, že na ně sahat nemá. Pokud se vrátíme k příkladu z prvního odstavce, bylo zde řečeno, že lidské učení umožní identifikaci poškozeného vlakového kola. To vede k myšlence, zda by nemohlo být strojové učení tím, co umožní další krok v automatizaci procesu zjišťování poruch v součástech a strukturách. Tato oblast je v současné době předmětem výzkumu. Automatizace celého procesu by vedla nejen k odstranění lidské chyby. Obrovské množství dat, které je možné v současné době naměřit je pro člověka obtížné pojmout, proto dochází k určitým redukcím – například zjištění vlastních frekvencí. Přitom zpracování hrubých dat může vést k odhalení souvislostí, příznaků na nižší úrovni, které by jinak zanikly. Na tomto místě je vhodné připomenout, že pro uváděné zpracování signálů bylo nutné učinit předpoklad linearity. Ovšem jak je známo, svět je nelineární, poruchy zřejmě nejsou výjimkou, a tak je možné, že nejvýraznější příznaky poruchy se skrývají v hrubých datech předtím, než je zpracování s předpokladem linearity skryje. Uvedenými skutečnostmi se bude zabývat tato diplomová práce s cílem vytvořit metodu pro detekci poruchy konstrukcí.

Rešerše

1 Rešerše existujících metod detekce poruchy

V [1] jsou metody detekce poruchy v konstrukci (Structural damage detection – SDD) rozděleny na lokální a globální metody, kde globální metody jsou založeny na vibracích konstrukce. Pro globální metody je typické, že by měly být schopny zjistit, že konstrukce je porušena, aniž by měření probíhalo v místě poruchy. Porucha totiž ovlivní globální vlastnosti konstrukce, které tyto metody zkoumají.

1.1 Poruchy ocelových konstrukcí

Jelikož poruchu bychom nejspíš mohli chápat jako stav, kdy konstrukce není schopna plnit svůj účel, závisí hodnocení nejčastějších poruch pravděpodobně na účelu dané konstrukce. Základních 6 mechanismů poruchy ocelových konstrukcí uvádí [3] jako:

Překročení meze kluzu, kdy dochází k plastické deformaci materiálu vlivem nadměrného namáhání.

Křehký lom, kde dochází k poruše materiálu bez plastické deformace. Ke křehkému lomu může docházet především při nižších teplotách a v místech koncentrace napětí**.**

Ztrátu stability, obvykle se projevující na součástech s malou tloušťkou, či malým průřezem v poměru k délce. Typickým příkladem může být překročení meze vzpěru u tenkého prutu, které vede k jeho prohnutí a trvalé deformaci či lomu.

Únavu materiálu, kde dochází k iniciaci a šíření prasklin. Ta je způsobena cyklickým namáháním, které může mít i nižší amplitudu, než je mez kluzu materiálu. Prasklina se nakonec může rozšířit do takové míry, že dojde k lomu.

Korozi, která, skrze oxidaci materiálu, způsobí úbytek materiálu. K tomu může docházet jak na povrchu součásti, tak uvnitř ní. Zmenšení průřezu může vést k inicializaci dalších mechanismů poruchy součásti, jako například překročení meze kluzu.

Creep, typický spíše pro plasty. Nicméně může k němu docházet i u oceli, například při požárech budov s ocelovou konstrukcí. Jde o časově závislou deformaci materiálu spojenou s teplotou. I při konstantním zatížení dochází k nárůstu deformace, nadměrná deformace může nakonec vést k destrukci konstrukce.

Při reálné poruše konstrukce dochází ke kombinaci uvedených mechanismů, kromě toho se uvedené mechanismy mohou navzájem spouštět. Uvedené příklady popisují porušení konstrukce na úrovni materiálu součástí. Kromě toho nicméně může docházet například k uvolnění šroubů, uvolnění nýtů a podobně. Přestože jsou tyto poruchy popsány pro ocel, uplatňují se i u dalších materiálů.

Dle [1] je porucha definována jako změna geometrických nebo materiálových charakteristik, která ovlivní bezpečnost, spolehlivost, životnost či výkonnost dané konstrukce či zařízení. Autoři [4] jako poškození chápou poruchu materiálu (praskliny, zlomení, delaminaci, zhroucení), uvolnění spojených dílů (uvolněné šrouby, nýty, lepidlo), ztrátu součásti a ztenčení průřezu.

1.2 Lokální (klasické) metody detekce poruchy konstrukcí

Častým místem výskytu poruch konstrukce jsou svarové spoje. Poruchy mohou vznikat již během procesu svařování, i proto byla vyvinuta řada nedestruktivních metod, používaných k detekci poruch v rámci výrobního procesu. Následující metody uvádí [5]:

Kontrola povrchovou prohlídkou, kde zkušený pracovník prohlíží povrch svaru po jeho očištění. Hledá především známky trhlin a vad spojení.

Kontrola svaru prozářením, při kterém je vytvořen radiogram svaru jeho prozářením. Tlustší materiál bude mít na výsledném obrazu jinou světlost, než tenčí, podobně, jako při rentgenu lidské kosti. Při vhodném nastavení je možné zjistit ve svaru vnitřní trhliny, dále bubliny, póry, vměstky a podobné poruchy ovlivňující množství či hustotu materiálu ve směru záření.

Kontrola svaru ultrazvukem. Často rychlejší a levnější než kontrola prozářením. Pomocí vysílaných impulzů a snímáním jejich odrazů identifikuje poruchy. Porucha totiž vytvoří rozhraní prostředí, od kterého se vlna může odrážet. Ve svaru dobře odhaluje ty vnitřní trhliny, které prozáření odhaluje obtížně.

Kontrola svaru magnetickou metodou práškovou je založená na zmagnetizování zkoušené feromagnetické součásti. Poruchy na povrchu, nebo těsně pod povrchem, se projeví rozptýlením magnetického toku. Toto rozptýlení je viditelné díky pokrytí součásti tekutinou s feromagnetickým práškem, který zvýrazňuje magnetický tok.

Kontrola svaru kapilárními metodami, kde je povrch součásti potřen detekční tekutinou, která pronikne do prasklin a podobných vad na povrchu. Následně je kapalina opláchnuta a nanesena vývojka, která zvýrazní praskliny, ve kterých kontrastní detekční tekutina zůstala. Tím se praskliny zvýrazní pro vizuální kontrolu.

Uvedené metody jsou použitelné pro hledání určitých poruch v konstrukcích i mimo svarové spoje.

1.3 Globální (vibrační) metody detekce poruchy konstrukcí

Dle [1] jsou globální metody detekce poruchy konstrukce často a s úspěchem aplikované na rotující stroje a strojní součásti. Měřeny jsou vibrace součástí, ze kterých se vyhodnocuje kondice ozubených kol, ložisek a podobných součástí. Proces se zakládá na získání „vibračního otisku“ struktury, který je porovnáván s referenčním „otiskem“ struktury bez poruchy. Dle stejného zdroje lze globální SDD rozdělit buď na ty používající strojové učení a nepoužívající strojové učení, nebo na parametrické a neparametrické metody. Parametrické metody jsou založené na zjištění neznámých dynamických parametrů mechanické struktury, tyto parametry nejčastěji představují modální frekvence, modální tlumení, modální tvary kmitu, nebo také tuhost a podobně. Neparametrické metody pro SDD používají statistické prostředky uplatněné na naměřené vibrace.

1.3.1 Parametrické globální SDD

Mezi nejčastější principy parametrického SDD patří získání modálních frekvencí, tvarů kmitu a jejich vyhodnocení pomocí plně propojených vícevrstvých sítí (Artificial neural network ANN) [1].

V [6] je popsána metoda, kde je z provozních dat nejprve pomocí korelace získána impulzní odezva konstrukce, ze které jsou získány modální parametry. Ty jsou následně zpracovány pro určení, kde je v konstrukci porucha. Metoda je vyvinuta speciálně pro sloupy vysokého napětí a

autoři tvrdí, že je vhodné, aby byla metoda SDD vyvinuta speciálně pro každou konkrétní konstrukci. Pro simulaci poškození byla použita redukce tuhosti o 80 % na jednom z nosníků.

Autoři [7] představili koncept zařízení, které by bylo připevněno ke sloupům pouličního osvětlení a získávalo by modální frekvence z kmitání vybuzeného vnějšími vlivy. Pozorováním postupné změny modálních frekvencí by podle autorů bylo možné odhadovat možnou poruchu konstrukce.

1.3.2 Neparametrické globální SDD

Neparametrické globální SSD je založeno na extrakci příznaků statistickými metodami, jako jsou (obvykle nazývány anglicky): signal variance, autoregressive modeling, principal component analysis (PCA), wavelet transform a podobné [8].

V [9] autoři použili Autoregressive exogenous model (ARX) na časové záznamy vibrací pro získání parametrů, které porovnávali s parametry nepoškozené konstrukce, aby zjistili přítomnost a přibližnou polohu poruchy.

Autoři [4] uvádí, že většina pokusů o SSD předcházející jejich práci, se zakládala na updatování FEM modelu podle naměřených dat a následném vyhodnocení poruchy z FEM modelu. Myšlenka byla taková, že lokální ztráta tuhosti v daném místě reálné struktury donutí algoritmus updatování FEM modelu snížit tuhost ve stejném místě i u modelu. Tato metoda byla založena na modálních parametrech konstrukce. Nicméně, jak [4] upozorňuje, jedná se o nepřímo měřené veličiny, ovlivněné nejen chybou měření, ale i chybou identifikace parametrů, která může být značná. Samotná práce se zabývá představením metody vyhodnocení polohy poruchy přímo z FRF dat pomocí algoritmu založeném na sledování odlišnosti naměřené FRF od analytické. Porucha je modelována jako 1 mm široký řez jednoho z nosníků roštu, vedoucí ke ztrátě tuhosti o 51 %.

Autoři [10] tvrdí, že hlavním problémem FRF dat je jejich velikost, proto provádějí PCA FRF dat, aby získali příznaky vystihující tvar FRF. Studují vliv prasklin na kola vlaků. Klíčové pro praktickou část této práce je, že uvádí pozorování většího množství peaků s menší výrazností u FRF rozbitých kol. Nicméně práce dále uvádí, že výrazné jsou také rozdíly ve FRF pro jednotlivé měření.

V [11] je zaznamenán pokus o hledání poruchy ve formě delaminace kompozitního nosníku. Delaminace u kompozitů nemusí být viditelná, ale vede k výrazné ztrátě mechanických vlastností. Práce používá metodu podobnou PCA pro získání příznaků z FRF. Jsou provedeny měření pro delaminace v různých místech nosníku. Jednotlivé pokusy jsou poté zobrazeny v prostoru dvou příznaků získaných z FRF, z grafů je zřejmé kumulování stejných druhů poruch u sebe. Další práci s těmito příznaky ponechává práce dalšímu výzkumu. Ke stanovení příznaků i testování metody dochází ovšem na stejných nosnících.

Práce [12] používá identifikaci autoregresních (AR) modelů na vibračních datech v časové oblasti. Parametry AR modelu jsou použity jako příznaky. Příznaky jsou dále vyhodnocovány různými metodami. Na měřené struktuře je uměle připravena nelineární porucha jako nastavitelná mezera mezi částí konstrukce a dorazem připevněným k rámu. Trénování probíhá pouze na konstrukci bez poruchy. Validace následně probíhá na datech s poruchou i bez poruchy. Algoritmus tak poruchu rozpoznává jako odlišnost od trénovacích dat.

1.3.3 Globální SDD založené na hlubokém učení (Deep learning)

Jak uvádí [1], neexistuje žádná záruka, že vybraný příznak, či vybraná statistická metoda fungující na jeden druh konstrukce, bude dobře fungovat i na jiné konstrukci. To samé platí i pro různé druhy poruchy konstrukce. Výhodou DNN je, že mohou být aplikovány přímo na surová naměřená data a pokud mají k dispozici správná trénovací data, naleznou vhodné příznaky samostatně.

Stejný zdroj také uvádí, že většina studií založených na strojovém učení vyjma DNN, se zaměřuje na použití modálních charakteristik, přičemž nedávné studie ukázali, že tyto parametry mohou být silně ovlivněny teplotou, vlhkostí a podobně, navíc mohou mít malou citlivost k některým druhům poruch.

Autoři [13] navázali na úspěšné použití konvolučních neuronových sítí (Convolutional neural network – CNN) pro SDD a představili aplikaci Graph convolutional network (GCN), kde se podle autorů neztrácí informace o poloze vstupních dat ze senzorů rozmístěných na konstrukci. Porucha byla simulována jako lokální ztráta tuhosti jednoho elementu 2D FEM modelu nosníku a poté také jako redukce průřezu jednoho z nosníků modelu budovy.

V [8] je představena metoda SDD pomocí CNN, kde trénovací data tvoří jen měření dvou struktur, jedné nerozbité, druhé rozbité s mnoha poruchami. Tato měření jsou rozsekána na kratší úseky, aby uspokojila potřebu CNN pro velké množství trénovacích dat. Verifikace pak proběhla pomocí naměření stejné struktury s menším množstvím poruch, výstupem potom je pravděpodobnost, že byla naměřena rozbitá či nerozbitá konstrukce. Ani trénování, ani verifikace nepoužívá simulovaná data, trénování i verifikace probíhá na stejné konstrukci.

2 Úvod do použitých metod a teorie

2.1 FEM model

Jak uvádí [14], metoda konečných prvků (FEM) je způsob diskretizace kontinua na podoblasti (konečné prvky). Celková potenciální energie je součtem potenciálních energií těchto podoblastí. Řešení (hledání rovnováhy) je potom provedeno pomocí hledání minima celkové potenciální energie soustavy.

Pohybové rovnice kontinua jsou v Lagrangeovském vyjádření uvedeny na *(2-1)*, *(2-2)* pak uvádí silové okrajové podmínky a *(2-3)* kinematické okrajové podmínky. Počáteční podmínky jsou dvě z následujících: $\underline{\boldsymbol{u}}\,(\underline{\boldsymbol{x}},0)$, $\underline{\dot{\boldsymbol{u}}}\,(\underline{\boldsymbol{x}},0)$, $\underline{\ddot{\boldsymbol{u}}}\,(\underline{\boldsymbol{x}},0)\ \forall\underline{\boldsymbol{x}}\in\Theta$ [14]. Použité značení je shrnuto v seznamu použitých symbolů. Po aplikaci metody konečných prvků na uvedenou úlohu dostáváme podobu rovnice *(2-4)*, odpovídajícím způsobem by došlo také k diskretizaci okrajových a počátečních podmínek. Člen s maticí M přestavuje síly od setrvačnosti, člen s maticí B představuje síly od tlumení v systému a člen s maticí K představuje síly od tuhosti materiálu. Je zřejmé, že koeficienty matic M, B, K obecně mohou záviset na čase a posuvu uzlů, tedy stavu systému. Rovnice *(2-4)* tak představuje soustavu nelineárních obyčejných diferenciálních rovnic.

$$\frac{\partial\sigma_{ij}\,(\underline{\boldsymbol{x}},\ \underline{\dot{\boldsymbol{x}}},\ t)}{\partial x_j}=f_i^{\,B}(\underline{\boldsymbol{x}},t)-\rho(\underline{\boldsymbol{x}},t)\frac{\partial^2u_i}{\partial t^2}\qquad\forall\underline{\boldsymbol{x}}\in\Theta \tag{2-1}$$

[14]

$$\sigma_{ij}\,(\underline{\boldsymbol{x}},\ \underline{\dot{\boldsymbol{x}}},\ t)\,n_j=f_i^{\,S}(\underline{\boldsymbol{x}},t)\qquad\forall\underline{\boldsymbol{x}}\in\partial\Theta_\sigma \tag{2-2}$$

[14]

$$\underline{\boldsymbol{u}}(\underline{\boldsymbol{x}},t)=\underline{\boldsymbol{u_0}}(\underline{\boldsymbol{x}},t)\qquad\forall\underline{\boldsymbol{x}}\in\partial\Theta_u \tag{2-3}$$

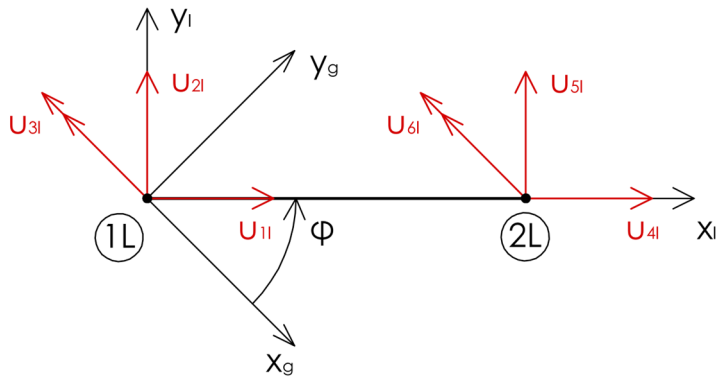
[14]

$$\underline{\underline{M}}(\underline{u},t)\,\underline{\ddot{u}}(t)+\underline{\underline{B}}(\underline{u},t)\,\underline{\dot{u}}(t)+\underline{\underline{K}}(\underline{u},t)\ \underline{u}(t)=\underline{f}(t) \tag{2-4}$$

[14]

Uvedenou aplikací metody konečných prvků tak dochází k diskretizaci úlohy v prostoru. Z hlediska času zůstává úloha spojitou.

Úloha určená k řešení pomocí FEM může být definovaná mnoha způsoby z hlediska dimenzionality úlohy i dimenzionality a typu použitých elementů. Dále bude věnována pozornost rovinné úloze s nosníkovými elementy a konstantními maticemi *M, B, K,* tedy lineární úloze. Odpovídající element je znázorněn na obrázku Obr. 2.1. Element má dva lokální uzly označené 1L a 2L. Definujeme vektor uzlových posuvů v lokálních souřadnicích x_l, y_l dle *(2-5)* se značením odpovídajícím obrázku, tedy u_{3l} a u_{6l} jsou natočení v místě odpovídajícího uzlu. Tomuto vektoru odpovídá matice tuhosti elementu v *(2-6)* a matice hmotnosti elementu v *(2-7)*.



Obr. 2.1 Nosníkový element pro rovinnou úlohu

$$\underline{\delta_l} = [u_{1l},\ u_{2l},\ u_{3l},\ u_{4l},\ u_{5l},\ u_{6l}]^T \tag{2-5}$$

$$\underline{\underline{K^e}} = \begin{bmatrix} \frac{EA}{l} & 0 & 0 & \frac{-EA}{l} & 0 & 0 \\ 0 & \frac{12\,EI}{l^3} & \frac{6\,EI}{l^2} & 0 & \frac{-12\,EI}{l^3} & \frac{6\,EI}{l^2} \\ 0 & \frac{6\,EI}{l^2} & \frac{4\,EI}{l} & 0 & \frac{-6\,EI}{l^2} & \frac{2\,EI}{l} \\ \frac{-EA}{l} & 0 & 0 & \frac{EA}{l} & 0 & 0 \\ 0 & \frac{-12\,EI}{l^3} & \frac{-6\,EI}{l^2} & 0 & \frac{-12\,EI}{l^3} & \frac{-6\,EI}{l^2} \\ 0 & \frac{6\,EI}{l^2} & \frac{2\,EI}{l} & 0 & \frac{-6\,EI}{l^2} & \frac{4\,EI}{l} \end{bmatrix} \tag{2-6}$$

[14]

$$\underline{\underline{M^e}} = \frac{A\rho l}{420}\begin{bmatrix} 140 & 0 & 0 & 70 & 0 & 0 \\ 0 & 156 & 22l & 0 & 54 & -13l \\ 0 & 22l & 4l^2 & 0 & 13l & -3l^2 \\ 70 & 0 & 0 & 140 & 0 & 0 \\ 0 & 54 & 13l & 0 & 156 & -22l \\ 0 & -13l & -3l^2 & 0 & -22l & 4l^2 \end{bmatrix} \tag{2-7}$$

[15]

$$\underline{\delta_g} = \underline{\underline{T(\varphi)}}\,\underline{\delta_l} \tag{2-8}$$

Následně hledáme transformační matici T provádějící transformaci z lokálního do globálního souřadnicového systému x_G, y_G podle *(2-8)*. Tato matice je definována v *(2-9)*. Potom transformace matice tuhosti elementu z lokálních souřadnic do globálních je uvedena v *(2-10)*, obdobným způsobem se transformuje i matice hmotnosti.

$$\underline{\underline{T(\varphi)}} = \begin{bmatrix} \cos(\varphi) & sin(\varphi) & 0 & 0 & 0 & 0 \\ -sin(\varphi) & \cos(\varphi) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(\varphi) & sin(\varphi) & 0 \\ 0 & 0 & 0 & -sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{matrix} (2\text{-}9) \\ [15] \end{matrix}$$

$$\underline{\underline{K^e{}_g}} = \underline{\underline{T(\varphi)^T}}\ \underline{\underline{K^e}}\ \underline{\underline{T(\varphi)}} \quad \begin{matrix} (2\text{-}10) \\ [15] \end{matrix}$$

Modelovaný systém bude tvořen propojenými elementy. Hledáme tedy způsob, jak popsat rozložení elementů v prostoru, jejich propojení, a tím i vztah mezi lokálním a globálním vektorem uzlových posuvů. Tyto požadavky splní mapovací funkce *(2-11)*, která i-tému uzlu e-tého elementu modelu přiřadí globální uzel $\hat{g}$ dle tabulky incidencí. Ta popisuje rozložení elementů modelu a vzniká při sestavování modelu. Proces sestavení globální matice tuhosti je pak popsán jako vztah pro algoritmus v *(2-12)* v indexovém zápisu. Tato operace se opakuje pro všechny i a j a pro všechny elementy e uvedené v incidenční tabulce, tedy pro všechny elementy modelu. Globální matice hmotnosti je sestavována obdobným způsobem.

$$G^e(i) = \hat{g} \quad \begin{matrix} (2\text{-}11) \\ [14] \end{matrix}$$

$$K_{G^e(i)\,G^e(j)} = K_{G^e(i)\,G^e(j)} + K^e{}_{g_{ij}} \quad \begin{matrix} (2\text{-}12) \\ [14] \end{matrix}$$

$$u_i = 0 \qquad \forall i \in h \quad \begin{matrix} (2\text{-}13) \\ [14] \end{matrix}$$

Pokud jsou definovány homogenní kinematické okrajové podmínky ve formě *(2-13)*. Zavedením homogenních kinematických okrajových podmínek pak dochází k takzvané modifikaci soustavy, kdy jsou z matice K vyjmuty i-té řádky a sloupce, stejně tak jsou vyjmuty i-té prvky z vektoru $\underline{u}$ [14]. Nově se v modifikovaném vektoru $\underline{u}$ nacházejí pouze neznámé posuvy. Obdobným způsobem je modifikována matice hmotnosti.

Dosud nebyla diskutována tvorba matice B. Její podoba závisí na zvoleném modelu tlumení soustavy. Jedním z modelů je Rayleigho tlumení, které patří mezi proporcionální tlumení, diskutované v dalších kapitolách. Je definováno jako lineární kombinace matic tuhosti a hmotnosti pouze pomocí dvou parametrů α, β dle *(2-14)*.

$$\underline{\underline{B}} = \alpha\ \underline{\underline{M}} + \beta\ \underline{\underline{K}} \quad \begin{matrix} (2\text{-}14) \\ [15] \end{matrix}$$

2.2 Energie ve FEM modelu

Kinetická energie systému FEM modelu vychází jako kvadratická forma popsaná v *(2-15)*, potenciální energie, tedy deformační energie, vychází obdobně *(2-16)*.

$$E_k = \frac{1}{2}\,\underline{\dot{u}}^{\,T}\,\underline{\underline{M}}\,\underline{\dot{u}} \qquad \textit{(2-15)}\ \textit{[15]}$$

$$E_p = \frac{1}{2}\,\underline{u}^{T}\,\underline{\underline{K}}\,\underline{u} \qquad \textit{(2-16)}\ \textit{[15]}$$

2.3 Řešení ODE v Python SciPy

Numerické řešení systému obyčejných diferenciálních rovnic s počátečními podmínkami je v Pythonu možné získat pomocí knihovny SciPy, která obsahuje funkci integrate.solve_ivp() [16]. Vstupem funkce je pravá strana vztahu $d\underline{s}/dt = \underline{F}(t, \underline{s})$, počáteční podmínka $\underline{s}(t0) = \underline{s0}$ a interval $[t0,\ t_{end}]$. Zde t označuje nezávislou proměnnou, obvykle čas, $\underline{s}$ označuje stavový vektor systému. Mezi volitelné vstupy patří t_eval, vektor hodnot nezávislé proměnné, ve které mají být stavy systému vyhodnoceny. Dále je také možné poskytnout řešiči Jacobiho matici ve formě *(2-17)*. Pokud Jacobiho matice není poskytnuta jako vstup, pro některé metody řešiče ODE je automaticky vypočítávána numericky. Je ovšem doporučováno ji poskytnout pokud je to možné, numerický výpočet může zpomalit celé řešení [16]. Důležitým volitelným vstupem je metoda řešení ODE. Obecně lze říci, že pro řešení stiff problémů jsou doporučovány implicitní metody (zde nabízené: Radau a BDF), pro nonstiff úlohy jsou doporučovány explicitní metody Runge-Kutta (zde nabízené: RK23, RK45 a DOP853) [16]. Definice stiff úloh není zcela jednoznačně přijatá, první dle [17] zněla, že stiff úlohy jsou takové, kde implicitní schémata, zejména BDF, vykazují mnohem lepší výsledky, než explicitní. Lze tedy nejspíš říci, že jde o úlohy, které ze své podstaty vyžadují jiné než explicitní metody, jelikož ty se mohou například ukázat jako nestabilní. Dle stejného zdroje to, jestli úloha bude stiff, může ovlivnit poměr mezi nejmenším a největším vlastním číslem úlohy, velké rozdíly vedou k nutnosti použití stiff řešičů.

$$\underline{\underline{J}}\,(t, \underline{s}) = \partial \underline{F}/\partial \underline{s} \qquad \textit{(2-17)}$$

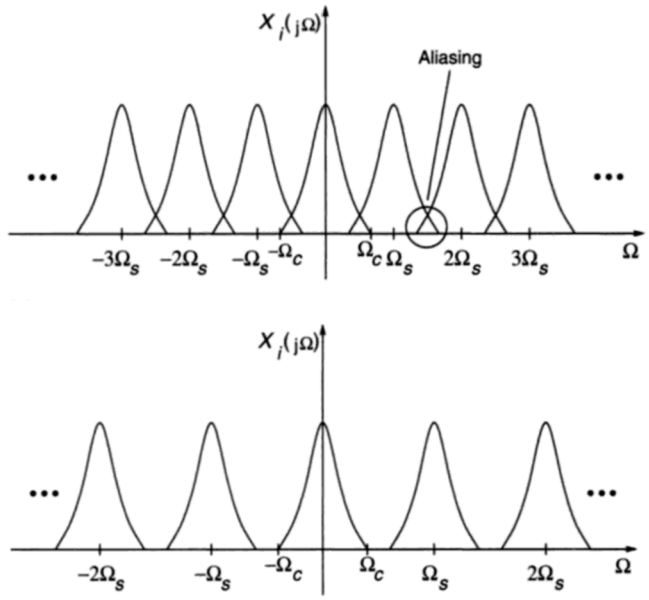
RK45 je explicitní metoda Runge-Kutta-Fehlbergova 4(5), což znamená, že metoda je 4. řádu přesnosti, kde je použit odhad chyby pomocí 5. řádu přesnosti k automatické volbě variabilního kroku [18]. Tato metoda je v současnosti velmi používaná a je doporučována jako první volba [16].

BDF (Backwork differential formulas) je implicitní vícekroková metoda proměnného řádu od 1 do 5 [16]. Tato metoda má velké oblasti stability a hodí se pro stiff úlohy [17]. Důležitou vlastností pro tuto práci také je, že vykazuje numerické tlumení, tedy spektrální radius metody je pro vysoké frekvence nižší jak jedna [19]. Zjednodušeně řečeno to znamená, že pokud touto metodou bude numericky řešena počáteční podmínkou vybuzená vibrace hmoty na netlumené pružině,

analytickým řešením bude nekonečná vibrace, ovšem při použití BDF by numerický výsledek byl nakonec utlumen do klidové polohy.

2.4 Aliasing

Vzorkovací teorém zní: Pokud spojitý signál neobsahuje vyšší frekvenční složky než Ω_c, pak jej lze plně obnovit z diskrétně navzorkovaného signálu pouze, pokud byla frekvence vzorkování $\Omega_s > 2 \cdot \Omega_c$ [20]. V opačném případě dochází k takzvanému aliasingu znázorněnému na Obr. 2.2. Spojitému signálu přísluší jeden peak ve středu zobrazených grafů, ostatní peaky ve vyšších frekvencích jsou projevem diskrétního vzorkování [20]. Je zřejmé, že pokud není dodržen uvedený vzorkovací teorém, dochází k překrytí (graf na obrázku výše) a informace obsažená v signálu je poškozena aliasingem. Aliasing bude později v této práci třeba řešit při numerické integraci.



Obr. 2.2 Spektrální výkonová hustota vzorkovaného signálu s aliasingem (výše) a bez (níže) [20]

2.5 Redukce modelu

Mějme poddajnou strukturu modelovanou ve formě stavového popisu *(2-18)*, kde $\underline{s}$ je vektor obsahující stavy systému. Pokud stavy rozdělíme na dva vektory $\underline{s_a}$ a $\underline{s_b}$ bez ohledu na pořadí stavů v původním vektoru, můžeme systém redukovat vypuštěním stavů $\underline{s_b}$. Toho lze dle [21] dosáhnout buď oříznutím $\underline{s_b}$ – anglicky truncation, kde postavíme $\underline{s_b} = \underline{0}$, nebo rezidualizací – anglicky residualization, match DC, či singular perturbation aproximation, kde postavíme $\underline{\dot{s}_b} = \underline{0}$. Název match DC napovídá, že při použití této metody je zachována DC složka přenosu systému. Spolu s rozdělením vektoru stavů, lze rozdělit i matice stavového systému dle *(2-19)*. Potom redukovaný systém získaný pomocí truncation je uveden v *(2-20)*. Z uvedeného lze odvodit, že model redukovaný pomocí residualization vychází ve tvaru *(2-21)*. V [21] je dále uvedena metoda volby stavového popisu poddajné struktury tak, aby se výpočet pomocí residualization výrazně zjednodušil z hlediska výpočetní náročnosti, zejména při velkém množství vyjmutých stavů. Úkolem redukce modelu je vytvořit jednodušší model, který vhodně aproximuje původní model a umožňuje například rychlejší výpočty.

$$\dot{\underline{s}} = \hat{\underline{\underline{A}}}\,\underline{s} + \hat{\underline{\underline{B}}}\,\underline{f}$$
$$\underline{y} = \hat{\underline{\underline{C}}}\,\underline{s} + \hat{\underline{\underline{D}}}\,\underline{f} \qquad \textit{(2-18)}$$

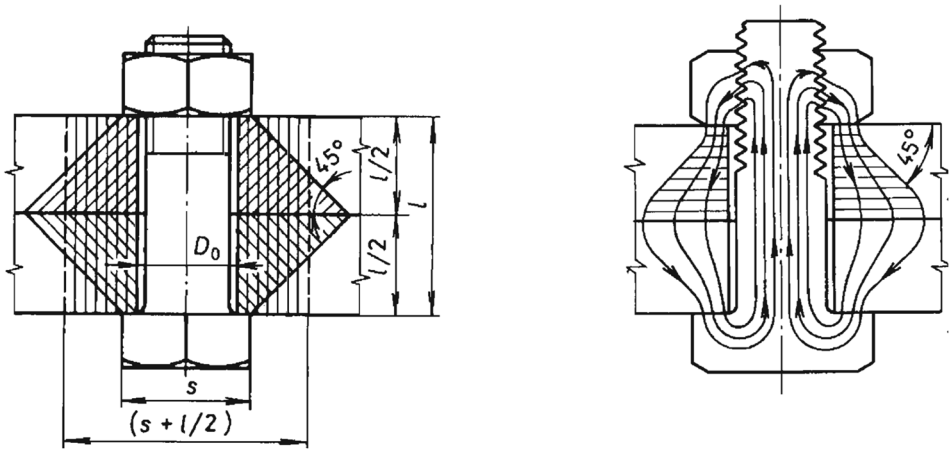
$$\begin{bmatrix} \dot{\underline{s}}_a \\ \dot{\underline{s}}_b \end{bmatrix} = \begin{bmatrix} \widehat{\underline{\underline{A_{aa}}}} & \widehat{\underline{\underline{A_{aa}}}} \\ \widehat{\underline{\underline{A_{aa}}}} & \widehat{\underline{\underline{A_{aa}}}} \end{bmatrix} \begin{bmatrix} \underline{s}_a \\ \underline{s}_b \end{bmatrix} + \begin{bmatrix} \widehat{\underline{\underline{B_a}}} \\ \widehat{\underline{\underline{B_b}}} \end{bmatrix} \underline{f}$$
$$\underline{y} = \begin{bmatrix} \widehat{\underline{\underline{C_a}}} & \widehat{\underline{\underline{C_b}}} \end{bmatrix} \begin{bmatrix} \underline{s}_a \\ \underline{s}_b \end{bmatrix} + \hat{\underline{\underline{D}}}\,\underline{f} \qquad \textit{(2-19) [21]}$$

$$\dot{\underline{s}}_a = \widehat{\underline{\underline{A_{aa}}}}\,\underline{s}_a + \widehat{\underline{\underline{B_a}}}\,\underline{f}$$
$$\underline{y} = \widehat{\underline{\underline{C_a}}}\,\underline{s}_a + \hat{\underline{\underline{D}}}\,\underline{f} \qquad \textit{(2-20) [21]}$$

$$\dot{\underline{s}}_a = \left(\widehat{\underline{\underline{A_{aa}}}} - \widehat{\underline{\underline{A_{ab}}}}\,\widehat{\underline{\underline{A_{bb}}}}^{-1}\,\widehat{\underline{\underline{A_{ba}}}}\right)\underline{s}_a + \left(\widehat{\underline{\underline{B_a}}} - \widehat{\underline{\underline{A_{ab}}}}\,\widehat{\underline{\underline{A_{bb}}}}^{-1}\,\widehat{\underline{\underline{B_b}}}\right)\underline{f}$$
$$\underline{y} = \left(\widehat{\underline{\underline{C_a}}} - \widehat{\underline{\underline{C_b}}}\,\widehat{\underline{\underline{A_{bb}}}}^{-1}\,\widehat{\underline{\underline{A_{ba}}}}\right)\underline{s}_a + \left(\hat{\underline{\underline{D}}} - \widehat{\underline{\underline{C_b}}}\,\widehat{\underline{\underline{A_{bb}}}}^{-1}\,\widehat{\underline{\underline{B_b}}}\right)\underline{f} \qquad \textit{(2-21) [21]}$$

2.6 Tlakový kužel pod šroubem

Komolým kuželem lze aproximovat oblast zasaženou tlakovým napětím vyvolaným sevřením šroubu a matice. Úhel komolého kužele se pohybuje okolo 45° [22], metoda se používá k odhadu tuhosti spoje, nicméně lze na základě ní odhadnout, v jaké vzdálenosti od šroubového spoje bude šroub zajišťovat těsné sevření součástí. Toho bude později využito pro návrh experimentu.



Obr. 2.3 Tlakový kužel ve svěrném šroubovém spoji [22]

2.7 Modální vlastnosti konstrukce

Pohyb diskrétní lineární mechanické soustavy s více stupni volnosti (MDOF) bývá obvykle popsán v následujícím maticovém tvaru, uvedeném mimo jiné i v [23, 24]:

$$\underline{\underline{M}}\,\underline{\ddot{u}} + \underline{\underline{B}}\,\underline{\dot{u}} + \underline{\underline{K}}\,\underline{u} = \underline{f} \qquad \textit{(2-22)}$$

Tvar rovnice odpovídá *(2-4)*, rovnice představuje rovnováhu silových účinků pro jednotlivé součásti soustavy. Matice M představuje matici hmotnosti soustavy, matice B představuje matici tlumení soustavy a matice K představuje matici tuhosti soustavy. Vektor $\underline{u}$ představuje polohu jednotlivých součástí soustavy (jde o výchylku v čase) a vektor $\underline{f}$ představuje vnější silové účinky působící na soustavu, závisí na čase. Časová derivace je znázorněna tečkou nad proměnnou. Pro lineární, časově neproměnnou soustavu s viskózním tlumením lze prohlásit, že matice M, B a K jsou konstantní, nezávisí na čase t, ani na výchylce $\underline{u}$. Netlumená soustava bez buzení pak vypadá následovně:

$$\underline{\underline{M}}\,\underline{\ddot{u}} + \underline{\underline{K}}\,\underline{u} = \underline{0} \qquad \textit{(2-23)}$$

Předpokládáme řešení soustavy ve tvaru:

$$\underline{u} = \underline{\hat{u}} e^{i\Omega t} \qquad \textit{(2-24)}\ \textit{[23]}$$

Po dosazení *(2-24)* do *(2-23)* dostáváme:

$$\left(\underline{\underline{K}} - \lambda \underline{\underline{M}}\right) \underline{v} = \underline{0} \quad \text{kde: } \lambda = \Omega^2 \qquad \textit{(2-25)}\ \textit{[23]}$$

Což je úloha vlastních čísel, jejímž řešením jsou vlastní čísla λ a vlastní vektory $\underline{v}$, jinak také vlastní tvary kmitu. Pro soustavu o n stupních volnosti získáme n vlastních čísel a vlastních vektorů. Potom lze definovat matici vlastních čísel Λ a modální matici V, kde jsou vlastní vektory normovány tak, aby splňovali *(2-28)*. Potom platí také vztah *(2-29)*.

$$\underline{\underline{\Lambda}} = diag([\lambda_1,\ \lambda_2, \ldots\ \lambda_n]) \qquad \textit{(2-26)}$$

$$\underline{\underline{V}} = [\underline{v}_1,\ \underline{v}_2, \ldots\ \underline{v}_n] \qquad \textit{(2-27)}$$

$$\underline{\underline{V^T}}\,\underline{\underline{M}}\,\underline{\underline{V}} = \underline{\underline{E}} \qquad \textit{(2-28)}\ \textit{[23]}$$

$$\underline{\underline{V^T}}\,\underline{\underline{K}}\,\underline{\underline{V}} = \underline{\underline{\Lambda}} \qquad \textit{(2-29)}\ \textit{[23]}$$

Proporcionální tlumení je definováno tak, že platí *(2-30)*, kde b_{ri} je modální tlumení i-té vlastní frekvence Ω_i. Potom je možné definovat modální souřadnici q a pro proporcionální tlumení provést úpravu původního vztahu *(2-22)* do tvaru *(2-32)*. Výsledná rovnice se potom rozpadá na soustavu samostatných rovnic, kde i-tá rovnice je vypsána v *(2-33)*.

$$\underline{\underline{V^T}}\,\underline{\underline{B}}\,\underline{\underline{V}} = diag([2b_{r1}\Omega_1,\ 2b_{r2}\Omega_2, \dots\ 2b_{rn}\Omega_n]) = \underline{\underline{B_p}} \qquad \textit{(2-30)}\ \textit{[23]}$$

$$\underline{\underline{V}}\,\underline{q} = \underline{u} \qquad \textit{(2-31)}\ \textit{[23]}$$

$$\begin{gathered}\underline{\underline{V^T}}\,\underline{\underline{M}}\,\underline{\underline{V}}\,\underline{\ddot{q}} + \underline{\underline{V^T}}\,\underline{\underline{B}}\,\underline{\underline{V}}\,\underline{\dot{q}} + \underline{\underline{V^T}}\,\underline{\underline{K}}\,\underline{\underline{V}}\,\underline{q} = \underline{\underline{V^T}}\,\underline{f} \\ \underline{\ddot{q}} + \underline{\underline{B_p}}\,\underline{\dot{q}} + \underline{\underline{\Lambda}}\,\underline{q} = \underline{\underline{V^T}}\,\underline{f}\end{gathered} \qquad \textit{(2-32)}\ \textit{[23]}$$

$$\ddot{q}_i + 2b_{ri}\Omega_i\dot{q}_i + {\Omega_i}^2 q_i = \underline{v_i}^T\,\underline{f} \qquad \textit{(2-33)}\ \textit{[23]}$$

Je tedy zřejmé, že řešení v časové oblasti je potom možné provádět pro každou modální souřadnici zvlášť. Řešení ve skutečných výchylkách $\underline{u}$ získáme dle vztahu *(2-31)*. Provádíme tak vlastně transformaci z prostoru modálních souřadnic do prostoru souřadnic *u*.

Provedením Fourierovy transformace rovnice *(2-33)* získáváme *(2-34)*, Fourierovy obrazy *u, f, q* jsou znázorněny velkými písmeny *U, F, Q*.

$$\left(-\omega^2 + 2i\omega b_{ri}\Omega_i + {\Omega_i}^2\right) Q_{i_{(\omega)}} = \underline{v_i}^T\,\underline{F_{(\omega)}} \qquad \textit{(2-34)}\ \textit{[23]}$$

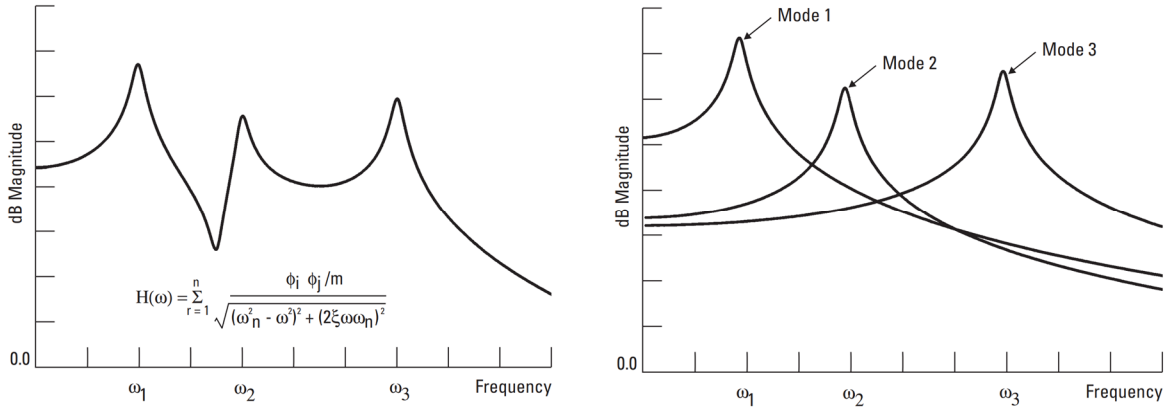
Lze definovat diagonální matici *D* podle *(2-35)*, dosazením do *(2-34)* při použití *(2-31)* získáme *(2-36)*.

$$\underline{\underline{D_{(\omega)}}} = diag([\left(-\omega^2 + 2i\omega b_{r1}\Omega_1 + {\Omega_1}^2\right), \dots \left(-\omega^2 + 2i\omega b_{rn}\Omega_n + {\Omega_n}^2\right)]) \qquad \textit{(2-35)}\ \textit{[23]}$$

$$\underline{U_{(\omega)}} = \underline{\underline{V}}\,\underline{\underline{D_{(\omega)}}^{-1}}\,\underline{\underline{V^T}}\,\underline{F_{(\omega)}} \qquad \textit{(2-36)}\ \textit{[23]}$$

Je tak možné definovat frekvenční přenos *G* mezi silovým vstupem a výstupem ve tvaru vektoru výchylek na *(2-37)*. Z tohoto tvaru vyplývá, že přenos soustavy lze vyjádřit jako součet příspěvků od jednotlivých vlastních frekvencí, jak je také znázorněno na Obr. 2.4.

$$\underline{\underline{G_{(\omega)}}} = \underline{U_{(\omega)}}\ \underline{F_{(\omega)}}^{-1} = \underline{\underline{V}}\ \underline{\underline{D_{(\omega)}}}^{-1}\ \underline{\underline{V^T}} \qquad (2\text{-}37)$$



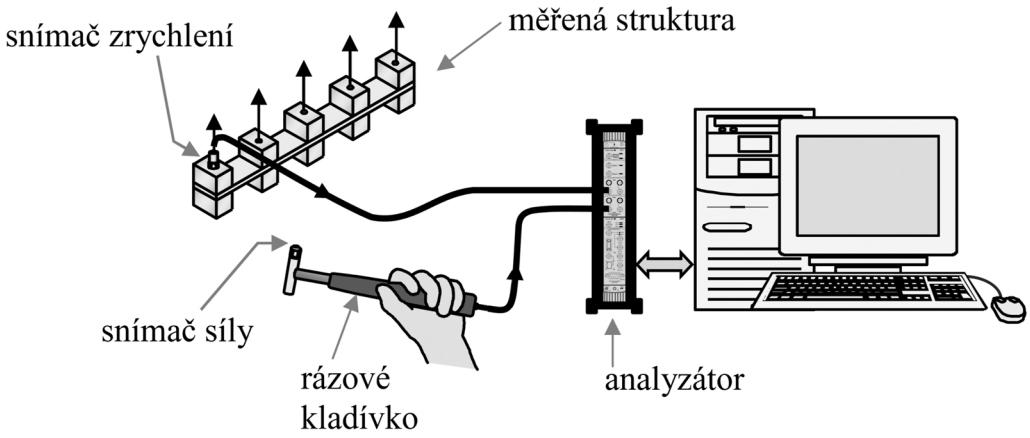
Obr. 2.4 Frekvenční přenos složen z jednotlivých modálních příspěvků [24]

Jak je uvedeno v [23, 24], parametry popisující jednotlivé módy, tedy vlastní frekvence a modální tlumení, jsou globální vlastnosti celé soustavy. Vlastní tvar kmitu příslušející danému modálnímu příspěvku určuje velikost modálního příspěvku v závislosti na poloze buzení a poloze měření výstupu. Při znalosti modálních parametrů soustavy je tak možné kompletně popsat dynamické chování lineární mechanické soustavy. Diskretizace soustavy představuje určitou míru aproximace, součást tvořená kontinuem má nekonečné množství stupňů volnosti, tedy nekonečné množství modálních příspěvků.

2.8 Postup provedení měření pro EMA

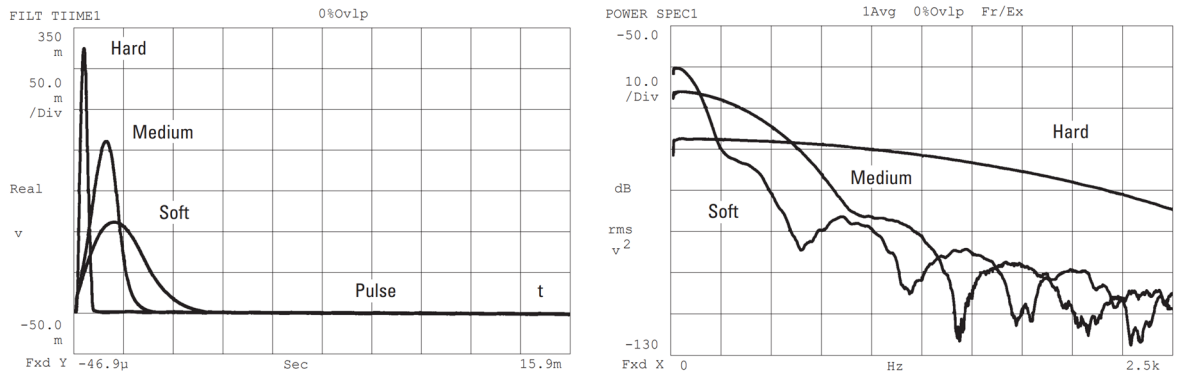
Zjištění modálních parametrů může být provedeno výpočtem, například vytvořením FEM modelu dané soustavy. Přesnost výsledků zde závisí na tom, jak přesně FEM model napodobuje reálnou soustavu. Druhou možností je provedení měření na reálné soustavě, například pomocí experimentální modální analýzy. V praktické části této práce byla použita metoda impaktního buzení vibrací pomocí kladívka, na ni se tedy zaměří i tato kapitola, uvedený postup je převzat z [25, 24, 23].

Měřící řetězec je zobrazen na Obr. 2.5. Na měřené součásti jsou rozmístěny senzory měřící výstupní veličinu. Na kladívku je umístěn snímač síly úderu kladívka. Měřící soustava zaznamenává měřené veličiny v čase. Obsluha provede úder do daného místa soustavy, čímž vybudí kmitání této soustavy. Z naměřeného silového vstupu a výstupů ve formě záznamu kmitání, se poté vypočítává frekvenční přenosová funkce (FRF). FRF v případě, kdy měřenou veličinou je poloha, bývá nazývána anglickým termínem compliance, případně receptance, pro FRF při měření rychlosti se užívá název mobility a pro FRF při výstupu ve tvaru zrychlení se užívá termín accelerance [24].



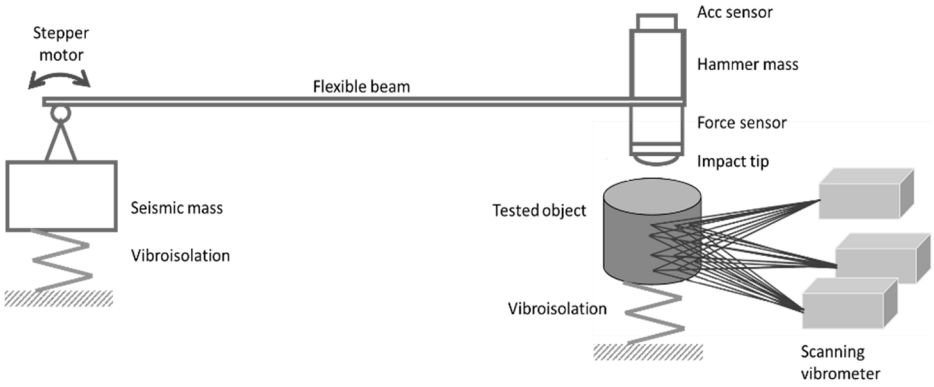
Obr. 2.5 Schéma měřícího řetězce při impaktně buzené EMA [25]

Silový pulz vzniká interakcí kladívka s měřenou soustavou, dochází zde k impaktu. Průběh silového pulzu tak závisí mimo jiné na tuhosti špičky kladívka. Toho je využíváno pro vybuzení požadovaného frekvenčního spektra vibrací soustavy. Pokud by silový pulz byl nekonečně krátký, vznikl by takzvaný Diracův impulz, při provedení Fourierovy transformace tohoto pulzu by bylo možné zjistit, že dochází k vybuzení všech frekvencí. Pro delší silové impulzy způsobené použitím špičky kladívka s nižší tuhostí, dochází k vybuzení nižšího spektra frekvencí, jak ilustruje Obr. 2.6. Za vybuzené se považují frekvence, kde spektrální výkonová hustota silového impulzu neklesne o více jak 10 až 20 dB [24].



Obr. 2.6 Vliv tuhosti kladívka na vybuzené frekvence soustavy [24]

V [26] byl představen řetězec měření znázorněný na Obr. 2.7. Měření odezvy obstarává laserový vibrometr na několika definovaných bodech na součásti. Rázové buzení provádí automatické kladívko, schopné provádět opakované údery do definovaného místa součásti a snížit tak odlišnost mezi jednotlivými údery, která se vyskytuje, pokud jsou údery prováděny ručně. Výhodou tohoto postupu je, že se budící ani měřící člen trvale nedotýkají měřené součásti a tím pádem nedochází k ovlivnění měření. Dále je takto možné provést automatizovaně velké množství opakovaných měření, jejichž zprůměrováním je možné získat přesnější výsledky.



Obr. 2.7 Schéma měřícího řetězce pro automatické měření [26]

2.9 Metody zpracování EMA

Frekvenční přenosová funkce (FRF) je definována jako poměr mezi Fourierovým obrazem výstupu systému a Fourierovým obrazem vstupu systému, jak je možné vidět také v *(2-37)*. Pro případ systému s jedním vstupem a jedním výstupem (SISO) platí níže uvedené vztahy pro přenos *(2-38)*, *(2-39)* a *(2-40)*. Zde písmeno c v pravém horním indexu znamená komplexně sdruženou veličinu a $S\_\_$ značí spektrální výkonovou hustotu.

$$G_{(\omega)} = \frac{U_{(\omega)}}{F_{(\omega)}} = \frac{U_{(\omega)}}{F_{(\omega)}}\frac{U^c_{(\omega)}}{U^c_{(\omega)}} = \frac{S_{uu(\omega)}}{S_{uf_{(\omega)}}} = \frac{U_{(\omega)}}{F_{(\omega)}}\frac{F^c_{(\omega)}}{F^c_{(\omega)}} = \frac{S_{fu_{(\omega)}}}{S_{ff_{(\omega)}}} \qquad \textit{(2-38)}\ \textit{[27]}$$

$$G^C{}_{(\omega)} = \frac{U^c_{(\omega)}}{F^c_{(\omega)}} = \frac{U^c_{(\omega)}}{F^c_{(\omega)}}\frac{U_{(\omega)}}{U_{(\omega)}} = \frac{S_{uu(\omega)}}{S_{fu_{(\omega)}}} = \frac{U^c_{(\omega)}}{F^c_{(\omega)}}\frac{F_{(\omega)}}{F_{(\omega)}} = \frac{S_{uf_{(\omega)}}}{S_{ff_{(\omega)}}} \qquad \textit{(2-39)}\ \textit{[27]}$$

$$S_{uu(\omega)} = G^C{}_{(\omega)}\, S_{fu_{(\omega)}} = G^C{}_{(\omega)}\, G_{(\omega)}\, S_{ff_{(\omega)}} = \left|G_{(\omega)}\right|^2 S_{ff_{(\omega)}} \qquad \textit{(2-40)}\ \textit{[27]}$$

Naměřením vstupu $f_{(t)}$ a výstupů $x_{(t)}$ s následným provedením Fourierovy transformace lze za použití výše uvedených vztahů získat FRF pro každou kombinaci vstupu a výstupu. Úkolem zpracování EMA je nalezení modálních parametrů takových, aby co nejlépe aproximovali naměřenou FRF. Pro tento účel byly vyvinuty různé algoritmy.

Jak uvádí [28], je možné algoritmy rozdělit na dvě kategorie. První skupinu tvoří algoritmy pracující ve frekvenční doméně, porovnávají tedy naměřenou FRF s FRF vytvořenou pomocí hledaných modálních parametrů a hledají takové modální parametry, aby byla shoda mezi těmito křivkami co nejlepší. Příkladem této skupiny metod jsou peak picking method a circle fit method, obě patří mezi SDOF metody. Pro tyto metody je dle [28] typické, že identifikují jednotlivé módy zvlášť, vliv ostatních módů buď zanedbávají, nebo jej berou v potaz v rámci algoritmu. Tento přístup může mít špatný vliv na přesnost výsledků.

Druhou skupinu tvoří algoritmy založené na identifikaci parametrů v časové oblasti. Mezi tyto algoritmy podle [28] patří complex exponential method (CE), least-squares complex exponential method (LSCE) a Ibrahim time domain method. LSCE je rozšířením CE metody schopné zpracovávat identifikaci přenosu s více výstupy. Tento přístup může zvýšit přesnost identifikace, jelikož, jak bylo uvedeno dříve, vlastní frekvence a modální tlumení jsou parametry vlastní pro

celou soustavu, tedy pro všechny výstupy. Nepřesnosti měření jednotlivých výstupů se tak mohou „vyrušit“.

V programovacím prostředí Matlab jsou algoritmy pro identifikaci modálních parametrů implementovány ve funkci modalfit. Jak uvádí dokumentace [29], vstupem této funkce je matice FRF ve tvaru receptance. Tato data poté funkce zpracovává pomocí některého z následujících algoritmů, které se volí: Peak picking method, LSCE, nebo Least-squares rational function estimation method. Poslední uvedený algoritmus nebyl výše popsán, jeho funkce byla představena v [30]. Vstupem funkce modalfit je dále parametr mnum, který představuje počet modálních příspěvků, ze kterých se FRF skládá. Jak uvádí [28], je často obtížné tento parametr stanovit, jelikož některé vlastní frekvence mohou mít blízké hodnoty a překrývat se. Jiné peaky na FRF zase nemusí představovat modální příspěvky a autor v uvedeném zdroji je přiřazuje vlivům šumu na naměřená data.

Data naměřená na reálné konstrukci nemohou začínat a končit přesně v nule. S daty, kde mimo naměřený úsek je nulová hodnota, nicméně počítají navazující operace, jako Fourierova transformace. Proto jsou na data aplikována takzvaná okénka, která hladkým způsobem přivedou okraje dat do nuly, skoky v datech by totiž vedly k chybám v jejich zpracování. V praxi je často používáno kosinové okénko tvořené posunutou kosinovou funkcí [27]. Při EMA identifikaci soustav s malým útlumem se používá exponenciální okénko, které dodá náhradní útlum [27]. Okénko se aplikuje jednoduše vynásobením s měřeným signálem, kdy má okénko stejnou délku jako záznam dat.

2.10 Převod spojitého stavového popisu na diskrétní

Stavový popis spojitého systému je popsán v *(2-18)* jeho řešením je *(2-41)*. Předpokládejme konstantní vstup pro daný krok diskrétního systému. To odpovídá použití tvarovače nultého řádu (zero order hold ZOH), který podrží poslední diskrétní vstup jako převod signálu z digitálního na analogový. Potom získáme uvedené vztahy *(2-42)* pro nové matice diskrétního stavového popisu *(2-43)*. Mezi další metody diskretizace patří first order hold (FOH), který předpokládá vstupy ve tvaru lineární interpolace mezi jednotlivými hodnotami diskrétního signálu [31].

$$\underline{s(t)} = e^{\hat{\underline{\underline{A}}}(t-t0)}\,\underline{s(t0)} + e^{-\hat{\underline{\underline{A}}}\,t}\int_{t0}^{t} e^{-\hat{\underline{\underline{A}}}\,\tau}\,\hat{\underline{\underline{B}}}\,\underline{f(\tau)}\,d\tau \qquad \textit{(2-41) [32]}$$

$$\widehat{\underline{\underline{M}}} = e^{\hat{\underline{\underline{A}}}\,\Delta t}, \qquad \widehat{\underline{\underline{N}}} = \int_0^{\Delta t} e^{-\hat{\underline{\underline{A}}}\,\tau}\,d\tau\;\hat{\underline{\underline{B}}}, \qquad \hat{\underline{\underline{O}}} = \hat{\underline{\underline{C}}}, \qquad \hat{\underline{\underline{P}}} = \hat{\underline{\underline{D}}} \qquad \textit{(2-42) [32]}$$

$$\begin{aligned} \underline{s(k+1)} &= \widehat{\underline{\underline{M}}}\,\underline{s(k)} + \widehat{\underline{\underline{N}}}\,\underline{f(k)} \\ \underline{y(k)} &= \hat{\underline{\underline{O}}}\,\underline{s(k)} + \hat{\underline{\underline{P}}}\,\underline{f(k)} \end{aligned} \qquad \textit{(2-43)}$$

2.11 Output error model

Output error (OE) je lineární model patřící mezi Infinite impulse response filtry (IIR) [33]. Předpis pro OE ve formě prediktoru je uveden v *(2-44)*, kde $\hat{y}$ značí výstup modelu, u značí vstup a m značí

řád modelu. Prediktor vytvořený z OE modelu nemá vstup ve formě výstupu reálného systému y a tak se shoduje s modelem ve formě simulátoru. Model v této formě není lineární, jelikož $\hat{y}(k-1) \dots \hat{y}(k-m)$ závisí na parametrech a a b, nicméně při nahrazení $\hat{y}$ za skutečný výstup reálného systému y vznikne forma odpovídající modelu Autoregressive exogenous model (ARX), která je lineární v parametrech, pro zjištění parametrů je potom možné použít metodu nejmenších čtverců [33].

$$\hat{y}(k) = b_1 u(k-1) \dots + b_m u(k-m) - a_1 \hat{y}(k-1) \dots - a_m \hat{y}(k-m) \qquad \textit{(2-44)}\ \textit{[33]}$$

2.12 Levenberg-Marquardtova optimalizace

Ztrátovou funkci L definujeme jako nelineární úlohu nejmenších čtverců *(2-45)*, kde $\underline{r}$ je vektor odchylek, například hledané funkce od měřených dat, a $\underline{p}$ je vektor parametrů. Úloha zní, nalézt takové $\underline{p}$, aby L byla minimální. Obecný předpis pro nelineární lokální optimalizační algoritmy založené na gradientu je uveden v *(2-46)*, kde k značí jednotlivé kroky optimalizace, η představuje velikost kroku a R je matice transformující gradient $\underline{g}$, určená konkrétním algoritmem. Nejjednodušší volbou je $R = E$, což vede na metodu sestupu ve směru nejprudšího klesání (Steepest gradient descent) , pokud je $R = H^{-1}$, kde H je Hessián, jedná se o Newtonovu metodu, pro lineární úlohu by při velikosti kroku 1 metoda našla minimum během jednoho kroku [33]. Pro nelineární úlohu nejmenších čtverců z Newtonovy metody vychází Gauss-Newtonova metoda, pro kterou platí vztahy *(2-47)*. Hessián je zde aproximován součinem Jacobiho matic, to pro daný problém představuje vhodnou aproximaci pro případy, kdy je počáteční stav rozumně blízko minima [33]. Rozšířením Gauss-Newtonovy metody je Levenberg-Marquardtova metoda, pro kterou platí *(2-48)*, jde vlastně o kombinaci Gauss-Newtonovy metody a Steepest gradient descent. Koeficient λ určuje poměr mezi těmito metodami, velká hodnota zvyšuje vliv Steepest gradient descent, který bude lépe fungovat ve velkých vzdálenostech od minima, blíže k minimu obvykle lépe zafunguje Gauss-Newton, přidáním E se navíc často řeší problém se špatnou podmíněností R [33].

$$L(\underline{p}) = \underline{\underline{r(\underline{p})^T}}\ \underline{\underline{r(\underline{p})}} \qquad \textit{(2-45)}\ \textit{[33]}$$

$$\underline{p_k} = \underline{p_{k-1}} - \eta_{k-1}\ \underline{\underline{R_{k-1}}}\ \underline{g_{k-1}}\ , \qquad kde: \quad \underline{g_{k-1}} = \frac{\partial L(\underline{p})}{\partial \underline{p}} \qquad \textit{(2-46)}\ \textit{[33]}$$

$$\underline{\underline{R_k}} = \underline{\underline{H_k}^{-1}} = \left(\underline{\underline{J_k}^T}\ \underline{\underline{J_k}}\right)^{-1}, \qquad \underline{g_{k-1}} = \underline{\underline{J_k}^T}\ \underline{r_k}\ , \qquad kde:\ J_{k\,ij} = \frac{\partial r(\underline{p})_i}{\partial p_j} \qquad \textit{(2-47)}\ \textit{[33]}$$

$$\underline{\underline{R_k}} = \left(\underline{\underline{J_k}^T}\ \underline{\underline{J_k}} + \lambda_k\ \underline{\underline{E}}\right)^{-1}, \qquad \underline{g_k} = \underline{\underline{J_k}^T}\ \underline{r_k} \qquad \textit{(2-48)}\ \textit{[33]}$$

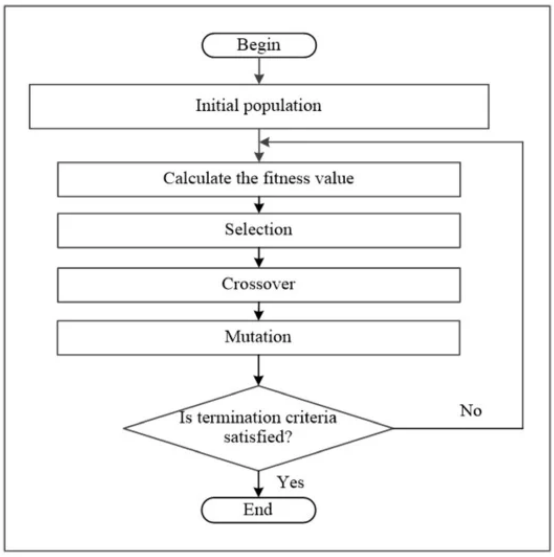
2.13 Hankelova singulární čísla

Hankelova singulární čísla jsou definována jako odmocnina vlastních čísel součinu gramiánu řiditelnosti a pozorovatelnosti [34]. Jsou invariantní vůči transformaci stavového popisu a mohou být vnímány jako ukazatele významnosti jednotlivých stavů balancované realizace [34]. Toho lze s výhodou využít při redukci modelu, stavy s nízkým singulárním Hankelovým číslem mají menší vliv na celkovou dynamiku systému, může tak být odhadnuto, kolikátého řádu systém je, případně mohou být zanedbány stavy s nízkým Hankelovým singulárním číslem.

2.14 Genetický algoritmus

Genetický algoritmus je globální optimalizační metoda patřící do skupiny evolučních algoritmů, je inspirovaná zkoumanými mechanismy evoluce v přírodě [33]. Z oboru věd zkoumajících evoluci tak metoda přebírá i názvosloví. Jedinec představuje bod v prostoru optimalizovaných parametrů, populace je množina jedinců, fitness funkce je cílová funkce, kterou se snažíme maximalizovat. Jednotlivé parametry jsou označovány jako geny. Problém optimalizace parametrů nejprve transformujeme do matematického problému hledání globálního maxima fitness funkce, jejímiž vstupy jsou optimalizované geny. Základní princip je zachycen na Obr. 2.8, nejprve je vytvořena počáteční populace, pro jedince v populaci je vyhodnocena fitness funkce, následuje selekce jedinců z populace na základě fitness funkce, vybraní jedinci pokračují do „křížení“, kde jsou jejich parametry kombinovány a je vytvořena nová populace – offspring. Na novou populaci je následně aplikována mutace parametrů a je pro ni vyhodnocena fitness funkce, takto algoritmus pokračuje až do ukončení, každé opakování se nazývá generace.

Genetický algoritmus je pro Python implementován například v knihovně PyGad [35]. Knihovna obsahuje třídu genetického algoritmu (Genetic algorythm – GA), při vytvoření objektu GA je zadána fitness funkce jako funkce vstupních parametrů. Dále se definují vlastnosti optimalizace jako počet generací, kolik jedinců je v populaci, kolik jedinců je z populace vybráno při selekci, jaká je pravděpodobnost mutace. Také se definují algoritmy použité pro jednotlivé metody uvedené v Obr. 2.8. Důležitou vlastností je parametr gene_space, který pro každý gen určí rozmezí, ve kterém se může pohybovat, pokud je rozmezí označeno jako None, je gen (parametr) neomezený a v průběhu optimalizace se může dostat úplně mimo hodnoty obsažené v původní populaci.



Obr. 2.8 Flowchart principu genetického algoritmu [36]

2.15 Hluboké neuronové sítě (DNN)

2.15.1 DNN v kontextu umělé inteligence

Cílem umělé inteligence je automatizace lidmi prováděné intelektuální činnosti. Jednou z významných oblastí umělé inteligence jsou expertní systémy, které pomocí sady pravidel napodobují rozhodování člověka, například konstruktéra při volbě rozměrů součástí. K tomu je ale nejprve třeba tuto sadu pravidel vytvořit, což je pro člověka obtížný úkol například při rozpoznávání obrazu či řeči. Člověk sice dokáže poznat objekt na obrázku, je už ale obtížné popsat logický proces, který to umožnil. I proto vzniklo další odvětví umělé inteligence, strojové učení.

Strojové učení obrací proces skrytý za expertními systémy. Místo toho, aby člověk zadal počítači sadu pravidel a následně pro vstupní data od počítače očekával výsledky, strojové učení dostane od člověka sadu vstupních dat a správných odpovědí, pro ně má strojové učení nalézt pravidla (takový postup je konkrétně nazýván řízené učení). Strojové učení se tedy pokouší nalézt vhodnou transformaci vstupních dat na požadované výsledky. Strojové učení se dělí na řízené učení (supervised machine learning), neřízené učení (unsupervised machine learning), samořízené učení (self-supervised machine learning) a posilované učení (reinforcement machine learning). U řízeného učení existují trénovací vstupní data (training samples), ke kterým jsou přiřazeny správné výsledky (labels), které vstupní data v případě klasifikace dělí do tříd (classes). Mezi obvyklé úkoly pro strojové učení mimo jiné patří:

Klasifikace, například určení čísla na obrázku. Výstupem je diskrétní hodnota – třída.

Regrese, například určení rozměrů objektu. Výsledkem spojitá hodnota.

Tvorba sekvencí, například popisků obrázku.

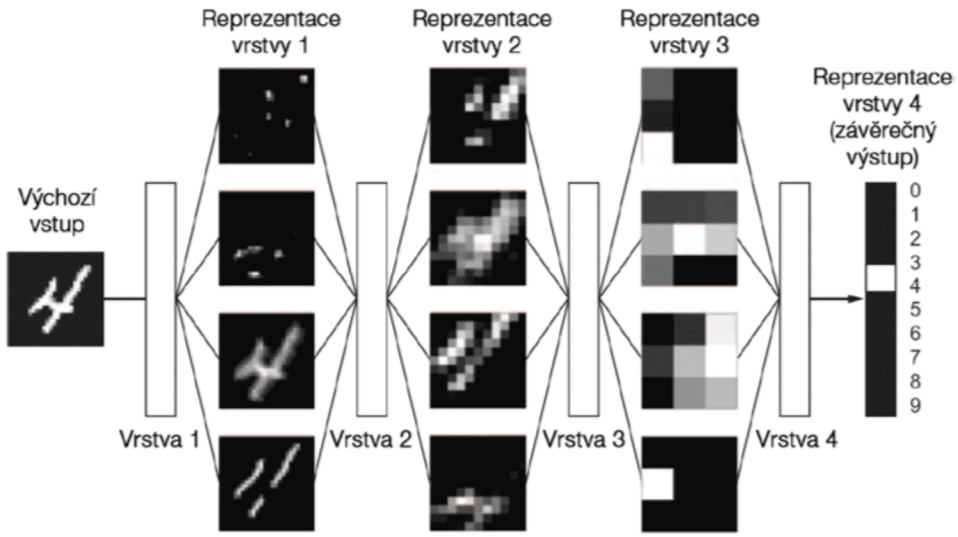
Detekce objektů, například nalezení semaforu na obrázku.

2.15.2 Hluboké neuronové sítě

Mezi metody pro dosažení strojového učení patří umělé neuronové sítě (artificial neural network – ANN). ANN obsahující více jak tři vnitřní skryté vrstvy, bývají označovány jako hluboké (Deep

neural network – DNN) [1]. Neuronová síť je definována svou topologií a svými parametry (vahami). Učení sítě je proces hledání ideálních parametrů. Síť je v podstatě sérií tenzorových operací, návaznost těchto operací je nazývána jako tvar sítě (topologie). Topologie je popsána takzvanými hyperparametry, například množstvím vrstev (layer) určitých operací. Určením topologie vytváříme prostor hypotéz, kde kombinace vah sítě představuje bod v tomto prostoru. Učením provádíme optimalizaci na prostoru hypotéz. Změnou topologie na vhodnější prostor hypotéz je možné dosáhnout lepších výsledků učení. Pro tento účel je možné provádět také optimalizaci hyperparametrů, například pomocí genetického algoritmu.

Na Obr. 2.9 je uveden příklad neuronové sítě, která vstupní data ve formě obrázku s číslem postupně transformuje v jednotlivých vrstvách na hluboké reprezentace těchto dat. Výsledkem je potom výstupní reprezentace sloužící ke klasifikaci čísla.

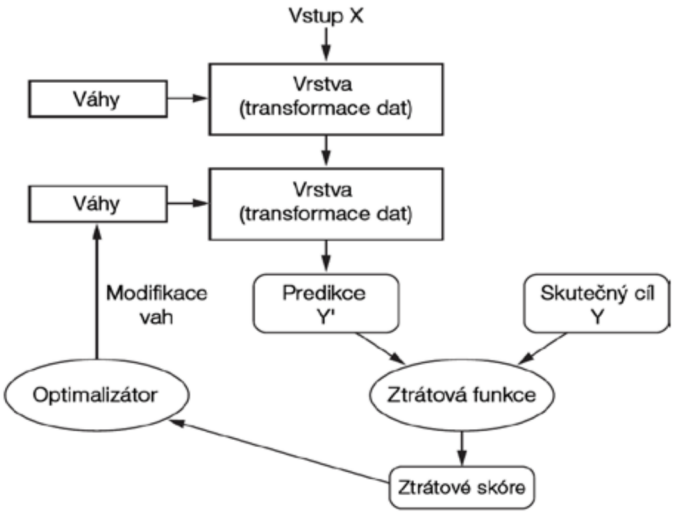


Obr. 2.9 Hluboké reprezentace naučené modelem pro klasifikaci číslic [37]

2.15.3 Učení DNN

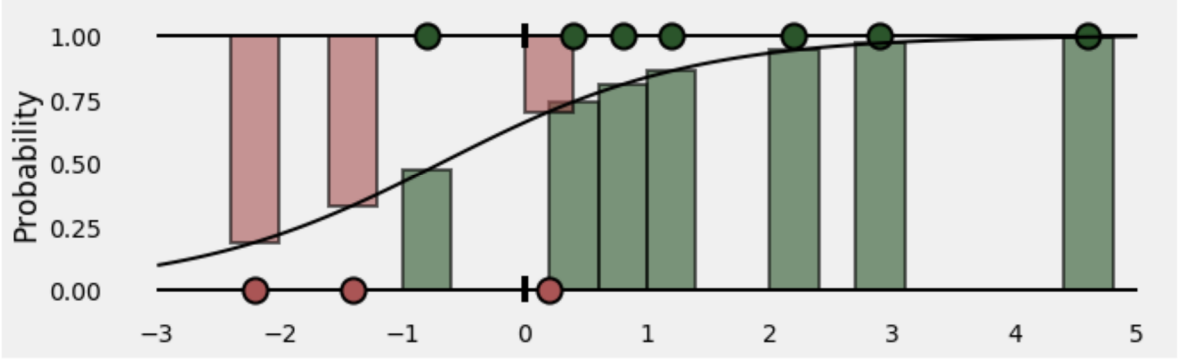
Aby bylo možné určit optimální parametry sítě, je třeba stanovit optimalizovanou funkci. Tou je v případě řízeného učení ztrátová funkce (loss function). Jak ukazuje Obr. 2.10, vstupem ztrátové funkce jsou výstupy (predikce) sítě pro vstupní vzorky a skutečné cíle (labels). Ztrátová funkce má určit míru úspěšnosti predikce. Loss function se snažíme minimalizovat pomocí stanovování lepších parametrů sítě. Tento proces se v oblasti strojového učení nazývá backpropagation algorythm. Pokud lze stanovit gradient – derivaci loss function podle parametrů, lze provádět optimalizaci například pomocí metody gradient descent algoritmu. Jelikož se síť skládá z mnoha tenzorových operací řazených za sebou, je pro derivaci dle parametrů těchto operací použita základní matematická operace – derivace složené funkce. Programové knihovny jako Tensorflow

mají pro jednotlivé vrstvy používané pro tvorbu DNN předdefinované symbolické derivace, pro sestavený model s vahami tak dokáží efektivně vrátit hodnotu gradientu [37].



Obr. 2.10 Ztrátová funkce (loss function), optimalizátor upravující váhy sítě [37]

Podle [37] je nejvhodnější loss function pro úlohy binární klasifikace takzvaná Binary crossentropy. Pokud je úkolem provést binární klasifikaci, výstupem sítě by mělo být rozdělení bodů do tříd 0 a 1. Funkce sigmoid (černá křivka na Obr. 2.11) reprezentuje pravděpodobnost správného zařazení do tříd pro jednotlivé predikce, hodnota na ose x odpovídá predikci, barva odpovídá správné třídě. Výsledkem Loss function binary crossentropy je průměr z hodnot záporných logaritmů sloupců na obrázku [38]. Matematické vyjádření je v rovnici *(2-49)*, kde o_i značí třídu a $prob(o_i)$ pravděpodobnost správného zařazení, *N* značí množství dat.



Obr. 2.11 Pravděpodobnosti správné klasifikace bodů do tříd 0 (červená) a 1 (zelená), osa x - predikce [38]

$$Loss = \frac{1}{N}\sum_{i=1}^{N} o_i \cdot \log\left(prob(o_i)\right) + (1 - o_i) \cdot \log\left(1 - prob(o_i)\right) \qquad \textit{(2-49)}\ \textit{[38]}$$

První osa vstupních dat bývá vyčleněna pro osu jednotlivých příkladů, ty nejsou do sítě dodávány najednou, ale po dávkách (batches) [37]. Optimalizátor (Obr. 2.10) potom počítá gradient pro každou dávku a aktualizuje parametry. Poté co jsou zpracována všechna trénovací data, končí epocha, v další epoše se proces opakuje, množstvím epoch lze ovlivnit délku trénování.

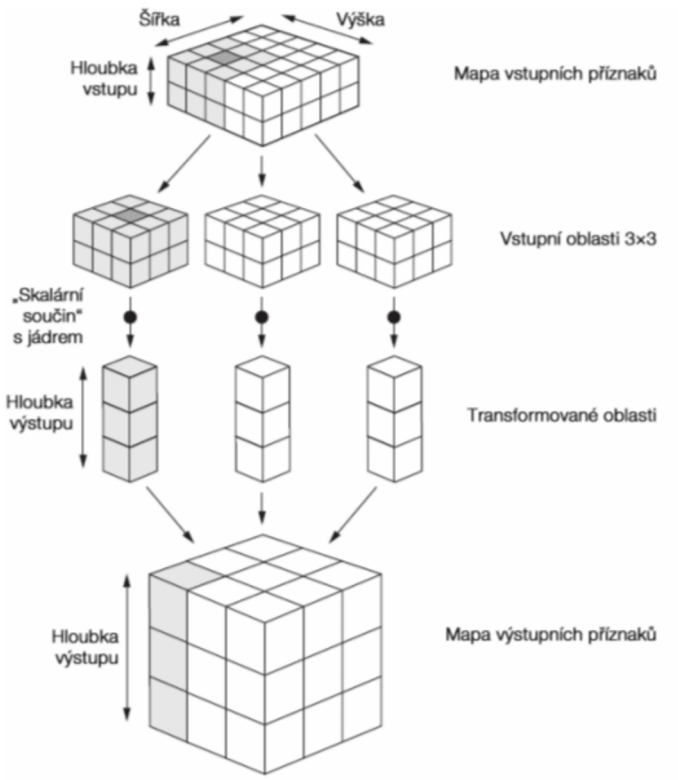
2.15.4 Některé vrstvy DNN

Základním stavebním blokem neuronových sítí je vrstva (layer). Hustě propojené vrstvy se nazývají dense, jejich operaci je možné zapsat jako *output=relu(dot(W, input)+b)* [37], kde relu je aktivační funkce, dot je operace násobení, W je matice jádra (kernel) a vektor b je bias. Členy W a b jsou parametry sítě, matice W má rozměry (rozměr vstupu x rozměr výstupu), tyto rozměry nemusí být totožné a může tak docházet ke změně rozměru dat průchodem vrstvou.

2D konvoluční vrstva je odvozená od matematické operace diskrétní konvoluce a bývá často využívána ke zpracování obrazů [37]. Jádro (kernel) je na rozdíl od dense vrstvy menší jak vstupní data. Při konvoluci je po daných krocích vybírán výsek ze vstupních dat, který je násoben jádrem, poslední dimenze jádra je nazývána hloubka výstupu či filtr a určuje rozměr výstupních dat. Tato operace je znázorněna na Obr. 2.12, kde operaci násobení s jádrem by bylo zřejmě možné zapsat $out_k = input_{ij} \cdot Kernel_{ijk}$, kde k představuje hloubku filtru. Tato operace se opakuje pro všechny segmenty vstupních dat, až vznikne nový výstup. Síť sestavená z konvolučních vrstev se nazývá konvoluční neuronová síť (convolutional neural network – CNN).

1D konvoluce je často úspěšně používána na sekvence dat [37]. Na rozdíl od 2D maticových konvolucí, 1D CNN užívají jednodušší vektorové operace, což znamená, že jejich výpočetní náročnost je podstatně nižší [1]. Například u časového záznamu se okénko 1D konvoluce posouvá pouze v jednom směru po 2D tenzoru o rozměru („čas“ x kanály). Vrstva separable convolution provádí konvoluci pro jednotlivé kanály vstupu zvlášť a následně provede konvoluci s jádrem o velikosti (1 x kanály), kterým jednotlivé výstupy sloučí, lze tak dosáhnout rychlé konvoluce s malým množstvím parametrů [37].

Operace maxpooling, podobně jako konvoluce, vybírá segmenty vstupních dat. Z vybraného segmentu vybere maximum, které slouží jako nový výstup. Výběr segmentů se posouvá s krokem větším jak 1 a tak dochází ke zmenšení výstupních dat. Tato operace přesouvá pozornost z detailních příznaků dat na větší celky, síť se díky tomu může naučit příznaky na různých úrovních, jako například hrany, obočí, obličej, skupina osob.



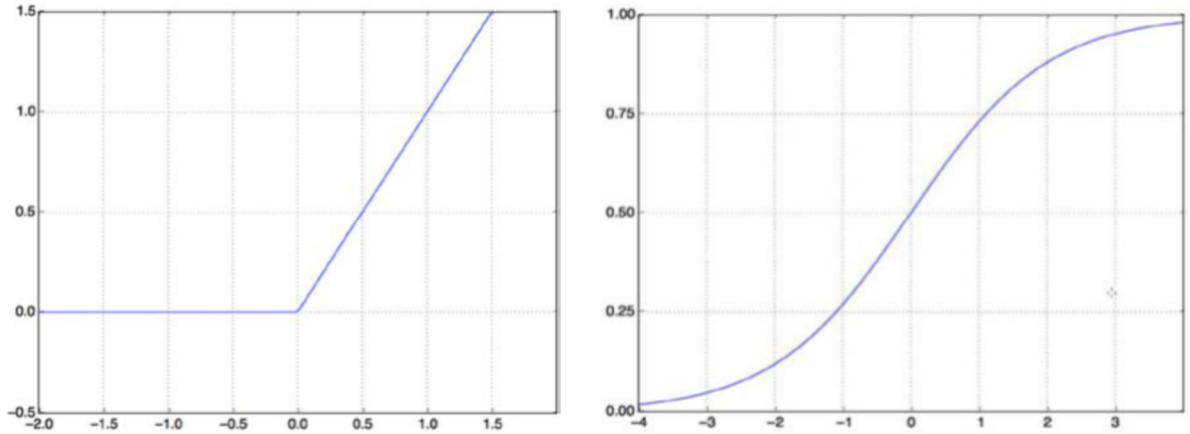
Obr. 2.12 Princip 2D konvoluční vrstvy [37]

Rekurentní neuronové sítě (recurent neural networks – RNN) mají na rozdíl od předchozích vlastní „paměť", předchozí vstup ovlivňuje zpracování vstupu následujícího, proto bývají využívány často pro zpracování časových záznamů či textu [37].

2.15.5 Aktivační funkce

Vrstvě DNN může být přiřazena aktivační funkce. Jednou z takových funkcí je relu (Obr. 2.13 vlevo), která již byla zmíněna u příkladu dense vrstvy. Lze ji snadno zapsat jako *relu(x)=max(x,0)* a slouží k zavedení nelinearity do modelu. Pokud by síť tvořila pouze velká série dense vrstev bez aktivačních funkcí, vytvořený prostor hypotéz pro optimalizaci by byl velmi omezený – síť by byla schopna provádět pouze lineární operace, zavedením nelinearity se síť může naučit složitější operace.

Funkce sigmoid (Obr. 2.13 vpravo) převádí hodnoty z rozmezí *<-inf, inf>* do rozmezí *<0, 1>*. Často je používána jako aktivační funkce poslední vrstvy pro binární klasifikaci [37].



Obr. 2.13 Funkce relu (vlevo) a funkce sigmoid (vpravo) [37]

2.15.6 Přeučení (overfitting)

Přeučení se projevuje tak, že na trénovacích datech síť dosahuje lepších výsledků, jak na testovacích. Cílem strojového učení je získat pravidla platná pro určitou skupin dat. Pokud model dosahuje dobrých výsledků na trénovacích datech, znamená to pouze, že tato pravidla nalezl pro trénovací data, nemusí jít o obecněji platná pravidla. Proto je účinnost sítě vyhodnocována na testovacích datech, která jsou odlišná od trénovacích. Pokud dochází k přeučení, autor [37] doporučuje následující kroky:

Redukce sítě – snížením množství hledaných parametrů je síť nucena hledat jednodušší transformace, které mohou mít obecnější platnost. K tomu dochází na úkor optimality vyhodnocení trénovacích dat.

Přidání váhové regularizace – síť je zjednodušena omezením velikosti vah.

Přidání výpadku (dropout) – některé z příznaků vstupních trénovacích dat jsou náhodně vynulovány.

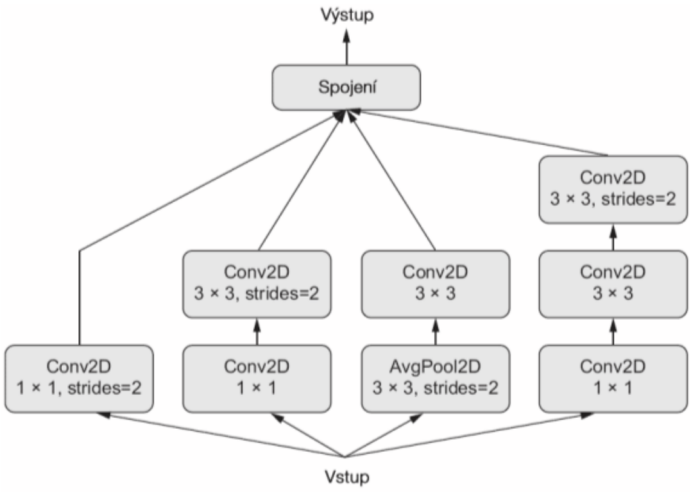
Zvýšení množství a rozmanitosti trénovacích dat.

2.15.7 Získávání příznaků (feature engineering)

Přestože je DNN schopná zpracovávat hrubá data, může být vhodné je předpřipravit pro zvýšení výkonnosti sítě. Dále je vhodné vstupní data normalizovat například tak, aby každý vzorek měl nulovou průměrnou hodnotu a směrodatnou odchylku rovnu jedné, může se tak zabránit nadměrnému nárůstu hodnot parametrů sítě [37].

2.15.8 Tvorba složitějších sítí

Při tvorbě sítě jsou propojovány jednotlivé vrstvy, kdy výstup jedné vrstvy může sloužit za vstup vrstvě jiné. Tímto způsobem je možné sítě větvit, aplikovat různé vrstvy na stejné či různé vstupy, jako například na Obr. 2.14. Dále je možné využít již natrénované sítě, začlenit je do sítí nových a provádět trénování jen části parametrů, toho se s výhodou využívá při zpracování obrazu, kde jsou k dispozici rozsáhlé předtrénované sítě, které dokáží rozpoznat vzory jako oči, hrany, semafory a podobně [37].



Obr. 2.14 Modul inception složený z více paralelních větví [37]

2.16 Confusion matrix a ROC křivka

Confusion matrix slouží k vyhodnocování výsledků klasifikace. Pro binární klasifikaci (Obr. 2.15) jsou data rozdělena na skutečně pozitivní (actually positive) a skutečně negativní (actually negative), vyhodnocovaná metoda rozdělí data na predicted positive a predicted negative. Porovnáním těchto skupin lze získat správně určené pozitivní (TP), správně určené negativní (TN), špatně určené negativní (FN), kdy metoda prohlásí, že výsledek je negativní, ale on ve skutečnosti není, špatně určené pozitivní (FP), kdy metoda prohlásí, výsledek je pozitivní, ale on ve skutečnosti není. Z těchto hodnot lze vyhodnotit další charakteristiky, které jsou uvedeny v tabulce.

Výstupem neuronové sítě pro binární klasifikaci může být spojitá hodnota mezi 0 a 1 (například díky použití aktivační funkce sigmoid v poslední vrstvě). K převedení této hodnoty na binární může sloužit treshold, pokud je výstupní hodnota nižší jak treshold, je diskrétním výsledkem nula, pokud je vyšší, je výsledkem 1. Vykreslením hodnot sensitivity (true positive rate) a false positive rate pro jednotlivé hodnoty treshold od 0 do 1 vznikne ROC křivka (Receiver operating characteristics) [39]. Diagonála na křivce od [0,0] do [1,1] odpovídá náhodnému klasifikátoru, ideální klasifikátor má průběh od [0,0] přes [0,1] do [1,1] [39].

Total Population	Predicted positive	Predicted negative	
$TP + FN + FP + TN$	$TP + FP$	$FN + TN$	
Actually positive	True positive (TP)	False negative (FN)	Sensitivity
$TP + FN$	TP	FN	$\frac{TP}{TP + FN}$
Actually negative	False positive (FP)	True negative (TN)	False positive rate
$FP + TN$	FP	TN	$\frac{FP}{FP + TN}$
Accuracy	Precision	False ommision rate	Positive likehood ratio
$\frac{TP + TN}{TP + FN + FP + TN}$	$\frac{TP}{TP + FP}$	$\frac{FN}{FN + TN}$	$\frac{TP\ (FP + TN)}{FP\ (TP + FN)}$

Obr. 2.15 Confusion matrix s dalšími charakteristikami a jejich vzorci [39]

Vlastní řešení

3 Cíle práce

Cílem této práce je vytvoření metody, která by vedla ke zjištění, zda je v konstrukci přítomna porucha. Jak naznačuje rešerše, existuje mnoho druhů poruch, které mohou mít různé projevy. Tato práce se zaměří na poruchy, které způsobí „odléhání a nárazy“. Může jít jak o uvolněné šrouby, tak například o praskliny, které se při kmitání střídavě rozevírají a dotýkají. Není tak cílem vytvořit univerzální metodu pro všechny poruchy, toho by bylo možné případně dosáhnout použitím různých metod, specifických pro různé poruchy, najednou.

Rešerše dosavadních řešení této problematiky naznačuje, že se velké množství prací soustředí na poruchu simulovanou či modelovanou pomocí ztráty tuhosti. Bez ohledu na použitou metodu potom základní princip spočívá v natrénování metody pro konkrétní konstrukci a jejím používání pouze na tuto konstrukci. Metoda potom dokáže identifikovat, že v konstrukci došlo ke změně, případně odhadnout její umístění a závažnost. Pro některé z uváděných metod přitom je nutné obstarat jak vzorky dat pro neporušenou reálnou konstrukci, tak vzorky dat pro porušený stav té samé konstrukce. S trochou nadsázky tak lze říci, že v praxi by například bylo nutné nejprve mostní konstrukci postavit, poté naměřit referenční data v neporušeném stavu. Následně konstrukci záměrně poškodit, naměřit data v poškozeném stavu. Nakonec by bylo třeba konstrukci znovu opravit a poté by již metoda mohla správně fungovat. Přitom natrénovanou metodu by nebylo možné použít u další konstrukce, která by se lišila rozměry či materiálovými vlastnostmi. Jedná se samozřejmě o extrémní případ, cílem uvedených metod bývá často identifikace polohy poruchy, což není snadný problém.

Autor této diplomové práce se domnívá, že klíčová problematika se skrývá v trénovacích datech, které jsou metodám poskytovány, konkrétně v metodách simulace poruchy. Jak již bylo řečeno, mnoho prací simuluje poruchu poklesem tuhosti. To je legitimní přístup, praskliny, uvolněné šrouby, koroze i další jistě pokles tuhosti způsobí. Důležité ovšem je, že samotný pokles tuhosti je „lineární“, tedy konkrétně: pokud u konstrukce s čistě lineárními vlastnostmi (například simulované ve FEM) zavedeme pokles tuhosti na nějakém místě, získáme opět lineární konstrukci. Na naměřené frekvenční charakteristice sice budou zřejmé změny, tyto změny však mohou být způsobeny nejen například změnou teploty, ale také odlišnostmi v rozměrech konstrukce. To je důležitý bod. Pro účely této práce definujme, že je možné požadovat po metodě pro identifikaci poruchy tři různě obtížné úkoly, které popíšeme pomocí příkladů výroby vlakového vagonu:

1. Konkrétní vagon je na konci výroby prověřen, že v něm nejsou přítomny žádné poruchy. Následně je na něm provedeno trénování metody detekce poruchy – získání referenčních dat v nepoškozeném stavu. V průběhu životnosti tohoto vagonu je potom možné provádět detekci poruchy pomocí porovnání měření v danou chvíli s referenčními hodnotami konkrétního vagonu z minulosti. Pokud je zaznamenána změna, jde pravděpodobně o poruchu.

2. Vagon patří do určité výrobní řady, na počátku je vytvořen jeden vagon (vagon A), který je prověřený, že v něm nejsou žádné poruchy. Na tomto vagonu je provedeno trénování metody detekce poruchy – získání referenčních dat v nepoškozeném stavu. Tato metoda s pomocí těchto dat potom může provádět detekci poruchy na všech vagonech výrobní řady. Lze jí tak použít i pro výstupní kontrolu nových vagonů na konci výrobního procesu.

3. Je vyvinuta metoda, která dokáže detekovat poruchy u různých vagonů různých výrobních řad, aniž by měla k dispozici referenční data. Metoda není trénována pro konkrétní konstrukci, ale pro konkrétní poruchu.

Metody zaměřující se na pokles tuhosti jsou stavěné k použití pro úkol 1. Na úkol 2 by mohli být využity, pokud by byla zajištěna dostatečná výrobní přesnost tak, aby nedocházelo k odlišnostem v modálních vlastnostech vlivem odlišných rozměrů dané výrobní řady. Takový požadavek by ovšem mohl vést ke zvýšení výrobních nákladů. Nicméně použití metod pro úkol 1 dává smysl, pokud místo vagonu uvažujeme například rozsáhlou mostní konstrukci. Metody se často zaměřují na identifikaci polohy poruchy, mohly by tak například napovědět, který nosník je v nepořádku. Pokud tak není nutné metodě poskytnout referenční data i z měření poškozené konstrukce, může být velmi přínosná.

Metoda vyvíjená v rámci této diplomové práce se však nebude pokoušet určit umístění poruchy. Místo toho se bude zaměřovat na aplikaci v uvedených úkolech 2 a 3. Jak bylo řečeno, klíčová je podoba trénovacích dat, odléhání a dosedání částí konstrukce totiž způsobí její nelineární chování. Změna rozměrů, tvarů a nejspíš ani teploty přitom vznik nelineárního chování obecně nevyvolá. Mějme tedy množinu ideálních (např. počítačově modelovaných) konstrukcí, v ní jsou rozměrově a tvarově naprosto odlišné konstrukce, z nichž některé mají simulovanou poruchu a jiné ne. Potom zřejmě je možné tuto množinu rozdělit na konstrukce s lineárním chováním a bez něj. Otázkou je, jak toto rozdělení provést v praxi, to bude cílem této práce.

Hlavní cíl práce lze rozdělit na tři dílčí cíle:

1. Prvním cílem bude vytvořit trénovací a verifikační data pro metody detekce poruchy konstrukce. Toho bude dosaženo jednak vytvořením FEM modelu konstrukcí s různými tvary a rozměry. Do těchto modelů bude na úrovni simulace zaveden model poruchy, který bude v rámci tohoto cíle vytvořen. Dále bude provedena experimentální modální analýza na reálné součásti s poruchou typu „odléhání“ a bez ní. Porovnání se simulačními výsledky umožní ověřit, zda byl zvolen správný simulační model poruchy. Nakonec bude vytvořen program pro generování velkého množství trénovacích dat zavádějící náhodné změny rozměrů, materiálových vlastností a tvarů konstrukcí s použitím simulačního modelu poruchy. Tato data budou použita v dalším kroku.

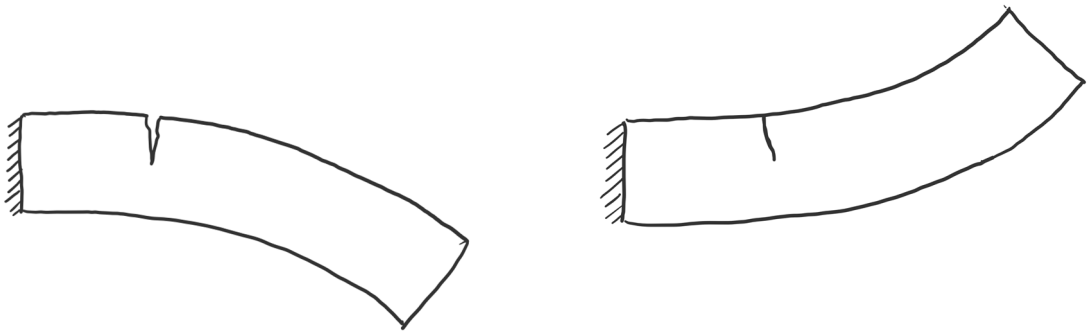
2. Druhým cílem bude vytvoření algoritmu pro binární klasifikaci vstupních dat. Vstupními daty bude časový záznam kmitání jednoho bodu soustavy po vybuzení kmitů silovým impulzem pomocí kladívka, případně i časový záznam silového pulzu. Algoritmus bude mít za úkol data roztřídit na dvě kategorie – s poruchou a bez poruchy. Pro tento účel budou vyvinuty dvě metody, jedna založená na optimalizaci pro identifikaci, kde budou parametry matematického modelu, nelineárního filtru, optimalizovány tak, aby výstup filtru co nejlépe napodoboval výstupní kmitání naměřené na soustavě. Na základě hodnot parametrů tohoto filtru, které představují vliv nelinearity, bude docházet ke klasifikaci dat. Druhá metoda bude založena na hlubokých neuronových sítích, které budou pomocí velkého množství nasimulovaných vstupních dat trénovány, aby prováděli klasifikaci.

3. Třetím cílem bude otestování navržených metod na testovacích datech, která budou zcela odlišná od trénovacích, pro splnění úkolů 2 a 3 a jejich vyhodnocení.

4 Tvorba dat

4.1 Model poruchy

Představme si nyní jednoduchý, jednostranně vetknutý nosník ve 2D prostoru, je tvořen homogenním materiálem, ale má v sobě rozsáhlou prasklinu (Obr. 4.1). Nosník kmitá tak, že jeho volný konec se střídavě nachází v horní a dolní úvrati, prasklina se v dolní úvrati rozevírá a v horní úvrati dochází k dosednutí plošek praskliny na sebe tak, že mohou přenášet tlakové napětí. Rozevřená prasklina tahové napětí přenášet nedokáže, z toho lze odvodit, že v obou případech bude mít nosník odlišnou tuhost.

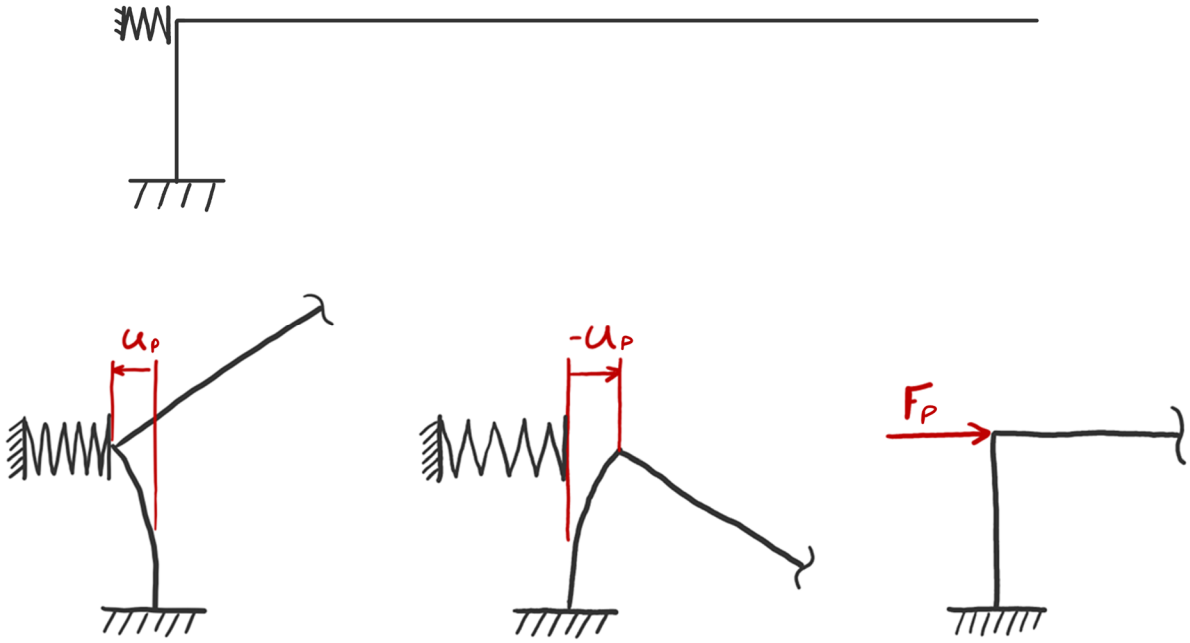


Obr. 4.1 Motivační příklad modelu poruchy – nosník s prasklinou

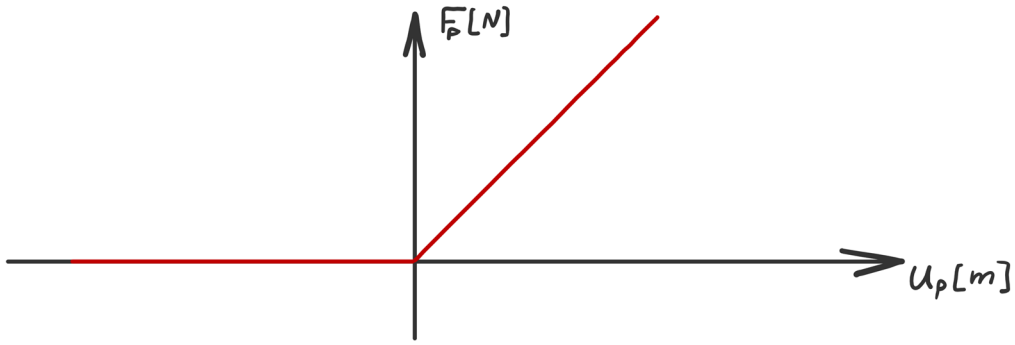
Ke změně tuhosti bude docházet ve chvíli, kdy se protilehlé plošky praskliny začnou dotýkat, nebo naopak když dotyk ustane. Tato změna nebude okamžitá, ovšem bude probíhat poměrně rychle. Toto chování se snaží napodobit mechanický model jednostranně vetknutého nosníku ve tvaru L na Obr. 4.2 nahoře. Nosníku se dotýká pružina, v klidové poloze pružina není stlačena. Vychýlení nosníku v místě dotyku pružiny je označeno u_p, síla, kterou pružina na nosník působí je označena F_p. Pružina není k nosníku připevněna, proto působí na nosník jen při stlačení pružiny, jak ukazují Obr. 4.2 dole, tedy jen kladnou silou F_p. Průběh síly v závislosti na výchylce ukazuje Obr. 4.3, tento průběh odpovídá funkci, která je v oblasti neuronových sítí označována jako aktivační relu funkce, či nelinearita (viz kapitola 2.15.5).

Pokud představený mechanický model bude modelován jako poddajná soustava, přítomnost této „dotykové“ pružiny povede k okamžité změně matice tuhosti při přechodu u_p přes nulu. Systém se bude chovat jako lineární při $u_p < 0$ a bude lineární i při $u_p > 0$, ovšem celkové chování systému bude nelineární, přesněji po částech lineární.

Tento model poruchy je velmi zjednodušující, ovšem splňuje požadavky cíle práce. Zavádí totiž do systému nelinearitu, která by mohla být charakteristickou vlastností poruch, při kterých dochází k odléhání a nárazům. Poruch jako prasklina, uvolněný šroub, kde část spoje přestane být pod tlakem od šroubu a začne odléhat, případně odlepená část lepeného spoje, či delaminace kompozitního materiálu, pokud bude vhodně orientovaná.



Obr. 4.2 Mechanický model struktury s nelineární poruchou – pružina pracující v jednom směru



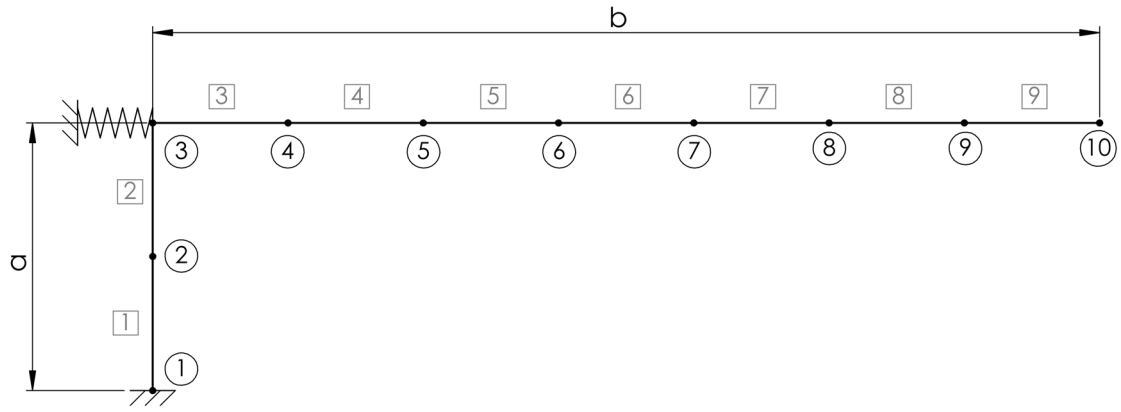
Obr. 4.3 Průběh síly F_p v závislosti na výchylce u_p

4.2 FEM model

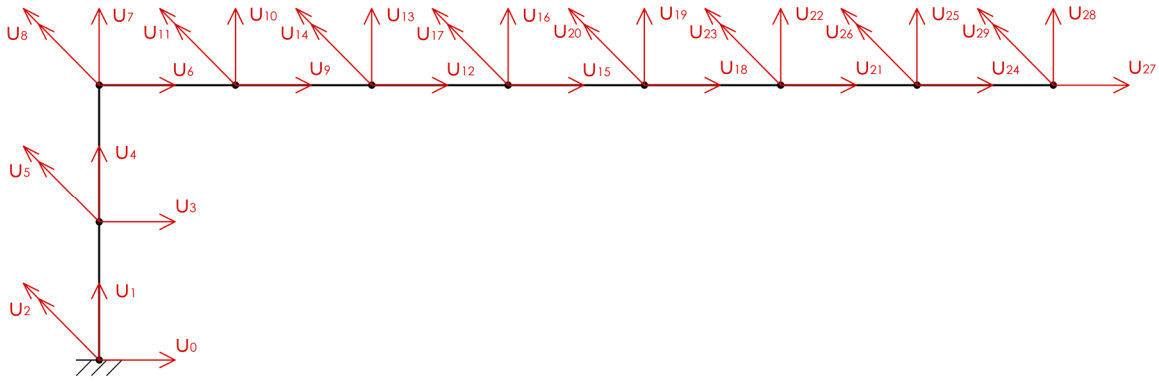
Existující mechanický model (například nosník ve tvaru L z předchozího příkladu) je třeba převést do formy, ve které by bylo možné provést simulaci úderu, vibrací a měření pohybu bodu. Nosník je v tuto chvíli považován za kontinuum, prvním krokem bude diskretizace kontinua pomocí metody konečných prvků (FEM), která je zpracována v rešerši. Cílem bude získání předpisu *(2-4),* kde matice *K, M, B* budou časově nezávislé, ovšem budou záviset na výchylkách $\underline{u}$. Vzhledem k tomu, že celý systém je po částech lineární, bude toho dosaženo vypočítáním dvou různých variant matic pro oba lineární časově nezávislé modely. Jinak řečeno, vznikne dvojce lineárních časově nezávislých modelů, z nichž jeden bude platný při $u_p < 0$, a druhý při $u_p > 0$.

FEM model bude vytvořen v 2D prostoru nosníkovými elementy, které byly popsány v kapitole 2.1. Rozdělení kontinua na elementy je naznačeno na Obr. 4.4 , čísla elementů jsou uvedena v rámečku, čísla globálních uzlů jsou uvedena v kroužku. Obr. 4.5 ukazuje model s vyznačenými

posuvy (stupni volnosti), zdvojená šipka značí natočení. Vetknutí bude realizováno homogenní okrajovou podmínkou $u_i = 0,\ i = 0, 1, 2$. Vliv pružiny bude realizován úpravou matice tuhosti.



Obr. 4.4 Model nosníku tvaru L rozdělený na elementy, čísla elementů v rámečcích, globální čísla uzlů v kroužcích



Obr. 4.5 Model nosníku tvaru L s globálními posuvy, šikmá dvojitá šipka označuje natočení

Výpočty FEM modelu jsou implementovány v příloze 9.3 FEM_computations (Python) – vlastní výpočetní program vytvořený v rámci této práce. Pro vytvoření FEM modelu je nejprve třeba vytvořit objekt třídy Model (ř.116), při inicializaci jsou zadány globální vlastnosti modelu jako Youngův modul pružnosti, hustota a podobně. Tím je vytvořen prázdný model, se kterým se manipuluje prostřednictvím jeho metod. Metoda add_element (ř.148) vytvoří objekt třídy Beam (ř.26) s informací ze kterého a do kterého globálního uzlu element vede a přidá jej do listu element_list. Pokud použitý globální uzel neexistuje, je do modelu přidán. Metoda add_constraint (ř.188) přidá směry, na které jsou aplikovány homogenní okrajové podmínky. Metoda add_measured_directions (ř.198) přidá směry, u kterých budou zaznamenávány jejich pohyby. Metoda add_impacted_dir (ř.206) přidá směr na který bude aplikováno silové buzení. Metoda add_spring_dir (ř.214) přidá směr, ve kterém bude působit pružina. Nakonec metoda compile_model (ř.166) nejprve vytvoří prázdné globální matice tuhosti a hmotnosti a direction_map – seznam globálních stupňů volnosti. Následně pro každý element vyvolá metodu elementu add2global_K a add2global_M (ř.97 a 106), které vlivy od těchto elementů přidají do globálních matic K a M podle rovnice *(2-12),* která je implementovaná ve funkci element_to_big_matrix (ř.9). Elementární matice tuhosti objektu třídy Beam se vypočítá sama při inicializaci (ř.38), metoda local_k (ř.45) implementuje rovnici *(2-6)* a získá lokální rovnici tuhosti v lokálních souřadnicích, metoda local_to_global následně převede lokální matici elementu do globálních souřadnic (ř.79) dle rovnice *(2-10).* Obdobným způsobem se zachází s maticí M. Následně metoda complie_model zavede homogenní okrajové podmínky dle *(2-13)* vymazáním

odpovídajících řádků a sloupců z matic *K* a *M* a vymazáním odpovídajících stupňů volnosti ze seznamu direction_map. Nakonec metoda vypočte Rayleigho matici tlumení dle *(2-14)*. Objekt třídy model nyní obsahuje globální matice *K, B* a *M* představující diskrétní popis zadaného rámu. Model zatím neobsahuje pružinu, ani silový impuls.

Objekt třídy model je následně předán jako vstup při inicializaci vzniku objektu třídy IntegrationProblem (ř.222). Aby byla práce s objektem uživatelsky přívětivější, jsou směrové vstupy všech metod zadávány ve směrech nemodifikované soustavy – bez zavedení okrajových podmínek, ty lze pro tuto chvíli označit pruhem nad označením. Příklad modifikace, kdy okrajová podmínka vynuluje směr $\overline{u_1}$ je uveden na *(4-1)* směry modifikované soustavy jsou uvedeny bez pruhu. Převod na směry modifikovaného systému umožňuje vnitřní metoda new_direction (ř.348), metoda prohledá list direction_map, ve kterém jsou zaznamenány původní směry soustavy, ale byly odstraněny ty, na které jsou aplikované homogenní okrajové podmínky. Pokud je metoda požádána o modifikovaný směr odpovídající starému, na který byla aplikována okrajová podmínka, zahlásí error. Pro uvedený příklad na *(4-1)* by $new\_direction(3) = 2$, ale $new\_direction(1) = error$.

$$\begin{bmatrix} \overline{u_0} \\ \overline{u_1} \\ \overline{u_2} \\ \overline{u_3} \\ \overline{u_4} \end{bmatrix} = \begin{bmatrix} u_0 \\ 0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \qquad (4\text{-}1)$$

Pro zavedení vlivu pružiny do modelu nejprve definujme vektor $\underline{v_k}$ označující, na který směr FEM modelu pružina působí dle *(4-2)*. Na zavedený model přidaná pružina působí jako vnější síla $\underline{f}(\underline{u}, t)$, tato síla je závislá na tuhosti pružiny k_p a výchylce u_p, konkrétní výchylka související s pružinou lze z vektoru výchylek získat dle *(4-4)*, využitím tohoto vztahu je možné získat předpis *(4-3)*, který zavádí matici $\widehat{H}$, která představuje přírůstek globální matice tuhosti *K* vlivem přidání pružiny, globální matice tuhosti systému se zavedenou pružinou je zavedena v *(4-5)* a označena K_l. Tento postup je implementován v metodě add_nonlinearity (ř.262) pro vznik nelineárního modelu. Lineární model, tedy model konstrukce bez poruchy, vznikne přidáním pružiny, která bude pevně připojena ke konstrukci a bude vytvářet tlakovou i tahovou sílu, taková pružina je přidána metodou add_spring (ř.281) obdobným způsobem. Významný rozdíl ovšem je, že metoda add_spring uchovává vektor $\underline{v_k}$ pod názvem vect_k, kdežto metoda add_nonlinearity stejný vektor pojmenuje vect_fu ($\underline{v_{fu}}$), význam tohoto kroku se ukáže později.

$$v_{k_i} = 0 \quad \forall i \neq p\,, \quad v_{k_i} = 1 \quad \forall i = p \qquad (4\text{-}2)$$

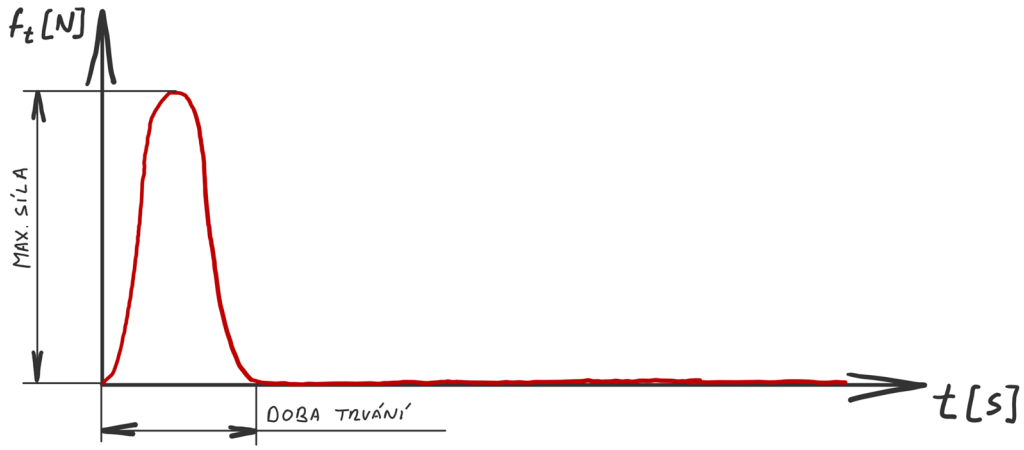
$$\underline{\underline{M}}\,\underline{\ddot{u}} + \underline{\underline{B}}\,\underline{\dot{u}} + \underline{\underline{K}}\,\underline{u} = \underline{f}(\underline{u}, t) = -\underline{v_k}\;\; k_p\;\; u_p = -\underline{v_k}\;\; k_p\;\underline{v_k}^T\,\underline{u} = -\underline{\underline{\widehat{H}}}\,\underline{u} \qquad (4\text{-}3)$$

$$u_p = \underline{v_k}^T\,\underline{u} \qquad (4\text{-}4)$$

$$\underline{\underline{M}}\,\underline{\ddot{u}} + \underline{\underline{B}}\,\underline{\dot{u}} + \underline{\underline{K}}\,\underline{u} + \underline{\underline{\hat{H}}}\,\underline{u} = \underline{\underline{M}}\,\underline{\ddot{u}} + \underline{\underline{B}}\,\underline{\dot{u}} + \underline{\underline{K_l}}\,\underline{u} = 0 \qquad (4\text{-}5)$$

Metoda add_impact (ř.299) přidá silové buzení, vstupem metody je směr, na který bude aplikována externí budící síla, maximální velikost síly a doba trvání silového impulzu. Metoda uloží buzený směr do seznamu, vytvoří vektor $\underline{v_{ft}}$, pro který platí *(4-6)*, kde q je index buzeného směru. Nakonec metoda vytvoří časově závislou funkci f_t, která napodobuje silový impuls kladívka, je tvořena posunutou funkcí cosinus a její průběh je naznačen na Obr. 4.6. Pokud je doba trvání nulová, je průběhem budící síly v čase Diracův impuls (prostřednictvím metody dirac_impact (ř.322)).

$$v_{ft_i} = 0 \quad \forall i \neq q\,, \quad v_{ft_i} = 1 \quad \forall i = q \qquad (4\text{-}6)$$



Obr. 4.6 Průběh síly f_t v závislosti na čase

Metoda rayleigh (ř.440) vypočte Rayleigho matici tlumení dle *(2-14)* pro zadané matice tuhosti a hmotnosti s konstantami, které jsou vlastní danému objektu třídy IntegrationProblem. Poslouží k výpočtu proporcionálního tlumení pro systém s přidanou pružinou, tato pružina tak bude mít, v rozporu s uvedeným schématem (Obr. 4.2), své vlastní tlumení.

4.3 Časová simulace

Pomocí FEM se podařilo úlohu diskretizovat v prostorové oblasti. Nyní je třeba model řešit v oblasti časové. Vzhledem k tomu, že se jedná o nelineární úlohu, je pro zjištění časové závislosti výchylek třeba provést integraci v časové oblasti. V tuto chvíli je model s nelinearitou určen dvojicí popisů ve tvaru *(2-22)*. Pro numerickou integraci je třeba tento tvar převést na stavový popis – soustavu diferenciálních rovnic prvního řádu.

4.3.1 Stavový popis

Pro danou úlohu existuje mnoho různých stavových popisů v závislosti na volbě stavů. Stavy odpovídající polohám a rychlostem se mohou zdát jako nejsnadnější první volba. Při převodu úlohy do modálních souřadnic dle *(2-32)* se soustava rozpadá na jednotlivé rovnice *(2-33)*, to ovšem neplatí pro nelineární úlohy. Nicméně volba stavů jako modální souřadnice a derivace

modální souřadnice přinese výhody během dalšího postupu. Výpočet stavového popisu je implementován v metodě get_ss (ř.386). Na (ř.394) jsou zjištěny matice vlastních čísel a vlastních vektorů matice *K* vážené dle matice *M* pomocí funkce knihovny SciPy scipy.linalg.eigh podle *(2-25)*, *(2-26)* a *(2-27)*. Matice systému v modálních souřadnicích jsou získány dle *(2-28)*, *(2-29)* a *(2-30)*.

Stavový popis má podobu *(2-18)*. Pro rovnici *(2-32)*, vektor stavů volený dle *(4-7)* a vektor výstupů zvolený dle *(4-8)* mají matice stavového popisu podobu dle: *(4-9)*, *(4-10)*, *(4-11)* a *(4-12)*, kde *Z* je nulová matice. Matice $\hat{C}$ provádí transformaci do původních souřadnic $\underline{u}$ dle *(2-31)*, výstupem stavového popisu jsou tedy $\underline{u}$ a $\underline{\dot{u}}$. Metoda get_all_ss (ř.450) vypočte matice stavového popisu pro dva systémy, jeden označen indexem *k*, druhý *l*. Systém označený indexem *l* je vypočten pro matici tuhosti K_l, tedy tu, ke které byl přidán nelineární vliv od pružiny.

$$\underline{s} = \begin{bmatrix} \underline{q} \\ \underline{\dot{q}} \end{bmatrix} \tag{4-7}$$

$$\underline{y} = \begin{bmatrix} \underline{u} \\ \underline{\dot{u}} \end{bmatrix} \tag{4-8}$$

$$\underline{\underline{\hat{A}}} = \begin{bmatrix} \underline{\underline{Z}} & \underline{\underline{E}} \\ -\underline{\underline{\Lambda}} & -\underline{\underline{B_p}} \end{bmatrix} \tag{4-9}$$

$$\underline{\underline{\hat{B}}} = \begin{bmatrix} \underline{\underline{Z}} \\ \underline{\underline{V^T}} \end{bmatrix} \tag{4-10}$$

$$\underline{\underline{\hat{C}}} = \begin{bmatrix} \underline{\underline{V}} & \underline{\underline{Z}} \\ \underline{\underline{Z}} & \underline{\underline{V}} \end{bmatrix} \tag{4-11}$$

$$\underline{\underline{\hat{D}}} = \begin{bmatrix} \underline{\underline{Z}} \\ \underline{\underline{Z}} \end{bmatrix} \tag{4-12}$$

4.3.2 Řešení ODE

K přepínání mezi systémy *k* a *l* dochází na základě hodnoty výchylky $\underline{u}$, tedy na základě výstupu $\underline{y}$, nikoliv na základě modální souřadnice, tedy ani na základě stavů $\underline{s}$. Standardně by během řešení soustavy diferenciálních rovnic stačilo integrovat stavy, nicméně z uvedených důvodů bude muset v tomto případě během integrace probíhat i zjišťování výstupů $\underline{y}$. Výpočet výstupu je implementován v metodě output_y (ř.459), vstupem metody je vektor stavů, pomocí matice $\hat{C}_k$ je získán vektor výstupů systému $\underline{y_k}$, první polovina tohoto vektoru je tvořena výchylkami $\underline{u}$. Vektor $\underline{v_{fu}}$ byl vytvořen metodou add_nonlinearity dle *(4-2)*, pokud nelinearita nebyla přidána, je vektor tvořen pouze nulami. Souřadnice u_p je zjištěna dle *(4-13)*, pokud je větší než nula, je aktivován systém *l*, vypočten výstup $\underline{y_l}$ a binární výstup isL je nastaven na hodnotu true.

$$u_p = \underline{v_{fu}}^T \cdot \underline{u} \tag{4-13}$$

Metoda start_integration_csv (ř.535) třídy IntegrationProblem provede numerickou integraci pro simulaci vytvořeného modelu, funkce get_B (ř.552) uvnitř metody pro dané stavy vrátí aktuální matici $\hat{B}$, funkce jacobian (ř.564) pro dané stavy vrátí aktuální matici $\hat{A}$, funkce tedy rozhodují, zda je zapojen systém *k*, nebo *l*. Na základě *(2-18)* a *(2-17)* lze sestavit *(4-14)* ukazující, že matice $\hat{A}$ je také Jacobiho maticí. Funkce int_problem (ř.577) pro stavy $\underline{s}$ a čas *t* vypočítává $\underline{\dot{s}}$. Pokud ovšem integrace trvá delší dobu než nastavený časový limit, tato funkce vrací pouze nuly. Toto opatření je zavedeno pro případ, kdy se integrace „zasekne", nulový výstup potom vede k rychlému dokončení integrace, jejíž výsledek je poté odstraněn. Numerická integrace je provedena pomocí funkce scipy.integrate.solve_ivp, jejímž vstupem je, mimo jiné, funkce int_problem, funkce jacobian jako Jacobiho matice a metoda pro řešení soustavy.

$$\underline{\underline{J}}(t, \underline{s}) = \frac{\partial \underline{\dot{s}}}{\partial \underline{s}} = \frac{\partial(\underline{\underline{\hat{A}}}\,\underline{s} + \underline{\underline{\hat{B}}}\,\underline{f})}{\partial \underline{s}} = \underline{\underline{\hat{A}}} \tag{4-14}$$

4.3.3 Celková energie soustavy

Pokud není známo analytické řešení úlohy, je obtížné ověřit, že numerická integrace nalezla správné řešení. Jedním z možných vodítek může být výpočet celkové energie soustavy z výsledných stavů. Celková energie soustavy odpovídající danému modelu zůstává konstantní, pokud není dodávána externí silou. Tudíž lze říci, že výsledek vedoucí k proměnné celkové energii není správný. Na druhou stranu konstantní hodnota celkové energie nezaručuje správnost výsledku.

Kinetická energie a potenciální energie jsou uvedeny na *(2-15)* a *(2-16)*. Model ovšem obsahuje také tlumení, které způsobuje disipaci energie. Rovnice *(2-22)* vyjadřuje silovou rovnováhu vnějších sil se silami setrvačnosti, silami od tlumení a silami potenciálními. Síly od tlumení tak odpovídají *(4-15)*. Výkon ztracený tlumením odpovídá síle násobené rychlostí dle *(4-16)*. Energie disipovaná v tlumení odpovídá integraci výkonu P_r. Integraci lze aproximovat součtem jednotlivých diskrétních hodnot dle *(4-17)*.

$$\underline{F_r}(t) = \underline{\underline{B}}\,\underline{\dot{u}}(t) \tag{4-15}$$

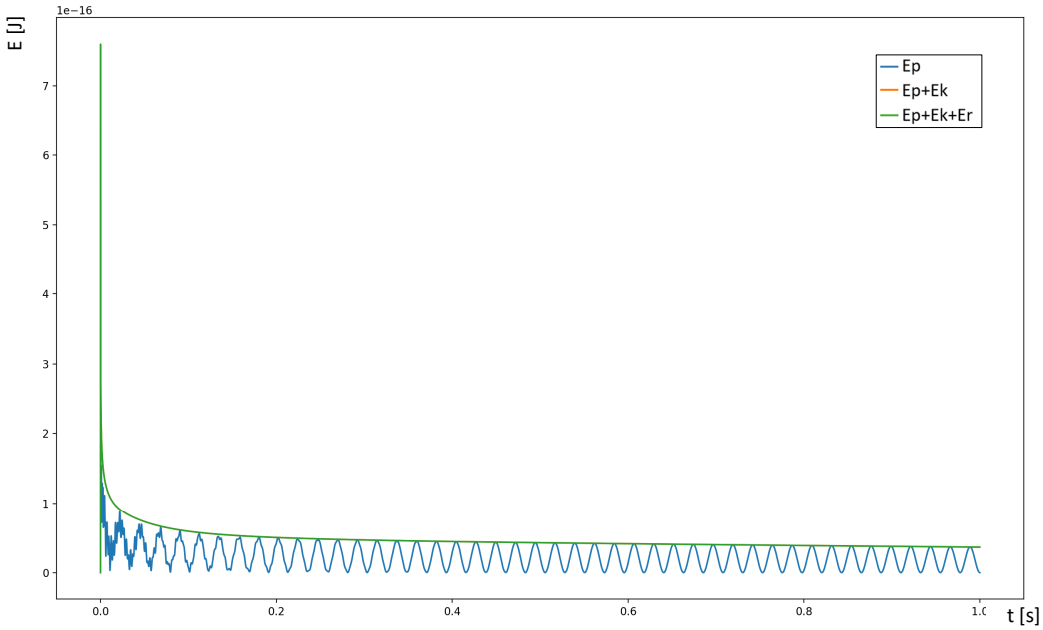
$$P_r(t) = \underline{F_r}^T\,\underline{\dot{u}}(t) = \underline{\dot{u}}^T(t)\,\underline{\underline{B}}^T\,\underline{\dot{u}}(t) = \underline{\dot{u}}^T(t)\,\underline{\underline{B}}\,\underline{\dot{u}}(t) \tag{4-16}$$

$$E_r(t) = \int_0^t P_r(t)\,dt \cong \sum_{k=0}^{N} P_r(\Delta t \cdot k) \cdot \Delta t \tag{4-17}$$

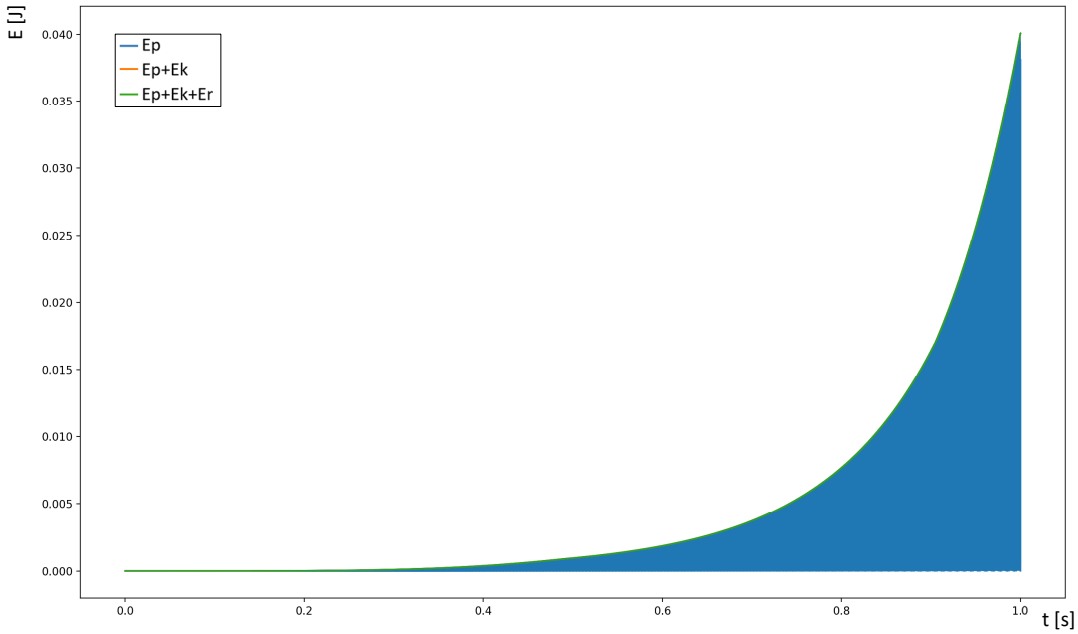
Výpočet energie kinetické, potenciální a disipované probíhá po numerické integraci prostřednictvím metody post_simulation_integration (ř.607), která bere v potaz i proměny v maticích B a K v průběhu integrace vlivem „spínání“ nelinearity. Poté co je ukončen silový impuls, měl by součet uvedených energií být konstantní. Metoda energy_difference porovná hodnoty celkové energie v určitém časovém bodě po začátku integrace a v určitém časovém bodě před koncem integrace. Po provedení integrace dojde k porovnání procentuální změny energie se stanovenou tolerancí. Pokud je změna energie větší, je výsledek integrace „zahozen“ a metoda start_integration_csv zahlásí překročení změny energie.

4.3.4 Volba řešiče ODE

Pro ideální výsledky integrace je možné volit metodu řešení, maximální velikost kroku, u některých metod limity odhadu relativní a absolutní chyby. Cílem je minimalizovat dobu výpočtu a maximalizovat přesnost. Parametry integrace byly voleny na příkladu simulace modelu bez poruchy.



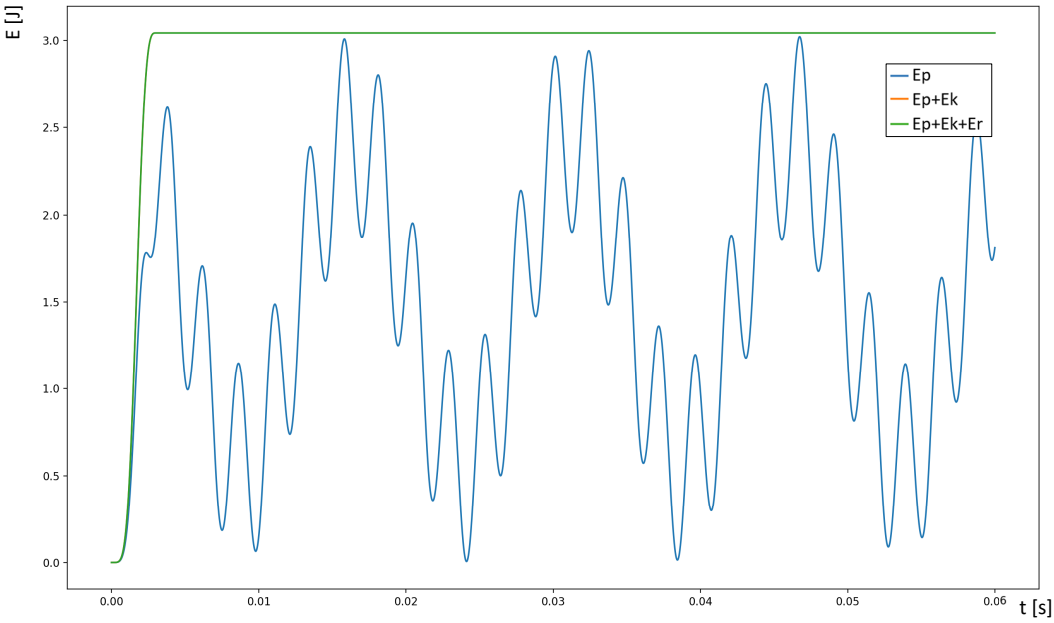
Obr. 4.7 Průběh celkové energie soustavy bez tlumení, Diracův impuls, integrace pomocí BDF, frekvence 20 000 Hz



Obr. 4.8 Průběh celkové energie soustavy bez tlumení, Diracův impuls, integrace pomocí RK45, frekvence 20 000 Hz

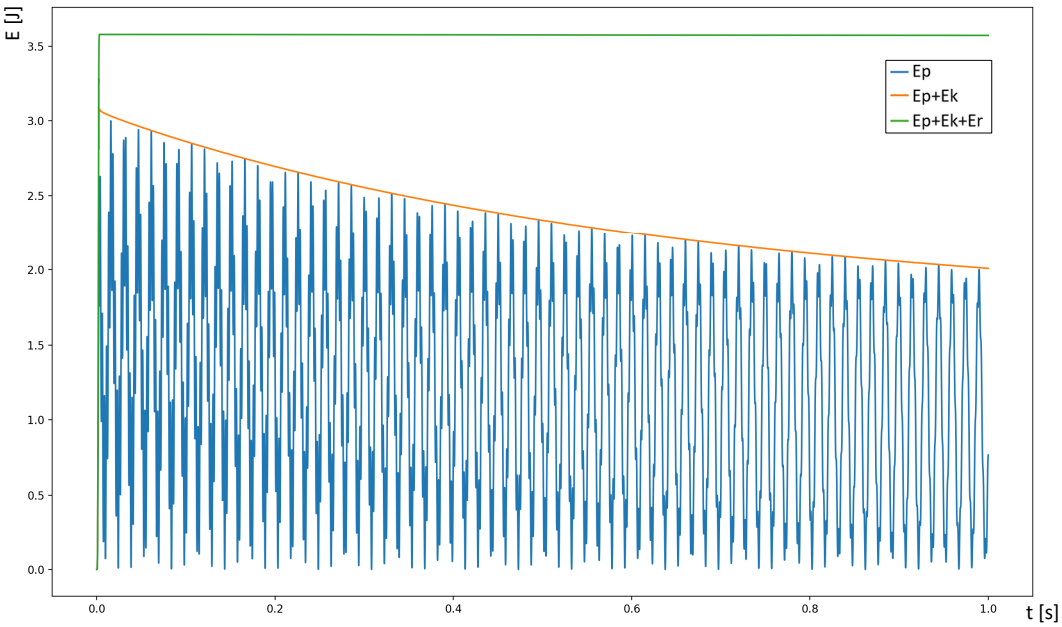
Při porovnání Obr. 4.7 a Obr. 4.8 je zřejmé, že metoda RK45 nedává dobré výsledky, zdá se, že energie exponenciálně narůstá. Důvodem může být velký rozdíl mezi nejmenší a největší vlastní frekvencí soustavy. Jak ukazuje Obr. 2.1, element může být ohýbán i stlačován, přičemž v tlaku může mít výrazně větší tuhost jak v ohybu, jiným důvodem mohou být rozdílně velké elementy. Podle *(2-25)* lze před počátkem integrace analyticky zjistit vlastní frekvence, pro náhodně vybranou úlohu je nejnižší vlastní frekvence 1.306e+01 Hz, a nejvyšší vlastní frekvence 1.413e+08 Hz. Lze se tak domnívat, že jde o takzvaný stiff problém, který je vhodné řešit pomocí stiff řešiče, jako je například BDF.

Obr. 4.7 ukazuje průběh celkové energie naznačující, že dochází ke ztrátě energie. V tomto konkrétním systému ovšem není zavedeno tlumení. Důvodem poklesu energie tak může být numerické tlumení úlohy, jak je popsáno v kapitole 2.3. Dojde tak k tomu, že výsledek je stabilní, ovšem nelze jej označit za správný. Při zmenšení kroku je výsledek pro stejnou úlohu na Obr. 4.9, integrace v tomto případě trvala extrémně dlouhou dobu, proto byl integrován pouze kratší časový úsek. Tento výsledek již vypadá věrohodně, nárůst energie na počátku je způsoben impulsem síly, energie se poté přelévá mezi potenciální a kinetickou, celkové množství energie se ale nemění.

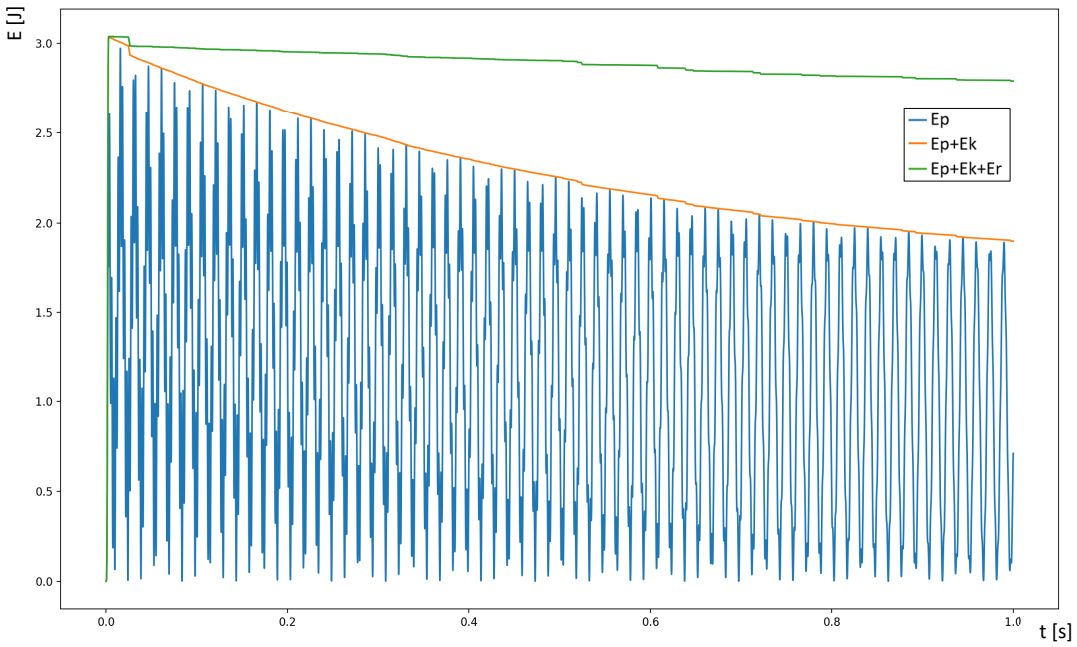


Obr. 4.9 Průběh celkové energie soustavy bez tlumení, integrace pomocí BDF, frekvence 160 000 Hz

Přidání tlumení do úlohy umožní pro stejně nastavený řešič rychlejší průběh výpočtu s výsledkem na Obr. 4.10, dochází opět k periodické výměně energie mezi potenciální a kinetickou, hodnota součtu těchto dvou energií ovšem s časem klesá. Množství disipované energie v čase pomalu narůstá, jak odpovídá fyzikální představě. Přidání vlastního tlumení soustavy umožní zachování přijatelné přesnosti i při snížení frekvence numerické integrace, jak ukazuje Obr. 4.11.



Obr. 4.10 Průběh celkové energie soustavy s tlumením, integrace pomocí BDF, frekvence 160 000 Hz



Obr. 4.11 Průběh celkové energie soustavy s tlumením, integrace pomocí BDF, frekvence 20 000 Hz

4.3.5 Redukce modelu

Pomocí použití stiff řešiče se sice podařilo dosáhnout věrohodných řešení úlohy i přes vysoké vlastní frekvence. Nicméně nejspíš se tak nepodaří vyhnout se možnému aliasingu, který je popsaný v kapitole 2.4. Vzorkovací frekvence integrace by musela být řádově okolo 1e+08 Hz, aby bylo aliasingu zabráněno, to by ovšem mohlo přinést zvýšení vlivu zaokrouhlovací chyby numerické integrace. Je sice možné předpokládat, že při vybuzení pouze nízkých frekvencí nebude vliv aliasingu významný, ovšem pro pozorování drobných vlivů nelinearit je vhodné se i takovým potenciálním vlivům vyvarovat.

Redukce modelu umožní zanedbání vlivů některých stavů. Při tvorbě stavového popisu bylo uvedeno, že volba stavů v podobě modálních souřadnic přinese v budoucnosti určité výhody. Ta chvíle přichází nyní, kdy každý mód a modální frekvence odpovídá modální souřadnici. Při zanedbání stavů odpovídajících modální souřadnici je zanedbán vliv daného módu, sníží se počet vlastních frekvencí. Redukcí stavů odpovídajících vysokým vlastním frekvencím je tak možné vytvořit redukovaný model, který již nebude vysoké vlastní frekvence obsahovat, nepůjde již o stiff úlohu. Pokud indexy módů, které mají být odstraněny označíme *b* a indexy ponechaných *a*, lze stavový systém rozdělit dle *(2-19)*, při aplikaci rozdělení na stavový popis dle *(4-9)*, *(4-10)*, *(4-11)* a *(4-12)* získáme vztahy *(4-18)* a *(4-19)*. Při redukci typu truncation dle *(2-20)*, získáme redukovaný stavový popis *(4-20)* a *(4-21)*. Redukce truncation neposkytuje tak dobrou aproximaci původního stavového popisu v oblasti nízkých frekvencí, to ovšem v tomto případě není problém. Pro tuto úlohu postačí zachování základního chování systému.

$$\begin{bmatrix} \underline{\dot{q}_a} \\ \underline{\dot{q}_b} \\ \underline{\ddot{q}_a} \\ \underline{\ddot{q}_b} \end{bmatrix} = \begin{bmatrix} \underline{\underline{Z}} & \underline{\underline{Z}} & \underline{\underline{E}} & \underline{\underline{Z}} \\ \underline{\underline{Z}} & \underline{\underline{Z}} & \underline{\underline{Z}} & \underline{\underline{E}} \\ -\underline{\underline{\Lambda_{aa}}} & -\underline{\underline{\Lambda_{ab}}} & -\underline{\underline{B_{p}}}_{aa} & -\underline{\underline{B_{p}}}_{ab} \\ -\underline{\underline{\Lambda_{ba}}} & -\underline{\underline{\Lambda_{bb}}} & -\underline{\underline{B_{p}}}_{ba} & -\underline{\underline{B_{p}}}_{bb} \end{bmatrix} \begin{bmatrix} \underline{q_a} \\ \underline{q_b} \\ \underline{\dot{q}_a} \\ \underline{\dot{q}_b} \end{bmatrix} + \begin{bmatrix} \underline{\underline{Z}} \\ \underline{\underline{Z}} \\ \underline{\underline{V^T}}_a \\ \underline{\underline{V^T}}_b \end{bmatrix} \underline{f} \qquad (4\text{-}18)$$

$$\underline{y} = \begin{bmatrix} \underline{\underline{V_a}} & \underline{\underline{V_b}} & \underline{\underline{Z}} & \underline{\underline{Z}} \\ \underline{\underline{Z}} & \underline{\underline{Z}} & \underline{\underline{V_a}} & \underline{\underline{V_b}} \end{bmatrix} \begin{bmatrix} \underline{q_a} \\ \underline{q_b} \\ \underline{\dot{q}_a} \\ \underline{\dot{q}_b} \end{bmatrix} + \begin{bmatrix} \underline{\underline{Z}} \\ \underline{\underline{Z}} \end{bmatrix} \underline{f} \qquad (4\text{-}19)$$

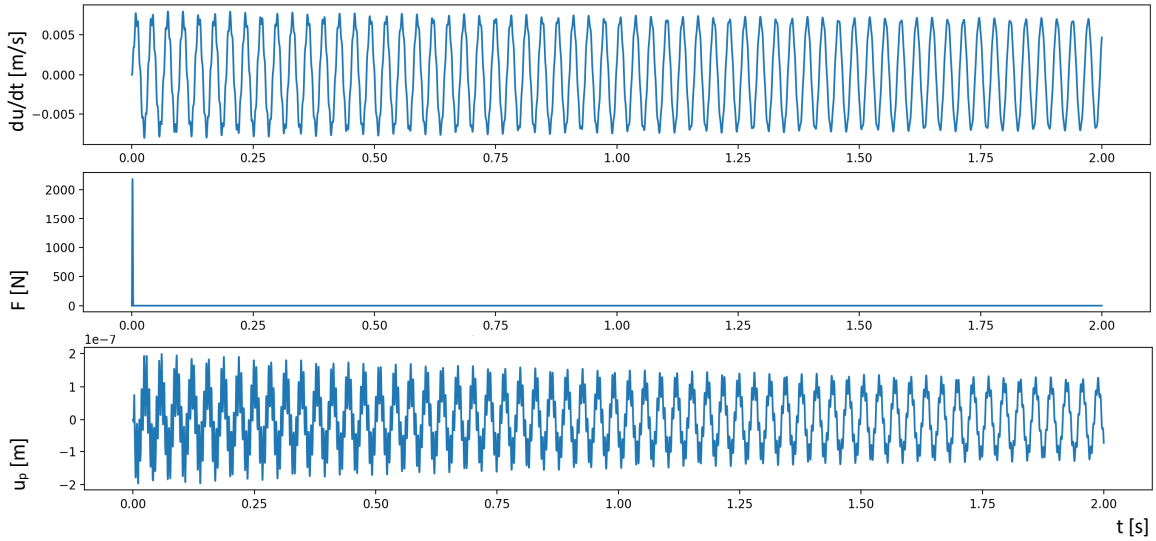
$$\begin{bmatrix} \underline{\dot{q}_a} \\ \underline{\ddot{q}_a} \end{bmatrix} = \begin{bmatrix} \underline{\underline{Z}} & \underline{\underline{E}} \\ -\underline{\underline{\Lambda_{aa}}} & -\underline{\underline{B_{p}}_{aa}} \end{bmatrix} \begin{bmatrix} \underline{q_a} \\ \underline{\dot{q}_a} \end{bmatrix} + \begin{bmatrix} \underline{\underline{Z}} \\ \underline{\underline{V^T}_a} \end{bmatrix} \underline{f} \qquad (4\text{-}20)$$

$$\underline{y} = \begin{bmatrix} \underline{\underline{V_a}} & \underline{\underline{Z}} \\ \underline{\underline{Z}} & \underline{\underline{V_a}} \end{bmatrix} \begin{bmatrix} \underline{q_a} \\ \underline{\dot{q}_a} \end{bmatrix} + \begin{bmatrix} \underline{\underline{Z}} \\ \underline{\underline{Z}} \end{bmatrix} \underline{f} \qquad (4\text{-}21)$$

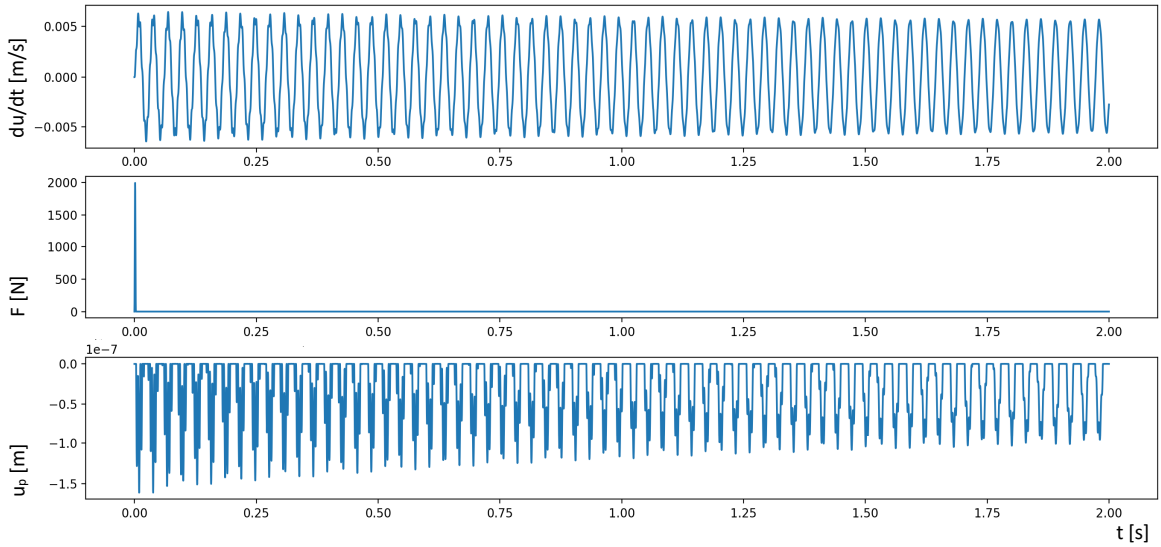
Redukce modelu je implementována v již zmiňované metodě get_ss (ř.386), kde jsou vypočteny vlastní frekvence, jsou zjištěny indexy stavů, jejichž frekvence překračují určenou vlastní frekvenci, následně je proveden postup uvedený na rovnicích v této kapitole. Na (ř.402) je také vytvořen postup, předcházející situaci, kdy by ze stavového popisu *k* byl odstraněn jiný počet módů než ze stavového popisu *l*. K tomu by mohlo docházet v situacích, kdy je jedna z vlastních frekvencí blízká maximální frekvenci oříznutí.

4.3.6 Výsledky simulace

Výsledky simulací redukovaných soustav jsou zobrazeny na grafech níže. Kromě měřeného výstupu v podobě rychlosti jednoho z bodů soustavy a impaktu budícího silového pulzu je na grafu zobrazena také souřadnice u_p. Tu se na reálné součásti nejspíš nepodaří naměřit, jelikož jde o výchylku uvnitř poruchy. Na simulaci se ovšem snadno zobrazí a může sloužit k ověření funkčnosti modelu poruchy. Na Obr. 4.12 spodní graf ukazuje výchylku v místě, kde by se v případě modelu s poruchou nacházela porucha. Jelikož zde není, je kmitání symetrické. Obr. 4.13 ukazuje výsledky modelu s poruchou, na grafu nahoře je možné si všimnout, že na měřitelných datech v časové oblasti nelze pouhým okem poruchu odhalit. Přítomnost poruchy je ovšem zřejmá z grafu dole na obrázku, dochází zde k narážení na nelineární pružinu fungující pouze při stlačení.

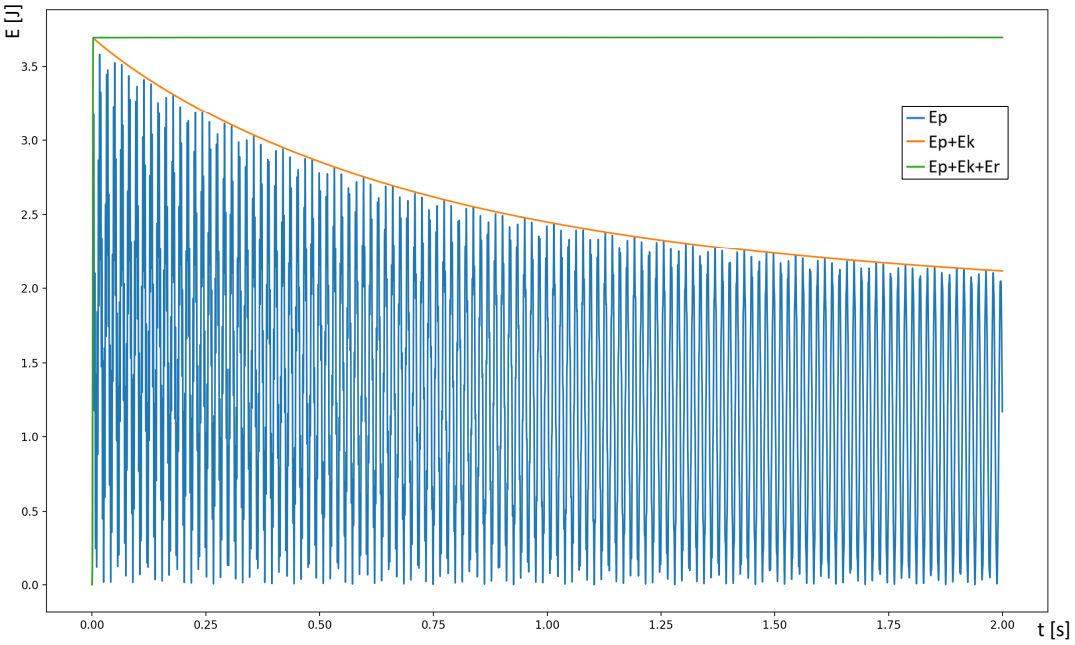


Obr. 4.12 Výsledky modelu bez poruchy (odshora): měřená rychlost uzlu, silový budící impuls, výchylka souřadnice v místě, kde by byla porucha

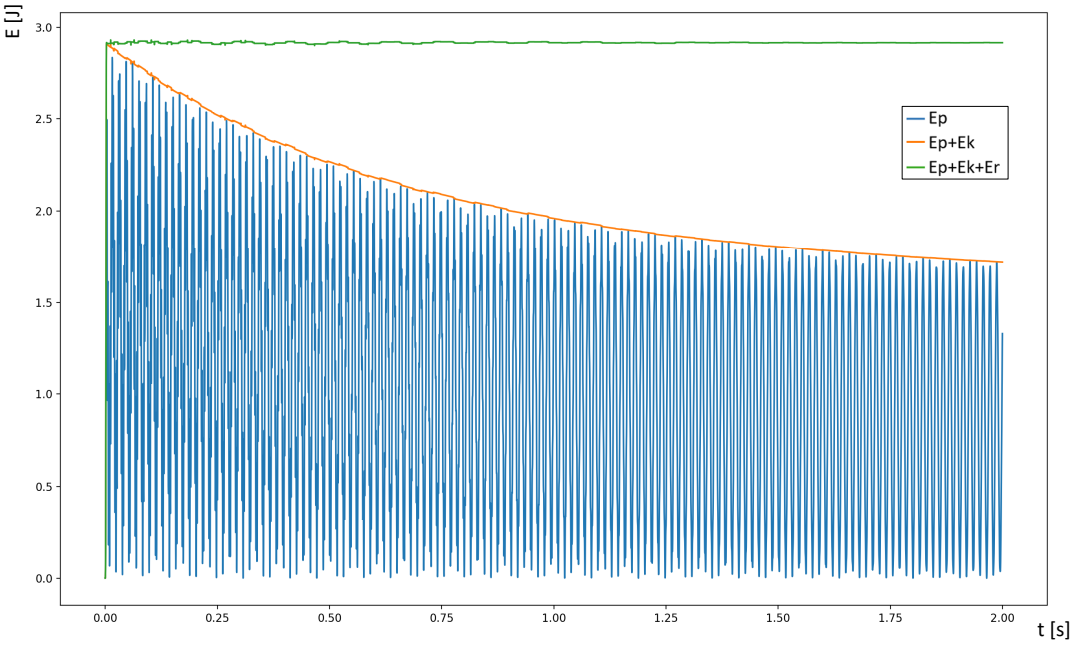


Obr. 4.13 Výsledky modelu s poruchou (odshora): měřená rychlost uzlu, silový budící impuls, výchylka v místě poruchy

Díky redukci modelu bylo možné velmi výrazně zrychlit numerickou integraci, byla získána věrohodná data i při měřící frekvenci 2000 Hz a bylo možné použít metodu RK45. Obr. 4.14 ukazuje průběh celkové energie systému bez poruchy. Systém s poruchou (Obr. 4.15) má jemně nekonstantní průběh celkové energie, ale jde skutečně jen o minimální odchylku.



Obr. 4.14 Průběh celkové energie soustavy bez poruchy, integrace pomocí RK45, frekvence 2 000 Hz



Obr. 4.15 Průběh celkové energie soustavy s poruchou, integrace pomocí RK45, frekvence 2 000 Hz

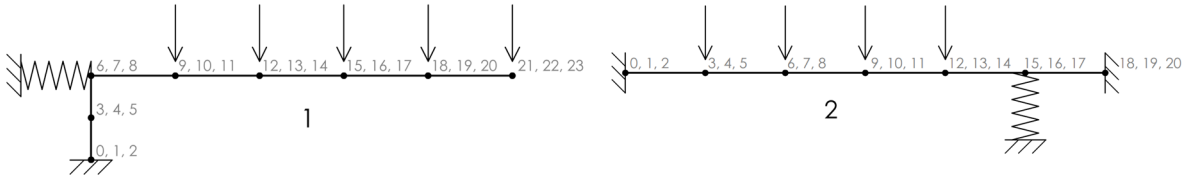
4.4 Tvorba dat pro neuronové sítě

Pro účely trénování neuronových sítí je obvykle třeba zajistit velké množství trénovacích dat. Aby se předešlo přetrénování, platí obecně, že čím větší množství parametrů má síť, tím více je třeba trénovacích dat. Pokud má síť málo parametrů, může se stát, že nebude schopna vytvořit dostatečně komplexní transformace dat, aby splnila požadované cíle. Proto pokud je to možné, je vhodné vytvořit trénovacích dat co nejvíce. Ideální by bylo provést mnoho měření na reálných konstrukcích, ovšem takový postup je velmi náročný. Alternativou může být trénovací data nasimulovat a pokud jsou k dispozici data z reálného měření, použít je pro testování. Důležitou

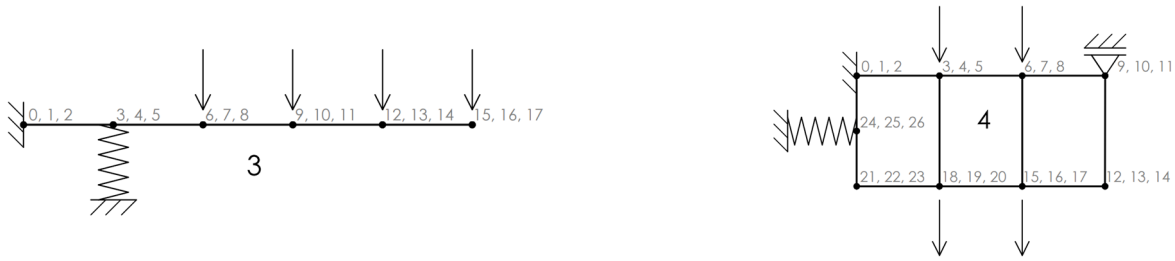
vlastností trénovacích dat však není jen jejich množství, ale také jejich „rozmanitost“. Pokud by například úkolem pro síť byla klasifikace fotografií na snímky psů a koček, ale trénování proběhlo jen na Labradorech a Perských kočkách, je možné, že klasifikace Jezevčíka by nedopadla úspěšně. Obecně tak lze říci, že trénovací data by měla být alespoň tak rozmanitá, jako budou data, na kterých je síť používána.

4.4.1 Modely rámových konstrukcí

Již byl popsán model rámové konstrukce ve tvaru L (Obr. 4.4), pouhou změnou rozměrů by nejspíš nevzniklo dostatečně rozmanité spektrum trénovacích dat, pokud má metoda identifikovat poruchy na různých konstrukcích, je třeba ji na různých konstrukcích trénovat. Za tímto účelem byly vytvořeny různé modely číslované jako tvary od jedné do patnácti. Jsou zobrazeny na Obr. 4.16 až Obr. 4.23. Čísla u každého uzlu představují indexy globálních posuvů nemodifikované soustavy v pořadí: posuv ve směru x, posuv ve směru y, rotace. Šipky označují směry, které mohou být buzeny silovým impaktem. Symbol pružiny označuje umístění nelinearity vytvořené stejným způsobem, jako v případě rámové konstrukce tvaru L.

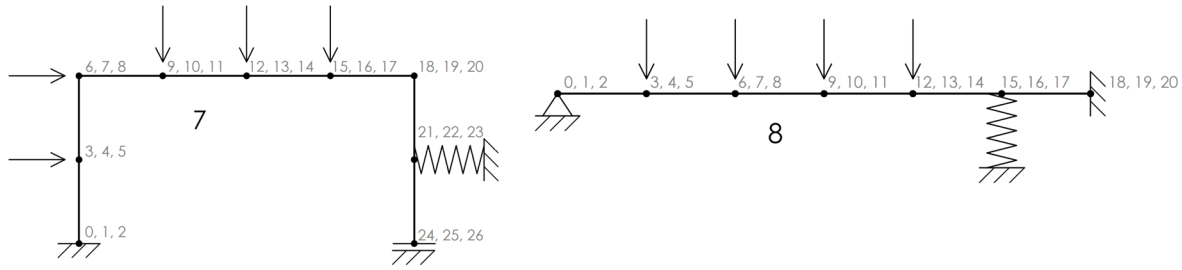


Obr. 4.16 FEM model, tvar 1 (vlevo) a tvar 2 (vpravo)

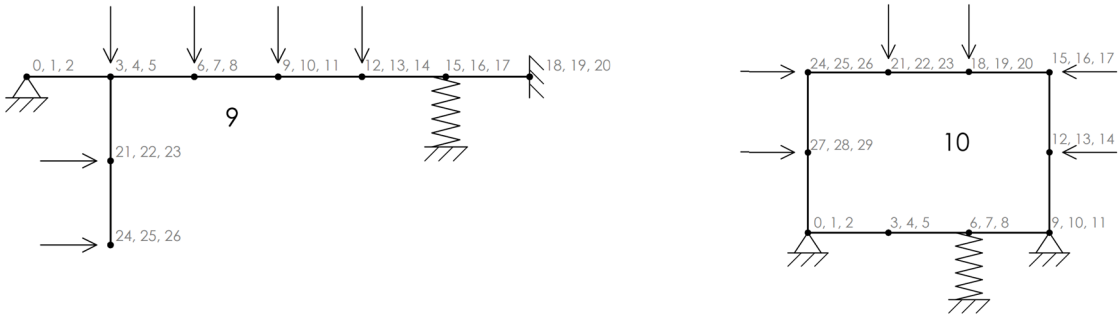


Obr. 4.17 FEM model, tvar 3 (vlevo) a tvar 4 (vpravo)

Obr. 4.18 FEM model, tvar 5 (vlevo) a tvar 6 (vpravo)

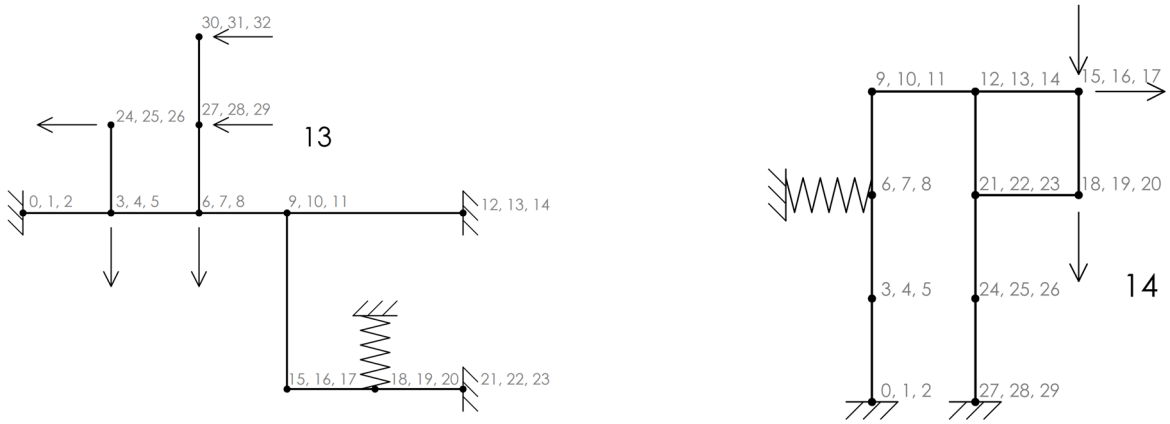


Obr. 4.19 FEM model, tvar 7 (vlevo) a tvar 8 (vpravo)

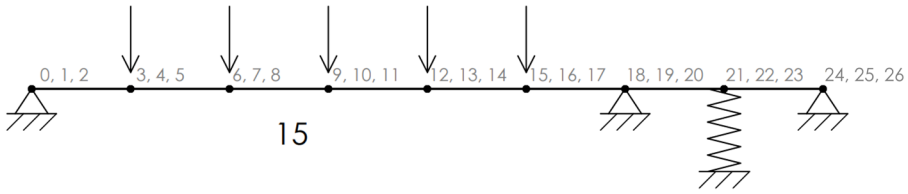


Obr. 4.20 FEM model, tvar 9 (vlevo) a tvar 10 (vpravo)

Obr. 4.21 FEM model, tvar 11 (vlevo) a tvar 12 (vpravo)



Obr. 4.22 FEM model, tvar 13 (vlevo) a tvar 14 (vpravo)



Obr. 4.23 FEM model, tvar 15

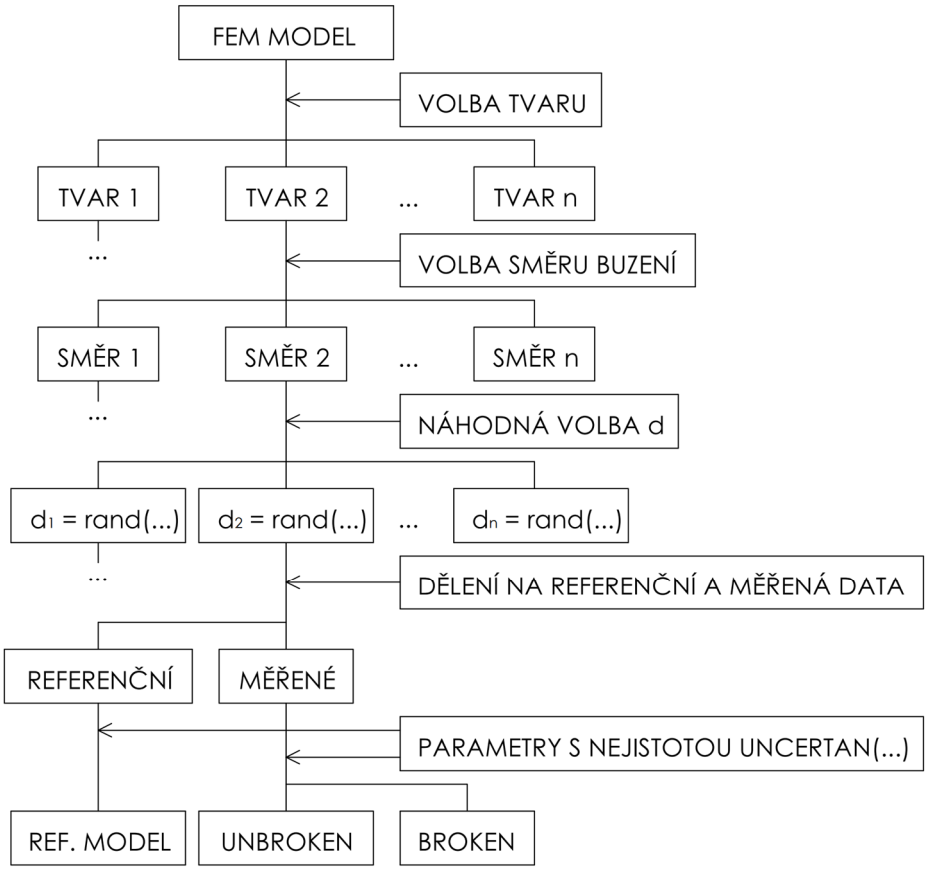
Pro každý model je v souboru v příloze 9.2 shapes_csv (Python) vytvořena funkce model1 až model15. Funkce vrací objekt třídy Model ze souboru 9.3 FEM_computations (Python). Uvnitř funkce dojde k použití parametrů modelu, uvedených v souboru v příloze 9.4 constants (Python). Tyto parametry jsou násobeny náhodnou hodnotou, aby byly drobně upraveny. Je vytvořen objekt třídy model, přidány elementy, vazby a měřené směry.

4.4.2 Referenční model

V kapitole 3 Cíle práce byly uvedeny tři úrovně úkolů pro strojové učení. Pro druhou úroveň se očekává získání referenčního měření nepoškozené konstrukce, se kterým se následně měření zkoumané konstrukce bude srovnávat. Tyto dvě konstrukce budou stejného typu, budou mít přibližně stejné rozměry, ovšem nejde o dvě totožné konstrukce, mohou mít do určité míry odlišné rozměry, materiálové parametry a podobně. Protože jsou trénovací data vytvořena simulací, musí simulace tuto odlišnost mezi referenčním a vyhodnocovaným měřením zohlednit.

Tvorba modelů pro trénovací data je znázorněna na schématu na Obr. 4.24. Nejprve je zvolen tvar z uvedených patnácti, pro vybraný tvar je z možností vybrán směr, na který je aplikováno buzení. Následně je v určitém rozmezí vybráno náhodné reálné číslo *d*, kterým se násobí rozměrové parametry. Na této úrovni modely odpovídají určité výrobní řadě se stejnými rozměry. Nyní dochází k dělení na referenční a vyhodnocovaný model, pro každý z těchto modelů je zvlášť na všechny parametry aplikována náhodná změna o určitou procentuální hodnotu. Tím se simuluje odlišnost mezi jednotlivými konstrukcemi stejné výrobní řady v materiálových a rozměrových parametrech. Vyhodnocovaný model je následně rozdělen na model s poruchou a bez poruchy.

Jsou tak vytvořeny tři modely: referenční model, model bez poruchy, model s poruchou. Při simulaci těchto tří modelů vznikne řada simulovaných výstupů pro každý model. Pokud jsou kroky tohoto schématu provedeny znovu, vznikne opět řada simulovaných výstupů, které ale budou pravděpodobně simulovány na konstrukci s jiným tvarem, jinými rozměry a buzeným na jiném místě.



Obr. 4.24 Schéma tvorby modelů s různými parametry

4.4.3 Simulace s proměnnými parametry

Aby bylo možné automaticky vytvářet trénovací data, funkce simulate_shape (ř.128) v příloze 9.2 shapes_csv (Python) přijímá za vstup proměnnou shape, maximální dobu integrace, frekvenci vzorkování simulace a cesty, kam mají být uloženy výsledky. Funkce zvolí náhodnou hodnotu *d* v rozmezí od ¼ do 4. Následně je pro hodnotu shape zvolena kombinace modelu a směru buzení, pro danou hodnotu *d* jsou vytvořeny referenční model a vyhodnocovaný model. Pro tyto dva modely je následně spuštěna funkce add_impacts (ř.14), tato funkce vytvoří objekt třídy IntegrationProblem, přidá impaktní buzení, pro referenční model a model bez poruchy přidá pružinu, k modelu s poruchou přidá nelineární pružinu. Poté provede simulaci pro všechny tyto modely a uloží výsledky pro měřené směry do csv tabulky v případě, že všechny simulace byly úspěšné. Pokud například jedna z integrací překročí limit pro změnu celkové energie, jsou odhozeny výsledky i ostatních simulací z dané trojce.

5 Experimentální modální analýza

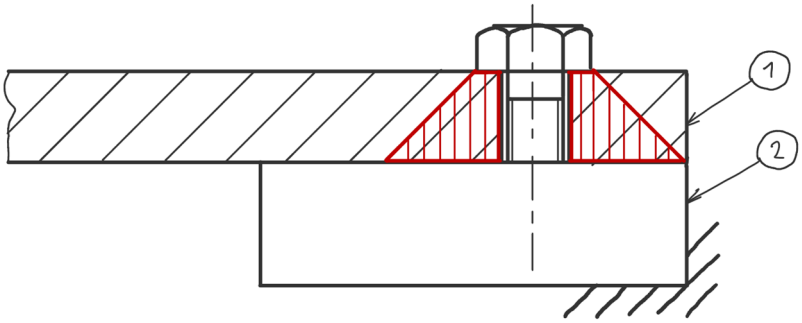
V předchozí kapitole byla pomocí simulace vytvořena data, která mají napodobovat reálná měření vibrací konstrukce s poruchou s „odléháním“ a bez ní. Aby byla ověřena relevance těchto dat, bude v této kapitole provedeno měření na reálné konstrukci. Cílem bude provedení měření konstrukcí s napodobenou poruchou a bez napodobené poruchy.

5.1 Napodobení poruchy v laboratorním prostředí

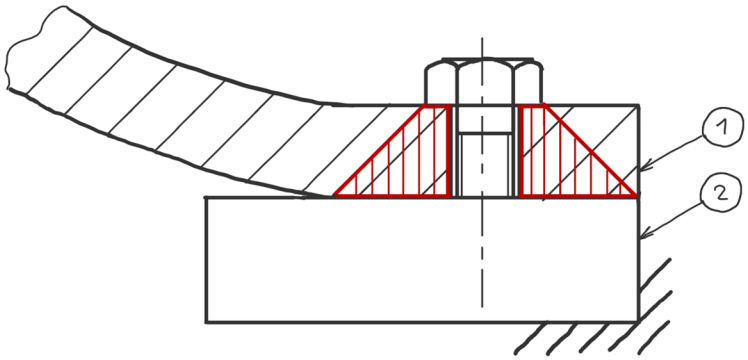
Existuje více možností, jak na reálné konstrukci napodobit poruchu typu „odléhání a nárazy“. Aby byla měření co nejlépe interpretovatelná, bude nejvhodnější provést měření případu bez poruchy a s poruchou na totožné konstrukci. Rozdílnost naměřených dat tak bude způsobena pouze poruchou, nikoliv odlišností mezi konstrukcemi, která by do jisté míry vznikla i při nejlepší snaze vytvořit konstrukce co nejshodnější. Jednou z možností je naměření dat na konstrukci, následné vytvoření praskliny na této konstrukci a provedení dalšího měření.

Pro snadné napodobení poruchy však v této práci bude využito odléhání ve šroubovém spoji. V kapitole 2.6 byl popsán tlakový kužel vytvořený pod hlavou utaženého šroubu. Na Obr. 5.1 jsou součásti 1 a 2 spojeny jedním šroubem, je zde červeně označený tlakový kužel. Do určité oblasti kontaktu mezi dvěma součástmi tlakový kužel nezasahuje. V této oblasti pravděpodobně nebude zajištěn kontakt mezi součástmi 1 a 2 při kmitání součásti 1. Jak ukazuje Obr. 5.2, pokud bude druhý konec dílu 1 v horní úvrati, dojde k odlehnutí součástí, jak se bude druhý konec vracet, dojde ke vzniku kontaktu mezi dvěma součástmi a zvýšení tuhosti spoje. Spoj v této podobě tak napodobuje poruchu. Z tohoto stavu bude následně spoj bez poruchy vytvořen slepením těchto dvou dílů pomocí lepidla na tenzometry. Lepidlo zabrání odlehnutí součástí, a tak by nemělo dojít ani k nárazu.

Tímto způsobem by měla vzniknout výrazná nelinearita, která vytvoří dostatečnou odlišnost mezi daty měření s poruchou a bez poruchy. Slepení dvou součástí je navíc velmi snadné, součásti se pro něj nemusí rozebírat, ani s nimi není třeba jinak manipulovat, tudíž by nemělo dojít k vytvoření jiné odlišnosti mezi měřeními, než je odstranění poruchy.



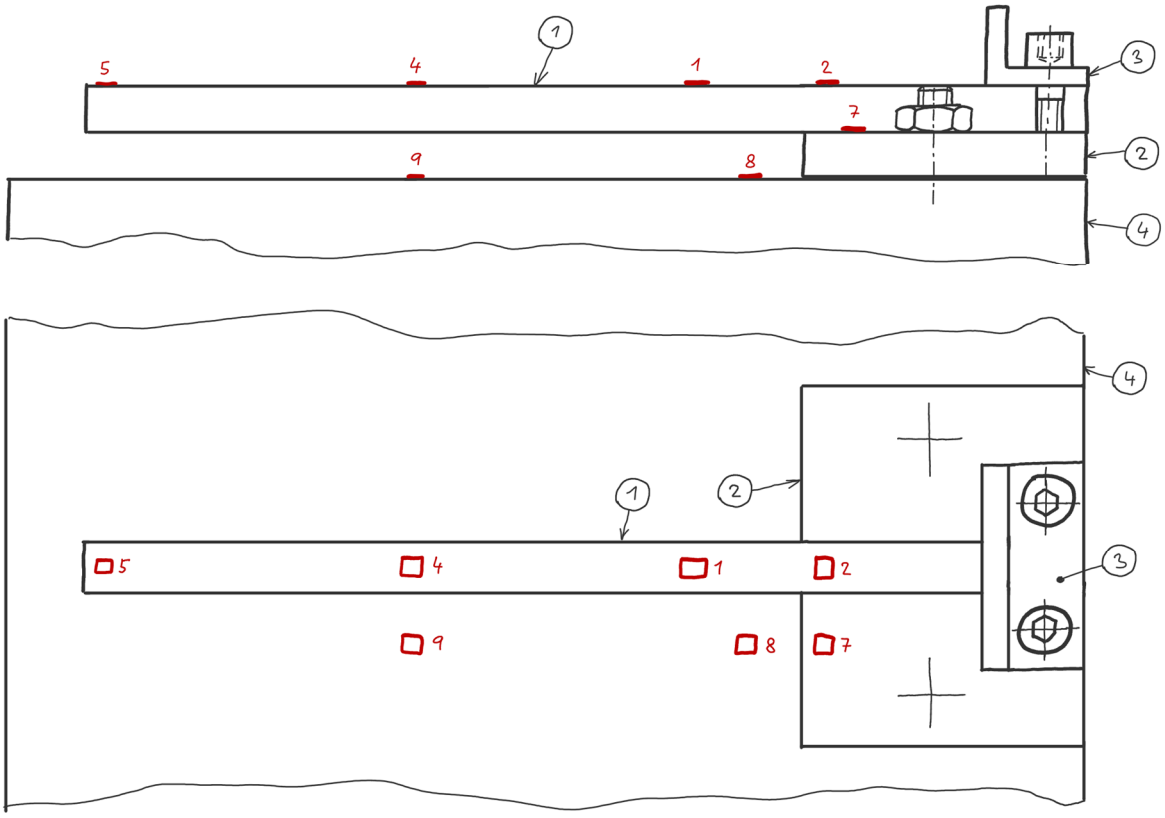
Obr. 5.1 Tlakový kužel pod šroubem spojujícím části 1 a 2



Obr. 5.2 „Odléhání“ dílu 1 od dílu 2 v oblasti mimo tlakový kužel

5.2 Průběh měření

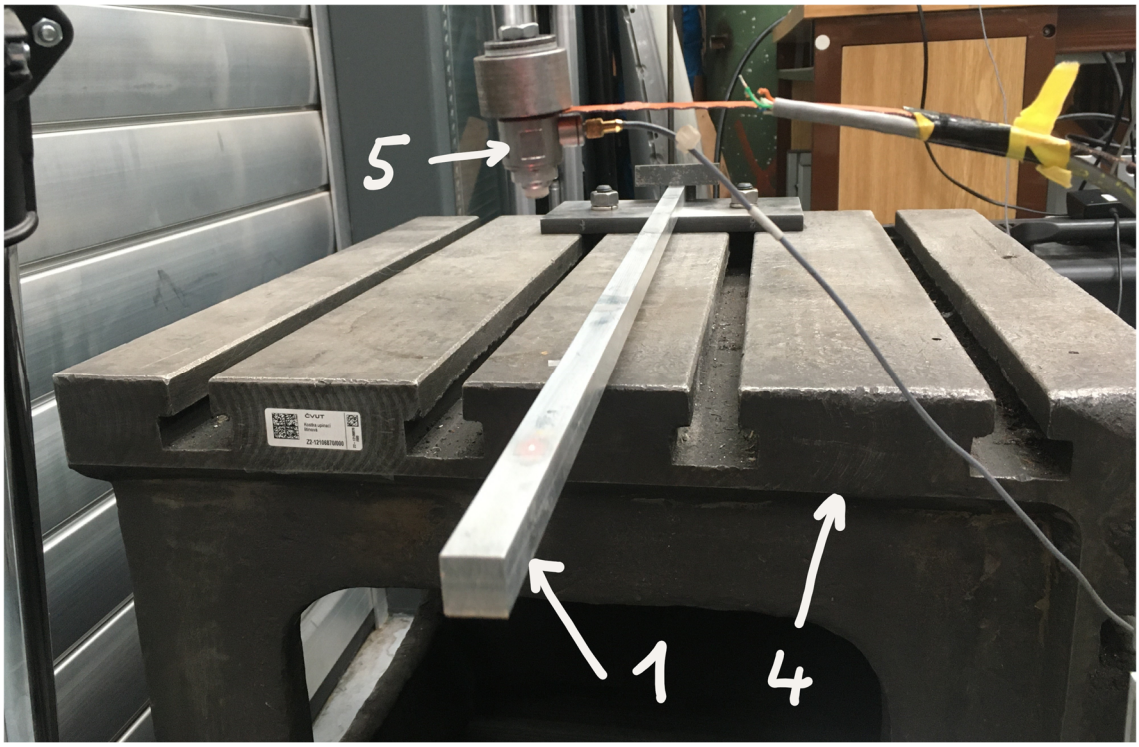
Měření bylo provedeno na jednoduchém nosníku tvořeném čtvercovou hliníkovou tyčí. Uspořádání je znázorněno na Obr. 5.3, nosník (1) je upnut šrouby mezi vrchní svorku (3) a spodní podkladový díl (2), podkladový díl je na vrchu do roviny obrobený. Seismická hmota (4) je přišroubována a přilepena k (2), má výrazně větší hmotu jak ostatní díly soustavy a je postavena na gumových plátech na podlaze. Mělo by tak dojít k odizolování od okolních vibrací a seismická hmota tak vytváří pevný rám, ke kterému je konstrukce ukotvena. Na červeně označených bodech byly připevněny odrazové plošky, výchylky těchto bodů byly měřeny laserovým vibrometrem. Kladívko provádělo buzení v poloze mezi měřenými body 4 a 5.



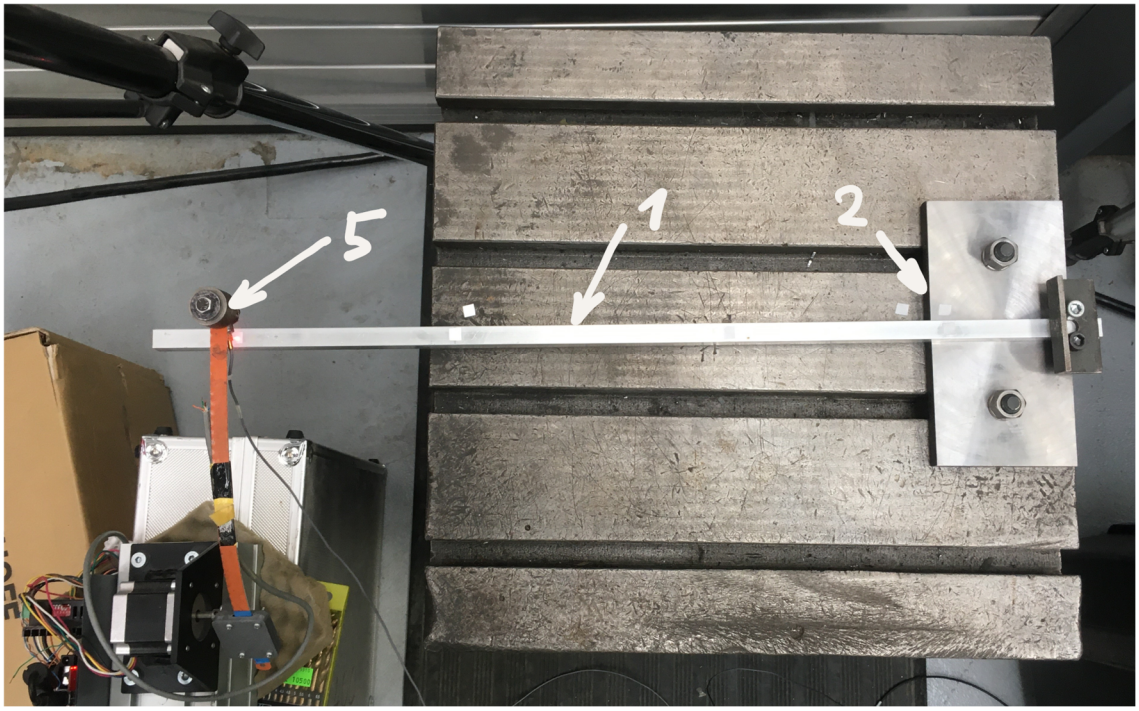
Obr. 5.3 Náčrtek uspořádání experimentu, červeně jsou označeny měřené body, v kruhu čísla dílů

Impaktní buzení bylo provedeno pomocí automatického kladívka vyvinutého v [26] - Obr. 5.4 (5) a Obr. 5.5 (5). Kladívko má plastový hrot a je schopné automaticky provádět opakované údery po dlouhou dobu, během úderů dochází k měření impaktní síly.

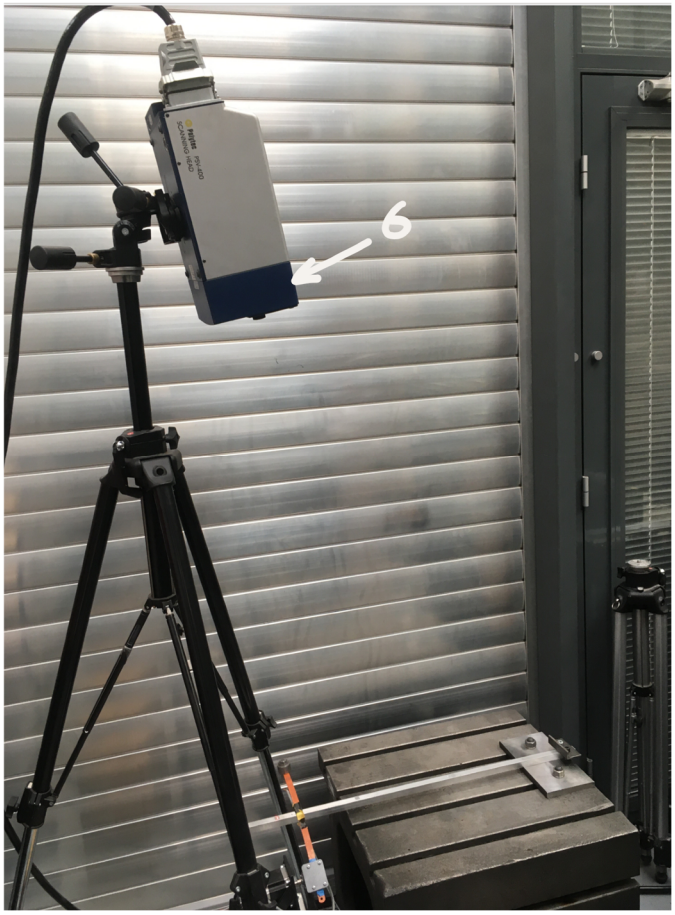
Laserový vibrometr - Obr. 5.6 (6) snímá vibrace měřených bodů 1, 2, 4, 5, 7, 8, 9 najednou. Měření také probíhá automaticky po dlouhou dobu.



Obr. 5.4 Fotografie uspořádání experimentu, 1 – nosník, 4 – seismická hmota, 5 – automatické kladívko



Obr. 5.5 Fotografie uspořádání experimentu, 1 – nosník, 2 – podkladový díl, 5 – automatické kladívko

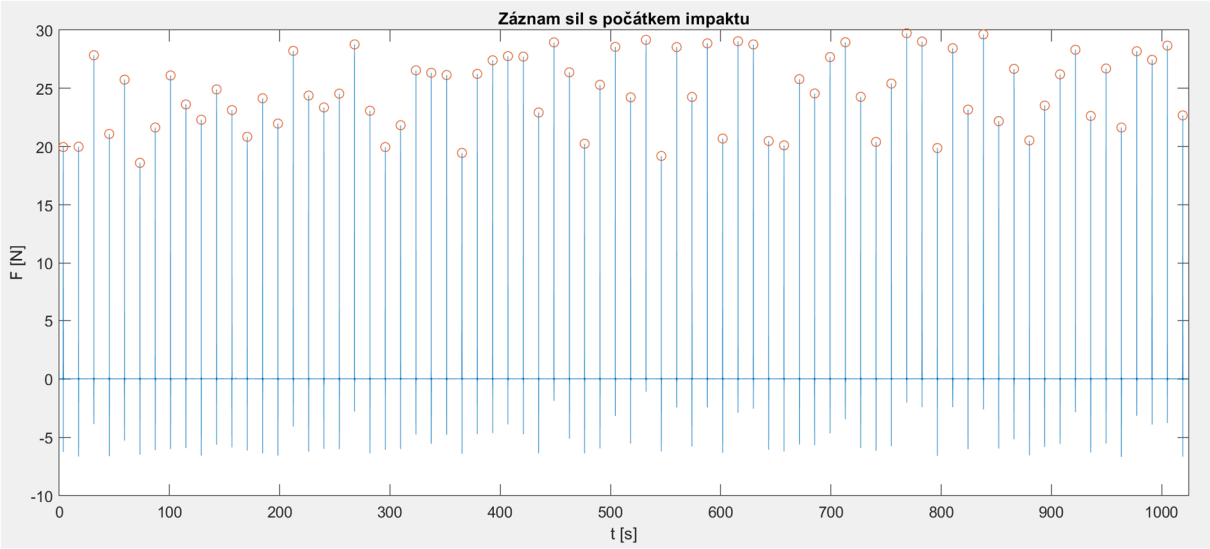


Obr. 5.6 Fotografie uspořádání experimentu, 6 – laserový vibrometr

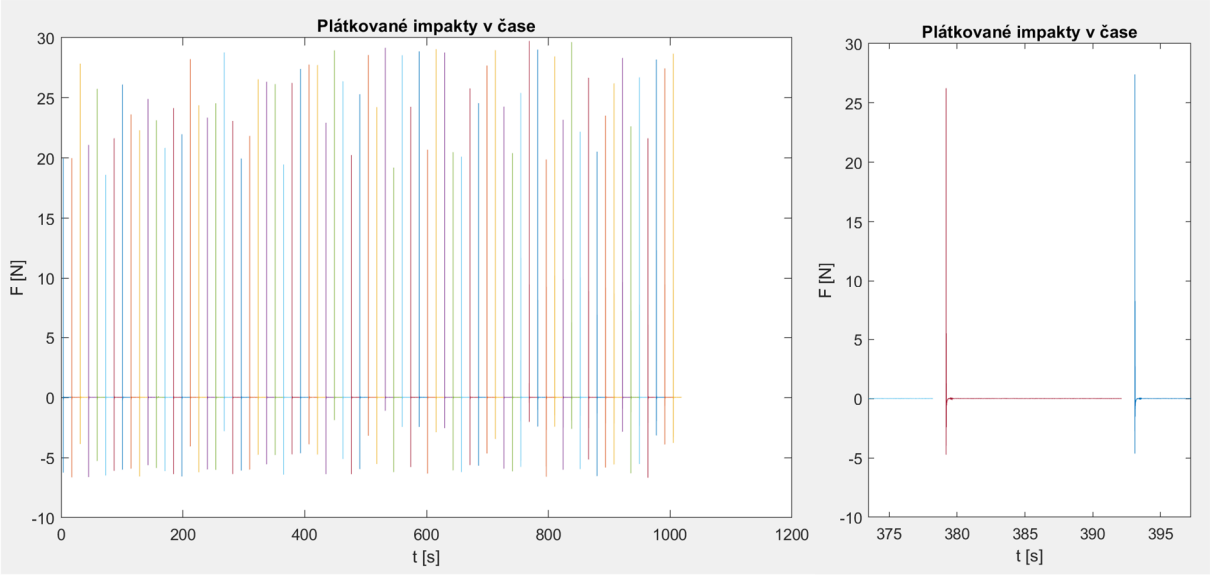
Nejprve byl tedy sestaven měřící řetězec (Obr. 5.3 až Obr. 5.6), bylo automaticky provedeno 73 úderů kladívkem po přibližně dvanácti vteřinách, během nich byly měřeny výchylky v měřených bodech. Následně byly díly (1) a (2) slepeny k sobě. Poté bylo provedeno 36 úderů kladívkem po přibližně dvanácti vteřinách. Hodnoty impaktních sil i rychlosti měřených uzlů byly zaznamenány s frekvencí 1024 Hz měřící aparaturou s anti-aliasingovým filtrem.

5.3 Vyhodnocení výsledků

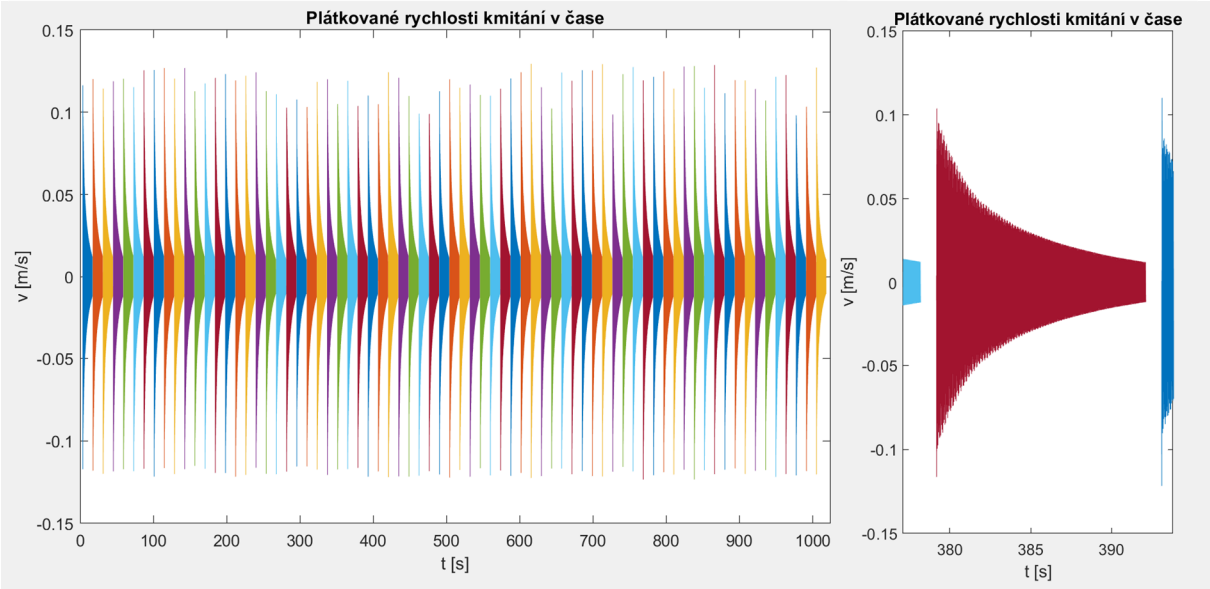
Vyhodnocení výsledků bylo provedeno prostřednictvím souboru v příloze 12.1 EMA_Zpracovani (Matlab). Data z jednotlivých impaktů byla zaznamenána souvislým měřením, pro zpracování je vhodné je rozdělit. Na datech impaktů proto nejprve byla nalezena maxima (ř.26) stanovením hranice 10 N od které probíhalo hledání bodu, jehož oba sousední body mají nižší hodnotu (vrcholky impaktů), tyto body zobrazuje Obr. 5.7. Od tohoto maxima byla stanovena vzdálenost proti směru času a po směru času o kterou budou data „odseknuta“ od ostatních, vzniká tak řada záznamů o stejné délce, jednotlivé „odseknuté“, či „naplátkované“ impakty jsou zobrazeny na Obr. 5.8. Tak, aby přesně časově odpovídali naplátkování impaktů, byly rozděleny i záznamy rychlostí kmitání bodů (Obr. 5.9).



Obr. 5.7 Síly impaktů v čase – oranžové body označují nalezená maxima

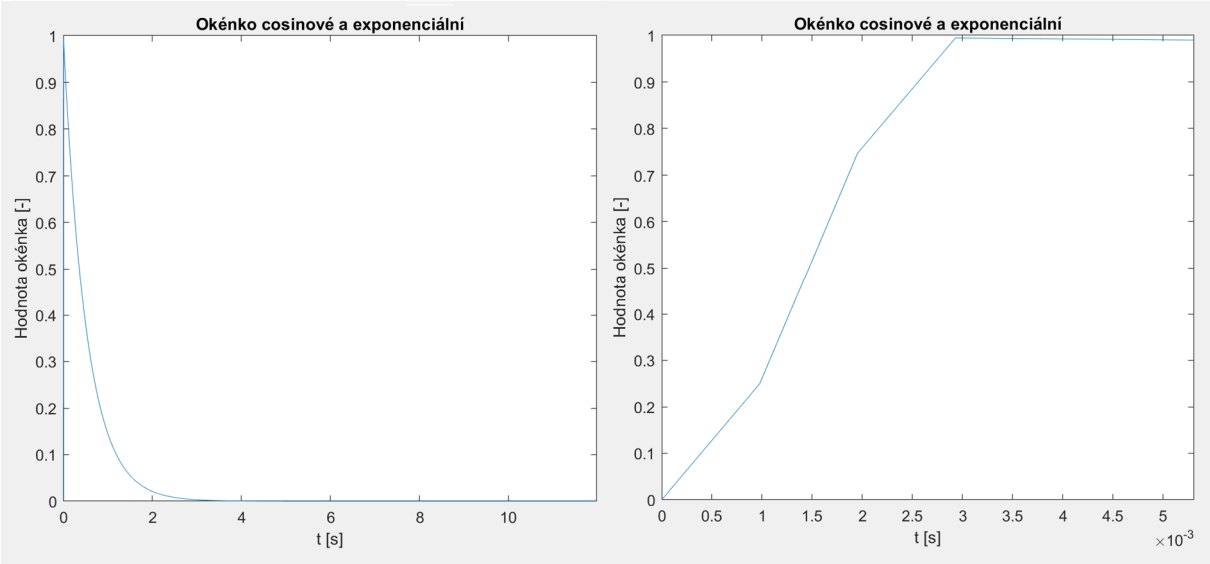


Obr. 5.8 Časový záznam sil rozdělený na jednotlivé impakty (vpravo detail)



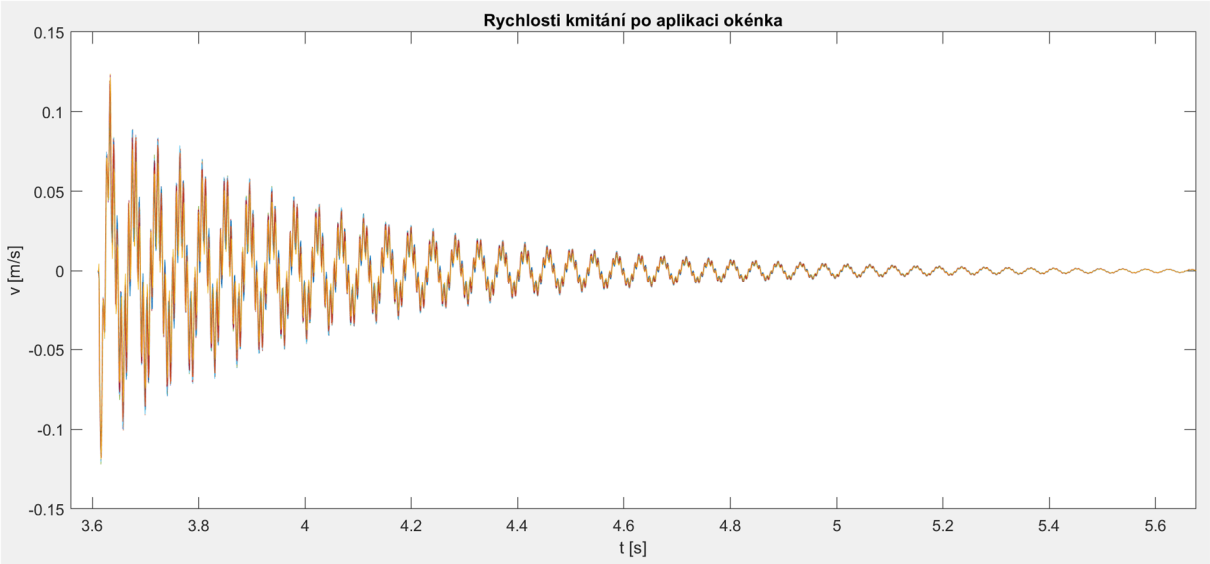
Obr. 5.9 Časový záznam rychlosti vibrace bodu 4 rozdělený na odezvy jednotlivých impaktů (vpravo detail)

Záznamy impaktů a kmitání jsou nyní vhodně rozdělené, ovšem jednotlivé záznamy nekončí ani nezačínají v nule. To může způsobit problémy při zpracování Fourierovou transformací, proto je třeba na záznamy aplikovat okénka (ř.79), která data na počátku i konci plynule přivedou k nule. V tomto případě bylo vytvořeno okénko exponenciální plynule jdoucí k nule a okénko s posunutou funkcí cosinus, jdoucí od nuly k jedničce a poté udržující konstantní hodnotu jedna. Tato dvě okénka jsou vynásobena a vytvářejí okénko (Obr. 5.10), které má maximum v místě maxima impaktu. Od počátku po maximum působí především cosinová část okénka, od maxima dále působí exponenciální okénko.

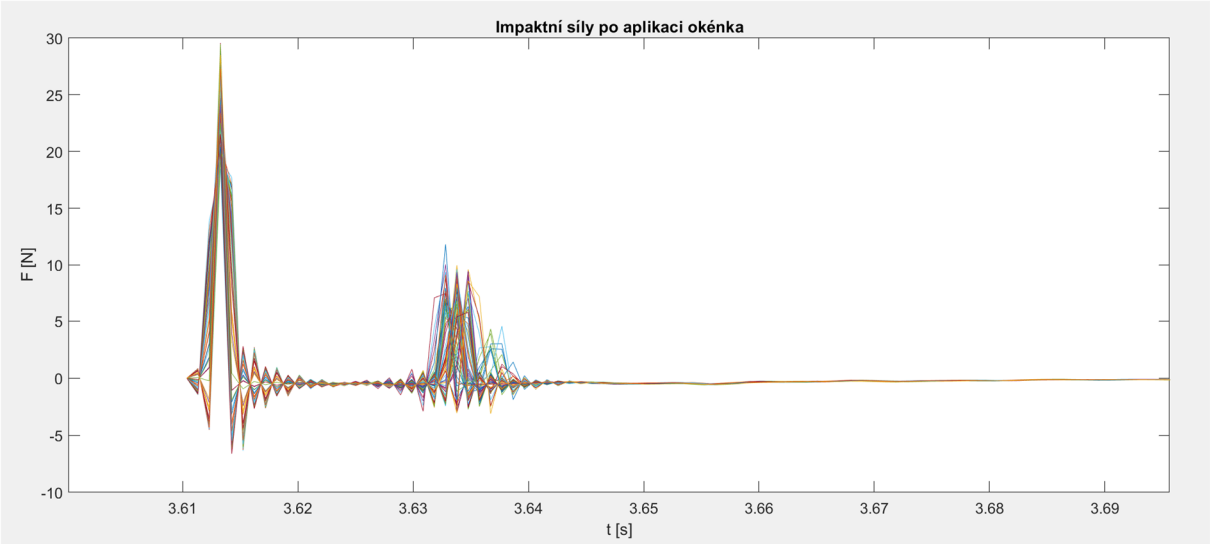


Obr. 5.10 Okénko pro přípravu signálů (vpravo detail na počátek okénka)

Na Obr. 5.11 jsou přes sebe promítnutá data rychlosti kmitání s aplikovaným okénkem, exponenciální okénko způsobuje rychlý útlum kmitání k nule. Obr. 5.12 ukazuje přes sebe promítnuté impakty, které byly vynásobené okénkem, je zřejmé, že došlo k takzvanému dvouklepu. Jde o situaci, kdy kladívko narazí do cíle dvakrát velmi krátce za sebou. Dvouklepy mohou způsobovat jisté nepřesnosti v měření, nicméně pro účely této práce by neměly být zdrojem velkých problémů.



Obr. 5.11 Rychlosti kmitání bodu 4 zpracované okénkem (detail na počátek záznamu)

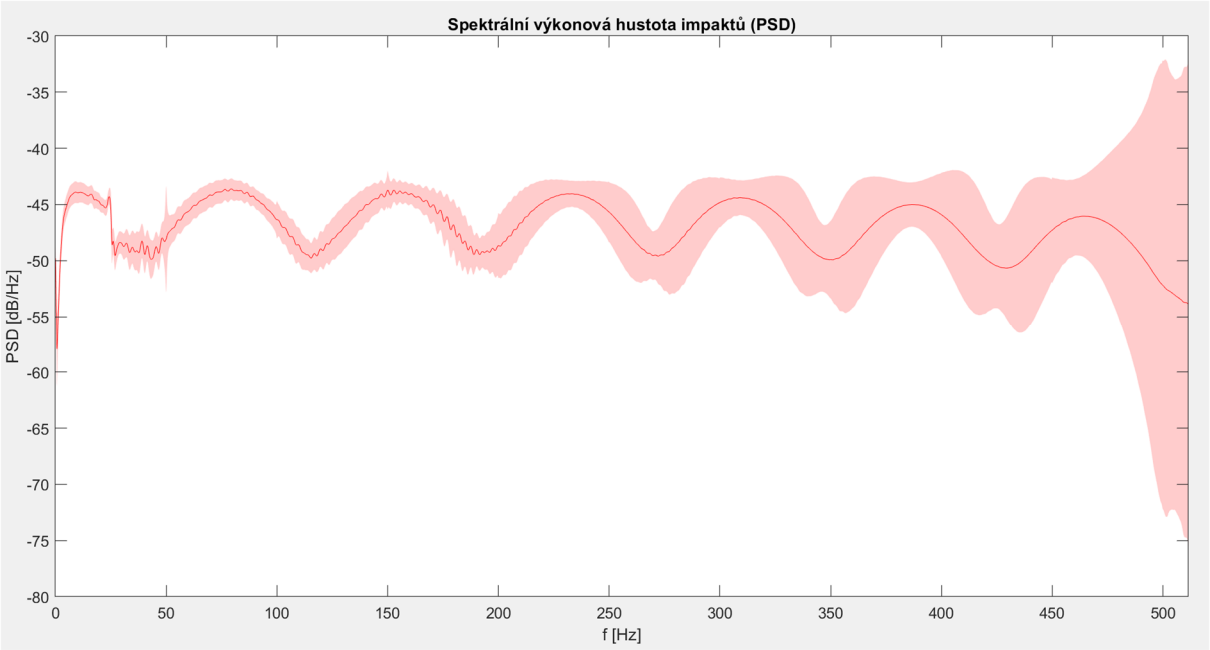


Obr. 5.12 Budící silové impulzy zpracované okénkem (detail počátku záznamu)

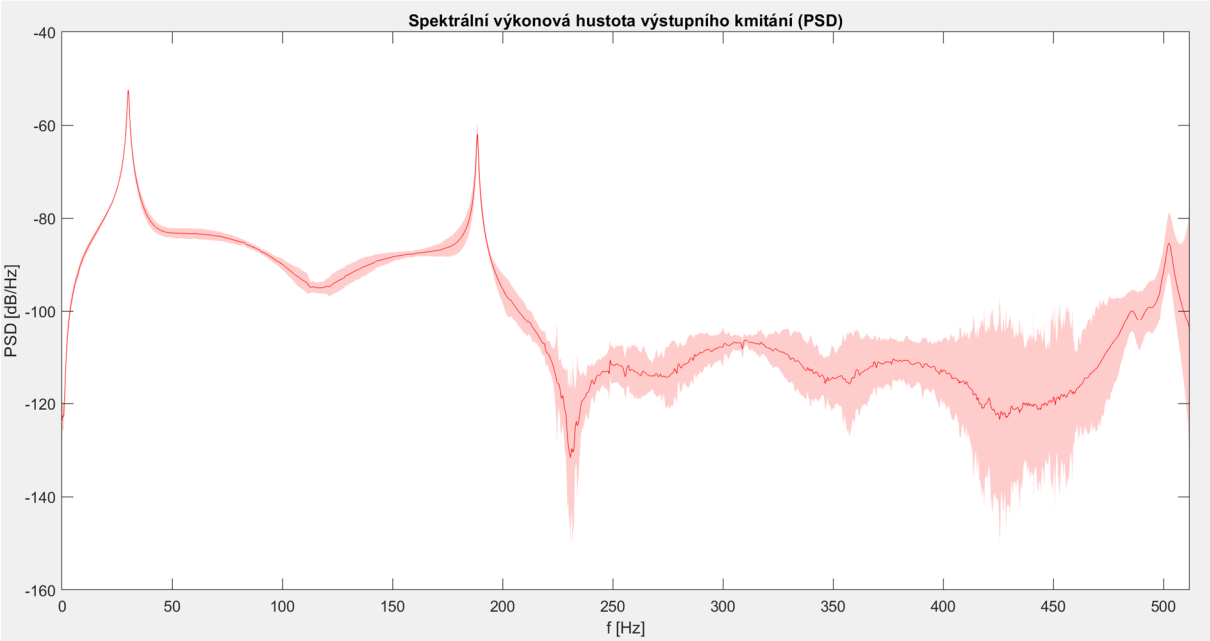
5.3.1 Výsledky pro měření bez nelinearity

Pro získání výsledků byla nejprve provedena rychlá Fourierova transformace pro jednotlivé plátky (ř.117), pro ně byla vypočtena spektrální výkonová hustota (Power spectral density – PSD). PSD impaktů je zobrazeno na Obr. 5.13, červená čára ukazuje průměrnou hodnotu napříč jednotlivými záznamy. Růžová plocha znázorňuje pásmo trojnásobku výběrové směrodatné odchylky. Při předpokladu normálního rozdělení dat v jednotlivých časových okamžicích představuje plocha oblast, kde se pohybuje přes 99 % průběhů dat ze všech možných. Je zřejmé, že impakty se od sebe navzájem poměrně výrazně liší, což je možné pozorovat již na Obr. 5.12.

Obr. 5.14 zobrazuje PSD rychlostí kmitání 4. bodu konstrukce bez napodobené poruchy. Z pásma směrodatné odchylky je zřejmé, že ve vyšších frekvencích se jednotlivé průběhy poměrně výrazně liší. Naproti tomu v místech peaků, které by měli odpovídat vlastním frekvencím, je pásmo velmi úzké, je tedy pravděpodobné, že průběh získaný zprůměrováním naměřených dat v těchto místech poměrně přesně odpovídá realitě.



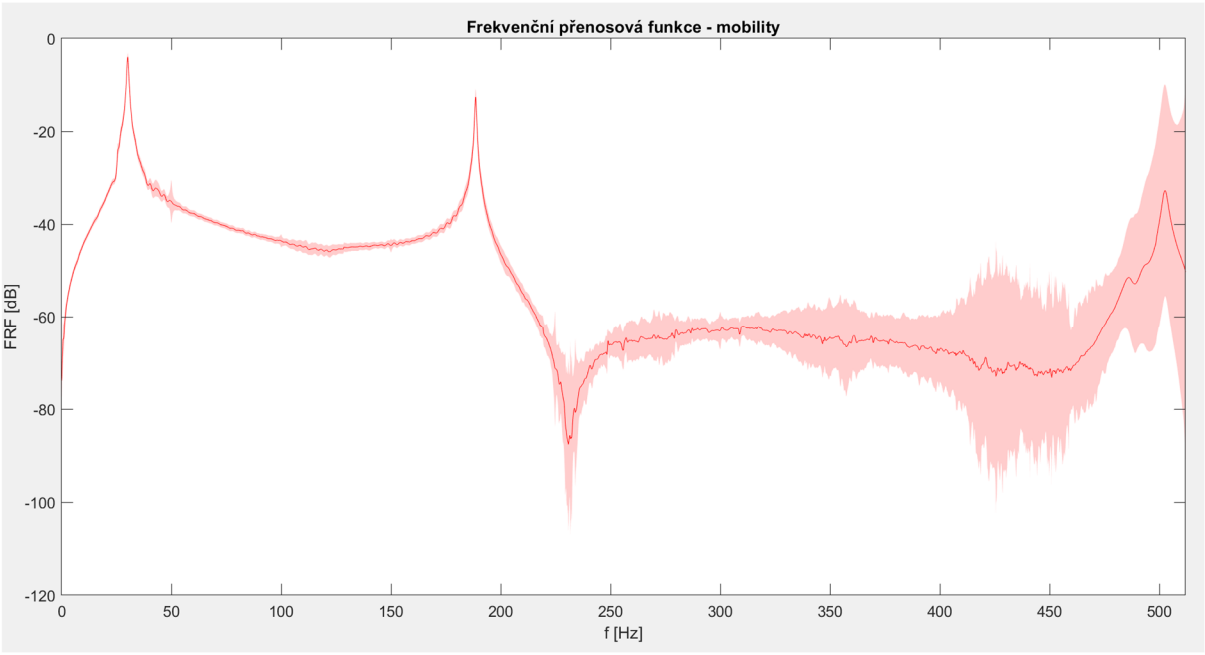
Obr. 5.13 PSD impaktů, konstrukce bez poruchy, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka



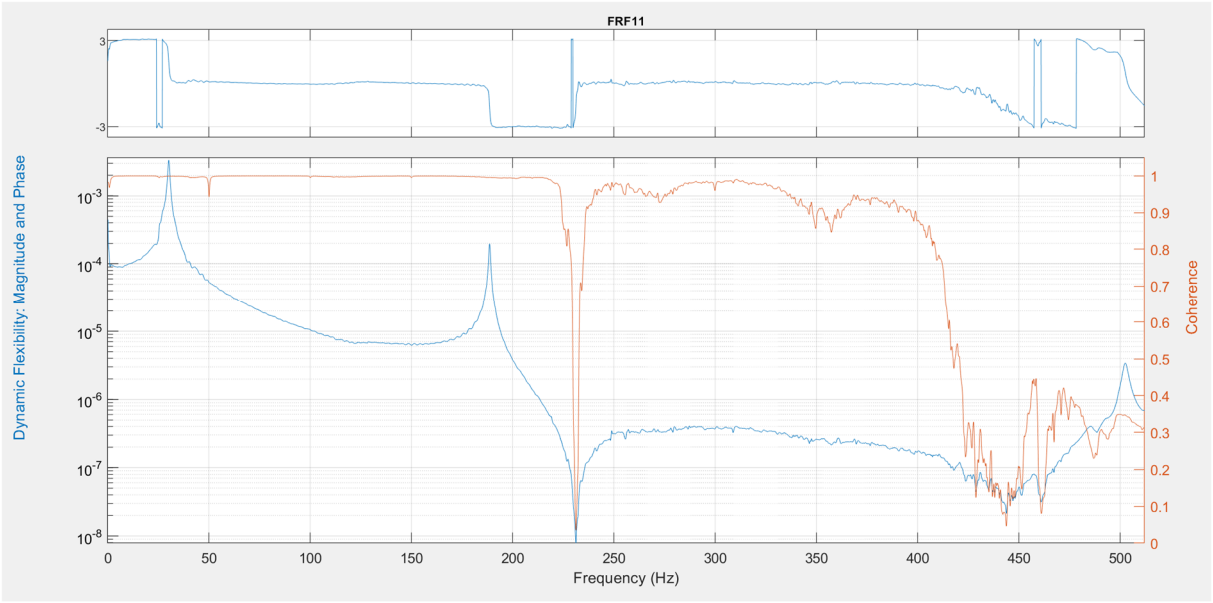
Obr. 5.14 PSD kmitání, konstrukce bez poruchy, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka

Ze spektrálních výkonových hustot byla vypočtena absolutní hodnota přenosové funkce pomocí *(2-40)*. FRF je zobrazena na Obr. 5.15, byla vypočtena pro rychlostní výstup nazývaný mobility. FRF vypočtená pro stejná data pomocí Matlab identification toolbox je zobrazena na Obr. 5.16, ovšem je vypočtena ve tvaru compliance, jelikož jiné verze toolbox neumožňuje. Průběhy na obou obrázcích se nicméně podobají. Z obrázků by zřejmě bylo možné usoudit, že měřený systém obsahuje vlastní frekvence hodnot okolo 30 Hz, 185 Hz a 500 Hz. Určení poslední uvedené frekvence není jisté, jelikož koherence je v těchto frekvencích pro toto měření již velmi nízká, také pásmo směrodatné odchylky je zde poměrně široké. Nula systému je zřejmě okolo 230 Hz. Ostatní výchylky od hladkého průběhu by se nejspíš dali označit za šum, případně za projevy nedokonalého buzení, jehož PSD (Obr. 5.13) je v oblasti 40-50 Hz silně zvlněn podobně jako průběh FRF. Že jde o projevy šumu napovídá také fakt, že je v okolí těchto zvlnění a malých peaků

velmi široké pásmo směrodatné odchylky, je tedy možné, že reálně zde žádné zvlnění a peaky nejsou.



Obr. 5.15 FRF mobility, konstrukce bez poruchy, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka

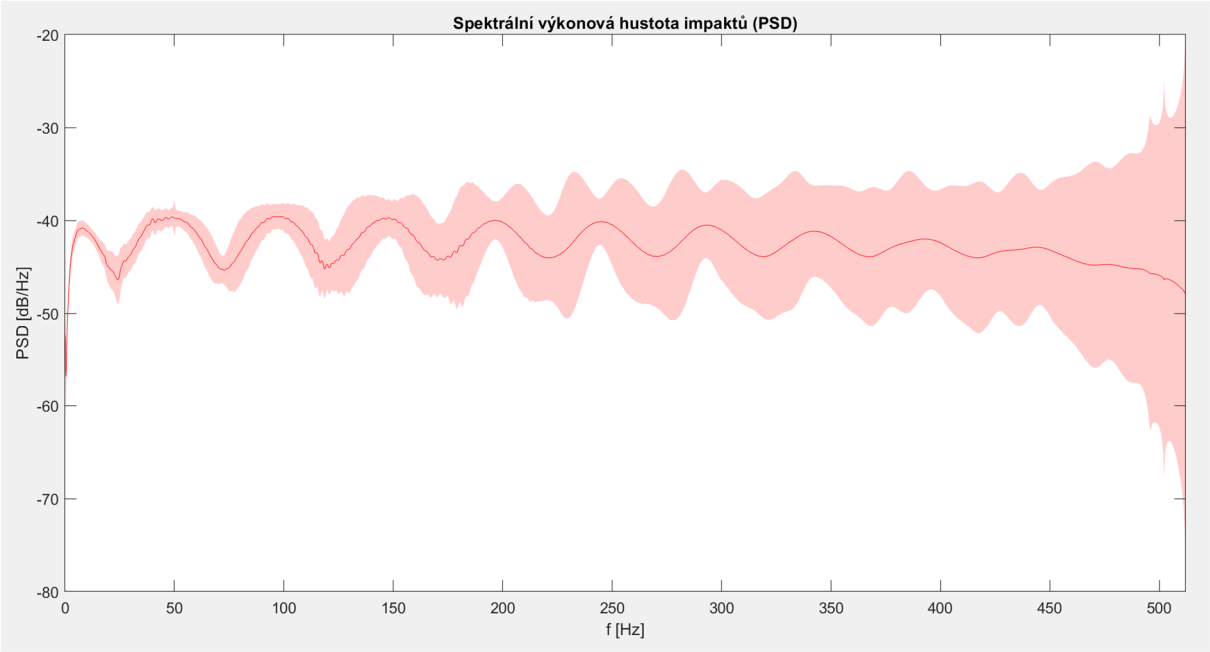


Obr. 5.16 FRF compliance konstrukce bez poruchy z Matlab identification toolbox

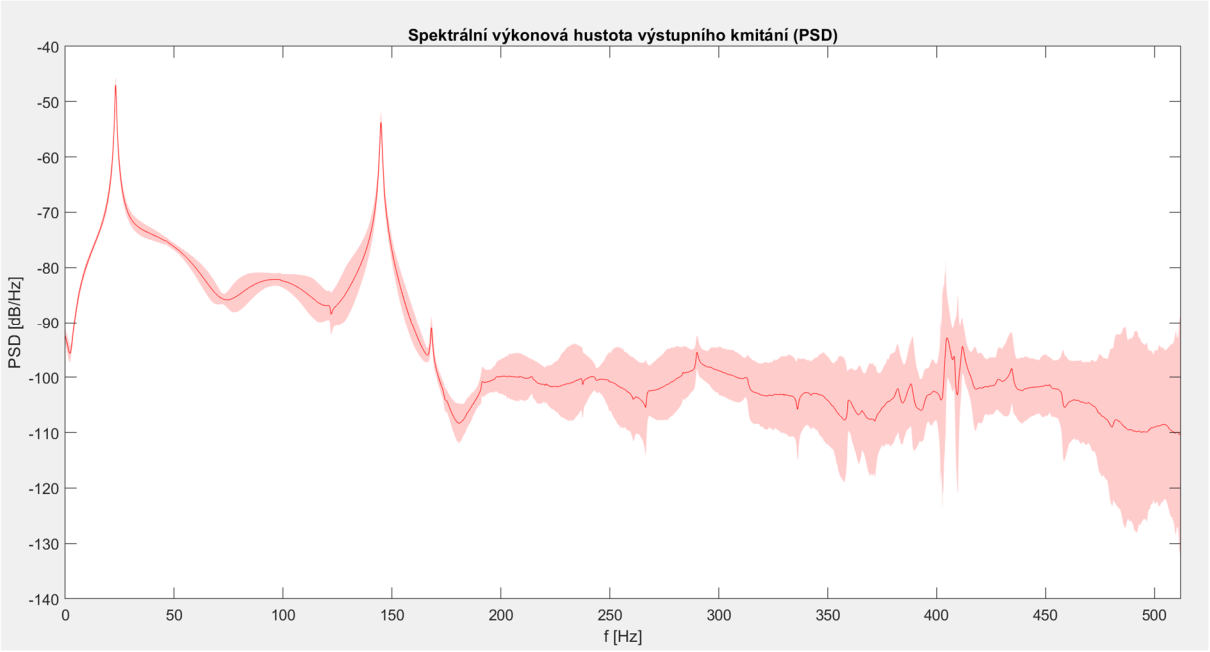
5.3.2 Výsledky pro měření s nelinearitou

Výsledky pro měření soustavy, kde byla napodobena porucha, byly zpracovány stejným způsobem, jako ty bez napodobené poruchy. Obr. 5.17 zobrazuje PSD impaktů, pásmo směrodatné odchylky je opět velmi široké.

Obr. 5.18 zobrazuje PSD kmitání 4. bodu konstrukce s napodobenou poruchou, PSD obsahuje tři peaky, které mají velmi úzká pásma směrodatné odchylky a je velmi pravděpodobné, že odpovídají skutečnému průběhu. Jde o peaky v místech okolo 30 Hz, 145 Hz a 165 Hz.



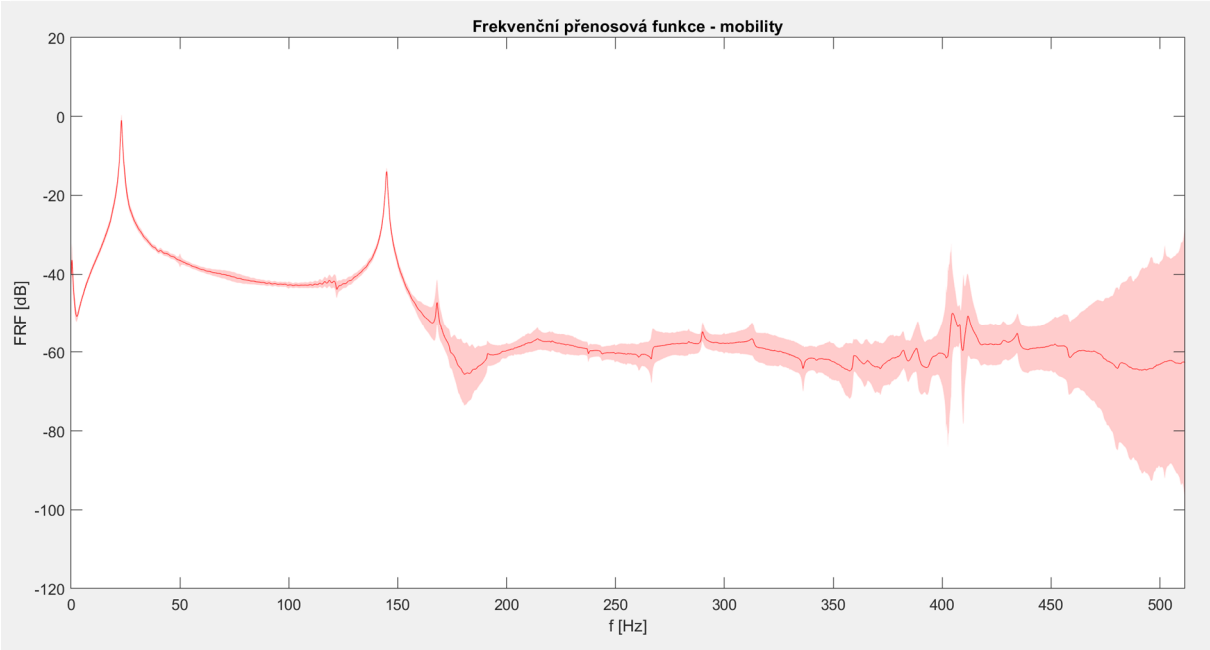
Obr. 5.17 PSD impaktů, konstrukce s poruchou, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka



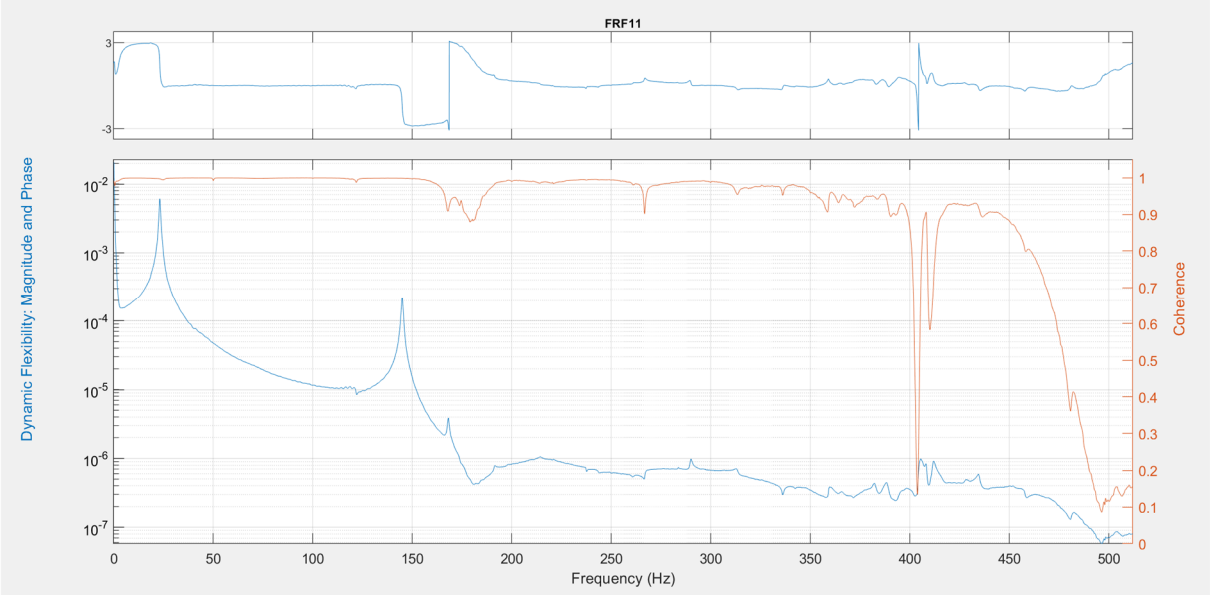
Obr. 5.18 PSD kmitání, konstrukce s poruchou, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka

Obr. 5.19 zobrazuje FRF získanou z PSD konstrukce s napodobenou poruchou pro výstup ve tvaru rychlostí, Obr. 5.20 zobrazuje FRF ve tvaru compliance získanou pomocí Matlab identification toolbox. Na obou obrázcích jsou zřejmé tři peaky pozorované již na Obr. 5.18. První dva jsou velmi výrazné, třetí je poměrně malý. FRF obsahuje i další peaky či zvlnění ve vyšších frekvencích, při některých z nich je poměrně vysoká hodnota koherence, a i pásmo směrodatné odchylky je v této oblasti poměrně malé. Není tedy možné jednoznačně prohlásit, že jde o projevy šumu. Ani PSD impaktů (Obr. 5.17) v těchto místech není takovýmto způsobem zvlněná.

Lze se domnívat, že první dvě frekvence systému se simulovanou poruchou odpovídají prvním dvěma frekvencím systému bez simulované poruchy s tím, že jsou posunuty vlivem simulované poruchy. Třetí peak na Obr. 5.18, Obr. 5.19 a Obr. 5.20 a další zvlnění bude třeba dále vysvětlit.



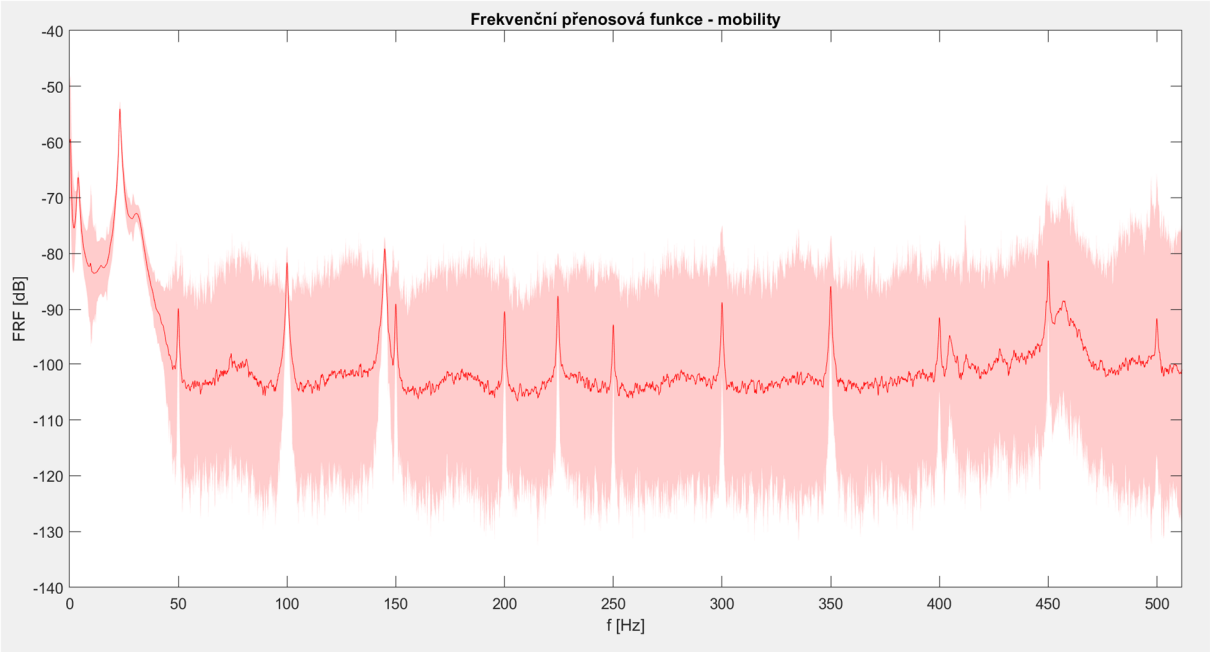
Obr. 5.19 FRF mobility, konstrukce s poruchou, červeně – průměrná hodnota, růžově – 3 x výběrová směrodatná odchylka



Obr. 5.20 FRF compliance konstrukce s poruchou z Matlab identification toolbox

5.3.3 Kontrola uspořádání experimentu

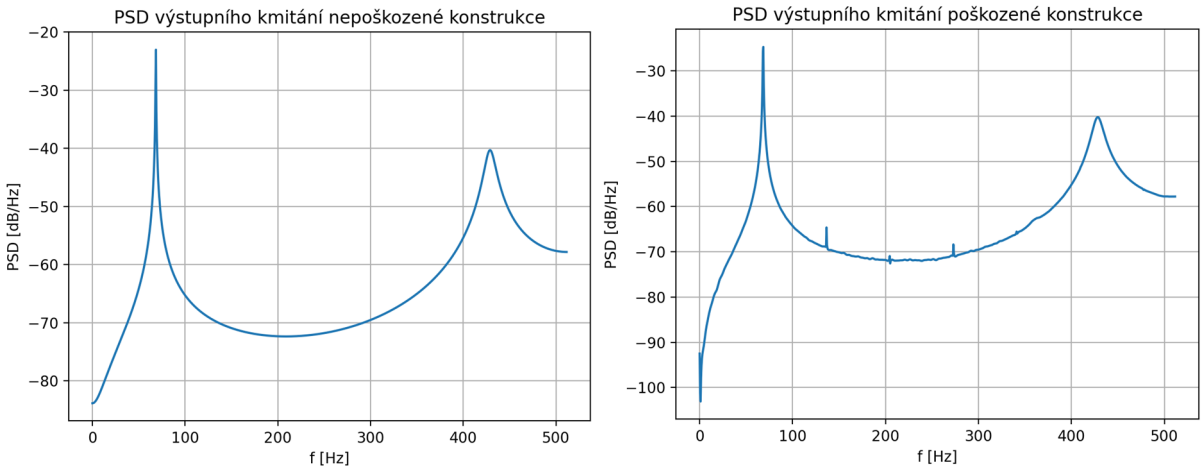
Při sestavení modelu bylo předpokládáno, že seismická hmota označená číslem 4 (Obr. 5.4) je dostatečně hmotná a odizolovaná od okolí, aby působila jako pevný rám. Tento předpoklad byl ověřován měřením kmitání kostky v měřícím bodě 8 (Obr. 5.3). Při zpracování kmitání tohoto bodu stejným způsobem, jako předcházející měření, vychází FRF zobrazená na Obr. 5.21. Nejvýraznější peak zřejmě odpovídá první vlastní frekvenci měřené konstrukce, nicméně celý průběh FRF pro tento bod je velmi výrazně nižší, než pro bod 4. Lze se tedy domnívat, že pohyby seismické hmoty výsledky měření ovlivní, ovšem ne výrazně.



Obr. 5.21 FRF compliance bodu 8

5.4 Kontrola simulované poruchy

Důvodem provedení měření na reálné konstrukci bylo porovnání těchto měření se simulací. Nejde přitom o porovnání konkrétních hodnot, ale spíše o tvar průběhů funkcí. Vzhledem k tomu, že cílem je pozorování nelineárního chování, je možné, že výpočet FRF, který slouží k analýze lineárních systémů, nepovede k úspěšné analýze nelineárního chování a spíše nelineární chování zastře. Proto budou projevy simulované poruchy analyzovány z PSD výstupu, tedy rychlosti kmitání bodu soustavy.

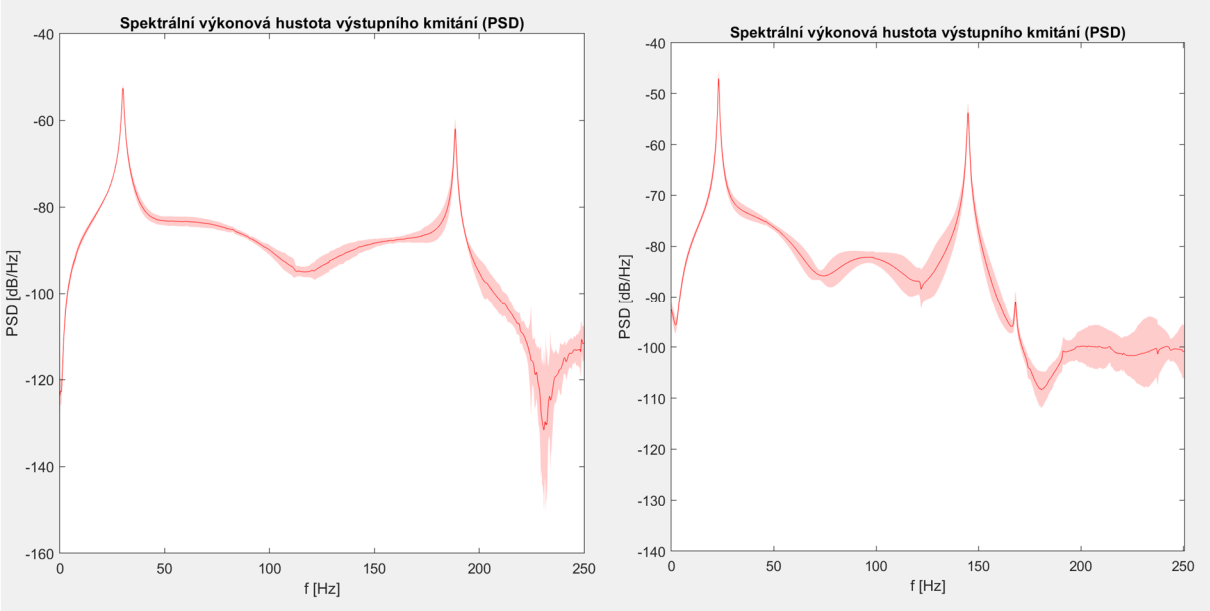


Obr. 5.22 Porovnání PSD simulovaných kmitání (vlevo) bez poruchy, (vpravo) s poruchou

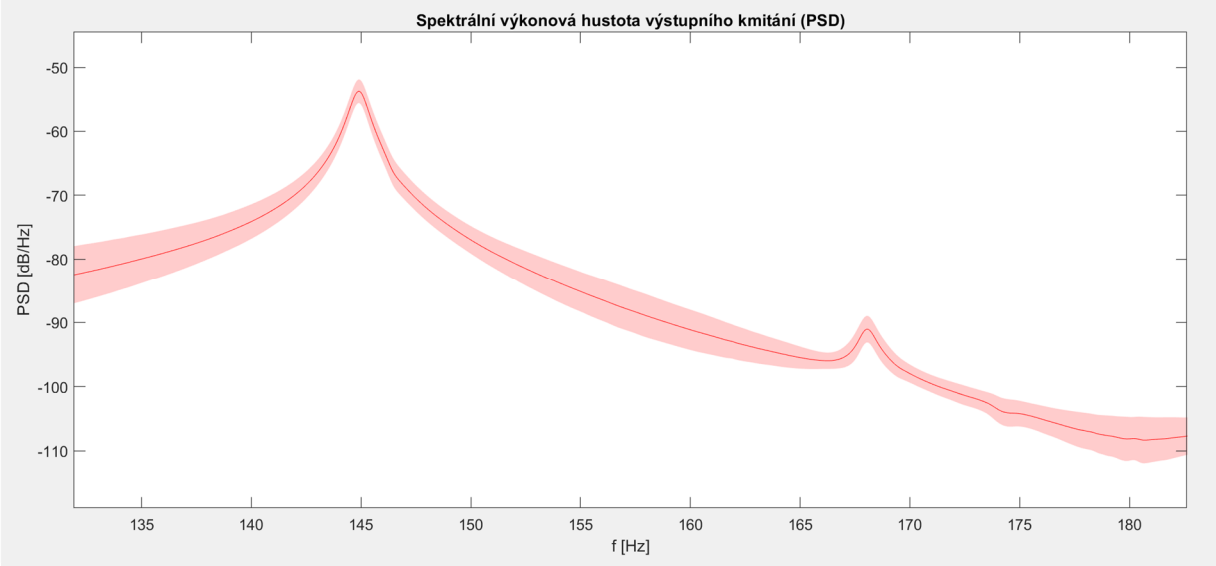
Na Obr. 5.22 je vidět PSD simulovaných výstupů pro modely bez poruchy a s poruchou. Dva peaky odpovídající vlastním frekvencím, které jsou vidět na PSD modelu bez poruchy, jsou zřetelné i na PSD modelu s poruchou. Celkově lze říci, že obě PSD jsou velmi podobné, hlavním rozdílem jsou malé peaky na PSD modelu s poruchou. Ty nejsou projevem šumu, jelikož ten nebyl do simulace zahrnut. Simulace probíhala stejným způsobem, jako pro model bez poruchy, není tedy

pravděpodobné, že jde o projev integrační metody. Vzniká tedy podezření, že jde o projevy nelinearity v systému

Obr. 5.23 ukazuje PSD naměřených výstupů kmitání bodu 4, které byly získány v předchozí kapitole. Dva nejvýraznější peaky nejspíš odpovídající vlastním frekvencím jsou opět společnou vlastností průběhů PSD pro systémy s napodobenou poruchou a bez ní. Stejně jako v případě výsledků simulací, jsou i zde pozorovatelné malé peaky v případě výsledků s napodobenou poruchou. Nejvýraznější z těchto peaků na frekvenci přibližně 167 Hz, popsán již v předchozí kapitole, má velmi malé pásmo směrodatné odchylky. Detail na něj je na Obr. 5.24 vpravo. Je tedy možné prohlásit, že nejspíš nejde o projev šumu, jde o vlastnost opakující se napříč všemi měřenými vzorky. Při porovnání se simulovanými výsledky vzniká podezření, že jde o obdobu drobných peaků pozorovaných na simulaci se simulovanou poruchou.



Obr. 5.23 Porovnání PSD měřených kmitání (vlevo) bez poruchy, (vpravo) s poruchou



Obr. 5.24 Detail (Obr. 5.18) na peak vlastní frekvence (vlevo) a „falešnou vlastní frekvenci“ (vpravo)

Pozorované podobnosti vedou k domněnce, že porucha typu „odléhání a nárazy“ se projevuje vznikem „falešných vlastních frekvencí“ ve frekvenční oblasti. Tuto domněnku podporuje

pozorování autorů [10], že u FRF rozbitých vlakových kol se vytvoří větší množství peaků, než u FRF nerozbitých kol.

6 Zpracování identifikací filtru

Cílem této kapitoly je vytvořit metodu pro binární klasifikaci vstupních dat do dvou tříd. První třídou jsou měření na systému bez poruchy a druhou třídou jsou měření na systému s poruchou. Prostředkem vedoucím k tomuto cíli může být vytvoření diskrétního filtru napodobujícího chování měřené mechanické soustavy. Vstupem do filtru i do měřené soustavy je silový impuls, výstupem soustavy je měřená rychlost kmitání jednoho bodu. Pokud pro filtr provádíme optimalizaci pro identifikaci, znamená to, že je stanoven způsob měření odchylky výstupu filtru od výstupu reálného systému, optimalizací parametrů filtru se potom pokoušíme tuto odchylku minimalizovat. Potom lze tvrdit, že filtr do určité míry napodobuje chování systému a nalezené parametry filtru mohou nést určitou informaci o tomto systému.

Lineární filtr je schopen napodobit pouze lineární chování soustavy, proto by bylo nejspíš obtížné z jeho parametrů identifikovat nelineární poruchu. Může se zdát vhodné použít lineární filtr na nelineární data a poruchu identifikovat z velikosti odchylky – pokud je porucha nelineární a filtr ji nenapodobí, odchylka výstupu filtru a reálných dat zřejmě bude velká. Nicméně takový přístup by nejspíš byl velmi citlivý na šum, který by také vyhodnotil jako poruchu. Navíc bylo v této práci odzkoušeno, že na experimentálně získaných datech standardní nástroje Matlab identification toolbox mohou mít menší odchylku při lineární identifikaci systému s poruchou než u systému bez poruchy.

Dalším možným přístupem tak je vytvoření nelineárního filtru, který bude opět optimalizován pro identifikaci. Pokud bude některý z parametrů filtru vyjadřovat „míru nelinearity“, potom by mělo být možné na základě tohoto parametru rozhodnout, zda měřený systém spadá do třídy s poruchou, či bez poruchy. Vzhledem k tomu, že pro účely simulace v této práci již byl vytvořen jeden z modelů poruchy, zdá se příhodné právě tento model zapracovat i do filtru, aby představoval uvedenou nelinearitu.

6.1 Lineární OE model podle flexibilního mech. systému

Z předpisu OE modelu *(2-44)* je možné určit přenos diskrétního SISO systému OE modelu *(6-1)*, kde q^{-i} je operátor posunutí zpět v čase o i časových kroků. Identifikace OE modelu představuje hledání parametrů a_i a b_i. Jak již bylo uvedeno v rešerši, hledání parametrů lineárního OE modelu může být provedeno pomocí metody nejmenších čtverců.

$$\hat{y}(k) = G(q)\,u(k)\,, \qquad G(q) = \frac{b_0 + b_1 q^{-1} + \ldots + b_m q^{-m}}{1 + a_1 q^{-1} + \ldots + a_m q^{-m}} \tag{6-1}$$

V této práci však bude do modelu přidána nelinearita, proto metodu nejmenších čtverců nebude možné použít. Přidaná nelinearita ve filtru bude mít stejné chování, jako simulovaná, bude tedy provádět změnu tuhosti systému na základě hodnoty některé souřadnice. Z předpisu *(6-1)* ovšem v této podobě není možné přímo určit, které parametry by se měli měnit, ani na základě které hodnoty. Zjednodušeně řečeno, není znám fyzikální význam těchto parametrů, ovšem model nelinearity, který si přejeme implementovat, je založen na fyzikální představě.

Fyzikální představu zachovává stavový popis *(2-18)* mechanického systému. Může mít mnoho podob v závislosti na volbě stavů, obecně jsou tak matice stavového popisu plné. To znamená velké množství parametrů, které by bylo třeba identifikovat, například v porovnání s *(6-1).* To může představovat problém při nelineární optimalizaci, která se s rostoucím množstvím

parametrů zpomalí. Hlavním problémem ovšem je, že velké množství parametrů umožní filtru napodobovat chování nesouvisející s fyzikální představou. Filtr se tak může naučit napodobovat chování, které nemá napodobovat. Je tedy vhodné učinit některé předpoklady, které množství parametrů sníží.

Převodem flexibilní mechanické soustavy do modálních souřadnic lze získat frekvenční přenosovou matici *(2-37)*. Hledání parametrů tohoto přenosu je v podstatě úkolem experimentální modální analýzy, matice $D_{(\omega)}$ z *(2-37)* je diagonální a lze říci, že takový popis má tudíž méně parametrů než stavový popis pro obecně zvolené stavy.

Ovšem množství parametrů stavového popisu lze také snížit, mimo jiné použitím stavů ve tvaru modálních souřadnic a jejich derivací. Takový stavový popis byl již vytvořen v kapitole 4.3.1, jeho matice jsou uvedeny v *(4-7)* až *(4-12)*. Tyto rovnice ukazují, že matice $\hat{A}$ je tvořena čtyřmi submaticemi Z (nulová), E (jednotková), Λ (diagonální), B_p (diagonální). Matice $\hat{B}$ je tvořena dvěma submaticemi Z a V^T. V případě jediné vstupní síly f a jednoho měřeného bodu, jde o SISO systém a matice $\hat{B}$ má jediný sloupec.

Cílem je získání diskrétního modelu, proto nyní proveďme převod tohoto stavového popisu na diskrétní dle *(2-42)*. Pro matici $\widehat{M}$ při vyjádření exponenciální funkce ve tvaru nekonečného součtu řady *(6-2)* vyplývá, že mimodiagonální prvky $\widehat{M}$ jsou nulové, pokud jsou nulové i odpovídající prvky $\hat{A}$. Dle *(2-42)* matice $\hat{O}$ a $\hat{P}$ odpovídají maticím spojitého stavového systému. Matice $\widehat{N}$ bude mít tvar odpovídající násobku matic $\widehat{M}$ a $\hat{B}$, půjde tedy o vektor obecně bez nulových prvků.

$$\underline{\underline{\widehat{M}}} = e^{\underline{\underline{\hat{A}}}\,\Delta t} = \sum_{i=0}^{\infty} (i!)^{-1} \left(\underline{\underline{\hat{A}}}\,\Delta t\right)^i = \underline{\underline{E}} + \underline{\underline{\hat{A}}}\,\Delta t + (2!)^{-1} \left(\underline{\underline{\hat{A}}}\,\Delta t\right)^2 + \ldots \qquad (6\text{-}2)$$

Z výše uvedeného vyplývá, že pro flexibilní mechanický systém se třemi stupni volnosti, budou mít matice diskrétního stavového popisu tvar dle *(6-3)*. Vlastnosti takového systému lze tedy popsat pomocí vektoru parametrů $\underline{p}$, který bude mít $7 \cdot n$ prvků, kde n je počet stupňů volnosti.

$$\underline{\underline{\widehat{M}}} = \begin{bmatrix} p_1 & 0 & 0 & p_4 & 0 & 0 \\ 0 & p_2 & 0 & 0 & p_5 & 0 \\ 0 & 0 & p_3 & 0 & 0 & p_6 \\ p_7 & 0 & 0 & p_{10} & 0 & 0 \\ 0 & p_8 & 0 & 0 & p_{11} & 0 \\ 0 & 0 & p_9 & 0 & 0 & p_{12} \end{bmatrix}, \underline{\underline{\widehat{N}}} = \begin{bmatrix} p_{13} \\ p_{14} \\ p_{15} \\ p_{16} \\ p_{17} \\ p_{18} \end{bmatrix}, \underline{\underline{\hat{O}}} = \begin{bmatrix} p_{19} \\ p_{20} \\ p_{21} \\ 0 \\ 0 \\ 0 \end{bmatrix}^T, \underline{\underline{\hat{P}}} = [0] \qquad (6\text{-}3)$$

Tímto bylo dokázáno, že lze získat stavový popis popisující chování jakékoliv lineární flexibilní mechanické soustavy, aniž by byly využity všechny prvky matic tohoto popisu. A dále, že při vhodném převodu tohoto popisu na diskrétní popis stále pro libovolný uvažovaný systém zůstávají některé členy matic tohoto popisu nulové. Tím dochází ke snížení množství parametrů identifikovaného systému. Nicméně přesto některé z parametrů mají nenulovou hodnotu pouze vlivem diskretizace a některé jsou na sobě ve fyzikálním systému závislé, proto uvedený počet parametrů je větší než minimální množství k úplnému popisu chování systému.

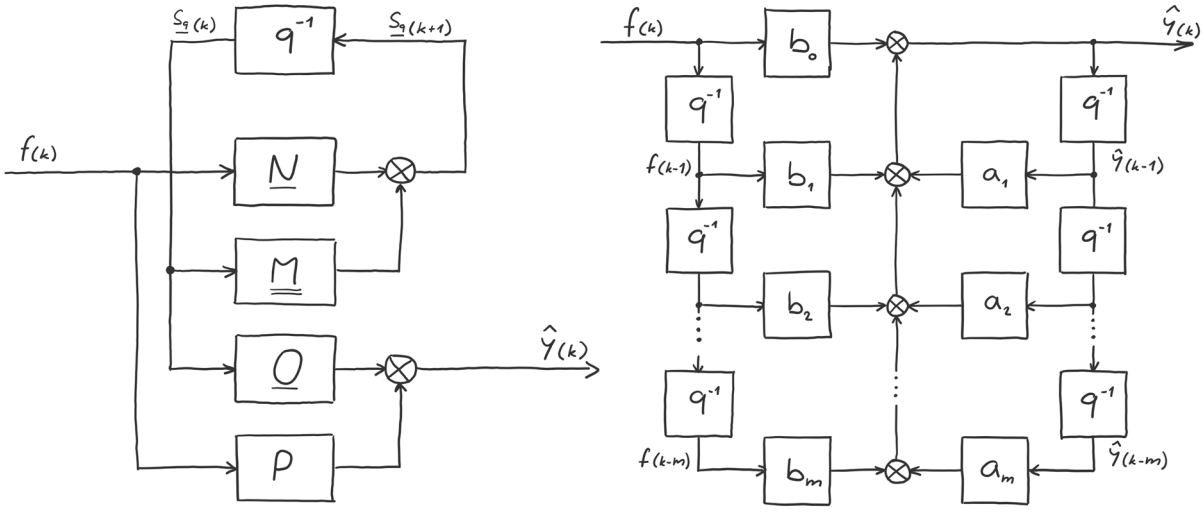
Při využití operátoru q^{-1}, který provádí diskrétní posunutí o jeden krok zpět v čase, lze výsledný stavový popis zapsat jako *(6-4)*. Z tohoto stavového popisu je potom možné získat diskrétní přenos *(6-5)*. Tento tvar připomíná vyjádření zmíněného frekvenčního přenosu *(2-37)* a bylo by

možné z něj vytvořit přenos OE modelu ve tvaru *(6-1)*. Je tak možné prohlásit, že z diskrétního stavového popisu lze vytvořit odpovídající OE model.

$$\begin{aligned} \underline{s}(k) &= \underline{\underline{\hat{M}}}\, q^{-1}\underline{s}(k) + \underline{\hat{N}}\, q^{-1} f(k) \\ \hat{y}(k) &= \underline{\hat{O}}\, \underline{s}(k) + \hat{P}\, f(k) \end{aligned} \qquad (6\text{-}4)$$

$$\hat{y}(k) = G(q)\, u(k)\,, \qquad G(q) = \underline{\hat{O}}\, \left(\underline{\underline{E}} - \underline{\underline{\hat{M}}}\, q^{-1}\right)^{-1} \underline{\hat{N}} q^{-1} + \hat{P} \qquad (6\text{-}5)$$

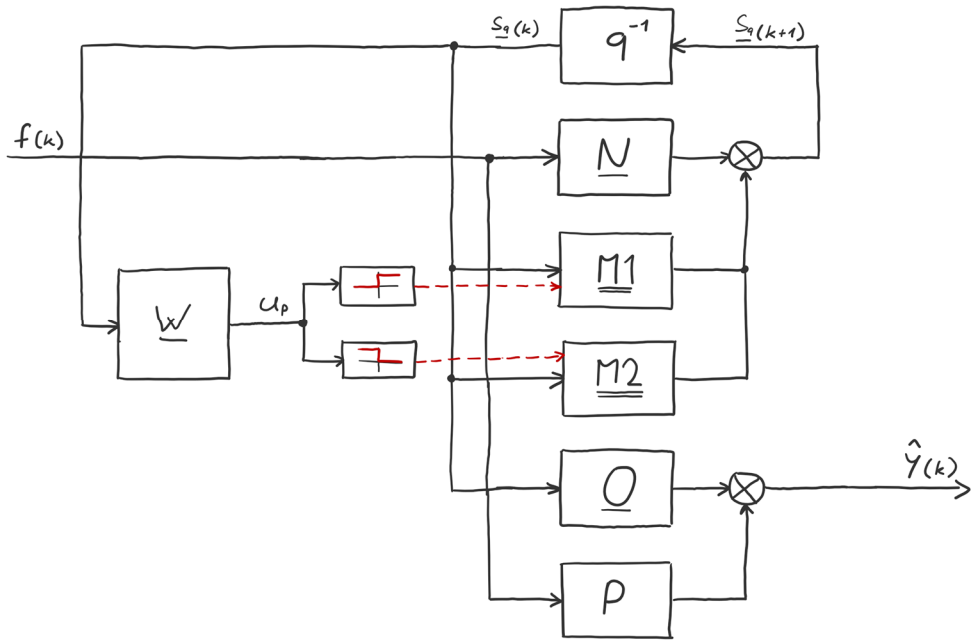
Graficky je přenos OE modelu *(6-1)* znázorněn na Obr. 6.1 vpravo. Pokud se ovšem spokojíme pouze s informací, že z matic diskrétního stavového popisu lze sestavit přenos OE modelu, je takový přenos možné znázornit graficky na Obr. 6.1 vlevo, kde vektor $\underline{s}_q$ představuje vektor stavů v modálních souřadnicích. Takový popis, jak již bylo uvedeno, má nadbytečné parametry, ovšem v dalších krocích umožní snadné zavedení takové nelinearity, která byla zvolena.



Obr. 6.1 Graf OE modelu pomocí matic stavového popisu (vlevo) a pomocí koeficientů čitatele a jmenovatele přenosu (vpravo)

6.2 Přidání nelinearity do modelu

V předchozí kapitole byl vytvořen grafický popis lineárního OE modelu pomocí matic diskrétního stavového popisu. Model poruchy, který má být zaveden se chová tak, že pokud určitá výchylka u_p překročí nulu, změní se skokově hodnota tuhosti. Při simulaci modelu z FEM v předchozích kapitolách byla transformace pro získání výchylky u_p předem známá. V případě identifikace ovšem známá není. Mějme k dispozici vektor modálních souřadnic $\underline{q}$, potom transformace do skutečných výchylek $\underline{u}$ probíhá dle *(2-31)* pomocí neznámé matice vlastních tvarů kmitu. Neznámá výchylka u_p potom zřejmě lze získat pomocí neznámého vektoru $\underline{w}$, použitém na Obr. 6.2 k získání u_p z vektoru stavů s_q. Tento vektor tak provádí transformaci z prostoru modálních souřadnic do prostoru výchylky poruchy u_p, ve které dochází k nárazům, kde na základě její velikosti dochází ke změně tuhosti systému.



Obr. 6.2 Schéma nelineárního OE modelu

Pokud má docházet ke skokové změně tuhosti, budou existovat dva diskrétní stavové popisy popisující lineární flexibilní mechanickou soustavu: $\widehat{M1}, \hat{N}, \hat{O}, \hat{P}$ a $\widehat{M2}, \hat{N}, \hat{O}, \hat{P}$. Tyto systémy se liší pouze v matici $\widehat{M}$, jelikož ta obsahuje informace o tuhosti systému. Dle *(4-9)* jsou ve spojitém stavovém popisu prvky nejvíce ovlivněné změnou tuhosti umístěny v poloze odpovídající submatici Λ, tedy v levé dolní submatici. Jak je zřejmé z *(6-2)*, poloha prvků odpovídajících tuhosti se diskretizací nemění. Tedy pro příklad matice $\widehat{M}$ 6x6 z *(6-3)* se se změnou tuhosti budou výrazně měnit pouze parametry p_7, p_8, p_9. Takové parametry jsou potom označeny $\underline{p_k}$. Vztah mezi $\widehat{M1}$ a $\widehat{M2}$ je vyjádřen změnou jejich parametrů $\underline{p_k}$ dle *(6-6)*, kde matice G je diagonální. Lze tak říci, že pokud je matice G jednotková, jsou si matice $\widehat{M1}$ a $\widehat{M2}$ rovny, tedy oba systémy, mezi kterými se „přepíná" jsou totožné. Nelineární filtr tak přechází v lineární. Prvky matice G tak při dokonalé identifikaci vyjadřují míru změny tuhosti mezi systémy a mohou být příznakem přítomnosti poruchy v měřené soustavě.

$$\underline{G}\ \underline{p_k(\widehat{M1})} = \underline{p_k(\widehat{M2})} \qquad (6\text{-}6)$$

Vytvořený nelineární OE model je zakreslen na Obr. 6.2, vektor $\underline{w}$ transformuje stavový vektor na výchylku u_p dva následující bloky produkují logickou 1 pokud je $u_p > 0$ respektive pokud je $u_p < 0$. Tato logická hodnota spustí buď $\widehat{M1}$, nebo $\widehat{M2}$. Vyhodnocování u_p probíhá v každém kroku, v každém kroku tak může systém užívat jinou matici $\widehat{M}$.

6.3 Hledání parametrů lineárního modelu

V rámci optimalizace pro identifikaci budou hledány parametry lineárního modelu $\underline{p}$, členy matice G a vektor $\underline{w}$. Všechny tyto parametry by bylo možné hledat pomocí nelineární globální optimalizační metody, jako je například genetický algoritmus. Parametrů lineárního modelu je ale poměrně velké množství a taková optimalizace by byla nadměrně časově náročná. Pokud je možné

vyjádřit derivaci výstupu modelu podle parametrů, lze využít gradientní metody pro nelineární lokální optimalizaci. Taková optimalizace může být výrazně rychlejší, ovšem je třeba zajistit počáteční podmínky rozumně blízko minima, jelikož jde o lokální optimalizaci. Za zjednodušujícího předpokladu jednotkové matice G lze získat derivace výstupu podle všech lineárních parametrů, proto pro jejich optimalizaci bude využit Levenberg-Marquardtův algoritmus z kapitoly 2.12.

Uvedený algoritmus dle *(2-46)* a *(2-48)* vyžaduje znalost Jacobiho matice definované dle *(2-47)*. Jelikož výstup měřeného kmitání nezávisí na parametrech modelu, lze Jacobiho matici vyjádřit jako *(6-7)*, tedy jako derivaci vektoru výstupů filtru v čase podle parametrů lineárního modelu. Jacobiho matici tak bude možné složit ze submatic derivací podle parametrů příslušejících jednotlivým maticím diskrétního stavového popisu dle *(6-8)*.

$$J_{kij} = \frac{\partial r(\underline{p})_i}{\partial p_j} = \frac{\partial \left(y_i - \hat{y}(\underline{p})_i\right)}{\partial p_j} = \frac{\partial \hat{y}(\underline{p})_i}{\partial p_j} \tag{6-7}$$

$$J_{kij} = \left[\left[\frac{\partial \hat{y}(\underline{p})_i}{\partial p_{M\,j}}\right], \quad \left[\frac{\partial \hat{y}(\underline{p})_i}{\partial p_{N\,j}}\right], \quad \left[\frac{\partial \hat{y}(\underline{p})_i}{\partial p_{O\,j}}\right]\right] \tag{6-8}$$

Nyní vyjádřeme derivaci výstupu filtru v daném časovém okamžiku podle celých matic stavového popisu. Výstup filtru v daném okamžiku je skalár, derivace dle matice $\widehat{M}$ je uvedena na *(6-9)*, na základě derivace složené funkce dojde k „posunutí zpět v čase" výpočet lze snadno provést, pokud je k dispozici derivace vektoru stavů podle matice $\widehat{M}$ z minulého časového kroku. Při derivaci matice podle totožné matice vznikne tenzor 4. řádu definovaný v *(6-13)*. Další derivace, které jsou provedeny obdobně jsou uvedeny na *(6-10)*, *(6-11)* a *(6-12)*, opět je zde výhodné znát derivace stavů z minulého kroku.

$$\begin{aligned}\frac{\partial \hat{y}(k)}{\partial \widehat{M}_{jk}} &= \frac{\partial \hat{O}_i\, s(k)_i}{\partial \widehat{M}_{jk}} = \hat{O}_i \frac{\partial s(k)_i}{\partial \widehat{M}_{jk}} = \hat{O}_i \frac{\partial\left(\widehat{M}_{il} s(k-1)_l + \widehat{N}_i f(k-1)\right)}{\partial \widehat{M}_{jk}} \\ &= \hat{O}_i\, s(k-1)_l \frac{\partial\left(\widehat{M}_{il}\right)}{\partial \widehat{M}_{jk}} + \hat{O}_i \widehat{M}_{il} \frac{\partial (s(k-1)_l)}{\partial \widehat{M}_{jk}} \\ &= \hat{O}_i\, s(k-1)_l\, \Delta_{ijkl} + \hat{O}_i \widehat{M}_{il} \frac{\partial (s(k-1)_l)}{\partial \widehat{M}_{jk}}\end{aligned} \tag{6-9}$$

$$\begin{aligned}\frac{\partial \hat{y}(k)}{\partial \widehat{N}_j} &= \frac{\partial \hat{O}_i\, s(k)_i}{\partial \widehat{N}_j} = \hat{O}_i \frac{\partial s(k)_i}{\partial \widehat{N}_j} = \hat{O}_i \frac{\partial\left(\widehat{M}_{il} s(k-1)_l + \widehat{N}_i f(k-1)\right)}{\partial \widehat{N}_j} \\ &= \hat{O}_i \widehat{M}_{il} \frac{\partial s(k-1)_l}{\partial \widehat{N}_j} + \hat{O}_i \frac{\partial \widehat{N}_i}{\partial \widehat{N}_j} f(k-1) \\ &= \hat{O}_i \widehat{M}_{il} \frac{\partial s(k-1)_l}{\partial \widehat{N}_j} + \hat{O}_i \delta_{ij} f(k-1)\end{aligned} \tag{6-10}$$

$$\frac{\partial \hat{y}(k)}{\partial \hat{O}_j} = \frac{\partial \hat{O}_i \, s_i}{\partial \hat{O}_j} = s_i \frac{\partial \hat{O}_i}{\partial \hat{O}_j} = s_i \, \delta_{ij} \qquad (6\text{-}11)$$

$$\frac{\partial \hat{N}_i}{\partial \hat{N}_j} = \frac{\partial \hat{O}_i}{\partial \hat{O}_j} = \delta_{ij} \,, \qquad \delta_{ij} = 1 \; pro \; i = j \,, \qquad \delta_{ij} = 0 \; pro \; i \neq j \qquad (6\text{-}12)$$

$$\frac{\partial \left(\hat{M}_{il} \right)}{\partial \hat{M}_{jk}} = \Delta_{ijkl} \,, \qquad \Delta_{ijkl} = 1 \; pro \; ij = kl \,, \qquad \Delta_{ijkl} = 0 \; pro \; ij \neq kl \qquad (6\text{-}13)$$

Uvedenými derivacemi jsou získány objekty o rozměrech matice, podle které probíhala derivace, pro získání derivací podle jednotlivých parametrů je třeba využít znalosti rozmístění parametrů v matici, například pro matici dle *(6-3)* by derivace dle některých parametrů bylo možné získat jako v *(6-14)*.

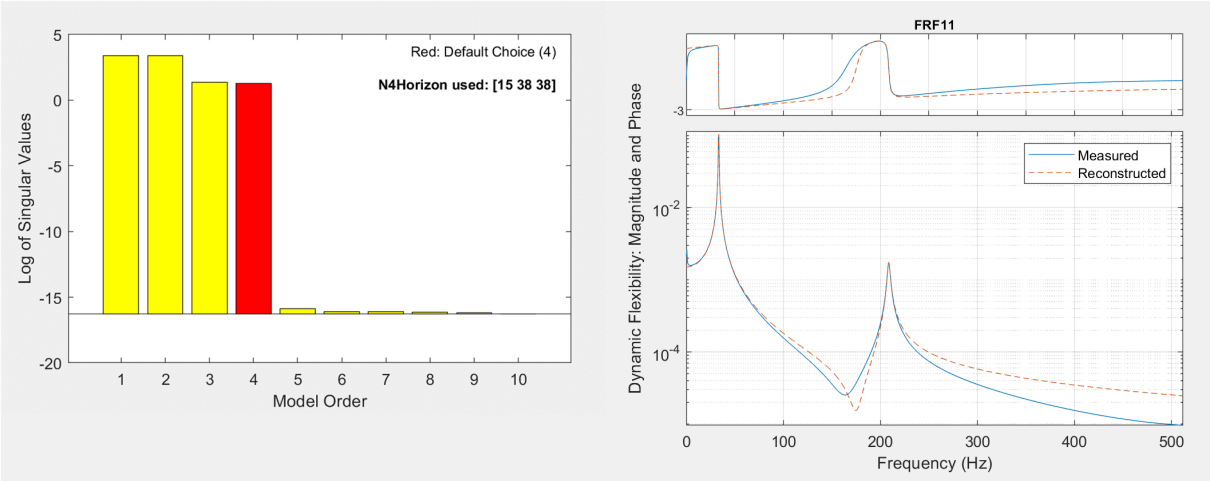
$$\frac{\partial \hat{y}(k)}{\partial p_{M_3}} = \frac{\partial \hat{y}(k)}{\partial \hat{M}_{33}} \,, \qquad \frac{\partial \hat{y}(k)}{\partial p_{M_4}} = \frac{\partial \hat{y}(k)}{\partial \hat{M}_{14}} \qquad (6\text{-}14)$$

6.4 Hledání prvotního odhadu

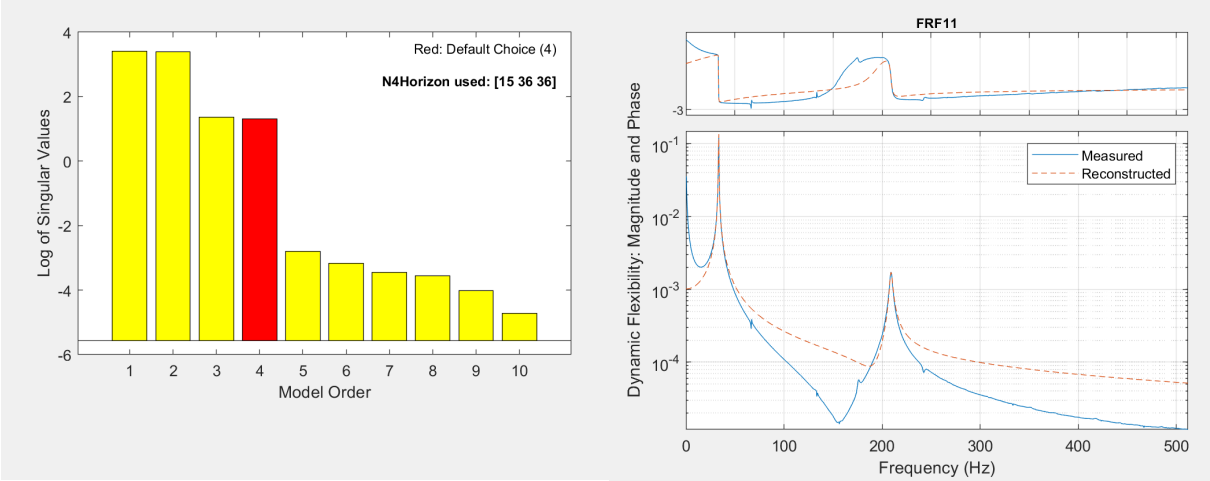
Jelikož je pro získání lineárních parametrů filtru použita lokální optimalizace, je třeba jí poskytnout počáteční hodnoty parametrů. Při poskytnutí náhodných hodnot hrozí, že optimalizace nebude konvergovat k minimu. Pro zajištění vhodného prvotního odhadu byly využity obvyklé metody zpracování EMA. Druhým, nejspíš ještě významnějším úkolem pro prvotní odhad je stanovení řádu systému. K tomu lze využít Hankelova singulární čísla, která naznačují, jak významný je daný řád pro systém, lze tak stanovit hranici, od které nebudou další řády systému považovány za významné.

Hledání prvotního odhadu bylo provedeno pomocí souboru v příloze 10.3 simulated_data_processing (Matlab). Na Obr. 6.3 je znázorněn výstup tohoto skriptu pro simulované měření systému bez poruchy. Vlevo je graf Hankelových singulárních čísel, ze kterého jasně vyplývá, že první 4 řády jsou pro systém významně důležitější než ostatní. Graf na obrázku vpravo zobrazuje měřenou frekvenční přenosovou funkci a přenosovou funkci zrekonstruovanou z modálních dat získaných algoritmem LSCE popsaným v 2.9, tento algoritmus podle vizuální kontroly nejlépe aproximoval poskytovaná data ze všech nabízených v rámci Matlab identification toolbox funkce modalfit, která byla použita (ř.36). Této funkci je poskytován také počet hledaných vlastních frekvencí, z průběhu FRF jasně plyne, že dominantní vliv mají 2 vlastní frekvence, což odpovídá také Hankelovým singulárním číslům, která naznačují, že systém má 4 stavy (pro mechanický systém 2 rychlosti a 2 polohy, tedy také 2 vlastní frekvence).

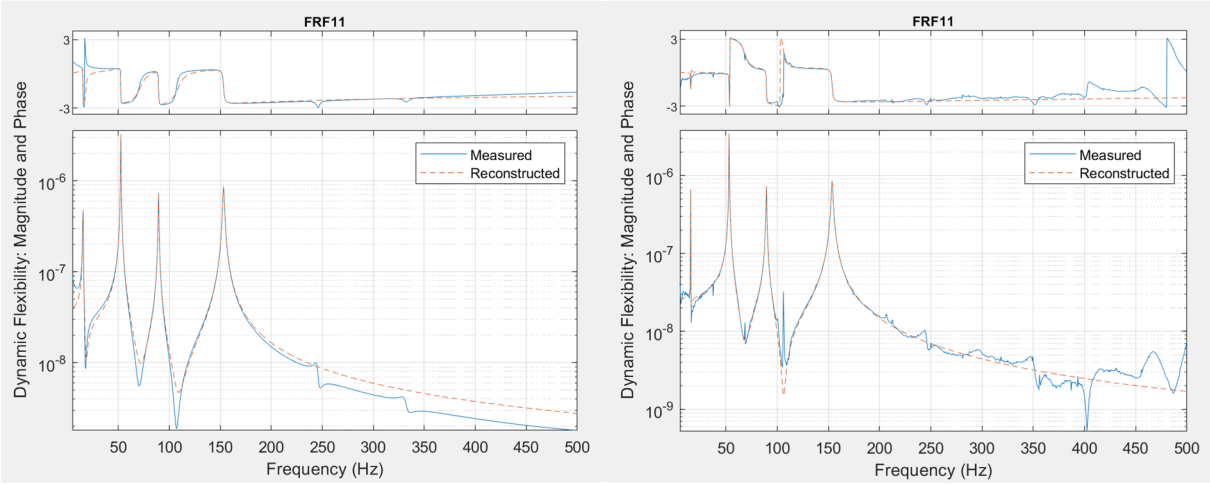
Na Obr. 6.4, jsou obdobné výsledky pro simulovaná data systému s poruchou. Výsledek Hankelových singulárních čísel již není tak jednoznačný, což odpovídá očekávání, vzhledem k určité nelinearitě v datech. Obr. 6.5 potom ukazuje FRF dalších simulovaných dat, tentokrát se čtyřmi výraznými vlastními frekvencemi. Skript v uvedeném souboru nakonec vypočítá odhad hodnot matic Λ a B_p z modálních vlastností určených LSCE, ty poslouží k získání prvotního odhadu parametrů lineárního modelu.



Obr. 6.3 (vlevo) Hankelova singulární čísla pro modely daných řádů, (vpravo) FRF ze simulace bez poruchy – modře a rekonstruované pomocí parametrů nalezených LSCE



Obr. 6.4 (vlevo) Hankelova singulární čísla pro modely daných řádů, (vpravo) FRF ze simulace s poruchou – modře a rekonstruované pomocí parametrů nalezených LSCE



Obr. 6.5 (vlevo) FRF ze simulace bez poruchy – modře a rekonstruované pomocí parametrů nalezených LSCE, (vpravo) FRF ze simulace s poruchou – modře a rekonstruované pomocí parametrů nalezených LSCE

6.5 Implementace L-M algoritmu pro nelineární filtr

Optimalizace parametrů lineárního filtru probíhá pomocí L-M, ovšem postup optimalizace ostatních parametrů nebyl stanoven. Pro tuto chvíli řekněme, že tyto parametry byly pevně stanoveny nadřazeným procesem. Máme tedy vstupní data ve formě naměřeného či simulovaného vstupu – silového pulzu a výstupu – záznamu rychlostí kmitání. Dále prvotní odhad matic Λ a B_p a pevně stanovenou matici G a vektor $\underline{w}$, souhrně nazývané jako nelineární parametry. Úkolem je provést optimalizaci lineárních parametrů nelineárního filtru tak, aby výstup filtru co nejlépe napodoboval naměřený výstup.

Toto je provedeno v souboru v příloze 10.2 L_M_Optimization (Python) v jazyku Python. Vzhledem k používání tenzorů vyššího řádu je zde využita knihovna TensorFlow, které je nicméně pro tento soubor zakázáno používat k výpočtům grafickou kartu, jelikož bylo zjištěno, že při tomto nastavení tento konkrétní výpočet probíhá rychleji. Nejspíš při častém sdílení dat mezi procesorem a grafickou kartou docházelo ke zpomalení výpočtu. Funkce get_MNO (ř.11) pro lineární parametry vrací matice diskrétního stavového popisu podle např. *(6-3)*. Funkce get_parameters (ř.38) provádí opak, tedy pro matice stavového popisu vrací lineární parametry. Funkce get_delta (ř.58) pro danou velikost systému (počet stavů) vrátí vhodný tenzor Δ_{ijkl}. Funkce get_R_mat získá parametry learning rate odpovídající členu $\alpha_k E$ v *(2-48)*. Tato funkce má možnost nastavovat různým způsobem learning rate, ovšem testováním nebylo zjištěno žádné zlepšení, proto jsou nastavením parametru dimple=0 tyto možnosti zablokovány. Funkce get_discrete (ř.100) provede sestavení stavového popisu a jeho diskretizaci prvního řádu, vrací tak diskrétní stavový popis dle *(6-4)*. Funkce get_M2 (ř.125) pro lineární a nelineární parametry vrátí matici $\widehat{M2}$ dle *(6-3)* a *(6-6)*. Funkce get_right_M (ř.144) pro matice $\widehat{M1}$, $\widehat{M2}$, vektor stavů a vektor $\underline{w}$ vrátí matici $\widehat{M}$, která má v daný okamžik být použita dle Obr. 6.2.

Dále je definována třída L_M_Nonlin (ř.163), která pro vytvoření objektu požaduje diagonály matic Λ a B_p a vlastní vektor $\underline{v}$ – tedy prvotní odhady těchto hodnot, dále frekvenci měření, měřený vstup a výstup systému, který má filtr napodobovat. Při inicializaci jsou následně vypočteny prvotní odhady lineárních parametrů pomocí funkce get_discrete a get_parameters. Metoda reset_init (ř.197) použije poslední optimalizací získané lineární parametry a nastaví je jako nový prvotní odhad. Metoda start (ř.205) odstartuje L-M optimalizaci pro zadané nelineární parametry po zadané množství epoch. Vstupní parametry jsou na počátku transformovány tak, že z rozsahu <-1; 1> vznikne rozsah <0,999; 1,001>, po této transformaci teprve vytvoří matici *G.* To způsobí, že nulový vstupní parametr metody vytvoří jednotkovou matici G, tedy lineární filtr. Další vstupní parametr určuje vektor $\underline{w}$. Velikost tohoto vektoru nemá vliv na okamžik, kdy souřadnice u_p prochází nulou, to je určeno pouze poměrem jednotlivých členů parametrů. Proto například vektor o velikosti 2 je určen pouze jedním parametrem a celý vektor je dopočten při inicializaci. Hlavní část této metody se odehrává ve dvou for cyklech.

Vnitřní for cyklus (ř.254) iteruje přes každý časový krok, pomocí funkce get_right_M je získána správná matice $\widehat{M}$ na základě stavů z minulého časového kroku, dále spočítá nové stavy a výstup filtru ze stavů z minulého kroku a ze vstupu pomocí *(6-4)*. Dále (ř.258) jsou vypočítány derivace stavů podle matic $\widehat{M}$ *a* $\widehat{N}$ a teprve poté derivace výstupů, to vše podle *(6-9)* až *(6-13)*. Důvodem výpočtu nejprve derivací stavů a poté výstupů je, že derivace stavů jsou poté uloženy pro příští časový krok. Dále je (ř.263) vypočten příslušný řádek Jacobiho matice příslušející danému časovému kroku dle *(6-8)* a pomocí funkce get_parameters, která zjistí derivace výstupu podle jednotlivých parametrů ze dříve získaných derivací podle matice. Dále je zjištěna odchylka od měřeného výstupu $r(\underline{p})$ a výpočet je opakován pro další časový krok, dokud není zjištěn celý výstupní vektor filtru $\hat{y}(\underline{p})_i$ a vektor odchylek $r(\underline{p})_i$ ve všech časových krocích.

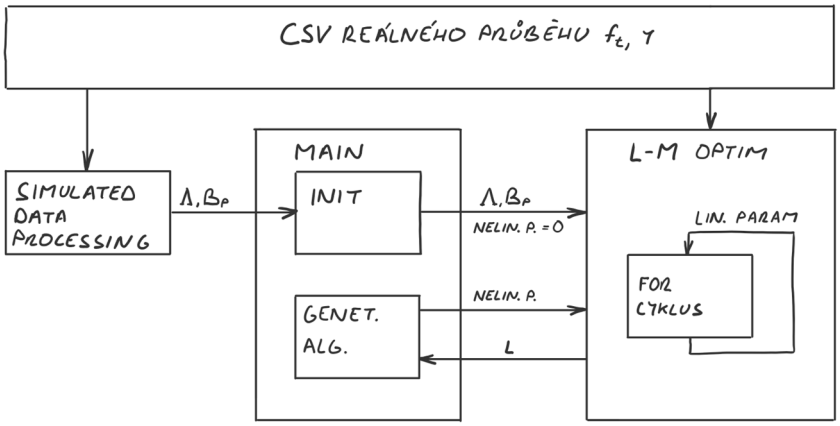
Vnější for cyklus (ř.242) iteruje přes jednotlivé epochy L-M algoritmu. Nejprve stanoví počáteční podmínky pro vnitřní for cyklus, poté jej spustí. Následně je zjištěna suma čtverců odchylek *(2-45) L*, v kódu označená Err2. Pomocí *(2-46)* a *(2-48)* je zjištěna aktualizace lineárních parametrů L-M algoritmem (ř.274). Následně se cyklus opakuje pro další generaci a postupně hledá optimální parametry jejich úpravou v každé generaci.

6.6 Hledání parametrů nelinearity

V předchozí kapitole byl použit předpoklad, že jsou parametry nelinearity pevně stanoveny. Důvodem je, že podle nich není stanovena derivace výstupu filtru. Nicméně pro identifikaci poruchy je třeba optimalizovat a identifikovat i tyto parametry, protože právě podle nich bude usuzováno, jestli měřený systém obsahuje poruchu. Proto byla celá optimalizace rozdělena do dvou cyklů, vnitřního a vnějšího. Vnější cyklus stanoví parametry nelinearity a spustí vnitřní cyklus – L-M optimalizaci. Ten pro dané nelineární parametry nalezne optimální lineární parametry tak, aby minimalizoval *L.* Vnější cyklus poté upraví nelineární parametry a opět spustí cyklus vnitřní, který nalezne další hodnotu *L.* Vnější cyklus má za úkol nalézt takové nelineární parametry, aby *L* bylo minimální.

6.6.1 Genetický algoritmus

Správná hodnota nelineárních parametrů není dopředu ani přibližně známa, pro jejich optimalizaci byl zvolen genetický algoritmus, tedy globální optimalizační metoda. Úloha globální optimalizace je vytvořena v souboru v příloze 10.1 main (Python) a jeho funkce je také zjednodušeně nastíněna na Obr. 6.6.



Obr. 6.6 Schéma vnitřního a vnějšího optimalizačního cyklu a sdílení dat mezi nimi.

Nejprve jsou načtena data z měření a počáteční odhad parametrů vytvořený v kapitole 6.4. Na základě těchto dat je vytvořen objekt model třídy L_M_Nonlin (ř.34) definované v minulé kapitole. Lokální optimalizace je spuštěna s dvaceti epochami bez nelineárních parametrů, pomocí metody reset_init jsou lineární parametry, nalezené pro filtr bez nelineárních parametrů (lineární filtr), nastavené jako počáteční odhad parametrů pro všechny další optimalizace. Dále je definována funkce fitness_func (ř.39), jejím vstupem jsou nelineární parametry optimalizované genetickým algoritmem, které jsou dále označovány jako geny. Funkce spustí 5 epoch optimalizace lineárních parametrů se zadanými geny. Dále zjistí hodnotu *L,* která je vlastností objektu model a vrátí její převrácenou hodnotu, která bude genetickým algoritmem maximalizována.

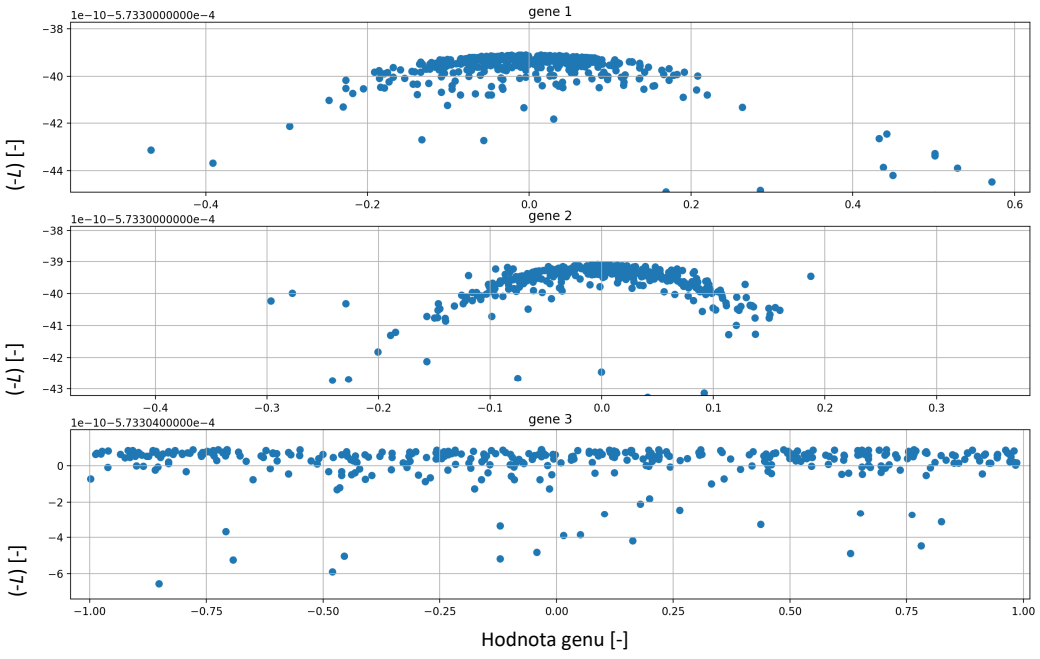
Následně jsou nastaveny vlastnosti genetického algoritmu na (ř.53) až (ř.67). Počet genů odpovídá počtu vstupů fitness_func. Počáteční rozsah genů je nastaven na <-1; 1>, což pro hodnoty

prvků matice G znamená rozsahu <0,999; 1,001>, nicméně oblast, na které se mohou geny odpovídající matici G pohybovat během optimalizace, není omezena. Pokud se optimální nelineární parametry nacházejí mimo původní rozsah, může k nim metoda postupně dospět. Pravděpodobnost mutace je 90 % a mutace je omezena na rozsah <-0,1; 0,1>. Po vytvoření objektu třídy pygad.GA, je metodou start optimalizace spuštěna.

Pro otestování metody byl vytvořený filtr otestován na simulovaných datech z kapitoly 4. Očekává se, že geny genetického algoritmu odpovídající matici G se budou, při použití na datech bez poruchy, blížit nule, zatímco při použití na datech se simulovanou poruchou, se budou od nuly vzdalovat, případně se ustálí na nenulové hodnotě.

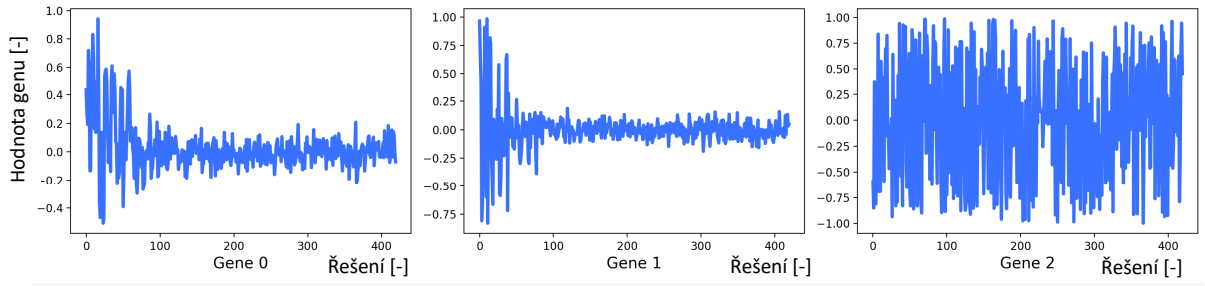
6.6.2 Vyhodnocení pro data bez poruchy

Pro prvotní otestování byla použita vstupní data simulace konstrukce bez poruchy tvaru 1, jimž odpovídá Obr. 6.3. Tato data mají zřejmě dominantní dvě vlastní frekvence, filtr tedy bude mít dva prvky matice G a jeden prvek určující vektor $\underline{w}$, genetický algoritmus bude mít tedy tři geny. Na Obr. 6.7 jsou tyto tři geny zobrazeny, každá tečka odpovídá jednomu jedinci, na ose x je zobrazena hodnota daného genu jedince, na ose y je záporná hodnota L, tedy obdoba fitness funkce. Na prvních dvou grafech je jednoznačně zřetelné, že pro hodnoty genů jdoucí k nule, se fitness funkce blíží maximu, tedy odchylka L je minimální. To odpovídá očekávání, lineární data nejlépe aproximuje lineární filtr, tedy filtr s jednotkovou maticí G, tedy nulovými hodnotami prvních dvou genů.

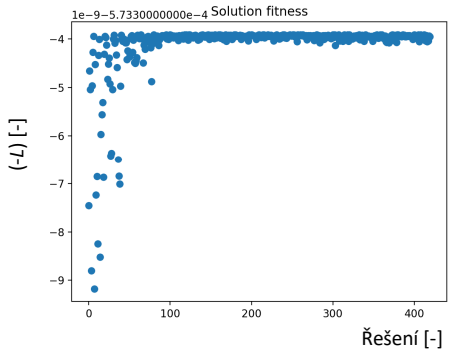


Obr. 6.7 Hodnoty tří genů ga při identifikaci dat bez poruchy, tečky odpovídají jednotlivým řešením

Obr. 6.8 zobrazuje hodnoty genů v průběhu simulace. První dva geny se rychle ustalují kolem nuly, další šum je způsoben 90 % pravděpodobností mutace. Třetí gen určující vektor $\underline{w}$ se neustaluje a Obr. 6.7 ukazuje, že na hodnotě třetího genu příliš nezáleží. Obr. 6.9 ukazuje průběh *(-L)* během optimalizace, je vidět, že algoritmus poměrně rychle sníží odchylku výstupu filtru od měřených dat.



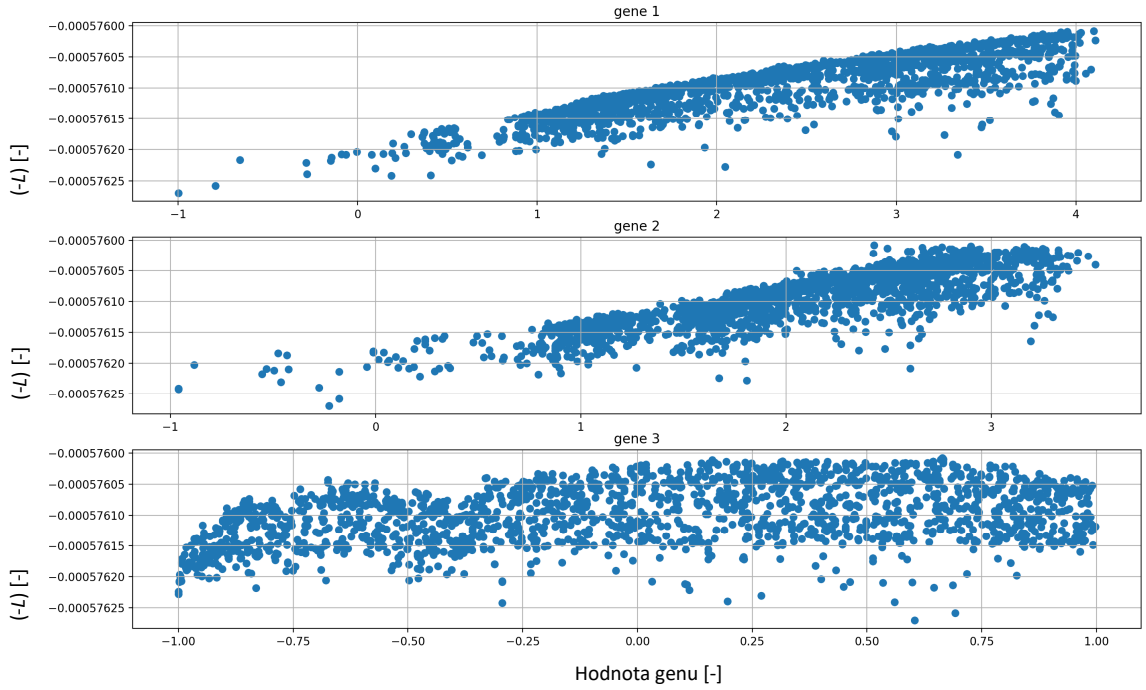
Obr. 6.8 Hodnoty genů při identifikaci dat bez poruchy v průběhu optimalizace pro všechna řešení



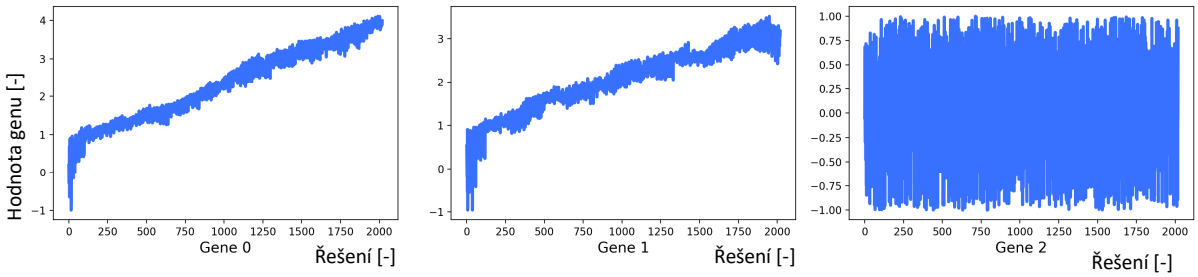
Obr. 6.9 Záporná hodnota součtu čtverců odchylky od dat bez poruchy v průběhu optimalizace pro všechna řešení

6.6.3 Vyhodnocení pro data s poruchou

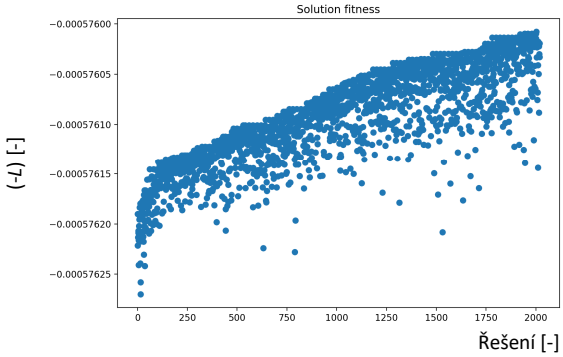
V předchozí kapitole byla identifikace aplikována na lineární data a byly nalezeny parametry odpovídající lineárnímu filtru. Dále je samozřejmě třeba otestovat, jak bude filtr reagovat na data s poruchou. Byla použita simulována data konstrukce tvaru 1 se simulovanou poruchou odpovídající Obr. 6.4. Tedy „porouchanou“ verzí dat použitých v předchozí kapitole. Na Obr. 6.10 je zřejmé, že nulová hodnota prvních dvou genů není optimální. Filtr tedy dosahoval lepší aproximace dat, když byl nelineární. Optimální hodnota prvních dvou parametrů je zřejmě mimo počáteční rozsah hodnot genů. Z Obr. 6.11 je zřejmé, že hodnoty prvních dvou genů se v průběhu optimalizace vzdalují rychle od nuly. Toto vzdalování je umožněno mimo jiné právě vysokou pravděpodobností mutace. Při optimalizaci po více generací by se zřejmě hodnoty ještě více vzdálili. Obr. 6.12 ukazuje průběh *(-L)* během optimalizace, je vidět, že algoritmus snižuje odchylku na podobnou hodnotu, jako v předchozím příkladu, ovšem je zde další potenciál ke snížení.



Obr. 6.10 Hodnoty tří genů ga při identifikaci dat s poruchou, tečky odpovídají jednotlivým řešením



Obr. 6.11 Hodnoty genů při identifikaci dat bez poruchy v průběhu optimalizace pro všechna řešení



Obr. 6.12 Záporná hodnota součtu čtverců odchylky od dat s poruchou v průběhu optimalizace pro všechna řešení

6.6.4 Vyhodnocení výsledků

Na základě výsledků a grafů v kapitolách 6.6.2 a 6.6.3 se zdá, že identifikace filtru dosahuje velmi dobrých výsledků. Ovšem úspěšnost klasifikace nelze posuzovat na základě jednoho testu, proto byla identifikace filtrem aplikována na další simulovaná data, která byla využita jako testovací data i v dalších kapitolách, jedná se o simulace konstrukcí tvarů 14 a 15. Tato data ovšem již

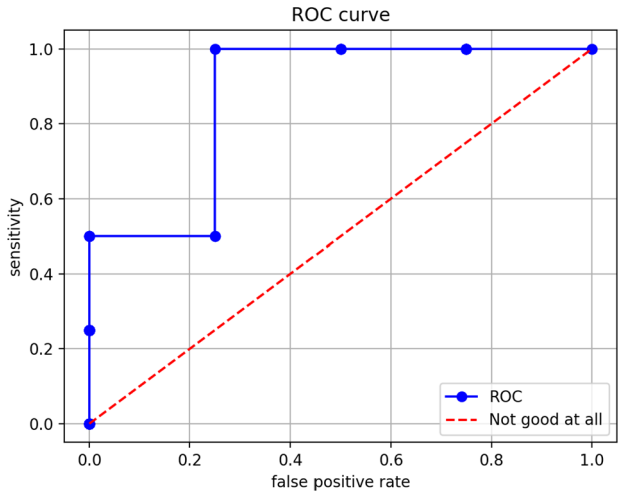
neměla natolik jednoznačně dominantní dvě vlastní frekvence, například FRF výstupních dat tvaru 14 je zobrazena na Obr. 6.5. Je zřejmé, že zde jsou dominantní alespoň 4 vlastní frekvence.

Vyhodnocení pro testovací data je provedeno v Obr. 6.13. Pro jednotlivé testovací konstrukce byla provedena identifikace na datech s poruchou a bez poruchy. Sloupec *L* uvádí součet čtverců odchylek pro nejlepší parametry identifikace, sloupce *G1* až *G4* uvádějí hodnoty genů nejúspěšnějších jedinců. Pro tvar 14 byla provedena nejprve identifikace pomocí filtru se dvěma vlastními frekvencemi – čtyřmi stavy a poté pomocí filtru se čtyřmi vlastními frekvencemi – tedy čtyřmi parametry matice *G.* Nicméně ani jeden z přístupů nevedl k vytvoření natolik dobrých výsledků, jakých bylo dosaženo pro tvar 1. Cílem je vytvořit parametry, u nichž by byl velmi výrazný rozdíl mezi jejich hodnotami pro data s poruchou a bez poruchy, to je předpoklad pro spolehlivou klasifikaci. Jako jednoduchá metoda porovnání výsledků byla zvolena suma čtverců genů odpovídajících matici *G.* Poměr mezi těmito hodnotami pro data s poruchou a bez poruchy je uveden v posledním sloupci, pro tvar 1 je tento poměr velmi vysoký, tedy rozdíl je výrazný. Výsledky pro ostatní data již nejsou tak přesvědčivé, nelineární parametry pro data bez poruchy se neblíží nijak výrazně nule. Zdá se, že je to způsobeno tím, že lineární filtr nedokáže aproximovat chování příslušející dalším vlastním frekvencím, které byly zanedbány, a tak se toto chování může snažit napodobit pomocí nelinearity.

		L	*G1*	*G2*	*G3*	*G4*	suma čtverců *G*	Poměr
tvar 1, směr buzení 10	nerozbitý	5,73E-04	-4,50E-03	-3,95E-03			3,59E-05	
	rozbitý	5,76E-04	1,33E+01	1,96E+00			1,80E+02	5018623,58
tvar 15, směr buzení 15	nerozbitý	5,39E-05	2,69E-01	-9,51E-01			9,77E-01	
	rozbitý	5,40E-05	7,23E-01	-1,12E+00			1,79E+00	1,83
tvar 14, směr buzení 4	nerozbitý	4,66E-11	-3,69E-01	-1,18E+00			1,52E+00	
	rozbitý	5,20E-11	-7,60E-01	-1,09E+00			1,76E+00	1,16
tvar 14, směr buzení 4	nerozbitý	1,14E-11	1,08E+00	4,55E-01	6,72E-01	5,67E-01	2,15E+00	
	rozbitý	1,63E-11	1,11E+00	-6,46E-01	5,03E-01	-1,03E+00	2,97E+00	1,38

Obr. 6.13 Tabulka s výsledky identifikace modelu na testovacích datech

Pokud by byly hodnoty sumy čtverců genů použity jako klasifikační parametry pro binární klasifikaci, bylo by třeba stanovit treshold, hodnotu, pomocí které by byly klasifikační parametry rozděleny do tříd. ROC křivka (Obr. 6.14) vyhodnocuje metriky sensitivity a false positive rate pro různé hodnoty tresholdu. Z průběhu křivky se zdá, že pro požadavek 100 % sensitivity je možné stanovit treshold takový, že false positive rate nebude vyšší než 30 %. Je třeba ovšem upozornit, že pro tak malé množství testovacích dat, jako je uvedeno zde, není možné kvalitně vyhodnotit ROC křivku i další metriky. Další důležitou informací je, že pro některá testovací data se nepodařilo vůbec získat výsledky, jelikož optimalizace lineárních parametrů byla nestabilní. To, společně s faktem, že identifikace byla velmi časově náročná, vedlo k vyhodnocení jen velmi malého vzorku testovacích dat.



Obr. 6.14 ROC křivka pro klasifikaci na základě sumy čtverců G

7 Zpracování dat neuronovou sítí

Cílem této práce je vytvoření metody pro detekci poruchy, nikoliv zjišťování umístění či závažnosti poruchy. Pro strojové učení tak jde o úlohu binární klasifikace. Jak již bylo uvedeno, strojové učení nevyžaduje zadání pravidel, podle kterých má vstupní data transformovat. Vstupem pro supervised machine learning je velké množství trénovacích dat, označené správnými výsledky. Strojové učení potom samo vytvoří pravidla, transformace, které bude na vstupní data aplikovat. Cílem zde vyvářeného algoritmu strojového učení bude splnit úkoly 2 a 3 uvedené v cílech práce. Strojové učení si vytváří vlastní pravidla na základě trénovacích dat, vytvoření účinných pravidel tak závisí na poskytnutých trénovacích datech. Trénovací data musí klasifikační algoritmus donutit ignorovat ty vlastnosti, které nejsou podstatné (nepřejeme si je vyhodnocovat) a vnímat podstatné vlastnosti, signifikantní pro skupiny, do kterých má data rozdělovat. To je jeden z důvodů, proč byl vytvořen algoritmus pro generování simulovaných měření vibrací konstrukcí s odlišnými tvary, rozměry, materiálovými vlastnostmi. Cílem totiž je, aby tyto vlastnosti byly ignorovány, přičemž by bylo obtížné vyrábět tisícovky odlišných konstrukcí a provádět jejich reálné měření.

Existuje mnoho metod supervised machine learning, jak již bylo řečeno v úvodu, tato práce se chce zaměřit na nelineární projevy poruch, které se projevují v hrubých naměřených datech. Není přitom jistota, zda se projeví také po určitém zpracování těchto dat. Hluboké učení umožňuje přímé použití hrubých dat bez jejich předchozího zpracování, a tak je pro tuto úlohu velmi vhodné. Mělo by dokázat samo najít příznaky sledovaného chování, aniž by podstata těchto příznaků byla známa. Proto budou v této části data zpracovávána pomocí hlubokých neuronových sítí (DNN).

Řekněme, že úlohu klasifikace pomocí supervised machine learning rozdělíme do tří operací: konstrukce příznaků, transformace příznaků (ta je učena pomocí trénovacích dat) a vyhodnocení transformovaných dat. Předchozí metoda – identifikace filtru, by se v kontextu strojového učení dala označit za předzpracování dat, konstrukci příznaků. Příznaky produkované takovou operací jsou přitom natolik malé a konkrétní – pouze jedno číslo, že úlohou strojového učení by bylo pouze nalezení vhodného předělu (tresholdu) od jaké hodnoty příznaku říci, že je konstrukce poškozená. Naproti tomu konstrukce příznaků pro hluboké učení obecně vytváří stále poměrně rozsáhlé balíky příznaků (například obrázky) a ponechává tak významnou část redukce dat na operaci transformace příznaků. Lze tak říci, že mezi dvěma zmíněnými metodami je rozdíl především v tom, jak hluboko zasahují jednotlivé operace.

Pro splnění úkolu 2 z cílů práce budou zde sloužit sítě s více vstupy, kdy jedním vstupem budou data referenčního měření konstrukce, o které víme, že je bez poruchy. Druhým vstupem budou data měření konstrukce, která se do určité míry odlišuje od referenční v lineárních parametrech a není známo, zda tato konstrukce obsahuje poruchu. Informace o tom, zda je porucha přítomna bude výstupem této sítě.

Pro splnění úkolu 3 z cílů práce bude sloužit síť s jedním vstupem. Tímto vstupem budou naměřená data na konstrukci, o které nevíme, zda obsahuje poruchu, či ne. Informace o tom, zda poruchu obsahuje bude výstupem sítě.

Pro splnění této úlohy byly jako nejvhodnější vybrány plně propojené sítě tvořené dense vrstvami a konvoluční sítě tvořené vrstvami convolution. V této práci nebudou neuronové sítě programovány „od začátku“, šlo by zřejmě o ztrátu času, vzhledem k tomu, že byly vyvinuty vhodné knihovny umožňující snadnou tvorbu neuronových sítí například v programu Python. Tyto knihovny umožňují uživateli nadefinovat síť, určit trénovací algoritmy, parametry sítí a podobně. Přitom samotné výpočty, které je nutné provádět během trénování, a které mohou být

pro velké sítě výpočetně velmi náročné, mají tyto knihovny předdefinované daleko efektivněji, než by byl uživatel schopen sám bez využití těchto knihoven. Pro účely této práce byla využita knihovna TensorFlow a knihovna Keras. TensorFlow umožňuje, po instalaci správných ovladačů, využít k výpočtům dedikovanou grafickou kartu počítače, díky které mohou výpočty prováděné při trénování sítě trvat výrazně kratší dobu [40].

7.1 Konstrukce příznaků (feature engineering)

Jak již bylo uvedeno, vhodným vstupem pro DNN mohou být i hrubá data. Pokud však znalosti problematiky umožňují provést před poskytnutím dat DNN takovou transformaci, která umožní její lepší výsledky, měla by být provedena. Taková transformace může například zvýraznit vlastnosti, které jsou významné pro požadovanou klasifikaci. Pro tuto úlohu jsou hrubými daty časové záznamy rychlosti kmitání jednoho bodu konstrukce. Tyto záznamy jsou buď nasimulovány, nebo naměřeny na reálné konstrukci. Na těchto záznamech lidské oko nedokáže rozpoznat rozdíl mezi daty konstrukce s poruchou a bez poruchy. To samozřejmě neznamená, že rozdílnost by nezaznamenala ani DNN, ale může být vhodné ji zvýraznit. Kapitola 5.4 ukazuje, že je možné pozorovat výrazné rozdíly mezi naměřenými daty ve frekvenční oblasti, kdy konstrukce s poruchou jak simulované, tak měřené, mají ve frekvenční oblasti malé peaky. To je rozdíl, který zaregistruje i lidské oko a lze tak říci, že jde o vhodné zvýraznění.

Pokud jsou k dispozici také data referenčního měření, mohlo by se zdát, že dalším vhodným zvýrazněním je odečtení referenčních dat od zkoumaných ve frekvenční oblasti, mohly by se tak zvýraznit drobné peaky, které naznačují poruchu. Nicméně, jak již bylo uvedeno, referenční konstrukce se drobně liší v různých parametrech, proto i vlastní frekvence jsou, více či méně, odlišné. Při odečtení dvou peaků posunutých v ose x se peaky nevyruší, vzniká pouze větší množství peaků, mezi kterými mohou být nelineární vlastnosti ztraceny. Navíc každá vlastní frekvence se může posunout o jinou vzdálenost. Přesto referenční data mohou poskytnout důležitou informaci pro DNN, která tak může mít „představu“ jak pro danou konstrukci může vypadat vstup nezatížený poruchou. Proto budou v případě sítí se dvěma vstupy skutečně obě měření vkládána do sítě zvlášť s předpokladem, že uvnitř sítě se „prolnou“ a výsledkem by měla být informace, zda zkoumané měření bylo získáno z konstrukce s poruchou.

Klíčová je také normalizace dat, jak již bylo uvedeno dříve. Tento fakt vyšel najevo také v průběhu sestavování DNN pro tuto práci, kdy vlivem chyby v kódu docházelo ke špatné normalizaci (data neměla nulovou průměrnou hodnotu) a DNN měla přesnost klasifikace 50 %, což pro binární klasifikaci odpovídá náhodné klasifikaci. Po opravě této chyby stejné DNN dosahovali výrazně lepších výsledků.

Zvolená konstrukce příznaků tak byla následující: naměřený či simulovaný vektor diskrétního záznamu rychlosti kmitání jednoho bodu soustavy $\underline{\dot{u}_m}$ *(7-1)* byl zpracován a normalizován dle *(7-2)*. Tedy byla provedena rychlá Fourierova transformace dat (FFT), z její absolutní hodnoty byl vypočten logaritmus o základu 10 (log), od tohoto výsledku byla odečtena jeho průměrná hodnota (mean) a to celé bylo vyděleno směrodatnou odchylkou těchto dat (std). Všechna vstupní data do DNN tak budou mít nulovou průměrnou hodnotu a směrodatnou odchylku rovnou jedné.

$$\underline{\dot{u}_m} = [\dot{u}(0), \ldots, \dot{u}(k \cdot \Delta t), \ldots, \dot{u}(t_{end})] \qquad (7\text{-}1)$$

$$N\_in = \frac{log\left(\left|FFT(\underline{\dot{u}_m})\right|\right) - mean\left(log\left(\left|FFT(\underline{\dot{u}_m})\right|\right)\right)}{std\left(log\left(\left|FFT(\underline{\dot{u}_m})\right|\right)\right)} \qquad (7\text{-}2)$$

Normalizovaná simulovaná data byla uložena do souborů csv, kde jeden soubor odpovídá jednomu záznamu *N_in*. Data jsou rozdělena do složek broken, reference a unbroken, každý záznam je nazván názvem složky a číslem záznamu, tedy například broken_3, unbroken_3 a reference_3 jsou k sobě patřící záznamy měření s poruchou, bez poruchy a referenčního měření odpovídající poslední větvi na schématu dle Obr. 4.24. Bylo vytvořeno 4395 trénovacích dat pro každou ze tří skupin, toto množství se později ukázalo jako dostatečné, ovšem je snadné případně vyprodukovat další data.

7.2 Tvorba datového toku

Při použití velkého množství trénovacích dat vzniká problém se zacházením s těmito daty. Může totiž nastat situace, že data pro trénování nemohou být „načtena“, jelikož se nevejdou do operační paměti počítače. Pro reprezentaci trénovacích dat je proto vytvořen objekt třídy tf.data.Dataset, ten vytvoří datový tok trénovacích dat z místa kde jsou uložena pro metody provádějící trénování DNN [41]. Datový tok má za úkol poskytovat algoritmu trénujícímu DNN balíčky trénovacích dat, které jsou tvořeny dvojicemi k sobě příslušejících vzorků (samples) a označeních správného výsledku (labels).

Jak již bylo zmíněno, je třeba vytvořit nejen trénovací, ale i testovací data. Testovací data jsou také tvořena dvojicemi samples a labels, ovšem labels jsou využity pouze k vyhodnocení úspěšnosti sítě, síť na jejich základě neupravuje své parametry. Aby bylo testování věrohodné, musí být testovací data odlišná od trénovacích. V případě, že máme k dispozici například množinu jedinečných obrázků, lze trénovací a testovací data vytvořit rozdělením této množiny. Zde použitá data jsou však vytvářena pomocí schématu dle Obr. 4.24. při náhodném rozdělení takto vyprodukovaných dat tak může nastat situace, kdy v trénovací a testovací množině bude měření simulované na stejném tvaru konstrukce se stejnými rozměry, pouze s odlišným místem buzení a podobně. Pro vhodnou odlišnost trénovacích a testovacích dat proto byla trénovací data produkována pouze z tvarů konstrukce 1 až 13 (viz Obr. 4.16 až Obr. 4.22) a testovací data byla vytvořena pouze za použití tvarů 14 a 15 (Obr. 4.22 a Obr. 4.23).

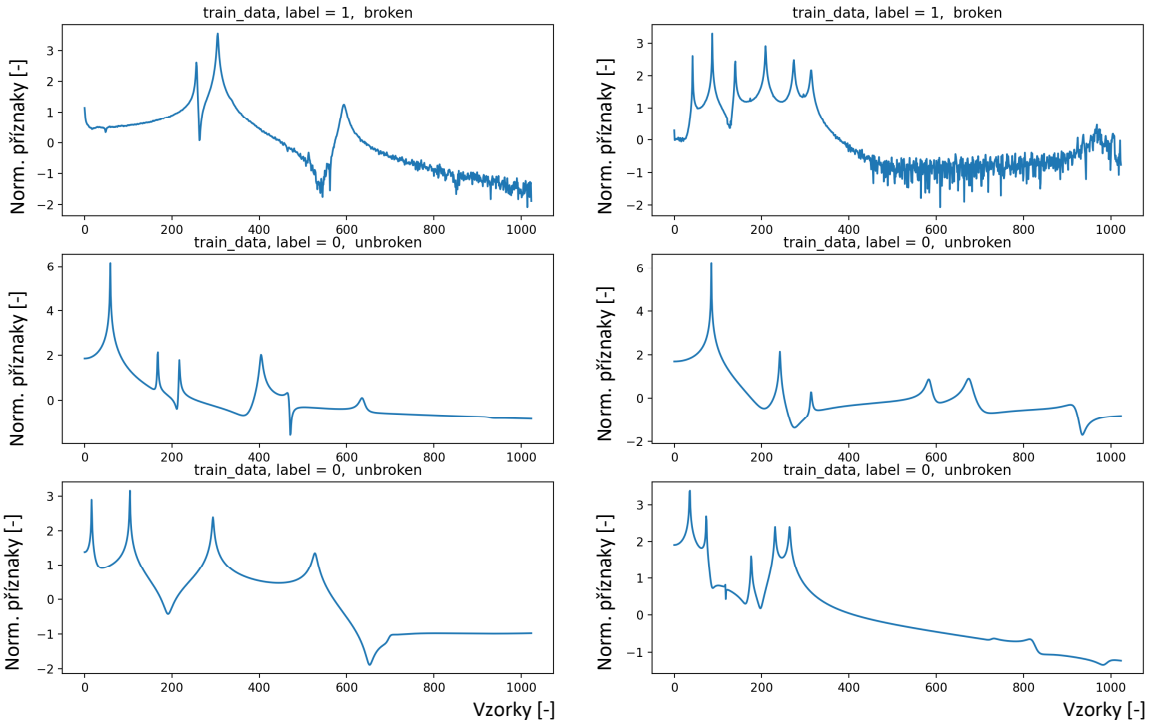
Pro účely stručnosti budou v následujícím textu data odpovídající vzorkům konstrukce s poruchou označována jako broken, data odpovídající vzorkům konstrukce bez poruchy jako unborken a data představující referenční měření budou označována reference.

7.2.1 Vytvoření datového toku – jeden vstup

Datový tok s jedním vstupem slouží k produkování dat pro modely s jedním vstupem. Tímto vstupem je vektor normalizovaných dat dle *(7-2)*. Vytvoření datového toku je provedeno v souboru v příloze 11.1 single_input_model (Python). Nejprve je definována funkce get_label (ř.24), která na základě tenzoru s daty typu string s cestou k trénovacím datům vrátí konstantu 0 pro data ze skupiny unbroken, konstantu 1 pro data ze skupiny broken a konstantu 2 pro data ze skupiny reference. Dále je definována funkce read_files (ř.43), která na základě tenzoru s daty typu string s cestou k trénovacím datům zjistí správný label (0 či 1) pomocí funkce get_label a následně načte data v odpovídajícím csv souboru, složí je do správného tvaru a vrátí odpovídající dvojici sample a label. Je třeba zdůraznit, že vzhledem k tomu, že data vstupující do těchto funkcí nejsou skutečně načtena v operační paměti (jsou využívány v rámci datového toku dataset) nelze tyto funkce sestavit z libovolných operací s csv soubory, funkce musí využívat pouze operace

knihovny TensorFlow (například pro načtení csv tf.io.decode_csv, místo knihovny csv či pandas) což může být zdrojem mnoha problémů.

Po deklaraci funkcí je vytvořen objekt dataset obsahující data typu string s cestami k trénovacím datům (ř.76). Metodou filter za pomoci get_label jsou z datasetu odstraněna data skupiny reference. Metodou map je vyvolána funkce read_files, která místo dat typu string „naplní“ dataset dvojicemi sample a label. Od této chvíle dataset obsahuje skutečná trénovací data, přesto v něm nejsou přítomna, jelikož nejsou načtena v operační paměti. Metoda cache umožní, aby po prvním použití trénovacích dat v první generaci byla data uložena v mezipaměti a v dalších generacích rychleji načtena [41]. Metoda batch nastaví velikost balíků po kterých budou trénovací data zasílána a metoda prefetch umožní, aby byla trénovací data pro další generaci načítána během toho, co probíhá trénování sítě předchozím balíkem dat, všechny tyto metody vedou ke zvýšení rychlosti trénování sítě [41]. Pro kontrolu jsou zobrazena náhodná trénovací data získaná skrz datový tok (Obr. 7.1), jde nyní o vektory představující normovaný tvar příznaků bez fyzikálních vlastností. Uvedené operace jsou aplikovány také na testovací data.

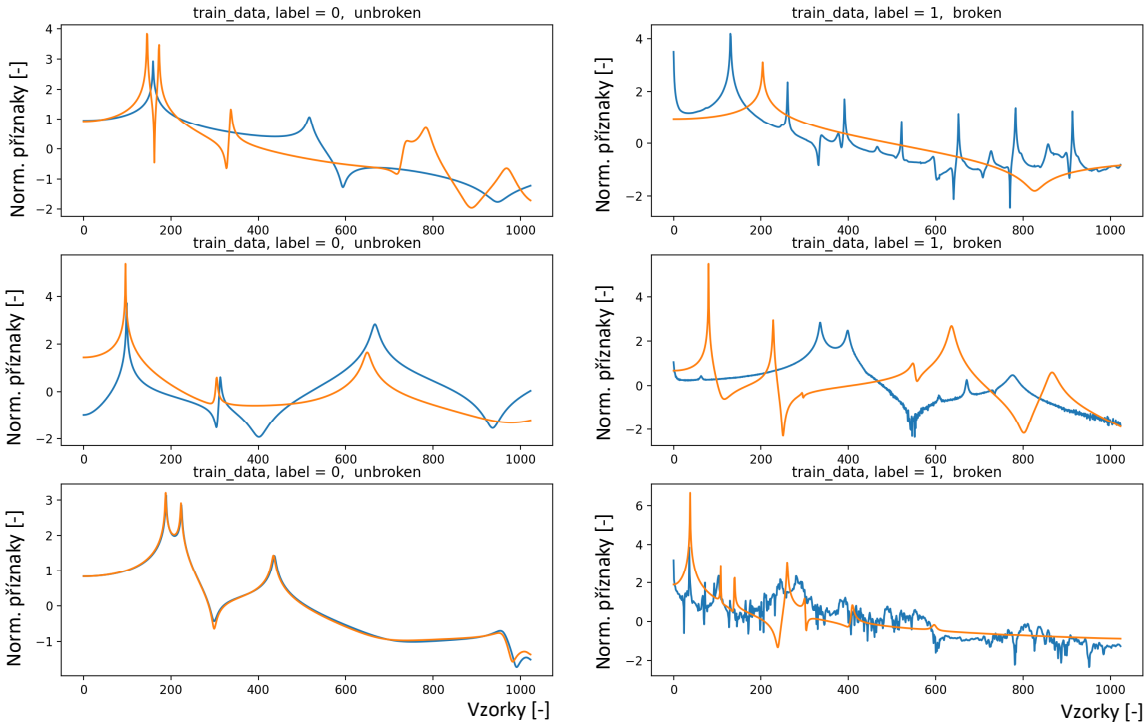


Obr. 7.1 Kontrolní zobrazení bezrozměrných vektorů trénovacích dat, označeny příslušnou hodnotou label

7.2.2 Vytvoření datového toku – dva vstupy

Datový tok se dvěma vstupy používá obdobné metody, jako datový tok s jedním vstupem. Je vytvořen v souboru v příloze 11.2 multiple_input_model (Python). Funkce get_label je definována obdobně jako v předchozím případě. Síť se dvěma vstupy však využívá také data reference, proto je definována funkce get_reference_path (ř.46), která pro tenzor typu string s cestou k datům skupiny broken nebo unbroken, vrací cestu k odpovídajícím datům skupiny reference. Při vstupu broken_34 tak vrátí cestu k souboru reference_34. Funkce read_file (ř.55) provádí čtení csv souboru obdobným způsobem jako v předchozí kapitole. Funkce map_files (ř.78) má za vstup tenzor typu string, s cestou k datům skupiny broken nebo unbroken. Funkce vyhledá odpovídající reference data pomocí uvedených funkcí a jako výstup vrátí list s dvojicí testovaných a referenčních dat a příslušný label. Tvorba datasetu probíhá obdobně, jako v předchozí kapitole

s tím, že pro metodu map je zde využita funkce map_files, která naplní dataset trojicemi: testovaná data, verifikační data a label. Pro kontrolu jsou náhodné trojice zobrazeny na Obr. 7.2.



Obr. 7.2 Kontrolní zobrazení bezrozměrných vektorů trénovacích dat (modře) testovaná a (oranžově) referenční, označeny příslušnou hodnotou label

7.3 Postup trénování a hodnocení výsledků

Po vytvoření datového toku a definici sítě (nastavení hyperparametrů), která bude probrána níže pro jednotlivé typy modelů, je třeba nastavit metody trénování sítě a spustit toto trénování. Trénování sítě je nastaveno pro sítě se dvěma i jedním vstupem obdobně, je tedy provedeno například v příloze 11.1 single_input_model (Python). Nejprve jsou nastaveny callbacky (ř.168), tedy instrukce, které budou prováděny v průběhu trénování sítě. Callback ModelCheckpoint [42] průběžně ukládá nejlepší verzi sítě, tedy uloží parametry pro které síť dosáhla na testovacích datech nejlepší hodnotu accuracy. Dále callback ReduceLROnPlateau [42], který sníží hodnotu learning rate na polovinu, pokud se hodnota ztrátové funkce po dobu pěti následujících generací nesníží, tím může pomoci pokud se minimalizace například „zasekne“ v lokálním minimu. Nakonec callback EarlyStopping [42] zastaví trénování, pokud se ztrátová funkce nesníží po padesát následujících generací. Síť je kompilována metodou compile, kdy je nastaven algoritmus pro optimalizaci, learning rate, definice ztrátové funkce a metriky. Při testovaných úlohách představených níže se nejlépe osvědčil algoritmus keras.optimizers.Adam, což je algoritmus stochastické gradientní optimalizace [43]. Hodnota learning rate (rychlosti učení) může díky callbacku ReduceLROnPlateau být na počátku vysoká (1e-3). Jako ztrátová funkce byla zvolena Binary crossentropy, která byla představena dříve v této práci. Metriky představují vyhodnocení úspěšnosti sítě, na rozdíl od ztrátové funkce mohou být diskrétní. Pro pozdější vyhodnocení jsou zde jako metriky nastaveny hodnoty zadávané do confusion matrix.

Metoda fit (použitá na (ř.190)) spustí trénování sítě se zadaným datovým tokem po zadané množství epoch. Při zadání testovacích dat je možné kontrolovat přeučení v průběhu trénování. Metoda evaluate vyhodnotí výslednou síť na testovacích datech a vytiskne hodnoty metrik. Pro

získání ROC křivek, které byly popsány dříve je však třeba získat výstup sítě před diskretizací na předpovídané klasifikační třídy 0 a 1. To umožní metoda predict, která pro modely vytvářené v této práci bude vracet spojité hodnoty od 0 do 1 jako výstupy aktivační funkce sigmoid. Výstupy si lze představit jako odhad pravděpodobnosti s jakou daný vzorek patří do dané třídy. Zjednodušeně tak pro výstup 0,8 tak síť „tvrdí“, že na 80 % patří vzorek do třídy 1 a na 20 % patří do třídy 0. Pro tvorbu ROC křivky je třeba získat ještě hodnoty label ve správném pořadí (ř.195), následně je tvorba ROC křivky spuštěna vytvořením objektu třídy ROC (ř.223) a spuštěním jejího vykreslení. Předtím jsou ovšem ještě vykresleny grafy průběhu trénování, které budou používány níže.

Třída ROC je definována v souboru v příloze 11.3 statistics_tools (Python) na (ř.29), při vytvoření objektu třídy jsou vstupem hodnoty predikcí (float mezi 0 a 1) a skutečné hodnoty (hodnoty 0 nebo 1). Funkce get_false_positive_rate (ř.5) a get_sensitivity (ř.17) jsou pouze implementací vzorců z Obr. 2.15. Metoda confusion_matrix (ř.43) třídy ROC má vstup treshold. Pro tento treshold, tedy přelom, od kterého rozdělí spojité hodnoty predikcí na diskrétní predikce hodnot 0 a 1, vypočítá hodnoty true positives, true negatives, false positives a false negatives. Je zřejmé, že pro různé hodnoty treshold budou i tyto výstupní hodnoty odlišné. Dále metoda get_ROC (ř.68) vypočítá hodnoty sensitivity a false positive rate pro hodnoty treshold od 0 do 1 pomocí předchozích funkcí a metod. Nakonec metoda plot_ROC (ř.81) ROC křivku vykreslí.

Pro hodnocení výsledků každé metody je vhodné stanovit minimální dosažitelnou úroveň pomocí výrazně snazší metody, či pouhého náhodného odhadování. Při binární klasifikaci bude klasifikátor, produkující náhodná čísla v rozmezí od 0 do 1, mít hodnotu accuracy 50 % a na ROC křivce, jak bylo zmíněno, se vykreslí šikmá přímka. Proto je tato přímka na grafu ROC křivky červeně vykreslena a každá metoda, která má ROC křivku pod touto přímkou či na ní je neúspěšná.

7.4 Dense sítě s jedním vstupem

Vzhledem k tomu, že samotná definice modelu (stanovení hyperparametrů) představuje pouze malou část kódu v souboru, který je jinak pro různé sítě stejný, byly do tištěné přílohy umístěny pouze dva příklady modelů (11.1 single_input_model (Python) a 11.2 multiple_input_model (Python)). Rozložení vrstev jednotlivých sítí, které budou diskutovány níže, je místo toho zaznamenáno na grafech sítí v příloze v kapitole Grafy DNN.

V knihovně Keras jsou modely sítí tvořené tak, že jsou definovány jednotlivé vrstvy (např. dense, conv1D a podobně), jejich vstupem mohou být výstupy přechozích vrstev, takto je možné sestavit jednoduché sítě s po sobě jdoucími vrstvami, i složitější sítě, které se větví. Příkladem může být (ř.143) 11.1 single_input_model (Python).

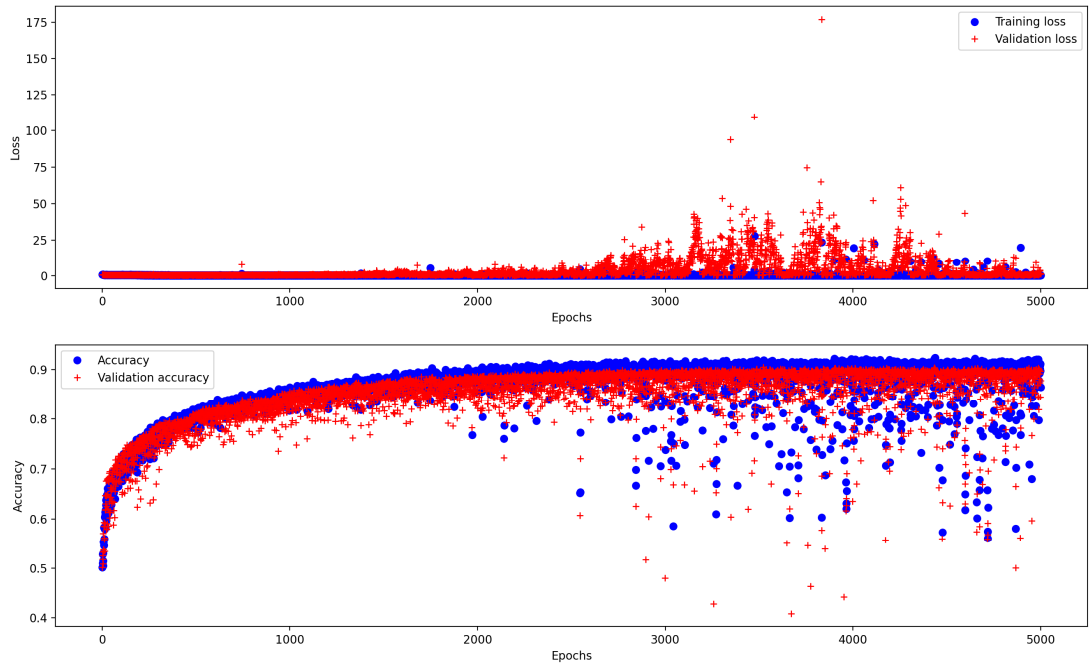
Hustě propojené DNN tvořené dense vrstvami by se nejspíš dali označit za základní volbu při tvorbě DNN, oproti které mohou být ostatní sítě porovnávány. Hustě propojená síť dostatečné hloubky by na vstupních datech představujících tvar záznamu kmitání jednoho bodu ve frekvenční oblasti, mohla najít významné vlastnosti ve vztahu k jejich umístění. Můžeme předpokládat, že se bude zaměřovat především na malé peaky „falešné vlastní frekvence“ pozorované v předchozích kapitolách a jejich vztah k velkým peakům skutečné vlastní frekvence. Je totiž možné, že v jejich vzájemném rozmístění je skrytý určitý vzorec.

7.4.1 Síť s mnoha dense vrstvami

První diskutovanou sítí je síť tvořená 16 vrstvami dense s aktivační funkcí relu a poslední vrstvou dense s aktivací sigmoid. Její graf je možno nalézt v příloze 13.1. Prvních 9 vrstev má vstup a výstup rozměru 1024. Příslušné matice W (kernel) těchto vrstev tak mají rozměr 1024x1024.

Oproti ostatním sítím v této práci má nejvíce vrstev. Rozměr výstupů se postupně snižuje, s nadsázkou lze říci, že síť má tvar trychtýře, který má dostupné informace postupně koncentrovat, odstraňovat nepotřebné informace až dospěje k výsledku ve formě binární klasifikace.

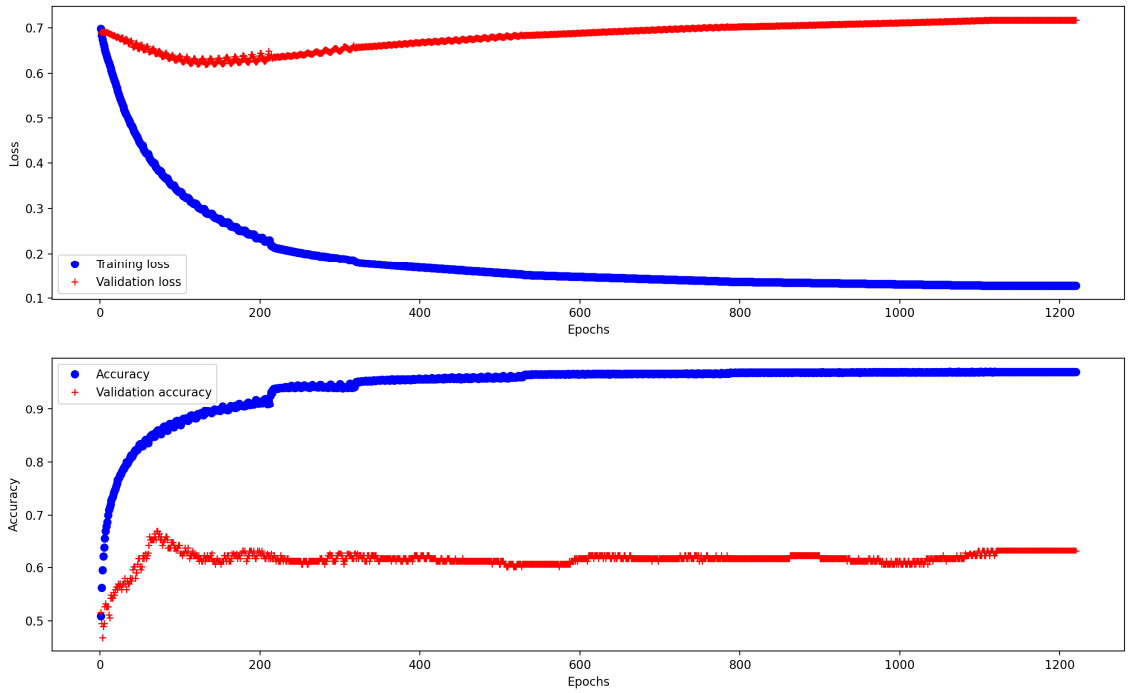
Na Obr. 7.3 je zobrazen graf vývoje loss funkce a metriky accuracy v průběhu trénovacích epoch. Tato data byla vyhodnocena jak pro trénovací data (modře), tak pro testovací/validační data (červeně). Přeučení se projeví tak, že výsledné hodnoty pro testovací data budou výrazně horší jak pro trénovací. Takový jev na uvedeném grafu nenastává, problém spočívá v tom, že jsou testovací data vybírána rozdělením náhodně promíchané množiny na testovací a trénovací. Tento problém byl poté vyřešen tvorbou nezávislých testovacích dat, která byla popsána v kapitole 4.4. Vzhledem k tomu, že i trénovací data dosahovala na této síti metriky accuracy pouze okolo 90 %, což bylo dalšími modely překonáno a vzhledem k časové náročnosti učení této sítě již nebyla síť znovu trénována pro vhodnější testovací data.



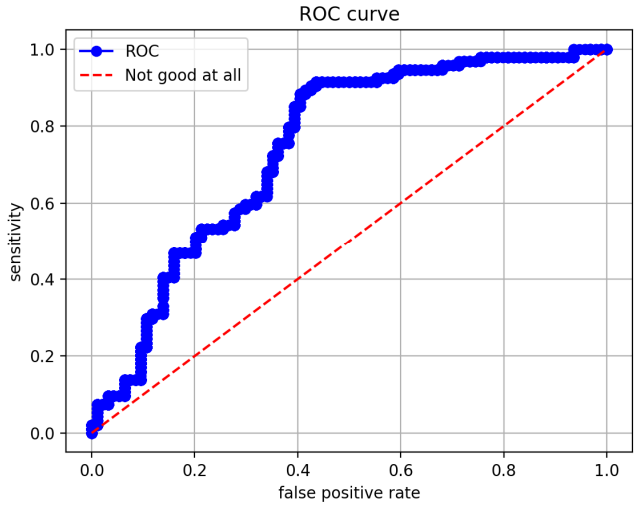
Obr. 7.3 Průběh loss funkce a accuracy metriky sítě s mnoha dense vrstvami, použitá nevhodná testovací data

7.4.2 Síť s vrstvami dense a layer_normalisation

Nejúspěšnější sítí ze skupiny dense sítí s jedním vstupem (kterých bylo vyzkoušeno více, než je zde uvedeno) je síť s 9 dense vrstvami s aktivací relu, 10. dense vrstvou s aktivací sigmoid a normalizačními vrstvami layer_normalization. Graf sítě je uveden v příloze 13.2. Trénovací data jsou v rámci konstrukce příznaků normalizována, což, pro zde uváděné sítě, vedlo k výraznému nárůstu efektivity. Po průchodu vrstvou však již data opět normalizována nejsou. Průběžnou normalizací je možné zvýšit efektivitu sítě [37], k čemuž zde také došlo. Na Obr. 7.4 je možné vidět poměrně rychlý pokles loss funkce, kde jsou viditelné skoky i na metrice accuracy, což jsou zřejmě místa, kdy callback ReduceLROnPlateau snížil hodnotu learning rate a umožnil tak další zlepšení výsledků. Důležitý ovšem je průběh metrik pro testovací data, ten ukazuje, že se síti nedaří dostatečná generalizace. Pravidla, která vytvoří, platí pouze pro trénovací data. Pro tuto síť byla vykreslena také ROC křivka na Obr. 7.5 a vytvořena confusion matrix na Obr. 7.6 dle vzorců v Obr. 2.15.



Obr. 7.4 Průběh loss funkce a accuracy metriky pro síť s vrstvami dense a layer_normalization



Obr. 7.5 ROC křivka pro síť s vrstvami dense a layer_normalization

Total Population	Predicted positive	Predicted negative	
188	51	137	
Actually positive	True positive	False negative	Sensitivity
94	38	56	40,43%
Actual negative	False positive	True negative	False positive rate
94	13	81	13,83%
Accuracy	Precision	False ommision rate	Positive likehood ratio
63,30%	74,51%	40,88%	2,923076923

Obr. 7.6 Confusion matrix pro síť s vrstvami dense a layer_normalization

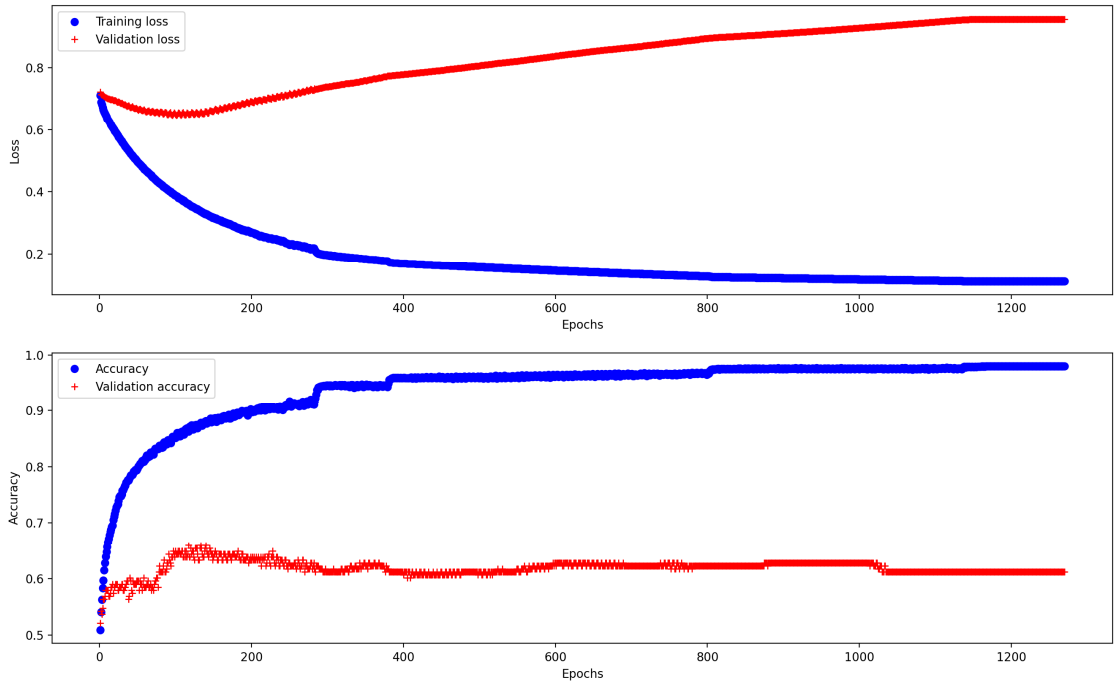
7.5 Dense sítě se dvěma vstupy

Použití dense sítě se dvěma vstupy je motivováno myšlenkou, že pokud bude mít síť možnost porovnat testovaná data s referenčními, bude mít lepší výsledky. Pokud budou testovaná data ze skupiny unbroken, měla by být alespoň podobná s referenčními.

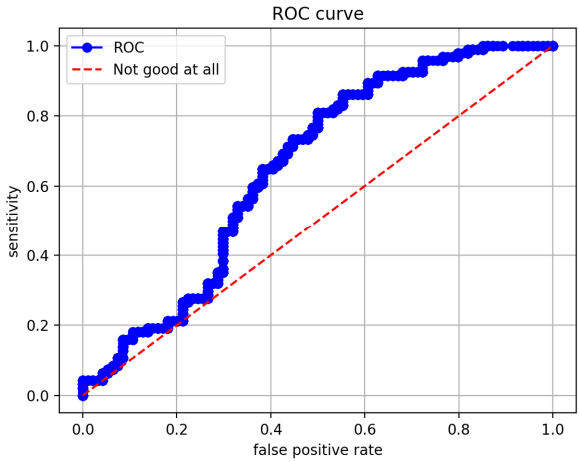
Vstupem do sítě je tenzor tvaru 2x1024, který je následně rozdělen na dva tenzory velikosti 1024, které jsou poté použity jako vstupy do nezávislých vrstev, tyto vrstvy jsou později spojeny. Příklad sítě se dvěma vstupy je na (ř.169) 11.2 multiple_input_model (Python).

7.5.1 Síť s dvěma vstupy, vrstvami dense a layer__normalization

Nejúspěšnější ze skupiny dense sítí s dvěma vstupy, je následující síť, její graf je uveden v příloze 13.3. Po čtyřech vrstvách dense s aktivací relu střídaných vrstvami layer_normalization, které jsou nezávisle aplikované na oba vstupy následuje spojení. Vrstva concatenate napojí oba vstupní vektory za sebe. Následuje dalších 6 dense vrstev, z nichž poslední má aktivaci sigmoid a ostatní relu. Výsledky jsou zobrazeny na Obr. 7.7, Obr. 7.8 a Obr. 7.9. Ani síť se dvěma vstupy nedosahuje dobrých výsledků na testovacích datech, přestože na trénovacích datech je poměrně úspěšná. Vykazuje dokonce ještě horší metriky než síť s jedním vstupem, to může být mimo jiné způsobeno větším množstvím parametrů sítě. U obou uvedených sítí dense, testovaných na vhodných testovacích datech, dochází po drobném zlepšení metrik na počátku k postupnému zhoršování výsledků vlivem přeučení.



Obr. 7.7 Průběh loss funkce a accuracy metriky pro síť s dvěma vstupy, vrstvami dense a layer_normalization



Obr. 7.8 ROC křivka pro síť s dvěma vstupy, vrstvami dense a layer_normalization

Total Population	Predicted positive	Predicted negative	
188	93	95	
Actually positive	True positive	False negative	Sensitivity
94	57	37	60,64 %
Actually negative	False positive	True negative	False positive rate
94	36	58	38,30 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
61,17 %	61,29 %	38,95 %	1,583333333

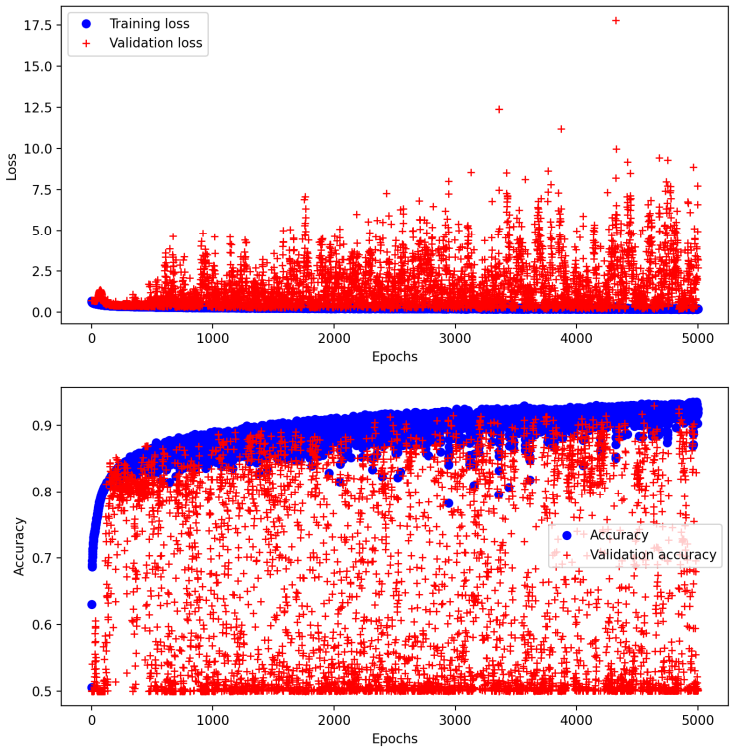
Obr. 7.9 Confusion matrix pro model s více vstupy, vrstvou dense a layer_normalization

7.6 Konvoluční modely single input

Konvoluční sítě jsou schopny prohledávat data postupně a hledat příznaky na různých úrovních jejich velikosti. Jak již bylo uvedeno, 1D konvoluční sítě zaznamenali mnoho úspěchů při zpracování sekvencí dat, proto je možné se domnívat, že konvoluční sítě budou pro danou úlohu úspěšné.

7.6.1 Síť s vrstvou convolution a batch_normalization

Síť s vrstvou batch_normalization byl vytvořen podle [44], kde vykazoval dobré výsledky. Graf sítě je uveden v příloze 13.4. Při testování této sítě byla použita stejná testovací data, jako v případě 7.4.1, tudíž by výsledky pro testovací data měly být velmi dobré. Nicméně tyto výsledky jsou po celou dobu trénování velmi silně zašumělé (Obr. 7.10). Nejde nejspíš přímo o přeučení, jelikož pro ty samá testovací data je síť v následujících epochách jednou velmi úspěšná a podruhé velmi neúspěšná. Důvodem by mohlo být použití vrstvy batch_normalization, která při trénování provádí normalizaci po balících trénovacích dat, což ovšem nemůže provádět při testování, proto se pro testovací data chová výrazně jinak jak pro trénovací. Jelikož síť nevykazovala dobré výsledky, nebyla ani přepočítávána pro vhodnější testovací data.

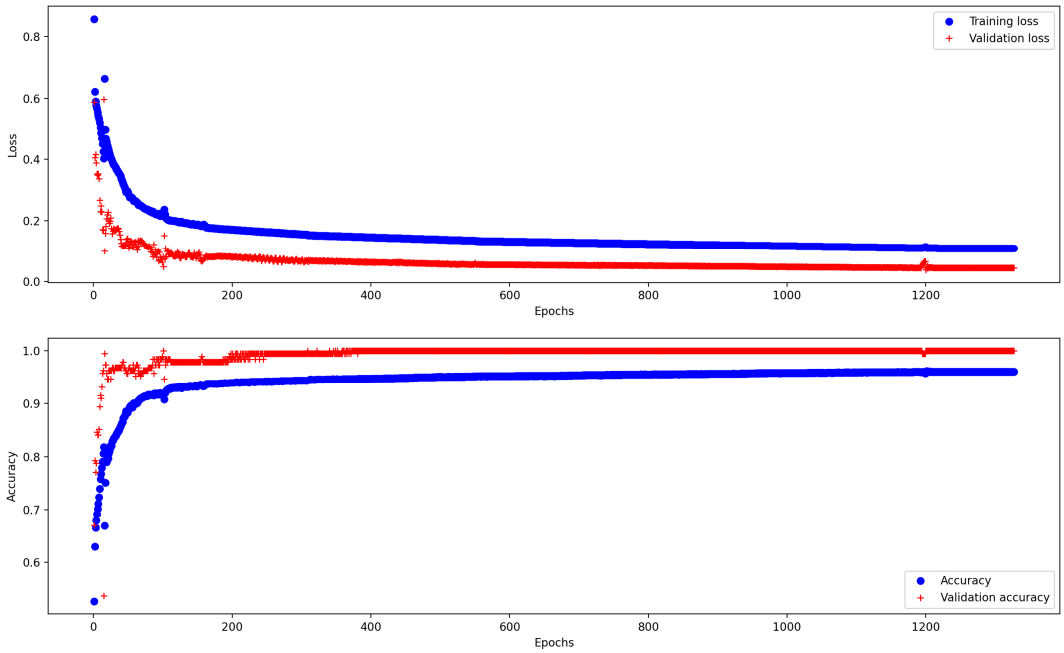


Obr. 7.10 Průběh loss funkce a accuracy metriky pro síť s convolution a batch_normalization, nevhodná testovací data

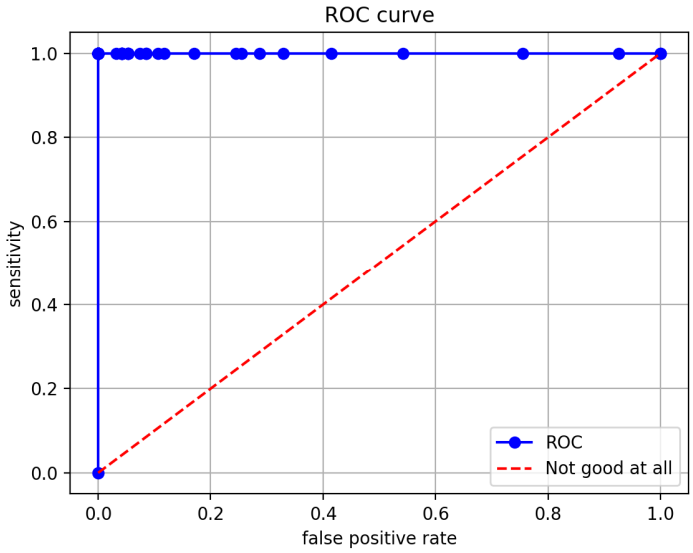
7.6.2 Síť s vrstvou convolution a layer__normalization

Síť tvořený dvěma vrstvami 1D convolution s aktivací relu, dvěma vrstvami max_pooling, dvěma vrstvami layer_normalization, zakončená vrstvou global_max_pooling a dense vrstvou s aktivací sigmoid vykazuje podstatně lepší výsledky než předešlé sítě. Graf je uveden v příloze 13.5. Hloubka filtru konvolučních vrstev je 64 a velikost jádra je 7. Velikost okénka maxpooling je 5. Vstupní data mají dimenzi příslušející kanálům o velikosti 1, první konvoluce tak z dat 1024x1 vytvoří 1024x64, první dimenzi pak zmenšuje vrstva max_pooling, která umožní další konvoluční vrstvě vnímat rozsáhlejší příznaky v datech. Vrstva global_max_pooling potom připraví data pro vstup do dense vrstvy, která transformovaná data vyhodnotí.

Oproti dense sítím je na Obr. 7.11 možné pozorovat rychlé zlepšování na počátku trénování, a především lepší výsledky pro testovací data než pro trénovací. Nakonec síť dosáhne správného roztřídění všech vzorků v testovacích datech. To může být zdrojem podezření, ovšem je třeba zdůraznit, že testovací i trénovací data zde jsou simulovaná na různých modelech. Může dojít k tomu, že jsou příznaky poruchy na testovacích datech výraznější a ve chvíli, kdy síť na testovacích datech naučí vnímat správné příznaky, dochází k úplně přesnému roztřídění dat trénovací množiny. ROC křivka na Obr. 7.12 ukazuje dokonalou klasifikaci, confusion matrix na Obr. 7.13 také. Pro lepší představu o průběhu na trénovacích datech byla vypočtena confusion matrix i pro ně na Obr. 7.14.



Obr. 7.11 Průběh loss funkce a accuracy metriky pro síť s vrstvou convolution a layer_normalization



Obr. 7.12 ROC křivka pro síť s vrstvou convolution a layer_normalization

Total Population	Predicted positive	Predicted negative	
188	94	94	
Actually positive	True positive	False negative	Sensitivity
94	94	0	100,00 %
Actually negative	False positive	True negative	False positive rate
94	0	94	0,00 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
100,00 %	100,00 %	0,00 %	-

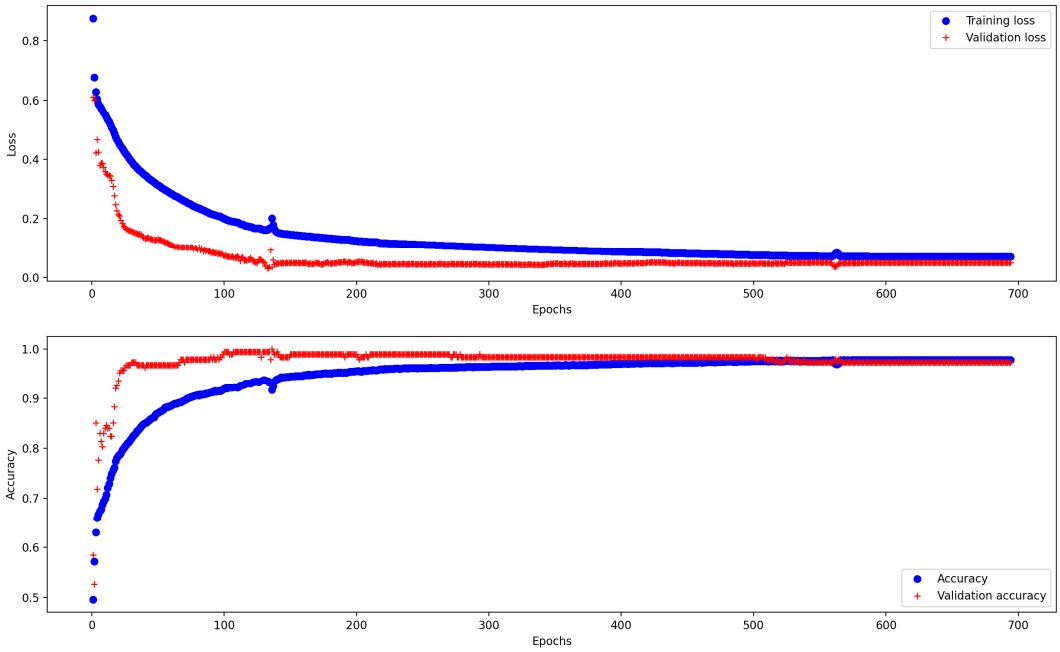
Obr. 7.13 Confusion matrix pro síť s vrstvou convolution a layer_normalization

Total Population	Predicted positive	Predicted negative	
7911	3701	4210	
Actually positive	True positive	False negative	True positive rate
3959	3675	284	92,83 %
Actually negative	False positive	True negative	False positive rate
3952	26	3926	0,66 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
96,08 %	99,30 %	6,75 %	141,0962364

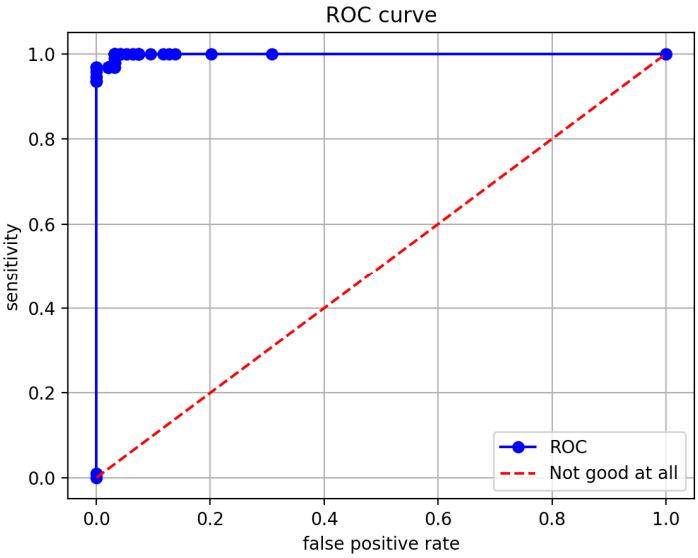
Obr. 7.14 Confusion matrix pro síť s vrstvou convolution a layer_normalization na trénovacích datech

7.6.3 Síť s vrstvou separable_convolution

Další zlepšení výsledků pro trénovací data přináší nahrazení vrstvy 1D convolution vrstvou separable 1D convolution, která byla popsána v kapitole 2.15, a přináší výhodu menšího množství parametrů. Síť je jinak sestavena stejně jako předešlá (graf v příloze 13.6) a její výsledky jsou zobrazeny na Obr. 7.15, Obr. 7.16 a Obr. 7.17. Trénování na rozdíl od předešlých trvalo méně jak 700 epoch. Přestože metriky pro testovací data jsou na počátku opět lepší jak pro trénovací, před koncem se tento poměr obrátí.



Obr. 7.15 Průběh loss funkce a accuracy metriky pro síť s vrstvou separable convolution



Obr. 7.16 ROC křivka pro síť s vrstvou separable convolution

Total Population	Predicted positive	Predicted negative	
188	95	93	
Actually positive	True positive	False negative	Sensitivity
94	92	2	97,87 %
Actually negative	False positive	True negative	False positive rate
94	3	91	3,19 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
97,34 %	96,84 %	2,15 %	30,66666667

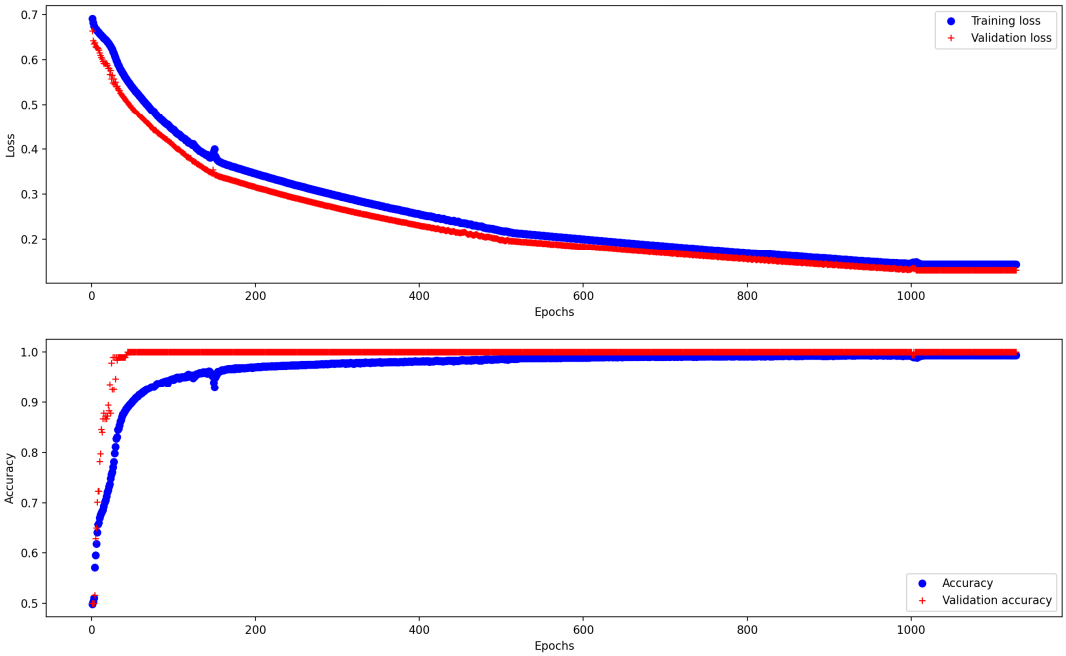
Obr. 7.17 Confusion matrix pro síť s vrstvou separable convolution

7.7 Konvoluční sítě s více vstupy

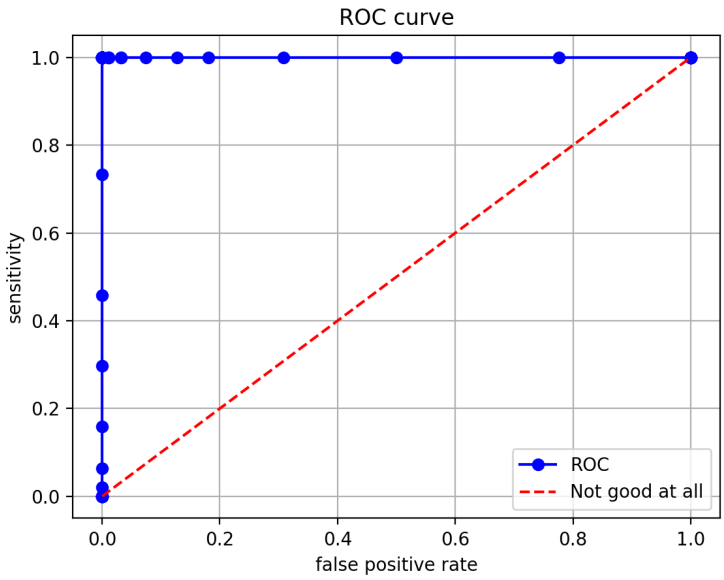
Použití konvolučních sítí s více vstupy je opět motivováno myšlenkou, že splynutí informací o aktuálním stavu konstrukce s informacemi z referenčních dat by mělo přinést zlepšení rozhodování sítě.

7.7.1 Síť se dvěma vstupy a vrstvou separable_convolution

Následující síť přináší celkově nejlepší výsledky ze všech uvedených. Je tvořena v podstatě dvěma sítěmi z předchozí kapitoly, jedna zpracovává referenční data, druhá testovaná data, obě větve jsou nezávislé, dokud nejsou výsledky dense vrstvy s aktivací sigmoid spojeny za sebe do vektoru o velikosti 2, který je opět vyhodnocen dense vrstvou s aktivací sigmoid. Graf sítě je uveden v příloze 13.7. Testovací data dosáhnou velmi rychle hodnoty accuracy 100 %, ovšem trénovací data (Obr. 7.21) se této hodnotě také přibližují. Je tak možné, že prolnutí s referenčními daty přináší zlepšení výsledků.



Obr. 7.18 Průběh loss funkce a accuracy metriky pro síť se dvěma vstupy a vrstvou separable_convolution



Obr. 7.19 ROC křivka pro síť se dvěma vstupy a vrstvou separable_convolution

Total Population	Predicted positive	Predicted negative	
188	94	94	
Actually positive	True positive	False negative	Sensitivity
94	94	0	100,00 %
Actually negative	False positive	True negative	False positive rate
94	0	94	0,00 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
100,00 %	100,00 %	0,00 %	-

Obr. 7.20 Confusion matrix pro síť se dvěma vstupy a vrstvou separable_convolution

Total Population	Predicted positive	Predicted negative	
7911	3959	3952	
Actually positive	True positive	False negative	True positive rate
3975	3942	33	99,17 %
Actually negative	False positive	True negative	False positive rate
3936	17	3919	0,43 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
99,37 %	99,57 %	0,84 %	229,6072808

Obr. 7.21 Confusion matrix pro síť se dvěma vstupy a vrstvou separable_convolution na trénovacích datech

8 Vyhodnocení vlastního řešení problému

Poslední kapitola by měla splnit třetí dílčí cíl této práce – otestování navržených metod na testovacích datech, která jsou zcela odlišná od trénovacích, pro splnění úkolů 2 a 3. Úkolem 2 je míněna schopnost sítě vyhodnotit, zda je v měřené součásti přítomné poškození na základě referenčních dat v nepoškozeném stavu a aktuálních dat, která se mohou mírně lišit v lineárních parametrech. Úkolem 3 je míněna schopnost sítě vyhodnotit, zda je v měřené součásti přítomné poškození pouze na základě měření této jedné součásti.

Výsledky testování jednotlivých metod na testovacích datech byly pro přehlednost uvedeny v příslušných kapitolách. Zde budou tyto výsledky komentovány a vyhodnoceny. Testování metody na simulovaných datech nepřináší příliš přesnou představu o potenciálu metody fungovat na reálných datech, proto v této kapitole budou vyhodnocované metody také testovány pomocí dat získaných v rámci měření pro experimentální modální analýzu.

8.1 Vyhodnocení výsledků DNN

Výsledky testování DNN jednoznačně ukazují, že pro vyhodnocení daných testovacích dat jsou představené CNN výrazně vhodnější než představené sítě s vrstvou dense. Tento závěr ovšem platí pouze pro topologie, které byly otestovány, nelze vyloučit, že je možné sestavit úspěšnější dense síť. 100 % metrika accuracy (získaná v 7.6.2 a 7.7.1) se může zdát podezřelá, ovšem stejná testovací data byla použita pro vyhodnocení dense sítí a zde dobrých výsledků dosaženo nebylo.

CNN se nejspíš zaměřuje na malé peaky diskutované v předchozích kapitolách, pokud jsou u testovacích dat výraznější a častější, je možné, že síť dosáhne na testovacích datech lepších výsledků. Sítě s jedním vstupem nepoužívají referenční data, přesto dosahují podobných výsledků, jako sítě se dvěma vstupy. Lze tedy říci, že na simulovaných datech byly úkoly 2 a 3 splněny s podobnou úspěšností. Je tak možné, že síť se dvěma vstupy příliš nevyužívá možnosti porovnání těchto vstupů. Je možné, že pouze jako síť s jedním vstupem, vyhodnocuje přítomnost peaků ve vyhodnocovaných datech.

8.2 Vyhodnocení výsledků identifikace filtru

Pravidla obsažená v metodě používající identifikaci nelineárního filtru nebyla učena na trénovacích datech, celá množina výsledků simulací tak lze označit za testovací data. Přesto byla pro testování vybrána stejná data, jako v případě vyhodnocení DNN, aby byly výsledky alespoň přibližně srovnatelné. Tato metoda neprováděla binární klasifikaci až na úroveň výsledků, spíše vyprodukovala příznaky, spojité hodnoty, na základě kterých může být klasifikace provedena. Proto byly výsledky vyhodnoceny ROC křivkou. Dat bylo ovšem malé množství a tak průběh ROC křivky není příliš vypovídající.

Z výsledných nelineárních parametrů, na základě kterých probíhá vyhodnocení ovšem lze říci, že metoda nedosahovala příliš dobrých výsledků při testování na datech s větším množstvím vlastních frekvencí. Může to být způsobeno narůstající složitostí filtru a obtížností úlohy identifikace ve chvíli, kdy je k dispozici pouze jeden výstup systému.

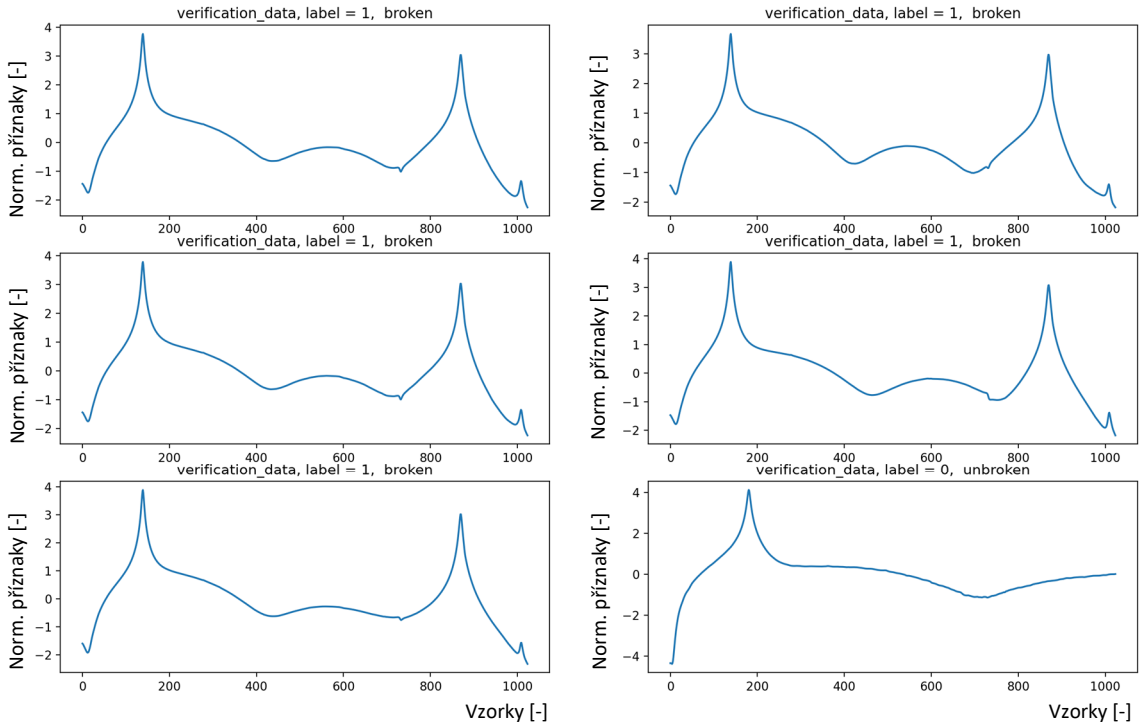
Výsledky identifikace filtru byly porovnáním ROC křivek výrazně horší, jak výsledky získané při použití CNN. Je třeba ovšem říci, že výsledky filtru vznikly bez použití referenčních dat, tedy pro úkol 3. Při použití referenčních dat by výsledky mohli být výrazně lepší.

8.3 Testování na reálných datech z EMA

Výsledky identifikace nelineárního filtru na testovacích datech z EMA se nepodařilo získat. Identifikace parametrů filtru nekonvergovala k minimu odchylky. To může být způsobeno šumem v datech. Filtr je identifikován na datech v časové oblasti, jejich přímé zprůměrování by vedlo k nepřesnostem, jelikož budící impakty pro jednotlivá měření nejsou stejné, tudíž jsou použita nezprůměrovaná data.

Naměřená data pomocí EMA byla získána pomocí více měření stejné konstrukce s poruchou a bez ní. Proto vzniká testovací skupina, která není příliš rozmanitá. Jinak řečeno, měření spadající do skupiny broken jsou si navzájem velmi podobná, stejně tak měření ve skupině unbroken.

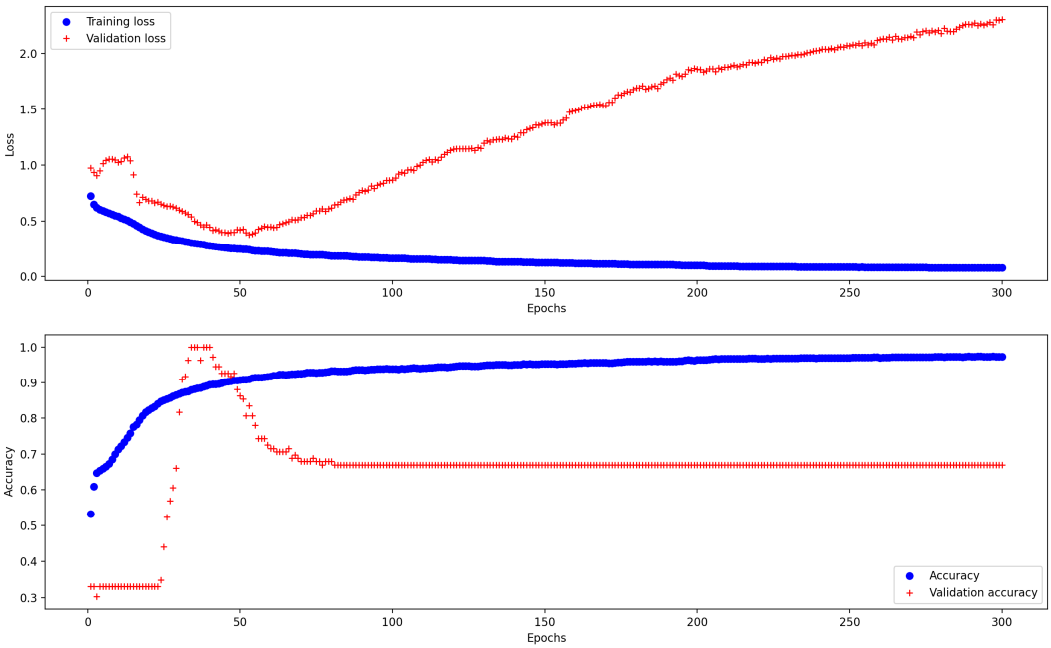
Testována byla síť s jedním vstupem, vrstvami convolution a layer normalization (kapitola 7.6.2). Testování proběhlo na datech zobrazených na Obr. 5.23. Výsledky byly velmi nepříznivé, což bylo možné očekávat, vzhledem ke značnému vlivu šumu ve vyšších frekvencích. Proto byla data oříznuta na frekvenci 170 Hz, náhodný výběr z použitých testovacích dat je zobrazen na Obr. 8.1.



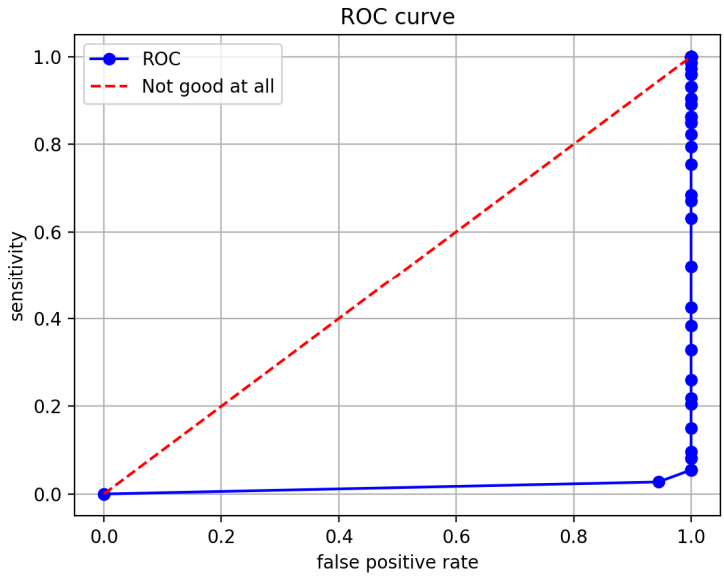
Obr. 8.1 Kontrolní zobrazení bezrozměrných vektorů testovacích dat z EMA, označeny příslušnou hodnotou label

Výsledky testování během učení jsou zobrazeny na Obr. 8.2. Průběh metriky accuracy pro testovací data je zajímavý, nejprve dojde k rychlému skoku na 100 % úspěšnost a následuje opět poměrně rychlý pokles. To je způsobeno již zmiňovanou malou rozmanitostí trénovacích dat, jinak řečeno, pokud síť správně vyhodnotí jeden vzorek, je pravděpodobné, že bude úspěšná i u ostatních, jelikož jsou podobné. Je třeba upozornit, že testovací skupina neobsahuje stejné množství vzorků skupiny broken a unbroken, proto metrika accuracy může být zavádějící. ROC křivka na Obr. 8.3 umožní lepší představu o kvalitě výsledné klasifikace. Hodnoceno dle ROC bylo dosaženo horšího výsledku, jak při náhodné klasifikaci. Confusion matrix na Obr. 8.4 ukazuje proč, síť vyhodnotila všechny vzorky jako pozitivní, tedy s poruchou. Z průběhu metriky accuracy tak lze odhadovat, že se síť nejprve naučila vnímat větší vzory jako větší peaky „falešné vlastní frekvence“, což vedlo ke správné klasifikaci testovacích dat. Následně ale síť dokázala rozpoznat i

menší vzory jako menší peaky „falešné vlastní frekvence“ v trénovacích datech. Ty se ovšem již podobají vlivům šumu v testovacích datech z EMA (malé zoubky na křivce). Proto síť nejspíš vyhodnotila všechna testovací data jako pozitivní.



Obr. 8.2 Výsledky pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení

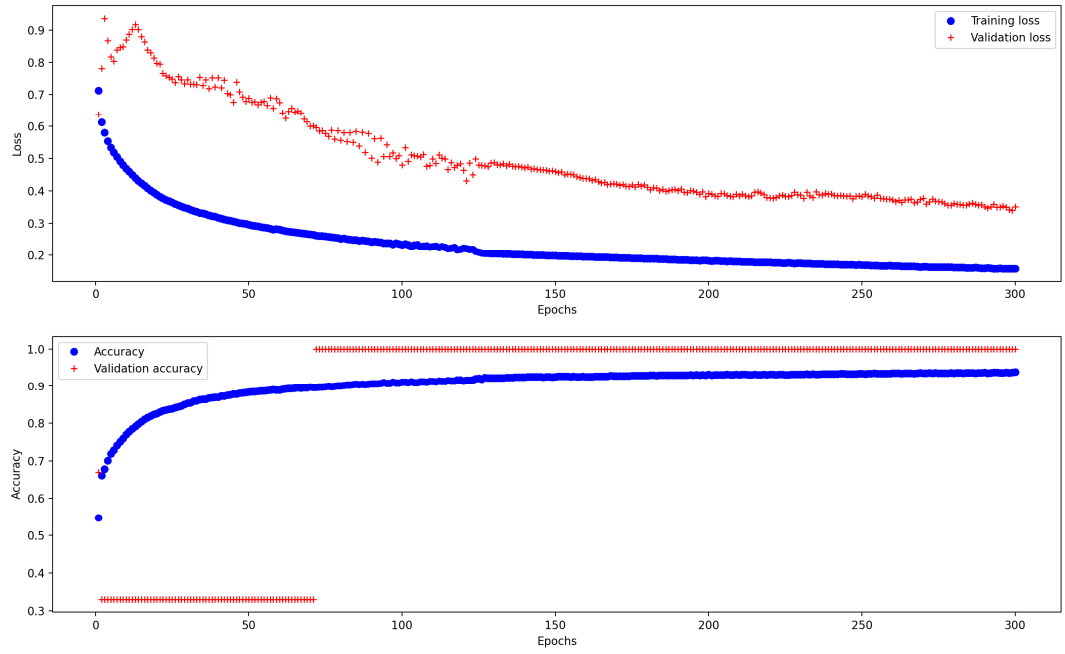


Obr. 8.3 ROC křivka pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení

Total Population	Predicted positive	Predicted negative	
109	109	0	
Actually positive	True positive	False negative	True positive rate
73	73	0	100,00 %
Actually negative	False positive	True negative	False positive rate
36	36	0	100,00 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
66,97 %	66,97 %	-	1

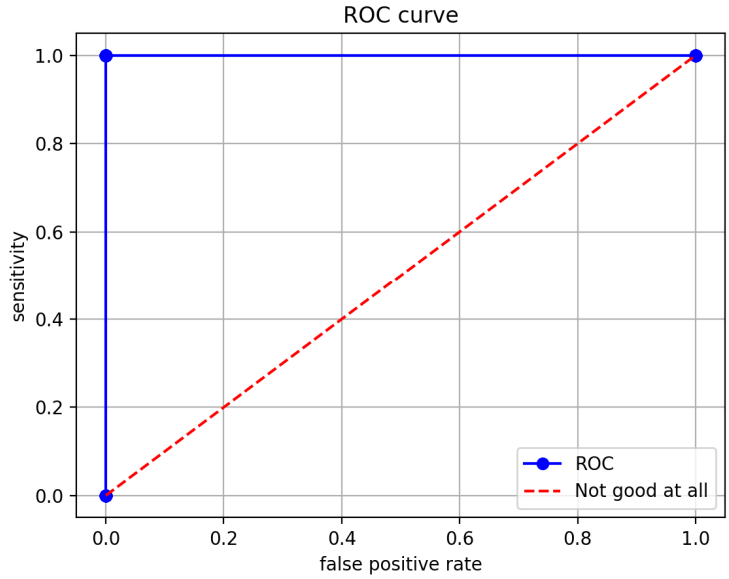
Obr. 8.4 Confusion matrix pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení

Při zpracování dat pomocí DNN je ovšem běžné data upravovat tak, aby byla pro DNN lépe zpracovatelná. Pokud jsou problémem malé zoubky na křivce vstupních dat, je možné provést jejich vyhlazení za předpokladu, že se stejná úprava aplikuje na všechna data z obou tříd. Vytvoření vstupních dat z měření EMA a jejich vyhlazení bylo provedeno v souboru v příloze 12.2 EMA_for_AI (Matlab). K vyhlazení byl použit Gaussův filtr s okénkem délky 10. Data byla poté poskytnuta jako testovací pro stejnou síť jako v předchozím případě. Průběh učení je zobrazen na Obr. 8.5. Metrika accuracy v určitý moment přeskočí na hodnotu 100 % a už zde zůstává. Síť tak opět dosahuje lepších metrik pro testovací, jak pro trénovací data. Zajímavý je ale také průběh loss function, ta je pro testovací data stále výrazně vyšší jak pro trénovací, na rozdíl od accuracy. Loss function v jistém smyslu odpovídá tomu, jak moc jsou od sebe vzdálené spojité výstupy sítě (odhady). Zdá se tedy, že ve smyslu oddělení tříd broken a unbroken od sebe, síť na testovacích datech nedosahuje lepších výsledků jak pro trénovací. Ovšem malý rozptyl uvnitř třídy broken a unbroken pro testovací data umožní úspěšnější binární klasifikaci při použití diskrétní metriky.



Obr. 8.5 Výsledky pro síť s vrstvou separable convolution na reálných datech z EMA vyhlazených oknem délky 10

ROC křivka (Obr. 8.6) a confusion matrix (Obr. 8.7) už jen potvrzují předchozí výsledky.



Obr. 8.6 ROC křivka pro síť s vrstvou separable convolution na reálných datech z EMA vyhlazených oknem délky 10

Total Population	Predicted positive	Predicted negative	
109	73	36	
Actually positive	True positive	False negative	True positive rate
73	73	0	100,00 %
Actually negative	False positive	True negative	False positive rate
36	0	36	0,00 %
Accuracy	Precision	False ommision rate	Positive likehood ratio
100,00 %	100,00 %	0,00 %	-

Obr. 8.7 Confusion matrix pro síť s vrstvou separable convolution na reálných datech z EMA bez vyhlazení

Stanovením dalších kroků konstrukce příznaků (oříznutí na frekvenci 170 Hz a vyhlazení), které byly stanoveny na základě dat naměřených při EMA se ovšem z těchto dat v jistém smyslu stávají trénovací data. Jinak řečeno, bylo by vhodné ověřit i na jiných naměřených datech, zda taková úprava vede k úspěšné klasifikaci. Další data ovšem nejsou momentálně k dispozici, tudíž věrohodnější otestování bude třeba provést v budoucnu.

Závěr

Cílem této práce bylo vytvoření metody, která by na základě měření vibrací dokázala zjistit, zda je v měřené konstrukci přítomna porucha. Byla provedena rešerše předchozích řešení tohoto problému a nejčastějších druhů poruchy konstrukce. Dále byla vybrána jedna z těchto poruch, pro kterou byl vytvořen mechanický model jejího chování. Následně byl vytvořen parametrický model konstrukce s poruchou a byly provedeny simulace. K ověření správnosti mechanického modelu poruchy byla provedena experimentální modální analýza impaktním buzením nosníku, na kterém byla napodobena porucha. Výsledky simulace a experimentu byly poté porovnány. Dále byla vytvořena metoda detekce poruchy v naměřených datech pomocí identifikace nelineárního filtru s modelem poruchy. Tento filtr byl následně testován na simulovaných datech. Z parametrizovaných modelů různých konstrukcí s poruchou byla simulací byla vytvořena sada trénovacích dat, která byla použita k trénování různých neuronových sítí s cílem naučit je provádět klasifikaci těchto dat na skupiny s poruchou a bez poruchy. Tyto neuronové sítě byly následně otestovány také na datech naměřených při experimentální modální analýze. Všechny cíle zadání byly splněny.

Rešerše ukázala, že velká část dosavadních prací simuluje poruchu poklesem tuhosti v určitém místě. Taková porucha způsobí pouze lineární změny v konstrukci, metoda detekce poruchy se tak musí zaměřit na změny lineárních vlastností konstrukce. Přitom lineární změnu může způsobit i například rozdílná teplota či variace rozměrů u různých výrobků. Tato práce se proto zaměřila na poruchu, která se projevuje skokovými změnami tuhosti. Takové projevy může mít například prasklina, či uvolněný šroub. Porouchaná a neporouchaná konstrukce se tak neliší pouze lineárními parametry, ale také tím, že porouchaná konstrukce je do určité míry nelineární.

Simulační model poruchy v konstrukci byl založen na pružině pracující pouze v tlaku. Aby bylo ověřeno, zda je simulační model vhodný, bylo provedeno měření pro experimentální modální analýzu na nosníku. Tento nosník měl napodobenou poruchu ve svěrném spoji, kde docházelo k odléhání nosníku od upevňovací desky. V simulaci i experimentu byly měřeny rychlosti vibrace jednoho z bodů na konstrukci. Na frekvenčních přenosových funkcích získaných experimentem i simulací byly nalezeny malé peaky, které v případě měření bez poruchy nebyly přítomny. Z toho bylo vyvozeno, že tyto peaky „falešných vlastních frekvencí“ jsou příznakem poruchy v konstrukci.

První z metod vytvořených pro detekci poruchy v konstrukci byla metoda založená na identifikaci parametrů nelineárního filtru. Parametry tohoto filtru byly optimalizovány tak, aby filtr co nejlépe aproximoval naměřené výstupy mechanické soustavy s poruchou či bez poruchy. Nelinearita filtru spočívala ve skokových změnách parametrů odpovídajících tuhosti, ty se spínali na základě vnitřních stavů filtru. Intenzita těchto změn byla vyjádřena parametry nelinearity, v případě nulových parametrů byl filtr lineární. Optimalizace pro identifikaci probíhala ve dvou cyklech, vnitřní cyklus optimalizoval lineární parametry pomocí Levenberg-Marquardtova algoritmu a vnější cyklus optimalizoval parametry nelinearity pomocí genetického algoritmu. Identifikace parametrů proběhla na simulovaných testovacích datech. Pro data, kde se vyskytovali pouze dvě dominantní vlastní frekvence dosáhla metoda velmi dobrých výsledků. Pro lineární data byly identifikované parametry nelinearity téměř nulové. Pro nelineární data se parametry jednoznačně vzdalovali od nuly. Pro kmitání s více vlastními frekvencemi již výsledky nebyly přesvědčivé, identifikované parametry nelinearity byly velmi blízké.

Druhou metodou, která byla vyvinuta pro detekci poruchy, bylo použití hlubokých neuronových sítí pro binární klasifikaci. Sítě byly trénovány na simulovaných datech rozdělených do dvou tříd: s poruchou a bez poruchy. Byly vytvořeny sítě s jedním vstupem – měřená data a sítě se dvěma

vstupy – měřená data a referenční data. Referenční data představují měření na obdobné konstrukci bez nelinearity a byla záměrně modelována jako do určité míry odlišná od vyhodnocovaných. Tím je simulován stav, kdy jsou referenční data získána například na konstrukci s drobně odlišnými rozměry. Otestovány byly hustě propojené sítě a konvoluční sítě. Zatímco u hustě propojených sítí se silně projevovalo přeučení, konvoluční sítě měli na testovacích datech velmi dobré výsledky. Sítě využívající referenční data neměli výrazně lepší výsledky jak sítě bez nich.

Nejlepší nalezená síť s jedním vstupem (síť s vrstvami separable convolution) byla následně testována na reálných datech získaných při experimentální modální analýze. Síť trénovaná čistě na simulovaných datech byla tedy testována na čistě experimentálních datech. Síť nejprve dosáhla velmi špatných výsledků, kdy klasifikovala všechny vstupní vzorky jako data s poruchou. Pro upravená data, která byla mírně vyhlazena, potom síť dosáhla dokonalé klasifikace. Je ovšem nutné uvést, že pro testování byla použita data měřená pouze na jedné konstrukci a úprava dat před zpracováním (jako vyhlazování a oříznutí) jim byla přizpůsobena.

Je možné, že problémy metody identifikace filtru pro data s více vlastními frekvencemi jsou způsobeny mísením příliš mnoha módů v datech pro identifikaci. Možným řešením by do budoucna mohlo být naměření dat s nižší vzorkovací frekvencí za použití antialiasingového filtru. Takovým způsobem by mohl být omezen počet přítomných vlastních frekvencí v datech. Pokud by vlivy poruchy zasahovali i do nižších frekvencí, mohlo by potom dojít k úspěšné identifikaci. Výhodou metody využívající identifikaci parametrů je, že v případě špatné identifikace nedojde k chybné klasifikaci. Místo toho, na základě velké odchylky identifikovaných dat od naměřených, může dojít k rozhodnutí, že identifikace se nezdařila.

Důležité pro zhodnocení výsledků hlubokých neuronových sítí na experimentálních datech je, že žádná měřená konstrukce v reálném světě není naprosto lineární a naprosto bez poruchy. V jistém smyslu tak síť klasifikovala všechna nevyhlazená data správně jako data s určitou mírou nelinearity. Abychom mohli provést úspěšnou detekci poruchy, je třeba stanovit míru porušení, od které již bude konstrukce klasifikována jako poškozená. Takovou míru mohou představovat referenční data, pokud budou naměřena na podobné konstrukci, může síť klasifikovat měřenou konstrukci jako porušenou pouze v případě, kdy bude vykazovat větší míru poruchy než je vykazují referenční data.

Další vývoj v oblasti detekce poruchy pomocí hlubokých neuronových sítí by se měl nejspíš zaměřit na obstarání většího množství experimentálně získaných dat na různých konstrukcích. Tato data by se potom používala pouze k testování. Mezi simulovanými trénovacími daty byl velký rozdíl z hlediska intenzity projevu poruchy. Parametry simulačních modelů by proto měli být naladěny tak, aby byly získány trénovací data s podobnou intenzitou projevu poruchy, jako mají testovací data.

Všechna naměřená data budou do určité, byť drobné, míry obsahovat šum. Proto by se tento šum buď měl zavést i v trénovacích datech, nebo by měla být vytvořena univerzální metoda pro jeho potlačení v datech vstupních (testovacích). Nabízí se vyhlazení dat, to ale nesmí vyhladit projevy nelinearit. Jednou z možností, jak toho dosáhnout by mohlo být vytvoření adaptivního vyhlazování na základě směrodatné odchylky v datech. V místech s vysokou jistotou, že je průběh správný, by bylo vyhlazení nízké. Data v místě s velkým rozptylem napříč měřeními by naopak byla vyhlazena velmi silně.

Sítě používající referenční data nedosahovali o mnoho lepších výsledků než sítě s jedním vstupem. Zdá se, že odlišnost referenčních dat od vyhodnocovaných byla příliš velká. Opět by tak bylo vhodné využít experimentálně získaná data pro určení, do jaké míry se mají referenční data lišit

od vyhodnocovaných. Pokud by se lišily jen v rozumně malé míře, umožnilo by to sítím se dvěma vstupy lépe využít informace, které jsou v nich obsažené.

Nejdůležitějším výsledkem této práce je nejspíš určení projevů zkoumaného druhu poruchy na data ve frekvenční oblasti. Drobné peaky, které nepředstavují vlastní frekvence systému bývají nejspíš často označovány za projevy šumu. Přitom byly získány jak z experimentálních, tak ze simulovaných dat, kdy v simulaci žádný šum zaveden nebyl. Navíc z průběhu směrodatné odchylky naměřených dat je zřejmé, že tyto projevy přetrvávají napříč všemi měřeními. Při zpracování dat pomocí metod vytvořených pro lineární systémy, může být snadné zapomenout, že významné informace o systému z reálného světa se mohou skrývat v oblasti projevů nelinearit, které bývají často zanedbávány.

Obě autorem vyvinuté metody detekce poruchy konstrukce pracují na principu, kdy není nutné znát referenční měření té samé konstrukce v nepoškozeném stavu. Prezentované výsledky byly získány ve stavu, kdy bylo metodě buď poskytnuto referenční měření konstrukce stejného tvaru s podobnými rozměry a podobnými materiálovými vlastnostmi, nebo nebyly poskytnuty vůbec žádná referenční data. Přesto byly metody schopné na určité úrovni provádět detekci poruchy konstrukce. Při dalším vývoji tak mohou obě tyto metody být dovedeny do stavu, kdy budou dobře využitelné v praxi. Jedním z příkladů potenciální aplikace může být výroba svařovaných rámů, kde by stačilo na právě vyrobený svařenec poklepat a z naměřených dat by bylo možné zjistit, zda rám neskrývá prasklinu. Další aplikací může být mostní konstrukce, kde by se metodě mohlo podařit detekovat uvolněný šroub. Nakonec lze uvést příklad kompozitů, kde by bylo možné detekovat delaminaci ihned po jeho výrobě za použití referenčních dat naměřených na jiném kusu stejného výrobku.

Použité zdroje

[1] AVCI, Onur, Osama ABDELJABER, Serkan KIRANYAZ, Mohammed HUSSEIN, Moncef GABBOUJ a Daniel J. INMAN. A review of vibration-based damage detection in civil structures: From traditional methods to Machine Learning and Deep Learning applications [online]. *Mechanical Systems and Signal Processing.* 2021, **147**, 107077. Dostupné z: 10.1016/j.ymssp.2020.107077.

[2] SAMUEL, A. L. Some Studies in Machine Learning Using the Game of Checkers [online]. *IBM Journal of Research and Development.* 1959, **3**(3), 210-229. Dostupné z: 10.1147/rd.33.0210.

[3] YANG DANG. *Investigations Into Steel Structure Failures Part I: Failure Mechanisms* [online]. 14. července 2022 [cit. 14. července 2022]. Dostupné z: https://www.hawkins.biz/insights/insight/investigations-into-steel-structure-failures-part-i-failure-mechanisms.

[4] WANG, Z., R. M. LIN a M. K. LIM. Structural damage detection using measured FRF data [online]. *Computer Methods in Applied Mechanics and Engineering.* 1997, **147**(1-2), 187-197. Dostupné z: 10.1016/S0045-7825(97)00013-3.

[5] NOVOTNÝ, Jiljí. *Technologie I. (slévání, tváření, svařování a povrchové úpravy).* Vyd. 2. Praha : České vysoké učení technické, 2001. 8001023516.

[6] YIN, T., H. F. LAM, H. M. CHOW a H. P. ZHU. Dynamic reduction-based structural damage detection of transmission tower utilizing ambient vibration data [online]. *Engineering Structures.* 2009, **31**(9), 2009-2019. Dostupné z: 10.1016/j.engstruct.2009.03.004.

[7] STEINBAUER, Pavel, Zdenek NEUSSER, Ivo BUKOVSKY a Milos NERUDA. Lighting Pole Health Monitoring for Predictive Maintenance [online]. *Procedia Structural Integrity.* 2019, **17**, 799-805. Dostupné z: 10.1016/j.prostr.2019.08.106.

[8] ABDELJABER, Osama, Onur AVCI, Mustafa Serkan KIRANYAZ, Boualem BOASHASH, Henry SODANO a Daniel J. INMAN. 1-D CNNs for structural damage detection: Verification on a structural health monitoring benchmark data [online]. *Neurocomputing.* 2018, **275**, 1308-1317. Dostupné z: 10.1016/j.neucom.2017.09.069.

[9] GUL, Mustafa a F. Necati CATBAS. Damage Assessment with Ambient Vibration Data Using a Novel Time Series Analysis Methodology [online]. *Journal of Structural Engineering.* 2011, **137**(12), 1518-1526. Dostupné z: 10.1061/(ASCE)ST.1943-541X.0000366.

[10] ZANG, C. a M. IMREGUN. STRUCTURAL DAMAGE DETECTION USING ARTIFICIAL NEURAL NETWORKS AND MEASURED FRF DATA REDUCED VIA PRINCIPAL COMPONENT PROJECTION [online]. *Journal of Sound and Vibration.* 2001, **242**(5), 813-827. Dostupné z: 10.1006/jsvi.2000.3390.

[11] GARCÍA, David a Irina TRENDAFILOVA. A multivariate data analysis approach towards vibration analysis and vibration-based damage assessment [online]. *Journal of Sound and Vibration.* 2014, **333**(25), 7036-7050. Dostupné z: 10.1016/j.jsv.2014.08.014.

[12] SANTOS, Adam, Eloi FIGUEIREDO, M.F.M. SILVA, C. S. SALES a J.C.W.A. COSTA. Machine learning algorithms for damage detection: Kernel-based approaches [online]. *Journal of Sound and Vibration.* 2016, **363**, 584-599. Dostupné z: 10.1016/j.jsv.2015.11.008.

[13] DANG, Viet-Hung, Tien-Chuong VU, Ba-Duan NGUYEN, Quang-Huy NGUYEN a Tien-Dung NGUYEN. Structural damage detection framework based on graph convolutional network

directly using vibration data [online]. *Structures.* 2022, **38**, 40-51. Dostupné z: 10.1016/j.istruc.2022.01.066.

[14] ŠPANIEL, Miroslav a Zdeněk HORÁK. *Úvod do metody konečných prvků.* 1. vyd. V Praze : České vysoké učení technické, 2011. 978-80-01-04665-4.

[15] THOMSON, William T. *Theory of vibration with applications.* 4th ed. London [etc.] : Chapman & Hall, 1997. 978-0-412-78390-6.

[16] THE SCIPY COMMUNITY. *scipy.integrate.solve_ivp* [online]. *SciPy v1.8.1 Manual.* 20. května 2022 [cit. 16. července 2022]. Dostupné z: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html.

[17] HAIRER, E. a Gerhard WANNER. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.* Berlin, Heidelberg : Springer Berlin Heidelberg, 1991. Springer series in computational mathematics. 14. 978-3-662-09949-0.

[18] HAIRER, Ernst, Syvert Paul NØRSETT a Gerhard WANNER. *Solving Ordinary Differential Equations I. Nonstiff Problems.* Cham : Springer International Publishing, 20. Springer Series in Computational Mathematics Ser. v.8. 9783662126073.

[19] ALBERDI CELAYA, Elisabete a Juan JOSE ANZA. BDF-α : A Multistep Method with Numerical Damping Control [online]. *Universal Journal of Computational Mathematics.* 2013, **1**(3), 96-108 [cit. 7. října 2022]. Dostupné z: 10.13189/ujcmj.2013.010305.

[20] DINIZ, P.S.R., E.A.B. DA SILVA a S. L. NETTO. *Digital Signal Processing: System Analysis and Design :* Cambridge University Press, 2002. 9781139433501.

[21] ŠIKA ZBYNEK, ZAVREL JAN a VALASEK MICHAEL. Residual Modes for Structure Reduction and Efficient Coupling of Substructures [online]. *Bulletin of Applied Mechanics.* 2009, **5**(19), 54-59. Dostupné z: https://www.researchgate.net/publication/47402296_Residual_Modes_for_Structure_Reduction_and_Efficient_Coupling_of_Substructures.

[22] BOLEK, Alfred a Josef KOCHMAN. *Části strojů.* 5., přeprac. vyd. (v SNTL 1. vyd.). Praha : SNTL, 1989. Technický průvodce. Sv. 6. 80-03-00046-7.

[23] MILÁČEK, Stanislav. *Modální analýza mechanických kmitů.* 2. vyd. Praha, CZ : Vydavatelství ČVUT, 2001. ISBN: 80-010-2333-8.

[24] HEWLETT PACKARD CO. *The fundamentals of modal testing: application note 243-3* [online]. HP memory project,1986 [cit. 5. ledna 2020]. Dostupné z: http://hpmemoryproject.org/an/pdf/an_243-3.pdf.

[25] BILOŠOVÁ, Alena. *Týmová cvičení předmětu Experimentální modální analýza. návody do cvičení předmětu "Experimentální modální analýza".* 1. vyd. Ostrava : Vysoká škola báňská - Technická univerzita Ostrava, Fakulta strojní, 2011. ISBN 978-80-248-2756-8.

[26] STEINBAUER, Pavel. Mechatronic modal hammer. *Computational mechanics 2018 book of extended abstracts: 34th conference with international participation.* Plzeň, CZ : Západočeská univerzita v Plzni, 2018, s. 115-116.

[27] MILÁČEK, Stanislav. *Náhodné a chaotické jevy v mechanice.* Vyd. 1. Praha : České vysoké učení technické, 2000. 80-01-02170-X.

[28] BC. MATÚŠ JAKUŠ. *APLIKACE ALGORITMŮ PRO ODHAD MODÁLNÍCH PARAMETRŮ PŘI EXPERIMENTÁLNÍ MODÁLNÍ ANALÝZE A STUDIE JEJICH CITLIVOSTI NA OKRAJOVÉ PODMÍNKY.* Diplomová práce. BRNO, 2016.

[29] THE MATHWORKS, INC. *Modal parameters from frequency-response functions - MATLAB modalfit* [online]. 13. července 2022 [cit. 13. července 2022]. Dostupné z: https://uk.mathworks.com/help/signal/ref/modalfit.html#bvogfwv_head.

[30] ARDA OZDEMIR, Ahmet a Suat GUMUSSOY. Transfer Function Estimation in System Identification Toolbox via Vector Fitting [online]. *IFAC-PapersOnLine.* 2017, **50**(1), 6232-6237. Dostupné z: 10.1016/j.ifacol.2017.08.1026.

[31] SCIPY COMMUNITY. *scipy.signal.cont2discrete* [online]. *SciPy v1.8.1 Manual.* 20. května 2022 [cit. 19. července 2022]. Dostupné z: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.cont2discrete.html.

[32] HAVLENA, Vladimír. *Moderní teorie řízení. Doplňkové skriptum.* Vyd. 1. Praha : Vydavatelství ČVUT, 1999. 8001020363.

[33] NELLES, Oliver. *Nonlinear system identification. From classical approaches to neural networks and fuzzy models : with 422 figures.* Berlin, New York, Barcelona [etc.] : Springer, op. 2001. 3-540-67369-5.

[34] MARTIN SMRŽ. *Návrh a vlastnosti mechatronické tuhosti.* Disertační práce. Praha, 2015.

[35] AHMED FAWZY GAD. *pygad Module* [online]. *PyGAD 2.17.0 documentation.* 8. července 2022 [cit. 19. července 2022]. Dostupné z: https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html.

[36] ALBADR, Musatafa Abbas, Sabrina TIUN, Masri AYOB a Fahad AL-DHIEF. Genetic Algorithm Based on Natural Selection Theory for Optimization Problems [online]. *Symmetry.* 2020, **12**(11), 1758. Dostupné z: 10.3390/sym12111758.

[37] CHOLLET, François. *Deep learning v jazyku Python. Knihovny Keras, Tensorflow.* První vydání. Praha : Grada Publishing, 2019. Myslíme v. 978-80-247-3100-1.

[38] GODOY, Daniel. Understanding binary cross-entropy / log loss: a visual explanation [online]. *Towards Data Science.* 21. listopadu 2018 [cit. 20. července 2022]. Dostupné z: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[39] THARWAT, Alaa. Classification assessment methods [online]. *Applied Computing and Informatics.* 2021, **17**(1), 168-192. Dostupné z: 10.1016/j.aci.2018.08.003.

[40] DIVE INTO DEEP LEARNING. *9.7. Backpropagation Through Time* [online]. *Dive into Deep Learning 1.0.0-alpha0 documentation.* 16. července 2022 [cit. 18. července 2022]. Dostupné z: https://d2l.ai/chapter_recurrent-neural-networks/bptt.html.

[41] TENSORFLOW. *tf.data.Dataset* [online]. 20. července 2022 [cit. 30. července 2022]. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/data/Dataset.

[42] TEAM KERAS. *Keras documentation: Callbacks API* [online]. 27. července 2022 [cit. 30. července 2022]. Dostupné z: https://keras.io/api/callbacks/.

[43] TEAM KERAS. *Keras documentation: Adam* [online]. 27. července 2022 [cit. 30. července 2022]. Dostupné z: https://keras.io/api/optimizers/adam/.

[44] TEAM KERAS. *Keras documentation: Timeseries classification from scratch* [online]. 27. července 2022 [cit. 30. července 2022]. Dostupné z: https://keras.io/examples/timeseries/timeseries_classification_from_scratch/.

Přílohy

9 Metoda konečných prvků

9.1 main (Python)

```
## SIMULATION FOR GIVEN TIME

import time
import shapes_csv
import numpy as np

timlim = time.perf_counter() + 120

while time.perf_counter() < timlim:
    shape = np.random.randint(1, 54)
    shapes_csv.simulate_shape(shape, 2, 1_024, 'broken_pokus', 'unbroken_pokus', 'reference_pokus')

print('___SIMULATION SUCCESSFULLY FINISHED___')

## SIMULATION OF VERIFICATION DATA FOR GIVEN TIME

import time
import shapes_csv
import numpy as np

timlim = time.perf_counter() + 120

while time.perf_counter() < timlim:
    shape = np.random.randint(55, 62)
    shapes_csv.simulate_shape(shape, 2, 1_024, 'verification_broken', 'verification_unbroken', 'verification_reference')

print('___SIMULATION SUCCESSFULLY FINISHED___')
```

9.2 shapes_csv (Python)

```
import numpy as np
import FEM_computations as FEM
import constants as c


def uncertain(value):
    """
    :param value: Value on which will be applied uncertainty
    :return: Value with applied uncertainty
    """
    return value + np.random.randn() * c.three_sigma_percent * value / 300


def add_impacts(max_time, freq, model, refmodel, broken_path, unbroken_path, reference_path):
    """
    :param max_time: Maximal time of simulation in seconds
    :param freq: Measurement frequency
    :param model: FEM Model object
    :param refmodel: FEM Model object
```

```
20.       :param broken_path: path to file
21.       :param unbroken_path: path to file
22.       :param reference_path: path to file
23.       :return: True when successful
24.       """
25.       spring_stiffness = uncertain(c.spring_stiffness)
26.       max_impact_force = uncertain(c.max_impact_force)
27.
28.       broken = FEM.IntegrationProblem(model)
29.       broken.add_nonlinearity(model.spring_dir, spring_stiffness, 1)
30.       broken.add_impact(model.impacted_dir, max_impact_force, c.impact_duration)
31.       broken_result, broken_force = broken.start_integration_csv(max_time, freq,
      model.measured_directions)
32.
33.       unbroken = FEM.IntegrationProblem(model)
34.       unbroken.add_spring(model.spring_dir, spring_stiffness)
35.       unbroken.add_impact(model.impacted_dir, max_impact_force, c.impact_duration)
36.       unbroken_result, unbroken_force = unbroken.start_integration_csv(max_time, freq,
      model.measured_directions)
37.
38.       reference = FEM.IntegrationProblem(refmodel)
39.       reference.add_spring(refmodel.spring_dir, uncertain(c.spring_stiffness))
40.       reference.add_impact(refmodel.impacted_dir, uncertain(c.max_impact_force),
      c.impact_duration)
41.       reference_result, reference_force = reference.start_integration_csv(max_time, freq,
      refmodel.measured_directions)
42.
43.       if np.any(broken_result) and np.any(unbroken_result) and np.any(reference_result):
44.           save_result_csv(broken_path+'.csv', broken_result)
45.           save_result_csv(unbroken_path+'.csv', unbroken_result)
46.           save_result_csv(reference_path+'.csv', reference_result)
47.           save_result_csv(broken_path+'_force.csv', broken_force)
48.           save_result_csv(unbroken_path+'_force.csv', unbroken_force)
49.           save_result_csv(reference_path+'_force.csv', reference_force)
50.           return True
51.       else:
52.           print('All results for shape discarded')
53.           return False
54.
55.
56.   def save_model_info(shape, d, model):
57.       """
58.       Saves model info to file
59.       :param shape: Number defining shapes
60.       :param d: Parameter of length parameters
61.       :param model: FEM model object
62.       :return:
63.       """
64.       import csv
65.       from os.path import exists as exists
66.
67.       if not(exists('model_info.csv')):
68.           file = open('model_info.csv', 'a', encoding='UTF8')
69.           writer = csv.writer(file)
70.           row = ['shape ', 'd ', 'measured direction ']
71.           writer.writerow(row)
72.
73.       file = open('model_info.csv', 'a')
74.       writer = csv.writer(file)
75.
76.       for direction in model.measured_directions:
77.           row = [shape, d, direction]
78.           writer.writerow(row)
79.
80.       file.close()
```

```
81.
82.
83.  def save_discarded_info(shape, d, model):
84.      """
85.      Saves model info if simulation was not succesfull
86.      :param shape: Number defining shapes
87.      :param d: Parameter of length parameters
88.      :param model: FEM model object
89.      :return:
90.      """
91.      import csv
92.      from os.path import exists as exists
93.
94.      if not(exists('discarded_info.csv')):
95.          file = open('discarded_info.csv', 'a', encoding='UTF8')
96.          writer = csv.writer(file)
97.          row = ['shape ', 'd ', 'measured direction ']
98.          writer.writerow(row)
99.
100.     file = open('discarded_info.csv', 'a')
101.     writer = csv.writer(file)
102.
103.     for direction in model.measured_directions:
104.         row = [shape, d, direction]
105.         writer.writerow(row)
106.
107.     file.close()
108.
109.
110. def save_result_csv(csv_path, results):
111.     """
112.     Saves results to csv file
113.     :param csv_path: Path to csv file
114.     :param results: Results that should be saved
115.     :return:
116.     """
117.     import csv
118.
119.     file = open(csv_path, 'a')
120.     writer = csv.writer(file)
121.
122.     for row in results:
123.         writer.writerow(row)
124.
125.     file.close()
126.
127.
128. def simulate_shape(shape: int, max_time: float, freq: float, broken_path: str,
     unbroken_path: str, reference_path: str):
129.     """
130.     Makes simulation for given parameters and saves results to csv files
131.     :param shape: Number defining shapes
132.     :param max_time: Maximal time of simulation in seconds
133.     :param freq: Measurement frequency
134.     :param broken_path: Path to csv file where results should be saved
135.     :param unbroken_path: Path to csv file where results should be saved
136.     :param reference_path: Path to csv file where results should be saved
137.     :return:
138.     """
139.     d = np.random.uniform(1 / 4, 4)   # Change of length parameters to create unique
     instance
140.     print(f'simulating case {shape}')
141.
142.     match shape:
143.         case 1:
```

```
144.             model = model1(10, d)
145.             refmodel = model1(10, d)
146.         case 2:
147.             model = model1(13, d)
148.             refmodel = model1(13, d)
149.         case 3:
150.             model = model1(16, d)
151.             refmodel = model1(16, d)
152.         case 4:
153.             model = model1(19, d)
154.             refmodel = model1(19, d)
155.         case 5:
156.             model = model1(22, d)
157.             refmodel = model1(22, d)
158.         case 6:
159.             model = model2(4, d)
160.             refmodel = model2(4, d)
161.         case 7:
162.             model = model2(7, d)
163.             refmodel = model2(7, d)
164.         case 8:
165.             model = model2(10, d)
166.             refmodel = model2(10, d)
167.         case 9:
168.             model = model2(13, d)
169.             refmodel = model2(13, d)
170.         case 10:
171.             model = model3(7, d)
172.             refmodel = model3(7, d)
173.         case 11:
174.             model = model3(10, d)
175.             refmodel = model3(10, d)
176.         case 12:
177.             model = model3(13, d)
178.             refmodel = model3(13, d)
179.         case 13:
180.             model = model3(16, d)
181.             refmodel = model3(16, d)
182.         case 14:
183.             model = model4(4, d)
184.             refmodel = model4(4, d)
185.         case 15:
186.             model = model4(7, d)
187.             refmodel = model4(7, d)
188.         case 16:
189.             model = model4(16, d)
190.             refmodel = model4(16, d)
191.         case 17:
192.             model = model4(19, d)
193.             refmodel = model4(19, d)
194.         case 18:
195.             model = model5(7, d)
196.             refmodel = model5(7, d)
197.         case 19:
198.             model = model5(10, d)
199.             refmodel = model5(10, d)
200.         case 20:
201.             model = model5(13, d)
202.             refmodel = model5(13, d)
203.         case 21:
204.             model = model6(13, d)
205.             refmodel = model6(13, d)
206.         case 22:
207.             model = model6(16, d)
208.             refmodel = model6(16, d)
```

```
209.          case 23:
210.              model = model6(19, d)
211.              refmodel = model6(19, d)
212.          case 24:
213.              model = model7(3, d)
214.              refmodel = model7(3, d)
215.          case 25:
216.              model = model7(6, d)
217.              refmodel = model7(6, d)
218.          case 26:
219.              model = model7(10, d)
220.              refmodel = model7(10, d)
221.          case 27:
222.              model = model7(13, d)
223.              refmodel = model7(13, d)
224.          case 28:
225.              model = model7(16, d)
226.              refmodel = model7(16, d)
227.          case 29:
228.              model = model8(4, d)
229.              refmodel = model8(4, d)
230.          case 30:
231.              model = model8(7, d)
232.              refmodel = model8(7, d)
233.          case 31:
234.              model = model8(10, d)
235.              refmodel = model8(10, d)
236.          case 32:
237.              model = model8(13, d)
238.              refmodel = model8(13, d)
239.          case 33:
240.              model = model9(4, d)
241.              refmodel = model9(4, d)
242.          case 34:
243.              model = model9(7, d)
244.              refmodel = model9(7, d)
245.          case 35:
246.              model = model9(13, d)
247.              refmodel = model9(13, d)
248.          case 36:
249.              model = model9(21, d)
250.              refmodel = model9(21, d)
251.          case 37:
252.              model = model9(24, d)
253.              refmodel = model9(24, d)
254.          case 38:
255.              model = model10(12, d)
256.              refmodel = model10(12, d)
257.          case 39:
258.              model = model10(15, d)
259.              refmodel = model10(15, d)
260.          case 40:
261.              model = model10(19, d)
262.              refmodel = model10(19, d)
263.          case 41:
264.              model = model10(22, d)
265.              refmodel = model10(22, d)
266.          case 42:
267.              model = model10(24, d)
268.              refmodel = model10(24, d)
269.          case 43:
270.              model = model10(27, d)
271.              refmodel = model10(27, d)
272.          case 44:
273.              model = model11(9, d)
```

```
274.             refmodel = model11(9, d)
275.         case 45:
276.             model = model11(10, d)
277.             refmodel = model11(10, d)
278.         case 46:
279.             model = model12(9, d)
280.             refmodel = model12(9, d)
281.         case 47:
282.             model = model12(10, d)
283.             refmodel = model12(10, d)
284.         case 48:
285.             model = model12(12, d)
286.             refmodel = model12(12, d)
287.         case 49:
288.             model = model12(13, d)
289.             refmodel = model12(13, d)
290.         case 50:
291.             model = model13(4, d)
292.             refmodel = model13(4, d)
293.         case 51:
294.             model = model13(7, d)
295.             refmodel = model13(7, d)
296.         case 52:
297.             model = model13(24, d)
298.             refmodel = model13(24, d)
299.         case 53:
300.             model = model13(27, d)
301.             refmodel = model13(27, d)
302.         case 54:
303.             model = model13(30, d)
304.             refmodel = model13(30, d)
305.         case 55:
306.             model = model14(15, d)
307.             refmodel = model14(15, d)
308.         case 56:
309.             model = model14(16, d)
310.             refmodel = model14(16, d)
311.         case 57:
312.             model = model14(19, d)
313.             refmodel = model14(19, d)
314.         case 58:
315.             model = model15(4, d)
316.             refmodel = model15(4, d)
317.         case 59:
318.             model = model15(7, d)
319.             refmodel = model15(7, d)
320.         case 60:
321.             model = model15(10, d)
322.             refmodel = model15(10, d)
323.         case 61:
324.             model = model15(13, d)
325.             refmodel = model15(13, d)
326.         case 62:
327.             model = model15(16, d)
328.             refmodel = model15(16, d)
329.         case _:
330.             return
331.
332.     success = add_impacts(max_time, freq, model, refmodel, broken_path, unbroken_path,
    reference_path)
333.     if success:
334.         save_model_info(shape, d, model)
335.     else:
336.         save_discarded_info(shape, d, model)
337.
```

```
338.
339. def model1(impacted_dir, d):
340.     """
341.     :param impacted_dir: Direction which will be impacted
342.     :param d: Random change of length parameters to create unique instance
343.     :return: FEM model
344.     """
345.     horizontal_nodes = 5
346.     horizontal_length = 1 * d
347.
348.     lmodel = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro),
     uncertain(c.width * d),uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
349.     lmodel.add_element(0, 1, np.pi / 2, 0.002)
350.     lmodel.add_element(1, 2, np.pi / 2, 0.002)
351.
352.     for i in range(horizontal_nodes):
353.         lmodel.add_element(2 + i, 3 + i, 0, horizontal_length / horizontal_nodes)
354.
355.     lmodel.add_constraint([0, 1, 2])
356.     lmodel.add_spring_dir(6)
357.     lmodel.add_measured_directions([9, 10, 12, 13, 15, 16, 18, 19, 21, 22])
358.     lmodel.add_impacted_dir(impacted_dir)
359.
360.     lmodel.compile_model()
361.
362.     return lmodel
363.
364.
365. def model2(impacted_dir, d):
366.     """
367.     :param impacted_dir: Direction which will be impacted
368.     :param d: Random change of length parameters to create unique instance
369.     :return: FEM model
370.     """
371.     horizontal_nodes = 5
372.     horizontal_length = 1 * d
373.
374.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d), uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
375.
376.     for i in range(horizontal_nodes):
377.         model.add_element(0 + i, 1 + i, 0, horizontal_length / horizontal_nodes)
378.     model.add_element(5, 6, 0, 0.001)
379.
380.     model.add_constraint([0, 1, 2])
381.     model.add_constraint([18, 19, 20])
382.     model.add_spring_dir(16)
383.     model.add_measured_directions([4, 7, 10, 13])
384.     model.add_impacted_dir(impacted_dir)
385.
386.     model.compile_model()
387.
388.     return model
389.
390.
391. def model3(impacted_dir, d):
392.     """
393.     :param impacted_dir: Direction which will be impacted
394.     :param d: Random change of length parameters to create unique instance
395.     :return: FEM model
396.     """
397.     horizontal_nodes = 5
398.     horizontal_length = 1 * d
399.
```

```
400.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
401.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
402.     model.add_element(0, 1, 0, 0.001)
403.     for i in range(horizontal_nodes):
404.         model.add_element(1 + i, 2 + i, 0, horizontal_length / horizontal_nodes)
405.
406.     model.add_constraint([0, 1, 2])
407.     model.add_spring_dir(4)
408.     model.add_measured_directions([7, 10, 13, 16])
409.     model.add_impacted_dir(impacted_dir)
410.
411.     model.compile_model()
412.
413.     return model
414.
415.
416. def model4(impacted_dir, d):
417.     """
418.     :param impacted_dir: Direction which will be impacted
419.     :param d: Random change of length parameters to create unique instance
420.     :return: FEM model
421.     """
422.     l = 0.3 * d
423.
424.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
425.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
426.     model.add_element(0, 1, 0, l)
427.     model.add_element(1, 2, 0, l)
428.     model.add_element(2, 3, 0, l)
429.     model.add_element(3, 4, np.pi * 3 / 2, l)
430.     model.add_element(4, 5, np.pi, l)
431.     model.add_element(5, 6, np.pi, l)
432.     model.add_element(6, 7, np.pi, l)
433.     model.add_element(0, 8, np.pi * 3 / 2, 0.01)
434.     model.add_element(8, 7, np.pi * 3 / 2, l - 0.01)
435.     model.add_element(1, 6, np.pi * 3 / 2, l)
436.     model.add_element(2, 5, np.pi * 3 / 2, l)
437.
438.     model.add_constraint([0, 1, 2])
439.     model.add_constraint([10])
440.     model.add_spring_dir(24)
441.     model.add_measured_directions([3, 4, 6, 7, 12, 13, 15, 16, 18, 19, 21, 22])
442.     model.add_impacted_dir(impacted_dir)
443.
444.     model.compile_model()
445.
446.     return model
447.
448.
449. def model5(impacted_dir, d):
450.     """
451.     :param impacted_dir: Direction which will be impacted
452.     :param d: Random change of length parameters to create unique instance
453.     :return: FEM model
454.     """
455.     horizontal_nodes = 4
456.     horizontal_length = 1 * d
457.
458.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
459.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
460.     model.add_element(0, 1, np.pi / 2, 0.0001)
461.     for i in range(horizontal_nodes):
```

```
462.         model.add_element(1 + i, 2 + i, 0, horizontal_length / horizontal_nodes)
463.
464.     model.add_constraint([0, 1])
465.     model.add_constraint([15, 16])
466.     model.add_spring_dir(3)
467.     model.add_measured_directions([6, 7, 9, 10, 12, 13])
468.     model.add_impacted_dir(impacted_dir)
469.
470.     model.compile_model()
471.
472.     return model
473.
474.
475. def model6(impacted_dir, d):
476.     """
477.     :param impacted_dir: Direction which will be impacted
478.     :param d: Random change of length parameters to create unique instance
479.     :return: FEM model
480.     """
481.     horizontal_nodes = 3
482.     horizontal_length = 1 * d
483.
484.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
    * d),
485.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
486.     model.add_element(0, 1, np.pi / 2, 0.01)
487.     model.add_element(1, 2, np.pi / 2, 0.1)
488.     model.add_element(2, 3, np.pi / 2, 0.1)
489.     for i in range(horizontal_nodes):
490.         model.add_element(3 + i, 4 + i, 0, horizontal_length / horizontal_nodes)
491.
492.     model.add_constraint([0, 1, 2])
493.     model.add_spring_dir(3)
494.     model.add_measured_directions([6, 7, 9, 10, 12, 13, 15, 16, 18, 19])
495.     model.add_impacted_dir(impacted_dir)
496.
497.     model.compile_model()
498.
499.     return model
500.
501.
502. def model7(impacted_dir, d):
503.     """
504.     :param impacted_dir: Direction which will be impacted
505.     :param d: Random change of length parameters to create unique instance
506.     :return: FEM model
507.     """
508.     horizontal_nodes = 4
509.     horizontal_length = 1 * d
510.
511.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
    * d),
512.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
513.     model.add_element(0, 1, np.pi / 2, 0.1)
514.     model.add_element(1, 2, np.pi / 2, 0.1)
515.     for i in range(horizontal_nodes):
516.         model.add_element(2 + i, 3 + i, 0, horizontal_length / horizontal_nodes)
517.     model.add_element(6, 7, np.pi * 3 / 2, 0.2 - 0.001)
518.     model.add_element(7, 8, np.pi * 3 / 2, 0.001)
519.
520.     model.add_constraint([0, 1, 2])
521.     model.add_constraint([24, 25, 26])
522.     model.add_spring_dir(21)
523.     model.add_measured_directions([3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 18, 19])
524.     model.add_impacted_dir(impacted_dir)
```

```
525.
526.     model.compile_model()
527.
528.     return model
529.
530.
531. def model8(impacted_dir, d):
532.     """
533.     :param impacted_dir: Direction which will be impacted
534.     :param d: Random change of length parameters to create unique instance
535.     :return: FEM model
536.     """
537.     horizontal_nodes = 5
538.     horizontal_length = 1 * d
539.
540.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
    * d),
541.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
542.     for i in range(horizontal_nodes):
543.         model.add_element(0 + i, 1 + i, 0, horizontal_length / horizontal_nodes)
544.     model.add_element(5, 6, 0, 0.001)
545.
546.     model.add_constraint([0, 1])
547.     model.add_constraint([18, 19, 20])
548.     model.add_spring_dir(16)
549.     model.add_measured_directions([4, 7, 10, 13])
550.     model.add_impacted_dir(impacted_dir)
551.
552.     model.compile_model()
553.
554.     return model
555.
556.
557. def model9(impacted_dir, d):
558.     """
559.     :param impacted_dir: Direction which will be impacted
560.     :param d: Random change of length parameters to create unique instance
561.     :return: FEM model
562.     """
563.     horizontal_nodes = 5
564.     horizontal_length = 1 * d
565.
566.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
    * d),
567.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
568.     for i in range(horizontal_nodes):
569.         model.add_element(0 + i, 1 + i, 0, horizontal_length / horizontal_nodes)
570.     model.add_element(5, 6, 0, 0.002)
571.     model.add_element(1, 7, np.pi * 3 / 2, 0.1)
572.     model.add_element(7, 8, np.pi * 3 / 2, 0.1)
573.
574.     model.add_constraint([0, 1])
575.     model.add_constraint([18, 19, 20])
576.     model.add_spring_dir(16)
577.     model.add_measured_directions([3, 4, 6, 7, 9, 10, 12, 13, 21, 22, 24, 25])
578.     model.add_impacted_dir(impacted_dir)
579.
580.     model.compile_model()
581.
582.     return model
583.
584.
585. def model10(impacted_dir, d):
586.     """
587.     :param impacted_dir: Direction which will be impacted
```

```
588.     :param d: Random change of length parameters to create unique instance
589.     :return: FEM model
590.     """
591.     l = 0.3 * d
592.
593.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
594.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
595.     model.add_element(0, 1, 0, l)
596.     model.add_element(1, 2, 0, 2 * l - 0.0005)
597.     model.add_element(2, 3, 0, 0.0005)
598.     model.add_element(3, 4, np.pi / 2, l)
599.     model.add_element(4, 5, np.pi / 2, l)
600.     model.add_element(5, 6, np.pi, l)
601.     model.add_element(6, 7, np.pi, l)
602.     model.add_element(7, 8, np.pi, l)
603.     model.add_element(8, 9, np.pi * 3 / 2, l)
604.     model.add_element(9, 0, np.pi * 3 / 2, l)
605.
606.     model.add_constraint([0, 1])
607.     model.add_constraint([9, 10])
608.     model.add_spring_dir(7)
609.     model.add_measured_directions([12, 13, 15, 16, 18, 19, 21, 22, 24, 25, 27, 28])
610.     model.add_impacted_dir(impacted_dir)
611.
612.     model.compile_model()
613.
614.     return model
615.
616.
617. def model11(impacted_dir, d):
618.     """
619.     :param impacted_dir: Direction which will be impacted
620.     :param d: Random change of length parameters to create unique instance
621.     :return: FEM model
622.     """
623.     l = 3 * d
624.
625.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
626.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
627.     model.add_element(0, 1, 0, l - 0.01)
628.     model.add_element(1, 2, 0, 0.01)
629.     model.add_element(2, 3, np.pi / 3, l)
630.     model.add_element(3, 4, np.pi, l)
631.     model.add_element(4, 2, np.pi * 5 / 3, l)
632.     model.add_element(4, 0, np.pi * 4 / 3, l)
633.
634.     model.add_constraint([0, 1])
635.     model.add_constraint([6, 7])
636.     model.add_spring_dir(4)
637.     model.add_measured_directions([12, 13, 9, 10])
638.     model.add_impacted_dir(impacted_dir)
639.
640.     model.compile_model()
641.
642.     return model
643.
644.
645. def model12(impacted_dir, d):
646.     """
647.     :param impacted_dir: Direction which will be impacted
648.     :param d: Random change of length parameters to create unique instance
649.     :return: FEM model
650.     """
```

```
651.     l = 3 * d
652.
653.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
654.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
655.     model.add_element(0, 1, 0, l)
656.     model.add_element(1, 2, 0, l)
657.     model.add_element(2, 3, np.pi * 2 / 3, l)
658.     model.add_element(3, 4, np.pi, l)
659.     model.add_element(3, 1, np.pi * 4 / 3, l)
660.     model.add_element(4, 1, np.pi * 5 / 3, l)
661.     model.add_element(4, 5, np.pi * 4 / 3, l - 0.01)
662.     model.add_element(5, 0, np.pi * 4 / 3, 0.01)
663.
664.     model.add_constraint([0, 1])
665.     model.add_constraint([3, 4])
666.     model.add_constraint([6, 7])
667.     model.add_spring_dir(15)
668.     model.add_measured_directions([9, 10, 12, 13])
669.     model.add_impacted_dir(impacted_dir)
670.
671.     model.compile_model()
672.
673.     return model
674.
675.
676. def model13(impacted_dir, d):
677.     """
678.     :param impacted_dir: Direction which will be impacted
679.     :param d: Random change of length parameters to create unique instance
680.     :return: FEM model
681.     """
682.     l = 0.5 * d
683.
684.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
685.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
686.     model.add_element(0, 1, 0, l)
687.     model.add_element(1, 2, 0, l)
688.     model.add_element(2, 3, 0, l)
689.     model.add_element(3, 4, 0, l)
690.     model.add_element(3, 5, np.pi * 3 / 2, l)
691.     model.add_element(5, 6, 0, l - 0.01)
692.     model.add_element(6, 7, 0, 0.01)
693.     model.add_element(1, 8, np.pi / 2, l)
694.     model.add_element(2, 9, np.pi / 2, l)
695.     model.add_element(8, 10, np.pi / 2, l)
696.
697.     model.add_constraint([0, 1, 2])
698.     model.add_constraint([12, 13, 14])
699.     model.add_constraint([21, 22, 23])
700.     model.add_spring_dir(19)
701.     model.add_measured_directions([3, 4, 6, 7, 9, 10, 15, 16, 24, 25, 27, 28, 30, 31])
702.     model.add_impacted_dir(impacted_dir)
703.
704.     model.compile_model()
705.
706.     return model
707.
708.
709. def model14(impacted_dir, d):
710.     """
711.     :param impacted_dir: Direction which will be impacted
712.     :param d: Random change of length parameters to create unique instance
713.     :return: FEM model
```

```
714.     """
715.     l = 0.5 * d
716.
717.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
718.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
719.     model.add_element(0, 1, np.pi / 2, l)
720.     model.add_element(1, 2, np.pi / 2, l)
721.     model.add_element(2, 3, np.pi / 2, l)
722.     model.add_element(3, 4, 0, l)
723.     model.add_element(4, 5, 0, l)
724.     model.add_element(5, 6, np.pi * 3 / 2, l)
725.     model.add_element(6, 7, np.pi, l)
726.     model.add_element(4, 7, np.pi * 3 / 2, l)
727.     model.add_element(7, 8, np.pi * 3 / 2, 2 * l - 0.05)
728.     model.add_element(8, 9, np.pi * 3 / 2, 0.05)
729.
730.     model.add_constraint([0, 1, 2])
731.     model.add_constraint([27, 28, 29])
732.     model.add_spring_dir(24)
733.     model.add_measured_directions([9, 10, 12, 13, 15, 16, 21, 22, 18, 19, 24, 25])
734.     model.add_impacted_dir(impacted_dir)
735.
736.     model.compile_model()
737.
738.     return model
739.
740.
741. def model15(impacted_dir, d):
742.     """
743.     :param impacted_dir: Direction which will be impacted
744.     :param d: Random change of length parameters to create unique instance
745.     :return: FEM model
746.     """
747.     horizontal_nodes = 6
748.     horizontal_length = 2 * d
749.
750.     model = FEM.Model(uncertain(c.E), uncertain(c.ni), uncertain(c.ro), uncertain(c.width
     * d),
751.                       uncertain(c.height * d), uncertain(c.a0), uncertain(c.a1))
752.     for i in range(horizontal_nodes):
753.         model.add_element(0 + i, 1 + i, 0, horizontal_length / horizontal_nodes)
754.     model.add_element(6, 7, 0, 0.1)
755.     model.add_element(7, 8, 0, 0.1)
756.
757.     model.add_constraint([0, 1])
758.     model.add_constraint([18, 19])
759.     model.add_constraint([24, 25])
760.     model.add_spring_dir(22)
761.     model.add_measured_directions([3, 4, 6, 7, 9, 10, 12, 13, 15, 16])
762.     model.add_impacted_dir(impacted_dir)
763.
764.     model.compile_model()
765.
766.     return model
```

9.3 FEM_computations (Python)

```
1.  import numpy as np
2.  from numpy.linalg import inv
3.  import matplotlib.pyplot as plt
4.  import scipy.integrate as ode
5.  import scipy.linalg as la
6.  import time
```

```
7.
8.
9.  def element_to_big_matrix(big_matrix, element_matrix, map1, map2):
10.     """
11.     Implementation of mapping function
12.     :param big_matrix: Global stiffness matrix
13.     :param element_matrix: Element stiffness matrix in global coordinates
14.     :param map1: Mapping of first local node
15.     :param map2: Mapping of second local node
16.     :return: big_matrix - Global stiffness matrix
17.     """
18.     big_matrix[np.ix_(map1, map1)] += element_matrix[0:3, 0:3]
19.     big_matrix[np.ix_(map1, map2)] += element_matrix[0:3, 3:6]
20.     big_matrix[np.ix_(map2, map1)] += element_matrix[3:6, 0:3]
21.     big_matrix[np.ix_(map2, map2)] += element_matrix[3:6, 3:6]
22.
23.     return big_matrix
24.
25.
26. class Beam:
27.     def __init__(self, model_data, from_point, to_point, angle, length):
28.         """
29.         :param model_data: Instance of class Model
30.         :param from_point: Global node belonging to first local node
31.         :param to_point: Global node belonging to second local node
32.         :param angle: Angle fi of element in global coordinates
33.         :param length: length of element
34.         """
35.         self.length = length
36.         self.model_data = model_data
37.         self.angle = angle
38.         self.K = self.local_to_global(self.local_k())
39.         self.M = self.local_to_global(self.local_m())
40.         self.map1 = np.array(np.array([0, 1, 2]) + 3 * from_point)
41.         self.map2 = np.array(np.array([0, 1, 2]) + 3 * to_point)
42.         self.from_point = from_point
43.         self.to_point = to_point
44.
45.     def local_k(self):
46.         """
47.         :return: Local stiffness matrix of element
48.         """
49.         l = self.length
50.         E = self.model_data.E
51.         J = self.model_data.J
52.         A = self.model_data.surface
53.
54.         K_local = np.array([[E * A / l, 0, 0, -E * A / l, 0, 0],
55.                             [0, 12 * E * J / l ** 3, 6 * E * J / l ** 2, 0, -12 * E * J /
    l ** 3, 6 * E * J / l ** 2],
56.                             [0, 6 * E * J / l ** 2, 4 * E * J / l, 0, -6 * E * J / l **
    2, 2 * E * J / l],
57.                             [-E * A / l, 0, 0, E * A / l, 0, 0],
58.                             [0, -12 * E * J / l ** 3, -6 * E * J / l ** 2, 0, 12 * E * J
    / l ** 3, -6 * E * J / l ** 2],
59.                             [0, 6 * E * J / l ** 2, 2 * E * J / l, 0, -6 * E * J / l **
    2, 4 * E * J / l]])
60.         return K_local
61.
62.     def local_m(self):
63.         """
64.         :return: Local weight matrix of element
65.         """
66.         ro = self.model_data.ro
67.         A = self.model_data.surface
```

```
68.          l = self.length
69.
70.          M_local = np.array([[140, 0, 0, 70, 0, 0],
71.                              [0, 156, 22 * l, 0, 54, -13 * l],
72.                              [0, 22 * l, 4 * l ** 2, 0, 13 * l, -3 * l ** 2],
73.                              [70, 0, 0, 140, 0, 0],
74.                              [0, 54, 13 * l, 0, 156, -22 * l],
75.                              [0, -13 * l, -3 * l ** 2, 0, -22 * l, 4 * l ** 2]]) \
76.                     * (ro * A * l / 420)
77.          return M_local
78.
79.      def local_to_global(self, local_matrix):
80.          """
81.          :param local_matrix: Matrix of element in local coordinates
82.          :return: Matrix of element in global coordinates
83.          """
84.          fi = self.angle
85.
86.          lam = np.array([[np.cos(fi), np.sin(fi), 0, 0, 0, 0],
87.                          [-np.sin(fi), np.cos(fi), 0, 0, 0, 0],
88.                          [0, 0, 1, 0, 0, 0],
89.                          [0, 0, 0, np.cos(fi), np.sin(fi), 0],
90.                          [0, 0, 0, -np.sin(fi), np.cos(fi), 0],
91.                          [0, 0, 0, 0, 0, 1]])
92.
93.          global_matrix = np.transpose(lam) @ local_matrix @ lam
94.
95.          return global_matrix
96.
97.      def add2global_K(self, K):
98.          """
99.          :param K: Global stiffness matrix
100.         :return: Global stiffness matrix with element matrix added
101.         """
102.         K = element_to_big_matrix(K, self.K, self.map1, self.map2)
103.
104.         return K
105.
106.     def add2global_M(self, M):
107.         """
108.         :param M: Global weight matrix
109.         :return: Global weight matrix with element matrix added
110.         """
111.         M = element_to_big_matrix(M, self.M, self.map1, self.map2)
112.
113.         return M
114.
115.
116. class Model:
117.     def __init__(self, E, ni, ro, width, height, a0, a1):
118.         """
119.         :param E: Young modulus
120.         :param ni: Poisson number
121.         :param ro: Material density
122.         :param width: Width of beam element
123.         :param height: Height of beam element
124.         :param a0: alpha parameter to get reynolds B matrix
125.         :param a1: beta parameter to get reynolds B matrix
126.         """
127.         self.E = E  # Pa
128.         self.ni = ni
129.         self.ro = ro  # kg/m3
130.         self.width = width  # m
131.         self.height = height  # m
132.         self.surface = width * height
```

```
133.         self.J = height ** 3 * width / 12
134.         self.G = E / (2 * (1 + ni))
135.         self.max_point = 0
136.         self.element_list = []
137.         self.constraints = []
138.         self.a0 = a0
139.         self.a1 = a1
140.         self.K = None
141.         self.M = None
142.         self.B = None
143.         self.direction_map = None
144.         self.measured_directions = None
145.         self.impacted_dir = None
146.         self.spring_dir = None
147.
148.     def add_element(self, from_point, to_point, angle, length):
149.         """
150.         Adds beam element to the system
151.         :param from_point: Global node belonging to first local node
152.         :param to_point: Global node belonging to second local node
153.         :param angle: Angle fi of element in global coordinates
154.         :param length: length of element
155.         :return:
156.         """
157.         if from_point + 1 > self.max_point:
158.             self.max_point = from_point + 1
159.         if to_point + 1 > self.max_point:
160.             self.max_point = to_point + 1
161.
162.         model_data = self
163.         self.element_list.append(Beam(model_data, from_point, to_point, angle, length))
164.         print(f'element {len(self.element_list)} added')
165.
166.     def compile_model(self):
167.         """
168.         Creates global matrices, adds constraints, prepares model for computation
169.         Creates direction map - directions that are free to move
170.         :return:
171.         """
172.         self.K = np.zeros((3 * self.max_point, 3 * self.max_point), dtype=float)
173.         self.M = np.zeros((3 * self.max_point, 3 * self.max_point), dtype=float)
174.         self.direction_map = range(0, (3 * self.max_point))
175.
176.         for element in self.element_list:  # Create global matrices
177.             element.add2global_K(self.K)
178.             element.add2global_M(self.M)
179.
180.         for constraint in self.constraints:  # Adding constraints
181.             for i in [0, 1]:
182.                 self.K = np.delete(self.K, constraint, i)
183.                 self.M = np.delete(self.M, constraint, i)
184.             self.direction_map = np.delete(self.direction_map, constraint, 0)
185.
186.         self.B = self.a0 * self.K + self.a1 * self.M  # Rayleigh damping
187.
188.     def add_constraint(self, constrained_move: list):
189.         """
190.         Adds constrained directions to the list
191.         :param constrained_move: List of directions that should be constrained
192.         :return:
193.         """
194.         for constraint in constrained_move:
195.             self.constraints.append(constraint)
196.         self.constraints = sorted(set(self.constraints), reverse=True)
197.
```

```
198.     def add_measured_directions(self, measured_directions):
199.         """
200.         Adds measured directions
201.         :param measured_directions: Directions whose movement should be recorded
202.         :return:
203.         """
204.         self.measured_directions = measured_directions
205.
206.     def add_impacted_dir(self, impacted_dir):
207.         """
208.         Adds impacted directions
209.         :param impacted_dir: Direction which will be impacted
210.         :return:
211.         """
212.         self.impacted_dir = impacted_dir
213.
214.     def add_spring_dir(self, spring_dir):
215.         """
216.         :param spring_dir: Direction on which will be applied nonlinear spring
217.         :return:
218.         """
219.         self.spring_dir = spring_dir
220.
221.
222. class IntegrationProblem:
223.     def __init__(self, model: Model):
224.         """
225.         :param model: Instance of class Model
226.         """
227.
228.         def ft(t):
229.             return 0
230.
231.         self.direction_map = model.direction_map
232.         self.vect_fu = np.zeros(len(self.direction_map))  # Vector which direction will
    have nonlinear spring
233.         self.vect_ft = np.zeros(len(self.direction_map))  # Vector which direction will
    be impacted
234.         self.vect_k = np.zeros(len(self.direction_map))
235.         self.ft = ft  # Impact force function of time t
236.         self.element_list = model.element_list
237.         self.node_coordinates = {'0': [0., 0.]}
238.         self.measured_direction = 0
239.         self.K = model.K
240.         self.L = model.K
241.         self.M = model.M
242.         self.B = model.B
243.         self.a0 = model.a0
244.         self.a1 = model.a1
245.         self.r = None
246.         self.f = np.array([])  # Vector of impact force in time
247.         self.Ep = np.array([])
248.         self.Ek = np.array([])
249.         self.Er = np.array([])
250.         self.spring_direction = None
251.         self.Ak = []
252.         self.Bk = []
253.         self.Ck = []
254.         self.Dk = []
255.         self.Al = []
256.         self.Bl = []
257.         self.Cl = []
258.         self.Dl = []
259.         self.u = np.array([[]])  # States of state system [u; u_dot]
260.         self.u_dot = np.array([[]])
```

```
261.
262.     def add_nonlinearity(self, direction, stiffness, direction_sign):
263.         """
264.         Creates matrix L - stiffer part of matrix K (by nonlinear spring)
265.         :param direction: Direction on which will be applied nonlinearity
266.         :param stiffness: Stiffness of nonlinear spring
267.         :param direction_sign: In which direction will nonlinearity act
268.         :return:
269.         """
270.         print('adding nonlinear spring')
271.         self.spring_direction = self.new_direction(direction)
272.         self.vect_fu[self.spring_direction] = 1 * direction_sign
273.
274.         v1 = self.vect_fu.reshape(1, len(self.vect_fu))
275.         v2 = self.vect_fu.reshape(len(self.vect_fu), 1)
276.
277.         H = v2 @ v1 * stiffness
278.
279.         self.L = self.K + H
280.
281.     def add_spring(self, direction, stiffness):
282.         """
283.         Adds stiffness of linear spring to matrix K
284.         :param direction: Direction on which will be applied spring
285.         :param stiffness: Stiffness of linear spring
286.         :return:
287.         """
288.         print('adding linear spring')
289.         self.spring_direction = self.new_direction(direction)
290.         self.vect_k[self.spring_direction] = 1
291.
292.         v1 = self.vect_k.reshape(1, len(self.vect_k))
293.         v2 = self.vect_k.reshape(len(self.vect_k), 1)
294.         H = v2 @ v1 * stiffness
295.
296.         self.K = self.K + H
297.         self.L = self.K
298.
299.     def add_impact(self, direction, max_force, duration):
300.         """
301.         Creates time function of impact force f(t)
302.         :param direction: Direction which will be impacted
303.         :param max_force: Impact force maximum
304.         :param duration: Duration of impact
305.         :return:
306.         """
307.         if duration == 0:
308.             self.add_dirac_impact(direction)
309.         else:
310.             direction = self.new_direction(direction)
311.             self.vect_ft[direction] = 1
312.
313.             def f(t):
314.                 if t < duration:
315.                     force = 0.5 * (1 + np.cos(2 * np.pi / duration * t + np.pi)) *
    max_force
316.                 else:
317.                     force = 0
318.                 return force
319.
320.             self.ft = f
321.
322.     def add_dirac_impact(self, direction):
323.         """
324.         Creates time function of impact force f(t) with dirac pulse
```

```
325.         :param direction: Direction which will be impacted
326.         :return:
327.         """
328.         direction = self.new_direction(direction)
329.         self.vect_ft[direction] = 1
330.
331.         def f(t):
332.             if t == 0:
333.                 force = 1000
334.             else:
335.                 force = 0
336.             return force
337.
338.         self.ft = f
339.
340.     def change_measured_direction(self, direction):
341.         """
342.         Changes measured direction
343.         :param direction: Direction which will be measured
344.         :return:
345.         """
346.         self.measured_direction = self.new_direction(direction)
347.
348.     def new_direction(self, direction):
349.         """
350.         :param direction: Direction of unmodified system (without constraints)
351.         :return: new_direction - direction of modified system (system with constraints)
352.         """
353.         try:
354.             new_direction = sum(np.where(self.direction_map == direction))
355.             return int(new_direction)
356.         except TypeError:
357.             print('!!! Wrong direction, perhaps already used by a constraint !!!')
358.
359.     def draw_problem(self):
360.         """
361.         Plots system prepared for simulation
362.         :return:
363.         """
364.         fig, ax = plt.subplots()
365.         ax.axis('equal')
366.         for element in self.element_list:
367.             start = self.node_coordinates[str(element.from_point)]
368.             end = [start[0] + element.length * np.cos(element.angle),
369.                    start[1] + element.length * np.sin(element.angle)]
370.             self.node_coordinates[str(element.to_point)] = end
371.             ax.plot([start[0], end[0]], [start[1], end[1]])
372.             plt.text(end[0], end[1], str(element.to_point))
373.
374.         element_fu = int(np.absolute(np.transpose(self.vect_fu)) @ self.direction_map //
    3)
375.         modulo_fu = int(np.absolute(np.transpose(self.vect_fu)) @ self.direction_map % 3)
376.         element_ft = int(np.transpose(self.vect_ft) @ self.direction_map // 3)
377.         modulo_ft = int(np.transpose(self.vect_ft) @ self.direction_map % 3)
378.         xt = self.node_coordinates[str(element_fu)][0]
379.         yt = self.node_coordinates[str(element_fu)][1]
380.         plt.text(xt, yt, s=f'fu_{modulo_fu}', va='top')
381.         xt = self.node_coordinates[str(element_ft)][0]
382.         yt = self.node_coordinates[str(element_ft)][1]
383.         plt.text(xt, yt, s=f'ft_{modulo_ft}', va='top', ha='right')
384.         plt.show()
385.
386.     def get_ss(self, K, M, B, fmax):
387.         """
388.         :param K: Stiffness matrix
```

```
389.         :param M: Mass matrix
390.         :param B: Damping matrix
391.         :param fmax: Maximal frequency - higher frequencies will be neglected
392.         :return: As, Bs, Cs, Ds - Returns matrices of state space system
393.         """
394.         D, V = la.eigh(K, b=M)
395.         Km = np.transpose(V) @ K @ V
396.         Bm = np.transpose(V) @ B @ V
397.         Mm = np.transpose(V) @ M @ V
398.         eig_freq = np.sqrt(D) / 2 / np.pi
399.         a_ind = np.where(eig_freq < fmax)[0]
400.         b_ind = np.where(eig_freq >= fmax)[0]
401.
402.         if isinstance(self.Ak, np.ndarray) and self.Ak.shape[0] / 2 < a_ind.shape[0]:
403.             b_ind = np.insert(b_ind, 0, a_ind[-1])
404.             a_ind = np.delete(a_ind, -1)
405.         elif isinstance(self.Ak, np.ndarray) and self.Ak.shape[0] / 2 > a_ind.shape[0]:
406.             a_ind = np.append(a_ind, b_ind[0])
407.             b_ind = np.delete(b_ind, 0)
408.
409.         print(f'vlastni cisla (Hz): {eig_freq}')
410.
411.         Kaa = Km[np.ix_(a_ind, a_ind)]
412.         Kab = Km[np.ix_(a_ind, b_ind)]
413.         Kba = Km[np.ix_(b_ind, a_ind)]
414.         Kbb = Km[np.ix_(b_ind, b_ind)]
415.
416.         Baa = Bm[np.ix_(a_ind, a_ind)]
417.         Bab = Bm[np.ix_(a_ind, b_ind)]
418.         Bba = Bm[np.ix_(b_ind, a_ind)]
419.         Bbb = Bm[np.ix_(b_ind, b_ind)]
420.
421.         Va = V[:, a_ind]
422.         Vb = V[:, b_ind]
423.
424.         zeros = np.zeros([len(K), len(a_ind)])
425.
426.         As = np.block([[np.zeros([len(Kaa), len(Kaa)]), np.eye(len(Baa))],
427.                        [-Kaa, -Baa]])
428.
429.         Bs = np.block([[np.transpose(zeros)],
430.                        [np.transpose(Va)]])
431.
432.         Cs = np.block([[Va, zeros],
433.                        [zeros, Va]])
434.
435.         Ds = np.block([[zeros],
436.                        [zeros]])  # Truncation D
437.
438.         return As, Bs, Cs, Ds
439.
440.     def rayleigh(self, K, M):
441.         """
442.         :param K: Stiffness matrix
443.         :param M: Mass matrix
444.         :return: B - Rayleigh damping matrix
445.         """
446.         B = self.a0 * K + self.a1 * M
447.
448.         return B
449.
450.     def get_all_ss(self, fmax):
451.         """
452.         Creates matrices of state space system for k and l systems
453.         :param fmax: Maximal frequency - higher frequencies will be neglected
```

```
454.         :return:
455.         """
456.         self.Ak, self.Bk, self.Ck, self.Dk = self.get_ss(self.K, self.M,
     self.rayleigh(self.K, self.M), fmax)
457.         self.Al, self.Bl, self.Cl, self.Dl = self.get_ss(self.L, self.M,
     self.rayleigh(self.L, self.M), fmax)
458.
459.     def output_y(self, s):
460.         """
461.         :param s: States of statespace system
462.         :return: y - output of SS system, isL - binary value if nonlinearity is activated
463.         """
464.         y = self.Ck @ s  # Dk = 0
465.         isL = False
466.
467.         Cl = self.Cl
468.         try:
469.             if self.vect_fu @ y[:len(y) // 2] > 0:  # If coordinate exceeds limit,
     nonlinearity is activated
470.                 y = Cl @ s
471.                 isL = True  # Nonlinearity is activated
472.         except:
473.             print('problems with matmul')
474.
475.         return y, isL
476.
477.     def start_integration(self, t_stop, freq):
478.         """
479.         :param t_stop: Maximal time of integration
480.         :param freq: Frequency of simulated measurement
481.         :return:
482.         """
483.         fmax = freq / 2
484.
485.         self.get_all_ss(fmax)  # Computes state space matrices with neglected modes
486.
487.         s0 = np.array(np.zeros(len(self.Ak)))   # Initial condition
488.         t_start = 0
489.         self.step = 1 / freq
490.         t = np.arange(t_start, t_stop, step=self.step)
491.
492.         def get_B(s):
493.             """
494.             :param s: States of statespace system
495.             :return: Correct matrix B in respect to nonlinearity
496.             """
497.             y, isL = self.output_y(s)
498.             if isL:
499.                 B = self.Bl
500.             else:
501.                 B = self.Bk
502.             return B
503.
504.         def jacobian(t, s):
505.             """
506.             :param t: Unused variable
507.             :param s: States of statespace system
508.             :return: Correct matrix J - jacobian in respect to nonlinearity
509.             """
510.             y, isL = self.output_y(s)
511.             if isL:
512.                 J = self.Al
513.             else:
514.                 J = self.Ak
515.             return J
```

```
516.
517.         def int_problem(t, s):
518.             """
519.             :param t: Unused variable
520.             :param s: States of statespace system
521.             :return: s_dot - derivation of statespace system states
522.             """
523.             J = jacobian(0, s)
524.             B = get_B(s)
525.             f = self.vect_ft * self.ft(t)
526.             s_dot = J @ s + B @ f
527.             print(t)
528.             return s_dot
529.
530.         self.r = ode.solve_ivp(int_problem, [t[0], t[-1]], s0, method='BDF', t_eval=t,
    max_step=self.step / 4,
531.                                r_tol=1e-8, a_tol=1e-30, jac=jacobian)
532.
533.         self.post_simulation_integration(t)
534.
535.     def start_integration_csv(self, t_stop, freq, measured_directions):
536.         """
537.         :param t_stop: Maximal time of integration
538.         :param freq: Frequency of simulated measurement
539.         :param measured_directions: Directions whose speed should be recorded
540.         :return: result - measured speeds and impact force - list
541.         """
542.         fmax = freq / 2
543.         timlim = time.perf_counter() + 120
544.         e_tol = 10
545.
546.         self.get_all_ss(fmax)  # Spocita matice stavoveho systemu
547.
548.         s0 = np.array(np.zeros(len(self.Ak)))
549.         t_start = 0
550.         self.step = 1 / freq
551.         t = np.arange(t_start, t_stop, step=self.step)  # Vector of timesteps


552.         def get_B(s):
553.             """
554.             :param s: States of statespace system
555.             :return: Correct matrix B in respect to nonlinearity
556.             """
557.             y, isL = self.output_y(s)
558.             if isL:
559.                 B = self.Bl
560.             else:
561.                 B = self.Bk
562.             return B
563.
564.         def jacobian(t, s):
565.             """
566.             :param t: Unused variable
567.             :param s: States of statespace system
568.             :return: Correct matrix J - jacobian in respect to nonlinearity
569.             """
570.             y, isL = self.output_y(s)
571.             if isL:
572.                 J = self.Al
573.             else:
574.                 J = self.Ak
575.             return J
576.
577.         def int_problem(t, s):
```

```
578.             """
579.             :param t: Actual time of simulation
580.             :param s: States of statespace system
581.             :return: s_dot - derivation of statespace system states
582.             """
583.             J = jacobian(0, s)
584.             B = get_B(s)
585.             f = self.vect_ft * self.ft(t)
586.             s_dot = J @ s + B @ f
587.             print(t)
588.             if time.perf_counter() > timlim:
589.                 print('____MAXTIME EXCEEDED____')
590.                 return np.zeros(s_dot.shape)    # If maxtime is exceeded, returns zeros
591.             return s_dot
592.
593.         self.r = ode.solve_ivp(int_problem, [t[0], t[-1]], s0, method='RK45', t_eval=t,
    max_step=self.step / 4, r_tol=1e-8, a_tol=1e-30, jac=jacobian)
594.
595.         self.post_simulation_integration(t)
596.         e_diff = self.energy_difference(0.05, t_stop, freq)
597.
598.         if e_diff < e_tol:
599.             result = self.get_result(measured_directions)
600.             print(f'Percentage difference of overall energy in system:{e_diff}')
601.             return result
602.         else:
603.             print('____overall energy of the system changes in time, simulation result
    discarded____')
604.             print(f'Percentage difference of overall energy in system:{e_diff}')
605.             return None, None
606.
607.     def post_simulation_integration(self, t):
608.         """
609.         Computes values of Ep, Ek, Er energy as it changes in time
610.         :param t: Time vector
611.         :return:
612.         """
613.         self.Er = np.array([0.])
614.
615.         for i in range(0, t.size):
616.             y, isL = self.output_y(self.r.y[:, i])
617.             self.f = np.append(self.f, self.ft(t[i]))
618.             u = y[:(len(y) // 2)]
619.             u_dot = y[len(y) // 2:]
620.             if i == 0:
621.                 self.u = [u]
622.                 self.u_dot = [u]
623.             else:
624.                 self.u = np.append(self.u, [u], axis=0)
625.                 self.u_dot = np.append(self.u_dot, [u_dot], axis=0)
626.             if isL:
627.                 K = self.L
628.                 B = self.rayleigh(self.L, self.M)
629.             else:
630.                 K = self.K
631.                 B = self.rayleigh(self.K, self.M)
632.             self.Ep = np.append(self.Ep, 0.5 * u.reshape(1, len(u)) @ K @
    u.reshape(len(u), 1))
633.             self.Ek = np.append(self.Ek, 0.5 * u_dot.reshape(1, len(u_dot)) @ self.M @
    u_dot.reshape(len(u_dot), 1))
634.             # Integration of rayleigh damping loss
635.             self.Er = np.append(self.Er, self.Er[-1] +
636.                                 u_dot.reshape(1, len(u_dot)) @ B @
    u_dot.reshape(len(u_dot), 1) * self.step)
637.         self.u = self.u.T
```

```
638.          self.u_dot = self.u_dot.T
639.
640.          self.Er = np.delete(self.Er, 0)
641.
642.      def energy_difference(self, t1, t2, freq):
643.          """
644.          :param t1: Time of first energy check
645.          :param t2: Time of second energy check
646.          :param freq: Frequency of measurement
647.          :return: e_diff - energy difference between times t1 and t2
648.          """
649.          p1 = int(t1 * freq)
650.          p2 = int(t2 * freq) - 1
651.          E1 = self.Ep[p1] + self.Ek[p1] + self.Er[p1]
652.          E2 = self.Ep[p2] + self.Ek[p2] + self.Er[p2]
653.          e_diff = np.abs(E1 - E2) / E1 * 100
654.          return e_diff
655.
656.      def get_result(self, measured_directions):
657.          """
658.          :param measured_directions: Directions in original unconstrained system, whose
      speed needs to be measured
659.          :return: speeds - measured speed of nodes, forces - impact force repeated in rows
      corresponding to the number of
660.          measured speeds
661.          """
662.
663.          speeds = np.empty(shape=(0, len(self.u_dot[0])), dtype='float32')
664.          forces = np.empty(shape=(0, len(self.u_dot[0])), dtype='float32')
665.
666.          for direction in measured_directions:
667.              new_dir = self.new_direction(direction)
668.              row = self.u_dot[new_dir]
669.              row = np.expand_dims(row.astype('float32'), axis=0)
670.              speeds = np.append(speeds, row, axis=0)
671.              forces = np.append(forces, self.f.reshape((1, len(self.u_dot[0]))), axis=0)
672.
673.          return speeds, forces
```

9.4 constants (Python)

```
1.   E = 2.1 * 10 ** 5 * 10 ** 6  # Young modulus Pa
2.   ni = 0.3    # Poisson number
3.   ro = 7800  # Density kg/m3
4.   width = 0.03  # m
5.   height = 0.04  # m
6.   a0 = 1e-5   # damping coef
7.   a1 = 1e-5   # damping coef
8.   three_sigma_percent = 10  # 99 percent of values is in this range
9.   max_impact_force = 2000  # N
10.  impact_duration = 0.001  # s
11.  spring_stiffness = 3e13     # N/m
```

10 Identifikace filtrem

10.1 main (Python)

```
import numpy as np
import pandas as pd
import pygad
import L_M_Optimization as L_M
import matplotlib.pyplot as plt
import pickle
from scipy.fft import fft, fftfreq, ifft

# Initialization

TestForce = np.array(pd.read_csv('TestForce.csv', header=None))
TestVelocity = np.array(pd.read_csv('TestVelocity.csv', header=None))
TestForce = np.insert(TestForce, 0, 0., axis=1)
TestVelocity = np.insert(TestVelocity, 0, 0., axis=1)
K_mod = np.array(pd.read_csv('K_mod.csv', header=None))
B_mod = np.array(pd.read_csv('B_mod.csv', header=None))
f = TestForce.reshape(TestForce.size)[0:1000]
yt = TestVelocity.reshape(TestVelocity.size)[0:1000]

freq = 1024     # Simulated

impact_fft = fft(yt)
fr = fftfreq(len(yt), 1/freq)
fig, ax21 = plt.subplots(1, 1)
ax21.plot(fr, np.abs(impact_fft))
ax21.set_yscale('log')
ax21.grid()
plt.show()

K = K_mod[0]
B = B_mod[0]
V = [1, 0.5]

model = L_M.L_M_Nonlin(K, B, V, freq, f, yt)
model.start(epoch=20, plot=True)
model.reset_init()
linear_Error2 = model.Err2

def fitness_func(sol):
    """
    :param sol: Vector of nonlinear parameters - genes
    :return: fitness - fitness function
    """
    model.start([sol[0], sol[1]], sol[2], epoch=5)
    out = model.Err2
    if np.isnan(out):
        out = 1
    fitness = 1.0 / out
    return fitness

# Definition of genetic algorithm object

fitness_function = fitness_func
num_generations = 20
num_parents_mating = 10
sol_per_pop = 20    # Number of individuals in population
num_genes = 3   # Length of inputs to model
init_range_low = -1
```

```
59.  init_range_high = 1
60.  parent_selection_type = "sss"
61.  keep_parents = 1
62.  random_mutation_min_val = -0.1
63.  random_mutation_max_val = 0.1
64.  crossover_type = "uniform"
65.  mutation_type = "random"
66.  mutation_probability = 0.9
67.  gene_space = [None, None, {'low': -1, 'high': 1}]
68.
69.  ga_instance = pygad.GA(num_generations=num_generations,
70.                           num_parents_mating=num_parents_mating,
71.                           fitness_func=fitness_function,
72.                           sol_per_pop=sol_per_pop,
73.                           num_genes=num_genes,
74.                           init_range_low=init_range_low,
75.                           init_range_high=init_range_high,
76.                           parent_selection_type=parent_selection_type,
77.                           keep_parents=keep_parents,
78.                           crossover_type=crossover_type,
79.                           mutation_type=mutation_type,
80.                           save_solutions=True,
81.                           gene_space=gene_space,
82.                           save_best_solutions=True,
83.                           random_mutation_min_val=random_mutation_min_val,
84.                           random_mutation_max_val=random_mutation_max_val,
85.                           mutation_probability=mutation_probability)
86.
87.  ga_instance.run()
88.
89.  # Plot results
90.
91.  ga_instance.plot_fitness()
92.  ga_instance.plot_genes()
93.  ga_instance.plot_new_solution_rate()
94.
95.  solution, solution_fitness, solution_idx = ga_instance.best_solution()
96.  print("Parameters of the best solution : {solution}".format(solution=solution))
97.  print("Fitness value of the best solution =
     {solution_fitness}".format(solution_fitness=solution_fitness))
98.  print("Err2 of the best solution =
     {solution_fitness}".format(solution_fitness=1/solution_fitness))
99.  print("Err2 of the linear model =
     {solution_fitness}".format(solution_fitness=linear_Error2))
100.
101. with open('last_result.pickle', 'wb') as handle:    # Saving results
102.     pickle.dump(ga_instance, handle, protocol=pickle.HIGHEST_PROTOCOL)
103.
104. fig, [ax1, ax2, ax3] = plt.subplots(3, 1)
105. ax1.scatter(np.array(ga_instance.solutions)[:, 0], -
     1/np.array(ga_instance.solutions_fitness))
106. plt.ylabel('fitness')
107. ax1.set_ylim([-0.0006, 0])
108. ax2.scatter(np.array(ga_instance.solutions)[:, 1], -
     1/np.array(ga_instance.solutions_fitness))
109. ax2.set_ylim([-0.0006, 0])
110. ax3.scatter(np.array(ga_instance.solutions)[:, 2], -
     1/np.array(ga_instance.solutions_fitness))
111. ax3.set_ylim([-0.0006, 0])
112. ax1.set_title('gene 1')
113. ax2.set_title('gene 2')
114. ax3.set_title('gene 3')
115. ax1.grid()
116. ax2.grid()
117. ax3.grid()
```

```
118. plt.show()
119.
120. fig, ax1 = plt.subplots(1,1)
121. ax1.scatter(np.arange(len(ga_instance.solutions_fitness)), -
     1/np.array(ga_instance.solutions_fitness))
122. ax1.set_title('Solution fitness')
123. plt.ylim([-0.0006, 0])
124. plt.show()
```

10.2 L__M__Optimization (Python)

```
1.   import os
2.   import tensorflow as tf
3.   os.environ['CUDA_VISIBLE_DEVICES'] = '-1'  # Forces TF to use CPU as it runs faster for
     this computation
4.   import numpy as np
5.   import matplotlib.pyplot as plt
6.   from scipy import signal
7.   import time
8.
9.   # L-M Optimization
10.
11.  def get_MNO(parameters):
12.      """
13.      :param parameters: Linear parameters of filter
14.      :return: M, N, O - matrices of discrete state system
15.      """
16.      p = parameters.size // 7
17.      in1 = np.arange(0, p)
18.      in2 = np.arange(p, 2 * p)
19.      M = np.zeros([2 * p, 2 * p])
20.      M[in1, in1] = parameters[0, in1]
21.      M[in1, in2] = parameters[0, in1 + p]
22.      M[in2, in1] = parameters[0, in1 + 2 * p]
23.      M[in2, in2] = parameters[0, in1 + 3 * p]
24.
25.      N = np.zeros([2 * p])
26.      N[in1] = parameters[0, in1 + 4 * p]
27.      N[in2] = parameters[0, in1 + 5 * p]
28.
29.      O = np.zeros([2 * p])
30.      O[in1] = parameters[0, in1 + 6 * p]
31.
32.      M = tf.convert_to_tensor(M, tf.float32)
33.      N = tf.convert_to_tensor(N, tf.float32)
34.      O = tf.convert_to_tensor(O, tf.float32)
35.
36.      return M, N, O
37.
38.  def get_parameters(M, N, O):
39.      """
40.      :param M: Matrix of discrete state system
41.      :param N: Matrix of discrete state system
42.      :param O: Matrix of discrete state system
43.      :return: parameters - linear parameters of filter
44.      """
45.      p = N.shape[0] // 2
46.      in1 = np.arange(0, p)
47.      in2 = np.arange(p, 2 * p)
48.      try:
49.          parameters = np.block([M.numpy()[in1, in1], M.numpy()[in1, in2], M.numpy()[in2,
     in1], M.numpy()[in2, in2],
50.                                 N.numpy().reshape([1, 2 * p]),
51.                                 O.numpy()[in1]])
```

```
52.      except:
53.          parameters = np.block([M[in1, in1], M[in1, in2], M[in2, in1], M[in2, in2],
54.                                 N.reshape([1, 2 * p]),
55.                                 O[0, in1]])
56.      return parameters
57.
58.  def get_DELTA(p):
59.      """
60.      :param p: Size of system
61.      :return: DELTA - 4D tensor
62.      """
63.      l = int(2 * p)
64.      zeros = tf.zeros([l, l, l, l], tf.float32)
65.      DELTA = tf.Variable(initial_value=zeros)
66.      for i in np.arange(l):
67.          for j in np.arange(l):
68.              for k in np.arange(l):
69.                  for m in np.arange(l):
70.                      if i == k and j == m:
71.                          DELTA[i, j, k, m].assign(1)
72.      return DELTA
73.
74.  def get_R_mat(learning, p, parameters, dimple, old_parameters, nonlin_parameters=[0]):
75.      """
76.      :param learning: Learning rate
77.      :param p: Size of system
78.      :param parameters: Linear parameters of filter
79.      :param dimple: Parameter of dimple function
80.      :param old_parameters: Old linear parameters of filter
81.      :param nonlin_parameters: Nonlinear parameters of filter
82.      :return: R_mat - Matrix parameter of L-M algorithm
83.      """
84.      if nonlin_parameters != [0]:
85.          R_mat = np.zeros([8 * p, 8 * p])
86.      else:
87.          R_mat = np.zeros([7 * p, 7 * p])
88.      k = 0
89.      for param in parameters[0, :]:
90.          R_mat[k, k] = (1 + dimple * (param - old_parameters[0, k]) * (param -
     old_parameters[0, k]) / param / param) / learning[k]  # Dimple function
91.          k += 1
92.
93.      if nonlin_parameters != [0]:
94.          for param in nonlin_parameters:
95.              R_mat[k, k] = 1 / learning[k]
96.              k += 1
97.
98.      return R_mat
99.
100. def get_discrete(B, K, V, p, dt):
101.     """
102.     :param B: Damping matrix B
103.     :param K: Stiffness matrix
104.     :param V: Modal vector matrix
105.     :param p: Size of system
106.     :param dt: Timestep
107.     :return: M, N, O, P, sys_d - matrices of discrete state system and discrete system
108.     """
109.     # Continuous system
110.     As = np.block([[-B, -K], [np.eye(p), np.zeros([p, p])]])
111.     Bs = np.block([[V.reshape([p, 1])], [np.zeros([p, 1])]])
112.     Cs = Bs.reshape([1, 2 * p])
113.     Ds = np.array([0.])
114.
115.     # Discrete time system
```

```
116.     sys_d = signal.cont2discrete((As, Bs, Cs, Ds), dt, method='foh')
117.
118.     M = sys_d[0]
119.     N = sys_d[1]
120.     O = sys_d[2]
121.     P = sys_d[3]
122.
123.     return M, N, O, P, sys_d
124.
125. def get_M2(lin_parameters, nonlin_M_param) -> np.ndarray:
126.     """
127.         :param lin_parameters: Vector of parameters of M N O discrete system
128.         :param nonlin_M_param: Multiplier of the stiffness
129.         :returns: Matrix of discrete state space system with modified stiffness
130.     """
131.     p = lin_parameters.size // 7
132.     in1 = np.arange(0, p)
133.     in2 = np.arange(p, 2 * p)
134.     M2 = np.zeros([2 * p, 2 * p])
135.     M2[in1, in1] = lin_parameters[0, in1]
136.     M2[in1, in2] = lin_parameters[0, in1 + p] * nonlin_M_param  # Stiffness
137.     M2[in2, in1] = lin_parameters[0, in1 + 2 * p]
138.     M2[in2, in2] = lin_parameters[0, in1 + 3 * p]
139.
140.     M2 = tf.convert_to_tensor(M2, tf.float32)
141.
142.     return M2
143.
144. def get_right_M(M1, M2, s_old, nonlin_R) -> np.ndarray:
145.     """
146.     :param M1: Default M of state space M N O
147.     :param M2: Alternative M - causing nonlinearity
148.     :param s_old: last state of ss system, [speed, position]
149.     :param nonlin_R: Transformation vector to transform from modal coordinates to
     discontinuity coordinates
150.     :return: matrix M with respect to actual state of the system
151.     """
152.
153.     R = np.sum(s_old * nonlin_R)
154.
155.     if R >= 0:
156.         M = M1
157.     else:
158.         M = M2
159.
160.     return M
161.
162.
163. class L_M_Nonlin:
164.
165.     def __init__(self, K_vect: list, B_vect: list, V_vect: list, freq: int, f, yt):
166.         """
167.         :param K_vect: stiffness matrix first guess in vector shape
168.         :param B_vect: damping matrix first guess in vector shape
169.         :param V_vect: modal vector
170.         :param freq: frequency of measurement
171.         :param f: measured force input
172.         :param yt: measured speed output
173.         """
174.         self.B = np.diag(B_vect)
175.         self.K = np.diag(K_vect)
176.         self.V = np.array(V_vect)
177.         self.p = len(K_vect)  # Degrees of freedom
178.         dt = 1 / freq
```

```
179.         self.M_start, self.N_start, self.O_start, self.P_start, self.sys_d =
     get_discrete(self.B, self.K, self.V, self.p, dt)
180.         self.freq = freq  # Frequency of measured signals
181.         self.f = f  # Measured force input in time
182.         self.yt = yt  # Measured speed output in time
183.         self.epoch = 5  # Number of learning epochs
184.         batch = yt.size * dt  # Seconds
185.         self.dimple = 0  # Multiplicator of dimple function
186.         self.learning = np.ones(self.p * 7) * 5  # How fast L-M learns (10 was ok)
187.         self.learning_improvement = 1  # For simulated input 10 was ok
188.         self.t = np.arange(0, batch, dt)  # Time during measurement
189.         self.DELTA = get_DELTA(self.p)
190.         self.Err2 = None
191.         self.lin_parameters = get_parameters(self.M_start, self.N_start,
192.                                              self.O_start)  # Parameters that change
     after optimization
193.         self.called_times = 0
194.
195.         print('L_M_Nonlin instance created')
196.
197.     def reset_init(self):
198.         """
199.         Sets latest optimized linear parameters as initial linear parameters for next
     optimization
200.
201.         """
202.
203.         self.M_start, self.N_start, self.O_start = get_MNO(self.lin_parameters)
204.
205.     def start(self, nonlin_M_params=None, nonlin_R_params=None, plot=False, epoch=5):
206.         """
207.         starts L-M Optimization of parameters
208.         :param nonlin_M_params: Multiplier of the stiffness to create nonlinear system.
     <-1;1>, to become <0.999;1.001>
209.         :param nonlin_R_params: For n dof system, n-1 vector to transform from modal
     coordinates to the ones in which
210.         discontinuity occurs. <-1;1>, to become <0;1>
211.         :return:
212.         """
213.         self.called_times += 1
214.         tim = time.time()
215.         self.epoch = epoch
216.
217.         if nonlin_M_params is None or nonlin_R_params is None:
218.             nonlin_M_params = np.zeros(self.p)
219.             nonlin_R = np.zeros(self.p - 1)
220.         else:
221.             nonlin_R = np.array(nonlin_R_params)
222.
223.         nonlin_M_params = np.array(nonlin_M_params) * 0.001 + 1  # Normalization
224.         nonlin_R = nonlin_R * 0.5 + 0.5  # Normalization
225.         residue = np.array([1 - np.sum(nonlin_R)])
226.         nonlin_R = np.block([np.zeros(np.size(nonlin_R) + 1), nonlin_R, residue])
227.
228.         # Step simulations
229.         if plot:
230.             td, yd, x = signal.dlsim(self.sys_d, self.f, t=self.t)
231.
232.         # Initial parameters acquisition
233.         parameters = get_parameters(self.M_start, self.N_start, self.O_start)
234.
235.         # Iteration
236.         print('L_M Iteration started')
237.         learning = self.learning
238.         Err = np.zeros(self.epoch)
```

```
239.         Err2 = np.zeros(self.epoch)
240.         old_parameters = np.zeros([self.epoch, self.p * 7])
241.
242.         for m in np.arange(0, self.epoch):  # Iterating for each epoch
243.             M1, N, O = get_MNO(parameters)
244.             M2 = get_M2(parameters, nonlin_M_params)
245.             s_old = tf.zeros([2 * self.p], tf.float32)
246.             s_dN_old = tf.zeros([2 * self.p, 2 * self.p], tf.float32)
247.             s_dM_old = tf.zeros([2 * self.p, 2 * self.p, 2 * self.p], tf.float32)
248.             f_old = 0
249.             y_sim = np.zeros(self.t.shape)
250.             e = np.zeros(self.t.shape)
251.             J = np.zeros([self.t.shape[0], self.p * 7])
252.             old_parameters[m, :] = parameters
253.
254.             for k in np.arange(y_sim.shape[0]):  # Iterating for each time step
255.                 M = get_right_M(M1, M2, s_old, nonlin_R)
256.                 s = tf.tensordot(M, s_old, [[1], [0]]) + N * f_old
257.                 y_sim[k] = tf.tensordot(O, s, [[0], [0]])
258.                 s_dN = tf.tensordot(M, s_dN_old, [[1], [0]]) +
    tf.convert_to_tensor(tf.eye(2 * self.p) * f_old, tf.float32)
259.                 s_dM = tf.tensordot(M, s_dM_old, [[1], [0]]) + tf.tensordot(s_old,
    self.DELTA, [[0], [1]])
260.                 y_dO = s
261.                 y_dN = tf.tensordot(O, s_dN, [[0], [0]])
262.                 y_dM = tf.tensordot(O, s_dM, [[0], [0]])
263.                 J[k, :] = np.block([get_parameters(y_dM, y_dN, y_dO)])
264.                 e[k] = self.yt[k] - y_sim[k]
265.                 s_old = s
266.                 s_dN_old = s_dN
267.                 s_dM_old = s_dM
268.                 f_old = self.f[k]
269.             e[0] = 0
270.             e[1] = 0
271.             Err[m] = np.mean(np.abs(e))
272.             Err2[m] = np.mean(e * e)
273.
274.             R_mat = get_R_mat(learning, self.p, parameters, self.dimple, old_parameters)
275.             dw = np.linalg.solve((np.transpose(J) @ J + R_mat), np.transpose(J) @
    np.transpose(e))
276.             parameters = parameters + dw
277.             print(f'epoch:{m}   diff Err: {Err[m - 1] - Err[m]}   diff Err2: {Err2[m - 1]
    - Err2[m]}')
278.             learning = learning * self.learning_improvement
279.
280.             if plot:
281.                 fig, ax = plt.subplots()
282.                 ax.plot(td, yd, linestyle='dashed', label='discretized')
283.                 ax.plot(self.t, self.yt, label='original')
284.                 ax.plot(self.t, y_sim, linestyle='dotted', label='LM output')
285.                 ax.legend()
286.                 plt.show(block=False)
287.                 plt.pause(1)
288.                 plt.close()
289.
290.         self.Err2 = Err2[-1]
291.         self.lin_parameters = parameters
292.         print(f'Err2 at the end: {self.Err2}')
293.         elapsed = time.time() - tim
294.         print(f'elapsed: {elapsed}')
295.         print('')
296.         print(f'Solution number: {self.called_times} FINISHED')
297.         print('')
```

10.3 simulated_data_processing (Matlab)

```
clear all
close all

%% For lin system

lin_vel = importdata('C:\Users\vojta\Documents\GitHub\MKP_Diplomka\test_unbroken.csv');
lin_force =
importdata('C:\Users\vojta\Documents\GitHub\MKP_Diplomka\test_unbroken_force.csv');

okenka = true;
nonlin = false;
position = 24;
Fs = 1024;      %Hz

vel_slice = lin_vel(position,:);
force_slice = lin_force(position,:);
disp('computing lin system...')

%% Saving data for filter

writematrix(force_slice, 'C:\Users\vojta\Documents\GitHub\filter\TestForce.csv')
writematrix(vel_slice, 'C:\Users\vojta\Documents\GitHub\filter\TestVelocity.csv')

%% Identification using matlab identification toolbox

test = iddata(vel_slice', force_slice', 1/Fs);
nx = 1:10;
sys = n4sid(test,nx);

mnum = 4;
force_tr = force_slice';
vel_tr = vel_slice';
winlen = size(force_tr,1);
[frf_mat, f_mat, coh] = modalfrf(force_tr(:),vel_tr(:),Fs,winlen,'Sensor','vel');

figure(180)
modalfit(frf_mat,f_mat,Fs,mnum,'FitMethod','lsrf','FreqRange',[5 500]);
[fn,dr,ms,ofrf] = modalfit(frf_mat,f_mat,Fs,mnum,'FitMethod','lsrf','FreqRange',[5 500]);

%% Recalculate

Omeg = fn*2*pi
K_mod = (fn*2*pi).^2
B_mod = dr.*Omeg*2

writematrix(K_mod', 'C:\Users\vojta\Documents\GitHub\filter\K_mod.csv')
writematrix(B_mod', 'C:\Users\vojta\Documents\GitHub\filter\B_mod.csv')

%% Again for nonlin system

clearvars -except fn position Fs

nonlin_vel = importdata('C:\Users\vojta\Documents\GitHub\MKP_Diplomka\test_broken.csv');
nonlin_force =
importdata('C:\Users\vojta\Documents\GitHub\MKP_Diplomka\test_broken_force.csv');

vel_slice = nonlin_vel(position,:);
force_slice = nonlin_force(position,:);
disp('computing nonlin system...')

%% Saving data for filter

```

```
writematrix(force_slice, 'C:\Users\vojta\Documents\GitHub\filter\TestForce_nonlin.csv')
writematrix(vel_slice, 'C:\Users\vojta\Documents\GitHub\filter\TestVelocity_nonlin.csv')

%% Identification using matlab identification toolbox

test = iddata(vel_slice', force_slice', 1/Fs);
nx = 1:10;
sys = n4sid(test,nx);

mnum = 4;
force_tr = force_slice';
vel_tr = vel_slice';
winlen = size(force_tr,1);
[frf_mat, f_mat, coh] = modalfrf(force_tr(:),vel_tr(:),Fs,winlen,'Sensor','vel');

figure(280)
modalfit(frf_mat,f_mat,Fs,mnum,'FitMethod','lsrf','FreqRange',[5 500]);
[fn,dr,ms,ofrf] = modalfit(frf_mat,f_mat,Fs,mnum,'FitMethod','lsrf','FreqRange',[5 500]);

%% Recalculate

Omeg_nonlin = fn*2*pi
K_mod_nonlin = (fn*2*pi).^2
B_mod_nonlin = dr.*Omeg_nonlin*2

writematrix(K_mod_nonlin', 'C:\Users\vojta\Documents\GitHub\filter\K_mod_nonlin.csv')
writematrix(B_mod_nonlin', 'C:\Users\vojta\Documents\GitHub\filter\B_mod_nonlin.csv')


```

11 Strojové učení

11.1 single_input_model (Python)

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import tensorflow as tf
from tensorflow import keras
from keras import layers
import time
import statistics_tools as stat
from keras.utils import plot_model
from keras.models import Model

## Declaring constants and functions

data_size = 1024
num_samples = 4395
num_ver_samples = 94
path_to_data = 'input_data/*/*.csv'
path_to_ver_data = 'verification_input_data/*/*.csv'
AUTOTUNE = tf.data.experimental.AUTOTUNE
BATCH_SIZE = 1000
train_size = int(num_samples * 2 * 0.9)


def get_label(file_path):
    """
    :param file_path: Path to the file with data
    :return: Label of data (broken, unbroken, reference) = (1, 0, 2)
    """
    import os
    if tf.strings.split(file_path, os.path.sep)[-2] == 'broken':
        return tf.constant(1, dtype=tf.int32)
    elif tf.strings.split(file_path, os.path.sep)[-2] == 'unbroken':
        return tf.constant(0, dtype=tf.int32)
    elif tf.strings.split(file_path, os.path.sep)[-2] == 'reference':
        return tf.constant(2, dtype=tf.int32)
    else:
        print(file_path)
        print(tf.strings.split(file_path, os.path.sep))
        print('ERROR FINDING LABEL')
        return tf.constant(0, dtype=tf.int32)


def read_files(path):
    """
    :param path: Path to the file with data
    :return: Input for DNN tested_dataseries, label
    """
    file = tf.io.read_file(path)
    file5 = tf.strings.split(
        file, sep='\r', maxsplit=-1, name=None
    )[0]

    dataseries = tf.io.decode_csv(
        file5,
        [float()] * data_size,
        field_delim=',',
        use_quote_delim=True,
        na_value='',
```

```
59.          select_cols=None,
60.          name=None
61.      )
62.
63.      dataseries2 = tf.stack(
64.          dataseries, axis=0, name='stack'
65.      )
66.      label = get_label(path)
67.      return dataseries2, label
68.
69.  ## Create dataset
70.
71.  tim = time.time()
72.  print('creating dataset started')
73.  print(time.time() - tim)
74.  tim = time.time()
75.
76.  dataset = tf.data.Dataset.list_files(path_to_data, shuffle=20)
77.  dataset = dataset.filter(lambda x: get_label(x) < 2)
78.
79.  train_ds = dataset.take(train_size)
80.
81.  train_ds = train_ds.filter(lambda x: get_label(x) < 2)
82.  train_ds = train_ds.map(read_files, num_parallel_calls=AUTOTUNE)
83.  train_ds = train_ds.cache()
84.  train_ds = train_ds.batch(BATCH_SIZE)
85.  train_ds = train_ds.prefetch(AUTOTUNE)
86.
87.  ver_ds = tf.data.Dataset.list_files(path_to_ver_data, shuffle=20)
88.  ver_ds = ver_ds.filter(lambda x: get_label(x) < 2)
89.  ver_ds = ver_ds.map(read_files, num_parallel_calls=AUTOTUNE)
90.  ver_ds = ver_ds.cache()
91.  ver_ds = ver_ds.batch(BATCH_SIZE)
92.  ver_ds = ver_ds.prefetch(AUTOTUNE)
93.
94.  ## Plot some train data
95.
96.  plot_ds = train_ds.take(1)
97.  fig, (axes) = plt.subplots(3, 2)
98.  for row in plot_ds:
99.      data = row[0]
100.     label = row[1]
101.
102. i = 0
103. for axes_hor in axes:
104.     for ax in axes_hor:
105.         ax.plot(np.arange(0, data_size), data[i, :])
106.         if label[i] == 1:
107.             textlabel = 'broken'
108.         else:
109.             textlabel = 'unbroken'
110.         ax.title.set_text(f'train_data, label = {label[i]},  ' + textlabel)
111.         i += 1
112. plt.show(block=False)
113. plt.pause(1)
114. plt.close()
115.
116.
117. ##  Plot some verification data
118.
119. plot_ds = ver_ds.take(1)
120. fig, (axes) = plt.subplots(3, 2)
121. for row in plot_ds:
122.     data = row[0]
123.     label = row[1]
```

```
124.
125. i = 0
126. for axes_hor in axes:
127.     for ax in axes_hor:
128.         ax.plot(np.arange(0, data_size), data[i, :])
129.         if label[i] == 1:
130.             textlabel = 'broken'
131.         else:
132.             textlabel = 'unbroken'
133.         ax.title.set_text(f'verification_data, label = {label[i]},  ' + textlabel)
134.         i += 1
135. plt.show(block=False)
136. plt.pause(1)
137. plt.close()
138.
139. print('creating dataset finished')
140. print(time.time() - tim)
141. tim = time.time()
142.
143. ## Buildind model
144.
145. input_layer = layers.Input((data_size, 1))
146.
147. conv1 = layers.SeparableConv1D(filters=64, kernel_size=7, padding="same",
     activation='relu')(input_layer)
148. conv1 = layers.MaxPooling1D(5)(conv1)
149. conv1 = layers.LayerNormalization()(conv1)
150.
151. conv2 = layers.SeparableConv1D(filters=64, kernel_size=7, padding="same",
     activation='relu')(conv1)
152. conv2 = layers.MaxPooling1D(5)(conv2)
153. conv2 = keras.layers.LayerNormalization()(conv2)
154.
155. conv3 = layers.SeparableConv1D(filters=32, kernel_size=7, padding="same",
     activation='relu')(conv2)
156. conv3 = layers.MaxPooling1D(5)(conv3)
157. conv3 = keras.layers.LayerNormalization()(conv3)
158.
159. gap = layers.GlobalMaxPooling1D()(conv2)
160.
161. output_layer = keras.layers.Dense(1, activation="sigmoid")(gap)
162.
163. model = Model(input_layer, output_layer)
164.
165. model.summary()
166. plot_model(model, to_file='model.png', show_shapes=True)
167.
168. callbacks_list = [tf.keras.callbacks.ModelCheckpoint(filepath='last_model_checkpoint.h5',
169.                                                     monitor='val_accuracy',
170.                                                     mode='max',
171.                                                     save_best_only=True),
172.                    tf.keras.callbacks.ReduceLROnPlateau(monitor='loss',
173.                                                         factor=0.5,
174.                                                         patience=5,
175.                                                         verbose=0,
176.                                                         mode='auto',
177.                                                         min_delta=0.0001,
178.                                                         cooldown=0,
179.                                                         min_lr=1e-20),
180.                    tf.keras.callbacks.EarlyStopping(monitor='loss',
181.                                                     patience=50)
182.                    ]
183.
184. model.compile(
185.     optimizer=keras.optimizers.Adam(learning_rate=1e-3),
```

```
186.     loss='binary_crossentropy',
187.     metrics=['accuracy', 'TrueNegatives', 'TruePositives', 'FalseNegatives',
     'FalsePositives']
188. )
189.
190. history = model.fit(train_ds, epochs=2000, verbose=2, validation_data=ver_ds,
     callbacks=callbacks_list)
191. print('Test:')
192. model.evaluate(ver_ds, verbose=2)
193. predict = model.predict(ver_ds, verbose=2)
194.
195. true_labels = np.concatenate([y for x, y in ver_ds], axis=0)
196.
197. ##  Plot results
198.
199. loss_values = history.history['loss']
200. validation_loss = history.history['val_loss']
201. accuracy = history.history['accuracy']
202. validation_accuracy = history.history['val_accuracy']
203. epochs = range(1, len(loss_values) + 1)
204.
205. fig, (ax1, ax2) = plt.subplots(2, 1)
206.
207. ax1.plot(epochs, loss_values, 'bo', label='Training loss')
208. ax1.plot(epochs, validation_loss, 'r+', label='Validation loss')
209. ax1.title.set_text('Training and validation loss')
210. ax1.set_xlabel('Epochs')
211. ax1.set_ylabel('Loss')
212. ax1.legend()
213.
214. ax2.plot(epochs, accuracy, 'bo', label='Accuracy')
215. ax2.plot(epochs, validation_accuracy, 'r+', label='Validation accuracy')
216. ax2.title.set_text('Training and validation loss')
217. ax2.set_xlabel('Epochs')
218. ax2.set_ylabel('Accuracy')
219. ax2.legend()
220.
221. plt.show()
222.
223. ROC = stat.ROC(predict, true_labels)
224. ROC.plot_ROC()
```

11.2 multiple_input_model (Python)

```
1.   import numpy as np
2.   import matplotlib.pyplot as plt
3.   import csv
4.   import tensorflow as tf
5.   from tensorflow import keras
6.   from keras import layers
7.   from keras.models import Model
8.   from keras import Input
9.   from keras.utils import plot_model
10.  import time
11.  import statistics_tools as stat
12.  import pydot
13.
14.  ## Declaring constants and functions
15.
16.  data_size = 1024
17.  num_samples = 4395
18.  num_ver_samples = 94
19.  path_to_data = 'input_data/*/*.csv'
20.  path_to_ver_data = 'verification_input_data/*/*.csv'
```

```
21.
22.  AUTOTUNE = tf.data.experimental.AUTOTUNE
23.  BATCH_SIZE = 1000
24.  train_size = int(num_samples * 2 * 0.9)
25.
26.
27.  def get_label(file_path):
28.      """
29.      :param file_path: Path to the file with data
30.      :return: Label of data (broken, unbroken, reference) = (1, 0, 2)
31.      """
32.      import os
33.      if tf.strings.split(file_path, os.path.sep)[-2] == 'broken':
34.          return tf.constant(1, dtype=tf.int32)
35.      elif tf.strings.split(file_path, os.path.sep)[-2] == 'unbroken':
36.          return tf.constant(0, dtype=tf.int32)
37.      elif tf.strings.split(file_path, os.path.sep)[-2] == 'reference':
38.          return tf.constant(2, dtype=tf.int32)
39.      else:
40.          print(file_path)
41.          print(tf.strings.split(file_path, os.path.sep))
42.          print('ERROR FINDING LABEL')
43.          return tf.constant(0, dtype=tf.int32)
44.
45.
46.  def get_reference_path(tested_path):
47.      """
48.      :param tested_path: Path to the file whose status DNN should guess
49.      :return: Path to the reference path
50.      """
51.      ref_path = tf.strings.regex_replace(tested_path, "broken|unbroken", "reference")
52.      return ref_path
53.
54.
55.  def read_file(path):
56.      """
57.      :param path: Path to the file with data
58.      :return: Dataseries
59.      """
60.      file = tf.io.read_file(path)
61.      file = tf.strings.split(file, sep='\r', maxsplit=-1, name=None)[0]
62.
63.      dataseries = tf.io.decode_csv(
64.          file,
65.          [float()] * data_size,
66.          field_delim=',',
67.          use_quote_delim=True,
68.          na_value='',
69.          select_cols=None,
70.          name=None
71.      )
72.
73.      dataseries = tf.stack(dataseries, axis=0, name='stack')
74.
75.      return dataseries
76.
77.
78.  def map_files(tested_path):
79.      """
80.      :param tested_path: Path to the file whose status DNN should guess
81.      :return: Input for DNN [tested_dataseries, reference_dataseries], label
82.      """
83.      ref_path = get_reference_path(tested_path)
84.
85.      tested_dataseries = read_file(tested_path)
```

```
86.       reference_dataseries = read_file(ref_path)
87.       label = get_label(tested_path)
88.
89.       return [tested_dataseries, reference_dataseries], label
90.
91.
92.   ## Create dataset
93.
94.   tim = time.time()
95.   print('creating dataset started')
96.   print(time.time() - tim)
97.   tim = time.time()
98.
99.   dataset = tf.data.Dataset.list_files(path_to_data, shuffle=20)
100.  dataset = dataset.filter(lambda x: get_label(x) < 2)
101.
102.  train_ds = dataset.take(train_size)
103.
104.  train_ds = train_ds.filter(lambda x: get_label(x) < 2)
105.  train_ds = train_ds.map(map_files, num_parallel_calls=AUTOTUNE)
106.  train_ds = train_ds.cache()
107.  train_ds = train_ds.batch(BATCH_SIZE)
108.  train_ds = train_ds.prefetch(AUTOTUNE)
109.
110.  ver_ds = tf.data.Dataset.list_files(path_to_ver_data, shuffle=20)
111.  ver_ds = ver_ds.filter(lambda x: get_label(x) < 2)
112.  ver_ds = ver_ds.map(map_files, num_parallel_calls=AUTOTUNE)
113.  ver_ds = ver_ds.cache()
114.  ver_ds = ver_ds.batch(BATCH_SIZE)
115.  ver_ds = ver_ds.prefetch(AUTOTUNE)
116.
117.  ## Plot some train data
118.
119.  plot_ds = train_ds.take(1)
120.  fig, (axes) = plt.subplots(3, 2)
121.  for row in plot_ds:
122.      data = row[0]
123.      label = row[1]
124.
125.  i = 0
126.  for axes_hor in axes:
127.      for ax in axes_hor:
128.          ax.plot(np.arange(0, data_size), data[i, :][0, :])
129.          ax.plot(np.arange(0, data_size), data[i, :][1, :])
130.          if label[i] == 1:
131.              textlabel = 'broken'
132.          else:
133.              textlabel = 'unbroken'
134.          ax.title.set_text(f'train_data, label = {label[i]},  ' + textlabel)
135.          i += 1
136.
137.  plt.show(block=False)
138.  plt.pause(1)
139.  plt.close()
140.
141.  ## Plot some verification data
142.
143.  plot_ds = ver_ds.take(1)
144.  fig, (axes) = plt.subplots(3, 2)
145.  for row in plot_ds:
146.      data = row[0]
147.      label = row[1]
148.
149.  i = 0
150.  for axes_hor in axes:
```

```
151.     for ax in axes_hor:
152.         ax.plot(np.arange(0, data_size), data[i, :][0, :])
153.         ax.plot(np.arange(0, data_size), data[i, :][1, :])
154.         if label[i] == 1:
155.             textlabel = 'broken'
156.         else:
157.             textlabel = 'unbroken'
158.         ax.title.set_text(f'verification_data, label = {label[i]},  ' + textlabel)
159.         i += 1
160.
161. plt.show(block=False)
162. plt.pause(1)
163. plt.close()
164.
165. print('creating dataset finished')
166. print(time.time() - tim)
167. tim = time.time()
168.
169. ## Building model
170.
171. input = Input(shape=(2, data_size, 1))
172.
173. conv1_1 = layers.SeparableConv1D(filters=64, kernel_size=7, padding="same",
     activation='relu')(input[:, 0, :])
174. conv1_1 = layers.MaxPooling1D(5)(conv1_1)
175. conv1_1 = layers.LayerNormalization()(conv1_1)
176.
177. conv1_2 = layers.SeparableConv1D(filters=64, kernel_size=7, padding="same",
     activation='relu')(conv1_1)
178. conv1_2 = layers.MaxPooling1D(5)(conv1_2)
179. conv1_2 = keras.layers.LayerNormalization()(conv1_2)
180.
181. gap1 = layers.GlobalMaxPooling1D()(conv1_2)
182.
183. dense1 = keras.layers.Dense(1, activation="sigmoid")(gap1)
184.
185. conv2_1 = layers.SeparableConv1D(filters=64, kernel_size=7, padding="same",
     activation='relu')(input[:, 1, :])
186. conv2_1 = layers.MaxPooling1D(5)(conv2_1)
187. conv2_1 = layers.LayerNormalization()(conv2_1)
188.
189. conv2_2 = layers.SeparableConv1D(filters=64, kernel_size=7, padding="same",
     activation='relu')(conv2_1)
190. conv2_2 = layers.MaxPooling1D(5)(conv2_2)
191. conv2_2 = keras.layers.LayerNormalization()(conv2_2)
192.
193. gap2 = layers.GlobalMaxPooling1D()(conv2_2)
194.
195. dense2 = keras.layers.Dense(1, activation="sigmoid")(gap2)
196.
197. concatenated = layers.concatenate([dense1, dense2], axis=-1)
198.
199. res = keras.layers.Dense(1, activation="sigmoid")(concatenated)
200.
201. model = Model(input, res)
202.
203. model.summary()
204. plot_model(model, to_file='model.png', show_shapes=True)
205.
206. callbacks_list = [tf.keras.callbacks.ModelCheckpoint(filepath='last_model_checkpoint.h5',
207.                                                      monitor='val_accuracy',
208.                                                      mode='max',
209.                                                      save_best_only=True),
210.                    tf.keras.callbacks.ReduceLROnPlateau(monitor='loss',
211.                                                         factor=0.5,
```

```
                                                        patience=5,
                                                        verbose=0,
                                                        mode='auto',
                                                        min_delta=0.0001,
                                                        cooldown=0,
                                                        min_lr=1e-20),
                    tf.keras.callbacks.EarlyStopping(monitor='loss',
                                                     patience=50)
                    ]

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss='binary_crossentropy',
    metrics=['accuracy', 'TrueNegatives', 'TruePositives', 'FalseNegatives',
'FalsePositives']
)

history = model.fit(train_ds, epochs=5_000, verbose=2, validation_data=ver_ds,
callbacks=callbacks_list)
print('Test:')
model.evaluate(ver_ds, verbose=2)
predict = model.predict(ver_ds, verbose=2)

true_labels = np.concatenate([y for x, y in ver_ds], axis=0)

##  Plot results

loss_values = history.history['loss']
validation_loss = history.history['val_loss']
accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
epochs = range(1, len(loss_values) + 1)

fig, (ax1, ax2) = plt.subplots(2, 1)

ax1.plot(epochs, loss_values, 'bo', label='Training loss')
ax1.plot(epochs, validation_loss, 'r+', label='Validation loss')
ax1.title.set_text('Training and validation loss')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Loss')
ax1.legend()

ax2.plot(epochs, accuracy, 'bo', label='Accuracy')
ax2.plot(epochs, validation_accuracy, 'r+', label='Validation accuracy')
ax2.title.set_text('Training and validation loss')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Accuracy')
ax2.legend()

plt.show()

ROC = stat.ROC(predict, true_labels)
ROC.plot_ROC()
```

11.3 statistics_tools (Python)

```
import numpy as np
import matplotlib.pyplot as plt


def get_false_positive_rate(false_positives, true_negatives):
    """
    :param false_positives: Number of false positives
    :param true_negatives: Number of true negatives
```

```
9.          :return: false_positive_rate
10.         """
11.         if false_positives == 0:
12.             return 0
13.         false_positive_rate = false_positives / (false_positives + true_negatives)
14.         return false_positive_rate
15.
16.
17.     def get_sensitivity(true_positives, false_negatives):
18.         """
19.         :param true_positives: Number of true positives
20.         :param false_negatives: Number of false negatives
21.         :return: sensitivity
22.         """
23.         if true_positives == 0:
24.             return 0
25.         sensitivity = true_positives / (true_positives + false_negatives)
26.         return sensitivity
27.
28.
29.     class ROC:
30.         def __init__(self, predictions, true_values):
31.             """
32.             :param predictions: Predicted outputs
33.             :param true_values: Labels, true outputs
34.             """
35.             self.predictions = predictions
36.             self.true_values = true_values
37.             self.threshold_step = 0.01
38.             self.sensitivity_vector = np.array(1)
39.             self.false_positive_rate_vector = np.array(1)
40.
41.             self.get_ROC()
42.
43.         def confusion_matrix(self, threshold):
44.             """
45.             :param threshold: Treshold to evaluate predicted output
46.             :return: true_positives, false_positives, false_negatives, true_negatives
47.             """
48.             i = 0
49.             true_positives = 0
50.             false_positives = 0
51.             true_negatives = 0
52.             false_negatives = 0
53.             for prediction in self.predictions:
54.                 if prediction >= threshold and self.true_values[i] == 1:    # Predicted
    damage and actual damage
55.                     true_positives += 1
56.                 elif prediction >= threshold and self.true_values[i] == 0:  # Predicted
    damage, but actually without damage
57.                     false_positives += 1
58.                 elif prediction < threshold and self.true_values[i] == 0:  # Predicted
    without damage and actually without damage
59.                     true_negatives += 1
60.                 elif prediction < threshold and self.true_values[i] == 1:  # Predicted
    without damage, but actually with damage
61.                     false_negatives += 1
62.                 else:
63.                     print('Error evaluating confusion matrix')
64.                 i += 1
65.
66.             return true_positives, false_positives, false_negatives, true_negatives
67.
68.         def get_ROC(self):
69.             """
```

```
70.          Computes ROC with changing threshold
71.          :return:
72.          """
73.          for threshold in np.arange(0, 1 + self.threshold_step, self.threshold_step):
74.              true_positives, false_positives, false_negatives, true_negatives =
    self.confusion_matrix(threshold)
75.              self.sensitivity_vector = np.append(self.sensitivity_vector,
    get_sensitivity(true_positives, false_negatives))
76.              self.false_positive_rate_vector = np.append(self.false_positive_rate_vector,
    get_false_positive_rate(false_positives, true_negatives))
77.
78.          self.false_positive_rate_vector =
    np.append(self.false_positive_rate_vector,np.array(0))
79.          self.sensitivity_vector = np.append(self.sensitivity_vector, np.array(0))
80.
81.      def plot_ROC(self):
82.          """
83.          Plots resulting ROC
84.          :return:
85.          """
86.          fig, (ax1) = plt.subplots(1, 1)
87.
88.          ax1.plot(self.false_positive_rate_vector, self.sensitivity_vector, '-bo',
    label='ROC')
89.          ax1.plot(np.array([0, 1]), np.array([0, 1]), '--r', label='Not good at all')
90.          ax1.title.set_text('ROC curve')
91.          ax1.set_xlabel('false positive rate')
92.          ax1.set_ylabel('sensitivity')
93.          ax1.legend()
94.          ax1.grid()
95.
96.          plt.show()
```

12 Zpracování dat EMA

12.1 EMA__Zpracovani (Matlab)

```
clear all
close all

nonlin_vel = importdata('Scan_time vel_4.txt');
nonlin_force = importdata('Scan_time force_4.txt');
lin_vel = importdata('Scan_lepene_time_vel_4.txt');
lin_force = importdata('Scan_lepene_time force_4.txt');

windows = true;
nonlin = true;

if nonlin
    time = nonlin_vel.data(:,1);
    vel = nonlin_vel.data(:,2);
    force = nonlin_force.data(:,2);
    disp('computing nonlin system...')
else
    time = lin_vel.data(:,1);
    vel = lin_vel.data(:,2);
    force = lin_force.data(:,2);
    disp('computing lin system...')
end

frequency = 1 / abs(time(1) - time(2));

%% Searching for impact time

i = 1;
start_ind = [];     % Times when impacts begin
start_before_maximum = 3;
stop_before_maximum = 1000;
while i < length(force)
    if force(i) > 10 & force(i-1) < force(i) & force(i+1) < force(i)
        start_ind = [start_ind, i-start_before_maximum];
        i = i+100;  % If impact is detected, search moves 100 points
    else
        i = i+1;    % If impact is not detected, search moves 1 point
    end
end

for i = 1:(length(start_ind)-1)
    % Lengths of data - time between impacts
    d_length(i) = abs(start_ind(i) - start_ind(i+1));
end
% Length of processed data
min_d_length = min(d_length) - stop_before_maximum;

%% Slicing

for i = 1:(length(start_ind)-1)
    force_slice(i, :) = force(start_ind(i):(start_ind(i)+min_d_length));
    vel_slice(i, :) = vel(start_ind(i):(start_ind(i)+min_d_length));
    tim_slice(i, :) = time(start_ind(i):(start_ind(i)+min_d_length));
    tim_matrix(i, :) = time(start_ind(1):(start_ind(1)+min_d_length));
end

figure(10)
plot(tim_slice', vel_slice')
```

```
59.  title('Plátkované rychlosti kmitání v čase')
60.  ylabel('v [m/s]')
61.  xlabel('t [s]')
62.
63.  figure(20)
64.  plot(tim_slice', force_slice')
65.  title('Plátkované impakty v čase')
66.  ylabel('F [N]')
67.  xlabel('t [s]')
68.
69.  figure(30)
70.  plot(time, force)
71.  hold on
72.  plot(time(start_ind), force(start_ind), 'o')
73.  hold off
74.  title('Záznam sil s počátkem impaktu')
75.  ylabel('F [N]')
76.  xlabel('t [s]')
77.
78.
79.  %% Windows
80.
81.  if windows
82.
83.      t = 0:(1/frequency):((size(tim_slice,2)-1)/frequency);
84.      T = t(end);
85.      alpha = 25;  % Force
86.      beta = 25;   % Speed
87.      exp1 = exp(-t*alpha/T);
88.      exp2 = exp(-t*beta/T);
89.      cos_win = ones(size(t));
90.      cos_win(1:start_before_maximum) = 0.5 * (1 -
     cos(frequency*t(1:start_before_maximum)*pi/start_before_maximum));
91.
92.      figure(40)
93.      plot(t, cos_win.*exp1)
94.      title('Okénko cosinové a exponenciální')
95.      ylabel('Hodnota okénka [-]')
96.      xlabel('t [s]')
97.
98.      for i = 1:size(force_slice, 1)
99.          force_slice(i,:) = force_slice(i,:) .* exp1 .* cos_win;
100.         vel_slice(i,:) = vel_slice(i,:) .* exp2 .* cos_win;
101.     end
102.
103.
104.     figure(50)
105.     plot(tim_matrix', vel_slice')
106.     title('Rychlosti kmitání po aplikaci okénka')
107.     ylabel('v [m/s]')
108.     xlabel('t [s]')
109.
110.     figure(60)
111.     plot(tim_matrix', force_slice')
112.     title('Impaktní síly po aplikaci okénka')
113.     ylabel('F [N]')
114.     xlabel('t [s]')
115. end
116.
117. %% FFT by slices
118.
119. n = length(force_slice(1,:));
120. for i = 1:size(force_slice,1)
121.     f(i,:) = abs(fft(force_slice(i,:))).^2./(n*frequency);
```

```
122.      v(i,:) = abs(fft(vel_slice(i,:))).^2./(n*frequency);
123.      freq(i,:) = (0:fix(length(f(i,:))/2)-1)*frequency/length(f(i,:));
124.  end
125.  v = v(:, 1:(size(v,2)/2));
126.  f = f(:, 1:(size(f,2)/2));
127.
128.  figure(70)
129.  stdshade(10*log10(f), 0.2, 'red', freq(1,:), 0)
130.  title('Spektrální výkonová hustota impaktů (PSD)')
131.  ylabel('PSD [dB/Hz]')
132.  xlabel('f [Hz]')
133.
134.  figure(80)
135.  stdshade(10*log10(v), 0.2, 'red', freq(1,:))
136.  title('Spektrální výkonová hustota výstupního kmitání (PSD)')
137.  ylabel('PSD [dB/Hz]')
138.  xlabel('f [Hz]')
139.
140.  h = v./f;
141.
142.  figure(90)
143.  stdshade(10*log10(h), 0.2, 'red', freq(1,:), 0)
144.  title('Frekvenční přenosová funkce - mobility')
145.  ylabel('FRF [dB]')
146.  xlabel('f [Hz]')
147.
148.  %% Identification using matlab identification toolbox
149.
150.  force_tr = force_slice';
151.  vel_tr = vel_slice';
152.  winlen = size(force_tr,1);
153.  figure(100)
154.  modalfrf(force_tr(:),vel_tr(:),frequency,winlen,'Sensor','vel');
155.  [frf_mat, f_mat, coh] = modalfrf(force_tr(:),vel_tr(:),frequency,winlen,'Sensor','vel');
```

12.2 EMA__for__AI (Matlab)

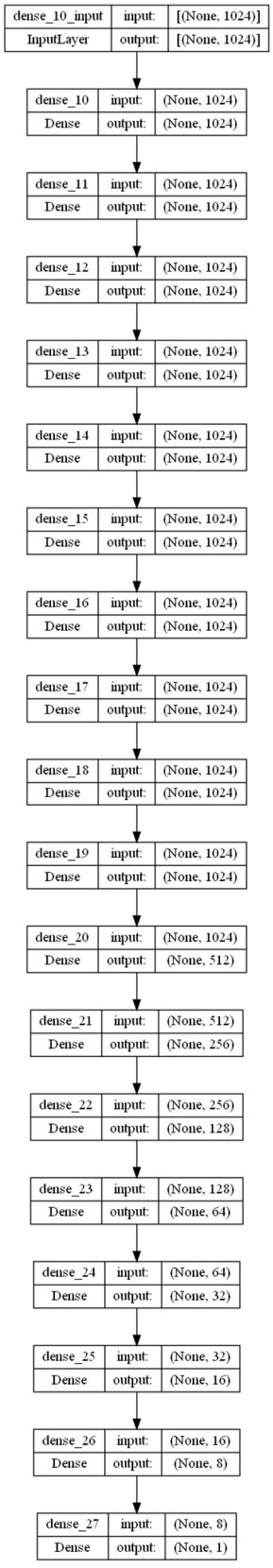
```
1.   nonlin_vel = importdata('Scan_time vel_4.txt');
2.   nonlin_force = importdata('Scan_time force_4.txt');
3.   lin_vel = importdata('Scan_lepene_time_vel_4.txt');
4.   lin_force = importdata('Scan_lepene_time force_4.txt');
5.
6.   windows = true;
7.   nonlin = false; % True to process nonlin data, false for lin data
8.
9.   if nonlin
10.      time = nonlin_vel.data(:,1);
11.      vel = nonlin_vel.data(:,2);
12.      force = nonlin_force.data(:,2);
13.      disp('computing nonlin system...')
14.  else
15.      time = lin_vel.data(:,1);
16.      vel = lin_vel.data(:,2);
17.      force = lin_force.data(:,2);
18.      disp('computing lin system...')
19.  end
20.
21.  frequency = 1 / abs(time(1) - time(2));
22.
23.  %% Searching for impact time
24.
25.  i = 1;
26.  start_ind = [];     % Times when impacts begin
27.  start_before_maximum = 3;
```

```
stop_before_maximum = 1000;
while i < length(force)
    if force(i) > 10 & force(i-1) < force(i) & force(i+1) < force(i)
        start_ind = [start_ind, i-start_before_maximum];
        i = i+100;  % If impact is detected, search moves 100 points
    else
        i = i+1;    % If impact is not detected, search moves 1 point
    end
end

for i = 1:(length(start_ind)-1)
    % Lengths of data - time between impacts
    d_length(i) = abs(start_ind(i) - start_ind(i+1));
end
% Length of processed data
min_d_length = min(d_length) - stop_before_maximum;
min_d_length = 2048*3;

%% Slicing

for i = 1:(length(start_ind)-1)
    force_slice(i, :) = force(start_ind(i):(start_ind(i)+min_d_length));
    vel_slice(i, :) = vel(start_ind(i):(start_ind(i)+min_d_length));
    tim_slice(i, :) = time(start_ind(i):(start_ind(i)+min_d_length));
    tim_matrix(i, :) = time(start_ind(1):(start_ind(1)+min_d_length));
end

%% Windows
if windows

    t = 0:(1/frequency):((size(tim_slice,2)-1)/frequency);
    T = t(end);
    alpha = 15;  % Force
    beta = 15;   % Speed
    exp1 = exp(-t*alpha/T);
    exp2 = exp(-t*beta/T);
    cos_win = ones(size(t));
    cos_win(1:start_before_maximum) = 0.5 * (1 -
cos(frequency*t(1:start_before_maximum)*pi/start_before_maximum));

    for i = 1:size(force_slice, 1)
        force_slice(i,:) = force_slice(i,:) .* exp1 .* cos_win;
        vel_slice(i,:) = vel_slice(i,:) .* exp2 .* cos_win;
    end

end

%% FFT by slices

n = length(force_slice(1,:));
for i = 1:size(force_slice,1)
    f(i,:) = abs(fft(force_slice(i,:))).^2./(n*frequency);
    v(i,:) = abs(fft(vel_slice(i,:))).^2./(n*frequency);
    freq(i,:) = (0:fix(length(f(i,:))/2)-1)*frequency/length(f(i,:));
end
v = v(:, 1:(size(v,2)/2));
f = f(:, 1:(size(f,2)/2));

%% Data for machine learning

v_smooth = smoothdata(v(i,1:frequency),'Gaussian',10);
FFT = log10(v_smooth);
FFTN = (FFT-mean(FFT))/std(FFT);
figure
plot(FFTN)
```

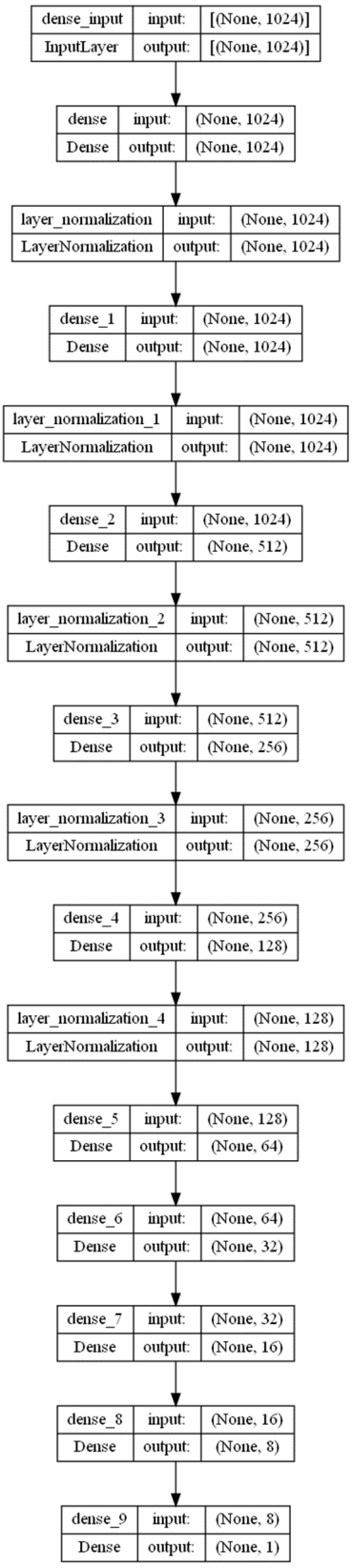
```
92.  std(FFTN)
93.  mean(FFTN)
94.
95.  if nonlin
96.      for i = 1:size(force_slice,1)
97.          FFT = log10(v_smooth);
98.          FFTN = (FFT-mean(FFT))/std(FFT);
99.          writematrix(FFTN,
     ['C:\Users\vojta\Documents\GitHub\StrojoveUceni\EMA_data\broken\broken_',num2str(i-
     1),'.csv'])
100.     end
101. else
102.     for i = 1:size(force_slice,1)
103.         FFT = log10(v_smooth);
104.         FFTN = (FFT-mean(FFT))/std(FFT);
105.         writematrix(FFTN,
     ['C:\Users\vojta\Documents\GitHub\StrojoveUceni\EMA_data\unbroken\unbroken_',num2str(i-
     1),'.csv'])
106.     end
107. end
```

13 Grafy DNN

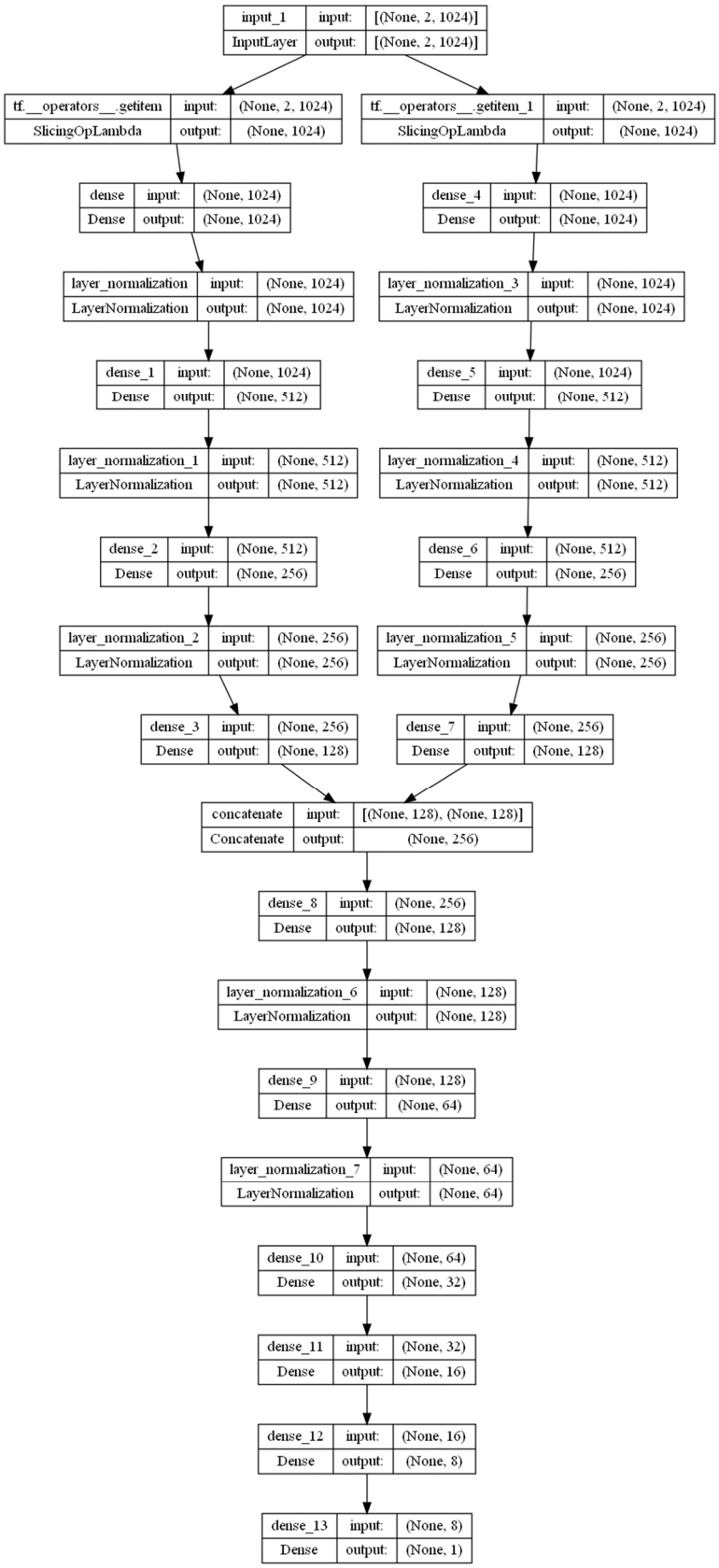
13.1 Síť s mnoha dense vrstvami



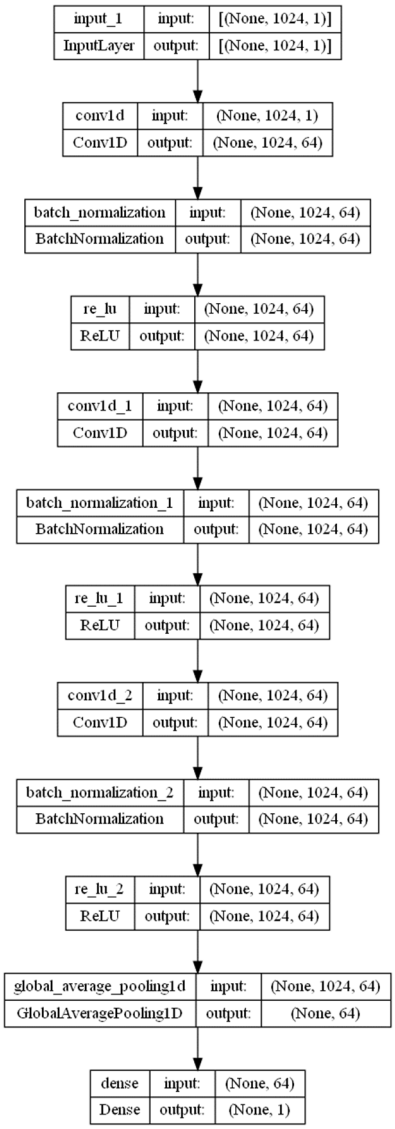
13.2 Síť s vrstvami dense a layer_normalization



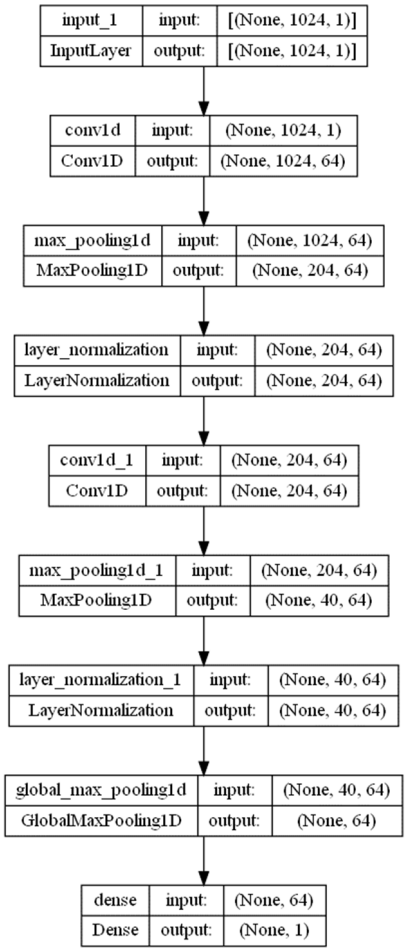
13.3 Síť dvěma vstupy, vrstvami dense a layer_normalization



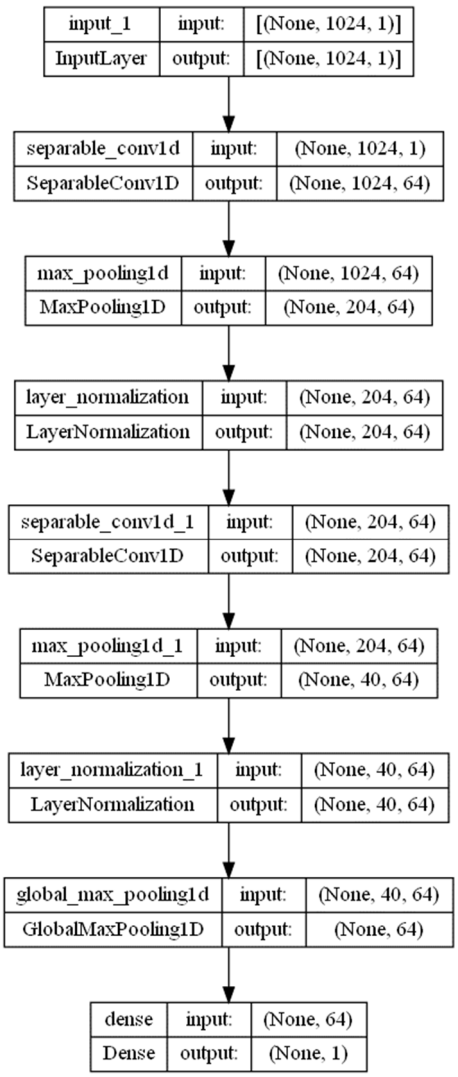
13.4 Síť s vrstvou convolution a batch_normalization



13.5 Síť s vrstvou convolution a layer_normalization



13.6 Síť s vrstvou separable_convolution



13.7 Síť s více vstupy a vrstvou separable_convolution