VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NOVÉ HYBRIDNÍ METODY PRO ROBUSTNÍ A AUTOMATIZOVANÝ ODHAD PARAMETRŮ MECHATRONICKÝCH SYSTÉMŮ

NEW HYBRID METHODS FOR ROBUST AND AUTOMATED PARAMETER ESTIMATION OF MECHATRONIC SYSTEMS

DIZERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE — Ing. Jan Najman
AUTHOR

ŠKOLITEL — doc. Ing. Robert Grepl, Ph.D.
SUPERVISOR

BRNO 2022

Abstrakt

Dizertační práce se zabývá vývojem nového hybridního optimalizačního algoritmu pro mechatronické modely. Úvodní kapitoly se věnují obecnému popisu problematiky odhadu neznámých parametrů systému, na základě vytvořeného matematického modelu a naměřených dat. Dále je uveden přehled a stručný popis dostupných optimalizačních algoritmů, které jsou vhodné pro řešení tohoto typu problému. Na základě rešeršní studie jsou pak formulovány konkrétní cíle práce. Ve druhé části práce je popsána nově vytvořená sada mechatronických modelů vytvořených pomocí nástrojů pro fyzikální modelování. Následně je s pomocí těchto modelů proveden srovnávací test rychlosti a úspěšnosti vybraných optimalizačních algoritmů. Na základě výsledků tohoto testu je navržen design nového hybridní algoritmu, který je následně otestován a srovnán s ostatními algoritmy. Na závěr práce je představeno několik nových pomocných funkcí a nástrojů pro detekci a analýzu nevhodně navržených úloh odhadu parametrů.

Klíčová slova

optimalizace, odhad parametrů, hybridní algoritmus, mechatronický model, MATLAB, Simulink, Simscape

Abstract

The thesis deals with the development of a new hybrid optimization algorithm for mechatronic models. The introductory chapters are devoted to a general description of the problem of estimating unknown system parameters, based on the developed mathematical model and measured data. Furthermore, an overview and a brief description of available optimization algorithms that are suitable for solving this type of problem is given. Based on the research study, the specific objectives of the paper are then formulated. In the second part of the thesis, a newly developed set of mechatronic models created using physical modelling tools is described. Subsequently, a comparative test of the speed and success rate of the selected optimization algorithms is performed using these models. Based on the results of this test, the design of a new hybrid algorithm is proposed, which is then tested and compared with the other algorithms. Finally, several new auxiliary functions and tools are presented to detect and analyze improperly designed parameter estimation problems.

Key words

optimization, parameter estimation, hybrid algorithm, mechatronic model, MATLAB, Simulink, Simscape

NAJMAN, Jan. *Nové hybridní metody pro robustní a automatizovaný odhad parametrů mechatronických systémů*. Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/137854. Dizertační práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Robert Grepl.

Prohlášení

Prohlašuji, že jsem dizertační práci na téma *Nové hybridní metody pro robustní a automatizovaný odhad parametrů mechatronických systémů* vypracoval samostatně s použitím odborné literatury a pramenů, uvedených v seznamu, který tvoří přílohu této práce.

29. 6. 2022 Jan Najman

Poděkování

Rád bych na tomto místě poděkoval svému školiteli doc. Ing. Robertu Greplovi, Ph.D, za jeho vedení, konzultace a cenné rady při tvorbě této práce i během celého doktorského studia.

Dále bych chtěl poděkovat kolegům z laboratoře Mechlab za jejich pomoc, podporu a vytvoření příjemného a většinou i produktivního pracovního prostředí.

V neposlední řadě také děkuji mé manželce za trpělivost a podporu, zejména v závěru mého doktorského studia. Dík patří také mým rodičům, přátelům a všem, kteří mě během studia podporovali, motivovali a inspirovali.

Obsah

1 Úvod

Jedním z problémů, kterým je třeba se zabývat v oblasti simulace, optimalizace a řízení jakýchkoliv reálných systémů, je odhad neznámých parametrů modelu na základě naměřených vstupně-výstupních dat.

Bez spolehlivého modelu zkoumaného systému totiž nelze využít pokročilých technik návrhů a testování řídicích systémů jako například HIL (Hardware-in-the-loop[1]) či dopředné řízení.

V technické praxi je zcela běžné, že jeden či více parametrů nelze na reálném systému změřit či vyčíst z tabulkových hodnot. V takovém případě je nutné parametry odhadovat a na základě srovnání odezvy simulovaného modelu s daty naměřenými na reálném systému iterativně dojít k parametrům, u kterých bude odchylka minimální.

Do této oblasti také spadá problematika vhodné struktury modelu, množství použitých parametrů a citlivosti na jejich změnu.

Navzdory tomu, že v oblasti teorie optimalizačních metod existuje nespočet publikací, studií a jiných vědeckých prací, narážíme v technické praxi na problémy, které jsou spojené se specifickými vlastnostmi konkrétního řešeného problému. Je totiž jasné, že žádná z metod nebude efektivně použitelná univerzálně ve všech případech.

Dalším častým problémem je nevhodná podoba vstupních (naměřených) dat, či již zmíněný počet parametrů.

Mezi hlavní směry v této oblasti, kterými má smysl se zabývat, potom patří snaha o výběr vhodných optimalizačních metod pro specifickou sadu modelů, se kterými se často setkáváme v technické praxi mechatronického oboru. Důležitý je také důraz na automatický běh optimalizačního algoritmu s minimálními požadavky na zásah na uživatele a maximalizace pravděpodobnosti úspěšného odhadu i bez hlubšího porozumění principu funkce použitých optimalizačních metod. V neposlední řadě je pak vhodné zkoumat i možnosti identifikace nedostatků v samotném zadání řešené úlohy, týkající se například formátu vstupních dat, vzájemné provázanosti hledaných parametrů modelu.

[1] HIL je technika pro vývoj real-time vestavěných řídicích jednotek, která využívá simulovaného modelu řízeného systému namísto reálného (např. matematický model chování motoru, namísto reálného motoru).

2 Rešeršní studie

2.1 Úloha estimace parametrů

Přestože z hlediska teorie jsou problémy nastíněné v úvodní kapitole nezávislé na použitém softwarovém nástroji, pro potřeby praktického vývoje a testování jednotlivých tezí a možných řešení je vhodné se zaměřit na konkrétní vývojové prostředí, jeho současné možnosti a případně již implementované funkce z kategorie optimalizace.

V současné době je v akademické i průmyslové sféře jedním z nejčastěji používaných softwarových nástrojů, v oblasti simulace a návrhu řízení, program MATLAB. Z tohoto důvodu má smysl se zabývat i jeho možnostmi v oblasti odhadu parametrů modelů. Veškerý vývoj v rámci této práce tedy bude probíhat právě v MATLABu. Stejně tak při zkoumání dostupných řešení se zaměříme pouze na algoritmy implementované v tomto programu.

Jak již bylo zmíněno v úvodu, úloha odhadu neznámých parametrů modelu reálného systému se skládá z několika hlavních částí, které lze do určité míry oddělit. Těmto částem se budou věnovat následující kapitoly. Stručně je lze rozdělit do níže uvedených bodů:

- Vytvoření matematického modelu systému, který co nejvěrněji napodobuje chování reálného systému a obsahuje neznámé parametry.
- Naměření dat na reálném systému.
    - Vstupní data (buzení), případně vychýlení systému z rovnovážného stavu.
    - Výstupní data (reakce systému).
- Nastavení úlohy pro nalezení neznámých parametrů modelu tak, aby simulovaný systém odpovídal reálnému.
    - Volba hodnoticí funkce, která měří „kvalitu“ nalezených parametrů. Typicky kvantifikace rozdílů mezi výstupními signály reálného a simulovaného systému.
    - Volba počátečního odhadu hledaných parametrů, případně omezení prostoru, ve kterém se mohou jejich hodnoty pohybovat.
    - Volba optimalizačního algoritmu, který prohledává prostor možných řešení (hodnot neznámých parametrů). Algoritmus se snaží iterativně přiblížit k optimálnímu řešení, kdy bude hodnoticí funkce (rozdíl reálné a simulované odezvy) minimální, v ideálním případě nulová. K tomu je třeba v každém kroku optimalizace provést simulaci s aktuálně nastavenými hodnotami parametrů, aby byla získána odezva a mohla být kvantifikována hodnoticí funkce.

2.1.1 Matematický model systému

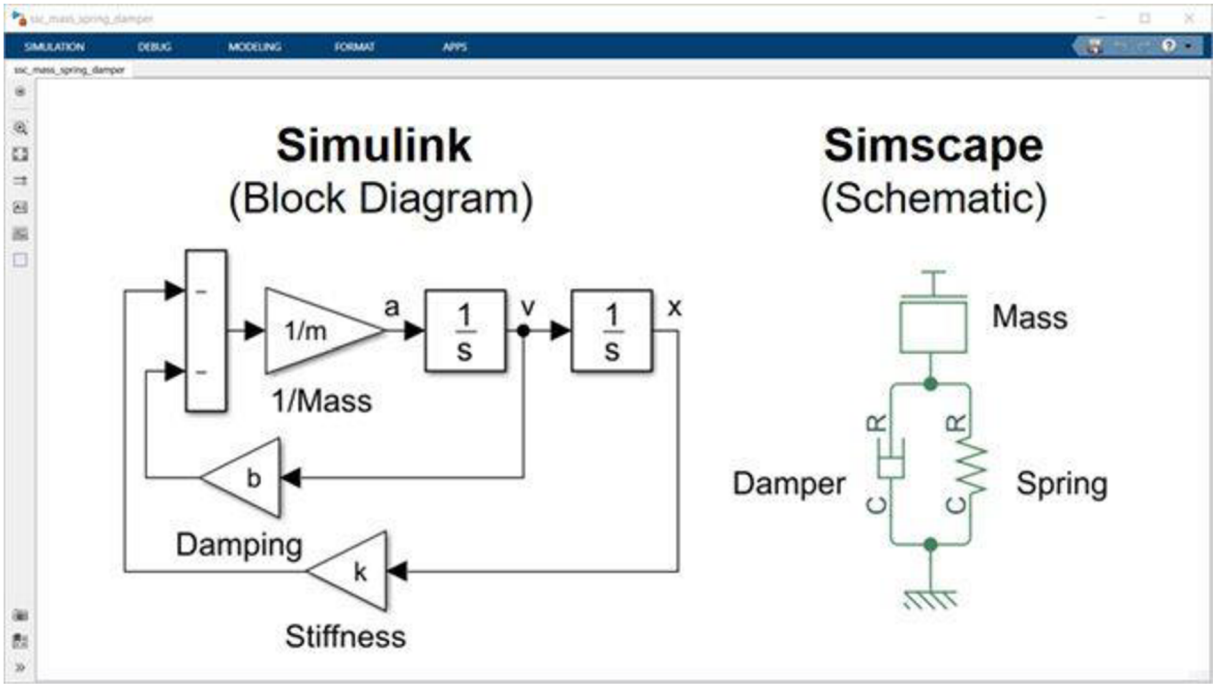
Modelem rozumíme matematickou reprezentaci chování reálného systému, například DC motoru. V inženýrské praxi se nejčastěji setkáváme s dynamickými modely, jejichž stav se mění v čase v závislosti na buzení a počátečních podmínkách.

Možností, jak vytvořit model dynamického systému je opět celá řada. V této práci uvedeme dva hlavním přístupy, které jsou používány ve spojení s nástrojem MATLAB, respektive jeho nadstavbou Simulink.

Prvním, z určitého pohledu nejjednodušším způsobem, je ruční sestavení diferenciálních rovnic chování systému. Tento přístup je ovšem použitelný pouze v případě, že uživatel je schopen tyto rovnice sestavit. To je v případě jednodušších systémů relativně jednoduché, ovšem s rostoucí komplexitou modelu a přítomností nelinearit se tento úkol stává rychle velmi složitým.

Jedním řešením této potenciálně náročné úlohy tvorby modelu, je využití nástroje pro fyzikální modelování – Simscape. Tento nástroj umožňuje rychle vytvářet modely fyzikálních systémů v prostředí Simulink. Modely jednotlivých fyzických komponent (např. hmotné těleso, tlumič, pružina, …) jsou propojovány na základě fyzických spojení, což umožňuje intuitivní tvorbu celého modelu dle jeho reálné předlohy.

Velmi jednoduchý příklad dvou výše zmíněných přístupů k tvorbě modelu lze vidět na Obr. 1. Jedná se o model mechanického oscilátoru. Již ze samotného obrázku je patrné, že Simscape má jasnou výhodu z hlediska jednoduchosti použití. Naopak výhodou ručního sestavení diferenciálních rovnic (např. ve formě blokového schématu) je kontrola nad strukturou modelu a vhled do vnitřních souvislostí.



Obr. 1. Srovnání reprezentace systému v Simulinku pomocí blokového schématu diferenciální rovnice a Simscape. [1]

Vzhledem k tomu, že v praxi je často kladen důraz na rychlý vývoj, je použití nástroje Simscape logickou volbou. V důsledku toho pak mohou vzniknout komplikace, které nejsou na první pohled patrné. Příkladem může být snaha estimovat hodnotu parametrů, které není možno vzájemně odseparovat pouze na základě naměřeného buzení a odezvy systému. Pokud zůstaneme u modelu mechanického oscilátoru, tak je z níže uvedené diferenciální rovnice (2.1) a následně provedené substituce (2.2) patrné, že nelze unikátně odhadnout konkrétní hodnotu všech tří parametrů k, m, b, protože existuje nekonečné množství kombinací, které se budou ve výsledku chovat identicky. Unikátně identifikovatelné jsou pouze substituční parametry p_1, p_2.

Tento problém se samozřejmě vyskytuje vždy při tvorbě modelu, nejen při použití Simscape, nicméně při použití nástroje, který paradoxně úlohu modelování usnadňuje, je toto nebezpečí vyšší.

$$\ddot{x} = -\frac{k}{m}x - \frac{b}{m}\dot{x} \tag{2.1}$$

$$\ddot{x} = -p_1 x - p_2 \dot{x} \tag{2.2}$$

Dalším aspektem tvorby modelu dynamického systému, je volba numerického řešiče a velikosti jeho kroku. Toto je kapitola sama pro sebe, a v této práci se touto problematikou nebudeme hlouběji zabývat. Pro úlohu estimace parametrů je však vhodné se alespoň krátce zmínit o volbě kroku řešiče. Krok řešiče totiž ovlivňuje numerickou přesnost výpočtu a také výpočetní náročnost simulace, což je pro úlohu estimace parametrů velmi důležitý údaj. Existují dvě možnosti volby kroku řešiče, fixní a variabilní. Jejich výhody a nevýhody jsou shrnuty níže.

Výhodnou variabilního kroku je automatická úprava délky kroku dle po potřeb konkrétního modelu a aktuálního stavu simulace. Většinou poskytuje přesnější výsledky než fixní krok, nevýhodou ale může být nedeterministický čas výpočtu jedné simulace (může velmi zpomalit v oblasti výrazných nelinearit v systému). Oproti tomu fixní krok není natolik flexibilní, nicméně je možné predikovat dobu simulace a mít kontrolu nad krokem výpočtu [2]. To může být důležité například v souvislosti s použitím modelů v HIL simulacích [3].

Na závěr této podkapitoly je vhodné se ještě krátce zmínit o postupu při vytváření modelu komplexních systémů. Je přirozené, že při konstrukci struktury modelu je snaha vytvořit co nejvěrnější obraz reálného chování systému, což může vést k tvorbě velmi rozsáhlého a detailního modelu. To sebou ovšem nese značná rizika, jelikož s rostoucím počtem neznámých parametrů roste exponenciálně i náročnost úlohy odhadu těchto parametrů. Současně také většinou roste i výpočetní náročnost jedné simulace. Z těchto důvodů je vhodné postupovat při tvorbě modelu a odhadu parametrů iterativně. Tedy nejprve vytvořit co nejjednodušší verzi, vystihující nejvýznamnější aspekty chování reálného modelu a odhadnout její parametry. Chování této základní verze logicky nebude zcela odpovídající naměřeným datům, avšak je možné ji pak dále postupně rozšiřovat o další prvky a parametry. Nejlepší nalezené řešení (hodnoty parametrů) z jednoduššího modelu pak lze většinou úspěšně použít jako výchozí bod pro odhad parametrů komplexnější verze, což výrazně zvyšuje šanci na úspěšné nalezení globálního optima.

2.1.2 Data naměřená na reálném systému

Kromě samotného modelu, je pro řešení úlohy estimace parametrů nutné dodat i data, naměřená na reálném systému, jehož chování chceme napodobit. Těmito daty se rozumí časový průběh vstupů do systému (buzení) a jeho výstupů.

Mohou samozřejmě nastat i případy, kdy model žádný vstup nemá, a jeho dynamické chování se projeví na základě jeho vychýlení z ustáleného stavu. Tyto počáteční stavy modelu pak lze brát jako parametry systému, které je možné buď explicitně zadat, nebo je odhadovat spolu s dalšími hlavními parametry. Je pak třeba mít na paměti, že tyto hodnoty počátečních

podmínek jsou specifické pro každou sadu naměřených dat (na rozdíl od hlavních parametrů popisujících chování systému jako takového).

Pro data měření reálného modelu platí v souvislosti s úlohou odhadu parametrů několik hlavních pravidel:

- Vhodná vzorkovací frekvence měření.
- Vhodná volba budicího signálu tak, aby se v odezvě systému projevily všechny jeho dynamické vlastnosti.
- Naměření více různorodých sad dat.

Co se týče vhodné vzorkovací frekvence platí Nyquistův–Shannonův vzorkovací teorém, tedy že přesná rekonstrukce spojitého, frekvenčně omezeného signálu z jeho vzorků je možná tehdy, pokud byla vzorkovací frekvence vyšší než dvojnásobek nejvyšší harmonické složky vzorkovaného signálu [4].

V praxi se doporučuje vzorkovat s vyšší frekvencí, než požaduje výše uvedený teorém (typicky desetinásobek nejvyšší harmonické složky), což následně může vést k tendenci vzorkovat pro jistotu signál co nejrychleji, tak jak to umožňuje použitý měřicí hardware. Nadměrně vysoká vzorkovací frekvence (oversampling) není z principu špatná, nicméně může způsobovat nežádoucí zpomalení výpočtu, vzhledem k nutnosti zpracovávat velký objem dat. Dalším nepřímým důsledkem může být nevhodná volba fixního kroku řešiče v simulaci, pokud by jej uživatel pro jednoduchost volil tak, aby se shodoval se vzorkovací frekvencí naměřených dat (což se v praxi, např. u nezkušených studentů skutečně děje).

Volbu buzení systému je nutné provést tak, aby se v naměřených datech projevily všechny jeho dynamické vlastnosti a parametry. Pokud bychom použili příklad jednoduchého stejnosměrného motoru, tak v případě, že motor budeme roztáčet velmi pomalu (malé zrychlení), neprojeví se v naměřených datech vliv momentu setrvačnosti rotoru.

V návaznosti na předchozí odstavec je možné říci, že pokud se parametr modelu v naměřených datech neprojeví, respektive změna hodnoty tohoto parametru zásadně neovlivní chování modelu, je to pravděpodobně z jednoho ze dvou důvodů:

1. Parametr není v modelu systému podstatný.
2. Parametr je v modelu systému podstatný, ale neprojeví se, protože systém je nevhodně buzen.

Nakonec je třeba se zmínit i o nutnosti mít pro úlohu odhadu parametrů k dispozici více různých sad naměřených dat. Tyto sady by měly sloužit k tomu, aby byly nalezené parametry systému funkční pro co nejširší spektrum možných budicích signálů. To že odhad funguje pro jednu konkrétní sadu dat, není automaticky zárukou, že bude fungovat na všech ostatních. Z toho důvodu se často naměřené datové sady rozdělují na část tréninkovou a verifikační. Pokud parametry nalezené s pomocí tréninkových dat posléze fungují i s použitím verifikačních dat, je možné předpokládat, že nalezené parametry odpovídají reálnému systému.

2.1.3 Nastavení úlohy estimace parametrů

Pokud již máme k dispozici matematický model s hledanými parametry, a máme i naměřená data z reálného systému, je nutné jako poslední krok provést nastavení samotné optimalizační úlohy pro nalezení parametrů.

Jak již bylo stručně zmíněno v kapitole 2.1, nastavení optimalizační úlohy se skládá z několika kroků:

- Výběr jedné nebo více sad dat, na kterých budeme odhad parametrů provádět (viz kapitola 2.1.2).
- Volba neznámých parametrů (příp. počátečních stavů systému), jejichž hodnotu chceme nalézt.
- Volba počátečního odhadu hodnoty neznámých parametrů.
- Volitelně – určení horních a dolních limitů, ve kterých se mohou hodnoty parametrů pohybovat.
- Výběr metody výpočtu hodnoticí funkce pro optimalizační algoritmus.
- Výběr optimalizačního algoritmu, případně nastavení jeho specifických parametrů.
- Volitelně – volba kritéria pro ukončení optimalizační úlohy.

Neznámé parametry a jejich hodnoty
Jak již bylo zmíněno v kapitole 2.1.2, s vyšším počtem neznámých parametrů, které jsou hledány současně, roste exponenciálně náročnost optimalizační úlohy, jelikož s každým parametrem přibývá jedna dimenze prohledávaného prostoru možných řešení.

Minimálně na počátku celé úlohy by tedy měla být snaha co nejvíce redukovat počet hledaných parametrů, i za cenu menší věrnosti matematického modelu vůči realitě.

Neméně důležitý je i počáteční odhad hodnoty neznámých parametrů. To může být v některých případech velmi obtížné. Pak je vhodné zvolit alespoň řádově limity, ve kterých se může daný parametr pohybovat. Limity parametrů jsou obzvláště důležité pro většinu globálních optimalizačních metod, které hledají řešení i mimo počáteční odhad. Obecně lze říci, že jakékoliv (i velmi konzervativní) omezení prostoru hledaných parametrů výrazně zvyšuje šanci na nalezení globálního optima.

Výpočet hodnoticí funkce
Hodnoticí funkce je (kromě samotných hledaných parametrů) hlavním vstupem pro optimalizační algoritmus. Ten se pak snaží najít optimální hodnoty parametrů modelu tak, že hledá globální minimum hodnoticí funkce.

V případě optimalizace na základě srovnávání měřených a simulovaných výstupních dat systému, se pro výpočet hodnoticí funkce obvykle používá jedna z níže uvedených metod [5]:

- **MAE** (Mean absolute error) představuje rozdíl mezi reálnými a simulovanými hodnotami, získaný zprůměrováním absolutního rozdílu v souboru dat.
- **MSE** (Mean Squared Error) je rozdíl mezi reálnými a simulovanými hodnotami, získaný čtvercem průměrného rozdílu v souboru dat.

- **RMSE** (Root Mean Squared Error) představuje míru chyby vyjádřenou druhou odmocninou z MSE.
- **R^2** (koeficient determinace) představuje koeficient toho, jak dobře odpovídají hodnoty simulace ve srovnání s hodnotami měření. Hodnota od 0 do 1 se interpretuje v procentech. Čím je hodnota vyšší, tím je model lepší.

Výše uvedené metody jsou matematicky vyjádřeny pomocí rovnic (2.3) až (2.6) takto:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.3}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.4}$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.5}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{2.6}$$

kde y_i jsou jednotlivé body výstupního signálu reálného systému, $\hat{y}_i$ jsou body výstupního signálu simulovaného modelu systému, N je celkový počet bodů měření a $\bar{y}$ je průměr hodnot y_i.

Z výše zmíněných jsou v případě úlohy odhadu parametrů modelu nejčastěji používanými metodami MSE, resp. RMSE případně MAE [6]. Hodnota jejich výstupu je přímo závislá na hodnotách vstupních dat. Najdeme však i argumenty pro použití metody R^2, jejíž výstup se vždy pohybuje v intervalu $(0, 1)$, bez ohledu na to, v jakých hodnotách se pohybují vstupní data [7].

Optimalizační algoritmus

Volba optimalizačního algoritmu je poslední částí celého procesu nastavení úlohy estimace parametrů. Následně již lze spustit samotný proces hledání parametrů, kdy optimalizační algoritmus iterativně hledá řešení, dokud nejsou splněny podmínky pro ukončení optimalizační úlohy. Vzhledem k široké paletě možností v oblasti volby optimalizačního algoritmu, je tato problematika podrobněji rozepsána v samostatné kapitole níže (2.2).

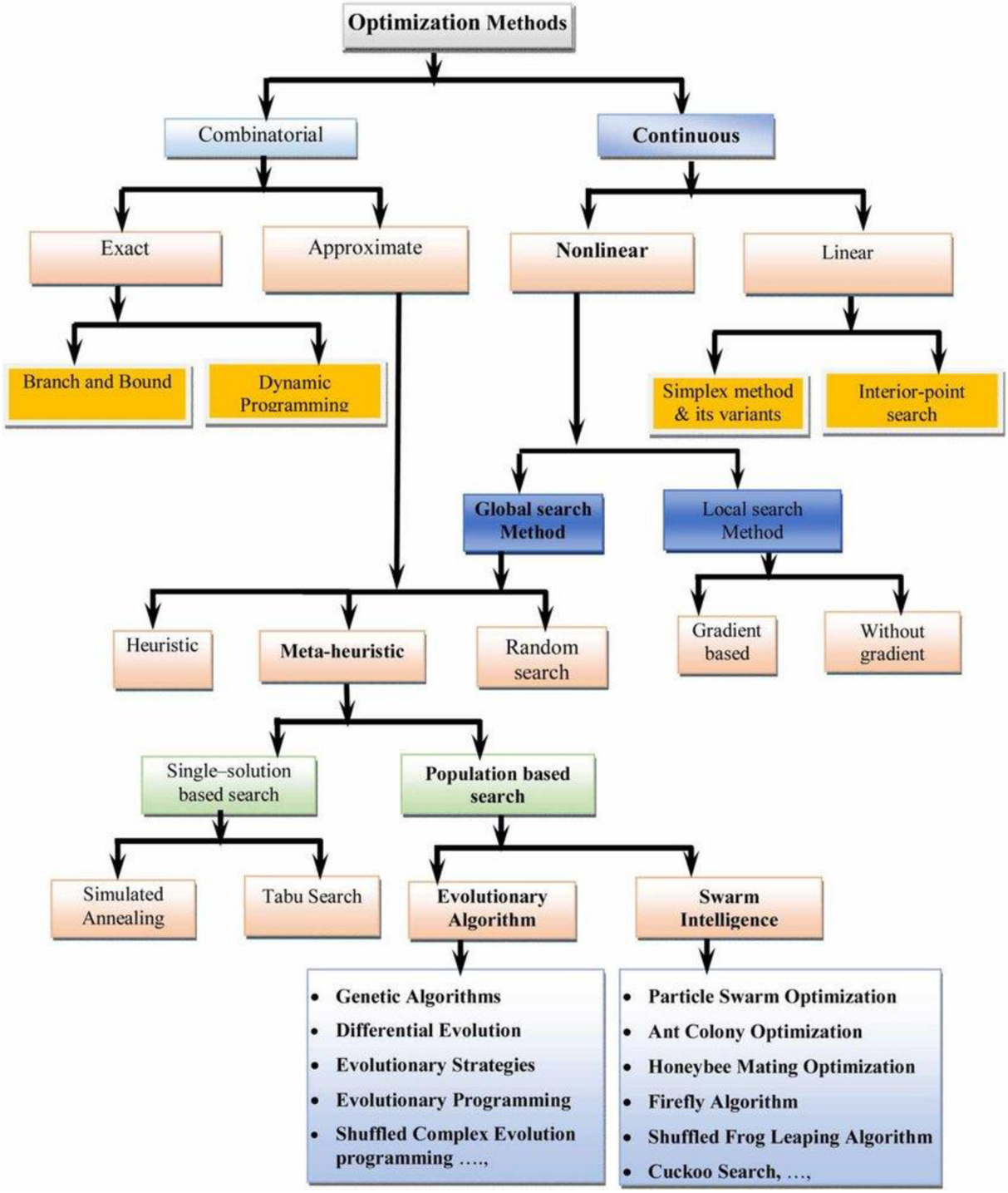
2.2 Optimalizační algoritmy

Optimalizačních algoritmů a jejich variant existuje velké množství, pro přehlednost je tedy vhodné je sdružovat do obecnějších kategorií. V literatuře můžeme nalézt mnoho publikací

zabývajících se kategorizací optimalizační algoritmů a analýzou vhodnosti jejich použití na různé druhy problémů [8], [9], [10], [11].

Několik variant klasifikace můžeme vidět na obrázcích níže (Obr. 2, Obr. 3, Obr. 4).

Vzhledem k tomu, že mnoho názvů algoritmů nemá zavedený český překlad, budeme v této práci převážně používat jejich anglický název. Díky tomu budou také označení v této práci odpovídat pojmům v anglicky psané odborné literatuře.

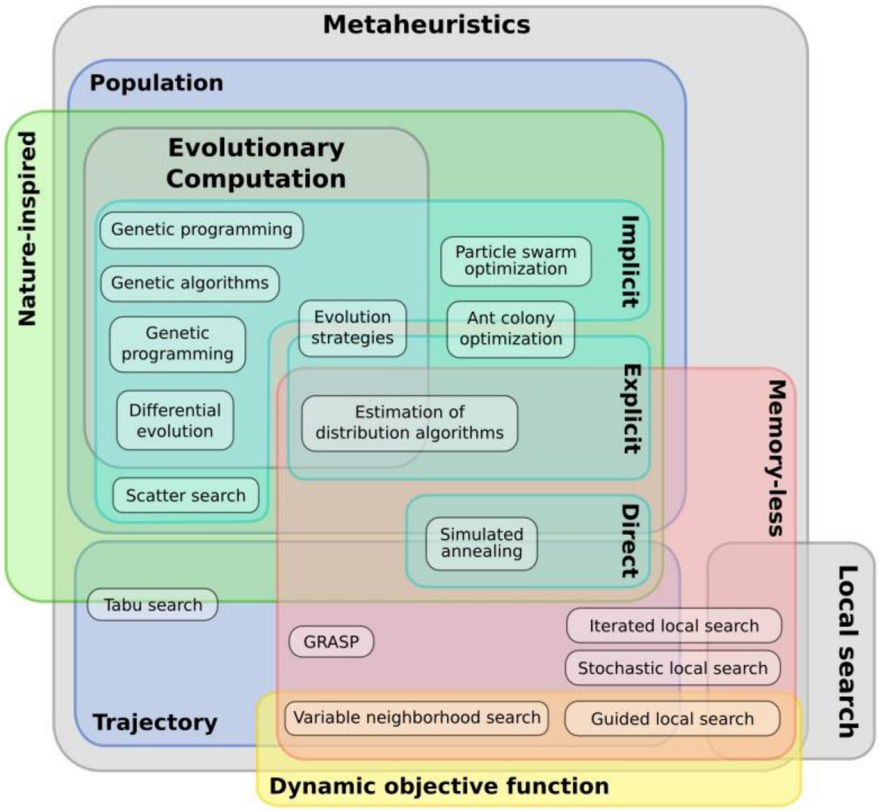


Obr. 2. Kategorizace optimalizačních metod dle [11].

Classification of mostly-used algorithms applied to building performance optimization.

Family	Strength and weakness	Typical algorithms
Direct search family (including generalized pattern search (GPS) methods)	– Derivative-free methods – Can be used even if the cost function have small discontinuities – Some algorithms cannot give exact minimum point – May be attracted by a local minimum – Coordinate search methods often have problems with non-smooth functions	Exhaustive search, Hooke–Jeeves algorithms, Coordinate search algorithm, Mesh adaptive search algorithm, Generating set search algorithm, Simplex algorithms
Integer programming family	Solving problems which consist of integer or mixed-integer variables	Branch and Bound methods, Exact algorithm, Simulated annealing, Tabu search, Hill climbing method, CONLIN method
Gradient-based family	– Fast convergence; a stationary point can be guaranteed – Sensitive to discontinuities in the cost function – Sensitive to multi-modal function	Bounded BFGS, Levenberg–Marquardt algorithm, Discrete Armijo Gradient algorithm, CONLIN method, etc.
Meta-heuristic method Stochastic population-based family	– Not to "get stuck" in local optima – Large number of cost function evaluations – Global minimum cannot be guaranteed	+ Evolutionary optimization family: GA, Genetic programming, Evolutionary programming, Differential evolution, Cultural algorithm + Swarm intelligence: Particle swarm optimization (PSO), Ant colony algorithm, Bee colony algorithm, Intelligent water drop
Trajectory search family	– Easy implementation even for complex problems – Appropriate for discrete optimization problems (continuous variables can also be used), e.g. traveling salesman problems – Only effective in discrete search spaces – Unable to tell whether the obtained solution is optimal or not – Problems of repeated annealing	Simulated annealing, Tabu search, Hill climbing method
Other		Harmony search algorithm, Firefly algorithm, Invasive weed optimization algorithm
Hybrid family	Combining the strength and limiting the weakness of the above-mentioned approaches	PSO-HJ, GA-GPS, CMA-ES/HDE, HS-BFGS algorithm

Obr. 3. Klasifikace nejčastěji používaných optimalizačních algoritmů dle [8].



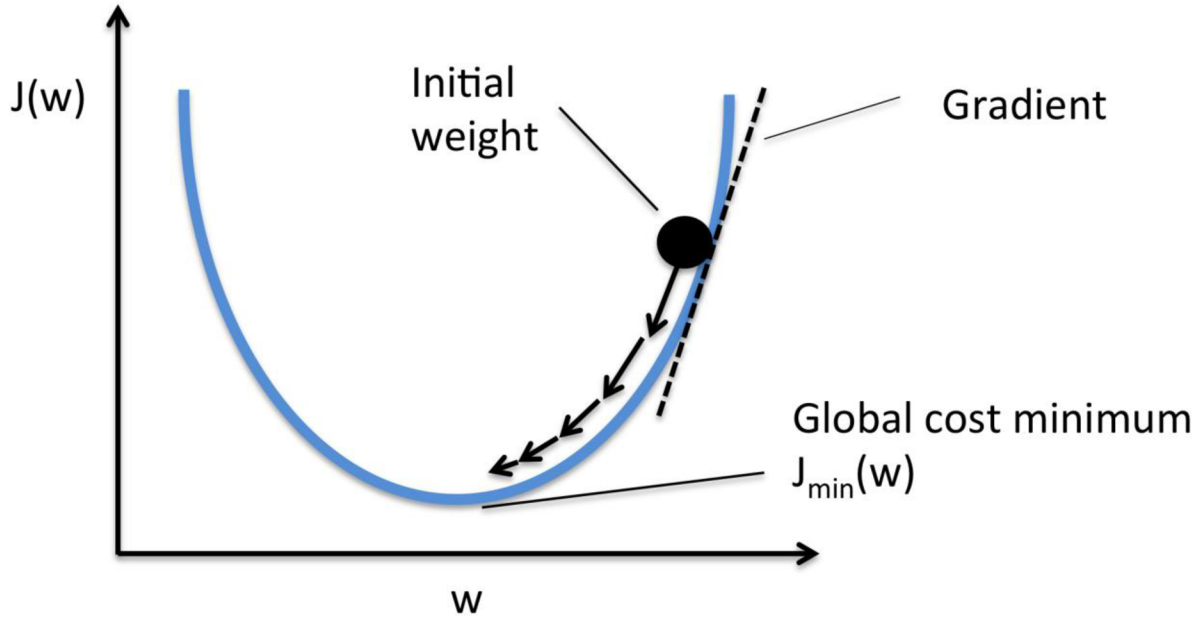
Obr. 4. Klasifikace metaheuristických optimalizačních metod dle [10].

Z výše uvedených zdrojů plyne, že kategorií a podkategorií optimalizačních algoritmů je celá řada, stejně jako možností jejich členění. Z výše uvedených kategorií vybereme nyní pouze ty, které jsou pro tuto práci nejpodstatnější. Níže je uveden jejich výpis a stručný popis. Detailněji se jednotlivými algoritmy budou zabývat následující podkapitoly (2.2.1 a 2.2.2).

Gradientní metody

Základním principem funkce této skupiny optimalizačních metod, je znalost hodnoty optimalizované funkce v daném bodě a současně jejího gradientu, resp. její derivace. Na základě těchto hodnot je pak určena velikost a směr kroku pro další iteraci, dokud se algoritmus nedostane do lokálního minima. Ilustrační příklad principu funkce gradientního algoritmu je na Obr. 5.

Výhodou je většinou velmi rychlá konvergence do lokálního minima.

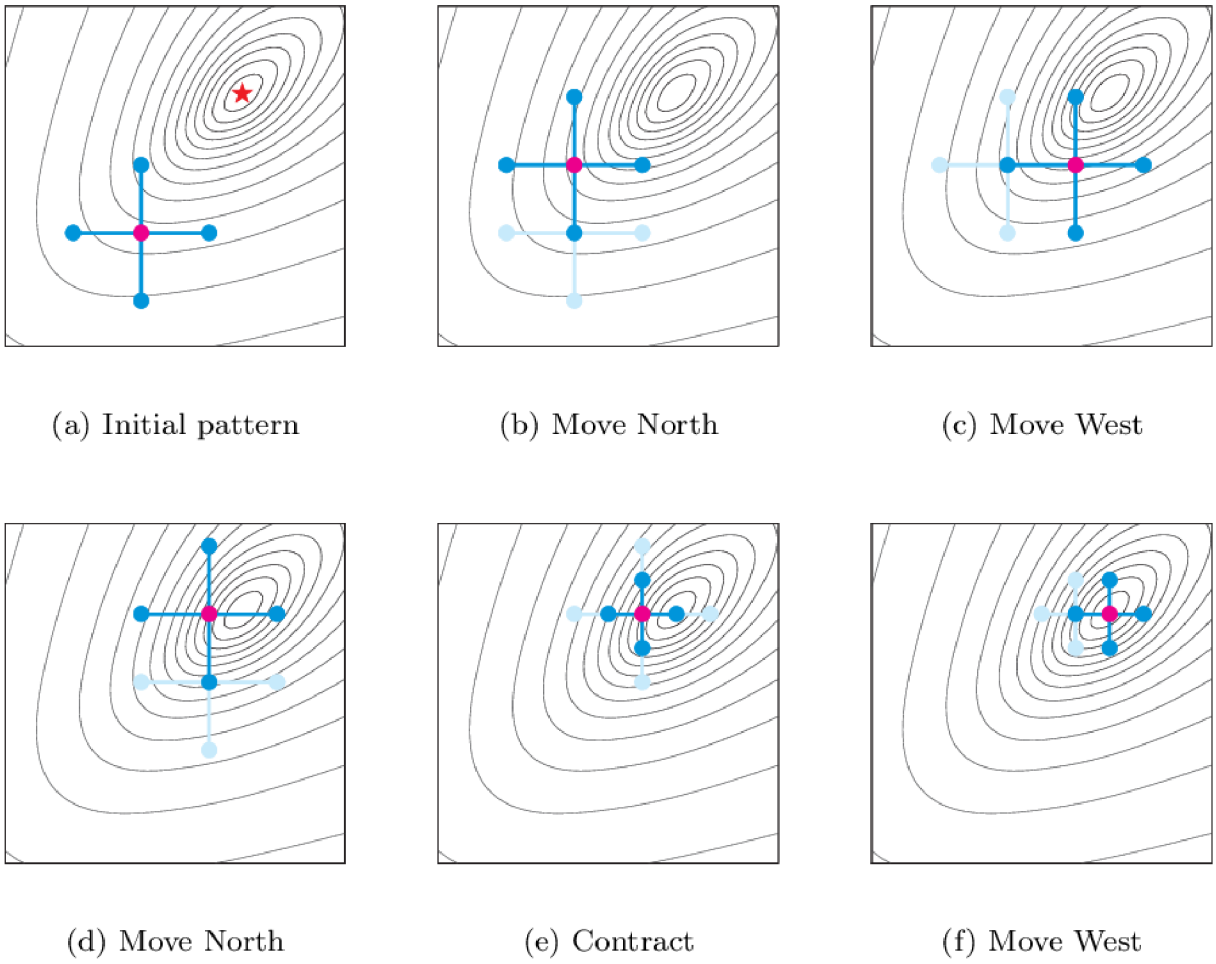


Obr. 5. Základní princip gradientních metod [12].

Direct search metody (metody přímého prohledávání)

Tyto metody na rozdíl od předchozí kategorie nevyžadují informaci o gradientu a vystačí si pouze s hodnotou funkce v daném bodě. Pro volbu kroku a směru prohledávání prostoru parametrů využívají testování sady bodů v okolí aktuálního bodu, dokud nenaleznou řešení s menší hodnotou kriteriální funkce. Ilustrační příklad fungování takového algoritmu (konkrétně patternsearch) je na Obr. 6. Podrobnější přehled vlastností těchto metod je možné získat například z [13].

Výhodou je, že jsou odolnější vůči uvíznutí v lokálním minimu a fungují i v případě, že je průběh hodnoticí funkce nespojitý.



Obr. 6. Příklad fungování direct search algoritmu (konkrétně patternsearch) [14].

Metaheuristické metody

Obecně řečeno, heuristika znamená najít nebo objevit metodou pokus-omyl. *Meta-* zde znamená mimo nebo na vyšší úrovni. Metaheuristika má obecně lepší výsledky než jednoduchá heuristika. Metaheuristiku lze považovat za hlavní strategii, která řídí a modifikuje ostatní heuristiky. Všechny metaheuristické algoritmy využívají určitý kompromis mezi náhodným a lokálním hledáním.

Dvěma hlavními složkami metaheuristických algoritmů jsou: intenzifikace a diverzifikace. Diverzifikace znamená generování různorodých řešení tak, aby byl prozkoumán prohledávaný prostor v globálním měřítku, zatímco intenzifikace znamená zaměření prohledávání v lokální oblasti s vědomím, že se v této oblasti nachází aktuální dobré řešení.

Při výběru nejlepších řešení je třeba najít vhodnou rovnováhu mezi intenzifikací a diverzifikací, aby se zlepšila rychlost konvergence algoritmu. Výběr nejlepších zajišťuje, že řešení budou konvergovat k optimu, zatímco diverzifikace prostřednictvím randomizace umožňuje hledání v prostoru od lokálních optim a zároveň zvyšuje rozmanitost řešení. Vhodná kombinace těchto dvou hlavních složek obvykle zajistí, že je možné dosáhnout globální optimality [15].

Náhradní model
Náhradním modelováním se označuje technika, která využívá vzorkování dat na původním modelu k vytvoření náhradního modelu, který jsou dostatečný k tomu, aby s relativně vysokou přesností předpovídal výstupy původního (typicky výpočetně náročného) modelu i v bodech mimo vzorkovaná data. [16]

V rámci problematiky optimalizace se pak těchto náhradních modelů využívá k aproximaci reálného tvaru hodnoticí funkce. Díky tomu, že náhradní model je výpočetně nenáročný, střídá se během optimalizační úlohy fáze hledání vhodných bodů k prozkoumání na náhradním modelu s fází vyhodnocení skutečné hodnoticí funkce v těchto bodech. Během střídání těchto dvou fází se náhradní model postupně zpřesňuje.

Metody implementované v MATLABu
Vzhledem k tomu, že tato práce je zaměřená na metody pro odhad parametrů v programu MATLAB, budeme v následujících kapitolách zkoumat pouze na algoritmy, které jsou v tomto programu k dispozici již jako předpřipravené funkce. Protože všechny tyto funkce v MATLABu mají k dispozici vlastní kvalitní dokumentaci, budeme se zabývat pouze jejich nejdůležitějšími vlastnostmi, vždy s odkazem na konkrétní zdroje.

Pro lepší orientaci, jsou jednotlivé algoritmy rozděleny do dvou obecných kategorií:

- **Lokální metody** – optimalizace probíhá pouze v okolí počátečního bodu prohledávaného prostoru parametrů. Nalezne lokální optimum, které může (ale nemusí být globálním optimem).
- **Globální metody** – optimalizace probíhá v celém prostoru parametrů. V případě dostatečně dlouhého času nalezne globální optimum.

2.2.1 Lokální optimalizační metody

2.2.1.1 fminunc (Unconstrained gradient solvers) [17]

Syntaxe funkce v MATLABu:

```
x = fminunc(fun,x0,options)
```

Algoritmus začíná v bodě `x0` a pokouší se najít lokální minimum `x` funkce $f(x)$ popsané ve `fun`. Vstup `options` pak obsahuje další možnosti nastavení algoritmu.

Pod funkcí fminunc se skrývají dvě varianty gradientních algoritmů, které řeší optimalizaci pomocí výpočtu gradientu hodnoticí funkce:

- quasi-newton [18]
- trust-region [19]

Z hlediska uživatele je největší praktický rozdíl mezi těmito dvěma metodami v požadavcích funkci $f(x)$ (vstupní parametr `fun`). V případě *trust-region* metody je nutné, aby výstupem této funkce byla nejen hodnota funkce v bodě x, ale i vektor gradientů v daném bodě. U metody

quasi-newton není toto povinné, a v případě že vektor gradientů není k dispozici, metoda si je sama vypočítá pomocí numerické derivace v okolí aktuálního bodu x.

Výhodou těchto algoritmů je jejich jednoduchost použití. Jediným povinným vstupem jsou samotná hodnoticí funkce a počáteční odhady hledaných parametrů. Není možné v tomto případě zadat žádné omezující podmínky (např. limity hodnot parametrů).

Pomocí vstupního parametru `options` je možné upravit mnoho dodatečných nastavení optimalizačního algoritmu. Mezi nejdůležitější patří:

`Algorithm`	Varianta gradientní metody: `'quasi-newton'` (výchozí) `'trust-region'`
`FiniteDifferenceType`	Metoda výpočtu numerické derivace: `'forward'` (výchozí) – dopředná diference, rychlejší `'central'` – centrální diference, přesnější
`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce
`TypicalX`	Typické hodnoty parametrů x, ke škálování velikosti kroku numerické derivace.
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 1. Vybrané možnosti nastavení funkce fminunc.

2.2.1.2 fmincon (Constrained gradient solvers) [20]

Syntaxe funkce v MATLABu:

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Vstupní parametry `fun`, `x0` a `options` mají stejný význam jako v předchozí kapitole 2.2.1.1. Dále `A`, `b` specifikují omezující podmínky lineární nerovnosti, aby platilo:

$$A \cdot x \leq b \tag{2.7}$$

Podobně `Aeq` a `beq` specifikují omezující podmínky lineární rovnosti, aby platilo:

$$Aeq \cdot x = beq \tag{2.8}$$

Parametr `nonlcon` je odkaz na funkci, která specifikuje a počítá nelineární omezující podmínky. Tato funkce na základě dodaných hodnot x musí vrátit vypočítané vektory $c(x)$ a $ceq(x)$, které pak slouží jako vstup pro omezující podmínky:

$$c(x) \leq 0 \tag{2.9}$$

$$ceq(x) = 0 \tag{2.10}$$

Parametry `lb` a `ub` pak určují limity, ve kterých se mohou pohybovat hledané hodnoty x a platí že:

$$lb \leq x \leq ub \tag{2.11}$$

Tato funkce opět obsahuje několik variant gradientních optimalizačních algoritmů, které tentokrát umějí pracovat i s výše uvedenými omezujícími podmínkami:

- interior-point [21]
- trust-region-reflective [19]
- sqp [22]
- active-set [23]

U algoritmu *trust-region-reflective* je opět nutnost dodat hodnotu gradientů (viz kap. 2.2.1.1). Ostatní algoritmy si dokáží gradienty spočítat numericky. Doporučení ohledně volby vhodné varianty z výše uvedených jsou uvedeny v [24]. Stručně je lze shrnout tak, že *interior-point* je univerzální první volba, pro zvýšení rychlosti zkusit zvolit *sqp*, případně *active-set*, který může dělat velké kroky, což dále zvyšuje rychlost.

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`Algorithm`	Varianta gradientní metody:
	`'iterior-point'` (výchozí)
	`'trust-region-reflective'`
	`'sqp'`
	`'active-set'`

`FiniteDifferenceType`	Metoda výpočtu numerické derivace: `'forward'` (výchozí) – dopředná diference, rychlejší `'central'` – centrální diference, přesnější
`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce
`TypicalX`	Typické hodnoty parametrů x, ke škálování velikosti kroku numerické derivace.
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 2. Vybrané možnosti nastavení funkce fmincon.

2.2.1.3 lsqnonlin (Nonlinear least squares) [25]

Syntaxe funkce v MATLABu:

```
x = lsqnonlin(fun,x0,lb,ub,options)
```

Vstupní parametry `fun`, `x0`, `lb`, `ub` a `options` mají stejný význam jako v předchozí kapitole 2.2.1.2.

Tento algoritmus opět využívá gradienty optimalizované (hodnoticí) funkce a nabízí volbu dvou variant:

- trust-region-reflective [19, 26]
- levenberg-marquardt [26–29]

U algoritmu *trust-region-reflective* je opět nutností dodat hodnotu gradientů (viz kap. 2.2.1.1). U algoritmu *levenberg-marquardt* je pak omezení, že tato metoda neumí pracovat s omezujícími podmínkami, tedy není možné specifikovat parametry `lb`, `ub`.

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`Algorithm`	Varianta gradientní metody: `'trust-region-reflective'` (výchozí) `'levenberg-marquardt'`
`FiniteDifferenceType`	Metoda výpočtu numerické derivace: `'forward'` (výchozí) – dopředná diference, rychlejší `'central'` – centrální diference, přesnější
`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce
`TypicalX`	Typické hodnoty parametrů x, ke škálování velikosti kroku numerické derivace.
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 3. Vybrané možnosti nastavení funkce lsqnonlin.

2.2.1.4 fminsearch (Nelder-Mead simplex search) [30]

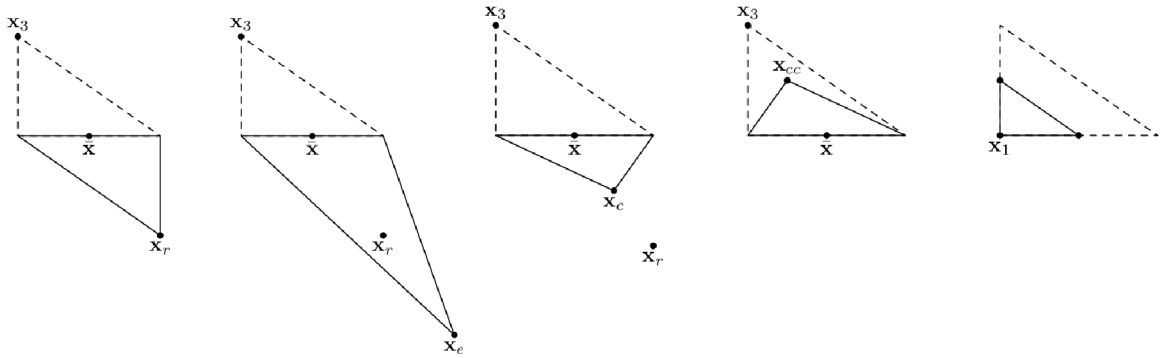
Syntaxe funkce v MATLABu:

```
x = fminsearch(fun,x0,options)
```

Vstupní parametry `fun`, `x0` a `options` mají stejný význam jako v předchozích kapitolách.

Na rozdíl od všech dříve zmíněných algoritmů, je Nelder-Mead simplex search odlišný tím, že jde o „direct search“ metodu (viz kap. 2.2), tudíž nevyžaduje hodnoty derivací optimalizované funkce. Tento algoritmus také nevyžaduje (a ani nepodporuje) žádné omezující podmínky, což může být v závislosti na typu úlohy jak výhodou, tak nevýhodou.

Jak již bylo zmíněno, funkce fminsearch používá Nelder-Mead simplex algoritmus, jenž je popsán v [31] a jeho praktická implementace v MATLABu je popsána v [32]. Metoda používá simplex [33] $n + 1$ bodů, pro prohledávání n-dimenzionálního prostoru parametrů, kde n je počet hledaných parametrů. Algoritmus podle předem daných pravidel opakovaně modifikuje simplex, jak je ilustrováno na Obr. 7 níže.



Obr. 7. Ilustrace možných trasformací simplexu během iterací prohledávacího algoritmu fminsearch [31]. *Původní simplex je čárkovaně. Aktuálně nejhorší bod představuje x_3. Zleva: překlopení, expanze, zmenšení ven, zmenšení dovnitř, smrštění.*

Striktně vzato, Nelder-Mead není globální optimalizační algoritmus, proto je zařazen do sekce lokálních metod. V praxi však obvykle funguje poměrně dobře pro problémy, které nemají mnoho lokálních minim [34].

Tento algoritmus má k dispozici jen velmi omezené možnosti nastavení pomocí parametru `options`, mezi které patří hlavně:

`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 4. Vybrané možnosti nastavení funkce fminsearch.

2.2.1.5 patternsearch (Pattern search) [35]

Syntaxe funkce v MATLABu:

```
x = patternsearch(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Všechny vstupní parametry mají stejný význam jako v kapitole 2.2.1.2.

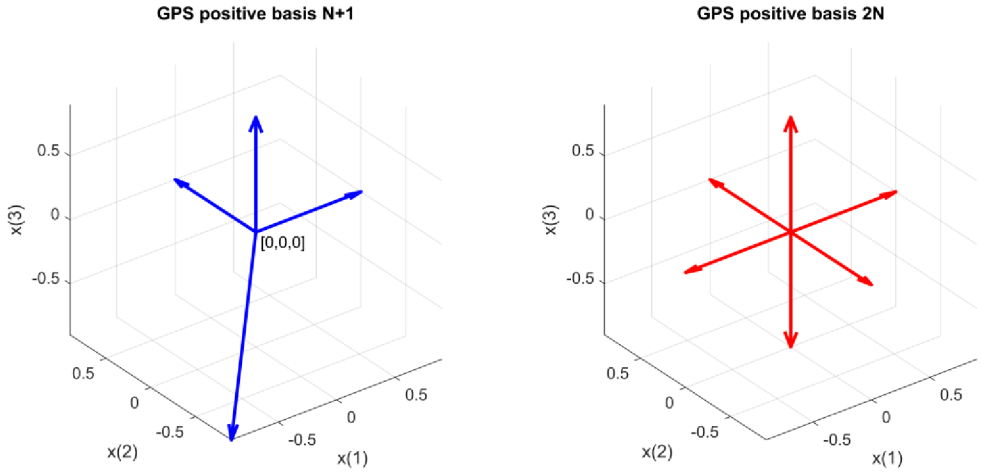
Stejně jako u algoritmu *fminsearch*, jde o „direct search“ metodu, která nevyžaduje hodnoty derivací optimalizované funkce. Na rozdíl od předchozího algoritmu však podporuje omezující podmínky a disponuje větším množstvím nastavení.

Podobně jako *fminsearch,* také tato metoda využívá prohledávání prostoru parametrů dle definovaného vzoru. Původní metoda, popsána v [36], byla dále rozšiřována a v dnešní době již existuje mnoho variant tohoto algoritmu. Jeho konkrétní implementace v MATLABu je popsána v [37–40].

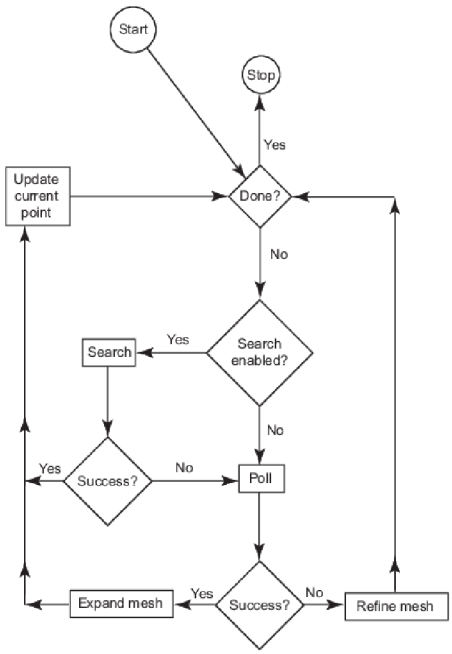
Základní vzor (v angličtině je používán termín *mesh*) je sada směrových vektorů dle kterých se vybírají další body k prohledání v okolí aktuálního bodu. U varianty GPS[2] (Generalized Pattern Search) jsou vektory fixní, u varianty MADS (Mesh Adaptive Direct Search) jsou směrové vektory voleny náhodně. Počet vektorů je závislý na počtu proměnných N (hledaných

[2] Existuje ještě varianta GSS (Generating Set Search), která pracuje identicky jako GPS, kromě případů, kdy je aktuální bod blízko omezující podmínce [74].

parametrů). Příklad dvou vzorů pro problém tří proměnných je na Obr. 8. Velikost vzoru (*mesh size*) se mění v závislosti na tom, zda algoritmus v daném kroku nalezl lepší řešení či ne (viz ilustrační příklad na Obr. 6 a diagram na Obr. 9).



Obr. 8. Směrové vektory prohledávání prostoru parametrů x(1),…,x(N), pro N = 3. Varianta GPS N+1 (vlevo) a GPS 2N (vpravo).



Obr. 9. Zjednodušený vývojový diagram funkce patternsearch [37].

Na Obr. 9 je také vidět možnost doplňkové prohledávací fáze (v tomto kontextu označována jako *Search*), která umožňuje v každé iteraci spustit v aktuálním bodě nejprve samostatnou optimalizační úlohu, zda nenajde lepší bod. Tato operace je většinou časově náročná, takže pokud je vůbec povolena, spouští se pouze v iteraci č.1, pak už probíhá jen standardní prohledávání, dle vybraného vzoru (*Poll*).

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce.
`MaxTime`	Maximální povolený čas výpočtu (v sekundách).

`PollMethod`	Varianta vzoru prohledávání prostoru:
	`'GPSPositiveBasis2N'` (výchozí)
	`'GPSPositiveBasisNp1'`
	`'GSSPositiveBasis2N'`
	`'GSSPositiveBasisNp1'`
	`'MADSPositiveBasis2N'`
	`'MADSPositiveBasisNp1'`

`SearchFcn`	Volba doplňkové prohledávací funkce:
	`[]` (prázdná funkce, výchozí)
	`'GPSPositiveBasis2N'`
	`'GPSPositiveBasisNp1'`
	`'GSSPositiveBasis2N'`
	`'GSSPositiveBasisNp1'`
	`'MADSPositiveBasis2N'`
	`'MADSPositiveBasisNp1'`
	`'searchga'` (genetický algoritmus)
	`'searchlhs'` (Latin hypercube prohledávání)
	`'searchneldermead'` (Nelder-Mead algoritmus)

`PollOrderAlgorithm`	Pořadí výběru jednotlivých směrů v prohledávacím vzoru: `'Consecutive'` (postupně, výchozí) `'Random'` (náhodně) `'Success'` (dle posledního úspěšného směru)
`MeshContractionFactor`	Koeficient zmenšení vzoru (*mesh*) pro neúspěšnou iteraci.
`MeshExpansionFactor`	Koeficient zvětšení vzoru (*mesh*) pro úspěšnou iteraci.
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 5. Vybrané možnosti nastavení funkce patternsearch.

2.2.2 Globální optimalizační metody

2.2.2.1 simulannealbnd (Simulated annealing) [41]

Syntaxe funkce v MATLABu:

```
x = simulannealbnd(fun,x0,lb,ub,options)
```

Všechny vstupní parametry mají stejný význam jako v kapitole 2.2.1.3.

Pro tuto metodu existuje i zavedený český pojem „simulované žíhání“, z důvodu konzistence se však v této práci budeme držet uvedeného anglického termínu. Metoda, původně popsaná v [42], napodobuje fyzikální proces zahřívání materiálu a následného pomalého snižování teploty

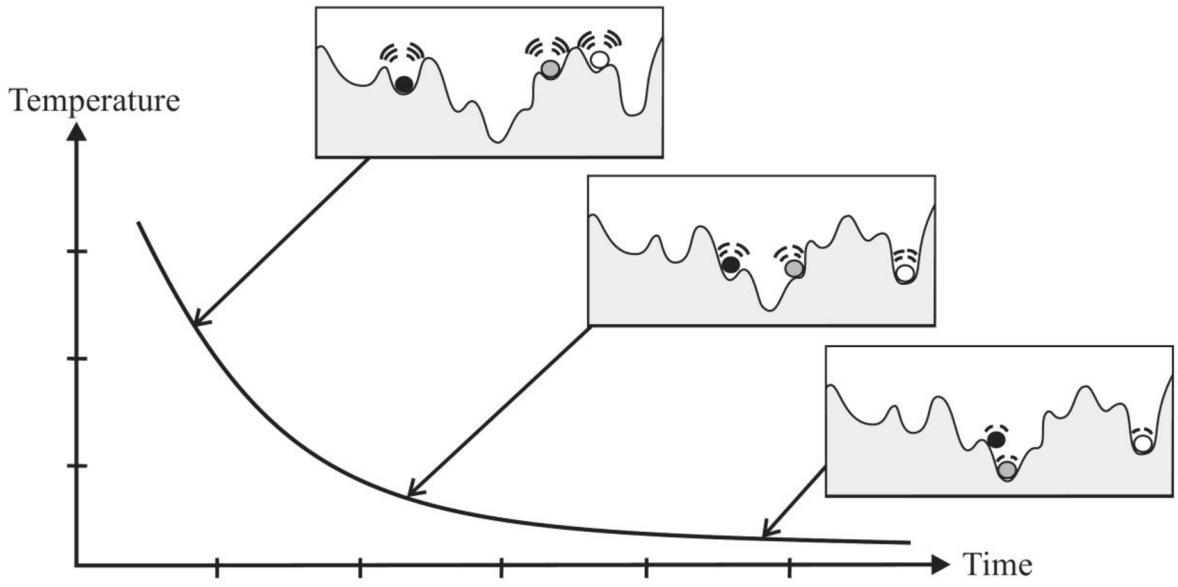
za účelem snížení počtu defektů, čímž se minimalizuje energie systému. Konkrétní implementace v MATLABu je popsána v [43, 44] a vychází z [45].

Algoritmus používá pojem *temperature* (teplota), který reprezentuje dva aspekty:

- Vzdálenost od aktuálního bodu, ve kterém probíhá hledání nových bodů.
- Pravděpodobnost přijetí nového bodu i navzdory tomu, že má vyšší hodnotu optimalizované funkce než aktuální bod.

Během iterativního procesu hledání a vyhodnocování nových bodů se teplota postupně snižuje, díky čemuž se zrychluje konvergence do lokálního minima. Ilustrace tohoto procesu je na Obr. 10. Je však také zaveden mechanismus nazvaný *reannealing* (opětovné žíhání), který po určitém počtu iterací opět zvedne teplotu, což dá algoritmu šanci na nalezení lepšího bodu i po konvergenci do lokálního minima.

Díky určité pravděpodobnosti přijetí i „horších" bodů a schopnosti periodicky zvyšovat teplotu je tento algoritmus odolný vůči uvíznutí v lokálním minimu. Je zde také možnost spustit po poslední iteraci další, alternativní optimalizační metodu, která může pomoci zpřesnit dosažený výsledek (viz Tabulka 6 níže).



Obr. 10. Princip metody simulovaného žíhání. [46]

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce.
`MaxTime`	Maximální povolený čas výpočtu (v sekundách).
`InitialTemperature`	Počáteční hodnota parametru teploty.
`ReannealInterval`	Počet iterací algoritmu, po kterých dojde ke zvýšení parametru teploty.
`TemperatureFcn`	Výběr funkce, která určuje profil změny teploty během iterací (více viz [43]):

	`'temperatureexp'` (výchozí)
	`'temperatureboltz'`
	`'temperaturefast'`

`HybridFcn`	Výběr funkce, která pokračuje v optimalizaci po ukončení poslední iterace:
	`[]` (prázdná funkce, výchozí)
	`'fminsearch'`
	`'patternsearch'`
	`'fminunc'`
	`'fmincon'`

`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 6. Vybrané možnosti nastavení funkce simulannealbnd.

2.2.2.2 ga (Genetic algorithm) [47]

Syntaxe funkce v MATLABu:

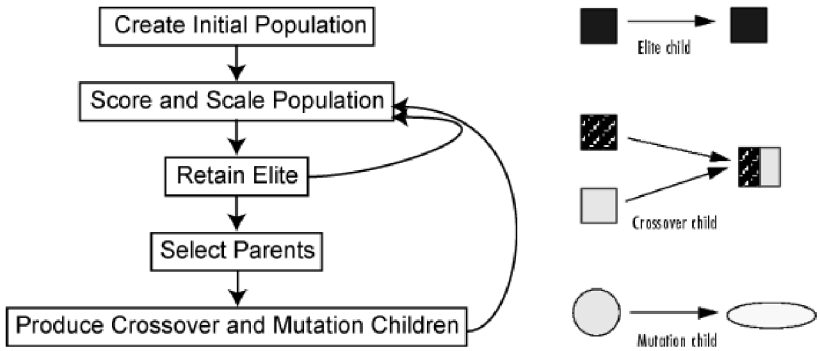
```
x = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Všechny vstupní parametry mají stejný význam jako v kapitole 2.2.1.2 s tím rozdílem, že oproti většině předchozích algoritmů není vstupem počáteční odhad hledaných parametrů, ale jejich počet `nvars`.

Optimalizace s využitím genetického algoritmu, původně popsaná v [48], je založena na přírodním výběru, procesu, který vychází z evoluční biologie. Genetický algoritmus opakovaně upravuje populaci jednotlivých řešení. V každém kroku genetický algoritmus vybere jedince z aktuální populace jako rodiče a použije je k vytvoření potomků pro další generaci. V průběhu dalších generací se populace "vyvíjí" směrem k optimálnímu řešení. Konkrétní implementace v MATLABu je popsána v [49–51].

Genetický algoritmus používá v každém kroku tři hlavní typy pravidel výběru jedinců pro vytvoření další generace z aktuální populace (viz Obr. 11) [52]:

- **Pravidla výběru** (*Selection*) vybírají jedince, tzv. rodiče, kteří přispějí do populace v příští generaci. Výběr je obecně stochastický a může záviset na skóre jedinců.
- **Pravidla křížení** (*Crossover*) spojují dva rodiče a vytvářejí potomky pro další generaci.
- **Pravidla mutace** (*Mutation*) aplikují náhodné změny na jednotlivé rodiče, aby se vytvořili potomci.



Obr. 11. Základní princip fungování genetického algoritmu v MATLABu. [52]

Oproti všem dosud zmíněným algoritmům je genetický algoritmus odlišný v tom, že si neuchovává pouze jeden aktuálně nejlepší bod řešení, ale má k dispozici celou sadu (populaci) těchto bodů. Podobně jako u simulovaného žíhání hraje nezanedbatelnou roli i náhoda a je zde taktéž možnost spustit po poslední iteraci další, alternativní optimalizační metodu.

Vzhledem k tomu, že vstupním argumentem funkce není počáteční odhad neznámých parametrů, je počáteční populaci možné zadat explicitně, nebo je vytvořena automaticky mezi limity parametrů `lb`, `ub` (pokud jsou zadány). Více viz Tabulka 7 níže.

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`MaxGenerations`	Maximální počet iterací algoritmu (generací).
`MaxTime`	Maximální povolený čas výpočtu (v sekundách).
`PopulationSize`	Velikost populace.
`InitialPopulationMatrix`	Matice počáteční populace. Má až `PopulationSize` řádků a *N* sloupců, kde *N* je počet hledaných parametrů.
`InitialPopulationRange`	Rozsah limitů pro vygenerování počáteční populace. Pokud není zadáno, tak platí výchozí nastavení: `[-10;10]` pro prvky bez omezení `[lb;ub]` pro prvky se specifikovanými hranicemi
`CrossoverFraction`	Koeficient zlomku z celkové populace, který je vytvořen křížením (*crossover*).
`EliteCount`	Počet nejlepších jedinců (*elite*), kteří mají garantované přežití do další generace.

`HybridFcn`	Výběr funkce, která pokračuje v optimalizaci po ukončení poslední iterace:
	`[]` (prázdná funkce, výchozí)
	`'fminsearch'`
	`'patternsearch'`
	`'fminunc'`
	`'fmincon'`

`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 7. Vybrané možnosti nastavení funkce ga.

2.2.2.3 surrogateopt (Surrogate optimization) [53]

Syntaxe funkce v MATLABu:

```
x = surrogateopt(fun,lb,ub,options)
```

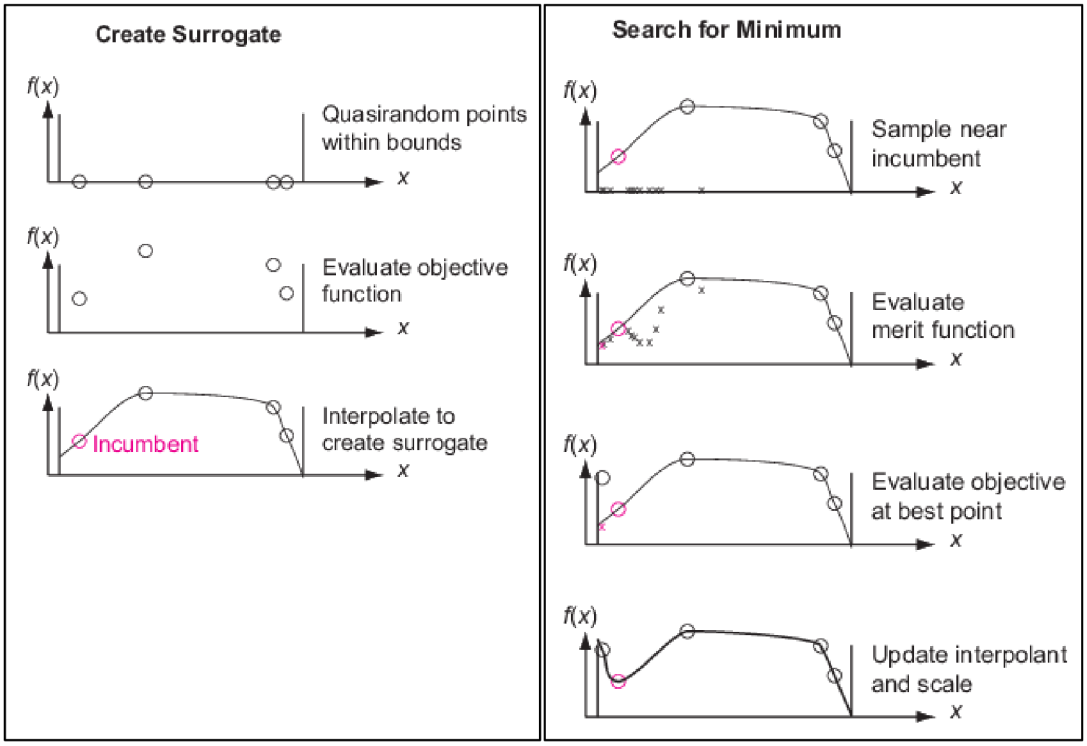
Všechny vstupní parametry mají stejný význam jako v kapitole 2.2.1.3, jediná odlišnost je, že tato funkce nemá jako vstup počáteční odhady hledaných parametrů.

Jak již bylo popsáno v úvodu kapitoly 2.2, jedná se o algoritmus využívající tzv. náhradní model (*surrogate*) metodou postupného iterativního vytváření aproximace prohledávaného prostoru parametrů. Tento náhradní model pak umožňuje efektivnější prohledávání prostoru parametrů, zejména ve chvíli, kdy je výpočet dodané (hodnoticí) funkce časově náročný. Konkrétní implementace v MATLABu je popsána v [54] a vychází z [55, 56].

Vzhledem k principu jeho fungování, je pro tento algoritmus nezbytné dodat limity hledaných parametrů `lb`, `ub`.

Zjednodušeně, algoritmus střídá dvě fáze (viz Obr. 12 níže):

- **Konstrukce náhradního modelu** (*Construct Surrogate*) – Vygenerování náhodných bodů mezi dodanými limity, ve kterých je pak vyčíslena hodnota zadané hodnoticí funkce. Následně je vytvořen náhradní model hodnoticí funkce pomocí interpolace mezi těmito vyhodnocenými body.
- **Hledání minima** (*Search for Minimum*) – Hledání minima hodnoticí funkce na náhradním modelu testováním několika tisíc náhodných bodů. Vyhodnocení těchto bodů na náhradním modelu je velmi rychlé. Nejlepší z těchto bodů je následně vyhodnocen na reálné (původní) hodnoticí funkci a náhradní model je pomocí tohoto bodu aktualizován.



Obr. 12. Princip fungování surrogate algoritmu v MATLABu. [53]

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`MaxFunctionEvaluations`	Maximální počet vyhodnocení optimalizované funkce.
`MaxTime`	Maximální povolený čas výpočtu (v sekundách).
`MinSampleDistance`	Minimální vzdálenost mezi testovanými body na náhradním modelu.
`MinSurrogatePoints`	Minimální počet náhodných bodů pro konstrukci náhradního modelu, generovaných na začátku optimalizace.
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 8. Vybrané možnosti nastavení funkce surrogateopt.

2.2.2.4 particleswarm (Particle swarm optimization) [57]

Syntaxe funkce v MATLABu:

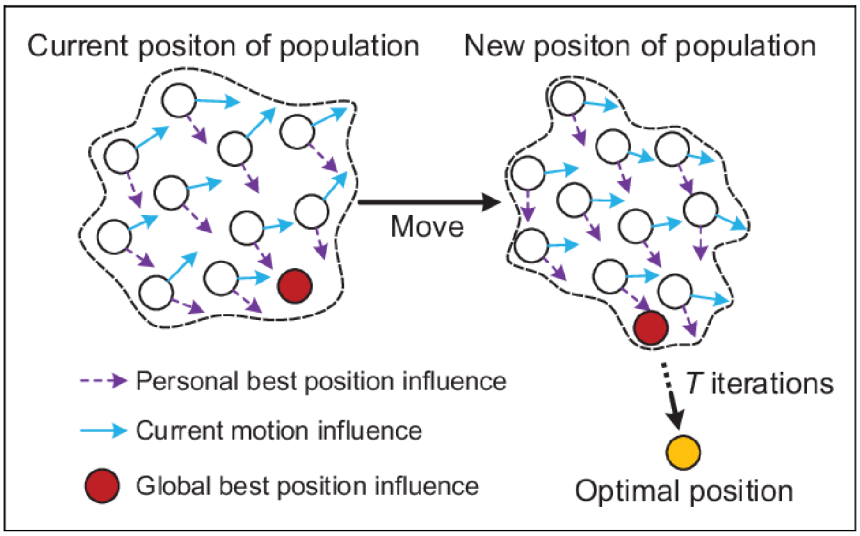
```
x = particleswarm(fun,nvars,lb,ub,options)
```

Všechny vstupní parametry mají stejný význam jako v kapitole 2.2.2.2.

Particle swarm algoritmus (česky *hejno částic*) byl původně popsaný v [58]. Konkrétní implementace v MATLABu je popsána v [59] a je založena na modifikacích z [60], [61].

Tento algoritmus, podobně jako genetický algoritmus, nepracuje pouze s jedním aktuálním bodem, ale se sadou různých bodů, která se v každé iteraci aktualizuje.

Každá částice je definována svojí polohou, rychlostí a pamětí předchozích úspěchů při hledání. Částice jsou ovlivňovány ostatními úspěšnějšími částicemi hejna. Úspěšnost částice je daná hodnoticí funkcí. Algoritmus iterativně aktualizuje pohyb hejna a upravuje hodnoty popisující částice [62]. Ilustrační obrázek popisující princip tohoto algoritmu je na Obr. 13.



Obr. 13. Princip particle swarm algoritmu [63].

Mezi nejdůležitější dodatečná nastavení pomocí vstupního parametru `options` patří:

`MaxIterations`	Maximální počet iterací algoritmu.
`MaxTime`	Maximální povolený čas výpočtu (v sekundách).
`SwarmSize`	Počet částic.
`InitialSwarmMatrix`	Matice počáteční populace částic. Má až `SwarmSize` řádků a *N* sloupců, kde *N* je počet hledaných parametrů.
`MinNeighborsFraction`	Koeficient zlomku počtu bodů z celkové populace, které tvoří minimální počet okolních bodů, které daná částice také vyhodnocuje.

`HybridFcn`	Výběr funkce, která pokračuje v optimalizaci po ukončení poslední iterace:
	`[]` (prázdná funkce, výchozí)
	`'fminsearch'`
	`'patternsearch'`
	`'fminunc'`
	`'fmincon'`

`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé iterace optimalizačního algoritmu.

Tabulka 9. Vybrané možnosti nastavení funkce particleswarm.

2.2.2.5 GlobalSearch [64]

Syntaxe funkce v MATLABu:

```
gs = GlobalSearch(Name,Value)
```

```
problem = createOptimProblem('solverName','options',options)
```

```
x = run(gs,problem)
```

Postup nastavení spuštění tohoto optimalizačního algoritmu je o něco složitější než v předchozích případech. Vzhledem k principu jeho fungování, který bude popsán níže, je nutné provést zvlášť nastavení samotného GlobalSearch algoritmu (`gs`) a také podřízeného lokálního algoritmu (`problem`). Následně je spuštěna optimalizační úloha příkazem `run`.

Argumenty funkce `GlobalSearch(Name,Value)` slouží k úpravě výchozích parametrů hlavního algoritmu, podobně jako v případě předchozích algoritmů. Pro nastavení parametrů lokálního algoritmu pomocí funkce `createOptimProblem` slouží argument `options` (stejně jako v kapitolách výše). Pomocí argumentu `options` jsou také předávány počáteční odhady (`x0`), reference na hodnoticí funkci (`objective`), a limity hledaných parametrů (`lb`, `ub`). Argument `'solverName'`, označující název lokálního algoritmu, může u GlobalSearch nabývat pouze hodnoty `'fmincon'`.

Algoritmus GlobalSearch je založen na publikaci [65] a jeho konkrétní implementace v MATLABu je popsána v [66]. Jeho podstatou je vygenerování náhodných zkušebních bodů

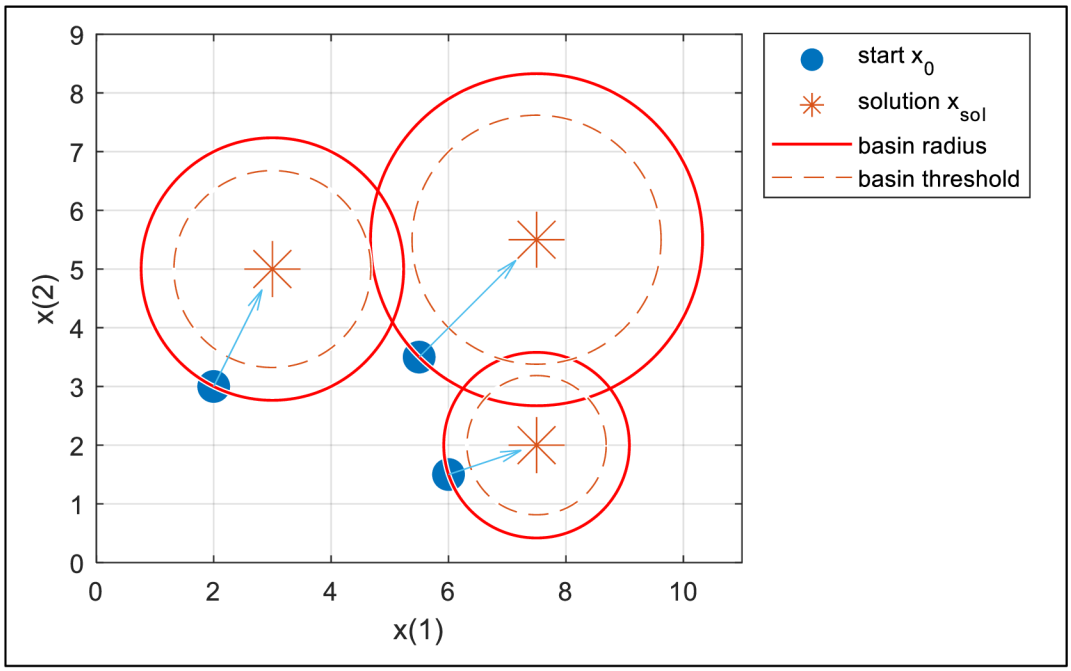
v prostoru parametrů na začátku algoritmu, jejich filtrace na základě výsledků hodnoticí funkce a následně postupné spouštění lokálního optimalizačního algoritmu z vybraných bodů.

Níže uvedený popis algoritmu vystihuje jeho hlavní rysy, ale nezahrnuje všechny implementační podrobnosti. Ty jsou uvedeny ve výše uvedených zdrojích.

Filtrace bodů k otestování probíhá na základě vytvoření spádové oblasti (*basin*), kolem každého nalezeného řešení. Předpokládá se její sférický (resp. hyperkulový) tvar s poloměrem daným vzdáleností mezi počátečním a finálním bodem řešení. Tento poloměr, zmenšen o volitelný koeficient (*basin threshold*), pak určuje oblast, ve které jsou případné další počáteční body vyřazeny. Ilustrační příklad pro tři dosud nalezená řešení problému se dvěma parametry je na Obr. 14.

GlobalSearch má několik fází, které můžeme zjednodušeně popsat takto:

1. Spuštění lokálního optimalizačního algoritmu (fmincon) z bodu počátečního odhadu.
2. Vygenerování náhodných bodů (*Trial Points*) mezi zadanými limity a výpočet hodnoticí funkce pro omezenou podmnožinu bodů (*Stage 1 Points*).
3. Seřazení vyhodnocených bodů (*Stage 1 Points*) a spuštění lokálního optimalizačního algoritmu z nejlepšího bodu.
4. Inicializace spádových oblastí (*basins*) pro dosud nalezená řešení.
5. Procházení zbývajících bodů (*Trial Points*)
    - Pokud bod spadá do spádové oblasti již nalezeného řešení, není lokální optimalizace spuštěna.
    - Pokud bod do žádné spádové oblasti nespadá, je spuštěn lokální optimalizační algoritmus a výsledek je přidán mezi ostatní dosud nalezená řešení.
6. Po vyčerpání všech dostupných bodů je předána nejlepší hodnota ze sady nalezených řešení.



Obr. 14. Princip spádových oblastí řešení (basins) u GlobalSearch.

Možnosti nastavení lokálního optimalizačního algoritmu (fmincon) jsou stejné, jako v tabulce u výše jeho výše uvedeného popisu (Tabulka 2). Mezi nejdůležitější dodatečná nastavení pomocí vstupních argumentů funkce GlobalSearch patří:

`MaxTime`	Maximální povolený čas výpočtu (v sekundách).
`NumTrialPoints`	Celkový počet vygenerovaných náhodných bodů.
`NumStageOnePoints`	Počet bodů, které jsou vyhodnocené v první fázi algoritmu.
`StartPointsToRun`	Nastavení, které body mají být testovány: `'all'` (všechny body, výchozí) `'bounds'` (pouze body, které splňují nastavené limity) `'bounds-ineqs'` (pouze body, které splňují nastavené limity a omezující podmínky nerovnosti)
`DistanceThresholdFactor`	Koeficient úpravy poloměru spádové oblasti (*basin*) pro určení, zda bod do této oblasti spadá: `0..1` = zmenšení poloměru `1..Inf` = zvětšení poloměru
`MaxWaitCycle`	Maximální počet po sobě jdoucích testovaných bodů, které se mohou ocitnout v konkrétní spádové oblasti, než dojde k jejímu umělému zmenšení.
`BasinRadiusFactor`	Koeficient umělého zmenšení poloměru spádové oblasti, pokud se v ní ocitne předem definovaný počet po sobě jdoucích bodů. Může nabývat pouze hodnot `0..1`.
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé (globální) iterace algoritmu.

Tabulka 10. Vybrané možnosti nastavení funkce GlobalSearch.

2.2.2.6 MultiStart [67]

Syntaxe funkce v MATLABu:

```
ms = MultiStart(Name,Value)

problem = createOptimProblem('solverName','options',options)

x = run(ms,problem,k)
```

Nastavení a spuštění této metody je velmi podobné jako u GlobalSearch (2.2.2.5), pouze se dvěma drobnými rozdíly. Při spuštění optimalizační úlohy příkazem `run` je nutné navíc dodat ještě argument `k`, který určuje počet náhodně generovaných počátečních bodů, který algoritmus testuje (případně je místo něj možné specifikovat přímo hodnoty jednotlivých hledaných parametrů pro jednotlivé počáteční body). Druhým rozdílem je, že argument `'solverName'`,

označující název lokálního algoritmu, může na rozdíl od GlobalSearch nabývat více hodnot, konkrétně `'fmincon'`, `'fminunc'` nebo `'lsqnonlin'`[3].

Algoritmus MultiStart funguje v principu téměř stejně jako GlobalSearch a jeho podstata a implementace v MATLABu vychází ze stejných zdrojů [65, 66]. MultiStart lze brát jako zjednodušenou verzi GlobalSearch, přičemž hlavní rozdíly jsou tyto:

- MultiStart spouští lokální optimalizační algoritmus ze všech počátečních bodů, kromě těch, které nesplňují omezující podmínky. Není prováděna žádná filtrace bodů. Finální řešení je vybráno podle nejnižší hodnoticí funkce ze všech prozkoumaných bodů.
- MultiStart má možnost volby více variant lokálních optimalizačních algoritmů (viz první odstavec kapitoly).
- Jednotlivé výpočty lokálních optimalizací z různých bodů lze díky jejich nezávislosti paralelizovat.
- MultiStart umožňuje kromě generování náhodných počátečních bodů, zadat i konkrétní uživatelem definované body v prostoru parametrů.

Nastavení lokálních optimalizačních algoritmů stejné je možné dohledat v předchozích kapitolách (viz 2.2.1).

Mezi nejdůležitější dodatečná nastavení pomocí vstupních argumentů funkce MultiStart patří:

`MaxTime`	Maximální povolený čas výpočtu (v sekundách).
`StartPointsToRun`	Nastavení, které body mají být testovány: `'all'` (všechny body, výchozí) `'bounds'` (pouze body, které splňují nastavené limity) `'bounds-ineqs'` (pouze body, které splňují nastavené limity a omezující podmínky nerovnosti)
`OutputFcn`	Reference na uživatelem specifikovanou funkci, která se spustí na konci každé (globální) iterace algoritmu.

Tabulka 11. Vybrané možnosti nastavení funkce MultiStart.

[3] Je možné nastavit i algoritmus `'lsqcurvefit'`, který ale pro naši úlohu není vhodný.

3 Formulace problému a cílů dizertační práce

V předchozích kapitolách jsme uvedli hlavní aspekty úlohy odhadu parametrů matematických modelů, kterou se v této práci chceme zabývat. Dále byly představeny nejdůležitější nástroje a optimalizační metody v programu MATLAB, které jsou pro řešení této úlohy k dispozici. MATLAB byl zvolen z důvodu jeho rozšířenosti v inženýrské praxi i v oblasti výzkumu.

Tuto oblast považujeme za velmi aktuální, vzhledem k současným požadavkům na rychlé prototypování řídicích systémů, což vyžaduje přesný model řízené soustavy. Spolu s rozšířením technik návrhu na základě modelu (Model-based design, MBD) do inženýrské praxe je nutné se zabývat i požadavkem na co největší automatizaci celého procesu odhadu parametrů modelu, analýzu výsledků a eliminaci možných chyb ze strany uživatele.

Na základě vlastních praktických zkušeností v této oblasti v rámci výzkumu a vývoje, a dále na základě zkušeností z výuky této látky v magisterském studijním programu[4] bylo vypozorováno několik hlavních problémů:

- I relativně jednoduché modely vytvořené pomocí nástrojů fyzikálního modelování mají nezanedbatelnou dobu výpočtu.
- Úloha hledání parametrů u mechatronických modelů není triviální, zejména s rostoucím počtem parametrů a nepřesností počátečního odhadu.
- V prostoru hledaných parametrů optimalizační úlohy se často vyskytují lokální minima.
- V případě omezených vstupních informací o matematickém modelu není snadné provést výběr vhodné optimalizační metody a nastavení úlohy, včetně počátečních odhadů parametrů a jejich limitů.
- V případě výběru lokálních optimalizačních metod, které konvergují relativně rychle, je nutné, aby uživatel úlohu neustále dozoroval, upravoval její nastavení či odhady parametrů, a znovu spouštěl.
- V případě výběru globálních optimalizačních metod je nutnost neustálých zásahů o něco menší, i tak však není snadné předem odhadnout rozsah úlohy. Pravděpodobnost nalezení globálního minima je vyšší než u lokálních metod, avšak výpočetní čas je často násobně vyšší, přičemž není snadné algoritmus přerušit během výpočtu.
- Nevhodnou volbou počtu odhadovaných parametrů či formátu dat naměřených na reálném systému si uživatel komplikuje úlohu a zvyšuje dobu výpočtu.

Na základě výše uvedených poznatků pak byly formulovány následující cíle dizertační práce:

1. **Vytvoření souboru technických modelů a jejich analýza**

Pro inženýrskou praxi není výhodné studovat problematiku identifikace sytému a odhadu parametrů obecně, jelikož obecná řešení z principu nemohou řešit problémy spojené s aplikací na konkrétní modely.

Pro vývoj efektivních postupů odhadu parametrů je tedy výhodné zaměřit se pouze na vybrané typy modelů, na nichž lze nové algoritmy otestovat a zhodnotit výsledky. Na základě těchto poznatků pak lze z těchto modelů vycházet při práci se složitějšími systémy.

[4] Předmět RPO – Simulace a řízení v reálném čase (Studijní program Mechatronika)

Výstupem bude sada mechatronických modelů s konkrétními parametry. Tato sada by měla zahrnovat zejména mechanické a elektrické systémy, se kterými se v inženýrské praxi často setkáváme.

S pomocí těchto modelů bude poté probíhat další výzkum v rámci níže uvedených cílů, kdy proběhne jejich analýza a návrh vhodných postupů při odhadu jejich parametrů v praxi.

2. **Kombinace globálních a lokálních optimalizačních metod pro mechatronické modely**

V praxi je na uživateli, aby správně nastavil optimalizační úlohu pro danou aplikaci což může vést ke komplikacím, způsobených výběrem nevhodné optimalizační metody, okrajových podmínek či počátečních odhadů. Tento praktický problém je ještě markantnější v případech, kdy má uživatel na počátku jen velmi málo informací o vlastnostech daného systému.

Jedním z cílů této práce je tedy výzkum a vývoj nových postupů pro identifikaci parametrů systému s využitím kombinace globálních a lokálních optimalizačních metod a jejich praktická implementace pro rychlejší a spolehlivější nalezení globálního optima.

Výstupem bude algoritmus hledání parametrů daného modelu kombinující globálního a lokální optimalizačních metody. Cílem je vytvořit algoritmus pro konkrétní sadu modelů (viz první cíl) s malou citlivostí na případné špatné počáteční odhady parametrů, vysokou pravděpodobností nalezení globálního optima, a co nerychlejší konvergencí k němu.

3. **Vytvoření algoritmů pro detekci a analýzu nevhodně navržených úloh odhadu parametrů**

Jedním z důvodů neúspěšného či pomalého odhadu parametrů je kromě použité optimalizační metody i nevhodná struktura modelu, počet parametrů či podoba vstupních naměřených dat. Dalším cílem této práce je tedy výzkum a vývoj metod pro odhalení některých těchto nedostatků.

Výstupem bude sada algoritmů, které budou schopny alespoň částečně vyřešit otázky týkající se vzájemné závislosti parametrů modelu a jejich separovatelnosti, vhodného vzorkování vstupních naměřených dat a jejich redundance.

4. **Nový nástroj pro odhad parametrů v inženýrské praxi**

Nástroje pro odhad parametrů v programu MATLAB mají v současné době omezenou funkcionalitu a pro netriviální případy musí uživatel nutně znát jejich úskalí a být schopen správně vyhodnotit jejich výsledky, aby se vyhnul jejich nevhodné interpretaci.

Výsledkem bude sada nástrojů (funkcí) v programu MATLAB, které prakticky implementují algoritmy vyvinuté v rámci předchozích cílů (2 a 3).

4 Modely mechatronických systémů

V této kapitole budou popsány modely vytvořené v prostředí MATLAB Simulink, s využitím nástroje pro fyzikální modelování Simscape.

Modely byly vytvořeny tak, aby zahrnovaly nějaký druh spíše jednoduššího mechanismu, kombinovaly mechanické i elektrické členy a obsahovaly nelinearity typu doraz, netriviální model tření apod. Ne u každého z modelů se však všechny tyto prvky vyskytují, některé jsou jednodušší, nebo zaměřené jen na konkrétní oblast.

Většina modelů má alespoň jeden vstup, většinou buzení silou, a jeden či více výstupů. Parametry modelů jsou voleny tak, aby chování modelu bylo co nejvíc reálné, stejně jako hodnoty těchto parametrů.

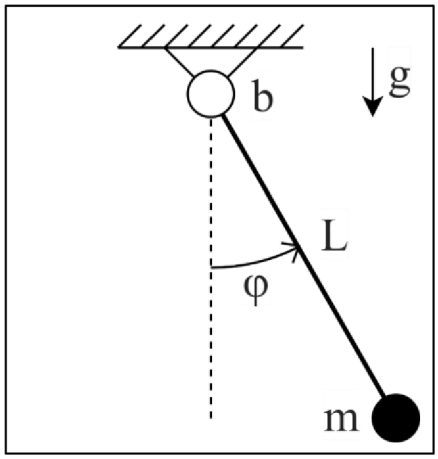
Nejdříve budou popsány jednotlivé modely a jejich vstupy, výstupy a parametry. Následně se budeme věnovat generování vhodných vstupů (buzení) pro tyto modely a následně provedeme u všech zmapování prostoru vybraných parametrů, abychom získali lepší představu o jejich tvaru a povaze.

4.1 Popis jednotlivých modelů

4.1.1 Jednoduché kyvadlo

Tento model představuje jednoduché kyvadlo (Obr. 15) a jako jediný z celé sady nevyužívá nástroje Simscape, pouze základních bloků (Obr. 16) sestavených na základě rovnice (4.3). Obsahuje hmotný bod na nehmotné tyči, která je uchycena do rotační vazby s viskózním třením. Na kyvadlo samozřejmě působí tíhové zrychlení[5].

Výhodou tohoto modelu je mnohonásobně nižší výpočetní náročnost než u ostatních vytvořených systémů. Díky tomu je vhodný pro počáteční rychlé testování optimalizačních algoritmů i pro jakékoliv jiné, časově náročné experimenty.



Obr. 15. Schematický diagram modelu Jednoduché kyvadlo.

[5] Pro tento i další modely považujeme tíhové zrychlení za konstantu s hodnotou $g = 9{,}81\ m/s^2$. Tuto konstantu pak již v tabulce parametrů znovu explicitně neuvádíme.

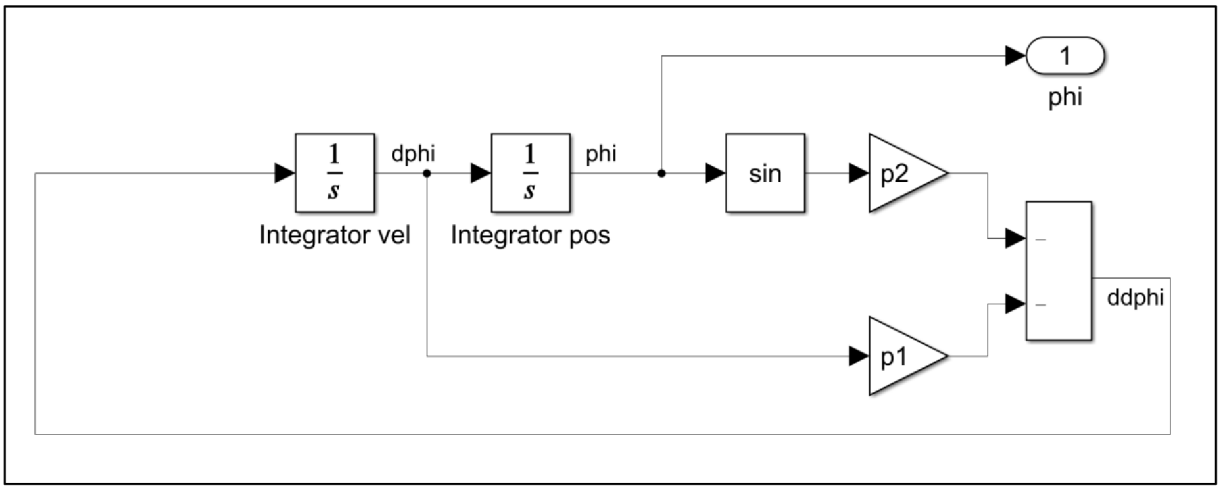
Dalším specifikem modelu je, že nemá žádný vstup. Jeho vybuzení je pak docíleno změnou počátečního stavu, který je reprezentován dvěma z celkových čtyř parametrů systému (viz Tabulka 12). Tyto parametry počátečního stavu kyvadla, považujeme v této úloze za pevně dané.

Zbylé dva parametry jsou tzv. substituční parametry, které nahrazují kombinaci několika konkrétních fyzikálních vlastností kyvadla (viz rovnice (4.1) až (4.3)). Tato substituce je provedena jen u tohoto modelu, u ostatních už ne. Je to jednak z důvodu zjednodušení celého modelu a taky pro demonstraci jevu, že ne vždy je možné na základě úspěšné estimace získat unikátní řešení pro každý jednotlivý parametr.

$$mL^2\ddot{\varphi} = -b\dot{\varphi} - mgLsin(\varphi) \tag{4.1}$$

$$\ddot{\varphi} = -\frac{b}{mL^2}\dot{\varphi} - \frac{g}{L}\sin(\varphi) \tag{4.2}$$

$$\ddot{\varphi} = -p_1\dot{\varphi} - p_2\sin(\varphi) \tag{4.3}$$



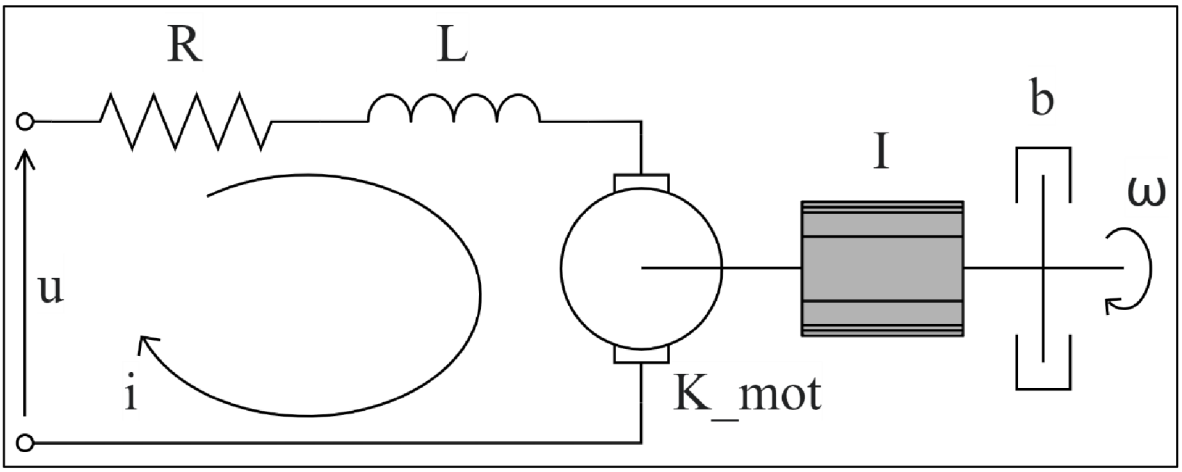
Obr. 16. Implementace modelu Jednoduché kyvadlo v Simulinku.

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
phi	Out 1	rad	Natočení kyvadla
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
p1	0,034	$N * s/(m * rad * kg)$	Substituční parametr
p2	47,7	s^{-2}	Substituční parametr
phi0	$-0{,}4$	rad	Počáteční natočení kyvadla
dphi0	-8	rad/s	Počáteční rychlost kyvadla

Tabulka 12. Vstupy, výstupy a parametry modelu Jednoduché kyvadlo.

4.1.2 DC motor

Tento model představuje základní verzi stejnosměrného motoru (Obr. 17) a obsahuje hlavní prvky elektrické části (odpor, indukčnost vinutí), mechanické části (moment setrvačnosti, viskózní tření) a elektromechanickou přeměnu energie. Vstupem do systému je napětí na svorkách motoru, výstupem pak jeho úhlová rychlost.



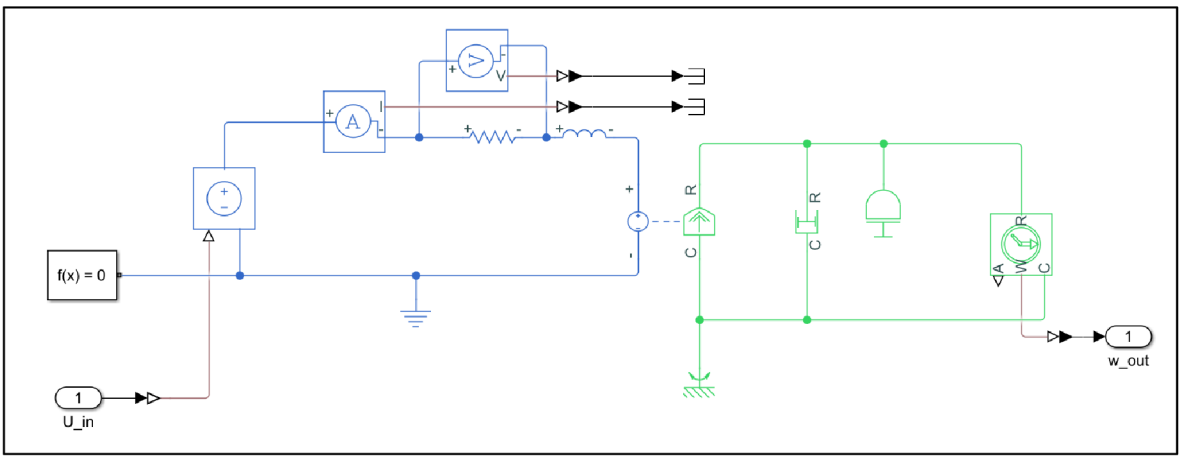
Obr. 17. Schematický diagram modelu DC motor.

Model je vytvořen s využitím Simscape knihoven pro elektrické a mechanické systémy, což je v Simulinku zvýrazněno různými barvami bloků (Obr. 18). Rozhraní mezi elektrickou a mechanickou částí je tvořeno blokem obecného rotačního elektromechanického převodníku [68], jenž přepokládá bezztrátovou přeměnu elektromechanickou přeměnu energie, a je popsán níže uvedenými rovnicemi (4.4) a (4.5):

$$T = K \cdot i \tag{4.4}$$

$$u = K \cdot \omega \tag{4.5}$$

kde u je napětí, i je proud, T je moment, ω jsou otáčky a K tzv. konstanta proporcionality s ekvivalentními jednotkami Nm/A nebo $V/(rad/s)$. Vzhledem k tomu, že se předpokládá bezztrátová přeměna elektromechanické energie, je možné použít stejnou konstantu v momentové i napěťové rovnici.



Obr. 18. Implementace modelu DC motor v Simulinku.

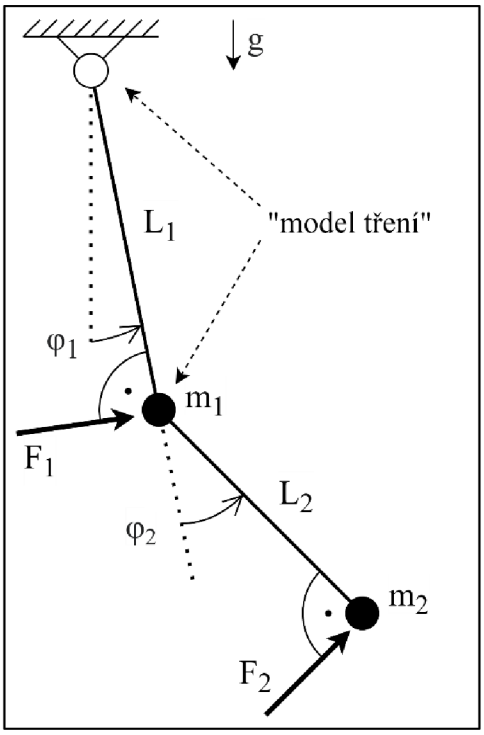
Seznam všech parametrů a jejich hodnot je uveden v tabulce níže (Tabulka 13).

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
U_in	In 1	V	Napětí na vinutí rotoru
w_out	Out 1	rad/s	Úhlová rychlost rotoru
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
I	$3{,}2 \cdot 10^{-4}$	$kg \cdot m^2$	Moment setrvačnosti rotoru
K_mot	0,12	$V/(rad/s)$	Konstanta proporcionality elektromechanického převodníku
L	$1{,}6 \cdot 10^{-3}$	H	Indukčnost vinutí
R	2,83	Ω	Odpor vinutí
b	0,01	$N \cdot m \cdot s/rad$	Koeficient viskózního tření

Tabulka 13. Vstupy, výstupy a parametry modelu DC motor.

4.1.3 Dvojité kyvadlo

Tento model je rozšířením jednoduché varianty z předchozí kapitoly 4.1.1, avšak na rozdíl od něj, obsahuje o jedno rameno víc a také na něj mohou působit dvě vnější síly, které jsou vždy kolmé na dané rameno (viz Obr. 19). V obou vazbách je pak implementován netriviální model tření, který je podrobněji popsán níže. Počáteční stav modelu je v ustálené poloze s nulovým natočením obou ramen.



Obr. 19. Schematický diagram modelu Dvojité kyvadlo.

Jak již bylo zmíněno, v obou rotačních vazbách je implementován model tření, který vychází z článku [69]. Rovnice pro výpočet třecí síly (4.6) jej uvedena níže:

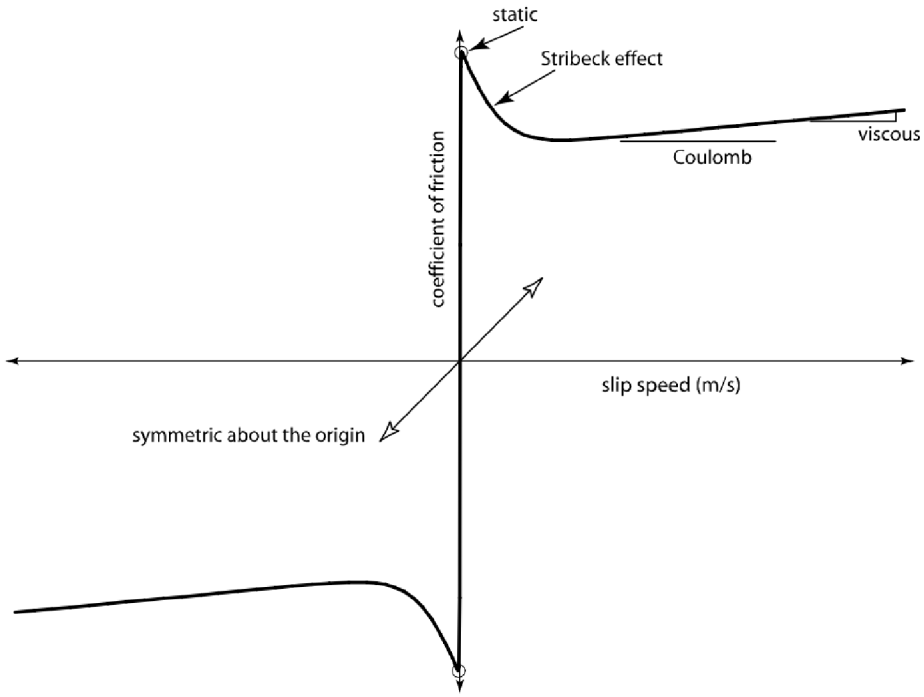
$$f(\dot{q}) = \gamma_1(\tanh(\gamma_2\dot{q}) - \tanh(\gamma_3\dot{q})) + \gamma_4 \tanh(\gamma_5\dot{q}) + \gamma_6\dot{q} \tag{4.6}$$

kde $f(\dot{q})$ představuje velikost třecí síly v závislosti na zobecněné souřadnici q, která v našem případě představuje úhel natočení ($\dot{q}$ je tedy úhlová rychlost ramene).

Parametry $\gamma_i \in \mathbb{R} \; \forall \; i = 1{,}2\,, \ldots 6$ pak představují neznámé kladné konstanty, jejichž význam lze stručně shrnout takto:

- Koeficient statického tření je aproximován pomocí parametrů $\gamma_1 + \gamma_4$.
- První člen rovnice, $\tanh(\gamma_2\dot{q}) - \tanh(\gamma_3\dot{q})$, zachycuje tzv. Striebeckův efekt[6].
- Coulombovské tření je modelováno výrazem $\gamma_4 \tanh(\gamma_5\dot{q})$.
- Viskózní tření je vyjádřeno posledním členem $\gamma_6\dot{q}$.

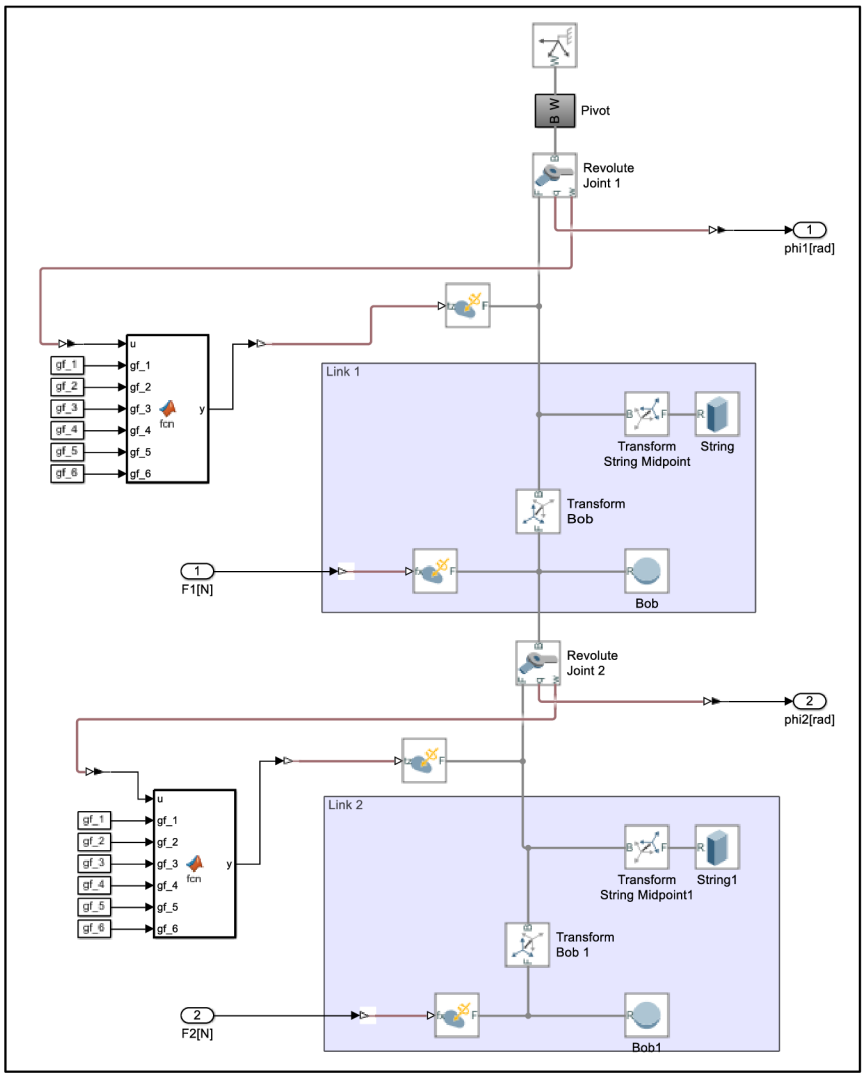
Ilustrační obrázek zahrnující vlivy všech výše uvedených druhů tření je na Obr. 20 níže.



Obr. 20. Charakteristika modelu tření [69]

Samotný model v Simulinku je zobrazen níže (Obr. 21). V tomto modelu je poprvé využito další knihovny z nástroje Simscape, která má název Simscape Multibody a slouží pro modelování mechanických systémů. K blokům z této knihovny byly poté přidány silové účinky výše uvedeného modelu tření, které jsou počítány pomocí vloženého bloku s funkcí napsanou v MATLABu.

[6] Striebeckův efekt je charakteristický pokles třecí síly který se projevuje při nízkých rychlostech. Podrobnější popis lze nalézt například v [75].



Obr. 21. Implementace modelu Dvojité kyvadlo v Simulinku.

Seznam všech parametrů a jejich hodnot je uveden v tabulce níže (Tabulka 14). Je zde důležité zmínit, že parametry nazvané b1 a gf_6 jsou redundantní, to samé platí pro b2 a gf_6. Tato volba je záměrná, zejména z důvodu demonstrace možné závislosti či neseparovatelnosti některých parametrů a také pro testování algoritmů, které mohou tyto jevy detekovat (více viz kap. 6.1).

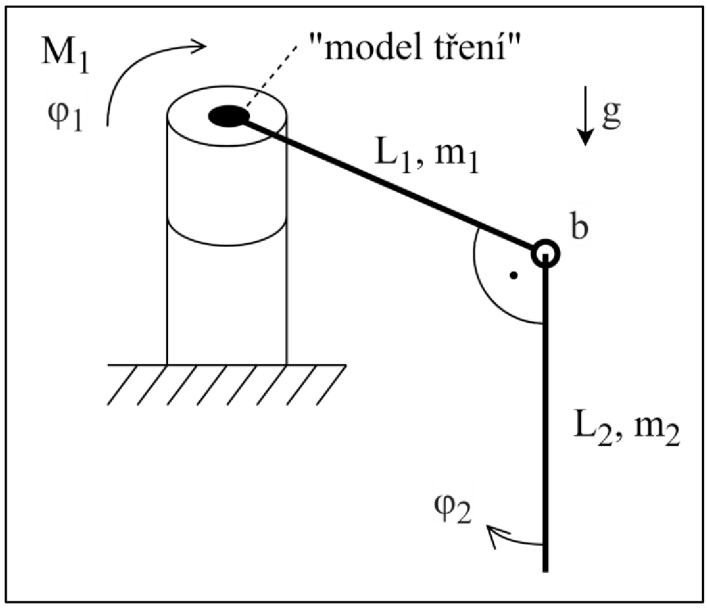
Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
F1	In 1	N	Síla kolmá na rameno 1
F2	In 1	N	Síla kolmá na rameno 2
phi1	Out 1	rad	Natočení ramene 1
phi2	Out 2	rad	Natočení ramene 2
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
L1	0,9	m	Délka ramene 1
L2	1,7	m	Délka ramene 2
b1	11,3	$N \cdot m \cdot s/rad$	Koeficient viskózního tření 1
b2	6,2	$N \cdot m \cdot s/rad$	Koeficient viskózního tření 2

gf_1	0,2	–	Konstanta modelu tření
gf_2	90	–	Konstanta modelu tření
gf_3	11	–	Konstanta modelu tření
gf_4	0,12	–	Konstanta modelu tření
gf_5	110	–	Konstanta modelu tření
gf_6	0,05	–	Konstanta modelu tření
m1	2	kg	Hmotnost ramene 1
m2	5	kg	Hmotnost ramene 2
phi1_init	0	rad	Počáteční natočení ramene 1
phi2_init	0	rad	Počáteční natočení ramene 2

Tabulka 14. Vstupy, výstupy a parametry modelu Dvojité kyvadlo.

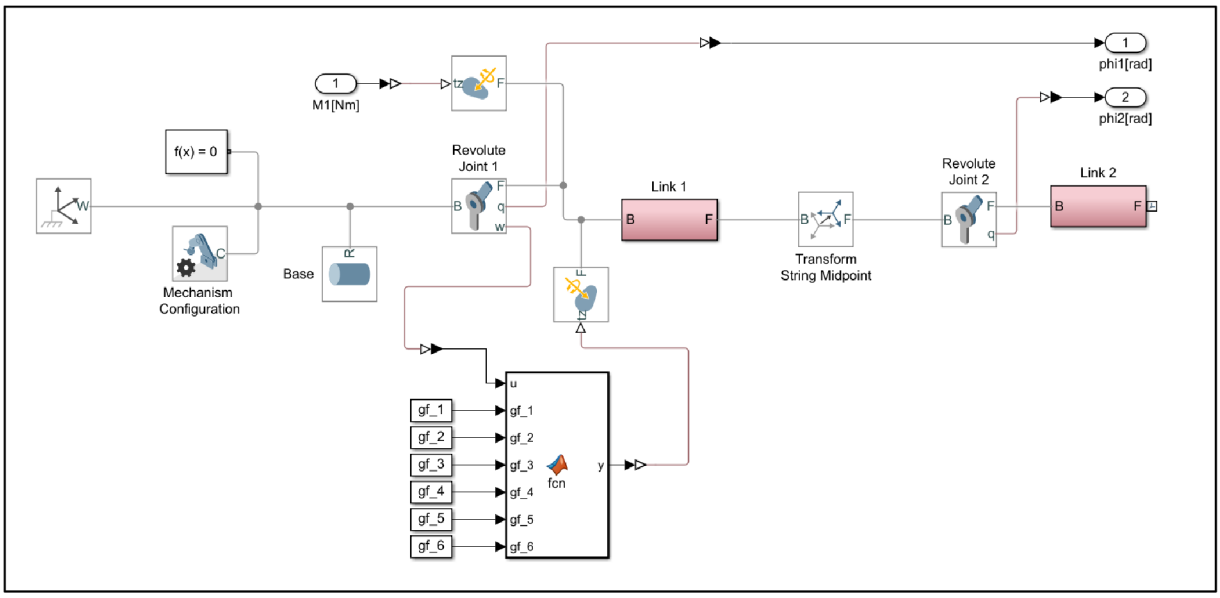
4.1.4 Rotační kyvadlo

Rotační kyvadlo (Obr. 22) je v mnoha ohledech podobné předchozímu modelu, ovšem s několika odchylkami. Model tření, popsaný v předchozí kapitole rovnicí (4.6), působí pouze v první rotační vazbě, ve druhé působí již jen viskózní tření. Také konfigurace ramen je modifikovaná, přičemž první rameno, hnané vstupním momentem, se pohybuje v horizontální rovině a druhé rameno se pohybuje v rovině kolmé na první rameno. Hmotnost ramen je zjednodušena na hmotný bod ve středu jeho délky.



Obr. 22. Schematický diagram modelu Rotační kyvadlo.

Taktéž model v Simulinku je velmi podobný předchozímu, jak je vidět na Obr. 23, stejně jako seznam parametrů (Tabulka 15).



Obr. 23. Implementace modelu Rotační kyvadlo v Simulinku.

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
M1	In 1	Nm	Moment hnacího motoru
phi1	Out 1	rad	Natočení ramene 1
phi2	Out 2	rad	Natočení ramene 2
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
L1	0,08	m	Délka ramene 1
L2	0,11	m	Délka ramene 2
b1	0	$N \cdot m \cdot s/rad$	Koeficient viskózního tření 1
b2	$2 \cdot 10^{-4}$	$N \cdot m \cdot s/rad$	Koeficient viskózního tření 2
gf_1	0,2	–	Konstanta modelu tření
gf_2	90	–	Konstanta modelu tření
gf_3	11	–	Konstanta modelu tření
gf_4	0,12	–	Konstanta modelu tření
gf_5	110	–	Konstanta modelu tření
gf_6	0,05	–	Konstanta modelu tření
m1	0,5	kg	Hmotnost ramene 1
m2	0,1	kg	Hmotnost ramene 2
phi1_init	0	rad	Počáteční natočení ramene 1
phi2_init	0	rad	Počáteční natočení ramene 2

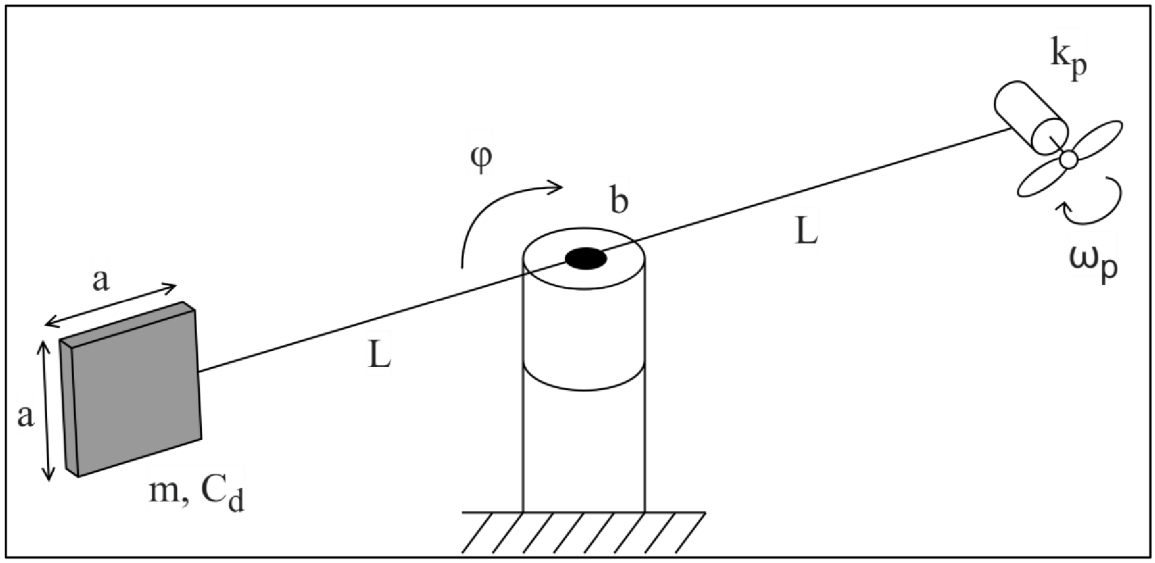
Tabulka 15. Vstupy, výstupy a parametry modelu Rotační kyvadlo.

4.1.5 Soustava s vrtulí (horizontální)

Soustava s vrtulí (Obr. 24) patří, co se týče složitosti mechanismu a počtu parametrů, mezi jednodušší modely. Je zde použit další druh buzení soustavy, kdy otáčky vrtule generují tahovou sílu, která roztáčí tyč, na jejímž druhém konci je umístěna hmotná deska. Hmotnost pohonu s vrtulí zanedbáváme, a jeho tah počítáme zjednodušeně jako součin konstanty tahu vrtule k_p a druhé mocniny otáček ω_p. Proti tahu motoru působí viskózní tření v rotační vazbě b a odpor vzduchu pohybem desky, který je vypočítán dle rovnice (4.7):

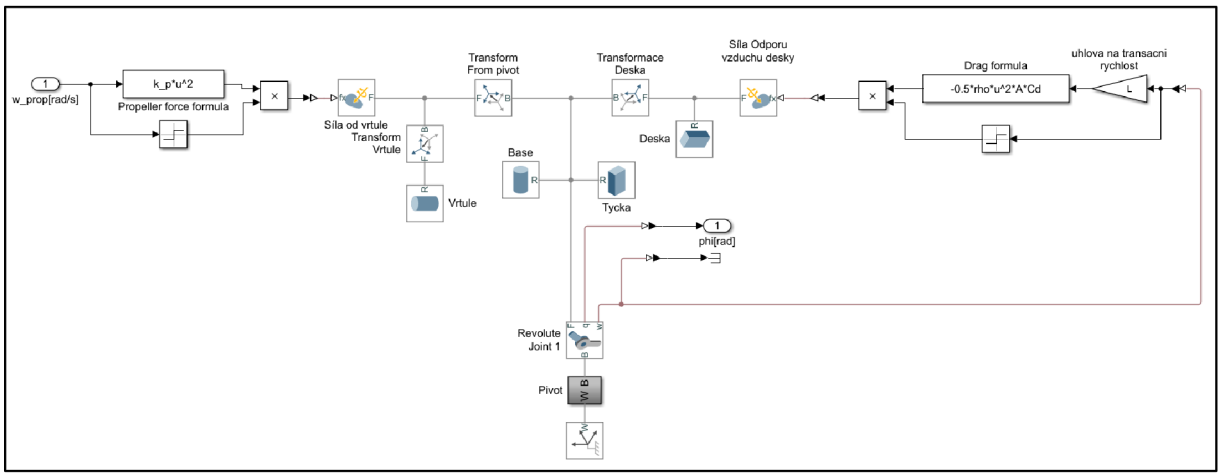
$$F_{odp} = \frac{1}{2}\rho v^2 A C_d \tag{4.7}$$

kde F_{odp} je odporová síla, ρ je hustota prostředí (vzduch), v je rychlost pohybu desky, A je plocha desky a C_d je součinitel odporu desky.



Obr. 24. Schematický diagram modelu Soustava s vrtulí (horizontální).

Model v Simulinku můžeme vidět níže na Obr. 25.



Obr. 25. Implementace modelu Soustava s vrtulí (horizontální) v Simulinku.

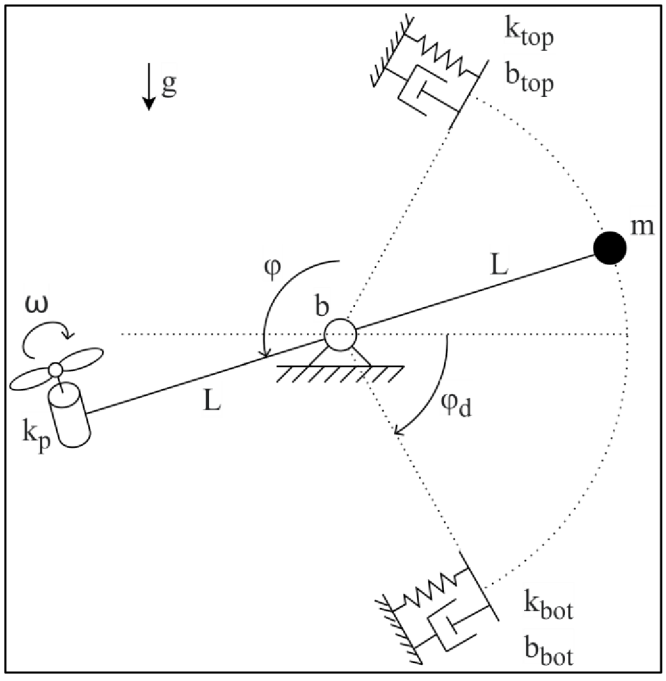
Co se týče jednotlivých parametrů modelu (Tabulka 16), je vhodné zmínit, že parametr délky strany desky a hraje roli pouze pro zadání rozměrů prvku v Simscape a pro výpočet parametru plochy desky A. Z výše uvedených vztahů lze také odtušit, že mnoho parametrů v tomto modelu nebude možné unikátně identifikovat, jelikož nejsou na základě naměřených dat separovatelné (např. plocha desky A a součinitel odporu C_d).

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
w_prop	In 1	rad/s	Otáčky vrtule
phi	Out 1	rad	Natočení ramene
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
a	0,4	m	Délka strany desky
A	0,16	m^2	Plocha čtvercové desky (a^2)
Cd	1,2	$-$	Součinitel odporu desky
L	1,2	m	Délka ramene
b	1,7	$N \cdot m \cdot s/rad$	Koeficient viskózního tření
k_p	$2{,}8 \cdot 10^{-3}$	$N/(rad/s)^2$	Konstanta tahu vrtule
m	2	kg	Hmotnost desky
rho	1,29	kg/m^3	Hustota prostředí (vzduchu)

Tabulka 16. Vstupy, výstupy a parametry modelu Soustava s vrtulí (horizontální).

4.1.6 Soustava s vrtulí (vertikální)

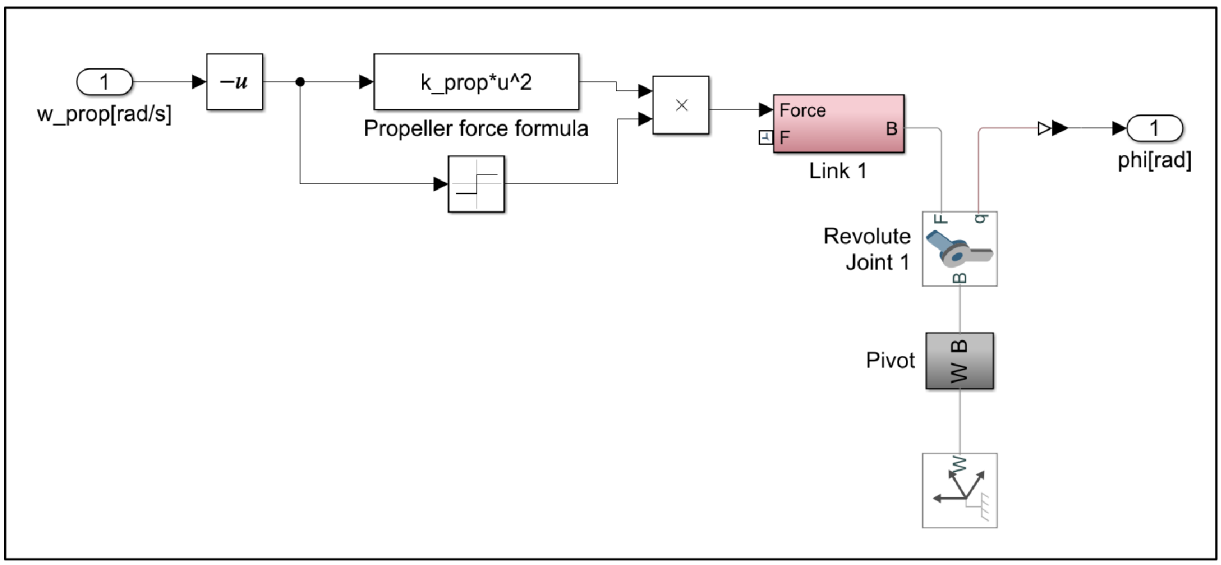
Model soustavy s vrtulí, pohybující se ve vertikální rovině, kombinuje několik prvků z předchozích modelů (kyvadlo, vrtulový pohon) a přidává navíc dva pružné dorazy s tlumením, které omezují pohyb soustavy na obou stranách (Obr. 26).



Obr. 26. Schematický diagram modelu Soustava s vrtulí (vertikální).

Z výše uvedeného diagramu je patrné, že vliv parametrů definující chování dorazů se projeví až ve chvíli, kdy na ně soustava narazí. Tato vlastnost je vhodná pro demonstraci nutnosti vhodné volby buzení modelu. Současně nabízí možnost rozdělit úlohu odhadu parametrů na několik dílčích částí, kdy mohou být zvlášť odhadovány parametry související s dynamickým chováním mezi dorazy a následně odhad parametrů horního, resp. dolního dorazu.

Model v Simulinku nepřináší žádné nové prvky (Obr. 27). Parametry dorazů jsou zadány jako limitní parametry přímo v blocích Simscape Multibody.



Obr. 27. Implementace modelu Soustava s vrtulí (vertikální) v Simulinku.

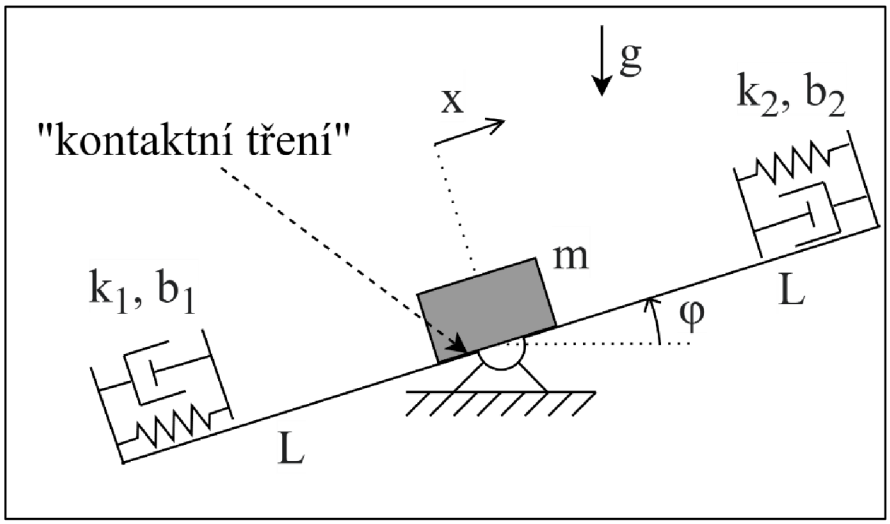
Seznam vstupů, výstupů a parametrů je uveden v tabulce níže (Tabulka 17).

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
w_prop	In 1	rad/s	Otáčky vrtule
phi	Out 1	rad	Natočení ramene
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
L	1,6948	m	Délka ramene
b_bot	5	$N \cdot m/(deg/s)$	Konstanta tlumení (spodní doraz)
b_top	10	$N \cdot m/(deg/s)$	Konstanta tlumení (horní doraz)
b_pivot	2,3	$N \cdot m/(rad/s)$	Koeficient viskózního tření (čep)
k_bot	$2 \cdot 10^4$	$N \cdot m/deg$	Tuhost pružiny (spodní doraz)
k_top	$1 \cdot 10^4$	$N \cdot m/deg$	Tuhost pružiny (horní doraz)
k_prop	$2{,}8 \cdot 10^{-3}$	$N/(rad/s)^2$	Konstanta tahu vrtule
m	2	kg	Hmotnost protizávaží
phi_lim	70,1942	deg	Úhel od nulové polohy k dorazům

Tabulka 17. Vstupy, výstupy a parametry modelu Soustava s vrtulí (vertikální).

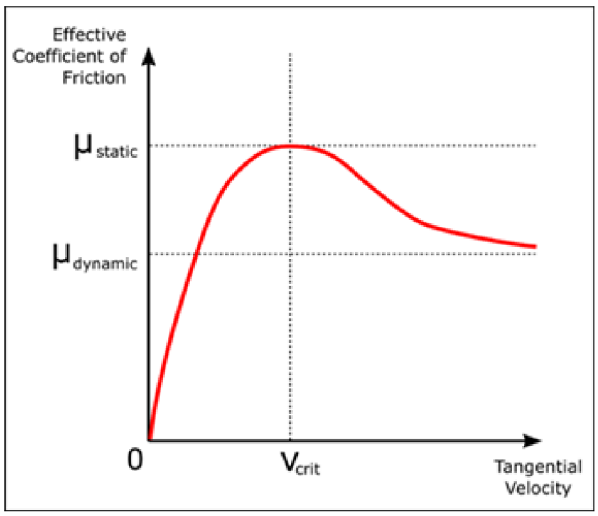
4.1.7 Těleso na naklápěcí plošině

Tento model tělesa na naklápěcí plošině (Obr. 28) je částečně podobný předchozím modelům, kromě upravené topologie však obsahuje několik důležitých modifikací. První z nich je model kontaktního tření mezi tělesem a naklápěcí plošinou, který je blíže popsán níže. Na rozdíl od většiny ostatních má tento model kinematické buzení, k čemuž je nutné přihlížet při volbě vstupu do modelu. V případě skokové změny polohy totiž může teoreticky dojít ke ztrátě kontaktu tělesa s plošinou. Z tohoto důvodu je vstup do modelu nejprve alespoň částečně vyhlazen.



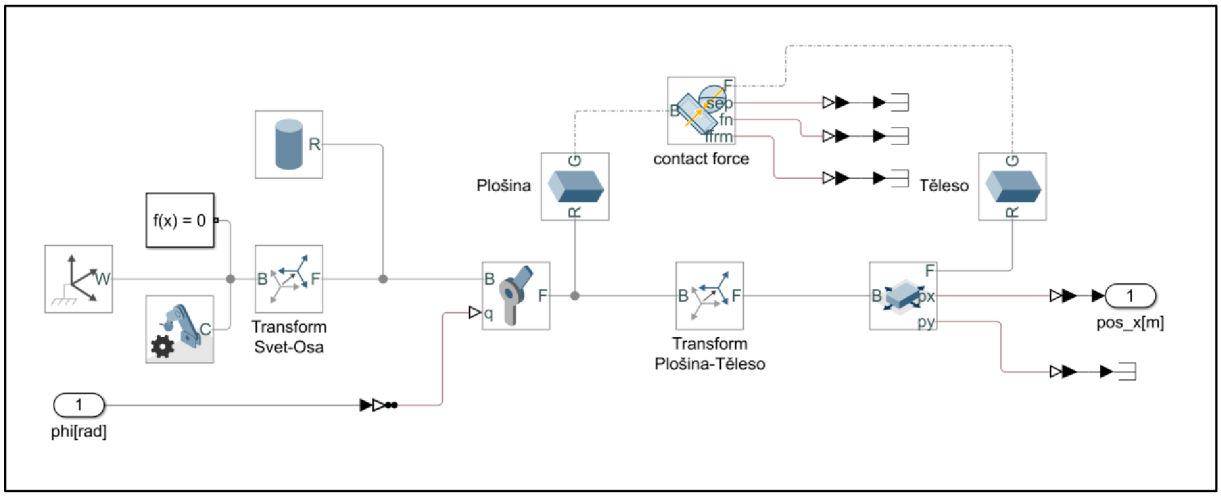
Obr. 28. Schematický diagram modelu Těleso na naklápěcí plošině.

Kontakt tělesa s plošinou je realizován pomocí bloku Spatial Contact Force [70] z knihovny Simscape Multibody. V rámci tohoto bloku je modelován malý vzájemný průnik tělesa a plošiny, na základě něhož jsou počítány normálové a třecí síly. Většina parametrů modelujících tento kontakt je ponechána ve výchozím nastavení, parametrizovány jsou pouze koeficienty statického a dynamického tření, jenž slouží k modelování chování třecí síly. Tento model tření je nazván „Smooth Stick-Slip“ a jeho charakteristiku je možné vidět na Obr. 29. Parametr kritické rychlosti v_{crit} byl ponechán na výchozí hodnotě ($1 \cdot 10^{-3}\ m/s$).



Obr. 29. Model kontaktního tření „Smooth Stick-Slip“ [70]

Model v Simulinku je uveden na Obr. 30. V aktuální podobě je vstup do modelu zadáván jako natočení plošiny a výstupem je poloha tělesa v podélném směru. V případě potřeby je však možné snímat i polohu ve směru kolmo k plošině, což může poskytnout informaci o ztrátě kontaktu tělesa s plošinou.



Obr. 30. Implementace modelu Těleso na naklápěcí plošině v Simulinku.

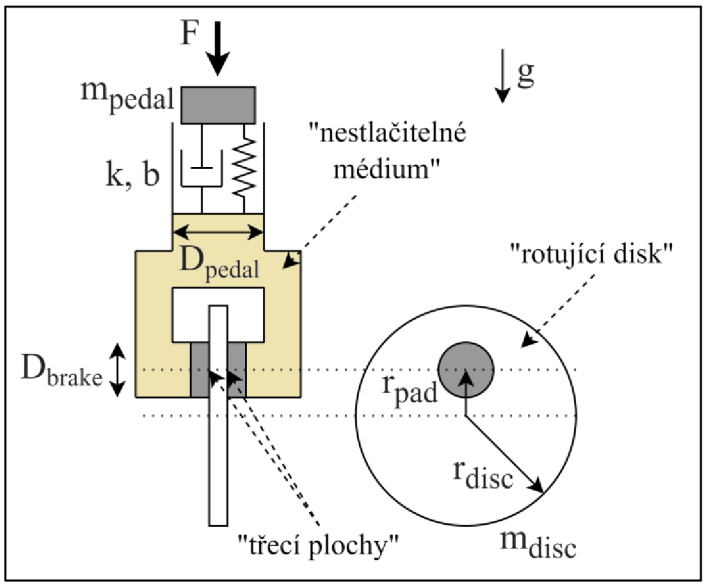
Seznam vstupů, výstupů a parametrů je opět uveden v tabulce níže (Tabulka 18).

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
phi	In 1	rad	Natočení naklápěcí plošiny
pos_x	Out 1	m	Poloha tělesa na naklápěcí plošině
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
L	1	m	Délka plošiny
b1	700	$N/(m/s)$	Konstanta tlumení dorazu 1
b2	600	$N/(m/s)$	Konstanta tlumení dorazu 2
f_dyn	0,35	$-$	Koeficient statického tření (kontaktní)
f_stat	0,42	$-$	Koeficient dynamického tření (kontaktní)
k1	$2 \cdot 10^5$	N/m	Tuhost pružiny dorazu 1
k2	$8 \cdot 10^5$	N/m	Tuhost pružiny dorazu 2
m	5	kg	Hmotnost tělesa

Tabulka 18. Vstupy, výstupy a parametry modelu Těleso na naklápěcí plošině.

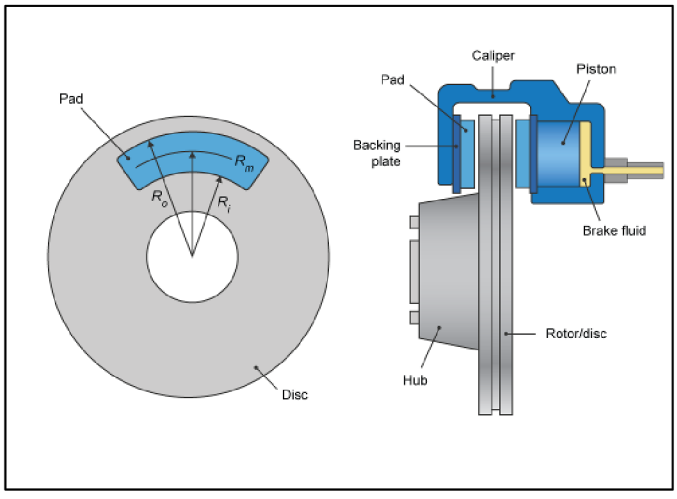
4.1.8 Kotoučová brzda

Kotoučová brzda je model, se vstupní silou, která je přenášena přes hmotné těleso s pružinou a tlumičem na nestačitelné médium. To pak dále působí na brzdné destičky, které brzdí rotující disk (Obr. 31). Disk má na počátku nenulovou úhlovou rychlost.

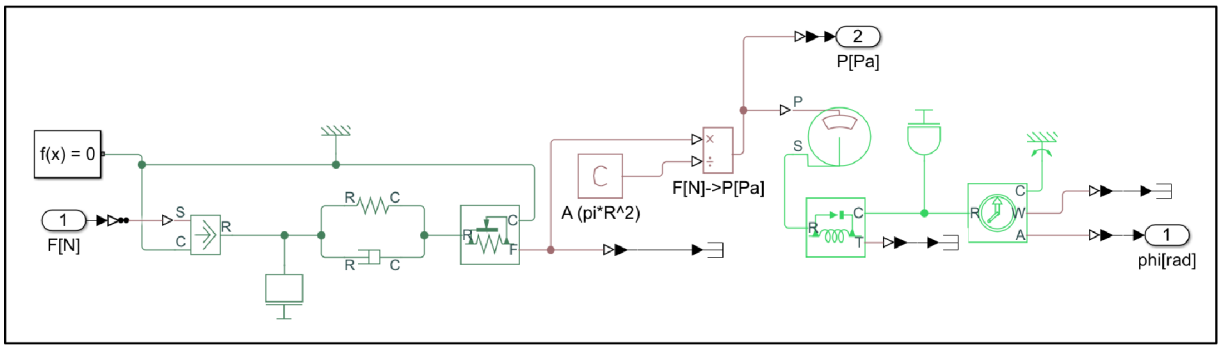


Obr. 31. Schematický diagram modelu Kotoučová brzda.

Model v Simulinku (Obr. 33) využívá bloků Simscape pro translační a rotační pohyb a také speciálního bloku Disc Brake [71], který modeluje chování kotoučové brzdy. Při přenosu síly na brzdné destičky se také uplatňuje různý průměr hydraulického vedení v místě působení sil (D_{pedal} a D_{brake}). Pro lepší ilustraci je kromě výše diagramu výše uvedeno i ilustrační schéma samotného bloku Disc Brake (Obr. 32). Vstupem do bloku je tlak působící na brzdné destičky, parametry jsou pak průměr kruhové plochy destiček D_{brake} na které působí nestlačitelné médium, střední vzdálenost destiček od osy rotace disku r_{pad}, počet brzdných destiček N_{pad} a dále koeficienty pro výpočet třecí síly fc_{static} a $fc_{kinetic}$, které podobně jako u předchozího modelu (4.1.7) určují velikost třecí síly v závislosti na úhlové rychlosti kotouče. V případě rychlosti blízké nule se použije koeficient statického tření, jinak koeficient kinetického (Coulombovského) tření. Hodnotu úhlové rychlosti pro přepnutí mezi těmito dvěma stavy určuje parametr fv_{break}. Na rozdíl od předchozího modelu tření („Smooth Stick-Slip“) není tento přechod hladký, ale okamžitý.



Obr. 32. Ilustrace modelu bloku Disc Brake z knihovny Simscape Mechanical [71].



Obr. 33. Implementace modelu Kotoučová brzda v Simulinku.

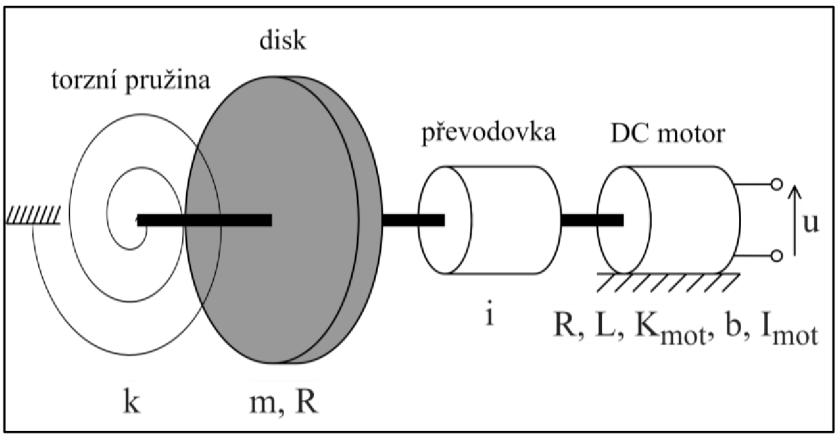
Kotoučová brzda opět patří do kategorie modelů, ve kterých se vyskytuje více vzájemně souvisejících, neseparovatelných parametrů, jako například D_{pedal}, D_{brake}, N_{pad} a r_{pad}. Kompletní seznam vstupů, výstupů a parametrů je opět v tabulce níže (Tabulka 19).

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
F	In 1	N	Síla působící na pedál
phi	Out 1	rad	Natočení rotujícího disku
P	Out 2	Pa	Tlak na brzdové kotouče
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
D_cyl_brake	10	mm	Průměr vedení kapaliny (destičky)
D_cyl_pedal	18	mm	Průměr vedení kapaliny (pedál)
I_disc	0,03	$kg \cdot m^2$	Moment setrvačnosti rotujícího disku
N_Pad	2	$-$	Počet brzdných destiček
r_pad	150	mm	Vzdálenost středu destiček od osy rotace disku
r_disc	0,2	m	Poloměr rotujícího disku
b_pedal	7	$N/(m/s)$	Konstanta tlumení pedálu
fc_kinetic	0,3	$-$	Koeficient Coulombovského tření (destičky)
fc_static	2,1	$-$	Koeficient statického tření (destičky)
fv_break	0,4	rad/s	Úhlová rychlost přechodu mezi statickým a Coulombovským třením
k_pedal	93	N/m	Tuhost pružiny pedálu
m_disc	1,5	kg	Hmotnost rotujícího disku
m_pedal	0,8	kg	Hmotnost pedálu
dphi0	10	rad/s	Počáteční úhlová rychlost disku

Tabulka 19. Vstupy, výstupy a parametry modelu Kotoučová brzda.

4.1.9 DC motor + setrvačník na pružině

Poslední dvojice modelů je podobná v tom, že oba obsahují model stejnosměrného motoru s převodovkou, u kterého je navíc možné ovládat připojení a odpojení zdroje napětí. U této první varianty modelu pak zátěž tvoří hmotný disk s torzní pružinou (viz Obr. 34).



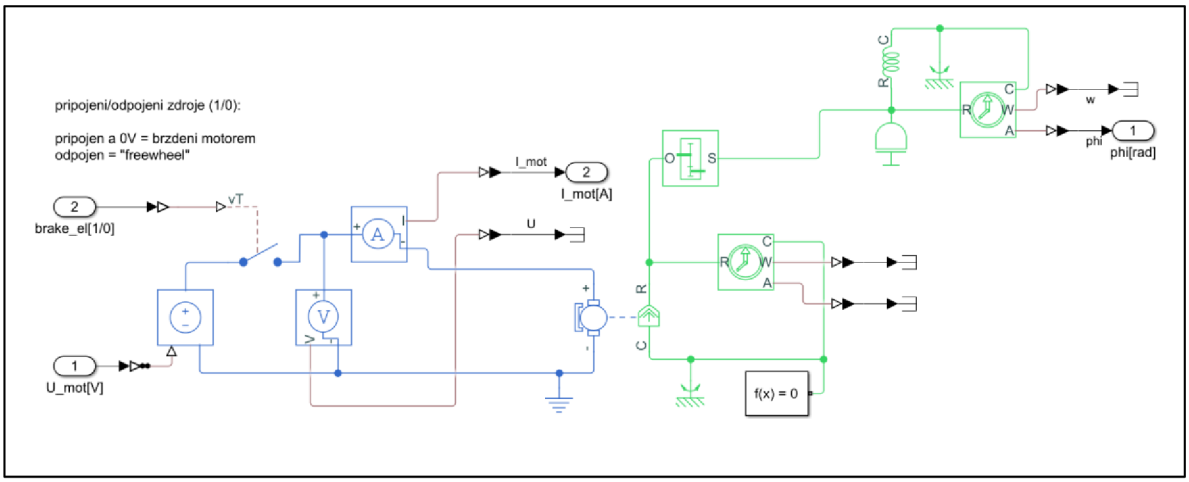
Obr. 34. Schematický diagram modelu DC motor + setrvačník na pružině.

Na rozdíl od modelu jednoduchého DC motoru (4.1.2) je v tomto případě využito pro propojení elektrické a mechanické části odlišného bloku, konkrétně DC Motor [72], který má větší možnost nastavení a parametrizace, než dříve použitý blok obecného rotačního elektromechanického převodníku. Mimo jiné v sobě rovnou zahrnuje parametry momentu setrvačnosti rotoru a viskózního tření.

Zmíněná možnost odpojení zdroje napětí, která je realizována pomocí samostatného logického vstupu, umožňuje uživateli volbu mezi dvěma módy, které hrají roli zejména při snižování otáček:

- Napětí je připojeno a nastaveno na 0V = motor aktivně brzdí.
- Napětí je odpojeno = tzv. volnoběžný režim (*freewheel*), kdy na soustavu působí jen mechanické síly (tření, pružina).

Všechny další parametry a bloky použité v modelu v Simulinku (Obr. 35) jsou obdobné jako v předchozích modelech a jsou uvedeny spolu se vstupy a výstupy v tabulce níže (Tabulka 20).



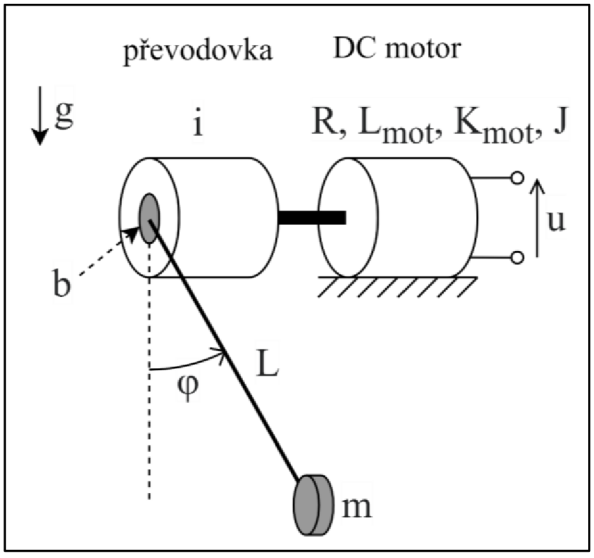
Obr. 35. Implementace modelu DC motor + setrvačník na pružině v Simulinku.

Veličina	Vstup/Výstup	Jednotka	Význam
U_mot	In 1	V	Napětí na svorkách vinutí DC motoru
brake_el	In 2	$-$	Spínač připojení napětí na DC motor
phi	Out 1	rad	Natočení rotujícího disku
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
Bemf	$2 \cdot 10^{-3}$	V/rpm	Konstanta motoru K_{mot}
I_disc	0,03	$kg \cdot m^2$	Moment setrvačnosti disku ($\frac{1}{2}mR^2$)
I_mot	$8 \cdot 10^{-3}$	$g \cdot cm^2$	Moment setrvačnost rotoru motoru
L	$6 \cdot 10^{-6}$	H	Indukčnost vinutí motoru
R	17,1	Ω	Odpor vinutí motoru
b	$1{,}5 \cdot 10^{-5}$	$N \cdot m/(rad/s)$	Koeficient viskózního tření v motoru
i	$2{,}5 \cdot 10^{-2}$	$-$	Převodový poměr převodovky
k_spring	0,2	$N \cdot m/rad$	Tuhost pružiny

Tabulka 20. Vstupy, výstupy a parametry modelu DC motor + setrvačník na pružině.

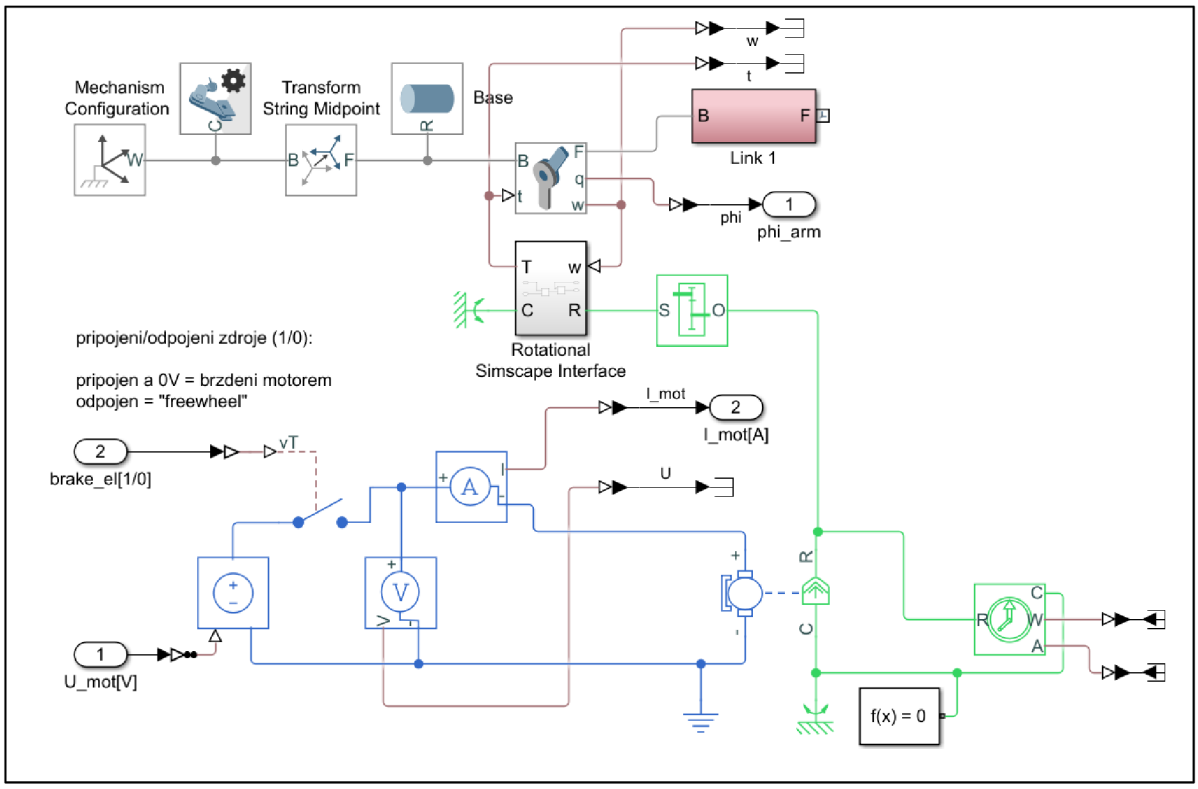
4.1.10 DC motor + závaží na rameni

Model DC motoru se závažím na rameni je alternativou k předchozímu modelu (4.1.9), kdy je zátěž tvořena závažím (hmotný bod) na rameni (Obr. 36). Elektrická část je stejná jako u přechozího modelu, včetně možnosti odpojit zdroj napětí motoru pomocí vstupního signálu.



Obr. 36. Schematický diagram modelu DC motor + závaží na rameni.

Vzhledem k tomu, že velká část mechanických komponent je vytvořena s pomocí knihovny Simscape Multibody, která není se zbytkem knihovny Simscape přímo kompatibilní, bylo nutné použít subsystém rozhraní, Rotational Simscape Interface, který předává hodnoty momentu a úhlové rychlosti mezi těmito dvěma doménami (viz Obr. 37).



Obr. 37. Implementace modelu DC motor + závaží na rameni v Simulinku.

Parametry a bloky použité v modelu jsou opět obdobné jako v předchozích modelech a jsou uvedeny spolu se vstupy a výstupy v tabulce níže (Tabulka 21).

Veličina	**Vstup/Výstup**	**Jednotka**	**Význam**
U_mot	In 1	V	Napětí na svorkách vinutí DC motoru
brake_el	In 2	–	Spínač připojení napětí na DC motor
phi	Out 1	rad	Natočení ramene se závažím
I_mot	Out 2	A	Proud ve vinutí DC motoru
Parametr (název proměnné)	**Hodnota**	**Jednotka**	**Význam**
Bemf	$2 \cdot 10^{-3}$	V/rpm	Konstanta motoru K_{mot}
Jm	$1 \cdot 10^{-2}$	$g \cdot cm^2$	Moment setrvačnost rotoru motoru
L	$1{,}2 \cdot 10^{-5}$	H	Indukčnost vinutí motoru
L1	0,12	m	Délka ramene se závažím
R	6,9	Ω	Odpor vinutí motoru
b1	$5 \cdot 10^{-4}$	$N \cdot m/(rad/s)$	Koeficient viskózního tření v čepu ramene se závažím
i	$4 \cdot 10^{-2}$	–	Převodový poměr převodovky
m1	0,5	kg	Hmotnost závaží na rameni

Tabulka 21. Vstupy, výstupy a parametry modelu DC motor + závaží na rameni.

4.2 Volba vstupů do modelů

Poté co byly vytvořeny jednotlivé modely a zvoleny jejich parametry, bylo třeba vytvořit signály vstupů do modelů a to tak, abychom pro účely dalšího testování algoritmů optimalizačních modelů byli schopni tyto systémy vhodně vybudit. Vhodným vybuzením máme namysli takové vstupy, aby se v reakci systému na ně projevily co nejvíce všechny vlastnosti a dynamika systému, což je samozřejmě propojeno s hodnotami parametrů modelu. Tato problematika již také byla nastíněna v předcházející kapitole 2.1.2.

Vzhledem k tomu, že v rámci optimalizační úlohy probíhá veškeré nastavení a ovládání modelu z MATLABu, je vhodné mít funkce, pomocí kterých dokážeme vygenerovat vstupní signály. Bohužel v současnosti neexistují hotové funkce, které by generovaly signály přesně dle našich potřeb, byla proto vytvořena vlastní sada těchto funkcí.

Základním požadavkem bylo generování základních tvarů signálů s pomocí několika ovládacích parametrů, přičemž u diskrétních signálů stačí generovat jen body v klíčových uzlech, zatímco u spojitých signálů je požadavek na možnost definice vzorkovací periody.

Níže je uveden seznam vytvořených funkcí a jejich parametrů. Výstup všech funkcí je proměnná `sig`, kde první sloupec matice jsou údaje o čase a druhý sloupec jsou hodnoty signálu v daném čase. Společné jsou také parametry `FinalTime`, což je celkové trvání signálu (v sekundách) a `SampleTime`, což je vzorkovací perioda signálu (v sekundách), přičemž pokud je u diskrétních signálů `SampleTime` nastaven na hodnotu `0`, jsou generovány pouze klíčové body signálu (začátek, konec, změna hodnoty).

Nulový signál

```
sig = SigGen_zero(FinalTime,SampleTime)
```

Všechny proměnné a parametry této funkce jsou vysvětleny na začátku této kapitoly.

Konstantní nenulový signál

```
sig = SigGen_const(Value,FinalTime,SampleTime)
```

Parametr `Value` představuje hodnotu konstantního signálu.

Skokový signál (step)

```
sig = SigGen_step(InitVal,FinalVal,StepTime,FinalTime,
SampleTime)
```

Parametr `InitVal` je počáteční hodnota signálu, `FinalVal` je finální hodnota signálu a `StepTime` je čas skokové změny signálu.

Tento typ signálu je vhodný na rychlé a efektivní vybuzení systému a často se používá v analýze chování modelů (tzv. odezva na skok).

Pulz

```
sig = SigGen_pulseSingle(Amplitude,StepTime,PulseWidth,
FinalTime,SampleTime)
```

Parametr `Amplitude` je amplituda pulzu z nulové hodnoty, `StepTime` je čas začátku pulzu a `PulseWidth` je doba trvání pulzu (v sekundách).

Pulz má podobné vlastnosti jako skokový signál, ale je vhodnější pro situace, kde je žádoucí vybudit systém velmi silným ale krátkým (časově omezeným) impulzem.

Náhodné schody

```
sig = SigGen_randSteps(AmplitudeMin,AmplitudeMax,StepWidth,
Delay,RngSeed,FinalTime,SampleTime)
```

Parametr `AmplitudeMin` je nejmenší povolená hodnota signálu, `AmplitudeMax` je největší povolená hodnota signálu, `StepWidth` je délka jednoho konstantního segmentu (v sekundách), `Delay` doba na začátku signálu, po kterou bude signál nulový, než se začne generovat první „schod“ a `RngSeed` je parametr pro inicializaci generátoru pseudonáhodných čísel.

Tento typ signálu je možné použít pro situace kdy chceme systém budit signálem skokového charakteru, ale s různou amplitudou, aby bylo možné pozorovat jeho chování pro různé velké změny vstupu.

Sinusovka

```
sig = SigGen_sine(Amplitude,Bias,Freq,Phase,FinalTime,
SampleTime)
```

Parametr `Amplitude` je amplituda sinusového signálu, `Bias` konstanta přičtená k sinusovému signálu, `Freq` je frekvence sinusovky (v rad/s) a `Phase` je fázový posun signálu (v rad).

Sinusovka je také velmi rozšířeným druhem buzení, jehož hodnota se na rozdíl od přechozích signálů mění plynule, což může být pro některé typy modelů výhodnější.

Chirp signál (čerp)[7]

```
sig = SigGen_chirp(Amplitude,f_Start,f_Final,FinalTime,
SampleTime)
```

Parametr `Amplitude` je amplituda sinusového signálu, `f_Start` je počáteční frekvence sinusovky a `f_Final` je frekvence na konci signálu.

Tento signál je modifikací sinusovky, jehož výhodou je možnost otestovat reakci systému na různé frekvence buzení.

[7] **Čerp** (z anglického chirp) je sinusový signál jehož frekvence roste lineárně s časem.

Jak již bylo naznačeno výše, vygenerované signály nemusí mít vždy jednotnou vzorkovací periodu. V případě modelu s více vstupy, je pak vhodné jednotit tyto signály tak, aby měly společnou vzorkovací periodu. Pro tento účel byla vytvořena níže uvedená funkce

```
sig_merged = SigGen_MERGE(InputSignalsCellArray)
```

kde `sig_merged` je výstupní matice sloučených signálů, přičemž první sloupec matice jsou údaje o čase a následující sloupce jsou hodnoty jednotlivých signálů v daném čase. Vstupem je proměnná `InputSignalsCellArray`, vytvořená jako:

```
InputSignalsCellArray = {input1,input2,input3,...}
```

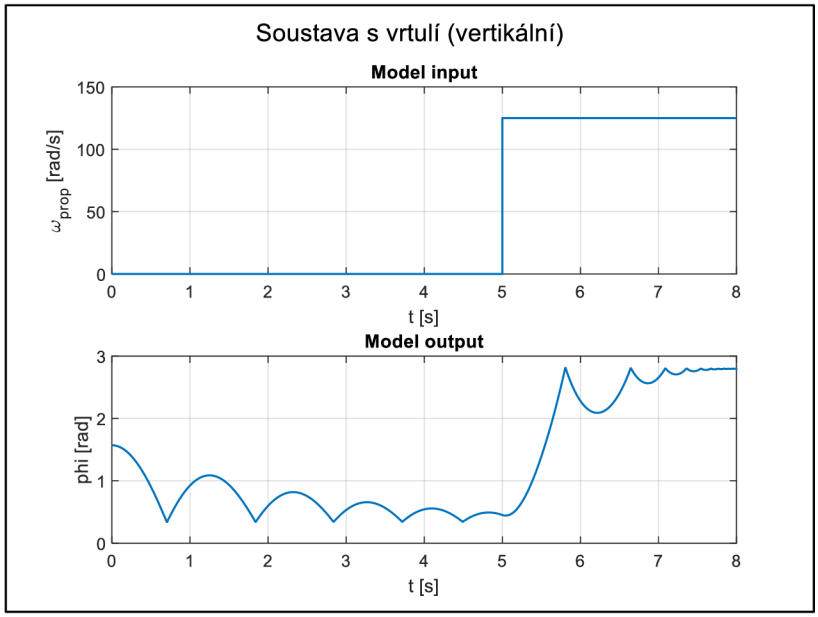
kde `inputN` jsou sloupcové matice jednotlivých signálů, podobně jako u funkcí popsaných výše (první sloupec čas, druhý sloupec hodnota signálu.

Vygenerování signálů pro konkrétní modely

Pro každý z modelů, byla za pomoci funkcí uvedených výše vytvořena sada vstupů skládající se vždy z těchto signálů:

1. Skokový signál
2. Sinusovka
3. Pulz
4. Náhodné schody
5. Chirp

Vstupy byly vždy vytvořeny tak, aby adekvátně vybudily model – dostatečně silné, ale ne tak silné, aby se model hned dostal do limitního či extrémního stavu. Příklad vstupního signálu pro model Soustava s vrtulí (vertikální) je uvede níže (Obr. 38). Buzení bylo zvoleno tak, aby se model dostal do obou limitních poloh a současně byla poskytnuta dostatečně dlouhá doba v obou polohách, aby se mohla projevit dynamika dorazů. Všechny vytvořené signály pak byly uloženy pro další použití v podobě .mat souboru pro každý model.



Obr. 38. Příklad volby buzení a odezva systému pro model Soustava s vrtulí (vertikální).

4.3 Vizualizace prostoru parametrů pro vybrané případy

Díky tomu, že v tuto chvíli máme k dispozici modely, jejich parametry i vstupní signály, můžeme provést vizualizaci hodnoticí funkce optimalizační úlohy v prostoru parametrů. Připomeňme, že hodnoticí funkcí je v našem případě kvantifikace rozdílu mezi odezvou reálného (původního) systému a odezvou modelu s hledanými parametry.

Vhledem k tomu, že nemáme k dispozici reálné modely, je tato úloha zjednodušena tak, že jako odezvu reálného systému bereme výstup modelu při použití původních parametrů (viz tabulky u jednotlivých modelů v kap. 4.1). Vychýlením těchto původních parametrů pak dochází i k změně chování modelu a jeho výstupu a tím i ke zvýšení odchylky mezi výstupy těchto dvou variant daného modelu.

Jak již bylo zmíněno v kapitole 2.1.3, existuje několik různých metod pro výpočet hodnoticí funkce. V rámci této práce byla zvolena jedna z nejrozšířenějších metod v této oblasti – RMSE.

Důvodem pro vytvoření vizualizace závislosti RMSE na vychýlených hodnotách parametrů je získání alespoň orientační představy o tvaru prostoru, ve kterém se pohybuje optimalizační algoritmus. Ten má zpravidla výchozí bod v počátečním odhadu parametrů zadaným uživatelem, který je více či méně vzdálen od skutečných hodnot.

Aby byla pro člověka snadno čitelná, můžeme tuto vizualizaci efektivně zobrazit v maximálně 3D grafu, což znamená závislost RMSE na dvou parametrech. Vzhledem k celkovému počtu modelů a jejich parametrů není vhodné uvádět všechny dostupné kombinace.

Pro každý z 10 modelů byly zvoleny 3 parametry a jeden vstup (skokový signál). Pro všechny tři kombinace zvolených parametrů pak byl vygenerován prostor v oblasti kolem jejich skutečné hodnoty a pro každý bod byla provedena simulace a vyhodnocení RMSE. Ostatní parametry byly konstantní.

Pro každý parametr bylo vygenerováno 200 bodů v okolí jeho původní hodnoty. Hranice tohoto okolí byly stanoveny jako:

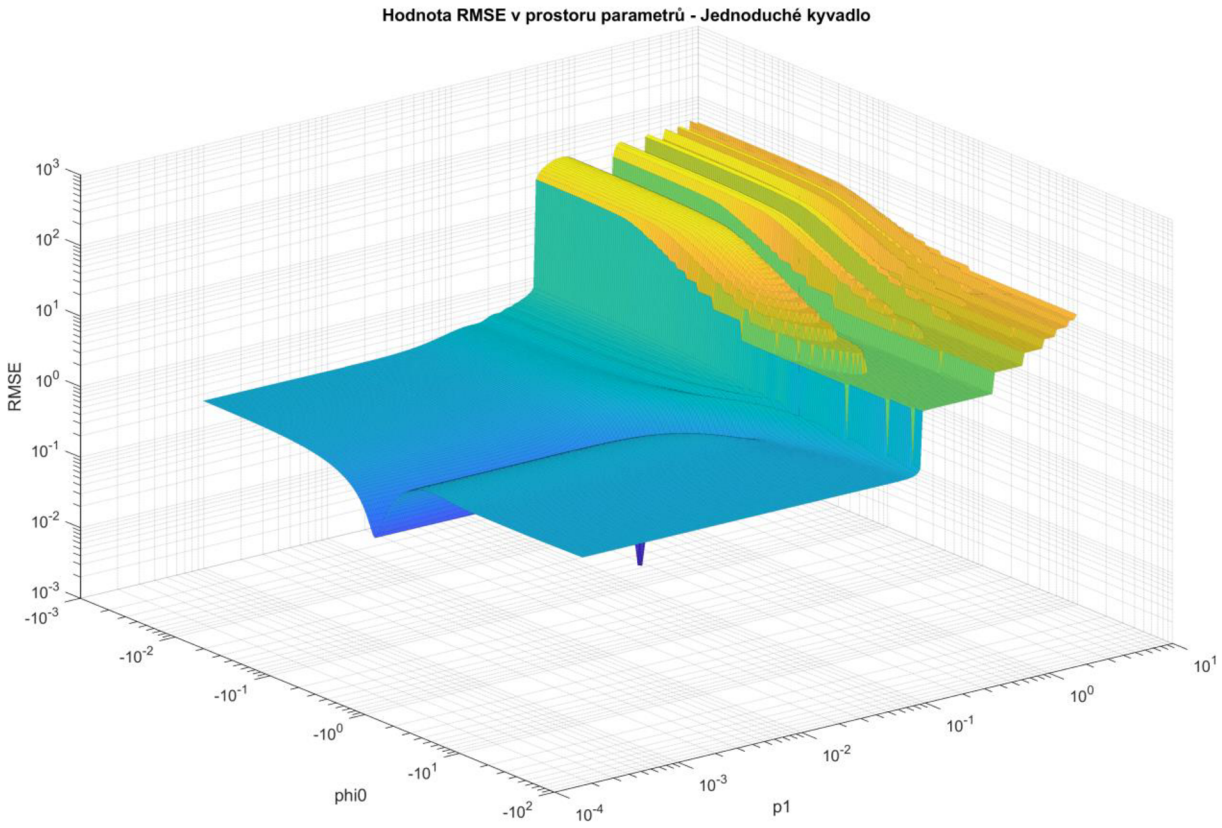
$$x_{top} = x_{true} \cdot 10^{2} \tag{4.8}$$

$$x_{bot} = x_{true} \cdot 10^{-2} \tag{4.9}$$

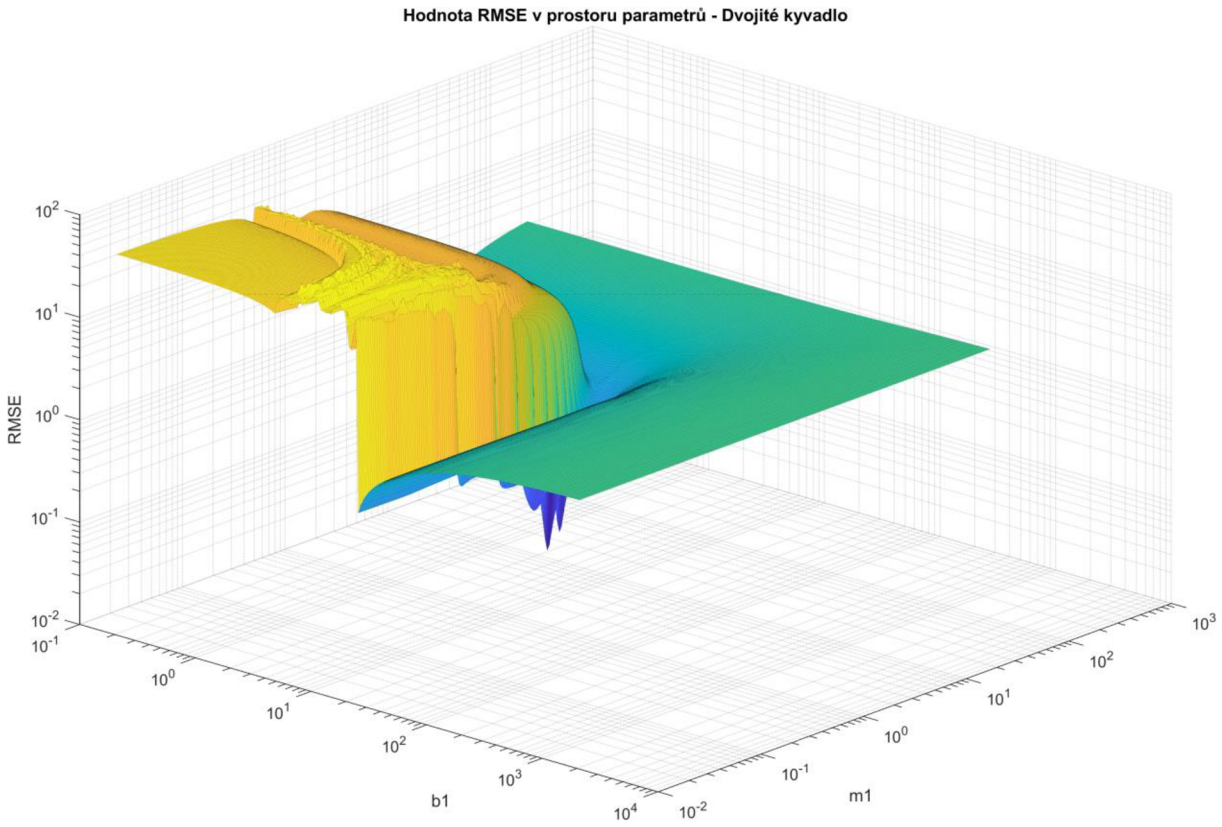
kde x_{true} je původní hodnota parametru, x_{top} je horní limit a x_{bot} je dolní limit. Rozložení vygenerovaných bodů mezi těmito hranicemi bylo logaritmické. Volba logaritmického měřítka byla zejména z důvodu lepší vizualizace celého mapovaného prostoru.

Celkem tedy bylo pro každou dvojici parametrů daného modelu provedeno 200^2 simulací, pro něž byla vyhodnocena hodnota RMSE. Ze všech výsledných 30 grafů závislosti RMSE na vychýlených parametrech pak bylo vybráno několik reprezentativních vzorků, které jsou uvedeny níže (Obr. 39 až Obr. 42).

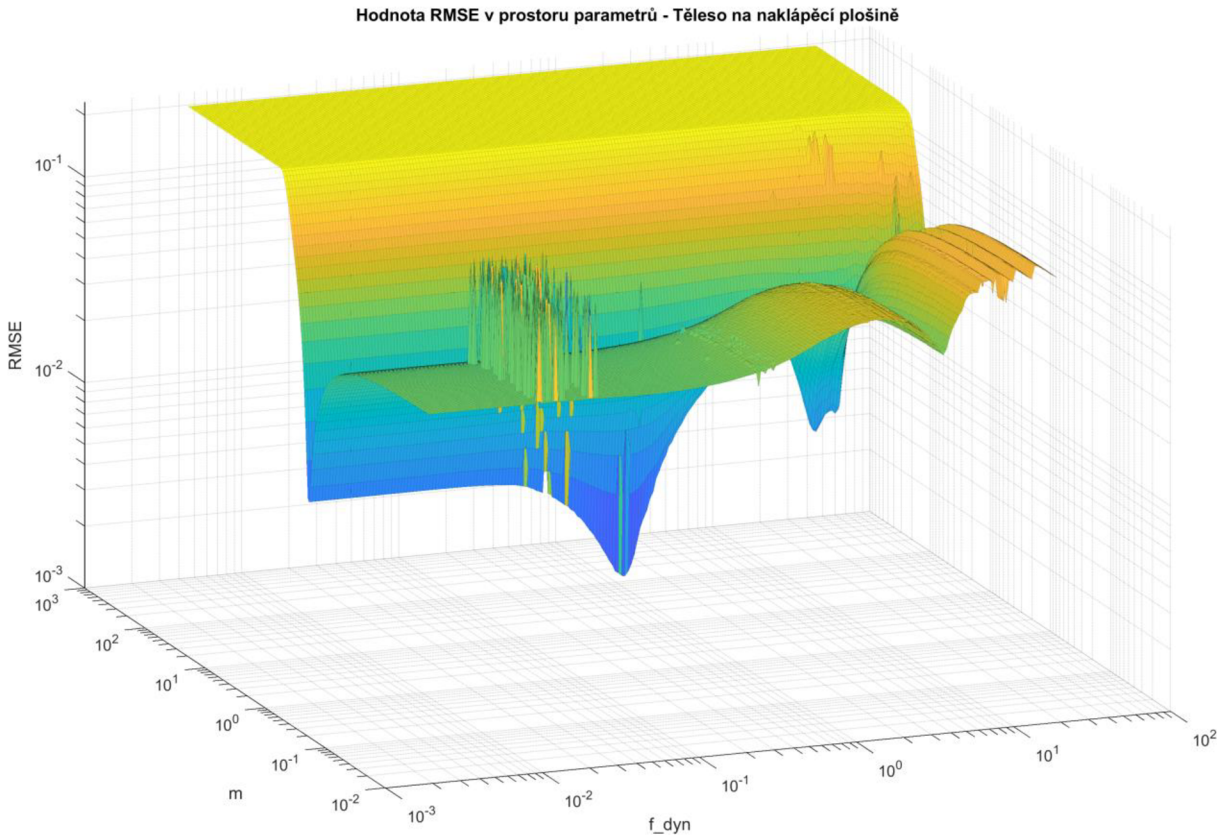
Jak je možné vidět na uvedených grafech, i pro relativně nízké dimenze prohledávaného prostoru se v systémech často vyskytuje mnoho lokálních minim, která někdy bývají i relativně „hluboká“, což může představovat problém zejména pro lokální řešiče. V jiných oblastech se pak hodnota RMSE mění jen minimálně, což může také v některých případech působit optimalizačnímu algoritmu problémy (např. volba příliš velkých kroků gradientních metod).



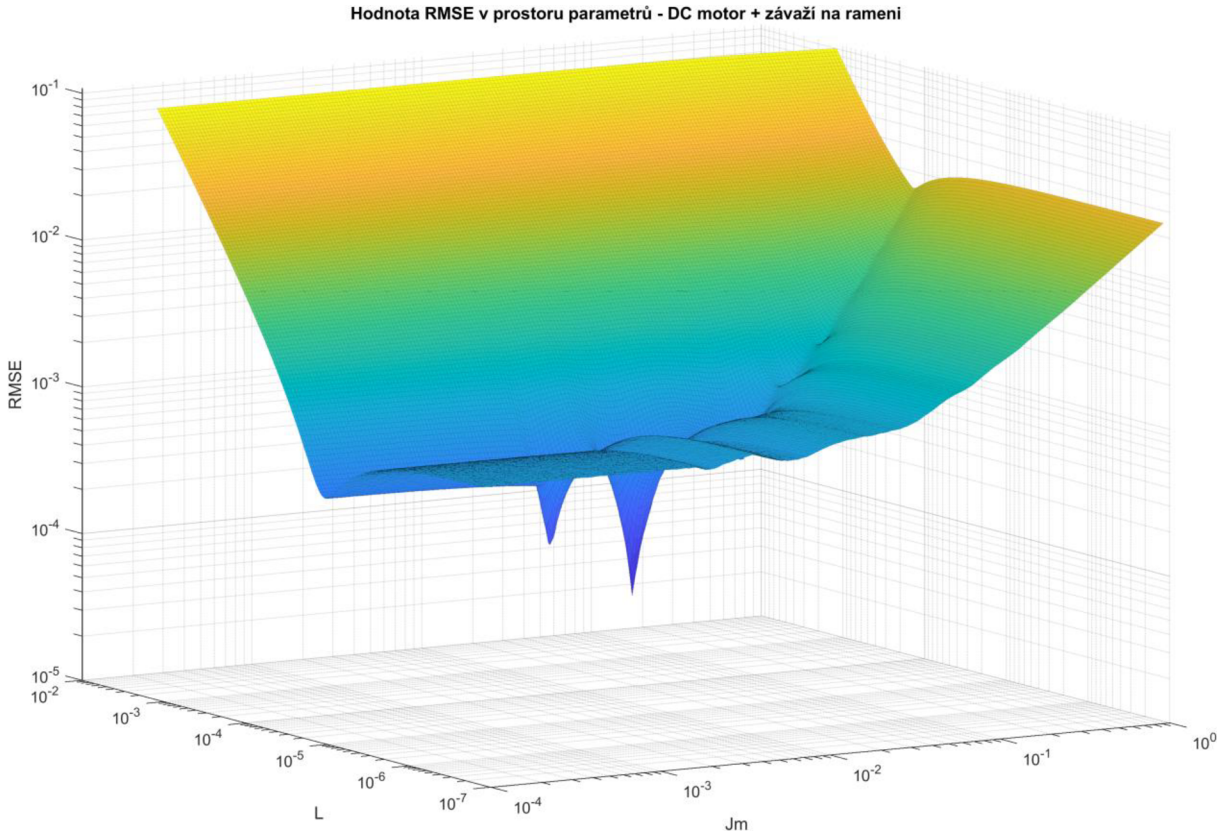
Obr. 39. Graf závislosti RMSE na vychýlených parametrech (Jednoduché kyvadlo).



Obr. 40. Graf závislosti RMSE na vychýlených parametrech (Dvojité kyvadlo).



Obr. 41. Graf závislosti RMSE na vychýlených parametrech (Těleso na naklápěcí plošině).



Obr. 42. Graf závislosti RMSE na vychýlených parametrech (DC motor + závaží na rameni).

4.4 Zhodnocení chování modelů a poznámky k nim

V průběhu kapitoly 4 (Modely mechatronických systémů), jsme popsali sadu modelů, které byly vytvořeny pro účely vývoje a testování optimalizačních algoritmů pro specifický typ úlohy – mechatronické systémy vytvořené za pomoci nástroje pro fyzikální modelování (Simscape).

Nejdůležitější výhody a pozorované vlastnosti vytvořené sady můžeme shrnout do bodů níže:

- Sada pokrývá a v různých variantách kombinuje prvky z oblasti mechaniky a jednoduché elektropohony, díky čemuž je vytvořen reprezentativní a konzistentní vzorek mechatronických systémů.
- Systémy obsahují několik typických zdrojů nelinearit v chování mechatronických systémů jako například pružné dorazy nebo několik variant modelů tření.
- U některých modelů jsou záměrně voleny parametry, které nelze samostatně identifikovat jen za základě naměřených vstupních a výstupních dat. Tyto parametry označujeme jako neseparovatelné. Tato vlastnost je výhodná pro edukativní účely a také pro testování algoritmů detekce těchto parametrů.
- Byly vytvořeny funkce pro generování typických průběhů budicích signálů a pro všechny modely bylo vytvořeno a otestováno několik vhodných variant buzení.
- Model jednoduchého kyvadla (4.1.1) se díky jednoduchosti počítá řádově rychleji. Je vhodný na počáteční testování vyvíjených algoritmů.
- Na základě testů v kapitole 4.3 je možné říci, že i pro dva parametry jsou prostory prohledávané optimalizačním algoritmem dostatečně netriviální a komplikované, minimálně pro lokální řešiče. S rostoucím počtem parametrů bude obtížnost ještě narůstat. Je tedy nutné se zaměřit na globální optimalizační algoritmy.
- Vytvořenou sadu modelů lze dále využívat pro i další účely, jako například vývoj a testování mimo oblast této dizertační práce nebo výuku problematiky odhadu parametrů.

5 Optimalizační algoritmy a nová metoda

5.1 Testování dostupných algoritmů

Jak bylo zmíněno v cílech dizertační práce (kap. 3), jádrem této práce je snaha o vytvoření nové metody pro robustní a co nejvíce automatizovaný odhad neznámých parametrů.

Vzhledem k tomu, že v oblasti optimalizace již bylo vytvořeno mnoho různých druhů algoritmů a jejich variant, nemá smysl se zabývat tvorbou zcela nového optimalizačního algoritmu od základů. Jako výhodnější se jeví otestování vlastností a chování již dostupných metod (kap. 2.2) pro námi definovanou úlohu, tedy odhad parametrů sady mechatronických modelů (kap. 4) na základě srovnání výstupných dat.

V následujících podkapitolách tedy provedeme nejprve návrh experimentu pro otestování vybraných optimalizačních algoritmů, následně odprezentujeme výsledky testů, a nakonec provedeme vyhodnocení těchto výsledků a jejich implikace pro návrh nového optimalizačního algoritmu.

5.1.1 Návrh srovnávacího testu

Jak bylo zmíněno v přechozí kapitole, před návrhem nového optimalizačního algoritmu je třeba prozkoumat chování a úspěšnost již dostupných algoritmů. Vzhledem k tomu, že modely i pomocné funkce byly dosud vytvářeny v programu MATLAB, zaměříme se pouze metody, které jsou v tomto nástroji dostupné (viz 2.2). Cílem je vytvořit srovnávací test (benchmark), který vyhodnotí výsledky vybraných optimalizačních algoritmů na vytvořené sadě modelů při různém nastavení experimentu.

Čistý výpočetní čas jedné simulace modelu se v průměru pohybuje v řádu desetin sekundy, maximálně jedné sekundy, avšak díky tomu, že optimalizační algoritmus zpravidla provádí minimálně stovky, případně i tisíce či desetitisíce simulací, než nalezne řešení, je výpočetní čas jednoho experimentu nezanedbatelný. V závislosti na počtu neznámých parametrů a nastavení algoritmu, trvá výpočet jedné úlohy minuty, někdy však i hodiny. Z tohoto důvodu bylo nutné učinit určité kompromisy ohledně počtu testovaných variant nastavení experimentu i samotných optimalizačních metod.

Výsledkem byl návrh srovnávacího testu, který je bodově popsán níže. Podrobný popis jednotlivých položek je pak uveden dále (Tabulka 22).

- 10 modelů systémů (viz 4.1).
- 11 optimalizačních algoritmů (viz 2.2).
- 2 varianty nastavení optimalizačního algoritmu.
- 7 variant kombinací hledaných parametrů.
- 2 druhy vstupních signálů do modelu.
- 4 varianty vychýlení počátečního odhadu hledaných parametrů od skutečné hodnoty.
- 2 varianty nastavení hranic prohledávaného prostoru parametrů.

Kombinací všech těchto variant dostáváme celkový počet 24640 optimalizačních úloh, jejichž výsledky budeme následně vyhodnocovat.

Parametr	**Varianty**
Model systému	viz 4.1
Optimalizační algoritmus	viz 2.2
Nastavení optimalizačního algoritmu	Viz příloha (Obr. 63 až Obr. 73)
Hledané parametry	Zvoleny 4 konkrétní parametry: a, b, c, d (ilustrační označení) Testováno 7 variant hledaných parametrů: [a], [b], [a, b], [a, c], [a, b, c], [a, b, d], [a, b, c, d]. Konkrétní parametry pro jednotlivé modely: • Jednoduché kyvadlo: dphi0, p1, p2, phi0 • DC motor: I, K_mot, L, b • Dvojité kyvadlo: b1, b2, gf_1, gf_2 • Rotační kyvadlo: L1, L2, gf_3, m1 • Soustava s vrtulí (horizontální): Cd, b, k_p, m • Soustava s vrtulí (vertikální): b_pivot, b_top, k_prop, k_top • Těleso na naklápěcí plošině: L, f_dyn, f_stat, m • Kotoučová brzda: I_disc, b_pedal, fc_kinetic, fv_break • DC motor + setrvačník na pružině: Bemf, I_disc, I_mot, L • DC motor + závaží na rameni: R, b1, i, m1
Buzení systému	Dva typy budicího signálu: • Skokový signál • Sinusový signál
Vychýlení počátečního odhadu	Čtyři typy vychýlení počátečního odhadu x_0 hledaných parametrů od skutečné hodnoty x_{true}: • $x_0 = x_{true} \cdot 2{,}34$ • $x_0 = x_{true} \cdot \frac{1}{2{,}34}$ • $x_0 = x_{true} \cdot 10{,}71$ • $x_0 = x_{true} \cdot \frac{1}{10{,}71}$
Nastavení hranic prostoru parametrů	Dvě varianty nastavení limitů prostoru hledaných parametrů: • Od 0 do ± Inf (dle znaménka konkrétního parametru). • Od 0 do $\pm\, x_{true} \cdot 10^2$ (dle znaménka konkrétního parametru).

Tabulka 22. Nastavení srovnávacího testu optimalizačních algoritmů.

Pro účely praktické realizace srovnávacího testu bylo potřeba vytvořit několik doplňkových funkcí a jednu novou třídu objektů:

Hlavní třída pro simulaci modelů
`forSim`

Tato univerzální třída slouží pro nastavení modelu v Simulinku, opakované spouštění simulací se změněnými parametry během optimalizace, výpočet hodnoticí funkce RMSE, ukládání výstupních dat a monitorování celkového průběhu optimalizace (uplynulý čas, počet simulací, počet chyb).

Popis třídy, jejích vlastností a metod je uveden v tabulce níže (Tabulka 23). Pravděpodobně nejdůležitější a nejpoužívanější metodou této funkce je metoda

```
error = forSim.step(params)
```

která s použitím dodaných hodnot parametrů `params` provede jednu simulaci modelu, vyhodnocení a vrácení výstupu hodnoticí funkce RMSE v proměnné `error`. Odkaz na tuto metodu je pak vždy předáván optimalizační funkci, která ji volá v každé své iteraci.

Je nutné zmínit, že tato třída byla vytvořena ve spolupráci s Ing. Martinem Appelem, vzhledem ke tematickému průniku jeho dizertační práce „*Výzkum metod pro významné zrychlení odhadu parametrů simulačních modelů*“[8] a této dizertace. Pan Appel vytvořil hlavní strukturu a metody třídy, v rámci této práce pak byla třída dále rozšiřována a modifikována dle aktuálních potřeb.

Název vlastní třídy, název nadřazené třídy:	**Popis**
forSim < matlab.System	Třída forSim dědí z systémové třídy matlab.System
Výstupy	
error	Výstup hodnoticí funkce simulace pro dané parametry
Vlastnosti (veřejné):	
SimulationInput	Data vstupních signálů do modelu
error_counter	Počítadlo chyb detekovaných během simulací
isBuildRapidAcceleratorTarget	Nastavení módu kompilace modelu „Rapid Accelerator“ v Simulinku
isSaveData	Nastavení, zda má objekt ukládat data výstupu modelu
ExternalInput	Viz výše – SimulationInput (prakticky to stejné)
SimulationTime	Nastavení délky simulace
model	Název souboru modelu
ParametersVar_Names	Názvy parametrů modelu, které se během optimalizace mění
ParametersConst_Names	Názvy parametrů modelu, které se během optimalizace nemění (konstanty)
ParametersVar_InitVals	Hodnoty parametrů modelu, které se během optimalizace mění

[8] V době odevzdání této dizertace nebyla ještě práce Ing. Appela dokončena.

ParametersConst_Vals	Hodnoty parametrů modelu, které se během optimalizace nemění (konstanty)
MeasuredData	Vstupní vzorová („naměřená“) data pro porovnání s výstupem simulace a výpočet hodnoticí funkce
Solver	Nastavení řešiče modelu
OutputDataFormat	Nastavení formátu výstupních signálů z modelu.
errorFCN	Nastavení typu hodnoticí funkce (např. RMSE)
FastRestart	Nastavení funkce rychlého restartu modelu v Simulinku
SimulationMode	Nastavení módu v Simulinku (např. normal, rapid, …)
Buffer	Proměnná, do které se ukládají všechna data výstupních signálů po jednotlivých simulacích během optimalizace
numOfSim	Aktuální počet provedených simulací během optimalizace
Vlastnosti (neveřejné):	
numOfVariable	Počet parametrů, pro které se provádí optimalizace
timer	Nastavení vnitřního počítadla času
startTime	Čas začátku optimalizace (vytvoření objektu)
minOfError	Nejmenší dosažená hodnota hodnoticí funkce
bestValue	Hodnoty nejlepších nalezených parametrů
Metody zděděné:	
setupImpl	Inicializace modelu v Simulinku
stepImpl	Spuštění modelu v Simulinku
resetImpl	Re-inicializace modelu v Simulinku
Metody vlastní:	
save	Uložení dat v proměnné Buffer do specifikovaného souboru
output	Předání proměnné Buffer
stop	Uzavření modelu v Simulinku a ukončení časovače.
RMSE	Výpočet hodnoticí funkce metodou RMSE
ResampleSimulated	Převzorkování simulovaných výstupních dat tak, aby seděla s dodanými („měřenými“) daty.

Tabulka 23. Popis třídy forSim.

Výstupní funkce optimalizačních metod

```
varargout = MGO_optsolver_outfcn_universal(varargin)

varargout = MGO_optsolver_outfcn_GSMS(varargin)
```

Tyto funkce jsou volány v každé iteraci optimalizačního algoritmu a slouží k průběžnému ukládání mezivýsledků optimalizace. Testované algoritmy totiž ukládání mezivýsledků samy o sobě nepodporují. Předání odkazu na tyto funkce je pomocí parametru nastavení daného algoritmu `OutputFcn` (viz např. Tabulka 1).

První varianta funkce je určena pro lokální algoritmy, druhá je určena pro GlobalSearch a MultiStart.

Nastavení limitů parametrů pro experimenty

```
[lb,ub] = MGO_optsolver_bounds(parameters,method,
deviation_order)
```

Tato funkce slouží zejména pro snadnější a automatické nastavení srovnávacího testu. Na základě dodaných počátečních odhadů parametrů (`parameters`), jsou vygenerovány horní a dolní limity těchto parametrů (`lb, ub`) s využitím zvolené metody (viz níže) a řádu odchylky n (`deviation_order`).

K dispozici je 5 variant generování limitů parametrů, podle čísla vstupní proměnné `method`:

0. Prázdná proměnná, limity nenastaveny.
1. Beze změny znaménka. Vynásobení/podělení parametru hodnotu 10^n
2. Se změnou znaménka, symetrické kolem nuly. Vynásobení hodnotou 10^n a změna znaménka.
3. S možnou změnou znaménka, symetrické kolem parametru. Přičtení/odečtení hodnoty 10^n.
4. Jednostranně neomezené, bez změny znaménka. Limit od 0 do +/- Inf.
5. Jednostranně omezené, bez změny znaménka. Limit od 0 do +/- násobku hodnotou 10^n.

Nyní již byly popsány všechny nástroje, poklady a pomocné funkce k provedení srovnávacího testu, jehož výsledky budou představeny v následující kapitole.

5.1.2 Výsledky srovnávacího testu

Jak již bylo nastíněno v přechozí kapitole, bylo spuštěno a vyhodnoceno celkem 24640 optimalizačních úloh. Testy byly spouštěny automaticky a běžely nepřetržitě na několika počítačích současně[9]. I přesto trval výpočet cca týden.

Ze všech testů se ukládala data z celkového výsledku optimalizační úlohy i z jednotlivých průběžných iterací. Navíc byly do textových souborů ukládány i kompletní záznamy výpisů z příkazové řádky MATLABu pro odhalení případných chybových hlášek a jiných problémů.

Hlavní sledované metriky pro vyhodnocení srovnávacího testu byly:

- Finální hodnota hodnotící funkce (RMSE).
- Počet simulací modelu (výpočet hodnoticí funkce).

Je vhodné upozornit, že pro objektivní srovnání je třeba se dívat právě na počet simulací modelu, protože například počet iterací optimalizačního algoritmu je zavádějící (jedna iterace může znamenat několik simulací modelu). Taktéž informace o čase výpočtu nemusí být vhodná, pokud bychom prováděli simulace na různě výkonných PC.

V první fázi vyhodnocení byly nejprve označeny všechny experimenty, jejichž výsledek byl vyhodnocen jako irelevantní, aby neovlivňovaly celkové statistiky. Za irelevantní považujeme např. experimenty, během nichž došlo k havárii optimalizačního algoritmu, nebo experimenty,

[9] Všechny počítače měly stejnou hardwarovou konfiguraci, a tedy i výpočetní výkon.

kdy se hledaný parametr v kombinaci s konkrétním typem buzení neprojevil. Tyto jevy byly spíše výjimečné. Následně byla zpracováním všech uložených dat vytvořena hlavní přehledová tabulka, shrnující všechna důležitá data ze všech provedených experimentů.

Aby bylo možné vzájemně srovnat jednotlivé řešiče, není možné přímo numericky srovnávat počet simulací a finální RMSE. Mezi experimenty je rozdíl v obtížnosti zadání (počet hledaných parametrů, nepřesnost počátečního odhadu) i v tom, že výstup každého modelu se pohybuje v jiném měřítku (a tedy i hodnota RMSE).

Z tohoto důvodu bylo navržen následující postup dalšího zpracování výsledků:

- Srovnat optimalizační algoritmy (a jejich dvě varianty) vždy pro každý unikátní experiment zvlášť a reprezentovat výsledky jako normalizovanou hodnotu.
- Unikátním experimentem máme na mysli optimalizační úlohu, která má stejná tato nastavení:
    - model systému
    - kombinace hledaných parametrů
    - druh vstupního signálu do modelu
    - vychýlení počátečního odhadu hledaných parametrů od skutečné hodnoty
    - nastavení hranic prohledávaného prostoru parametrů
- Normalizace hlavních metrik, tedy počtu simulací a finálního RMSE v rámci každého unikátního experimentu byla provedena metodou škálování hodnot do intervalu od 0 do 1 (viz rovnice (5.1)).

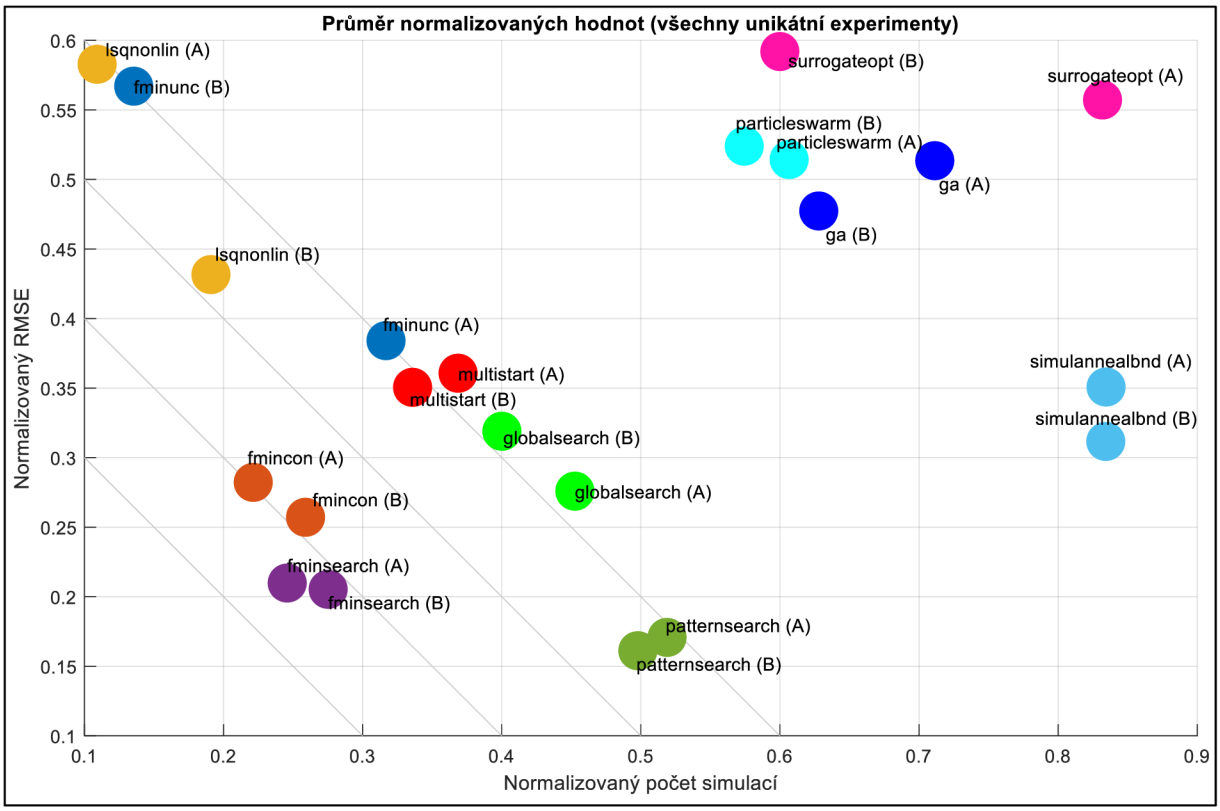
$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{5.1}$$

Celkem tedy dostaneme 2240 normalizovaných výsledků pro každý z testovaných optimalizačních algoritmů (24640 experimentů / 11 algoritmů). Vzhledem k tomu, že pro každý algoritmus jsme testovali dvě varianty jeho nastavení (A a B), dostáváme se na počet 1120 unikátních výsledků pro každou variantu všech algoritmů.

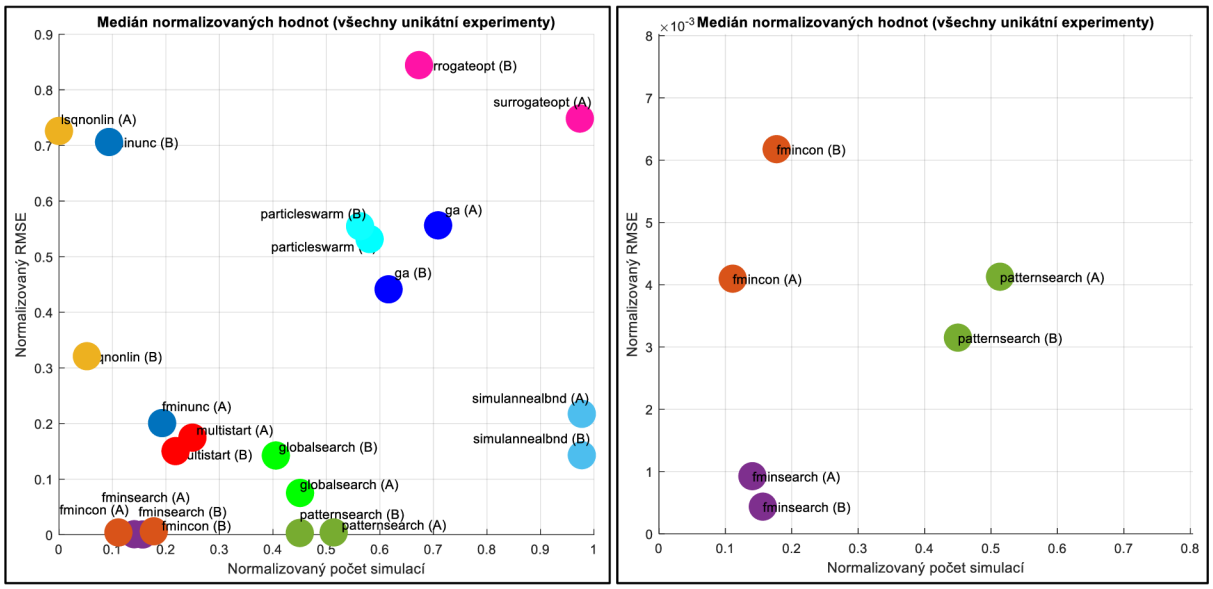
Shrňme tedy ještě jednou finální podobu zpracovaných výsledků, ze kterých budeme vycházet:

- Máme k dispozici 1120 unikátních experimentů, pro každý optimalizační algoritmus a jeho varianty.
- V rámci každého jednoho unikátního experimentu, jsou normalizovány dvě hlavní sledované metriky:
    - Počet simulací modelu v rámci dané optimalizační úlohy.
    - Výstupní (finální) hodnota hodnoticí funkce (RMSE) optimalizační úlohy.

Z těchto výsledků byl nakonec vytvořen výsledný graf, který je na Obr. 43. Tento graf byl vytvořen zprůměrováním normalizovaných hodnot obou sledovaných metrik napříč všemi experimenty. Pokud bychom místo průměrné hodnoty zvolili výpočet mediánu, tak se výsledné hodnoty samozřejmě změní, nicméně rozložení výsledků zůstane do značné míry zachováno (viz Obr. 44).



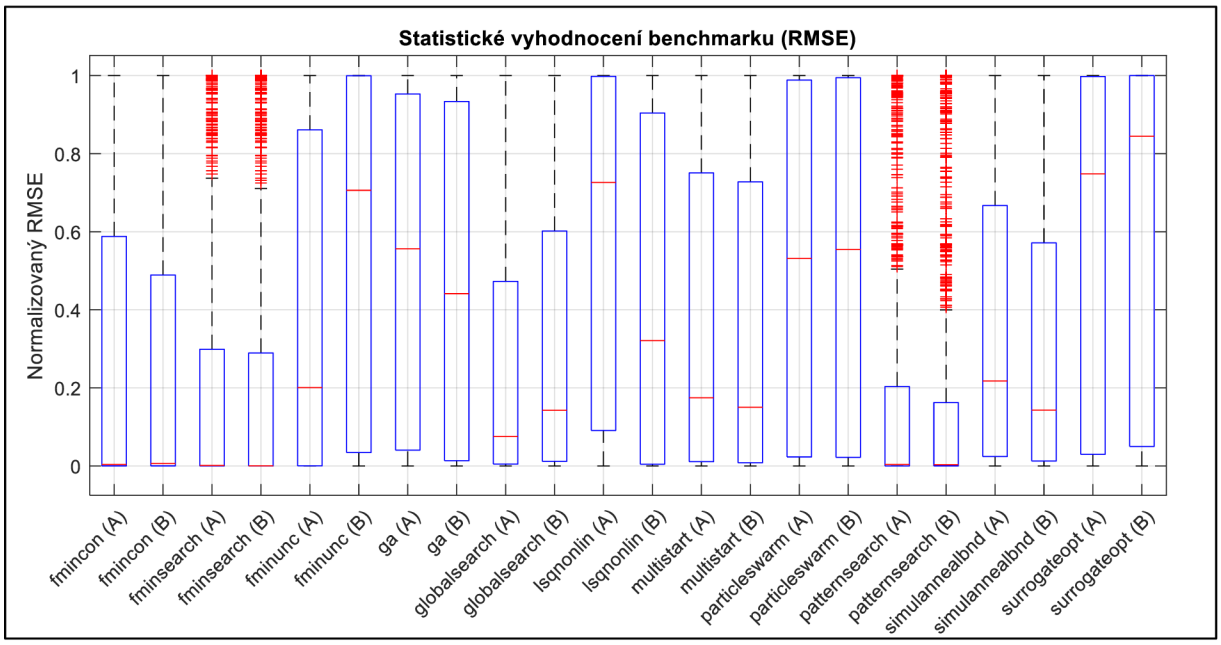
Obr. 43. Srovnání normalizovaných výsledků optimalizačních algoritmů (průměr).



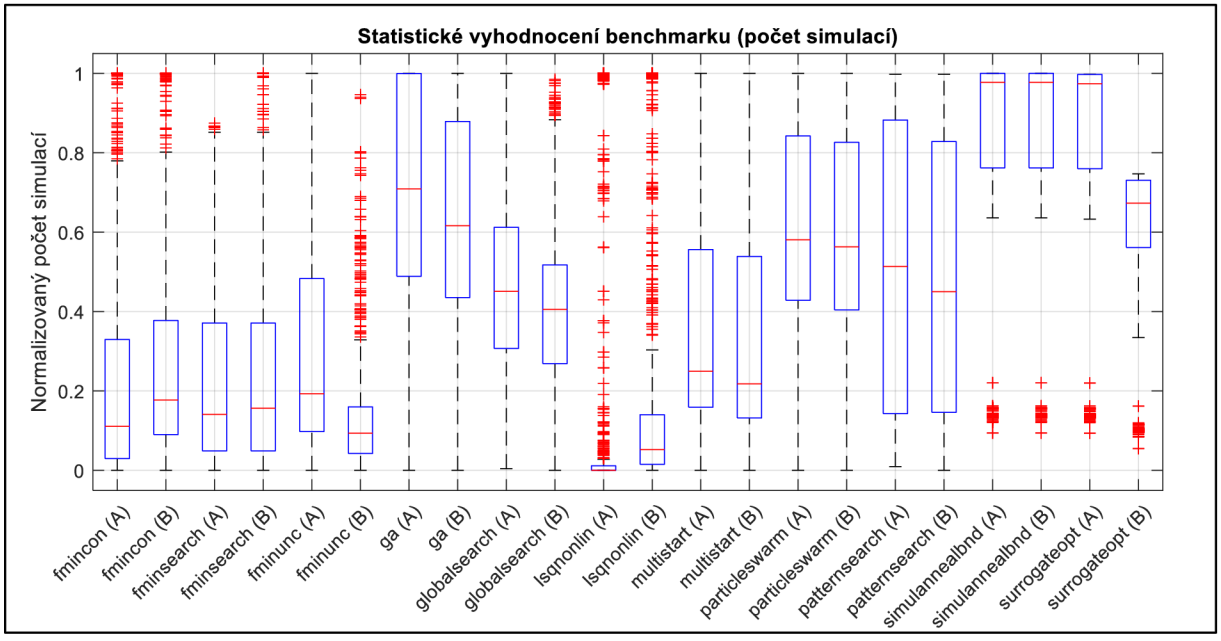
Obr. 44. Srovnání normalizovaných výsledků optimalizačních algoritmů (medián). Vlevo – celý prostor, vpravo – přiblížení levého dolního rohu.

Pokud bychom se nechtěli spoléhat jen na statistické ukazatele průměru a mediánu, můžeme se podívat i na podrobnější statistické vyhodnocení obou sledovaných metrik (Obr. 45, Obr. 46). Uvedené grafy byly vytvořeny pomocí funkce MATLABu `boxplot`. Středová značka každého rámečku označuje medián a spodní a horní okraj rámečku 25., resp. 75. percentil.

Metličky sahají k nejextrémnějším datovým bodům, které nejsou považovány za odlehlé hodnoty, a odlehlé hodnoty jsou vykresleny jednotlivě pomocí symbolu "+".



Obr. 45. Statistické vyhodnocení normalizovaných výsledků srovnávacího testu (RMSE)



Obr. 46. Statistické vyhodnocení normalizovaných výsledků srovnávacího testu (počet simulací)

5.1.3 Zhodnocení a poznatky pro návrh nových metod

Z výsledků uvedených v předchozí kapitole plyne, že nejúspěšnější optimalizační algoritmy pro námi definované nastavení experimentů jsou:

- fmincon – velmi rychlý, méně přesný.
- fminsearch – relativně rychlý i přesný.
- patternsearch – pomalejší, ale velmi přesný.

Může být zarážející, že lokální optimalizační algoritmy si vedly lépe než globální metody. To může být s největší pravděpodobností zapříčiněno tím, že globální metody potřebují v průměru větší počet simulací, než se projeví jejich výhoda, oproti lokálním algoritmům. Omezení maximálního počtu simulací pro každý experiment však bylo nutným předpokladem pro to, abychom získali dostatečně velký a rozmanitý statistický vzorek za rozumný výpočetní čas.

I přes toto omezení můžeme pozorovat, že některé globální algoritmy si vedou lépe než jiné. Nejlepší globální algoritmy jsou dle výsledků:

- GlobalSearch – spíše přesnější.
- MultiStart – spíše rychlejší.

Další globální optimalizační algoritmy (particleswarm, surrogateopt, simulanneal, ga) dosáhly viditelně horších výsledků. Jednou z příčin může být, že vzhledem k časové náročnosti výpočtu jedné iterace jsou pro náš typ úlohy nevhodné, protože vyžadují velké množství simulací, než začnou bezpečně konvergovat k řešení (i ve srovnání s GlobalSearch či MultiStart).

Na základě těchto poznatků z prvního velkého srovnávacího testu nyní v další kapitole popíšeme návrh a otestování nového optimalizačního algoritmu.

5.2 Nový optimalizační algoritmus

Na základě výsledků z předchozí kapitoly 5.1 budou nyní popsány hlavní požadavky na nový hybridní optimalizační algoritmus, který bude následně navržen, otestován a srovnán s dalšími globálními algoritmy pomocí druhého srovnávacího testu.

5.2.1 Požadavky pro tvorbu nového algoritmu

Důležitými požadavky na nový optimalizační algoritmus je mimo jiné jeho robustnost a automatické prohledávání prostoru parametrů. Typický uživatel tohoto algoritmu nemusí mít velké zkušenosti s optimalizačními úlohami, ale má k dispozici výpočetní výkon. Například mu nevadí, že optimalizace bude běžet několik hodin (třeba přes noc), hlavně když najde globální optimum, a to ideálně bez nutnosti zásahu uživatele během celého procesu.

Zkoumané prostory parametrů obsahují mnoho lokálních minim a nemusí být spojité což znevýhodňuje algoritmy využívající gradient hodnoticí funkce. Naopak výhodnými se jeví „direct search“ metody, což potvrzují i dosavadní výsledky testů.

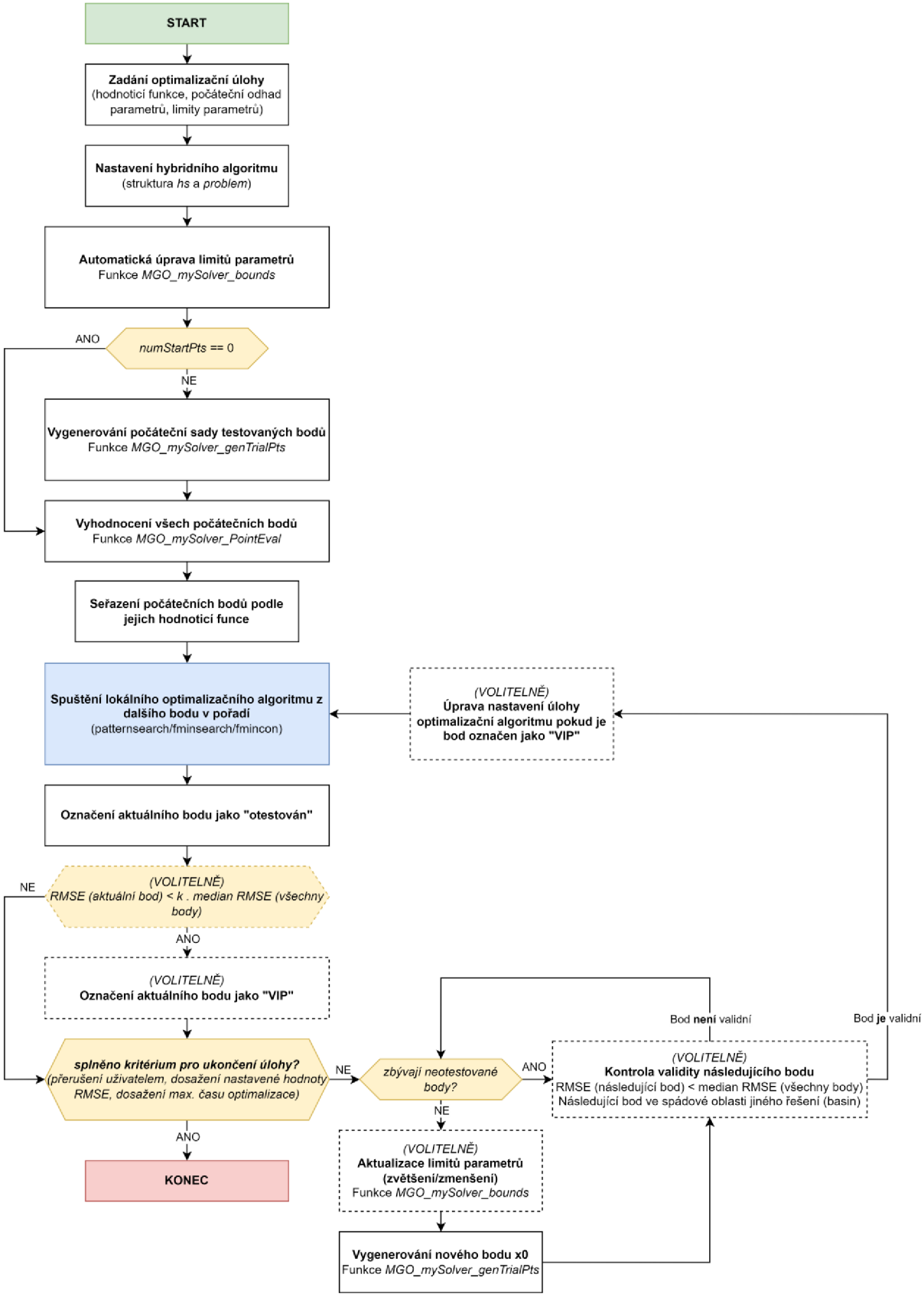
Globální optimalizační algoritmy jako GlobalSearch a MultiStart jsou výhodné z hlediska důkladného prohledávání prostoru parametrů (mnoho náhodných počátečních bodů), nicméně trpí různými nedostatky jako je omezená možnost volby lokálních optimalizačních metod či závislost na nastavení počtu náhodně generovaných počátečních bodů, což může vést k předčasnému ukončení optimalizace nebo naopak příliš mnoha iteracím. Předčasné (korektní) ukončení těchto algoritmů zásahem uživatele je také poměrně komplikované.

Nový hybridní algoritmus by se tedy měl inspirovat metodami globálního prohledávání a zaměřit se na „direct search“ metody. Dále by měl implementovat mechanismy pro automatické prohledávání bez zásahu uživatele a případně implementovat dodatečné heuristiky pro zrychlení konvergence optimalizačního algoritmu.

5.2.2 Design nového algoritmu

V této kapitole představíme námi navrženou strukturu nového hybridního optimalizačního algoritmu. Označení „hybridní“ volíme z důvodu, že kombinuje metody lokálních optimalizačních metod s globálním prohledáváním, a také pro jednoznačné odlišení od názvu jiných globálních algoritmů, zejména GlobalSearch.

Přehledový vývojový diagram hybridního algoritmu je vidět na Obr. 47. Jeho jednotlivé kroky a funkce pak postupně popíšeme níže.



Obr. 47. Vývojový diagram navrženého hybridního algoritmu

Zadání optimalizační úlohy
Jako první je nutné, podobně jako u ostatních optimalizačních algoritmů, zadat tři hlavní vstupy:

- Reference na hodnoticí funkci (v naší implementaci metoda objektu `forSim.step`)
- Počáteční odhad hledaných parametrů `x0`
- Limity hledaných parametrů `lb,ub` (pokud jsou zadány)

Nastavení hybridního algoritmu
Stejně jako u algoritmů GlobalSearch a MultiStart je nastavení všech parametrů optimalizační úlohy provedeno pomocí dvou proměnných. Struktura `hs` obsahuje nastavení a data nadřazené (globální) části algoritmu, `problem` pak obsahuje nastavení lokálního optimalizačního algoritmu.

Možnosti nastavení lokálního algoritmu v proměnné `problem` se odvíjí od zvolené optimalizační metody. Pro hybridní algoritmus je možné zvolit v zásadě libovolnou optimalizační metodu, my se však budeme zabývat pouze metodami `fmincon` (viz 2.2.1.2), `fminsearch` (viz 2.2.1.4) a `patternsearch` (viz 2.2.1.5).

Možnosti nastavení nadřazeného globálního algoritmu ve struktuře `hs` je uvedeno v tabulce níže (Tabulka 24). Některá nastavení budou blíže vysvětlena dále v rámci popisu jednotlivých kroků hybridního algoritmu.

Název proměnné ve struktuře hs	**Význam proměnné**
`x0`	Počáteční odhad hledaných parametrů zadaný uživatelem
`BestFval`	Nejmenší dosažená hodnota hodnoticí funkce
`BestX`	Hodnoty nejlepších nalezených parametrů
`Status`	Aktuální status algoritmu (text)
`MaxTime`	Maximální čas simulace (v sekundách). Pokud je nastaveno na Inf, nemá algoritmus pevné časové omezení.
`FvalTolerance`	Limitní hodnota hodnoticí funkce, pokud je nalezené řešení lepší než nastavená hodnota, optimalizační algoritmus se ukončí.
`ContinueForever`	Proměnná určující, zda má algoritmus pokračovat i po nalezení bodu s hodnoticí funkcí lepší, než je nastavený limit. Pokud je nastavena na `true`, běží algoritmus dál a snaží se najít ještě lepší řešení.

`StartPoints`
`∟ NumPts`
`∟ PtsDistribution`
`∟ DistributionSigma`
	Sada nastavení (struktura) pro vygenerování počáteční sady testovacích bodů (počátečních odhadů parametrů).

`NewPoints`
`∟ NumPts`
`∟ PtsDistribution`
`∟ DistributionSigma`
	Sada nastavení (struktura) pro generování nových testovacích bodů (počátečních odhadů parametrů) v průběhu hlavního cyklu algoritmu.

`BoundsGrowCoeff`	Koeficient, kterým se mají vynásobit limity prohledávaného prostoru parametrů. Výchozí hodnota je 1 (žádná změna).
`lb`	Aktuální hodnota dolních limitů hledaných parametrů.
`ub`	Aktuální hodnota horních limitů hledaných parametrů.

`trialPts`
`∟ ID`
`∟ status`
`∟ x0`
`∟ fval_x0`
`∟ x_sol`
`∟ fval_sol`
	Tabulka obsahující všechny testované body (počáteční odhady parametrů), jejich status a případně výsledky lokální optimalizační metody spouštěné z těchto bodů:
	• *ID* = identifikátor daného bodu
	• *Status* = textový popis stavu bodu
	• *x0* = počáteční odhad parametrů daného bodu
	• *fval_x0* = hodnota hodnoticí funkce v bodě počátečního odhadu
	• *x_sol* = hodnoty hledaných parametrů po ukončení lokální optimalizační úlohy pro daný bod
	• *fval_sol* = hodnota hodnoticí funkce po ukončení lokální optimalizační úlohy pro daný bod

Tabulka 24. Popis struktury hs obsahující nastavení a data hybridního optimalizačního algoritmu.

Po vytvoření proměnných `hs` a `problem` je možné spustit hybridní optimalizační algoritmus pomocí níže uvedené funkce, která zahrnuje všechny kroky algoritmu. Finálním výstupem této funkce je proměnná `x` obsahující nejlepší nalezené parametry.

```
x = MGO_run_hs(hs,problem)
```

V následující části budou nyní popsány jednotlivé kroky nového hybridního algoritmu.

Automatická úprava limitů parametrů
Syntaxe funkce v MATLABu:

```
[lb,ub]= MGO_mySolver_bounds(numOfParams,lb,ub,boundsGrowCoeff)
```

kde `numOfParams` je počet hledaných parametrů, `lb` a `ub` jsou hodnoty (příp. vektory) určující limity hledaných parametrů a `boundsGrowCoeff` je volitelný parametr koeficientu, kterým se mají vynásobit původní limity prohledávaného prostoru parametrů.

Tato funkce má dva hlavní účely:

- Kontrola nastavení parametrů `lb`, `ub`
  - Kontrola, zda počet členů `lb` a `ub` sedí se zadaným počtem parametrů `numOfParams`.
  - Kontrola, zda pro všechny parametry platí, že `lb` < `ub`.
  - Pokud jsou parametry `lb`, `ub` prázdné, nebo obsahují hodnotou ±Inf, je provedeno automatické přenastavení na výchozí hodnotu limitů $\pm 10^4$ (stejně jako u GlobalSearch).
- Rozšíření/zmenšení limitů parametrů `lb`, `ub`
  - Pouze pokud je funkce zavolána včetně parametru `boundsGrowCoeff`.

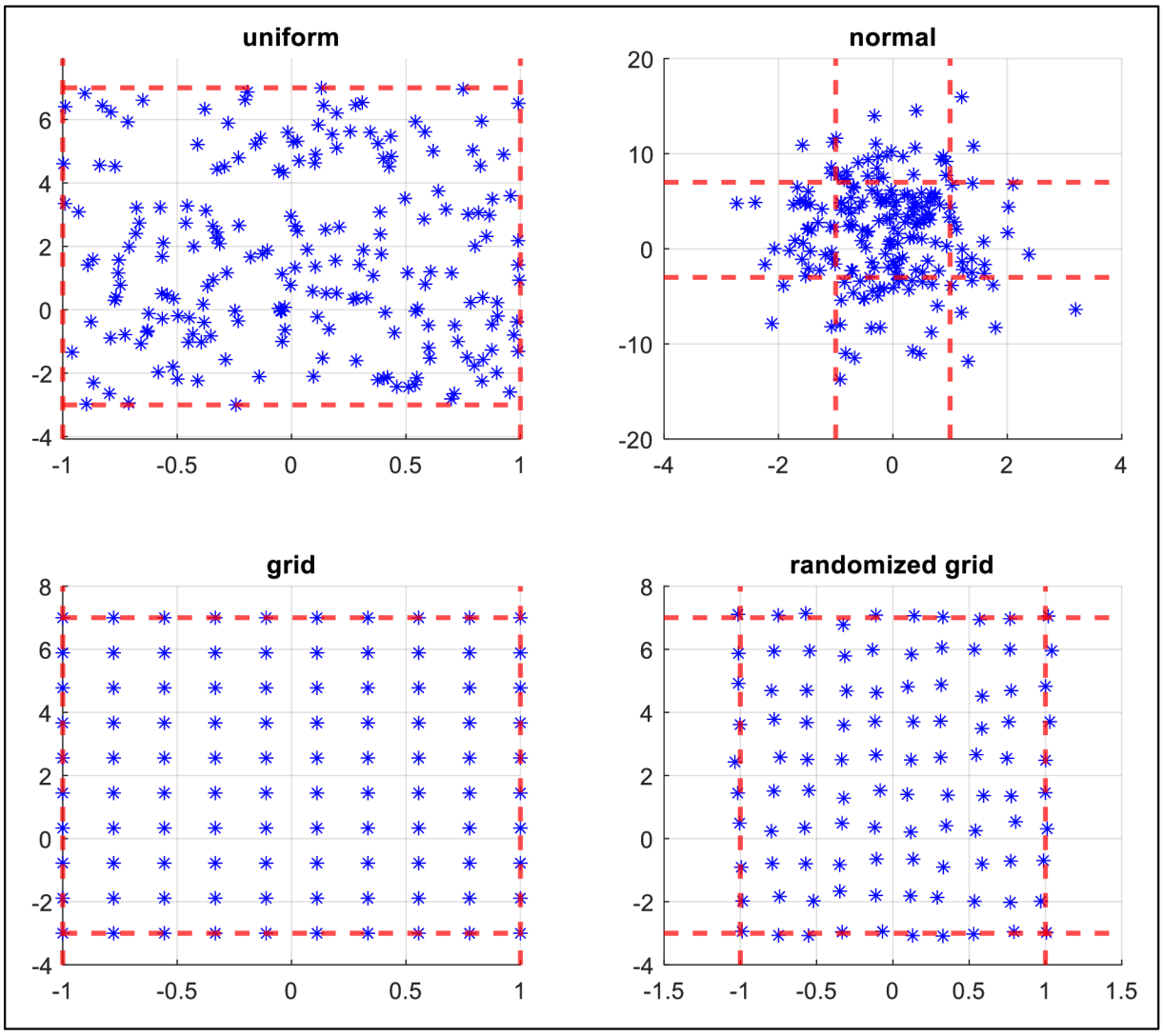
- Vstupní parametry limitů jsou vynásobeny koeficientem v proměnné `boundsGrowCoeff`.

Vygenerování počáteční sady testovaných bodů
Syntaxe funkce v MATLABu:

```
[trialPts] = MGO_mySolver_genTrialPts(numOfParams,lb,ub,
numPts,ptsDistribution,sigma_val)
```

kde výstupem funkce je tabulka testovaných bodů `trialPts` (viz Tabulka 24) a vstupními argumenty jsou počet hledaných parametrů (`numOfParams`), jejich limity (`lb,ub`), požadovaný počet vygenerovaných bodů `numPts`, typ náhodného rozdělení generovaných bodů `ptsDistribution` a směrodatná odchylka náhodného rozdělení `sigma_val`.

Dle nastavení parametru `ptsDistribution` jsou k dispozici čtyři varianty rozložení náhodně vygenerovaných počátečních testovaných bodů (viz Obr. 44).



Obr. 48. Ukázka variant rozložení náhodně generovaných bodů (modře), v rámci limitů (červeně).

Vyhodnocení všech počátečních bodů
Syntaxe funkce v MATLABu:

```
[hs] = MGO_mySolver_PointEval(hs,forSim)
```

Tato funkce s použitím reference na hodnoticí funkci (v našem případě objekt `forSim`) vypočítá její hodnotu pro všechny body počátečních odhadů parametrů v tabulce `hs.trialPts`, které mají status „untested“. Výsledky tohoto vyhodnocení jsou zapsány zpět do tabulky (sloupec `fval_x0`) a status těchto bodů je změněn na „tested_x0“ (případně „error_tested_x0“, pokud by došlo k chybě během simulace).

Seřazení počátečních bodů podle jejich hodnoticí funkce
Všechny body v tabulce `hs.trialPts` jsou následně seřazeny vzestupně dle jejich hodnoty `fval_x0`. Optimalizační algoritmus pak bude díky tomu testovat nejdříve body s nejmenší hodnoticí funkcí v místě počátečního odhadu, což zvyšuje pravděpodobnost na rychlejší nalezení globálního optima.

Spuštění lokálního optimalizačního algoritmu z dalšího bodu v pořadí
Hybridní algoritmus v tomto kroku vybere z tabulky `hs.trialPts` první bod v pořadí, který má status „tested_x0“ (případně „tested_VIP“) a spustí lokální optimalizační úlohu zadanou pomocí struktury `problem`. Jak již bylo zmíněno, v rámci této práce používáme pouze metody `fmincon`, `fminsearch` nebo `patternsearch`.

Pokud je bod označen jako „tested_x0“, je jako počáteční odhad parametrů optimalizačního algoritmu zvolena hodnota bodu `x0`. Pokud je status bodu „tested_VIP“, je namísto toho použita hodnota parametrů `x_sol` (viz Tabulka 24).

Označení aktuálního bodu jako "otestován"
Po ukončení běhu lokálního optimalizačního algoritmu je do tabulky `hs.trialPts` zapsána hodnota nalezených parametrů `x_sol` a hodnota hodnoticí funkce v tomto bodě `fval_sol`. Současně je změněn status bodu na „solver_finished“ (případně „solver_finished_VIP“).

Označení aktuálního bodu jako "VIP" (volitelné)
Pokud se uživatel rozhodne tento krok provést, je po ukončení lokální optimalizační úlohy proveden test, zda je dosažená hodnoticí funkce v daném bodě menší než medián výsledků hodnoticí funkce řešení pro všechny předchozí otestované body, vynásobený konstantou k. Na základě experimentů jsme zvolili hodnotu konstanty $k = 0{,}7$. V případě potřeby je však možné ji upravit.

Pokud aktuální bod tento výsledek splní, je jeho status změněn na „tested_VIP“, což ovlivňuje chování hybridního algoritmu v jednom z dalších kroků.

Test splnění kritéria pro ukončení úlohy
Dále je provedena kontrola, zda nebylo splněno některé z kritérií pro ukončení hybridního optimalizačního algoritmu:

- Manuální přerušení výpočtu uživatelem.
- Dosažení požadované hodnoty hodnoticí funkce, dle parametru `hs.FvalTolerance` (pokud není současně nastaven parametr `hs.ContinueForever`).
- Dosažení maximálního povoleného času výpočtu, dle parametru `hs.MaxTime`.

Pokud je některé z těchto kritérií splněno, algoritmus se ukončí a vrátí jako řešení nejlepší nalezený bod z tabulky `hs.trialPts`. V opačném případě algoritmus pokračuje v prohledávání prostoru parametrů.

Kontrola zbývajících neotestovaných bodů
Pokud v tabulce `hs.trialPts` zbývají body, ze kterých ještě nebyl spuštěn lokální optimalizační algoritmus (status „tested_x0"), je vybrán další takovýto bod v pořadí a následuje volitelný krok *Kontrola validity následujícího bodu*.

Pokud se již žádný takový bod v tabulce nenachází, pokračuje algoritmus krokem *Aktualizace limitů parametrů* (volitelný) nebo rovnou *Vygenerování nového bodu*.

Aktualizace limitů parametrů (volitelné)
V případě, že si uživatel není jistý hodnotami zvolených limitů parametrů a chce tyto limity během běhu optimalizace postupně zvětšovat nebo naopak zmenšovat, je možné využít v tomto kroku již dříve zmíněné funkce `MGO_mySolver_bounds`, která při zadání vstupního argumentu koeficientu úpravy limitů parametrů (`boundsGrowCoeff`) provede aktualizaci proměnných `lb` a `ub`.

Vygenerování nového bodu
V případě že již nejsou k dispozici další body počátečních odhadů parametrů pro lokální optimalizační algoritmus, je znovu využita dříve zmíněná funkce `MGO_mySolver_genTrialPts`, pro vygenerování dalšího bodu. Vstupní argument `numPts` je v tomto případě nastaven na hodnotu 1 a je možné volit pouze varianty rozložení `ptsDistribution` „normal" a „uniform". Pro nově vygenerovaný bod je pak vypočítána hodnoticí funkce (`fval_x0`) a bod je přidán na konec tabulky `hs.trialPts`.

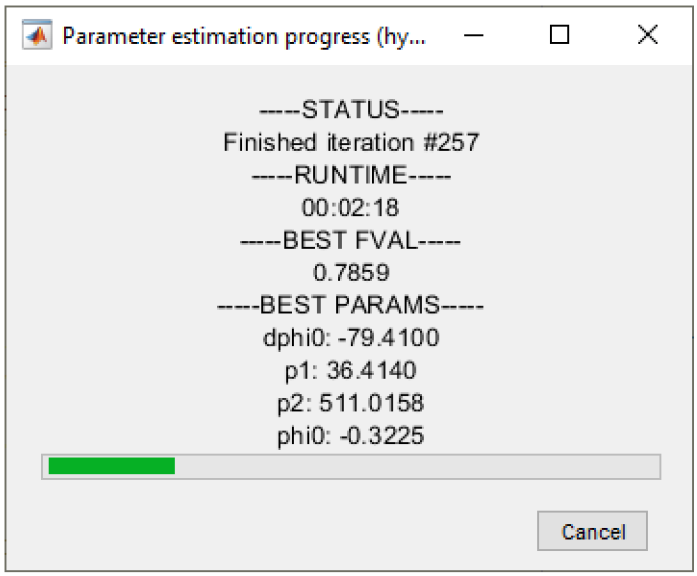
Kontrola validity následujícího bodu (volitelné)
V případě potřeby je možné v tomto kroku provést kontrolu validity bodu, který je další v pořadí pro testování. Je tak možné vyfiltrovat body, které například nemají lepší hodnotu `fval_x0`, než je medián této hodnoty pro všechny dosud testované body, případně ignorovat body, které leží ve spádové oblasti jiného řešení (podobně jak u GlobalSearch, viz kap. 2.2.2.5).

Pokud je následující bod validní, pokračuje algoritmus dalším krokem v pořadí. V opačném případě se vrátí ke kroku *Kontrola zbývajících neotestovaných bodů*.

Úprava nastavení úlohy optimalizační algoritmu, pokud je bod označen jako "VIP" (volitelné)
Pokud je další testovaný bod v pořadí označen jako „tested_VIP", je před spuštěním lokálního optimalizačního algoritmu provedena dočasná úprava jeho nastavení tak, aby bylo možné tento bod důkladněji prozkoumat. Toho můžeme dosáhnout například zvýšením maximálního počtu vyhodnocení hodnoticí funkce, které může lokální algoritmus provést (parametr `problem.MaxFunctionEvaluations`).

Poté již následuje krok *Spuštění lokálního optimalizačního algoritmu z dalšího bodu v pořadí* a celý proces se opakuje, dokud není splněna některá z podmínek pro ukončení hybridního optimalizačního algoritmu.

V průběhu optimalizace se také uživateli zobrazí jednoduché okno zobrazující aktuální stav (Obr. 49). Pomocí tohoto dialogového okna je navíc možné jednoduše manuálně ukončit optimalizaci bez toho, aby došlo ke ztrátě dat.



Obr. 49. Dialogové okno hybridního algoritmu s aktuálním stavem optimalizace.

5.2.3 Testování nového hybridního algoritmu a srovnání s jinými metodami

Poté co byl dokončen návrh nového hybridního algoritmu, byl proveden další velký srovnávací test algoritmů, podobně jako v kapitole 5.1.

Nastavení tohoto srovnávacího testu bylo v mnoha ohledech podobné, jako v předchozím případě, bylo však provedeno několik důležitých změn.

Níže uvedené položky srovnávacího testu byly ponechány stejné, jako v kapitole 5.1.1, resp. v tabulce tamtéž (Tabulka 22):

- 10 modelů systémů
- 7 variant kombinací hledaných parametrů
- 4 varianty vychýlení počátečního odhadu hledaných parametrů od skutečné hodnoty
- 2 varianty nastavení hranic prohledávaného prostoru parametrů

Následující položky byly upraveny:

- 6 optimalizačních algoritmů
  - Hybridní algoritmus (lokální algoritmus patternsearch)
  - Hybridní algoritmus (lokální algoritmus fminsearch)
  - Hybridní algoritmus (lokální algoritmus fmincon)
  - GlobalSearch
  - MultiStart
  - Genetický algoritmus
- 1 varianta nastavení optimalizačního algoritmu
  - Viz příloha (Obr. 74 až Obr. 79)
- 1 druh vstupního signálu do modelu.
  - Skokový signál

Kombinací všech těchto variant dostáváme celkový počet 3360 optimalizačních úloh, jejichž výsledky budeme následně vyhodnocovat.

Hlavní rozdíl oproti předchozímu srovnávacímu testu (mimo výše uvedených bodů), spočíval v tom, že pro každý optimalizační algoritmus byl výrazně zvýšen časový limit pro výpočet. Vyšší limit času pro výpočet dává algoritmům větší prostor pro důkladné prozkoumání prostoru parametrů a mělo by omezit případy, kdy se díky tomuto omezení nemohly projevit všechny silné stránky daného optimalizačního algoritmu.

Tento limit byl nastaven pro každý algoritmus stejně, dle následujícího vztahu (5.2):

$$t_{max} = n \cdot 1800\ s \tag{5.2}$$

kde n je počet hledaných parametrů. Časový limit pro jednu optimalizační úlohu se tak pohyboval v rozmezí $0{,}5 - 2$ hodiny, dle aktuálního nastavení experimentu.

Z důvodu výrazně delšího výpočetního času pro jeden experiment byla také provedena některá omezení v nastavení srovnávacího testu (např. pouze jeden druh vstupního signálu do modelu). I tak byla při celkovém počtu 3360 experimentů celková časová náročnost tohoto testu větší než v přechozím případě.

Než přejdeme k vyhodnocení výsledků tohoto testu, je nutné zmínit, že u hybridního algoritmu nebyly v tomto případě aplikovány všechny z volitelné kroky, které jsou popsány v přechozí kapitole 5.2.2 a na přehledovém diagramu algoritmu (Obr. 47). Použit byl pouze volitelný krok výběru VIP bodů pro důkladnější otestování.

Důvod pro toto rozhodnutí je, že v testu chceme primárně zkoumat výhodnost využití metod přímého prohledávání pro lokální optimalizaci. Pokud se ukáže, že hybridní algoritmus dosahuje dobrých výsledků i bez použití doplňkových funkcí filtrace bodů či úpravy limitů parametrů, bude s těmito funkcemi pravděpodobně fungovat ještě lépe. Druhým důvodem je, že v době testování ještě nebyla funkcionalita těchto volitelných kroků zcela odladěna a mohla by paradoxně způsobit zhoršení výsledků hybridního algoritmu.

5.2.4 Výsledky testů nového hybridního algoritmu a jejich zhodnocení

Experimenty nadefinované v předcházející kapitole bylo opět spouštěny automaticky na několika počítačích současně s průběžným ukládáním výsledků.

Na rozdíl od přechozího srovnávacího testu, jsme se v tomto případě zaměřili na dvě částečně odlišné metriky pro jeho vyhodnocení:

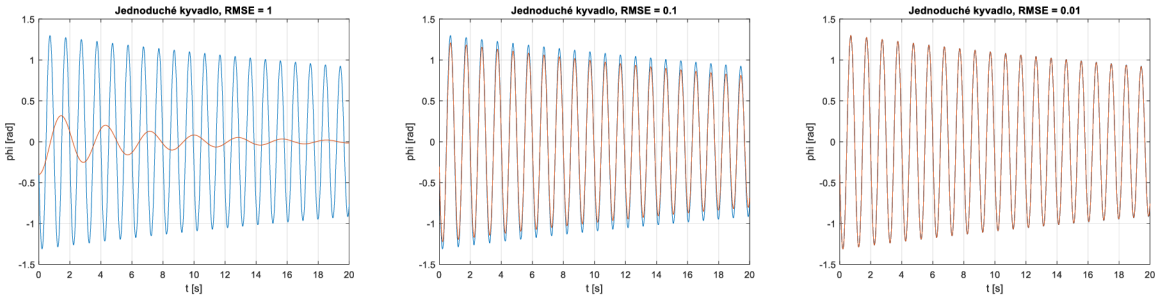
- Kolikrát (pro daný model) nalezl algoritmus globální minimum.
- V případě že algoritmus globální minimum nalezl, kolik vyhodnocení hodnoticí funkce (spuštění simulace) k tomu bylo zapotřebí.

Předpokladem pro volbu těchto dvou metrik bylo, že všechny algoritmy měly relativně dlouhý čas na nalezení globálního optima, proto nás v tomto případě zajímá už pouze fakt, zda jej nalezly či nikoliv a za jaký čas.

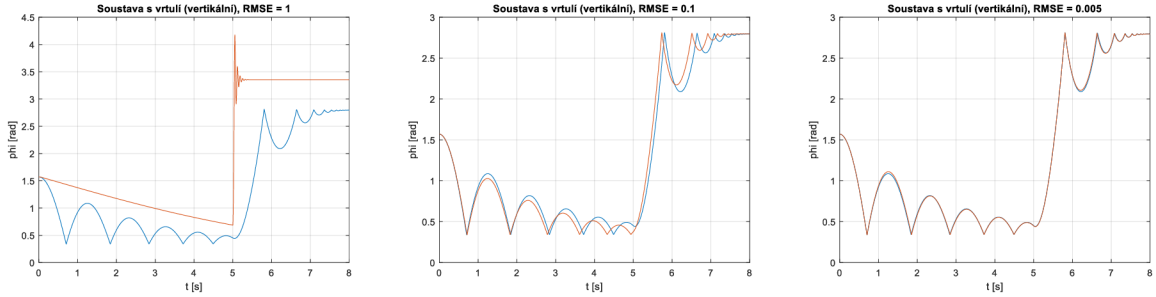
Ve chvíli, kdy máme takto zvolené sledované metriky, musíme samozřejmě určit limit pro to, co považujeme za globální optimum, na základě finální hodnoticí funkce (RMSE). Volba tohoto

limitu závisí nakonec vždy na uživateli, který zpravidla určuje kvalitu nalezených parametrů na základě vizuální inspekce grafů odezvy původního systému a modelu s odhadnutými parametry. Tento postup jsme zvolili i v případě této práce.

Pro každý model jsme tedy provedli vizuální inspekci grafů jejich odezvy. V grafech byla vždy zobrazena odezva modelu s původními (reálnými) parametry a dále odezva systému s odhadnutými parametry. Toto srovnání jsme u každého modelu provedli pro několik hodnot výsledného RMSE a rozhodli jsme, jaká je limitní hodnota RMSE, kdy se již oba průběhy signálu dostatečně shodují. Příklady tohoto postupu jsou uvedeny na Obr. 50 a Obr. 51.



Obr. 50. Srovnání původního signálu (modře) s odezvou modelu s nalezenými parametry (červeně) pro různé hodnoty RMSE (model Jednoduché kyvadlo).
Vlevo – velmi špatná shoda, Střed – relativně dobrá shoda, Vpravo – téměř dokonalá shoda



Obr. 51. Srovnání původního signálu (modře) s odezvou modelu s nalezenými parametry (červeně) pro různé hodnoty RMSE (model Soustava s vrtulí (vertikální)).
Vlevo – velmi špatná shoda, Střed – relativně dobrá shoda, Vpravo – téměř dokonalá shoda

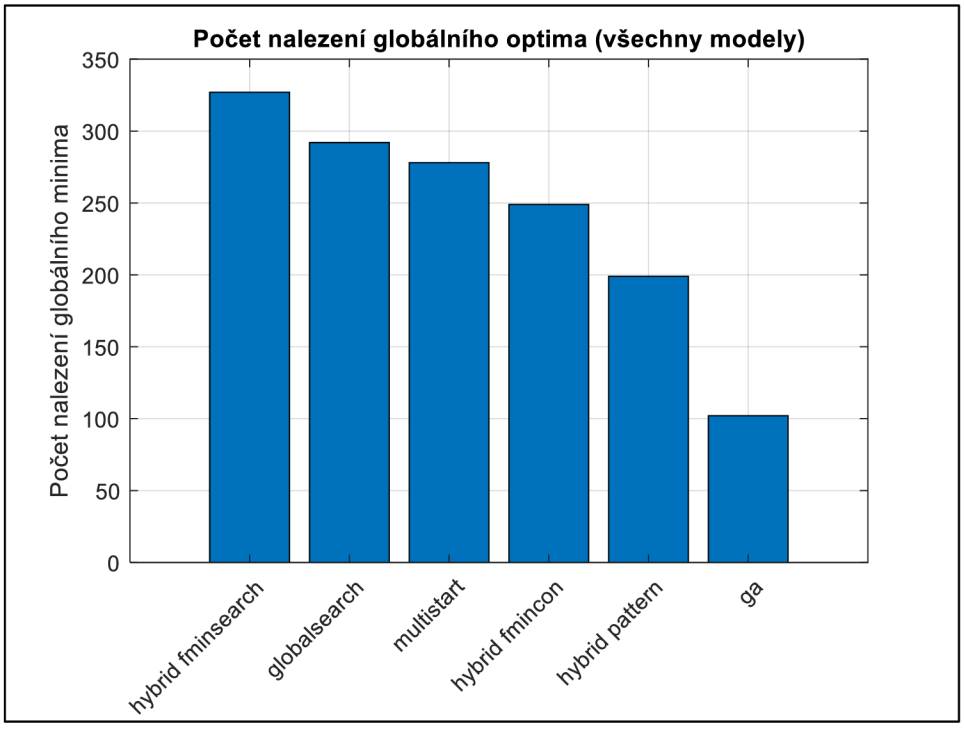
Konkrétní limity hodnoty RMSE, určující zda již nalezené řešení považujeme za globální optimum, jsou uvedeny v tabulce níže (Tabulka 25).

Název modelu	**Limit hodnoticí funkce (RMSE) pro uznání globálního optima**
Jednoduché kyvadlo	$1 \cdot 10^{-2}$
DC motor	$5 \cdot 10^{-4}$
Dvojité kyvadlo	$1 \cdot 10^{-2}$
Rotační kyvadlo	$1 \cdot 10^{-3}$
Soustava s vrtulí (horizontální)	$1 \cdot 10^{-4}$
Soustava s vrtulí (vertikální)	$5 \cdot 10^{-3}$

Těleso na naklápěcí plošině	$5 \cdot 10^{-4}$
Kotoučová brzda	$1 \cdot 10^{-1}$
DC motor + setrvačník na pružině	$1 \cdot 10^{-1}$
DC motor + závaží na rameni	$1 \cdot 10^{-1}$

Tabulka 25. Limity hodnoticí funkce (RMSE) pro uznání globálního optima (všechny modely)

Když nyní máme definované limity pro prohlášení řešení za globální optimum, můžeme provést vyhodnocení výsledků srovnávacího testu. Počet nalezení globálního optima napříč všemi modely a experimenty je níže na Obr. 52. V grafu můžeme pozorovat, že nejlepších výsledků dosáhl náš nově navržený hybridní optimalizační algoritmus ve variantě s lokálním algoritmem fminsearch.



Obr. 52. Počet nalezení globálního optima napříč všemi experimenty.

Abychom mohli lépe vyhodnotit výsledky srovnávacího testu, je vhodnější se zaměřit na jednotlivé modely, v rámci nichž by měla být lépe srovnatelná i metrika počtu potřebných simulací pro dosažení globálního optima. Srovnávat tuto metriku napříč všemi modely není vhodné.

Jako příklad na tomto místě nyní uvedeme grafy výsledků pro jeden konkrétní model (DC motor + závaží na rameni). Tyto výsledky jsou níže na Obr. 53 a skládají se ze tří grafů:

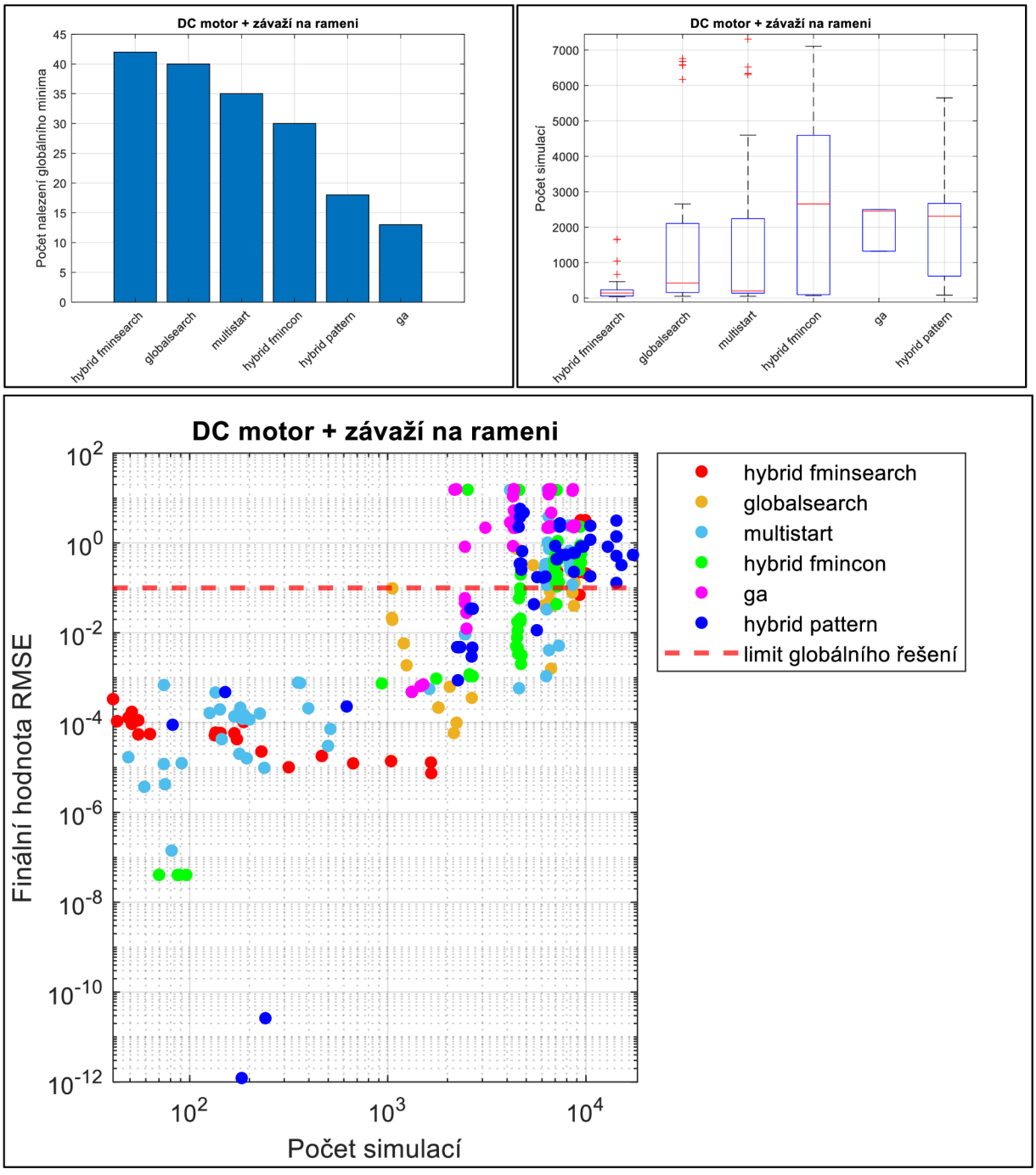
- Vlevo nahoře: graf počtu nalezení globálního optima jednotlivých algoritmů.
- Vpravo nahoře: statistické vyhodnocení[10] počtu potřebných simulací k dosažení globálního optima. Toto vyhodnocení je provedeno pomocí funkce `boxplot`, stejně jako v kapitole 5.1.2.

[10] Středová značka každého rámečku označuje medián a spodní a horní okraj rámečku 25., resp. 75. percentil. Metličky sahají k nejextrémnějším datovým bodům, které nejsou považovány za odlehlé hodnoty, a odlehlé hodnoty jsou vykresleny jednotlivě pomocí symbolu "+".

- Dole: vykreslení všech výsledků optimalizačních úloh pro daný model, bez ohledu na nalezení globálního optima.

Z uvedených grafů pro tento konkrétní model můžeme vyčíst, že nový hybridní algoritmus ve variantě fminsearch má nejen nejlepší úspěšnost v počtu dosažených globálně optimálních řešení, ale je i v průměru výrazně rychlejší.

Všechny přehledové grafy výsledků tohoto srovnávacího testu pro jednotlivé modely, jsou pro větší přehlednost uvedeny v přílohách (Obr. 80 až Obr. 89).



Obr. 53. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (DC motor + závaží na rameni)

Pokud bychom měli pro jednotlivé modely stručně shrnout dosažené výsledky hybridního algoritmu v porovnání mezi sebou a s ostatními testovanými globálními metodami, je možné tyto výsledky slovně heslovitě vyjádřit pomocí tabulky níže (Tabulka 26).

Model	**Hybridní algoritmus (fminsearch)**	**Hybridní algoritmus (fmincon)**	**Hybridní algoritmus (patternsearch)**
Jednoduché kyvadlo	Nejúspěšnější a výrazně nejrychlejší	Velmi úspěšný a rychlý	Málo úspěšný a pomalý
DC motor	Středně úspěšný, středně rychlý	Středně úspěšný ale nejrychlejší	Málo úspěšný a pomalý
Dvojité kyvadlo	Nejúspěšnější a nejrychlejší	Průměrně úspěšný a rychlý	Velmi úspěšný, středně rychlý
Rotační kyvadlo	Nejúspěšnější ale pomalejší	Málo úspěšný ale rychlý	Velmi málo úspěšný a pomalý
Soustava s vrtulí (horizontální)	Nejúspěšnější, středně rychlý	Málo úspěšný, ale rychlý	Velmi málo úspěšný a pomalý
Soustava s vrtulí (vertikální)	Středně úspěšný, středně rychlý	Velmi málo úspěšný, středně rychlý	Nejúspěšnější a středně rychlý
Těleso na naklápěcí plošině	Nejúspěšnější a výrazně nejrychlejší	Neúspěšný, a pomalý	Středně úspěšný a středně rychlý
Kotoučová brzda	Nejúspěšnější a nejrychlejší	Nejúspěšnější a rychlý	Nejúspěšnější ale pomalý
DC motor + setrvačník na pružině	Nejúspěšnější a nejrychlejší	Středně úspěšný a rychlý	Středně úspěšný a velmi pomalý
DC motor + závaží na rameni	Nejúspěšnější a výrazně nejrychlejší	Středně úspěšný a pomalý	Málo úspěšný a pomalý

Tabulka 26. Slovní shrnutí úspěšnosti nalezení globálního optima a rychlosti konvergence hybridního algoritmu pro jednotlivé modely.

Po uvážení všech výše uvedených informací, lze konstatovat, že nově navržený algoritmus Hybrid-fminsearch, pro námi zkoumanou úlohu odhadu parametrů mechatronických systémů, přináší nezanedbatelné zlepšení oproti jiným optimalizačním algoritmům, které jsou dostupné v programu MATLAB. Toto zlepšení se projevuje v průměru větší pravděpodobností nalezení globálního optima za kratší čas.

5.3 Zhodnocení nově navrženého algoritmu

V průběhu kapitoly 5 (Optimalizační algoritmy a nová metoda) jsme nejdříve popsali návrh prvního srovnávacího testu optimalizačních algoritmů, které jsou vhodné pro naši úlohu a jsou již implementovány v programu MATLAB.

V rámci tohoto prvního testu jsme připravili velkou sadu 24640 experimentů, zahrnující všechny optimalizační algoritmy popsané v kapitole 2.2, vytvořené modely z kapitoly 4.1 a

sadu různých nastavení a počátečních podmínek optimalizační úlohy. Hlavní sledované metriky prvního testu byly hodnota hodnoticí funkce a počet simulací po ukončení optimalizační úlohy.

Na základě vyhodnocení výsledků prvního srovnávacího testu jsme formulovali základní požadavky na nový hybridní optimalizační algoritmus, který byl následně popsán a otestován v rámci druhého srovnávacího testu, který zahrnoval tři varianty nového hybridního algoritmu a pro srovnání tři globální algoritmy z prvního testu. Hlavní sledované metriky druhého testu byly počet nalezení globálního optima a počet simulací pro jeho dosažení.

Z výsledků druhého srovnávacího testu plyne, že navzdory rozdílům ve výsledcích mezi jednotlivými modely systémů, dosahuje nově navržený algoritmus Hybrid-fminsearch v průměru nejlepších výsledků.

6 Pomocné algoritmy pro úlohu odhadu parametrů

Jak již bylo nastíněno v jedné z úvodních kapitol (2.1) a následně formulováno v cílech dizertační práce (kap. 3), jedním z důvodů neúspěšného či pomalého odhadu parametrů je kromě samotné optimalizační metody i nevhodná struktura modelu, počet parametrů či podoba vstupních naměřených dat. I v případě že je úloha úspěšně vyřešena, může dojít k nevhodné interpretaci dosažených výsledků, jako například nalezení nesmyslných hodnot parametrů díky jejich vzájemné provázanosti.

V této kapitole budou tedy nastíněny oblasti, kde má smysl se zabývat analýzou samotného nastavení a vstupů pro optimalizační úlohu.

Ke každé podkapitole bude popsána její problematika a navrženo řešení (algoritmus), pomocí kterého bude možné problém buď vyřešit, nebo alespoň vizualizovat tak, aby jej mohl uživatel vyřešit sám.

6.1 Detekce korelovaných parametrů

V kapitole popisující námi vytvořenou sadu mechatronických modelů (4.1) jsme několikrát zmínili neseparovatelné (korelované) parametry. Při tvorbě modelu může nastat situace, že vzhledem k jeho struktuře není možné jednoznačně určit hodnotu některých parametrů pouze na základě reakce výstupu systému na vstup (buzení). Je to z toho důvodu, že výstup či stav modelu je závislý na kombinaci několika parametrů a je možné dosáhnout stejné odezvy například současným zvětšením jednoho parametru a zmenšením jiného.

Tato situace není příliš častá u jednoduchých systémů, u kterých si uživatel tvoří dynamické rovnice a blokové schéma ručně. V tom případě je vzájemná závislost parametrů evidentní již ze samotných rovnic.

Problém může nastat v případě využití nástrojů, které rovnice chování systému sestavují automaticky namísto uživatele, jako je tomu právě v případě nástroje pro fyzikální modelování Simscape, který také v této práci využíváme. Uživatel (zejména méně zkušený) si pak nemusí při tvorbě modelu uvědomit všechny souvislosti a vzájemnou provázanost parametrů.

Pokud řešíme úlohu odhadu neznámých parametrů, můžeme se pak v případě, že jsou tyto parametry vzájemně závislé, dopracovat k řešení, které je z pohledu optimalizačního algoritmu kvalitní, ovšem parametry neodpovídají reálnému systému. V tomto případě totiž existuje nekonečně mnoho optimálních řešení této úlohy.

Níže uvedené algoritmy tedy nabízí uživateli nástroj, pro detekci takovýchto závislostí, případně jejich vyloučení.

6.1.1 Detekce korelovaných parametrů metodou Monte Carlo

První algoritmus slouží k otestování vzájemné závislosti parametrů metodou Monte Carlo a byl popsán ve vlastním samostatném článku s názvem „Monte Carlo Based Detection of Parameter Correlation in Simulation Models“ [73].

Vzhledem k tomu, že je algoritmus důkladně popsán v samotném článku, uvedeme zde jen jeho stručné shrnutí.

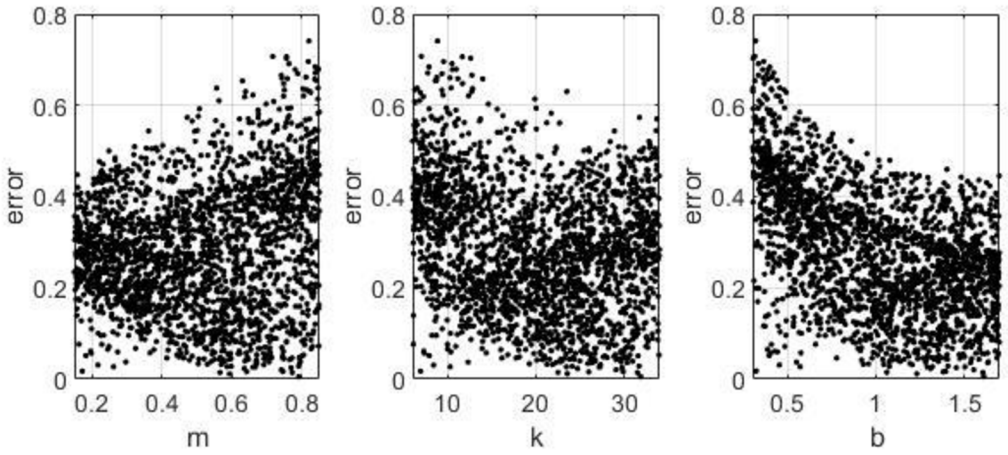
V zásadě nám uvedená metoda neumožní přímo detekovat závislé (korelované) parametry, ale dokáže ověřit, zda jsou skutečně nezávislé, tedy samostatně a jednoznačně identifikovatelné. Pokud se jejich nezávislost nepotvrdí, je pravděpodobné, že by mohly být závislé.

Tento test má také smysl dělat až poté, co je ukončena úloha estimace parametrů a bylo nalezeno globální (nebo alespoň lokální) minimum, aby se mohla projevit typická charakteristika v simulovaných datech.

Základní myšlenka testu je následující. Pokud již máme úspěšně nalezeny hodnoty hledaných parametrů na základě optimalizační úlohy, můžeme hodnoty všech nalezených parametrů náhodně vychýlit a provést vyhodnocení hodnoticí funkce (např. RMSE). Toto náhodné vychýlení a vyhodnocení výsledku musíme provést tolikrát, abychom dostali reprezentativní statistický vzorek.

Výsledné závislosti hodnoticí funkce na jednotlivých vychýlených parametrech vykreslíme do bodového grafu. Pokud jsou parametry vzájemně závislé, bude existovat více řešení se stejnou hodnotou hodnoticí funkce jako u původně nalezeného řešení. Viz rovnice (6.1) pro jednouchých mechanický oscilátor se třemi (závislými) parametry k, m, b, a graf výsledku na Obr. 54.

$$\ddot{x} = -\frac{k}{m}x - \frac{b}{m}\dot{x} \tag{6.1}$$



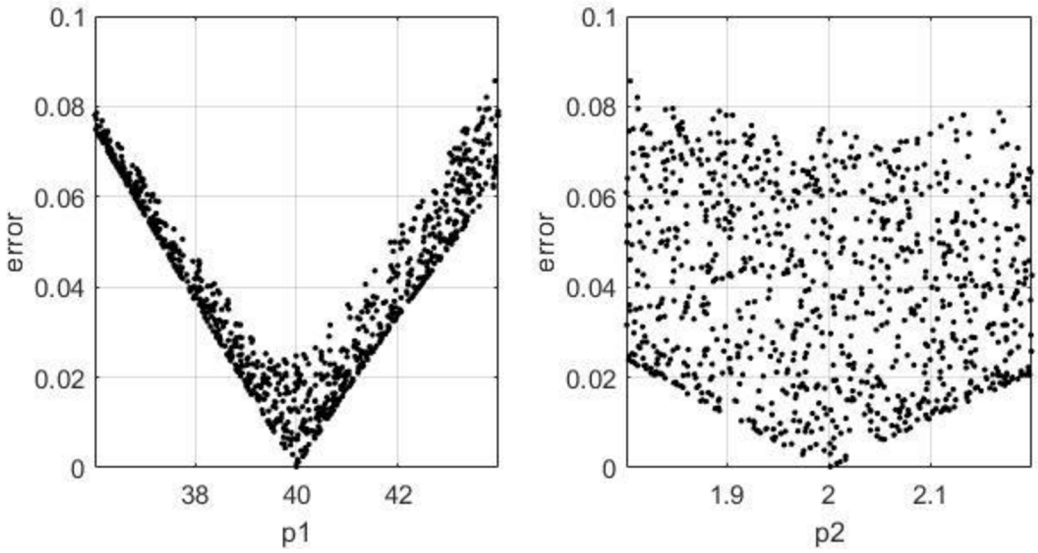
Obr. 54. Grafy hodnoticí funkce pro mechanický oscilátor se třemi závislými parametry. [73]

Naopak pokud jsou parametry nezávislé, měl by být ve výsledných grafech jasně patrný tvar „V“, protože při vychýlení nezávislého parametru z optimální hodnoty se hodnoticí funkce vždy zvětší. Úprava rovnic oscilátoru tak, aby obsahovala pouze nezávislé parametry p_1, p_2, je uvedena níže v rovnicích (6.2) až (6.4). Korespondující graf hodnoticí funkce s charakteristickým „V“ tvarem je na Obr. 55.

$$p_1 = \frac{k}{m} \tag{6.2}$$

$$p_2 = \frac{b}{m} \tag{6.3}$$

$$\ddot{x} = -p_1 x - p_2 \dot{x} \tag{6.4}$$



Obr. 55. Grafy hodnoticí funkce pro mechanický oscilátor se dvěma nezávislými parametry. [73]

6.1.2 Rychlá preventivní detekce neseparovatelných parametrů

Nevýhodou předchozího algoritmu je, že pro jeho správné fungování je nutné jej provést prakticky až po ukončení optimalizační úlohy, je tedy pro uživatele spíše indikací, že nalezené řešení nemusí být unikátní. Další nevýhodnou je relativně vysoká výpočetní náročnost z důvodu nutnosti vygenerovat dostatečný počet bodů, přičemž vyhodnocení každého bodu vyžaduje výpočet simulace modelu.

Proto byla navržena další, v zásadě velmi triviální, zato však rychlá a efektivní metoda, která dokáže detekovat závislé, neseparovatelné parametry ještě před začátkem optimalizační úlohy.

Při vytváření modelu se může stát, že parametry jsou zvoleny tak, že jsou např. v součinu a nelze je odseparovat. Pro tyto nejjednodušší případy lze udělat rychlý test, kdy při provedení jednoduchých operací s neznámými parametry vyjde zcela identický výsledek simulace.

Hlavní myšlenka spočívá v tom, že pokud například při součinu dvou parametrů tyto parametry prohodíme, výsledek simulace se nezmění. Pro lepší ilustraci je několik základních případů uvedeno v tabulce níže (Tabulka 27).

V tabulce je vždy v prvním sloupci uvedena základní rovnice, určující vzájemný vztah dvou parametrů a a b. Proměnnou x pak můžeme považovat za vstup do systému a y za výstup. V následujících sloupcích jsou pak uvedeny dva příklady ekvivalentních operací (substitucí), po jejichž provedení by se pro danou základní rovnici neměl výsledek změnit. Proměnná k v posledním sloupci představuje libovolnou pomocnou konstantu.

Jinými slovy, pokud provedeme se dvěma zkoumanými parametry a a b některou z ekvivalentních operací a zjistíme, že se výstupní signál systému vůbec nezměnil, je důvodné podezření že tyto dva parametry jsou závislé a neseparovatelné.

Základní rovnice	**Ekvivalentní operace 1**		**Ekvivalentní operace 2**	
$y = (ab)x$	$y = (ba)x$	$a = b$ $b = a$	$y = \left(ak\ b\frac{1}{k}\right)x$	$a = ak$ $b = b\frac{1}{k}$
$y = \left(\frac{a}{b}\right)x$	$y = \left(\frac{\frac{1}{b}}{\frac{1}{a}}\right)x$	$a = \frac{1}{b}$ $b = \frac{1}{a}$	$y = \left(\frac{ak}{bk}\right)x$	$a = ak$ $b = bk$
$y = (a + b)x$	$y = (b + a)x$	$a = b$ $b = a$	$y = \big((a + k) + (b - k)\big)x$	$a = a + k$ $b = b - k$
$y = (a - b)x$	$y = \big(-b - (-a)\big)x$	$a = -b$ $b = -a$	$y = \big((a + k) - (b + k)\big)x$	$a = a + k$ $b = b + k$
$y = (a^b)x$	$y = \left((a^2)^{\frac{b}{2}}\right)x$	$a = a^2$ $b = \frac{b}{2}$	$y = \left((a^k)^{\frac{b}{k}}\right)x$	$a = a^k$ $b = \frac{b}{k}$

Tabulka 27. Příklady ekvivalentních matematických operací a souvisejících substitucí.

Tento postup má také výhodu, že je nezávislý na přesnosti počátečních odhadů zkoumaných parametrů. V praxi může mít vliv pouze zaokrouhlovací chyba, případně přítomnost náhodné veličiny v simulaci (toto je možné vyřešit inicializací generátoru pseudonáhodných čísel v MATLABu před každou simulací).

Nevýhodou je, že tento test funguje pouze pokud se parametry vyskytují jen na jednom místě rovnice. Uvedený postup pro parametry a a b nebude fungovat například u níže uvedené rovnice (6.5):

$$y = (ab)x + \sin(bx) \tag{6.5}$$

Dalším úskalím této metody, které je nutné mít při aplikaci na paměti je, že nezměněný výsledek simulace nutně nezaručuje, že jsou parametry závislé. Tento stav totiž může nastat i v jiných případech, například když se ani jeden z parametrů modelu pro daný vstup do modelu neprojeví. Jakákoliv změna parametrů pak nemá vůbec žádný vliv na výsledek simulace. Takovýto případ však lze naštěstí také poměrně snadno odhalit.

Varianty závislostí dvou parametrů uvedené v tabulce výše jsou samozřejmě jen reprezentativním příkladem pro nejčastější případy a je možné je dále rozšiřovat.

6.2 Analýza vstupních dat

Druhou kategorií pomocných algoritmů, které budou popsány níže, je analýza vstupních dat naměřených na reálném systému, které uživatel dodává jako referenční vstup pro optimalizační úlohu, resp. vybuzení systému a výpočet hodnoticí funkce (rozdílu mezi měřeným a simulovaným výstupem systému).

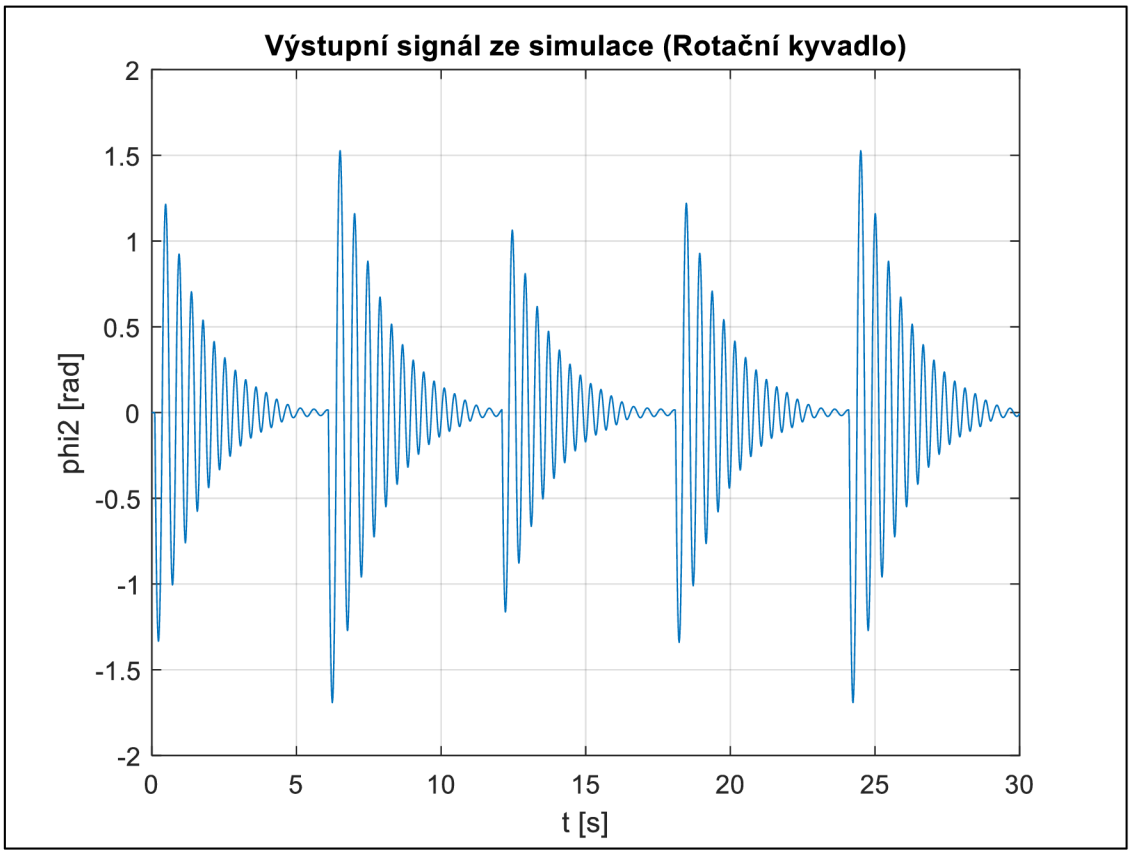
Problematika týkající se právě podoby vstupních dat byla již nastíněna v kapitole 2.1.2. V níže uvedených podkapitolách pak představíme dva algoritmy, které se některými ze zmíněných problémů zabývají.

6.2.1 Detekce opakování signálu

Volba vstupního signálu do systému ovlivňuje několik aspektů úlohy odhadu parametrů a jedním z nejdůležitějších bývá samotná délka simulace. Ta je totiž přímo závislá na délce vstupního signálu, pro který je nutné odsimulovat odezvu systému.

V níže uvedeném algoritmu se zaměříme na jeden konkrétní případ nevhodné volby signálu a jeho detekci, a tím je opakující se signál bez přidané hodnoty.

Je zřejmé, že pokud se tvar vstupního, ale zejména výstupního signálu periodicky opakuje bez výrazné odchylky, je možné říci, že z dat následujících po první periodě již pravděpodobně nezískáme žádné nové informace o chování systému (viz příklad na Obr. 56). V tomto případě je pak vhodné neužitečnou část dat odstranit, čímž se zkrátí doba simulace modelu, a tedy i rychlost výpočtu samotné optimalizační úlohy.



Obr. 56. Příklad opakujícího se výstupního signálu (model Rotačního kyvadla).

Námi navržený algoritmus pro automatickou detekci opakujícího se signálu se skládá z několika níže uvedených kroků a jeho základní myšlenka je postavena na výpočtu hodnoty autokorelace signálu.

Automatické odstranění stejnosměrné složky signálu (volitelně)
Pojmem stejnosměrná složka signálu označujeme složky signálu, které mají nulovou frekvenci (nebo velmi blízkou nule). Její odstranění může být vhodné v případě, že nás zajímá pouze dynamika chování systému, a ne hodnota jeho ustáleného stavu.

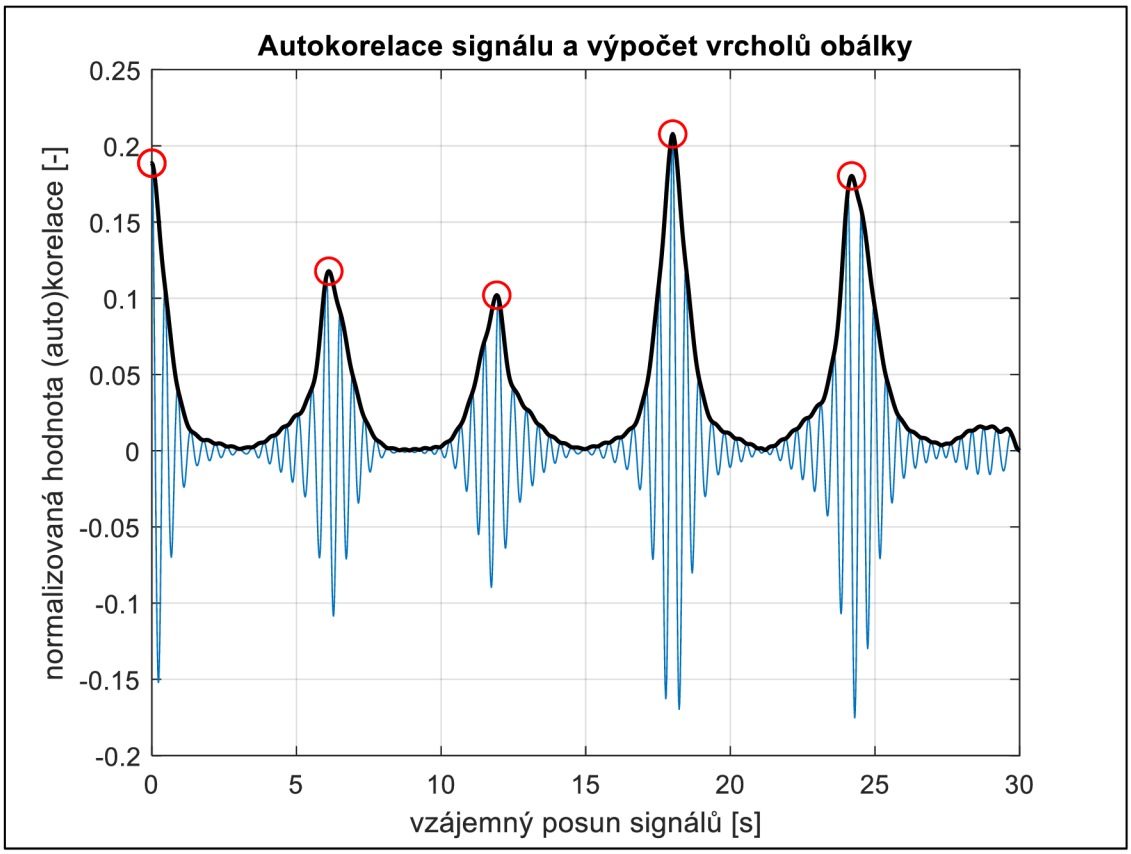
Abychom tuto složku signálu odstranili, využíváme pro zjednodušení specializovanou funkci (resp. objekt) v MATLABu s názvem `dsp.DCBlocker` z toolboxu DSP System Toolbox. Tato funkce automaticky odstraní ze vstupních dat stejnosměrnou složku signálu. Podobného efektu bychom mohli dosáhnout i použitím filtru typu horní propust.

Výpočet autokorelace signálu
Hlavní myšlenka detekce opakujícího se signálu je využití autokorelace tohoto signálu. Zjednodušeně korelace vyjadřuje míru podobnosti dvou signálů. V případě autokorelace je zjišťována podobnost signálu se sebou samým (v závislosti na vzájemném posunu). Pokud se ve výstupu autokorelační funkce objeví charakteristické špičky, může to ukazovat na přítomnost periodických sekcí (viz Obr. 57).

Pro výpočet autokorelace signálu můžeme opět použít vestavěnou funkci MATLABu s názvem `xcorr`, která v případně pouze jednoho vstupního argumentu spočítá jeho autokorelaci.

Výpočet obálky autokorelace a detekce špiček
Abychom mohli zmíněné špičky detekovat, je třeba nejprve vypočítat obálku autokorelace signálu a následně detekovat její špičky.

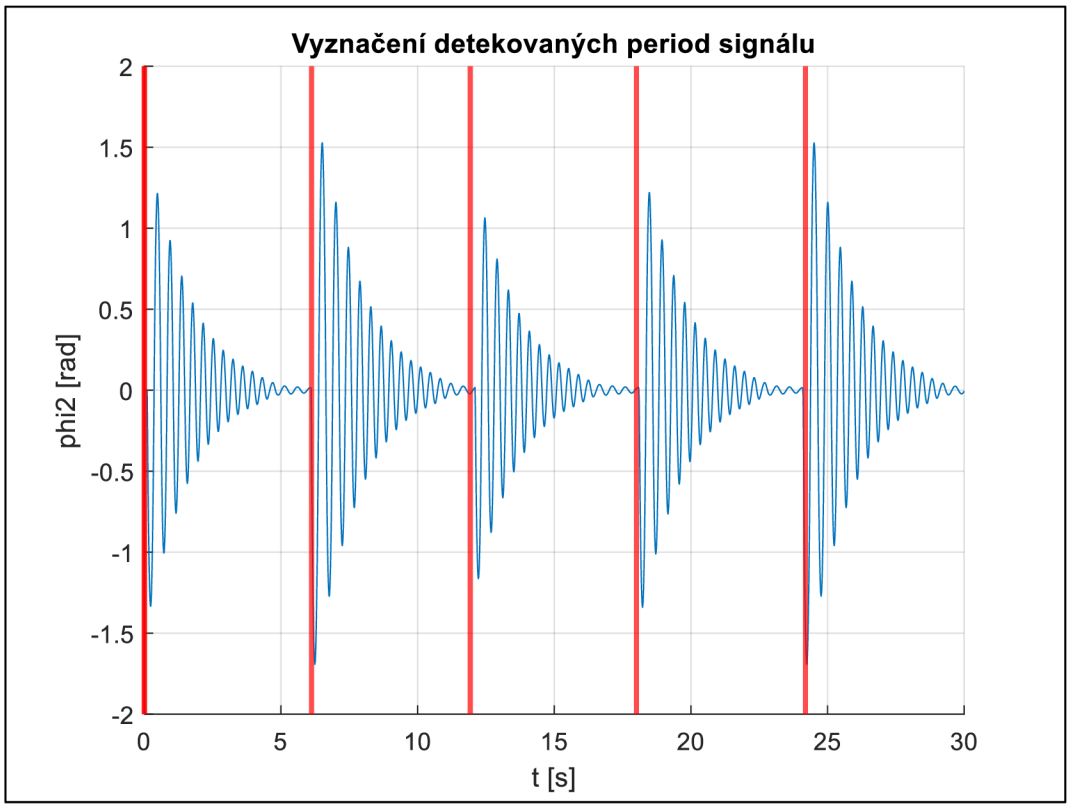
Pro výpočet obálky využijeme můžeme využít Hilbertovy transformace, pro jejíž výpočet použijeme MATLAB funkci `hilbert`. Pro následnou detekci špiček pak použijeme funkci `findpeaks`. Vstupem pro tuto funkci jsou data vypočítané obálky a dále nastavení parametru `MinPeakProminence`, který zjednodušeně vyjadřuje, jak moc musí detekovaný vrchol vyčnívat nad své okolí, aby byl zaregistrován. Tento parametr byl na základě testů s různými datovými sadami nastaven na hodnotu 1/3 výšky vrcholu při nulovém posunu signálů. Výsledek výše zmíněných operací je níže na Obr. 57.



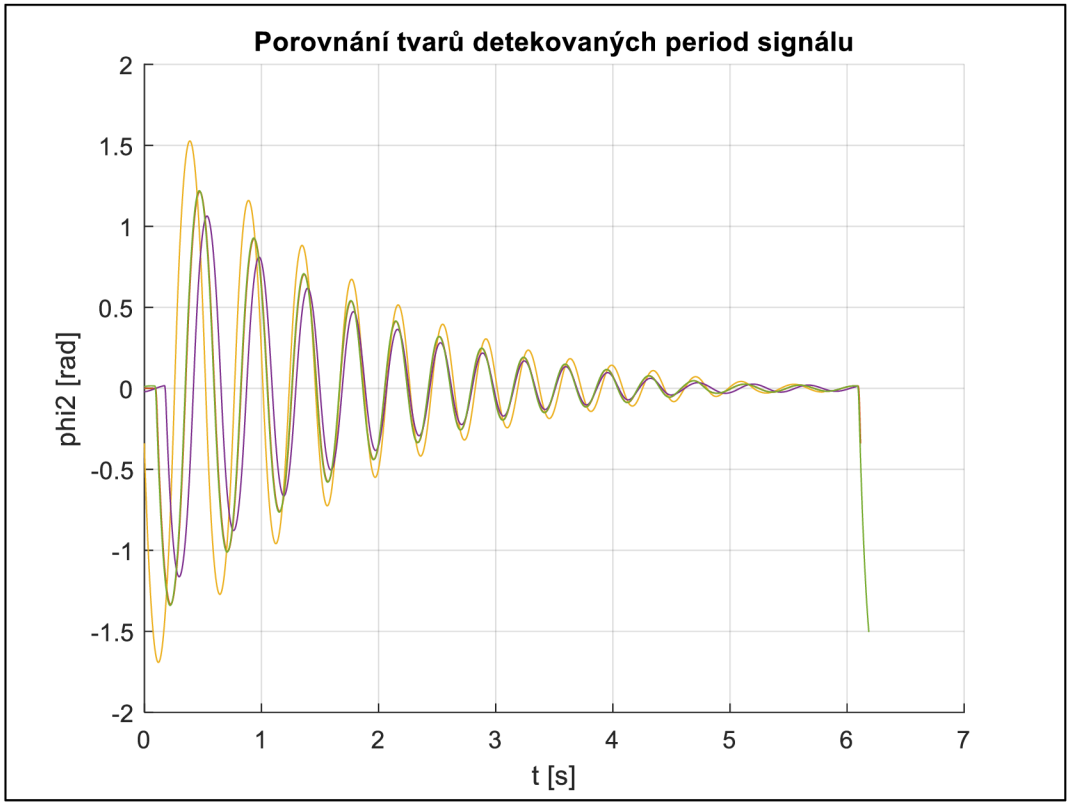
Obr. 57. Autokorelace signálu (modrá), obálka pomocí Hilbertovy transformace (černá), detekce vrcholů obálky (červená).

Vykreslení detekovaných opakujících se sekcí signálu

Jakmile máme detekované potenciální hranice opakujících se sekcí signálu v místě vrcholů autokorelace, můžeme tyto sekce vykreslit do původního signálu a separovat jednotlivé sekce pro jejich vzájemné porovnání (viz Obr. 58 a Obr. 59). Vzhledem k tomu, že algoritmus detekce není nikdy zcela dokonalý, je v konečném důsledku na samotném uživateli, aby předložené výsledky zhodnotil a případně provedl úpravu vstupních dat pro optimalizační algoritmus.

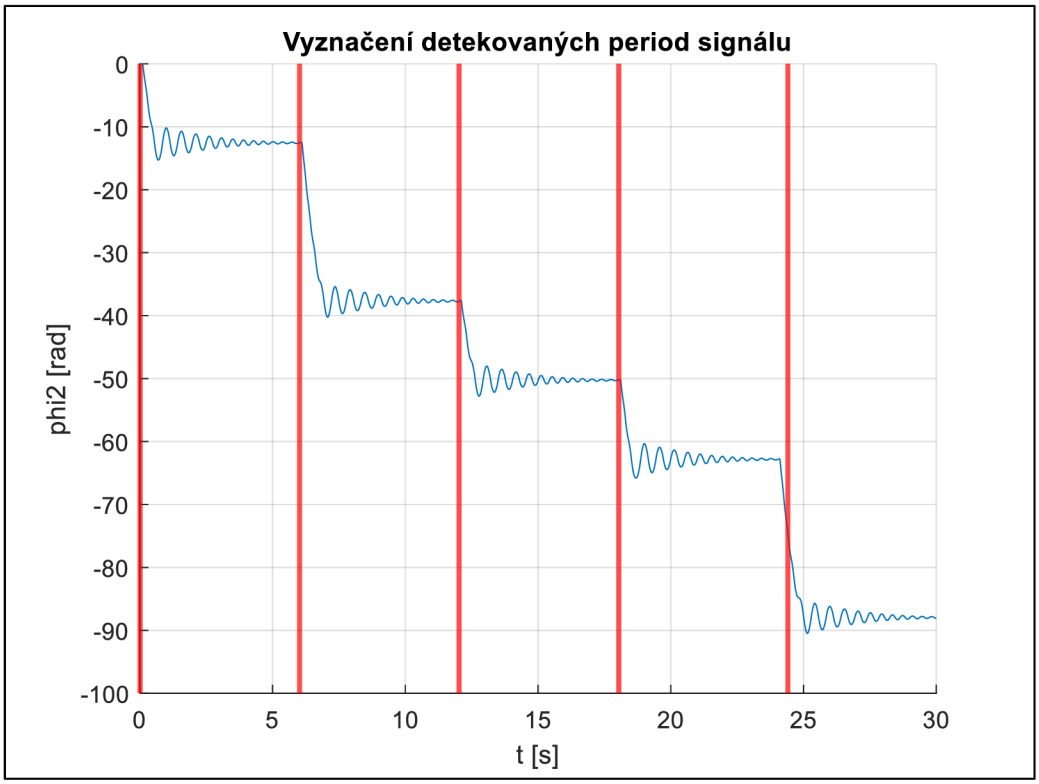


Obr. 58. Vyznačení vzájemně podobných sekcí v původním signálu.

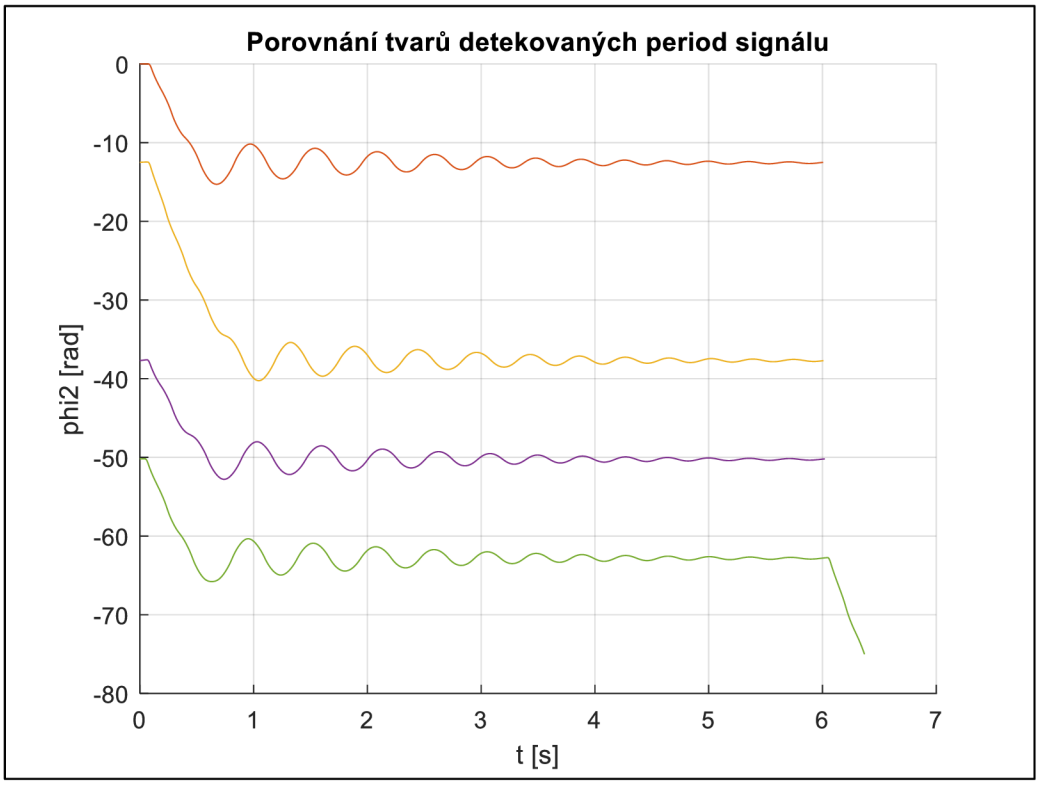


Obr. 59. Vykreslení detekovaných podobných sekcí signálu přes sebe.

Pro ilustraci také uvádíme výsledek algoritmu pro signál s nenulovou stejnosměrnou složkou, pokud provedeme její odstranění před vyhodnocením autokorelace. Viz níže, Obr. 60, Obr. 61.



Obr. 60. Ukázka funkce algoritmu na signálu s nenulovou střední hodnotou (automatické odstranění stejnosměrné složky dat).



Obr. 61. Ukázka funkce algoritmu na signálu s nenulovou střední hodnotou (automatické odstranění stejnosměrné složky dat).

Na závěr je nutné zmínit, že pro správnou funkci algoritmů uvedených v této kapitole je třeba, aby měl zkoumaný signál fixní vzorkovací periodu (resp. frekvenci). V případě, že máme signál s variabilním krokem, je nutné ho převzorkovat. Jednou z možností jak toto provést, je využít dalšího vytvořeného nástroje, který je popsán v následující kapitole (6.2.2) a provádí mimo jiné i převzorkování s fixním krokem.

Všechny výše uvedené kroky algoritmu byly také implementovány do podoby níže uvedené funkce:

```
MGO_sig_repeating_detector(signal,fs,DC_remove)
```

Funkce nemá v aktuální podobě žádný výstup, pouze zobrazí grafy uživateli. Jejími vstupy jsou zkoumaný signál (`signal`) v podobě vektoru datových bodů, vzorkovací frekvence `fs`, a parametr `DC_remove`, který určuje, zda má být provedeno odstranění stejnosměrné složky signálu.

6.2.2 Převzorkování dat

Dalším potenciálním problémem v oblasti podoby naměřených dat na reálném modelu, je jejich vzorkovací frekvence. Snadno se totiž může stát, že naměřená data budou vzorkována zbytečně vysokou frekvencí. Toto je opět nejčastěji v důsledku nezkušenosti uživatele, který pro jistotu volí co nejvyšší vzorkovací frekvenci, kterou mu měřicí hardware umožňuje.

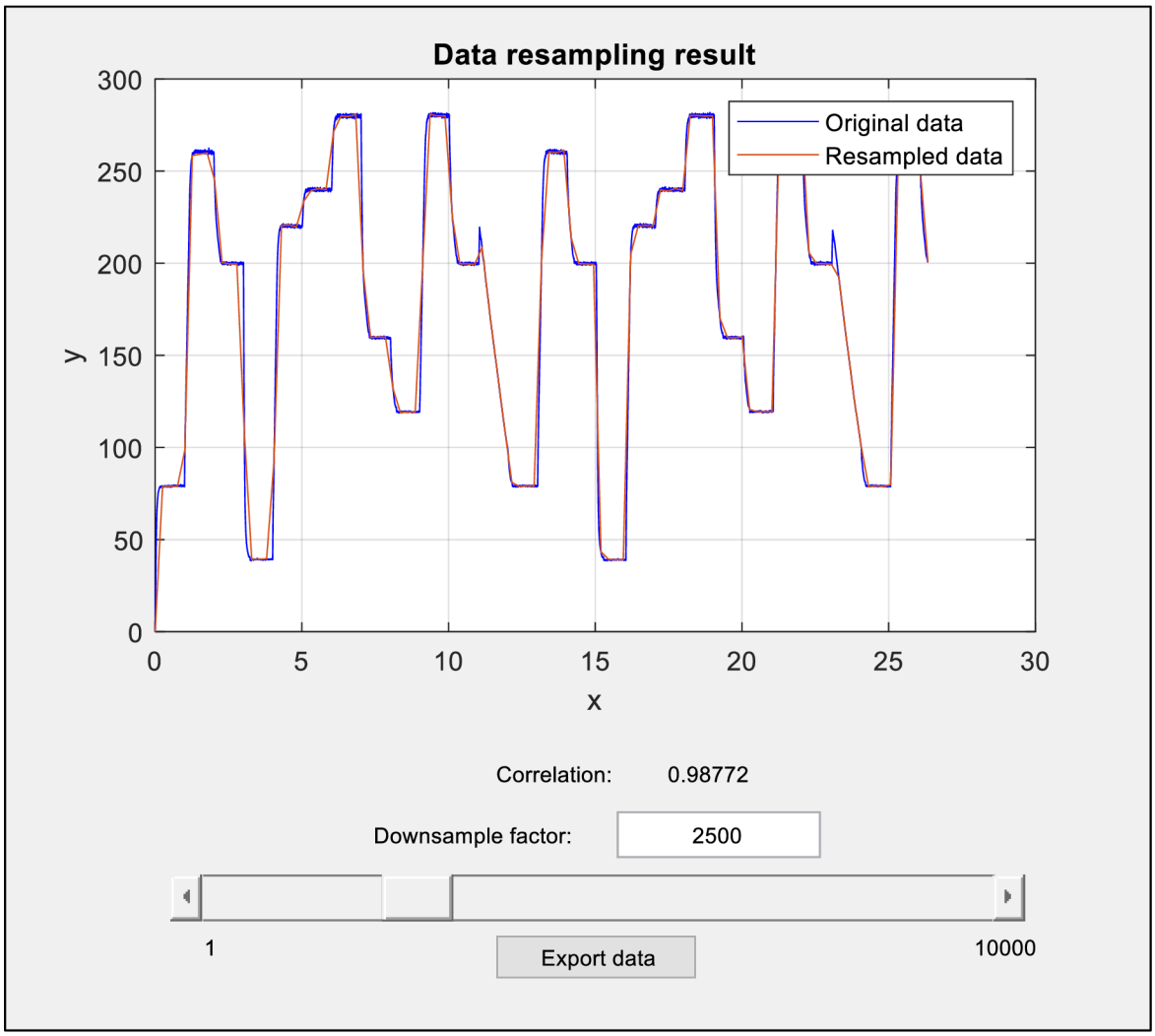
Vysoká vzorkovací frekvence samozřejmě sama o sobě problém nepředstavuje. Ten však může nastat ve chvíli, kdy bychom například pro jednoduchost zvolili krok řešiče simulovaného modelu tak, aby byl stejný jako vzorkovací perioda naměřených dat. Zbytečně malý krok řešiče simulace pak opět zbytečně zpomaluje celou optimalizační úlohu.

Z tohoto důvodu byl navržen interaktivní nástroj, který uživateli umožňuje měnit koeficient převzorkování, vždy na menší počet vzorků než původní signál. Ukázka grafického uživatelského rozhraní (GUI) tohoto nástroje je na Obr. 62.

Nástroj je spuštěn zavoláním funkce

```
MGO_data_downsample_GUI(x,y)
```

která nemá sama o sobě žádné výstupy, a jejími vstupy jsou parametry x a y, které představují vektory hodnot signálu y v závislosti x.



Obr. 62. Vytvořené GUI pro převzorkování dat.

Uživatel může koeficient převzorkování měnit manuálně pomocí posuvníků nebo přímým zadáním hodnoty do pole „Downsample factor“. Vizuálně je pak možné porovnat tvar původních a převzorkovaných dat pro aktuálně zvolený koeficient. Graf se automaticky aktualizuje při změně hodnoty.

Primárním vodítkem pro výběr vhodného koeficientu je samozřejmě zmíněná vizuální inspekce tvaru signálu tak, aby byl nalezen kompromis mezi zachováním tvaru signálu a vzorkovací periodou. Pro kvantifikaci míry podobnosti obou signálů je pak navíc vypočítána i hodnota koeficientu korelace obou signálů s využitím MATLAB funkce `corrcoef`.

S využitím hodnoty koeficientu korelace byl pak navržen i postup automatické volby koeficientu pro převzorkování. Hodnota koeficientu korelace 0,999 pak byla experimentálně určena jako vhodná výchozí hodnota, kdy je ještě zachována většina hlavních složek tvaru původního signálu. Automatický algoritmus pak během několika iterací nalezne takovou hodnotu koeficientu převzorkování, aby byla výsledná hodnota koeficientu korelace právě 0,999. Takto nalezený koeficient převzorkování je pak uživateli prezentován jako výchozí hodnota při spuštění nástroje.

Nástroj samozřejmě nabízí i export převzorkovaného signálu pomocí tlačítka „Export data“.

6.3 Vytvořené algoritmy a jejich použití v praxi

V kapitole 6 (Pomocné algoritmy pro úlohu odhadu parametrů) bylo navrženo a implementováno několik druhů algoritmů, které řeší otázku jednoznačné identifikovatelnosti hledaných parametrů modelu a formát signálů (naměřených dat) pro úlohu odhadu parametrů.

Obě výše uvedené problematiky vycházejí z praktických zkušeností, co se týče problémů, které nastávají při řešení úlohy odhadu parametrů, zejména pokud nemá uživatel dostatečné zkušenosti v této oblasti, a používá optimalizační nástroje bez dostatečného vhledu do problematiky.

Navržené algoritmy jsou v praxi použitelné i samostatně, nejvýhodnější je však jejich zakomponování do přímo do nástrojů pro odhad neznámých parametrů systémů.

7 Nový nástroj pro odhad parametrů v inženýrské praxi

V rámci řešení jednotlivých cílů dizertační práce, adresovaných v kapitolách 4, 5 a 6, vzniklo mnoho funkcí, které jsou samostatně použitelné a většinou na sobě nezávislé.

V budoucnu se předpokládá integrace těchto funkcí do ucelené podoby nového MATLAB toolboxu pro úlohu hledání parametrů. V současné chvíli jsou tyto funkce připraveny pro spuštění přímo z příkazové řádky, případně ze skriptu či jiné funkce.

Funkce vytvořené v této práci mají specifické pojmenování a jsou identifikovatelné dle předpon v jejich názvech:

- `MGO_`
  - Funkce týkající se nastavení optimalizační úlohy a nového hybridního algoritmu
  - Funkce pomocných algoritmů (detekce závislých parametrů, převzorkování dat)
- `SigGen_`
  - Funkce pro generování vstupních signálů systému
- `forSim`
  - Hlavní třída pro simulaci modelů

Praktický návod pro řešení úlohy a aplikace jednotlivých výsledků této práce s využitím námi vytvořených funkcí je bodově popsán níže:

1. Nastavení úlohy (model systému, neznámé parametry, naměřená data, nastavení úlohy)
2. Analýza vstupních dat
   a. Detekce neužitečného opakování signálu (viz kap. 6.2.1)
   b. Převzorkování naměřených dat (viz kap. 6.2.2)
3. Preventivní detekce závislých parametrů (viz kap. 6.1.2)
4. Spuštění optimalizační úlohy – Hybridní algoritmus (viz kap. 5.2)
5. Detekce korelovaných parametrů po ukončení optimalizační úlohy (viz kap. 6.1.1)

Vzhledem k návaznosti tématu této dizertační práce na dizertační práci Ing. Martina Appela – „Výzkum metod pro významné zrychlení odhadu parametrů simulačních modelů“, bude případná integrace námi vytvořených funkcí do podoby toolboxu provedena v návaznosti na jeho praktické výsledky. Práce Ing. Appela nebyla v době tvorby této dizertace ještě dokončena.

Bez ohledu výše uvedenou situaci lze však i nyní použít vyvinuté funkce a nástroje pro efektivnější řešení úlohy hledání parametrů.

8 Závěr

Motivací pro výzkum v této oblasti byly zejména zkušenosti z inženýrské praxe, spojené s úlohou odhadu parametrů mechatronického modelu.

Bylo zjištěno, že při praktické implementaci optimalizačních algoritmů na konkrétní třídě modelů, je výhodné znát její specifické vlastnosti. Tyto informace mohou totiž zásadním způsobem ovlivnit volbu vhodného postupu a nastavení algoritmů, což v důsledku vede ke zrychlení celého procesu estimace parametrů a zvýšení šance na nalezení globálního optima.

Pro tuto práci byly stanoveny čtyři hlavní cíle:

1) Vytvoření souboru technických modelů a jejich analýza
2) Kombinace globálních a lokálních optimalizačních metod pro mechatronické modely
3) Vytvoření algoritmů pro detekci a analýzu nevhodně navržených úloh odhadu parametrů
4) Nový nástroj pro odhad parametrů v inženýrské praxi

Přínosy disertační práce:

Vytvoření souboru technických modelů a jejich analýza

- Byla vytvořena sada mechatronických modelů v prostředí MATLAB Simulink s využitím principů fyzikálního modelování (Toolbox Simscape). Tyto modely a jejich parametry jsou odvozeny od reálných mechanismů a zařízení, která se v inženýrské praxi vyskytují.
- Námi vytvořená sada modelů má několik důležitých vlastností, díky kterým je možné ji využít i mimo téma této disertační práce:
    - Jde o konzistentní sadu modelů, které jsou zjednodušenými ale funkčními obrazy reálných strojů. Díky využití principů fyzikálního modelování jsou dostatečně přehledné a lze je jednoduše upravovat a rozšiřovat.
    - Všechny parametry modelů jsou známy. Vyskytují se navíc i korelované parametry, čehož bylo využito při řešení třetího cíle této práce.
    - Vhodné budicí signály byly připraveny pro všechny modely tak, aby se projevily všechny dynamické vlastnosti.
    - Byla provedena analýza vlastností modelů v souvislosti s úlohou odhadu parametrů. Ta zahrnuje detailní zmapování závislosti hodnotící funkce na zvolených parametrech. Z provedených analýz jsou patrné charakteristické vlastnosti prostoru parametrů pro tyto modely.
- Soubor lze využít pro výuku i další vědecké práce v oblasti identifikace systémů, řídicích algoritmů apod.

Kombinace globálních a lokálních optimalizačních metod pro mechatronické modely

- Prvním krokem v řešení tohoto cíle, bylo zmapování dostupných optimalizačních algoritmů nástroje MATLAB a jeho toolboxů. Byla provedena jejich analýza a možnosti

nastavení. Pro každý algoritmus bylo vytipováno několik parametrů, jejichž vliv na úspěšnost řešení byl zkoumán v následujících krocích.

- Následně byl proveden praktický srovnávací test optimalizačních algoritmů zahrnující:
  - 10 fyzikálních modelů (viz cíl 1).
  - 7 kombinací hledaných parametrů pro každý model (max. 4 hledané parametry).
  - 11 optimalizačních algoritmů, které byly vybrány na základě analýzy z předchozího kroku.
  - 2 varianty nastavení každého algoritmu.
  - 2 druhy vstupních signálů do modelů (buzení).
  - 4 varianty vychýlení počátečního odhadu hledaných parametrů.
  - 2 varianty nastavení hranic prohledávaného prostoru parametrů.
- Celkem bylo tedy testováno 2464 variant optimalizační úlohy pro každý z 10 modelů. Výsledky tohoto srovnávacího testu byly následně zpracovány do souhrnného přehledu, přičemž hlavními sledovanými indikátory byla rychlost konvergence algoritmu a také odchylka nalezeného řešení (hodnoty hledaných parametrů) od skutečnosti.
- V návaznosti na výsledky benchmarku byly identifikovány algoritmy s nejlepšími výsledky a byla navržena a otestována nová varianta hybridního optimalizačního algoritmu s robustním a automatizovaným prohledáváním prostoru parametrů na globální úrovni v kombinaci s lokální optimalizací využívající metody přímého prohledávání.
- Následně byl nový algoritmus porovnán s jinými globálními optimalizačními metodami. Při výsledném srovnání jsou jasně patrné výhody nového algoritmu:
  - Pravděpodobnost nalezení globálního minima je stejná nebo lepší.
  - Počet potřebných simulací (výpočtů hodnoticí funkce) je výrazně menší.
  - Algoritmus umožňuje omezenou i neomezenou dobu výpočtu, stejně jako manuální přerušení uživatelem.
  - Algoritmus je připraven na případné rozšíření o volitelné kroky či použití jiné lokální prohledávací metody.

Vytvoření algoritmů pro detekci a analýzu nevhodně navržených úloh odhadu parametrů

- V rámci analýzy řešených úloh jako celku byly navrženy algoritmy, umožňující odhalit některé skryté problémy, které mohou bránit úspěšnému nalezení optima:
  - Detekce redundantních parametrů upozorní na případy, kdy jsou parametry v rovnicích vzájemně provázané a nelze je od sebe separovat (viz náš článek [73]). V těchto případech je pak vhodné parametry sloučit a řádově tak snížit obtížnost úlohy.
  - Byl vyvinut algoritmus pro analýzu vstupních (měřených) dat od uživatele, který díky detekci signálů s nulovou či periodickou složkou, případně s nevhodným vzorkováním, umožní zkrácení doby simulace modelu a tím i celého optimalizačního algoritmu. Algoritmus byl otestován na reálných datech a s relativně vysokou úspěšností detekuje periodické signály, a to i s nenulovou střední odchylkou. Vždy je ale nutná kontrola uživatele, proto má funkce spíše informativní charakter.

Nový nástroj pro odhad parametrů v inženýrské praxi

- Výše uvedené výsledky byly prakticky implementovány v podobě sady funkcí a skriptů v programu MATLAB. Tyto funkce jsou určeny jak pro použití koncovým uživatelem (např. v rámci výuky), tak i jako výchozí bod pro další výzkum a vývoj v této oblasti.
- Vytvořené skripty tedy nejsou pouze oddělenými, samostatně funkčními bloky, ale umožňují propojení do celku – nového nástroje pro odhad parametrů. Vzhledem k návaznosti na zatím nedokončenou disertační práci Ing. Appela, nebyl zatím plánovaný nový toolboxu finalizován. Bez ohledu na to lze však už nyní použít vyvinuté funkce a nástroje pro efektivnější řešení úlohy hledání parametrů.

Závěrem lze konstatovat, že všechny cíle vytyčené cíle se podařilo splnit, přičemž v rámci této dizertační práce vznikla nová sada mechatronických modelů, nový hybridní optimalizační algoritmus a sada pomocných algoritmů pro úlohu odhadu parametrů.

Další předpokládaný vývoj v této oblasti spočívá v transformaci praktických výsledků této práce do podoby nového toolboxu pro odhad parametrů v programu MATLAB, rozšíření hybridního algoritmu o další funkcionalitu, a rozšíření sady pomocných algoritmů pro detekci nevhodného nastavení úlohy (významnost parametrů, interaktivní asistent pro nastavení optimalizační úlohy apod.).

9 Seznam vlastních publikací

CELIK, M.; HANCIOGLU, O.; BOGOSYAN, S.; BASTL, M.; NAJMAN, J.; GREPL, R. Cascade Control of SATCOM on the Move (SOTM) Antennas with Jacobian Operator. In *2019 IEEE 7th International Conference on Control, Mechatronics and Automation ICCMA 2019*. Delft,Nizozemsko: IEEE, 2020. s. 416-422. ISBN: 978-1-7281-3787-2.

SPÁČIL, T.; RAJCHL, M.; BASTL, M.; NAJMAN, J.; APPEL, M. Design of deterministic model for compensation of acceleration sensitivity in MEMS gyroscope. In *Advances in Intelligent Systems and Computing - Mechatronics 2019: Recent Advances Towards Industry 4.0. Advances in Intelligent Systems and Computing*. 1. Warsaw: Springer Verlag, 2019. s. 285-291. ISBN: 9783030299927. ISSN: 2194-5357.

BASTL, M.; NAJMAN, J.; SPÁČIL, T. Design of an antenna pedestal stabilization controller based on cascade topology. In *Advances in Intelligent Systems and Computing - Mechatronics 2019: Recent Advances Towards Industry 4.0. Advances in Intelligent Systems and Computing*. 1. Warsaw: Springer Verlag, 2019. s. 469-475. ISBN: 9783030299927. ISSN: 2194-5357.

NAJMAN, J.; BASTL, M.; APPEL, M.; GREPL, R. Computationally Fast Dynamical Model of a SATCOM Antenna Suitable for Extensive Optimization Tasks. *Advances in Military Technology*, 2019, roč. 14, č. 1, s. 21-30. ISSN: 1802-2308.

NAJMAN, J.; BRABLC, M.; RAJCHL, M.; BASTL, M.; SPÁČIL, T.; APPEL, M. Monte carlo based detection of parameter correlation in simulation models. In *Advances in Intelligent Systems and Computing - Mechatronics 2019: Recent Advances Towards Industry 4.0. Advances in Intelligent Systems and Computing*. 1. Warsaw: Springer Verlag, 2019. s. 54-61. ISBN: 9783030299927. ISSN: 2194-5357.

10 Seznam zdrojů a použité literatury

[1] *Simscape - MATLAB & Simulink* [online]. [vid. 2022-06-16]. Dostupné z: https://www.mathworks.com/products/simscape.html#mulsch

[2] KISABO, Aliyu B, C A OSHEKU, Adetoro M A LANRE a Aliyu FUNMILAYO. Ordinary Differential Equations: MATLAB/Simulink ® Solutions. *International Journal of Scientific & Engineering Research* [online]. 2012, **3**(8) [vid. 2022-06-17]. ISSN 2229-5518. Dostupné z: http://www.ijser.org

[3] ISERMANN, R., J. SCHAFFNIT a S. SINSEL. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice* [online]. 1999, **7**(5), 643–653. ISSN 0967-0661. Dostupné z: doi:10.1016/S0967-0661(98)00205-6

[4] SHANNON, Claude E. Communication in the Presence of Noise. *Proceedings of the IRE* [online]. 1949, **37**(1), 10–21. ISSN 00968390. Dostupné z: doi:10.1109/JRPROC.1949.232969

[5] *DataTechNotes: Regression Model Accuracy (MAE, MSE, RMSE, R-squared) Check in R* [online]. [vid. 2022-06-18]. Dostupné z: https://www.datatechnotes.com/2019/02/regression-model-accuracy-mae-mse-rmse.html

[6] CHAI, T. a R. R. DRAXLER. Root mean square error (RMSE) or mean absolute error (MAE)? -Arguments against avoiding RMSE in the literature. *Geoscientific Model Development* [online]. 2014, **7**(3), 1247–1250. ISSN 19919603. Dostupné z: doi:10.5194/GMD-7-1247-2014

[7] CHICCO, Davide, Matthijs J. WARRENS a Giuseppe JURMAN. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Computer Science* [online]. 2021, **7**, 1–24. ISSN 23765992. Dostupné z: doi:10.7717/PEERJ-CS.623/SUPP-1

[8] NGUYEN, Anh Tuan, Sigrid REITER a Philippe RIGO. *A review on simulation-based optimization methods applied to building performance analysis* [online]. B.m.: Elsevier. 1. leden 2014 [vid. 2018-11-15]. ISBN 0306-2619. Dostupné z: doi:10.1016/j.apenergy.2013.08.061

[9] FIGUEIRA, Gonçalo a Bernardo ALMADA-LOBO. Hybrid simulation-optimization methods: A taxonomy and discussion [online]. 2014 [vid. 2018-11-15]. Dostupné z: doi:10.1016/j.simpat.2014.03.007

[10] DRÉO, Johann, Alain PÉTROWSKI, Patrick SIARRY a Eric TAILLARD. *Metaheuristics for Hard Optimization* [online]. Berlin/Heidelberg: Springer-Verlag, 2006. ISBN 3-540-23022-X. Dostupné z: doi:10.1007/3-540-30966-7

[11] JANGA REDDY, M. a D. NAGESH KUMAR. Evolutionary algorithms, swarm intelligence methods, and their applications in water resources engineering: A state-of-the-art review. *H2Open Journal* [online]. 2020, **3**(1), 135–188. ISSN 26166518. Dostupné z: doi:10.2166/H2OJ.2020.128/699810/H2OJ2020128.PDF

[12] *What are gradient descent and stochastic gradient descent?* [online]. [vid. 2022-06-18]. Dostupné z: https://sebastianraschka.com/faq/docs/gradient-optimization.html

[13] LEWIS, Robert Michael, Virginia TORCZON a Michael W. TROSSET. Direct search methods: then and now. *Journal of Computational and Applied Mathematics* [online]. 2000, **124**(1–2), 191–207. ISSN 0377-0427. Dostupné z: doi:10.1016/S0377-0427(00)00423-4

[14] KOLDA, Tamara G., Robert Michael LEWIS a Virginia TORCZON. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review* [online]. 2003, **45**(3), 385–482. ISSN 00361445. Dostupné z: doi:10.1137/S003614450242889

[15] *Metaheuristic Optimization - Scholarpedia* [online]. [vid. 2022-06-18]. Dostupné z: http://www.scholarpedia.org/article/Metaheuristic_Optimization

[16] HAN, Zhong-Hua a Ke-Shi ZHANG. Surrogate-Based Optimization. *Real-World Applications of Genetic Algorithms* [online]. 2012. Dostupné z: doi:10.5772/36125

[17] *Find minimum of unconstrained multivariable function - MATLAB fminunc* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/optim/ug/fminunc.html

[18] SHANNO, D. F. Conditioning of Quasi-Newton Methods for Function Minimization. *Mathematics of Computation* [online]. 1970, **24**(111), 647. ISSN 00255718. Dostupné z: doi:10.2307/2004840

[19] COLEMAN, Thomas F a Yuying LI. An Interior Trust Region Approach for Nonlinear Minimization Subject to Bounds. *SIAM Journal on Optimization* [online]. 1996, **6**(2), 418–445. Dostupné z: doi:10.1137/0806023

[20] *Find minimum of constrained nonlinear multivariable function - MATLAB fmincon* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/optim/ug/fmincon.html

[21] BYRD, Richard H., Jean Charles GILBERT a Jorge NOCEDAL. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming 2000 89:1* [online]. 2000, **89**(1), 149–185 [vid. 2022-06-19]. ISSN 1436-4646. Dostupné z: doi:10.1007/PL00011391

[22] Sequential Quadratic Programming. In: *Numerical Optimization* [online]. New York, NY: Springer New York, 2006, s. 529–562. ISBN 978-0-387-40065-5. Dostupné z: doi:10.1007/978-0-387-40065-5_18

[23] GILL, Philip E, Walter MURRAY a Margaret H WRIGHT. Practical optimization. In: . 1981.

[24] *Choosing the Algorithm - MATLAB & Simulink* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/optim/ug/choosing-the-algorithm.html

[25] *Solve nonlinear least-squares (nonlinear data-fitting) problems - MATLAB lsqnonlin* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/optim/ug/lsqnonlin.html

[26] *Least-Squares (Model Fitting) Algorithms - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/optim/ug/least-squares-model-fitting-algorithms.html#f204

[27] LEVENBERG, Kenneth. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics* [online]. 1944, **2**(2), 164–168.

ISSN 0033-569X. Dostupné z: doi:10.1090/QAM/10666

[28] MARQUARDT, Donald W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics* [online]. 1963, **11**(2), 431–441. ISSN 0368-4245. Dostupné z: doi:10.1137/0111030

[29] MORÉ, Jorge J. The Levenberg-Marquardt algorithm: Implementation and theory BT - Numerical Analysis. *Numerical Analysis* [online]. 1978, **630**(Chapter 10), 105–116 [vid. 2022-05-13]. Dostupné z: http://www.springerlink.com/index/10.1007/BFb0067700

[30] *Find minimum of unconstrained multivariable function using derivative-free method - MATLAB fminsearch* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/matlab/ref/fminsearch.html

[31] LAGARIAS, Jeffrey C., James A. REEDS, Margaret H. WRIGHT a Paul E. WRIGHT. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization* [online]. 1998, **9**(1), 112–147. ISSN 10526234. Dostupné z: doi:10.1137/S1052623496303470

[32] *Optimizing Nonlinear Functions - MATLAB & Simulink* [online]. [vid. 2022-05-13]. Dostupné z: https://www.mathworks.com/help/matlab/math/optimizing-nonlinear-functions.html#bsgpq6p-11

[33] *Simplex - Wikipedia* [online]. [vid. 2022-06-20]. Dostupné z: https://en.wikipedia.org/wiki/Simplex

[34] *Numerical Nonlinear Global Optimization—Wolfram Language Documentation* [online]. [vid. 2022-06-20]. Dostupné z: https://reference.wolfram.com/language/tutorial/ConstrainedOptimizationGlobalNumerical.html

[35] *Find minimum of function using pattern search - MATLAB patternsearch* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/gads/patternsearch.html

[36] TORCZON, Virginia. On the Convergence of Pattern Search Algorithms. *SIAM Journal on Optimization* [online]. 1997, **7**(1), 1–25. Dostupné z: doi:10.1137/S1052623493250780

[37] *Searching and Polling - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/searching-and-polling.html

[38] *Pattern Search Terminology - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/pattern-search-terminology.html#f7816

[39] *How Pattern Search Polling Works - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/how-pattern-search-polling-works.html

[40] *Pattern Search Options - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/pattern-search-options.html

[41] *Find minimum of function using simulated annealing algorithm - MATLAB simulannealbnd* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/gads/simulannealbnd.html

[42] KIRKPATRICK, S., C. D. GELATT a M. P. VECCHI. Optimization by Simulated Annealing [online]. 1986, 339–348. Dostupné z: doi:10.1142/9789812799371_0035

[43] *How Simulated Annealing Works - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/how-simulated-annealing-works.html

[44] *Simulated Annealing Terminology - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/understanding-simulated-annealing-terminology.html

[45] INGBER, Lester. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics* [online]. 2000, **25**(1), 32–54 [vid. 2022-05-13]. ISSN 03248569. Dostupné z: doi:10.48550/arxiv.cs/0001018

[46] LEDESMA, Sergio, Jose RUIZ a Guadalupe GARCIA. Simulated Annealing Evolution. In: Marcos DE SALES GUERRA TSUZUKI, ed. *Simulated Annealing* [online]. Rijeka: IntechOpen, 2012. Dostupné z: doi:10.5772/50176

[47] *Find minimum of function using genetic algorithm - MATLAB ga* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/gads/ga.html

[48] GOLDBERG, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st vyd. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

[49] *How the Genetic Algorithm Works - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html

[50] *Genetic Algorithm Terminology - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/some-genetic-algorithm-terminology.html

[51] *Genetic Algorithm Options - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/genetic-algorithm-options.html

[52] *What Is the Genetic Algorithm? - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

[53] *Surrogate Optimization Algorithm - MATLAB & Simulink* [online]. [vid. 2022-05-13]. Dostupné z: https://www.mathworks.com/help/gads/surrogate-optimization-algorithm.html

[54] *Surrogate Optimization Algorithm - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/surrogate-optimization-algorithm.html

[55] REGIS, Rommel G. a Christine A. SHOEMAKER. A Stochastic Radial Basis Function Method for the Global Optimization of Expensive Functions. *https://doi.org/10.1287/ijoc.1060.0182* [online]. 2007, **19**(4), 497–509 [vid. 2022-06-20]. ISSN 15265528. Dostupné z: doi:10.1287/IJOC.1060.0182

[56] WANG, Yilun a Christine A. SHOEMAKER. A General Stochastic Algorithmic Framework for Minimizing Expensive Black Box Objective Functions Based on Surrogate Models and Sensitivity Analysis [online]. 2014 [vid. 2022-06-20]. Dostupné z: http://arxiv.org/abs/1410.6271

[57] *Particle swarm optimization - MATLAB particleswarm* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/gads/particleswarm.html

[58] KENNEDY, J a R EBERHART. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks* [online]. 1995, s. 1942–1948 roč.4. Dostupné z: doi:10.1109/ICNN.1995.488968

[59] *Particle Swarm Optimization Algorithm - MATLAB & Simulink* [online]. [vid. 2022-06-20]. Dostupné z: https://www.mathworks.com/help/gads/particle-swarm-optimization-algorithm.html

[60] MEZURA-MONTES, Efrén a Carlos A COELLO COELLO. Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation* [online]. 2011, **1**(4), 173–194. ISSN 2210-6502. Dostupné z: doi:https://doi.org/10.1016/j.swevo.2011.10.001

[61] PEDERSEN, M E H. Good Parameters for Particle Swarm Optimization. In: . 2010.

[62] *Optimalizace hejnem částic – Wikipedie* [online]. [vid. 2022-06-20]. Dostupné z: https://cs.wikipedia.org/wiki/Optimalizace_hejnem_částic

[63] CHEN, Zhanying, Xuekun LI, Zongyu ZHU, Zeming ZHAO, Liping WANG, Sheng JIANG a Yiming RONG. The optimization of accuracy and efficiency for multistage precision grinding process with an improved particle swarm optimization algorithm. *International Journal of Advanced Robotic Systems* [online]. 2020, **17**(1). ISSN 17298814. Dostupné z: doi:10.1177/1729881419893508

[64] *Find global minimum - MATLAB* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/gads/globalsearch.html

[65] UGRAY, Zsolt, Leon LASDON, John PLUMMER, Fred GLOVER, James KELLY a Rafael MARTI. Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *https://doi.org/10.1287/ijoc.1060.0175* [online]. 2007, **19**(3), 328–340 [vid. 2022-06-21]. ISSN 15265528. Dostupné z: doi:10.1287/IJOC.1060.0175

[66] *How GlobalSearch and MultiStart Work - MATLAB & Simulink* [online]. [vid. 2022-06-21]. Dostupné z: https://www.mathworks.com/help/gads/how-globalsearch-and-multistart-work.html

[67] *Find multiple local minima - MATLAB* [online]. [vid. 2022-06-19]. Dostupné z: https://www.mathworks.com/help/gads/multistart.html?s_tid=doc_ta

[68] *Interface between electrical and mechanical rotational domains - MATLAB* [online]. [vid. 2022-06-23]. Dostupné z: https://www.mathworks.com/help/physmod/simscape/ref/rotationalelectromechanicalconverter.html

[69] MAKKAR, C., W. E. DIXON, W. G. SAWYER a G. HU. A new continuously differentiable friction model for control systems design. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM* [online]. 2005, **1**, 600–605. Dostupné z: doi:10.1109/AIM.2005.1511048

[70] *Apply contact forces between a pair of connected bodies - MATLAB* [online]. [vid. 2022-06-23]. Dostupné z: https://www.mathworks.com/help/releases/R2020a/physmod/sm/ref/spatialcontactforce.html

[71] *Frictional brake with pressure-applying cylinder and pads with faulting - MATLAB* [online]. [vid. 2022-06-23]. Dostupné z: https://www.mathworks.com/help/physmod/sdl/ref/discbrake.html

[72] *DC motor model with electrical and torque characteristics and fault modeling - MATLAB* [online]. [vid. 2022-06-23]. Dostupné z: https://www.mathworks.com/help/physmod/sps/ref/dcmotor.html

[73] NAJMAN, Jan, Martin BRABLC, Matej RAJCHL, Michal BASTL, Tomáš SPÁČIL a Martin APPEL. Monte Carlo Based Detection of Parameter Correlation in Simulation Models. *Advances in Intelligent Systems and Computing* [online]. 2020, **1044**, 54–61 [vid. 2022-05-06]. ISSN 21945365. Dostupné z: doi:10.1007/978-3-030-29993-4_7

[74] LEWIS, Robert Michael (College of William and Mary, Williamsburg, VA), Virginia Joanne (College of William and Mary, Williamsburg, VA) TORCZON a Tamara Gibson KOLDA. A generating set direct search augmented Lagrangian algorithm for optimization with a combination of general and linear constraints. [online]. 2006 [vid. 2022-06-20]. Dostupné z: doi:10.2172/893121

[75] OLSSON, H., K. J. ÅSTRÖM, C. CANUDAS DE WIT, M. GÄFVERT a P. LISCHINSKY. Friction Models and Friction Compensation. *European Journal of Control* [online]. 1998, **4**(3), 176–195. ISSN 0947-3580. Dostupné z: doi:10.1016/S0947-3580(98)70113-X

```
%% fminunc
problem.objective = @MyforSim.step; %must be set for 'fminunc'
problem.x0 = MyforSim.ParametersVar_InitVals; %must be set for 'fminunc'
problem.solver = 'fminunc'; %must be set for 'fminunc'

% A variant:
problem.options = optimoptions(@fminunc,'Display','iter-detailed',...
        'Algorithm','quasi-newton',...
        'FiniteDifferenceType','central',...
        'TypicalX',ones(length(MyforSim.ParametersVar_InitVals),1),...
        'MaxFunctionEvaluations',400,...
        'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimoptions(@fminunc,'Display','iter-detailed',...
        'Algorithm','quasi-newton',...
        'FiniteDifferenceType','forward',...
        'TypicalX',ones(length(MyforSim.ParametersVar_InitVals),1),...
        'MaxFunctionEvaluations',400,...
        'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 63. Nastavení optimalizačního algoritmu fminunc (první srovnávací test algoritmů)

```
%% fmincon
problem.objective = @MyforSim.step; %must be set for 'fmincon'
problem.x0 = MyforSim.ParametersVar_InitVals; %must be set for 'fmincon'
problem.solver = 'fmincon'; %must be set for 'fmincon'
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

% A variant:
problem.options = optimoptions(@fmincon,'Display','iter',...
        'Algorithm','active-set',...
        'MaxFunctionEvaluations',400,...
        'FiniteDifferenceType','central',...
        'TypicalX',ones(length(MyforSim.ParametersVar_InitVals),1),...
        'ScaleProblem',false,...
        'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimoptions(@fmincon,'Display','iter',...
        'Algorithm','sqp',...
        'MaxFunctionEvaluations',400,...
        'FiniteDifferenceType','central',...
        'TypicalX',ones(length(MyforSim.ParametersVar_InitVals),1),...
        'ScaleProblem',true,...
        'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 64. Nastavení optimalizačního algoritmu fmincon (první srovnávací test algoritmů)

```
%% lsqnonlin
problem.objective = @MyforSim.step; %must be set for 'lsqnonlin'
problem.x0 = MyforSim.ParametersVar_InitVals; %must be set for 'lsqnonlin'
problem.solver = 'lsqnonlin'; %must be set for 'lsqnonlin'
problem.lb=[]; %Levenberg-Marquardt nepodporuje limity!
problem.ub=[]; %Levenberg-Marquardt nepodporuje limity!

% A variant:
problem.options = optimoptions(@lsqnonlin,'Display','iter-detailed',...
        'Algorithm','levenberg-marquardt',...
        'FiniteDifferenceType','forward',...
        'TypicalX',ones(length(MyforSim.ParametersVar_InitVals),1),...
        'ScaleProblem','none',...
        'MaxFunctionEvaluations',400,...
        'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimoptions(@lsqnonlin,'Display','iter-detailed',...
        'Algorithm','levenberg-marquardt',...
        'FiniteDifferenceType','central',...
        'TypicalX',ones(length(MyforSim.ParametersVar_InitVals),1),...
        'ScaleProblem','none',...
        'MaxFunctionEvaluations',400,...
        'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 65. Nastavení optimalizačního algoritmu lsqnonlin (první srovnávací test algoritmů)

```
%% fminsearch
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
problem.solver = 'fminsearch';

% A variant:
problem.options = optimset('Display','iter',...
        'MaxFunEvals',350,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimset('Display','iter',...
        'MaxFunEvals',500,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 66. Nastavení optimalizačního algoritmu fminsearch (první srovnávací test algoritmů)

```
%% patternsearch
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
problem.solver = 'patternsearch';

[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb; %must be set for 'patternsearch'
problem.ub=ub; %must be set for 'patternsearch'
problem.Aineq=[]; %must be set for 'patternsearch'
problem.bineq=[]; %must be set for 'patternsearch'
problem.Aeq=[]; %must be set for 'patternsearch'
```

```
problem.beq=[]; %must be set for 'patternsearch'
problem.nonlcon=[]; %must be set for 'patternsearch'

% A variant:
problem.options = optimoptions(@patternsearch,'Display','iter',...
        'MaxTime',Inf,...
        'MaxFunctionEvaluations',400,...
        'MaxIterations',Inf,...
        'PollMethod','GPSPositiveBasis2N',...
        'SearchFcn',[],...
        'PollOrderAlgorithm','Consecutive',...
        'InitialMeshSize',1,...
        'MeshContractionFactor',0.5,...
        'MeshExpansionFactor',2,...
        'ScaleMesh',false, ...
        'UseParallel',false,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimoptions(@patternsearch,'Display','iter',...
        'MaxTime',Inf,...
        'MaxFunctionEvaluations',400,...
        'MaxIterations',Inf,...
        'PollMethod','GPSPositiveBasis2N',...
        'SearchFcn',[],...
        'PollOrderAlgorithm','Success',...
        'InitialMeshSize',1,...
        'MeshContractionFactor',0.5,...
        'MeshExpansionFactor',2,...
        'ScaleMesh',true, ...
        'UseParallel',false,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 67. Nastavení optimalizačního algoritmu patternsearch (první srovnávací test algoritmů)

```
%% simulannealbnd
rng default % For reproducibility
problem.solver='simulannealbnd';
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

% A variant:
problem.options = optimoptions(@simulannealbnd,...
        'Display','iter','MaxTime',Inf,'MaxIterations',Inf,...
        'MaxFunctionEvaluations',400,...
        'InitialTemperature',1000,...
        'ReannealInterval',20,...
        'TemperatureFcn','temperaturefast',...
        'AnnealingFcn','annealingfast',...
        'HybridFcn',[],...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
```

```
problem.options = optimoptions(@simulannealbnd,...
        'Display','iter','MaxTime',Inf,'MaxIterations',Inf,...
        'MaxFunctionEvaluations',400,...
        'InitialTemperature',100,...
        'ReannealInterval',50,...
        'TemperatureFcn','temperatureexp',...
        'AnnealingFcn','annealingfast',...
        'HybridFcn',[],...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 68. Nastavení optimalizačního algoritmu simulannealbnd (první srovnávací test algoritmů)

```
%% ga
rng default % For reproducibility
problem.solver='ga';
problem.fitnessfcn = @MyforSim.step;
problem.nvars=length(MyforSim.ParametersVar_InitVals);
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

% A variant:
problem.options = optimoptions(@ga,...
        'Display','iter','MaxTime',180,'MaxGenerations',100,...
        'UseParallel',false,...
        'PopulationSize',50,...
        'CrossoverFraction',0.8,...
        'HybridFcn',[],...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimoptions(@ga,...
        'Display','iter','MaxTime',180,'MaxGenerations',100,...
        'UseParallel',false,...
        'PopulationSize',20,...
        'CrossoverFraction',0.8,...
        'HybridFcn',[],...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 69. Nastavení optimalizačního algoritmu ga (první srovnávací test algoritmů)

```
%% surrogateopt
rng default % For reproducibility
problem.solver='surrogateopt';
problem.objective = @MyforSim.step;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
lb(lb == -Inf)=-1e6; %limity nesmi byt +/- Inf
ub(ub == Inf)=1e6; %limity nesmi byt +/- Inf
problem.lb=lb;
problem.ub=ub;

% A variant:
problem.options = optimoptions('surrogateopt',...
        'MaxFunctionEvaluations',400,...
```

```
        'MaxTime',Inf,...
        'MinSampleDistance',1e-3,...
        'MinSurrogatePoints',max(20,2*length(MyforSim.ParametersVar_InitVals)),...
        'OutputFcn',@MGO_optsolver_outfcn_universal,...
        'PlotFcn',[],...
        'Display','iter',...
        'UseParallel',false);

% B variant:
problem.options = optimoptions('surrogateopt',...
        'MaxFunctionEvaluations',300,...
        'MaxTime',Inf,...
        'MinSampleDistance',1e-4,...
        'MinSurrogatePoints',max(20,2*length(MyforSim.ParametersVar_InitVals)),...
        'OutputFcn',@MGO_optsolver_outfcn_universal,...
        'PlotFcn',[],...
        'Display','iter',...
        'UseParallel',true);
```

Obr. 70. Nastavení optimalizačního algoritmu surrogateopt (první srovnávací test algoritmů)

```
%% particleswarm
rng default % For reproducibility
problem.solver='particleswarm';
problem.objective = @MyforSim.step;
problem.nvars=length(MyforSim.ParametersVar_InitVals);
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

% A variant:
problem.options = optimoptions(@particleswarm,...
        'Display','iter',...
        'MaxTime',180,...
        'MaxIterations',Inf,...
        'UseParallel',false,...
        'SwarmSize',min(100,10*problem.nvars),...
        'InertiaRange',[0.1,1.1],...
        'InitialSwarmSpan',2000,...
        'MinNeighborsFraction',0.25,...
        'HybridFcn',[],...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
problem.options = optimoptions(@particleswarm,...
        'Display','iter',...
        'MaxTime',180,...
        'MaxIterations',Inf,...
        'UseParallel',false,...
        'SwarmSize',min(20,10*problem.nvars),...
        'InertiaRange',[0.1,1.1],...
        'InitialSwarmSpan',10000,...
        'MinNeighborsFraction',0.25,...
        'HybridFcn',[],...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 71. Nastavení optimalizačního algoritmu particleswarm (první srovnávací test algoritmů)

```
%% GlobalSearch
rng default % For reproducibility
gs = GlobalSearch;
gs.Display='iter';
gs.MaxTime=180;
gs.OutputFcn=@MGO_optsolver_outfcn_GSMS;

problem = createOptimProblem('fmincon'); %must always be 'fmincon' pro Global
search
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

% A variant:
gs.NumTrialPoints=50;
        gs.NumStageOnePoints=10;
        gs.StartPointsToRun='bounds';
        gs.DistanceThresholdFactor=0.75;
        gs.MaxWaitCycle=5;
        gs.BasinRadiusFactor=0.2;
        gs.PenaltyThresholdFactor=0.2;
        problem.options = optimoptions(@fmincon,'Display','final',...
        'Algorithm','active-set',...
        'MaxFunctionEvaluations',35,'UseParallel',false,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
gs.NumTrialPoints=50;
        gs.NumStageOnePoints=10;
        gs.StartPointsToRun='bounds';
        gs.DistanceThresholdFactor=0.75;
        gs.MaxWaitCycle=5;
        gs.BasinRadiusFactor=0.2;
        gs.PenaltyThresholdFactor=0.2;
        problem.options = optimoptions(@fmincon,'Display','final',...
        'Algorithm','active-set',...
        'MaxFunctionEvaluations',20,'UseParallel',false,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 72. Nastavení optimalizačního algoritmu GlobalSearch (první srovnávací test algoritmů)

```
%% MultiStart
rng default
ms = MultiStart;
ms.Display='iter';
ms.MaxTime=180;
ms.OutputFcn=@MGO_optsolver_outfcn_GSMS;
ms.UseParallel=false;

problem = createOptimProblem('fmincon');
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
```

```
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

% A variant:
ms.StartPointsToRun='bounds';
MS_StartPoints_num = 20;
problem.options = optimoptions(@fmincon,'Display','iter',...
        'Algorithm','active-set',...
        'MaxFunctionEvaluations',30,'UseParallel',false,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);

% B variant:
ms.StartPointsToRun='bounds';
MS_StartPoints_num = 16;
problem.options = optimoptions(@fmincon,'Display','iter',...
        'Algorithm','active-set',...
        'MaxFunctionEvaluations',35,'UseParallel',false,...
        'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 73. Nastavení optimalizačního algoritmu MultiStart (první srovnávací test algoritmů)

```
%% hybrid patternsearch
rng default
hs.x0 = MyforSim.ParametersVar_InitVals;
hs.BestFval = Inf; %initial value
hs.BestX = NaN(size(hs.x0')); %initial value
hs.Status = "Hybrid search initialized"; %initial value
hs.MaxTime = 1800*length(hs.x0);
hs.FvalTolerance = 1e-3; %initial value
hs.ContinueForever = false;
hs.StartPoints.NumPts = 4^length(hs.x0);
hs.StartPoints.PtsDistribution = 'grid';
hs.StartPoints.DistributionSigma = 1/16;
hs.NewPoints.NumPts = 1; % must be set to 1
hs.NewPoints.PtsDistribution = 'normal';
hs.NewPoints.DistributionSigma = 1;
hs.BoundsGrowCoeff = 1;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
[lb,ub] = MGO_mySolver_bounds(length(hs.x0),lb,ub,hs.BoundsGrowCoeff);
hs.lb=lb;
hs.ub=ub;

problem.objective = @MyforSim.step;
problem.x0 = hs.x0; % only for initial settings
problem.solver = 'patternsearch';
problem.lb=hs.lb;
problem.ub=hs.ub;
problem.Aineq=[];
problem.bineq=[];
problem.Aeq=[];
problem.beq=[];
problem.nonlcon=[];

problem.options = optimoptions(@patternsearch,'Display','iter',...
    'MaxTime',Inf,...
    'MaxFunctionEvaluations',700*length(problem.x0),...
    'MaxIterations',Inf,...
    'PollMethod','GPSPositiveBasis2N',...
    'SearchFcn',[],...
    'PollOrderAlgorithm','Success',...
    'InitialMeshSize',1,...
    'MeshContractionFactor',0.5,...
    'MeshExpansionFactor',2,...
    'ScaleMesh',true, ...
    'UseParallel',false,...
    'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 74. Nastavení optimalizačního algoritmu Hybrid-patternsearch (druhý srovnávací test – nové algoritmy)

```
%% hybrid fminsearch
rng default
hs.x0 = MyforSim.ParametersVar_InitVals;
hs.BestFval = Inf; %initial value
hs.BestX = NaN(size(hs.x0')); %initial value
hs.Status = "Hybrid search initialized"; %initial value
hs.MaxTime = 1800*length(hs.x0);
hs.FvalTolerance = 1e-3; %initial value
hs.ContinueForever = false;
hs.StartPoints.NumPts = 4^length(hs.x0);
hs.StartPoints.PtsDistribution = 'grid';
hs.StartPoints.DistributionSigma = 1/16;
hs.NewPoints.NumPts = 1; % must be set to 1
hs.NewPoints.PtsDistribution = 'normal';
hs.NewPoints.DistributionSigma = 1;
hs.BoundsGrowCoeff = 1;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
[lb,ub] = MGO_mySolver_bounds(length(hs.x0),lb,ub,hs.BoundsGrowCoeff);
hs.lb=lb;
hs.ub=ub;

problem.objective = @MyforSim.step;
problem.x0 = hs.x0; % only for initial settings
problem.solver = 'fminsearch';

problem.options = optimset('Display','iter',...
    'MaxFunEvals',700*length(problem.x0),...
    'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 75. Nastavení optimalizačního algoritmu Hybrid-fminsearch (druhý srovnávací test – nové algoritmy)

```
%% hybrid fmincon
rng default
hs.x0 = MyforSim.ParametersVar_InitVals;
hs.BestFval = Inf; %initial value
hs.BestX = NaN(size(hs.x0')); %initial value
hs.Status = "Hybrid search initialized"; %initial value
hs.MaxTime = 1800*length(hs.x0);
hs.FvalTolerance = 1e-3; %initial value
hs.ContinueForever = false;
hs.StartPoints.NumPts = 4^length(hs.x0);
hs.StartPoints.PtsDistribution = 'grid';
hs.StartPoints.DistributionSigma = 1/16;
hs.NewPoints.NumPts = 1; % must be set to 1
hs.NewPoints.PtsDistribution = 'normal';
hs.NewPoints.DistributionSigma = 1;
hs.BoundsGrowCoeff = 1;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
[lb,ub] = MGO_mySolver_bounds(length(hs.x0),lb,ub,hs.BoundsGrowCoeff);
hs.lb=lb;
hs.ub=ub;

problem.objective = @MyforSim.step;
```

```
problem.x0 = hs.x0; % only for initial settings
problem.solver = 'fmincon';
problem.lb=hs.lb;
problem.ub=hs.ub;

problem.options = optimoptions(@fmincon,'Display','iter',...
    'Algorithm','sqp',...
    'MaxFunctionEvaluations',700*length(problem.x0),...
    'FiniteDifferenceType','central',...
    'TypicalX',ones(length(problem.x0),1),...
    'ScaleProblem',true,...
    'UseParallel',false,'OutputFcn',@MGO_optsolver_outfcn_universal);
```

Obr. 76. Nastavení optimalizačního algoritmu Hybrid-fmincon (druhý srovnávací test – nové algoritmy)

```
%% GlobalSearch
rng default % For reproducibility
gs = GlobalSearch;
gs.Display='iter';
gs.OutputFcn=@MGO_optsolver_outfcn_GSMS_v2;

problem = createOptimProblem('fmincon'); %must always be 'fmincon' pro Global
search
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

gs.MaxTime=180;
gs.NumTrialPoints=500*length(problem.x0);
gs.NumStageOnePoints=4^length(problem.x0);
gs.StartPointsToRun='bounds';
gs.DistanceThresholdFactor=0.75;
gs.MaxWaitCycle=20;
gs.BasinRadiusFactor=0.2;
gs.PenaltyThresholdFactor=0.2;
problem.options = optimoptions(@fmincon,'Display','final',...
    'Algorithm','active-set',...
    'MaxFunctionEvaluations',700*length(problem.x0),...
    'UseParallel',false,...
    'OutputFcn',@MGO_optsolver_outfcn_universal);
```

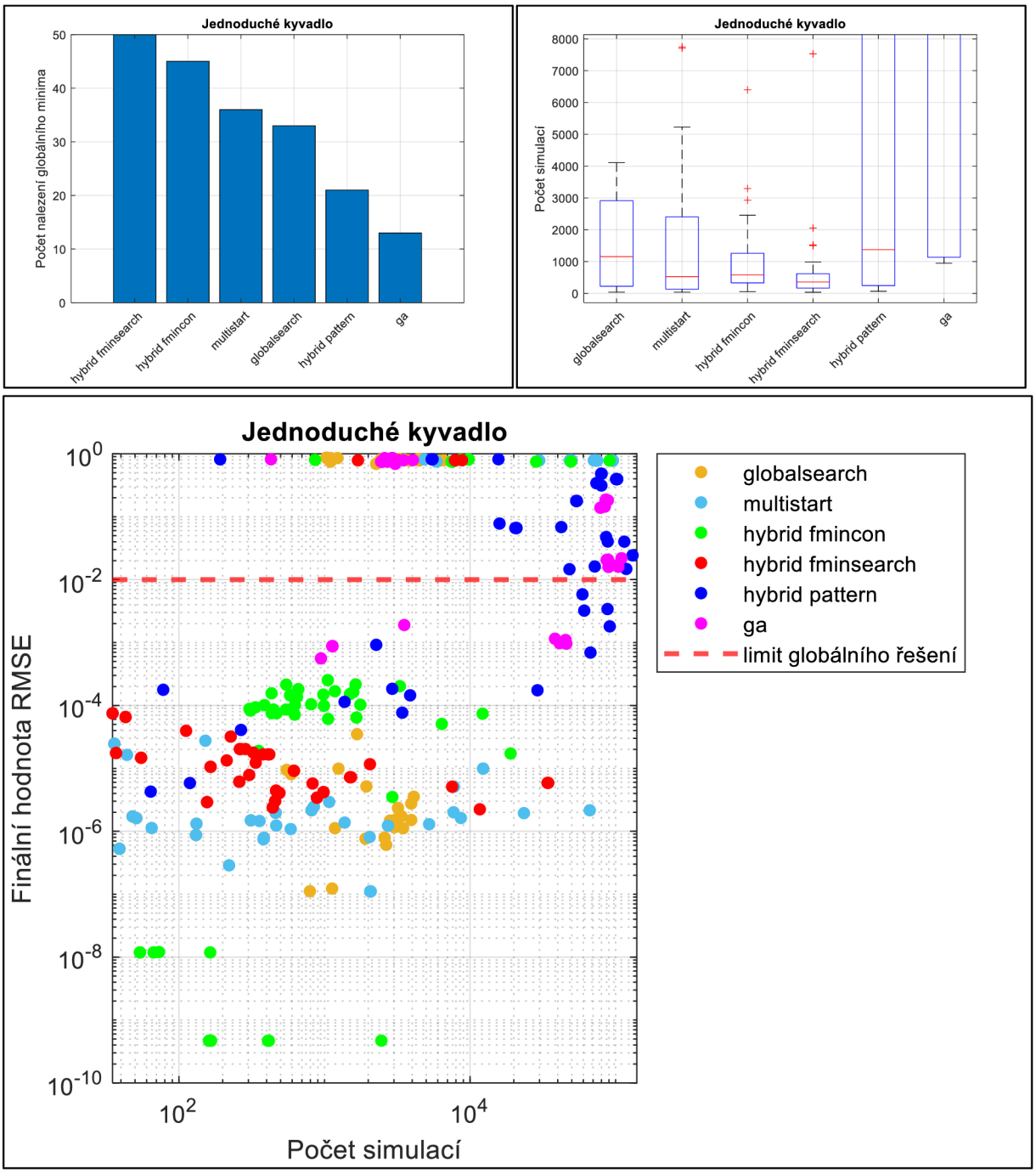
Obr. 77. Nastavení optimalizačního algoritmu GlobalSearch (druhý srovnávací test – nové algoritmy)

```
%% MultiStart
rng default
ms = MultiStart;
ms.Display='iter';
ms.OutputFcn=@MGO_optsolver_outfcn_GSMS_v2;
ms.UseParallel=false;

problem = createOptimProblem('fmincon');
problem.objective = @MyforSim.step;
problem.x0 = MyforSim.ParametersVar_InitVals;
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

ms.MaxTime=1800*length(problem.x0);
ms.StartPointsToRun='bounds';
MS_StartPoints_num = 10000;
problem.options = optimoptions(@fmincon,'Display','iter',...
    'Algorithm','active-set',...
    'MaxFunctionEvaluations',700*length(problem.x0),...
    'UseParallel',false,...
    'OutputFcn',@MGO_optsolver_outfcn_universal);
```

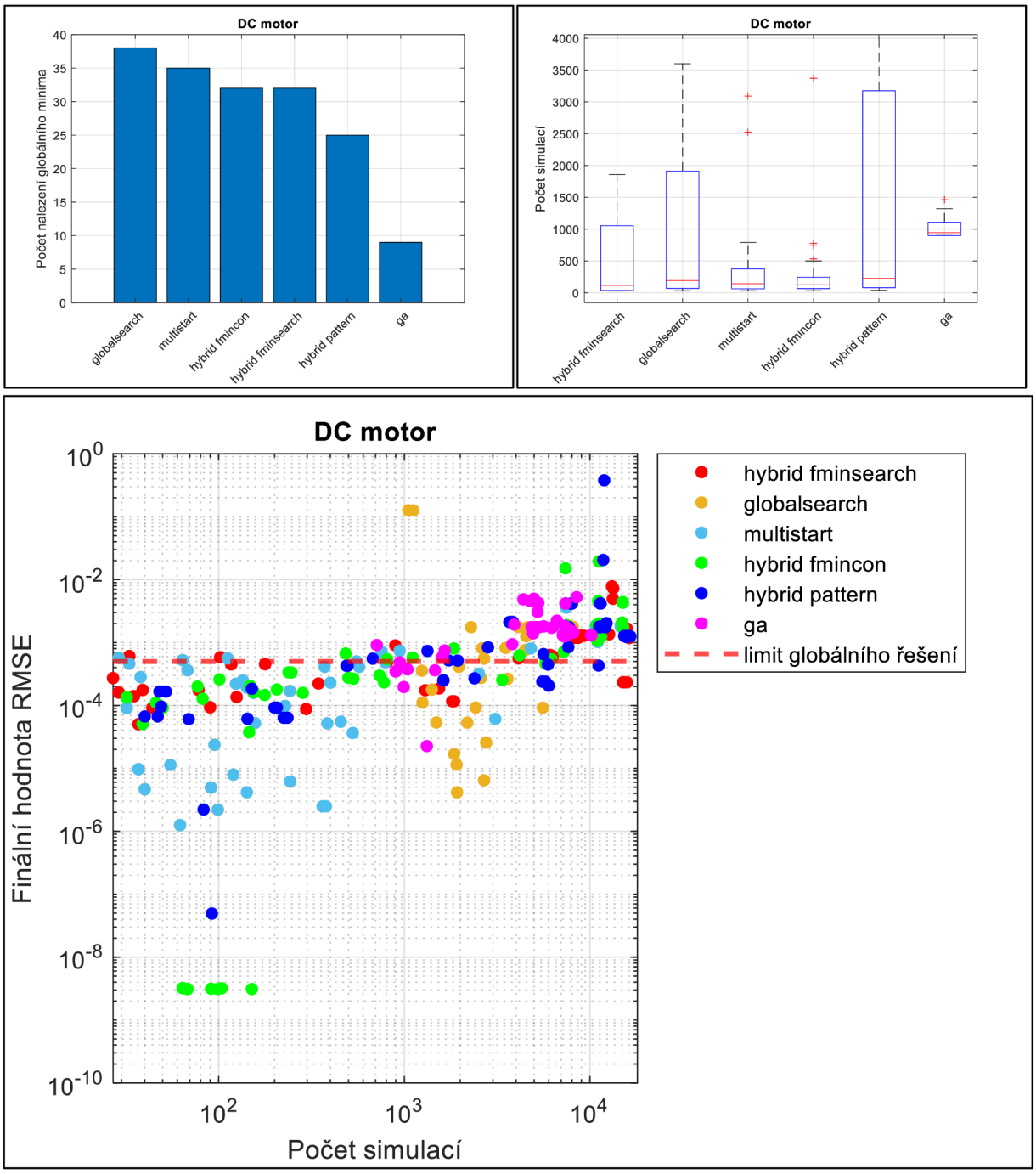
Obr. 78. Nastavení optimalizačního algoritmu MultiStart (druhý srovnávací test – nové algoritmy)

```
%% ga
rng default % For reproducibility
problem.solver='ga';
problem.fitnessfcn = @MyforSim.step;
problem.nvars=length(MyforSim.ParametersVar_InitVals);
[lb,ub] =
MGO_optsolver_bounds(MyforSim.ParametersVar_InitVals,limits_params(1),limits_param
s(2));
problem.lb=lb;
problem.ub=ub;

problem.options = optimoptions(@ga,...
    'Display','iter','MaxTime',1800*length(MyforSim.ParametersVar_InitVals),...
    'MaxGenerations',10000,...
    'UseParallel',false,...
    'PopulationSize',50,...
    'CrossoverFraction',0.8,...
    'HybridFcn',[],...
    'OutputFcn',@MGO_optsolver_outfcn_universal_v2);
```

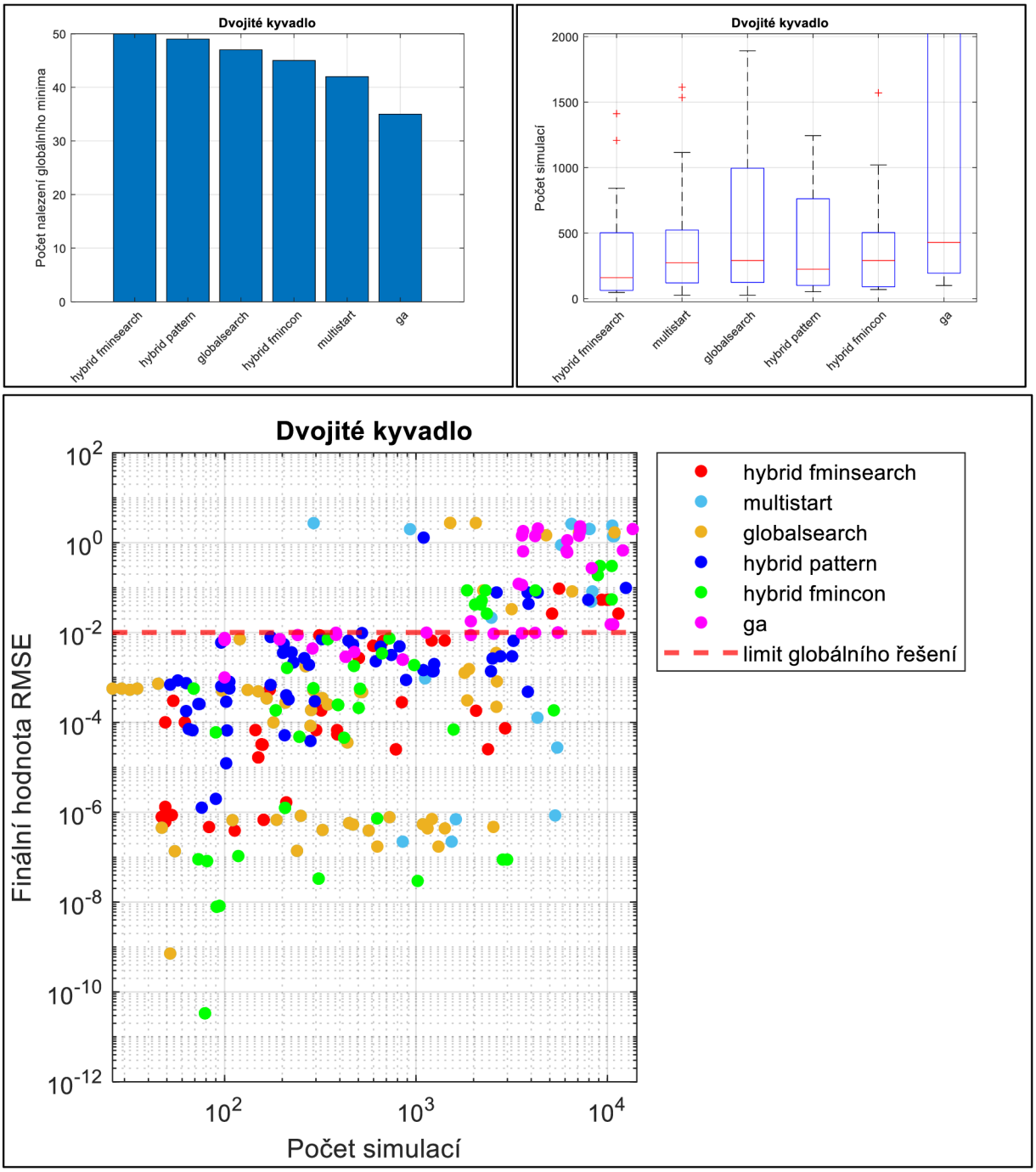
Obr. 79. Nastavení optimalizačního algoritmu ga (druhý srovnávací test – nové algoritmy)



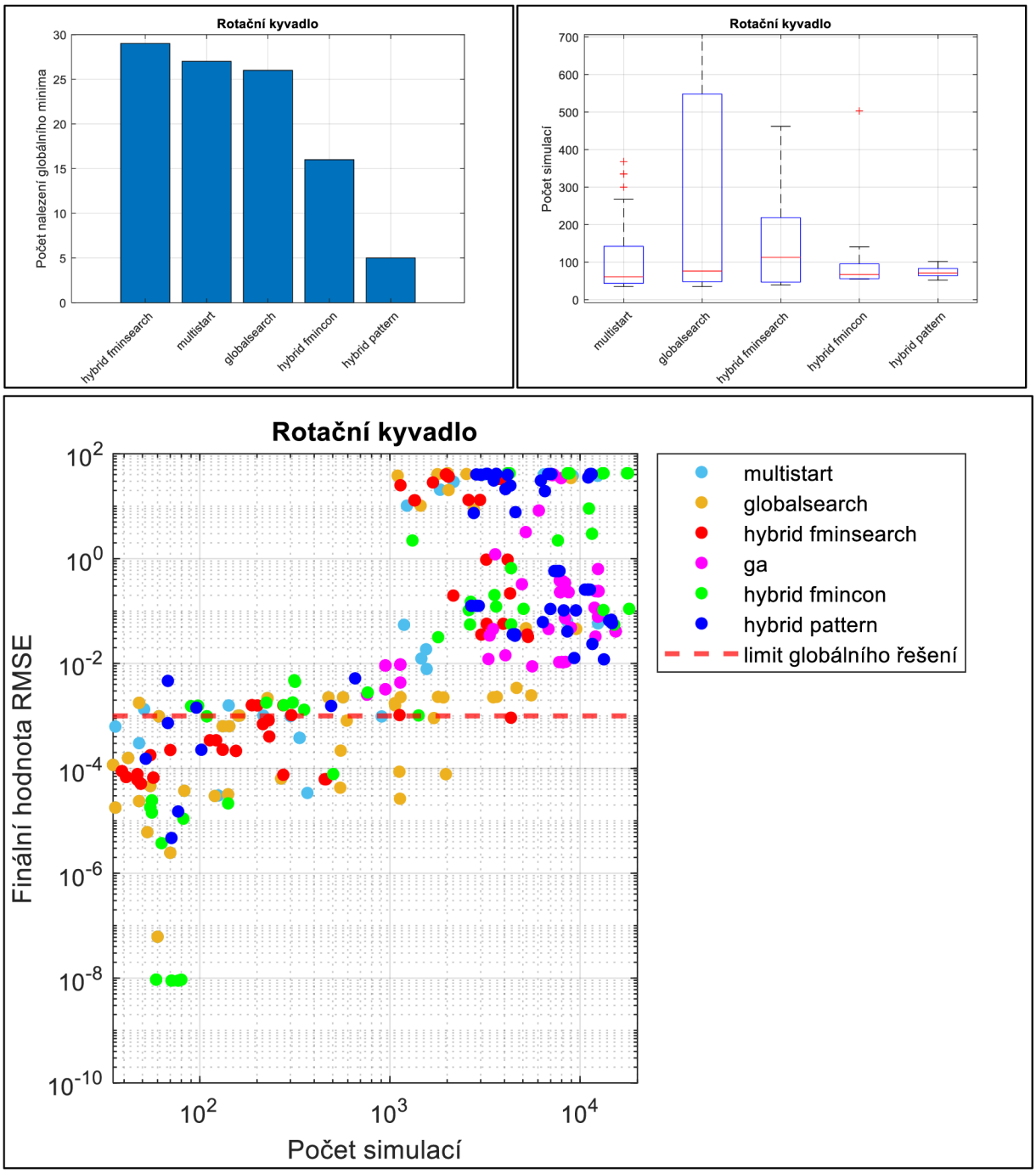
Obr. 80. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Jednoduché kyvadlo).



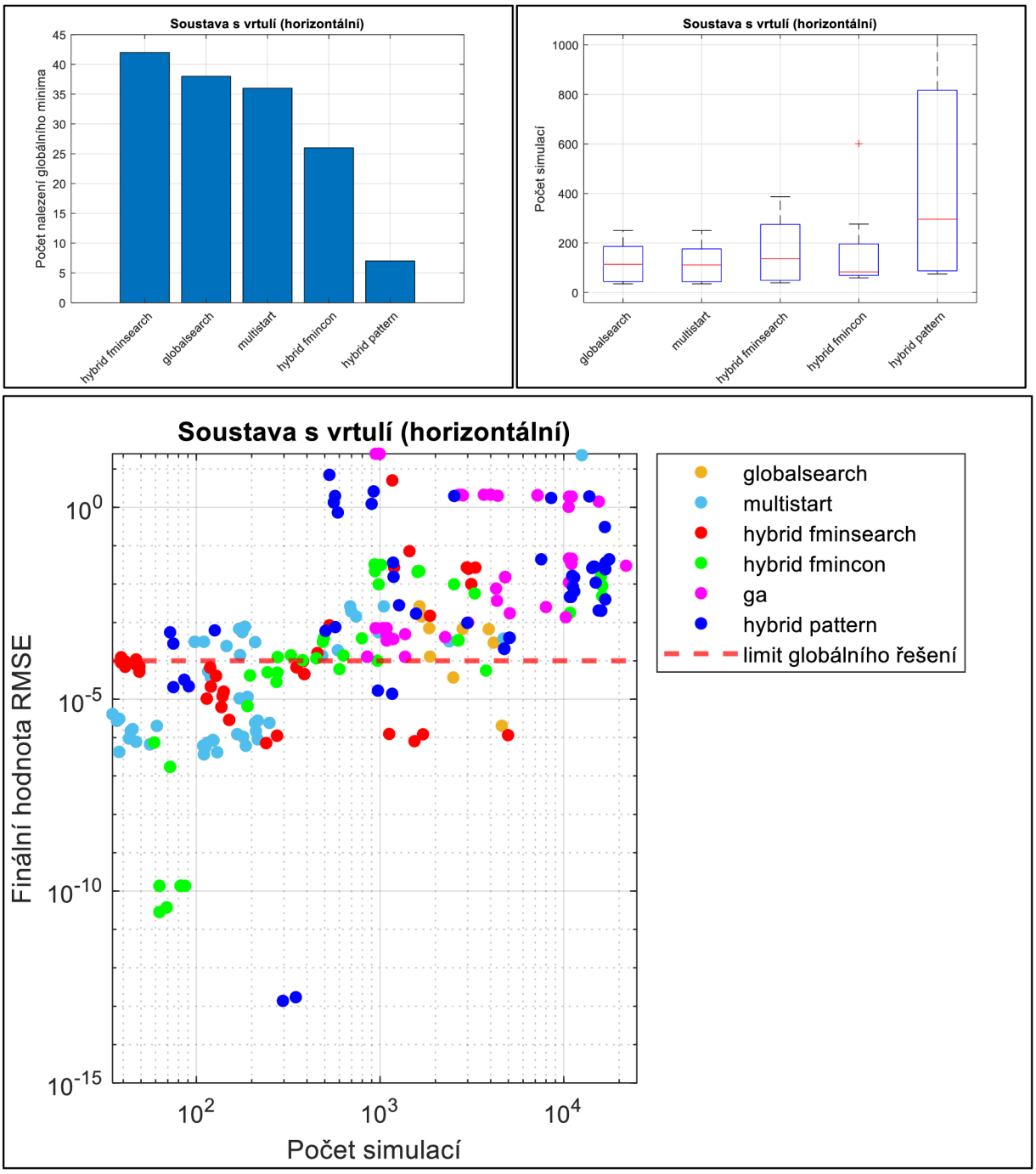
Obr. 81. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (DC motor).



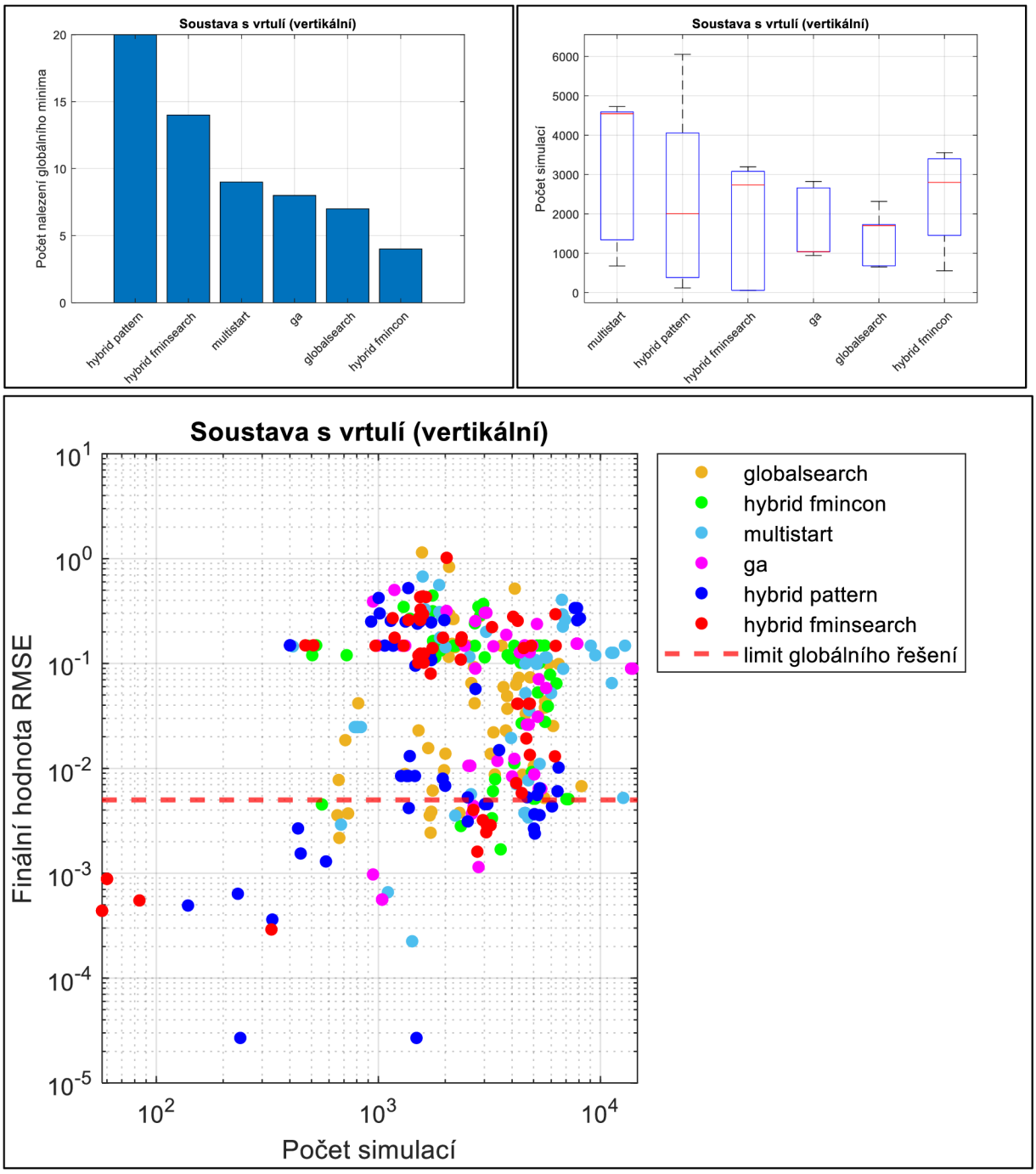
Obr. 82. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Dvojité kyvadlo).



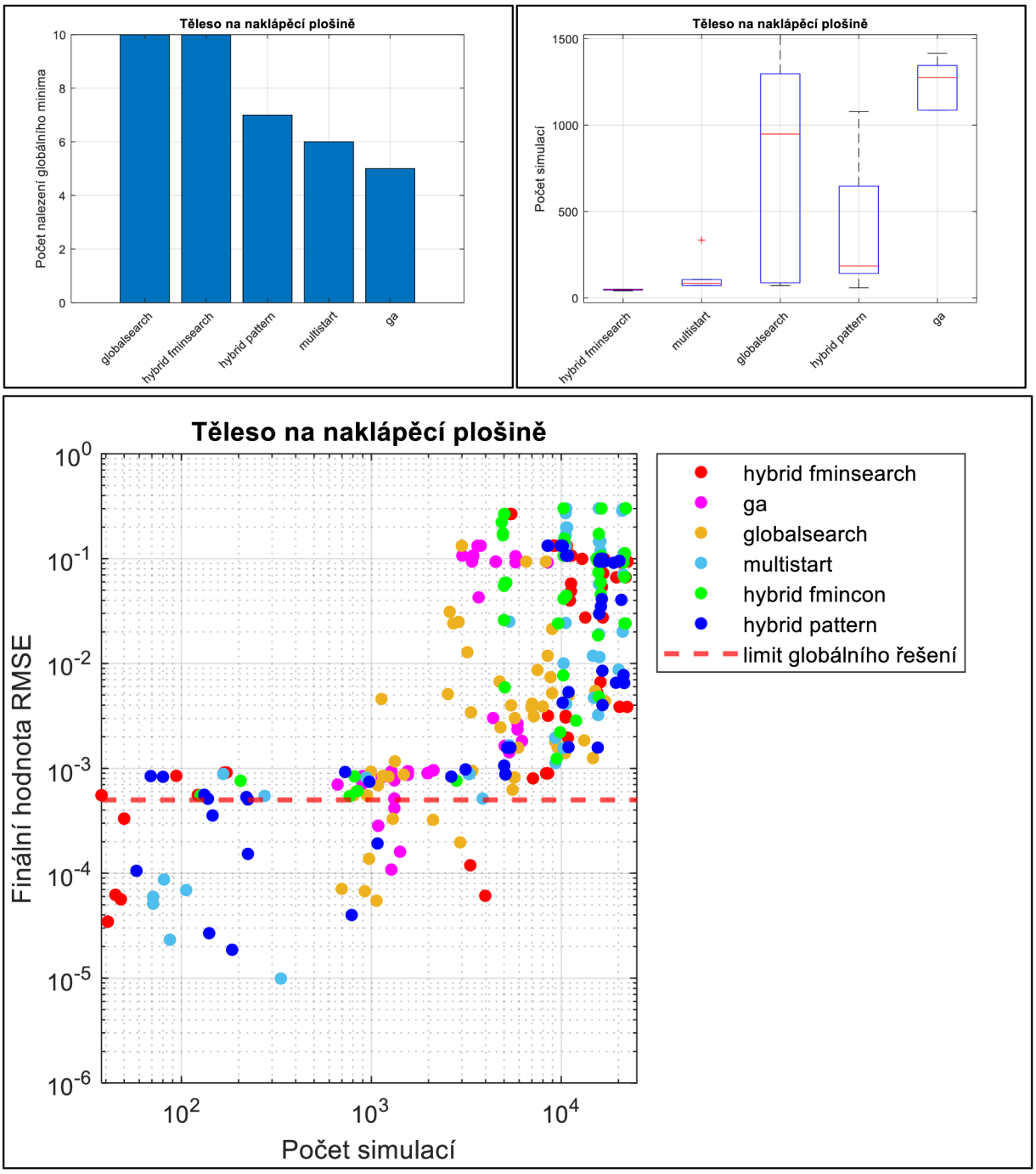
Obr. 83. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Rotační kyvadlo).



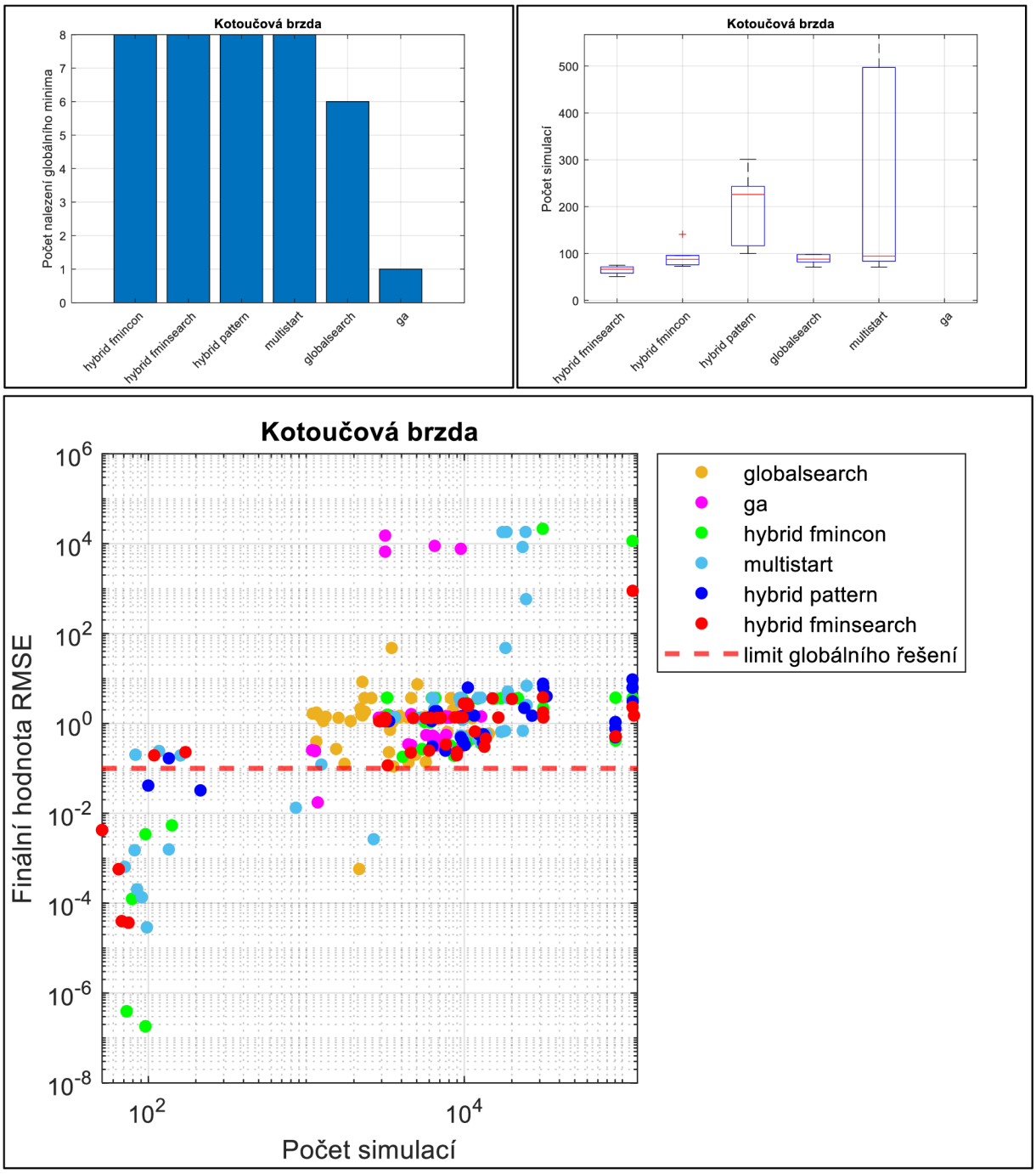
Obr. 84. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Soustava s vrtulí (horizontální)).



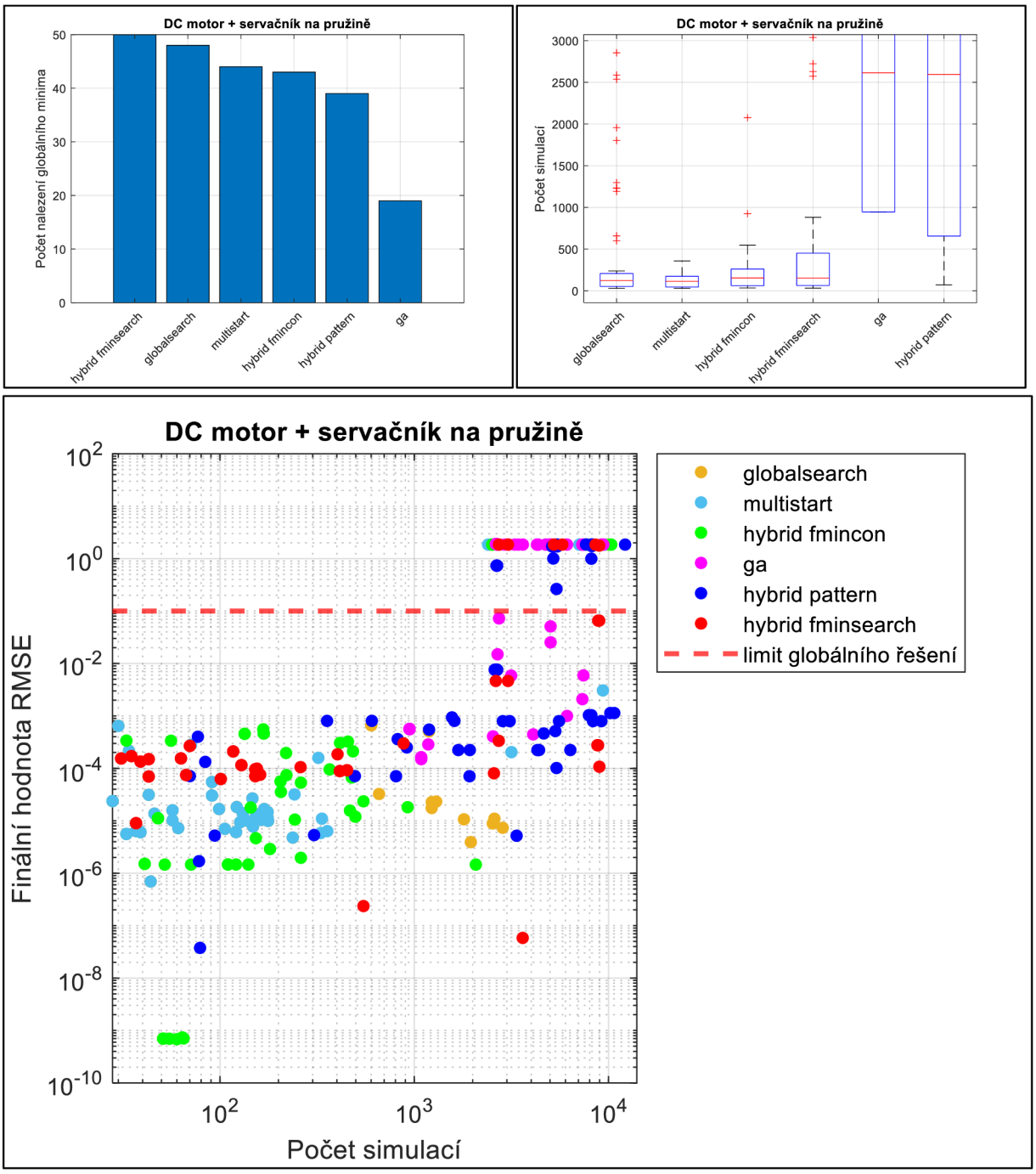
Obr. 85. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Soustava s vrtulí (vertikální)).



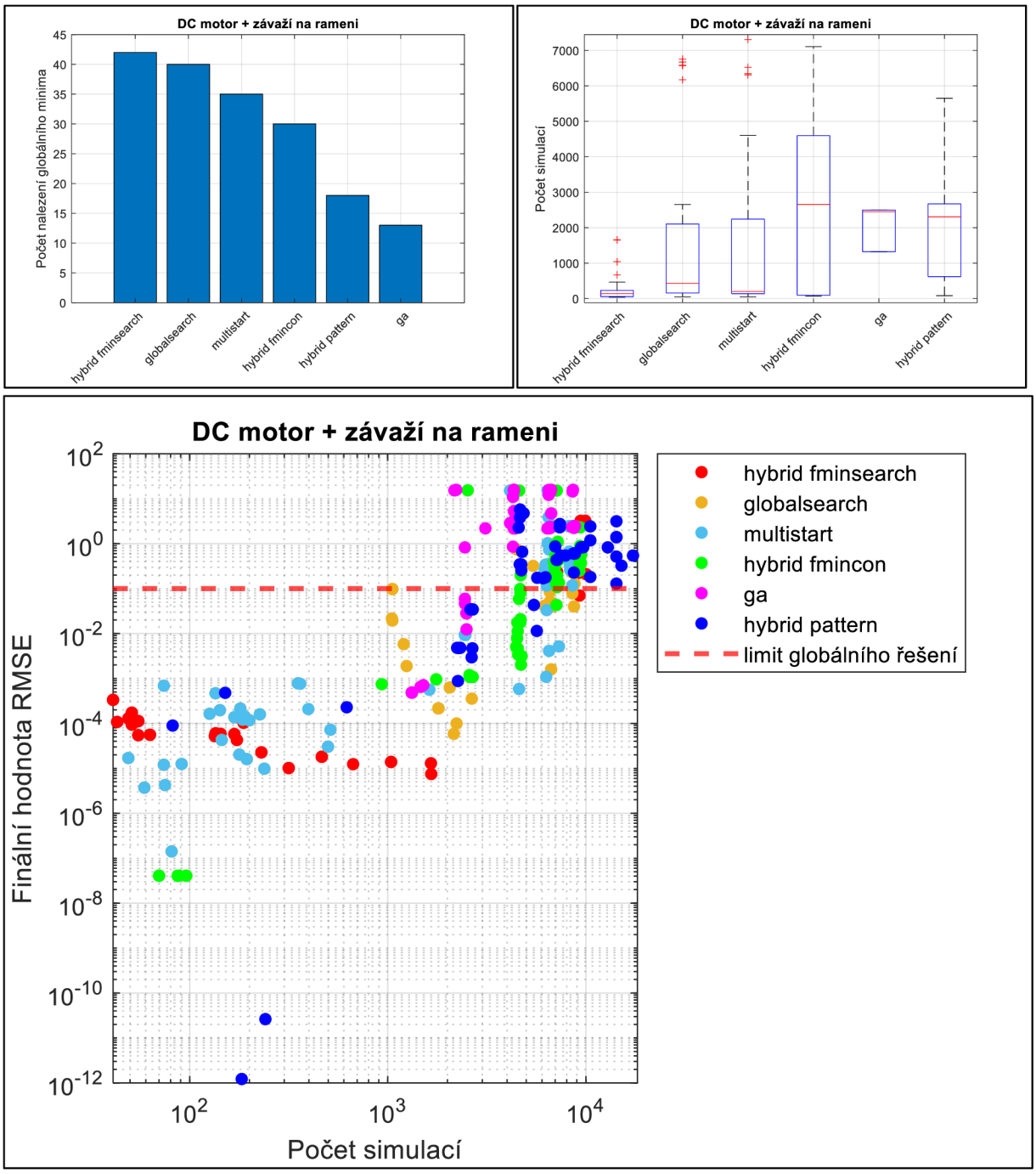
Obr. 86. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Těleso na naklápěcí plošině).



Obr. 87. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (Kotoučová brzda).



Obr. 88. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (DC motor + setrvačník na pružině).



Obr. 89. Grafy výsledků srovnávacího testu globálních optimalizačních algoritmů (DC motor + závaží na rameni).