ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

BAKALÁŘSKÁ PRÁCE

2017

KATEŘINA
BERÁNKOVÁ

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Beránková** Jméno: **Kateřina** Osobní číslo: **410459**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídicí techniky**

Studijní program: **Strojírenství**

Studijní obor: **Informační a automatizační technika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Identifikace systémů využitím hejnových algoritmů

Název bakalářské práce anglicky:

System Identification Using Swarm Algorithms

Pokyny pro vypracování:

1. Seznamte se s metodami globální optimalizace inspirované chováním živočichů v hejnech.
2. Proveďte rešerši hejnových algoritmů vhodných pro identifikaci nelineárních systémů.
3. Vyberte, naprogramujte v prostředí Matlab a simulačně ověřte vhodný hejnový algoritmus pro odhad matematického modelu systému.
4. Na vybrané laboratorní úloze ověřte praktickou aplikovatelnost naprogramovaného algoritmu pro identifikaci.

Seznam doporučené literatury:

ZELINKA, Ivan, Zuzana OPLATKOVÁ, Miloš ŠEDA, Pavel OŠMERA a František VČELAŘ. Evoluční výpočetní techniky: principy a aplikace. 1. vyd. Praha: BEN, 2009, 534 s. ISBN 978-80-7300-218-3.
YANG, Xin-She. Nature-inspired metaheuristic algorithms. 2nd ed. Frome,: Luniver Press, 2010, vi, 148 s. ISBN 978-1-905986-28-6.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

prof. Ing. Milan Hofreiter CSc., U12110.3

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

Datum zadání bakalářské práce: **19.04.2017** Termín odevzdání bakalářské práce: **16.06.2017**

Platnost zadání bakalářské práce: _____

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

19. 4. 2017 Beránková

Datum převzetí zadání Podpis studentky

Poděkování

Děkuji profesoru Milanu Hofreiterovi za vedení bakalářské práce, odborné konzultace a cenné rady a inženýru Vladimíru Hlaváčovi za přístup do počítačové učebny a podporu při testování algoritmů.

Dále bych chtěla poděkovat své kamarádce za pomoc s překlady anglických textů a svému příteli za psychickou podporu.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně s tím, že její výsledky mohou být dále použity podle uvážení vedoucího bakalářské práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků bakalářské práce nebo její podstatné části, pokud budu uvedena jako její spoluautor.

V Praze, 16. června 2017

Kateřina Beránková

Anotace

Tato bakalářská práce se zabývá identifikací parametrů anisochronního modelu. K tomuto účelu je využito hejnových algoritmů. Jedná se o metaheuristické metody inspirované biologickými systémy a inteligencí hejna. V teoretické části práce je seznámení s těmito algoritmy. V praktické části jsou simulovány systémy (nekmitavý a kmitavý) a je provedena jejich identifikace. Následně je provedena identifikace reálného systému. V závěru je porovnána rychlost konvergence použitých algoritmů v jednotlivých případech identifikace.

Klíčová slova

Identifikace, anisochronní model, hejnový algoritmus, netopýří algoritmus, algoritmus světlušek, optimalizace hejnem částic, diferenciální evoluce

Abstract

This bachelor's thesis deals with identification of parameters of an anisochronic model. Swarm algorithm are utilised for this purpose. Those are meta-heuristic methods inspired by biological systems and swarm intelligence. The theoretical part of the thesis familiarizes the reader with the algorithms in question. In practical part, both the oscillating and non-oscillating systems are simulated and their identification is performed. Subsequently, identification of the real system is carried out. In conclusion, the convergence rate of the algorithms used in individual cases of identification is compared.

Key words

Identification, Anisochronic model, Swarm Algorithm, Bat Algorithm, Firefly Algorithm, Particle Swarm Optimization, Differential Evolution

Seznam zkratek

ACO	Ant colony optimization
PSO	Particle swarm optimization
APSO	Accelerated particle swarm optimization
HBA	Honeybee algorithm
VBA	Virtual bee algorithm
ABC	Artificial bee colony optimization
FA	Firefly algorithm
CS	Cuckoo search
BA	Bat algorithm
DE	Differential evolution

OBSAH

1. Úvod

Cílem této práce je seznámit s teorií hejnových algoritmů a vybrat vhodné zástupce pro identifikaci systémů. Tito vybraní zástupci budou následně naprogramováni v prostředí *MATLAB*, otestováni a porovnáni při identifikaci nekmitavého, kmitavého a reálného systému. Výsledky budou porovnány s identifikací provedenou pomocí genetického algoritmu diferenciální evoluce.

Při identifikaci bude vyhodnocován průběh odezvy systému na vstupní signál. Z těchto dat vybrané algoritmy určí parametry anisochronního modelu. Tímto modelem se dají popsat systémy vyšších řádů s dopravním zpožděním. Díky znalosti parametrů můžeme systém řídit.

2. HEJNOVÉ ALGORITMY

2.1. Historie

V roce 1992 Marco Dorigo ve své disertační práci zabývající se optimalizací a přírodními algoritmy popsal algoritmus *optimalizace mravenčí kolonií*, který byl inspirován hejnovou inteligencí kolonie mravenců využívajících feromonové stopy jako chemický způsob komunikace a zanechání zprávy. [1]

Významnější pokrok v této oblasti přišel roku 1995 s vývojem algoritmu *hejna částic*, publikovaným sociologem Jamesem Kennedym a inženýrem Russellem C. Eberhartem. Tento algoritmus byl inspirován chováním živočichů v hejnech, jako jsou ryby nebo ptáci, ale mimo jiné i lidským chováním. Šlo o multiagentní systém, kde byli jednotliví agenti označeni jako *částice*. Agenti prohledávali daný prostor, přičemž inicializační rozmístění bylo náhodné, a během chodu algoritmu spolu navzájem komunikovali. Ukázalo se, že tento algoritmus je lepší metoda, než tradiční vyhledávací přístupy a to včetně genetických algoritmů. [1] [2]

V roce 1997 D. H. Wolpert a W. G. Macready publikovali práci *No free lunch theorem*. Podle tohoto teorému, pokud budeme mít algoritmus A, který řeší danou skupinu problémů efektivněji než algoritmus B, určitě pak máme i skupinu problémů, které algoritmus B řeší efektivněji než algoritmus A. Není tedy možné najít univerzální řešič, který bychom mohli nadřadit nad všechny ostatní algoritmy. Od tohoto okamžiku se výzkum algoritmů zaměřil na takové, které vždy řeší konkrétní skupinu problémů. [3]

V roce 2004 S. Nakrani a C. Tovey navrhli *algoritmus včelí kolonie* a jeho aplikaci pro optimalizaci dynamického přidělování serverů. Tento algoritmus je inspirovaný chováním včelího roje při rozmnožování nebo při shánění potravy. V následujících letech bylo vyvinuto několik modifikací. V roce 2008 byl XS Yangem publikován *algoritmus světlušek*, který napodobuje chování kolonie světlušek při hledání potravy. O rok později XS Yang a S Deb publikovali *kukaččí algoritmus*, který vychází z parazitické hnízdní strategie některých druhů kukaček. XS Yang v následujících letech navrhl ještě *netopýří algoritmus* inspirovaný echolokací a chováním netopýrů a *algoritmus opylení květin.* [1] [4] [5] [6] [7] [8]

2.2. Teorie hejnových algoritmů

Hejnové algoritmy jsou algoritmy založené na kolektivní inteligenci systémů. Systém se obvykle skládá ze skupiny agentů, z nichž každý řeší daný problém sám, ale současně vnímá ostatní agenty ve svém okolí a podle kvality jejich výsledku určuje svůj další postup. Většina těchto algoritmů je inspirována živočišnými organismy nebo biologickými systémy. Jde například o kolonie mravenců, včel, světlušek nebo bakterií.

Tyto metody se velmi dobře hodí při řešení optimalizačních problémů, kde množství parametrů nebo složitost problémů výrazně komplikuje použití analytických metod výpočtu. Ukázali se jako velmi výkonné i v případech, kdy je pro výpočet klasickými metodami nutný příliš dlouhý výpočetní čas. Další výhodou je, že při použití hejnových algoritmů není nutné do detailů rozumět optimalizované úloze.

Hejnové algoritmy nejčastěji pracují s účelovou funkcí, jejíž složitost se odvíjí od složitosti dané úlohy. Daný algoritmus pak hledá minimum/maximum dané funkce. Oproti jiným výpočetním metodám, jako jsou například evoluční algoritmy, nemají hejnové algoritmy tak výrazný sklon uvíznout v lokálním extrému, ale skutečně naleznou extrém globální.

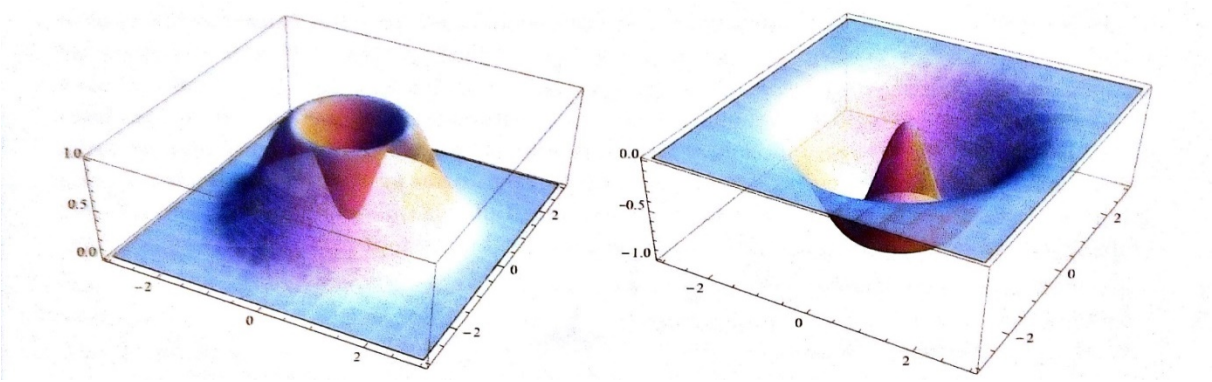
Za nevýhodu můžeme považovat skutečnost, že hejnové algoritmy neurčí vždy extrém účelové funkce zcela přesně, ale s určitou chybou. Ovšem vezmeme-li v úvahu jejich rychlost a relativní jednoduchost, jde o nedostatek velmi malý, ve většině případů dokonce zanedbatelný.

Jako takové v sobě hejnové algoritmy kombinují výhody deterministických a stochastických algoritmů. Deterministické algoritmy mají jasně dané kroky, kterými se řídí. Tím pádem, pokud budeme jeden problém s danými vstupními hodnotami řešit několikrát jedním deterministickým algoritmem, dosáhneme vždy stejného výsledku. Typickým problémem těchto algoritmů je uvíznutí v lokálním extrému. Stochastické algoritmy jsou založeny na určité náhodě. To znamená, že jestliže budeme opět několikrát řešit stejný problém jedním stochastickým algoritmem, dostaneme pokaždé rozdílný výsledek. Hejnové algoritmy mají na počátku zpravidla náhodné rozmístění agentů po ploše představující účelovou funkci. Ti se následně začnou chovat podle přesně definovaných a obvykle velmi jednoduchých pravidel, čímž se postupně dostanou k požadovanému řešení.

2.3. Účelová funkce

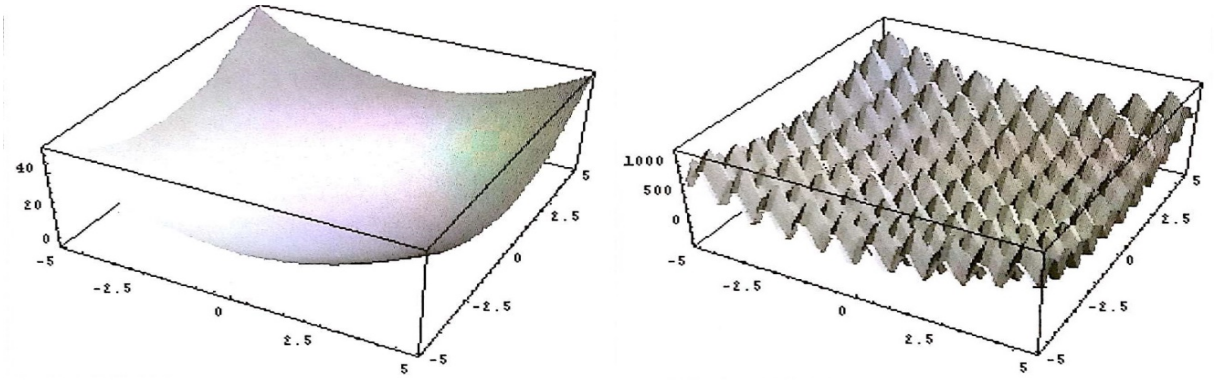
Výrazem *účelová funkce* je označena taková funkce, jejíž optimalizací určíme optimální argumenty. Účelová funkce bývá v literatuře označena jako $\boldsymbol{f}(x)$ nebo $\boldsymbol{f}_{cost}(x)$, kde *cost* je z anglického výrazu *cena*.

Na účelovou funkci lze pohlížet jako na N rozměrnou hyperplochu, na které hledáme její globální extrém, tedy minimum nebo maximum. Pokud máme algoritmus uzpůsobení k hledání minima a chceme určit maximum účelové funkce, vynásobíme ji (-1), čímž dostaneme převrácenou funkci, u níž se polohy extrémů shodují s polohami extrémů u původní účelové funkce.



Obr. 1.: Příklad výměny minim/maxim pomocí násobení (-1) (převzato z [8] str. 68)

Účelová funkce může být unimodální (tj. mít jenom jeden extrém, tím pádem automaticky globální), nebo multimodální (s více extrémy, lokálními nebo globálními). Proto je třeba při řešení volit vhodný algoritmus, aby řešení neuvízlo v lokálním extrému.



Obr. 2.: Příklad a) unimodální, b) multimodální dvourozměrné účelové funkce (převzato z [9] str. 32)

2.4. Mravenčí algoritmus

Mravenčí algoritmus (*Ant colony* optimization – *ACO*) je inspirovaný chováním mravenců. Ti tvoří kolonie o několika milionech jedinců, a přestože při pohybu mimo mraveniště nejsou nijak centrálně řízeni, dokáží spolu spolupracovat a komunikovat. Dosahují toho především díky *feromonu*, kterým označují výhodné cesty vedoucí k potravě.

Algoritmy toho typu mají největší uplatnění při řešení problémů typu *obchodního cestujícího*, tj. mějme N bodů (uzlů) mezi nimiž chceme najít co nejvýhodnější (nejkratší) cestu a to tak, aby cesta byla uzavřená, každým bodem bychom prošli právě jednou a na konci se vrátili do výchozího bodu. Pro N bodů máme $N!$ možných řešení. I kdybychom úlohu uvažovali jako symetrickou, tedy že cesta z bodu A do bodu B je stejně výhodná jako cesta u bodu B do bodu A, budeme mít $(N-1)!/2$ možných řešení. Tato hodnota s počtem měst roste velice rychle. Jde tedy o problém ze skupiny NP-obtížných úloh, tedy nedeterministický polynomiální problém. Analytické řešení takového problému a porovnání všech možných kombinací by nemuselo být možné v reálném čase. Přitom jde o úlohu, kterou například dopravci a doručovatelé musí řešit v podstatě denně. Vzhledem k tomu je výhodný mravenčí algoritmus, který nám může poskytnout velmi dobré řešení v přijatelném čase.

Při startu se mravenci rozejdou z inicializačního bodu (*mraveniště*) náhodným směrem. Cestu označují feromonem, který postupně vyprchává. Když mravenec dosáhne uzlu (*rozcestí*), vydá se po cestě, která je feromonem označena výrazněji (pokud není označena žádná, jde opět náhodně).

Jelikož feromon postupně ubývá, vyprchá z dlouhých a nevýhodných cest dříve, než z krátkých a výhodných. Po čase jsou tím pádem mravenci přitahováni především k optimální trase, zatímco ostatní pro ně nejsou atraktivní.

Pro tento algoritmus jsou hlavní následující vztahy:

$$p_{ij} = \frac{\phi_{ij}^{\alpha} d_{ij}^{\beta}}{\sum_{i,j=1}^{n} \phi_{ij}^{\alpha} d_{ij}^{\beta}} \tag{1}$$

$$\phi_{ij}^{t} = (1-\gamma)\phi_{ij}^{t-1} + \delta\phi_{ij}^{t-1} \tag{2}$$

Rovnice (1) určuje pravděpodobnost, že si mravenec vybere cestu z bodu i do bodu j, kde $\alpha > 0$ a $\beta > 0$ jsou parametry vlivu obvykle nabývající hodnoty $\alpha \approx \beta \approx 2$. Parametr ϕ_{ij} určuje míru koncentrace feromonu na cestě z bodu i do bodu j a d_{ij} určuje vhodnost dané cesty. Hodnota d_{ij} může například vyjadřovat nepřímou úměru délku cesty, pokud položíme $d_{ij} = 1/\overline{ij}$, kde $\overline{ij}$ je hodnota délky cesty z bodu i do bodu j.

Rovnice (2) popisuje, jak se mění koncentrace feromonu ϕ_{ij} na cestě z bodu i do bodu j v čase. Kdyby nebylo do algoritmu implementováno odpařování feromonu, největší váhu by získala cesta na počátku zvolena prvními mravenci náhodně. Díky odpařování se také algoritmus pravděpodobně nezasekne v lokálním minimu. Parametr $\gamma \in < 0{,}1 >$ je rychlost odpařování feromonu. Přírůstek $\delta\phi_{ij}^{t}$ udává množství feromonu na cestě z bodu i do bodu j v čase t.

Zrychlení ACO může být provedeno limitováním koncentrace feromonů a pouze mravenci s dosavadním nejlepším řešením mohou ukládat na cesty nový feromon.

Algoritmus ACO a jeho varianty jsou velmi dobré při řešení komplexních problémů sítí. Kromě problému obchodního cestujícího byly úspěšně aplikovány na problémy internetového směrování. [1]

2.4.1. Pseudokód ACO

```
definuj účelovou funkci f(x), x = (x_1, ..., x_d)^T;
definuj parametry vypařování feromonu;
dokud (není splněna ukončovací podmínka):
  pro všechny mravence i ∈ {1, ..., n}:
      spočítej nová řešení;
      přiřaď novým řešením míru feromonu φ_ij;
      aktualizuj míru feromonu všech řešení;
  konec
  najdi nejlepší řešení;
  t = t + 1;
konec
```

2.5. Optimalizace hejna částic

Optimalizace hejna částic (*Particle swarm optimization – PSO*), je inspirovaný chováním ryb a ptáků v hejnech. Částice se pohybují prohledávaným prostorem, přičemž v každém okamžiku má částice svoji individuální pozici a rychlost. Částice si pamatuje svoji nejoptimálnější pozici, na které byla a také nejoptimálnější pozici, na které se nacházela některá z okolních částic. Z těchto údajů si částice v každém kroku určí novou rychlost a tou se posune na novou pozici. Matematicky lze tento postup vyjádřit následujícími rovnicemi:

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + \alpha\boldsymbol{\epsilon_1}\left[\boldsymbol{g}^* - \boldsymbol{x}_i^{t-1}\right] + \beta\boldsymbol{\epsilon_2}\left[\boldsymbol{x}^* - \boldsymbol{x}_i^{t-1}\right] \tag{3}$$

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \boldsymbol{v}_i^t \tag{4}$$

kde vektory $\boldsymbol{x}_i^t$ a $\boldsymbol{v}_i^t$ představují polohu a rychlost částice *i* v čase *t*, vektor $\boldsymbol{x}^*$ představuje nejoptimálnější z předchozích poloh částice *i* a vektor $\boldsymbol{g}^*$ nejoptimálnější z předchozích poloh okolních částic. Vektory $\boldsymbol{\epsilon}_1$ a $\boldsymbol{\epsilon}_2$ jsou náhodné vektory, jejichž prvky nabývají hodnoty mezi 0 a 1 a parametry α a β jsou parametry učení a konstanty zrychlení, které typicky nabývají hodnoty $\alpha \approx \beta \approx 2$. Inicializační rychlost můžeme volit nulovou, tedy $\boldsymbol{v}_i^{t=0} = 0$. [1]

Pro dobrý chod algoritmu je velmi žádoucí, aby byly částice v inicializačním kroku rozmístěny náhodně po celém (nebo alespoň po co největší části) prohledávaném prostoru a to především při řešení multimodálního problému. Jelikož se při následném výpočtu rychlosti částice bere v úvahu nalezené globální optimum, je pravděpodobné, že algoritmus neuvízne v lokálním extrému.

2.5.1. Zrychlený PSO

Zrychlený algoritmus hejna částic (*Accelerated particle swarm optimization – APSO*), publikovaný XS Yangem v roce 2008 je jednou z modifikací klasického PSO. Na rozdíl od původního algoritmu, APSO nevyužívá při výpočtu nové rychlosti částice *i* její dosavadní optimum $\boldsymbol{x}^*$, ale pouze optimum globální $\boldsymbol{g}^*$. Výpočet rychlosti tedy vypadá následovně:

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + \alpha(\boldsymbol{\epsilon} - 1/2) + \beta\left[\boldsymbol{g}^* - \boldsymbol{x}_i^{t-1}\right] \tag{5}$$

kde vektor $\boldsymbol{\epsilon}$ je opět náhodný vektor s hodnotami mezi 0 a 1. Parametry učení α a β v tomto případě obvykle nabývají hodnot $\alpha \approx (0{,}1 - 0{,}4)$ a $\beta \approx (0{,}1 - 0{,}7)$, přičemž pro unimodální problém můžeme volit $\alpha \approx 0{,}2$, $\beta \approx 0{,}5$. [1] [2]

2.5.1. Pseudokód PSO

definuj účelovou funkci $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \dots, x_d)^T$;
pro všechny částice $i \in \{1, \dots, n\}$:
  inicializuj počáteční polohy $\boldsymbol{x}_i^0$;
  nastav $\boldsymbol{x}^* = \boldsymbol{x}_i^0$;
urči $\boldsymbol{g}^* = \min\big(f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_n)\big)$;
dokud (není splněna ukončovací podmínka):
  pro všechny částice $i \in \{1, \dots, n\}$:
    spočítej novou rychlost $\boldsymbol{v}_i$;
    spočítej novou polohu $\boldsymbol{x}_i$;
    urči kvalitu řešení $f(\boldsymbol{x}_i)$;
    urči novou hodnotu $\boldsymbol{x}^*$;
  konec
  urči novou hodnotu $\boldsymbol{g}^*$;
  t = t + 1;
konec

2.6. Včelí algoritmus

Další třídou hejnových algoritmů jsou algoritmy inspirované chováním včelího roje. Včely tvoří kolonie, ve kterých se společně brání své území, a sbírají potravu. K těmto činnostem jim pomáhají dva druhy komunikace. První druh je založen na chemických látkách (feromonech), které včely vylučují a mohou tím ovlivnit chování ostatních včel ve svém okolí. Tuto komunikaci včely využívají zejména při napadení. Druhým způsobem je zvláštní druh pohybu, obvykle označovaný jako včelí tanec. Pokud včela najde zdroj potravy, předá tuto informaci ostatním pomocí charakteristického natřásavého tance. Pomocí něho je včela schopna sdělit informace o směru, vzdálenosti a kvality zdroje.

Chováním včel je inspirována celá řada algoritmů. Nejčastěji jsou postaveny na principech chování včel medonosných.

2.6.1. Základní včelí algoritmus

Základní včelí algoritmus *(Honeybee algorithm – HBA)* napodobuje snahu včel optimalizovat sběr nektaru. Rozdělení dělnic mezi jednotlivé zdroje potravy tady závisí na mnoha faktorech. Především jde o kvalitu zdroje potravy a jeho vzdálenost od úlu. Jde o podobný problém jako je přidělování hostingových serverů na internetu (což byla jedna z prvních aplikací tohoto algoritmu v roce 2004).

Včelí roj o N včelách je rozdělen na dvě skupiny: n_f průzkumnic a $N - n_f$ pozorovatelek. Pokud průzkumnice najde vhodný zdroj potravy, začne signalizovat ostatním včelám v roji. Pravděpodobnost, že ji pozorovatelka bude následovat je dána vztahem:

$$p_i = \frac{w_i^t}{\sum_{i=1}^{n_f} w_i^t} \tag{6}$$

kde w_i^t vyjadřuje sílu tance průzkumnice i v čase t. Alternativně můžeme definovat pravděpodobnost prohledávání Gaussovského typu $p_e = 1 - p_i = \exp\left[-w_i^2/2\sigma^2\right]$, kde σ je volatilita včelí kolonie a ovládá průzkum. Je zjevné, že pro $w_i \to 0$ je $p_e = 1$, takže všechny včely průzkumnice budou náhodně prohledávat oblast. Zároveň je v takovém případě $p_i = 0$, takže je žádná z včel pozorovatelek nebude následovat.

Tento algoritmus je možné jednoduše modifikovat pro dynamické hodnocení jednotlivých tras mezi jednotlivými body úpravou rovnice (6):

$$p_{ij} = \frac{w_{ij}^{\alpha} d_{ij}^{\beta}}{\sum_{i,j=1}^{n} w_{ij}^{\alpha} d_{ij}^{\beta}} \tag{7}$$

kde $\alpha > 0, \beta > 0$ jsem parametry vlivu, w_{ij} je síla tance podél cesty z bodu i do bodu j a d_{ij} vhodnost dané trasy. Nejvhodnější trasa je tedy následovaná největším počtem včel a stane se preferovanou (podobně jako u mravenčího algoritmu).

2.6.2. Virtuální včelí algoritmus

Virtuální včelí algoritmus *(Virtual Bee Algorithm – VBA)* je upravený včelí algoritmus speciálně upravený pro řešení jak diskrétních tak spojitých problémů.

Spojitá optimalizovaná funkce je zakódovaná jako virtuální nektar a řešení je pozice nektaru. Síla včelího tance je kombinována s faktory, jako je koncentrace nektaru, jako vhodnost řešení. U diskrétních problémů je účelová funkce vyjádřena množstvím nektaru daného zdroje potravy, který láká včely. Výhodnější pozice způsobí silnější natřásavý tanec. V tomto ohledu je VBA velmi podobný jako HBA. Rozdílem je, že při VBA zná současnou nejlepší pozici každá včela v kolonii a je možné je tímto směrem všechny vyslat. Dělnice se tedy nemusejí pokaždé vracet k úlu a signalizovat pozorovatelkám pomocí tance, což šetří čas. Tím je VBA podobný PSO a může být výhodnější než HBA.

2.6.3. Optimalizace umělé včelí kolonie

V algoritmu *Artificial Bee Colony Optimization (ABC optimization)* jsou včely rozděleny do tři skupin: dělnice, pozorovatelky a průzkumnice. Každému zdroji potravy odpovídá jedna včela dělnice. Dělnice z vyčerpaného (vyřazeného) zdroje potravy se stane průzkumnicí a začne v oblasti náhodně hledat nový zdroj potravy. Dělnice také sdílejí informace o zdroji potravy s pozorovatelkami v úlu. Ty se tedy mohou rozhodnout pro daný zdroj. Na rozdíl od HBA, kde máme pouze dvě skupiny včel, jsou zde včely více specializované.

Účinnost sběru se pro konkrétní zdroj určí ze vztahu F/T, kde F je množství nektaru a T je čas strávený u daného zdroje.

Včelí algoritmy si v posledních letech našli mnohá uplatnění, například v kombinatorické optimalizaci, plánování úloh nebo při optimalizacích inženýrských konstrukcí. [6] [4]

2.6.4. Pseudokód HBA

definuj účelovou funkci $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_d)^T$;
dokud (není splněna ukončovací podmínka):
  pro všechny včely $i \in \{1, \ldots, n\}$:
    spočítej nová řešení;
    ohodnoť nová řešení;
    urči p_i;
  konec

```
    komunikuj;
    urči nejlepší řešení;
    t = t + 1;
konec
```

2.7. Algoritmus světlušek

Algoritmus světlušek *(Firefly algorithm – FA)* je založen na světélkování a chování tropických světlušek. Podobně jako včely využívají včelí tanec, aby informovali ostatní členy roje o polohách a kvalitách zdrojů potravy, světlušky regulují míru svého záření, čímž komunikují se svým okolím.

Světlušky jsou náhodně rozmístěny v prohledávaném prostoru. Každá světluška vyhodnocuje kvalitu své pozice a podle toho upravuje míru svého světélkování – čím lepší pozice, tím více září. Aby v dalším kroku svoji pozici vylepšila, zamíří směrem ke světlušce, která má v jejím pozorovatelném okolí nejlepší pozici. Po určitém množství kroků tak budou světlušky shluknuté kolem nejvýhodnější pozice.

Zjednodušeně můžeme algoritmus charakterizovat následujícími výroky:

- Atraktivita světlušky je přímo úměrná její zářivosti a nepřímo úměrná její vzdálenosti.
- Každá světluška je přitahována k nejatraktivnější světlušce ve svém okolí. Pokud žádná taková v jejím okolí není, světluška se pohybuje náhodně.
- Zářivost světlušky je dána její polohou na základě účelové funkce.

Při inicializaci algoritmu je důležité, aby byly světlušky po ploše rozprostřeny pokud možno rovnoměrně, aby měly co nejlepší pravděpodobnost nalezení globálního optima účelové funkce.

Matematické vyjádření algoritmu je:

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \beta_0 e^{-\gamma r_{ij}^2}\left(\boldsymbol{x}_j^{t-1} - \boldsymbol{x}_i^{t-1}\right) + \alpha \boldsymbol{\epsilon}_i^t \tag{8}$$

kde $\boldsymbol{x}_i^t$ je poloha světlušky i v čase t, β_0 je parametr určující atraktivitu světlušky j při nulové vzdálenosti a výraz $e^{-\gamma r_{ij}^2}$ vyjadřuje pokles atraktivity s rostoucí vzdáleností, kde

$\gamma > 0$ je parametr absorpce světla. Zjednodušeně můžeme uvažovat případ, kdy je atraktivita světlušky dána pouze její zářivostí, neboli přímo její pozicí ve vztahu k účelové funkci. Poslední sčítanec vyjadřuje náhodnost, kde $\boldsymbol{\epsilon}^t$ je vektor náhodných čísel v čase t a α je parametr náhodnosti.

Z rovnice (8) můžeme vidět, že položíme-li parametr $\beta_0 = 0$, dostaneme rovnici harmonického vyhledávání (*harmony search – HS*), což je jedna ze stochastických metod optimalizace. Další možností je položit hodnotu $\boldsymbol{x}_j^t = \boldsymbol{g}^*$, kde $\boldsymbol{g}^*$ je dosavadní optimální hodnota, stane se z FA algoritmus APSO. A nakonec, položíme-li $\alpha = 0$, získáme algoritmus diferenciální evoluce (DE), který patří do skupiny genetických algoritmů. FA kombinuje vlastnosti všech těchto algoritmů. Díky tomu může pokrýt větší množství optimalizačních problémů a dosahuje vyšší konvergenční účinnosti.

Přestože základní FA je velmi efektivní algoritmus, existuje řada jeho variant, které základní verzi ještě vylepšují. Jde například o diskrétní algoritmus světlušek (*Discrete firefly algorithm – DFA*), memetický algoritmus světlušek (*Memetic firefly algorithm – MFA*), chaotický algoritmus světlušek (*Chaotic firefly algorithm – CFA*), hybridní algoritmus světlušek (*Hybrid firefly algorithm – HFA*) a další. [1] [5]

2.7.1. Pseudokód FA

```
definuj účelovou funkci f(x), x = (x_1, ..., x_d)^T;
pro všechny světlušky i ∈ {1, ..., n}:
  inicializuj počáteční polohy x_i^0;
  urči atraktivitu světlušky A = f(x_i^0);
dokud (není splněna ukončovací podmínka):
  pro všechny částice i ∈ {1, ..., n}:
      pro všechny částice j ∈ {1, ..., n}:
          pokud (A_i < A_j):
              pohni světluškou i ke světlušce j;
          jinak:
              pohni světluškou i náhodně;
              uprav zářivost světlušky i;
          konec
      konec
  konec
```

```
    urči novou hodnotu g*;
    t = t + 1;
konec
```

2.8. Kukaččí algoritmus

Princip kukaččího algoritmu (*Cuckoo search – CS*) je inspirován parazitismem při hnízdění některých druhů kukaček. Kukačky obvykle snášejí svá vejce do cizích hnízd, obvykle do hnízd jiných druhů ptáků. V tomto ohledu existují tři základní druhy parazitismu: vnitrodruhový, kooperační a přebíraní hnízd. Pokud hostitelský pták zjistí, že má v hnízdě i cizí vejce, může se jich zbavit, nebo jednoduše hnízdo opustit a vybudovat si jiné. Některé kukačky jsou proto přímo specializovány na parazitování na určitém druhu ptactva a jejich vejce jsou si podobná velikostí a vzhledem skořápky. Také načasování snášení vajec musí být přizpůsobeno době, kdy vejce snáší i hostitelský druh. Přitom se ale kukaččí mláďata líhnou zpravidla dříve než ostatní vejce v hnízdě a jejich prvním instinktem je zbavit se těchto vajec, aby si tak zajistili výhody při krmení.

Kromě inspirace kukaččím parazitismem, je tento algoritmus doplněn tzv. Lévyho letem, což je výhodnější než použití čistě náhodného kroku. Jak ukazují studie, může být díky tomu CS výrazně efektivnější než PSO nebo genetické algoritmy.

Kukaččí algoritmus se řídí následujícími pravidly:

- Každá kukačka v každém časovém kroku snese jedno vejce do náhodně vybraného hnízda.
- Do dalších generací bude uchováno hnízdo s nejkvalitnějšími vejci.
- Počet hostitelských hnízd je konstantní. Pravděpodobnost, že hostitelský pták objeví kukaččí vejce, je $p_a = \langle 0, 1 \rangle$. Pokud je kukaččino vejce objeveno, hostitel se ho buď zbaví, nebo hnízdo opustí a vybuduje si nové.

Poslední předpoklad také můžeme nahradit aproximací, kde část hnízd p_a je nahrazena n novými hostitelskými hnízdy s náhodným řešením.

Při implementaci můžeme využít reprezentaci, kde každé vejce v hnízdě odpovídá jednomu řešení problému a každá kukačka může snést jenom jedno vejce.

Cílem je, v každém kroku nahradit řešení v hnízdě lepším (kukaččím) vejcem. Pochopitelně se tento algoritmus může rozšířit, pokud povolíme, aby jedno hnízdo obsahovalo víc vajec. Ve zjednodušeném případě uvažujeme v každém hnízdě pouze jedno vejce. V takovém případě každé kukačce odpovídá právě jedno vejce a právě jedno hnízdo. Z pohledu algoritmu je tedy není nutné rozlišovat a mluvíme pouze o daném řešení.

Matematické vyjádření nové generace kukaček tedy je:

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \alpha L(s, \lambda) \tag{9}$$

kde $\boldsymbol{x}$ je řešení v hnízdě i v čase t, $\alpha > 0$ je krok úměrný velikosti prohledávaného pole (ve většině případů můžeme použít zjednodušení $\alpha = 1$). $L(s, \lambda)$ je funkce vyjadřující Lévyho let (viz rovnice (10)).

Rovnice (9) je vyjádření stochastické metody *náhodné procházky*. Náhodná procházka je v podstatě Markovův řetězec, u kterého následující pozice závisí pouze na současné pozici a pravděpodobnosti přechodu. Řada nových řešení díky tomu bude generována v okolí dosavadního nejlepšího řešení, čímž se urychlí lokální vyhledávání. Nicméně výrazná část nových řešení by měla být generována dostatečně daleko od dosavadního nejlepšího řešení, čímž je zajištěno, aby algoritmus neuvízl v lokálním minimu.

V mnohém je CS velmi podobný PSO, ale je rozšířený elitismem a selekcí. Také náhodný krok je zde účinnější, protože je přípustný libovolně dlouhý krok. V neposlední řadě, u CS musíme volit méně parametrů než u PSO, čímž může mít CS širší užití v optimalizačních problémech. [1] [7]

2.8.1. Pseudokód CS

```
definuj účelovou funkci f(x), x = (x_1, ..., x_d)^T;
pro všechny hnízda i ∈ {1, ..., n}:
  inicializuj počáteční polohy x_i^0;
dokud (není splněna ukončovací podmínka):
  vyber kukačku i;
  generuj nové řešení (pomocí Lévyho letu);
  vyber hnízdo j;
```

```
    pokud (f(x_i) < f(x_j)) :
        nahraď řešení v hnízdě řešením kukačky;
    konec
    znič p_a část nejhorších hnízd;
    náhodně generuj nová hnízda;
    ohodnoť řešení v hnízdech a najdi nejlepší;
    t = t + 1;
konec
```

2.8.2. Lévyho lety

Lévyho lety jsou inspirované pohybem zvířat, která jsou schopná se pohybovat i na dlouhou vzdálenost pomocí různě dlouhých přeletů. Jejich implementace do optimalizačních procesů přináší zrychlení výpočtů a mnohdy i přesnější výsledky.

Jednoduše řečeno jsou Lévyho lety náhodná procházka, u které je délka kroku brána z Lévyho distribuce. Matematicky je definován následovně:

$$L(s,\lambda) = \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi}\frac{1}{s^{1+\lambda}} \tag{10}$$

kde $s \in \mathbb{R}$ je váha, $\lambda \in \langle 0{,}2\rangle$ je exponent funkce hustoty a $\Gamma(\lambda)$ vyjadřuje Gama funkci (11). [6] [9]

$$\Gamma(\lambda) = \int_0^\infty t^{\lambda-1}e^{-t}dt \tag{11}$$

2.9. Netopýří algoritmus

Netopýří algoritmu (*Bat algorithm – BA*) je inspirován netopýry a jejich schopností echolokace. Netopýři dokáží vydávat velmi silné zvukové pulsy a na základě vyhodnocení odražených zvukových vln mohou najít kořist, se vyhnout okolním předmětům nebo rozlišit ostatní netopýry.

Algoritmus můžeme popsat následujícími idealizovanými pravidly:

- Všichni netopýři používají echolokaci k určení vzdálenosti od kořisti. Také jsou schopni rozeznat kořist od překážky.

- Netopýři poletují náhodnou rychlostí $\boldsymbol{v}_i$ na pozici $\boldsymbol{x}_i$. Mohou automaticky měnit svou ji frekvenci a rychlost vysílání zvukových pulsů v závislosti na vzdálenosti svého cíle.
- Přestože se hlasitost pulzů může lišit, pohybuje se v rozpětí od kladné hodnoty A_0 do minimální hodnoty A_{min}.

Pro jednoduchost můžeme uvažovat, že frekvence vysílaných pulzů jednotlivých netopýrů je $f \in \langle 0, f_{max} \rangle$. Rychlost vysílání pulzů stanovíme jednoduše v rozmezí $\langle 0{,}1 \rangle$, kde 0 znamená, že netopýr nevysílá žádné pulzy a naopak 1 znamená maximální rychlost vysílání pulzů.

Při simulaci BA používáme virtuální netopýry, jejichž pohyb je stanoven následujícími vztahy:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{12}$$

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + \left(\boldsymbol{x}_i^{t-1} - \boldsymbol{x}_*\right) f_i \tag{13}$$

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \boldsymbol{v}_i^t \tag{14}$$

kde $\beta \in \langle 0{,}1 \rangle$ je vektor náhodnosti z rovnoměrné distribuce a $\boldsymbol{x}_*$ je prozatímní globální optimum. Výpočet nové rychlosti a polohy může připomínat PSO, nicméně BA může být v mnoha případech efektivnější, protože používá frekvenci, jako faktor ovlivňující průzkum prostoru.

V každém kroku je zároveň nutné aktualizovat parametry r_i a A_i, které vyjadřují rychlost vysílání pulzů a hlasitost pulzů. Tyto aktualizace se řídí rovnicemi:

$$A_i^t = \alpha A_i^{t-1} \tag{15}$$

$$r_i^t = r_i^0 [1 - \exp(-\gamma t)] \tag{16}$$

kde $\alpha \in (0{,}1)$ a $\gamma > 0$ jsou konstanty. Je zjevné, že pro $t \to \infty$: $A_i^t \to 0, r_i^t \to r_i^0$. V nejjednodušším případě můžeme uvažovat $\alpha = \gamma = 0{,}7$.

Stejně jako jiné hejnové algoritmy, má i BA mnoho dalších rozšiřujících variant. Nejvýznamnější je asi binární netopýří algoritmus (*Binary bat algorithm – BBA*), který na rozdíl od základního BA velmi dobře pracuje na disktrétních nebo některých

kombinatorických problémech. Dalšími variantami jsou například chaotický netopýří algoritmus (*Chaotic bat algorithm – CBA*), fuzzy logický netopýří algoritmus (*Fuzzy logic bat algorithm – FLBA*) multiobjektivní netopýří algoritmus (*Multi-objective bat algorithm – MOBA*) nebo modifikovaný netopýří algoritmus (*Modified bat algorithm – MBA*).[1] [8]

2.9.1. Pseudokód BA

definuj účelovou funkci $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \dots, x_d)^T$;
pro všechny netopýry $i \in \{1, \dots, n\}$:
  inicializuj počáteční polohy $\boldsymbol{x}_i^0$;
  inicializuj frekvenci f_i, rychlost vysílání pulzů r_i a hlasitost A_i;
dokud (není splněna ukončovací podmínka):
  pro všechny netopýry $i \in \{1, \dots, n\}$:
    generuj novou frekvenci f_i;
    urči novou rychlost $\boldsymbol{v}_i$ a novou polohu $\boldsymbol{x}_i$;
    pokud ($náhodné\ číslo > r_i$):
      vyber nejlepší řešení $\boldsymbol{x}_*$;
      generuj lokální řešení kolem $\boldsymbol{x}_*$;
    konec
    urči nová řešení náhodným letem;
    pokud ($náhodné\ číslo < A_i\ \&\ f(\boldsymbol{x}_i) < f(\boldsymbol{x}_*)$):
      přijmi nové řešení;
      zvyš r_i, sniž A_i;
    konec
    ohodnoť netopýry
    najdi nejlepší řešení $\boldsymbol{x}_*$;
    t = t + 1;
  konec
konec

2.10. Další hejnové algoritmy

Výše zmíněné algoritmy byly zvoleny jako nejvýznamnější zástupci ze skupiny hejnových algoritmů, ale v žádném případě nejsou jediné. Problematika hejnových

algoritmů se dočkala velkého zájmu matematiků zejména v posledních deseti letech. Neustále jsou vyvíjeny nové algoritmy, varianty algoritmů stávajících nebo specializovanější algoritmy, které jsou určené pro řešení specifické skupiny problémů. Příkladem algoritmů vyvinutých v posledních letech může být například algoritmus opylení květin (*Flower Pollination Algorithm – FPA*, 2012) nebo vlčí algoritmus (*Wolf Search Algorithm – WSA*, 2012).

Vzhledem k tomu, že jsou hejnové algoritmy stále relativně mladým tématem, je okolo nich ještě mnoho prostoru ke studiu. Například jenom část z těch algoritmů je podložena matematickou analýzou. V tomto ohledu je třeba řady studií, které by ve výsledku mohli přinést zajímavý pohled na metaheuristické algoritmy obecně. Dále zde stále není plně ujasněná terminologie a klasifikace. A v neposlední řadě je nutné ujasnit volbu parametrů, které značně ovlivňují kvalitu a plynulost běhu algoritmu. [1] [6] [9]

2.11. Diferenciální evoluce

Vzhledem k cílům této práce je na tomto místě vhodné zmínit algoritmus diferenciální evoluce (*Differential evolution – DE)*, přestože nejde o zástupce hejnových algoritmů. DE patří do skupiny genetických algoritmů, které také patří mezi stochastické algoritmy.

Genetické algoritmy pracují na principu křížení. Na počátku náhodně inicializují náhodná řešení. V dalším kroku se provede *ohodnocení* jednotlivých řešení. Nejlepší řešení jsou vybrána do další generace. Dále se provede *křížení*, kdy se vyberou dvě řešení – *rodiče* – a vygenerují se z nich další řešení – *potomci.* Následně proběhne *mutace* potomků, kdy jsou jejich parametry stochasticky upraveny. Dále proběhne další ohodnocení a celý proces se opakuje, dokud není nalezeno řešení nebo není splněna ukončovací podmínka. Různé genetické algoritmy se mohou v jednotlivých krocích lišit. Podobně jako u hejnových algoritmů je zde výhodou, že můžeme genetické algoritmy implementovat, aniž bychom měli rozsáhlé znalosti o optimalizované funkci, a při správné implementaci jsou odolné proti uvíznutí v lokálním extrému. Na druhou stranu je vždy třeba provést řadu iterací (*generací*) a není zde jistota, že je nalezené řešení přesnou hodnotou globálního optima.

Zvláštností diferenciální evoluce oproti jiným genetickým algoritmům je, že k vytvoření nového potomka využívá vždy čtyři řešení z rodičovské generace a ne dva,

jak je tomu obvykle. V každém časovém kroku t jsou pro každý vektor řešení $\boldsymbol{x}_i$ náhodně vybrány další tři vektory řešení $\boldsymbol{x}_p$, $\boldsymbol{x}_q$ a $\boldsymbol{x}_r$. DE probíhá vždy ve třech krocích: mutace, křížení a výběr. Tyto kroky jsou matematicky vyjádřeny následovně:

$$\boldsymbol{v}_i^t = \boldsymbol{x}_p^{t-1} + F\left(\boldsymbol{x}_q^{t-1} - \boldsymbol{x}_r^{t-1}\right) \tag{17}$$

$$\boldsymbol{u}_{j,i}^t = \begin{cases} \boldsymbol{v}_{j,i}^t, & pokud\ r_i \leq C_r \\ \boldsymbol{x}_{j,i}^{t-1}, & jinak \end{cases} \tag{18}$$

$$\boldsymbol{x}_i^t = \begin{cases} \boldsymbol{u}_i^t, & pokud\ \boldsymbol{f}\left(\boldsymbol{u}_i^t\right) \leq \boldsymbol{f}\left(\boldsymbol{x}_i^{t-1}\right) \\ \boldsymbol{x}_i^{t-1}, & jinak \end{cases} \tag{19}$$

kde $\boldsymbol{v}_i^t$ je šumový vektor řešení, $F \in \langle 0,2 \rangle$ je mutační konstanta a $C_r \in \langle 0,1 \rangle$ je práh křížení. Hodnota $r_i \in \langle 0,1 \rangle$ je určena náhodně z rovnoměrného rozložení a určuje pravděpodobnost, že $j$tá složka vektoru řešení $\boldsymbol{v}_i^t$ bude složkou $j$tou složkou potomka $\boldsymbol{u}_i^t$. Nakonec rovnice (19) vyjadřuje výběr lepšího řešení, které postoupí do další generace. [1] [10]

2.11.1. Pseudokód DE

```
definuj účelovou funkci f(x), x = (x_1, ..., x_d)^T;
pro všechna i ∈ {1, ..., n}:
  inicializuj počáteční řešení x_i^0;
dokud (není splněna ukončovací podmínka):
  pro všechna i ∈ {1, ..., n}:
      náhodně zvol 3 vektory řešení x_p, x_q a x_r;
      urči vektor šumu v_i;
      generuj náhodný vektor r_i;
          pro všechna j ∈ {1, ..., n}:
              pokud (r_i ≤ C_r):
                  u_{j,i}^t = v_{j,i}^t;
              jinak:
                  u_{j,i}^t = x_{j,i}^{t-1};
              konec
          konec
      pokud f(u_i^t) ≤ f(x_i^{t-1}):
          x_i^t = u_i^t;
```

jinak:

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1}$$

konec

urči nejlepší řešení;

konec

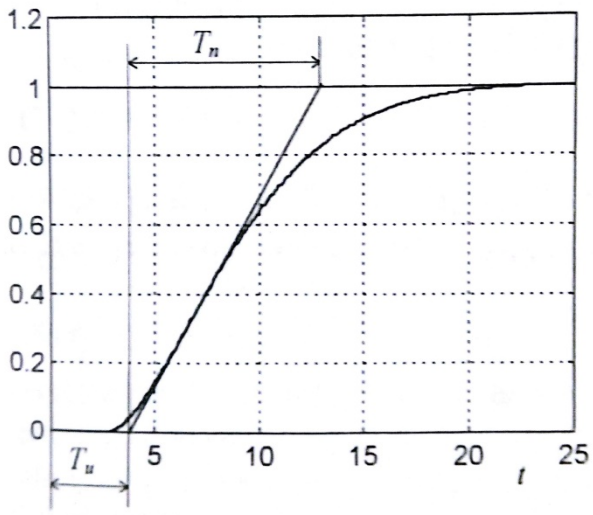
3. IDENTIFIKACE SYSTÉMU

3.1. Simulovaný systém s dopravním zpožděním

Pro identifikaci systému je použit anisochronní model s dopravním zpožděním nejen na vstupu, ale i na zpětné vazbě. Touto soustavou lze nahradit soustavy vyšších řádů. Přenos anisochronní soustavy je dán následujícím vztahem:

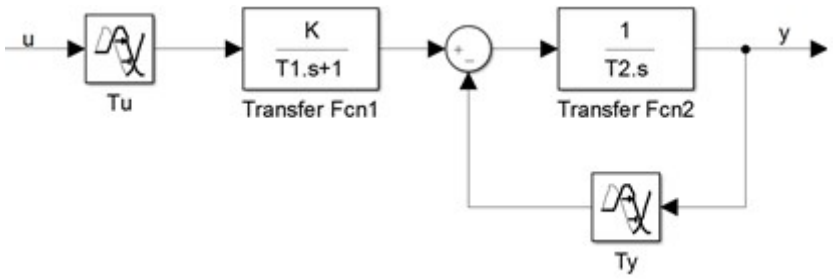
$$G(s) = \frac{K \exp(-\tau_u s)}{(\tau_1 s + 1)(\tau_2 s + \exp(-\tau_y s))} \tag{20}$$

kde K je koeficient statické citlivost, τ_u je dopravní zpoždění na vstupu soustavy a τ_y je zpoždění na zpětné vazbě. Hodnota τ_1 souvisí s rozběhem přechodu, můžeme psát $\tau_1 + \tau_u = T_u$, kde T_u je doba náběhu (viz obr. 3). Hodnota τ_2 pak odpovídá době průtahu T_n. [11]



Obr. 3.: Přechodová charakteristika, T_u = doba náběhu, T_n = doba průtahu (převzato z [11] str. 14)

Pro simulaci soustavy je použito prostředí *MATLAB SIMULINK*. Zde je přenos zapsán soustavou bloků, které mají jako vstupní parametry nastaveny jednotlivé koeficienty přenosu.

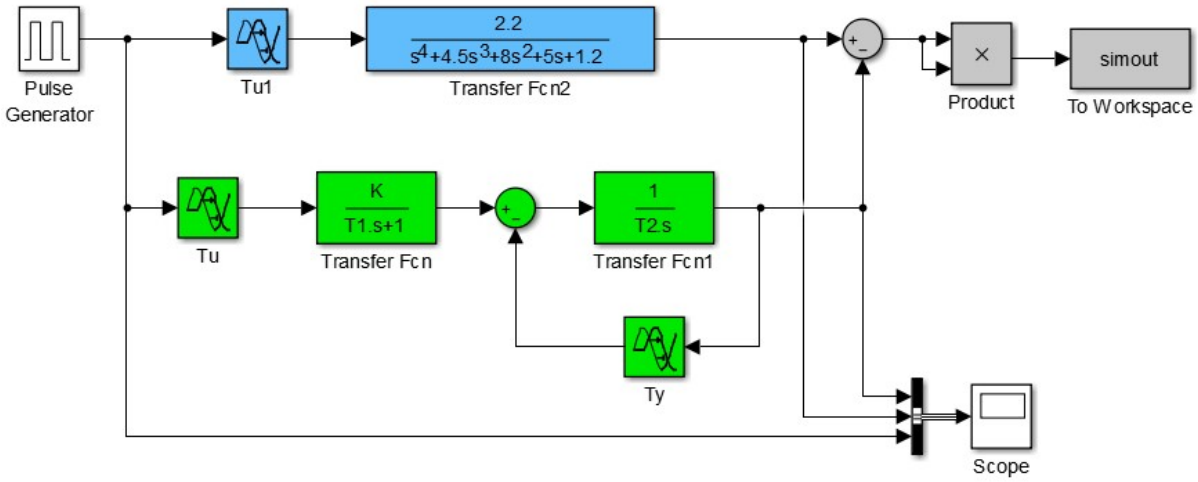


Obr. 4.: Zápis přenosu soustavy (20) v prostředí *SIMULINK*

Pokud do bloků vložíme konkrétní hodnoty, získáme graf přenosu nebo přímo hodnoty, se kterými můžeme dál pracovat.

Pro identifikaci soustavy použiji dvě paralelně zapojené soustavy, přičemž parametry jedné z nich budou pevně nastaveny a parametry druhé budou vypočítávány pomocí zvoleného algoritmu. Program následně porovná křivky obou průběhů a vyhodnotí kvadrát rozdílu mezi nimi. Ten bude následně sloužit jako účelová funkce algoritmu, kterou bude cílem minimalizovat.

Celkové schéma blokové soustavy v prostředí *SIMULINK* je následující:



Obr. 5.: Schéma simulované soustavy v prostředí *SIMULINK*

V obrázku 5 modrá část představuje identifikovanou soustavu s pevně danými parametry a zelená část soustavu s parametry určenými algoritmem. Šedá část pak představuje výpočet účelové funkce f_{cost}. Simulujeme dvoupolohovou regulaci. Blok *Scope* slouží k zobrazení grafu průběhů vstupního signálu a obou průběhů.

Simulace, vyhodnocení a porovnání bude nutné v každé iteraci provést zvlášť pro každou částici/agenta daného algoritmu. Z toho plyne, že větší množství agentů prodlouží výpočetní čas. Na druhou stranu, čím více agentů použijeme, tím více informací v každé iteraci získáme. Je tedy zřejmé, že počet agentů je nutné volit pečlivě.

Aby bylo možné porovnat kvality jednotlivých algoritmů při identifikaci systému, jsou pro všechny použity stejné vstupní hodnoty identifikované soustavy, stejný počet agentů a stejný vyhodnocovaný úsek. K určení kvality algoritmu bude použit vztah mezi kvalitou dosaženého řešení (tj. velikostí účelové funkce pro nejlepší nalezené řešení) a počtem potřebných iterací. Dalším faktorem, který bude sledován je rychlost konvergence algoritmu k ideálnímu řešení. Jelikož se hejnové algoritmy řadí mezi metaheuristické algoritmy, bude výpočet proveden vždy několikrát a dosažené výsledky budou porovnány.

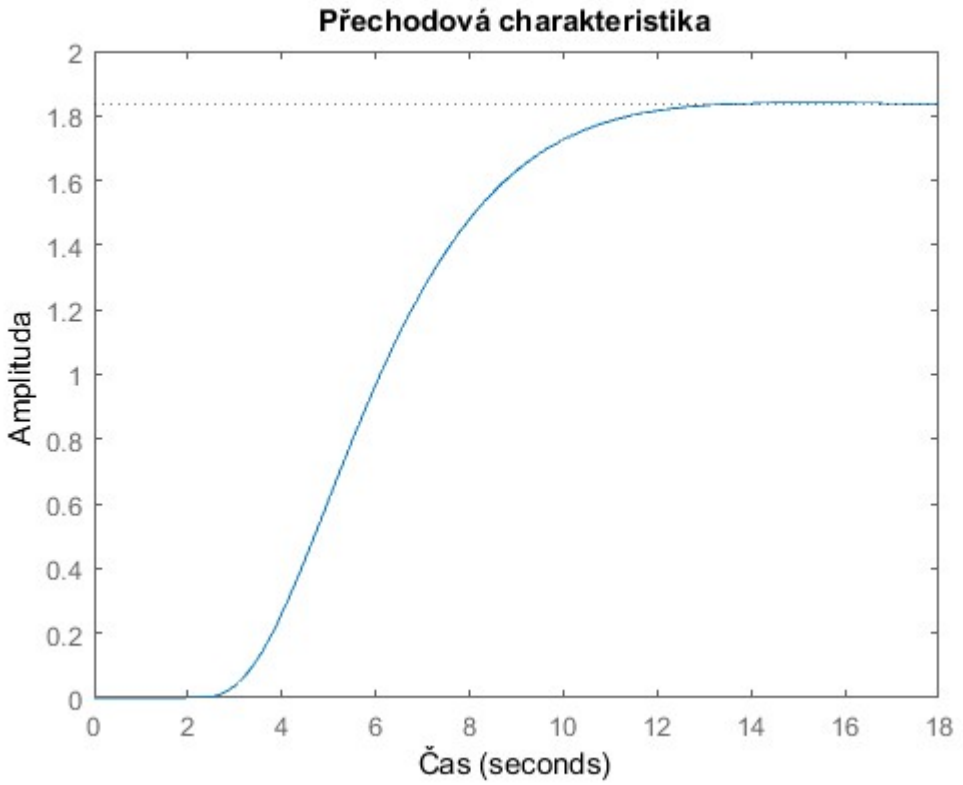
Před spuštěním optimalizace je nutné zvolit omezení hledaných parametrů. Například dopravní zpoždění τ_u a τ_y musí být z definice nezáporné. Pro hladší běh simulace v programu *MATLAB* je zároveň vhodné zamezit nulové hodnotě. Takže pro dopravní zpoždění je zvoleno $\tau_u > 0, \tau_y > 0$. Stejně tak z definice nemohou být záporné hodnoty τ_1 a τ_2. U těchto hodnot jsou připuštěny nulové hodnoty, jelikož nijak zásadně neovlivní běh programu. Platí tedy $\tau_1 \geq 0, \tau_2 \geq 0$.

Pro identifikaci byly z výše uvedených algoritmů vybrány netopýří algoritmus (*BA*), algoritmus světlušek (*FA*) a optimalizace hejnem částic (*PSO*). Výsledky budou porovnány s identifikací provedenou pomocí diferenciální evoluce (*DE*) jakožto zástupcem evolučních algoritmů. Ostatní algoritmy nebyly voleny, protože jsou vhodnější pro diskrétní kombinatorické problémy.

3.2. Identifikace simulovaného nekmitavého systému

Jako nekmitavý systém byla k identifikaci zvolena soustava čtvrtého řádu s dopravním zpožděním na vstupu daná přenosem:

$$G(s) = \frac{2{,}2 \exp(-2\,s)}{s^4 + 4{,}5\,s^3 + 8\,s^2 + 5\,s + 1{,}2} \tag{21}$$



Obr. 6.: Přechodová charakteristika soustavy (21)

Pro identifikaci nekmitavého systému bylo u všech algoritmů použito 40 agentů a požadovaná hodnota účelové funkce $\boldsymbol{f}_{cost} \leq 0{,}1$. Vyhodnocovaný úsek byl 20 sekund. Každý algoritmus byl aplikován desetkrát. Ukončovací podmínkou je nalezení dostatečně kvalitního řešení, případně provedení 40 iterací bez nalezení lepšího řešení (k čemuž může dojít v případě uvíznutí v lokálním extrému).

3.2.1. Identifikace netopýřím algoritmem

Pro identifikaci nekmitavé soustavy pomocí netopýřího algoritmu se jako nejlepší nastavení volitelných parametrů algoritmu ukázala následující volba:

$f_{min} = 0$ $\quad A^0 = 0{,}25$ $\quad \alpha = 0{,}9$

$f_{max} = 2$ $\quad r^0 = 0{,}5$ $\quad \gamma = 0{,}7$

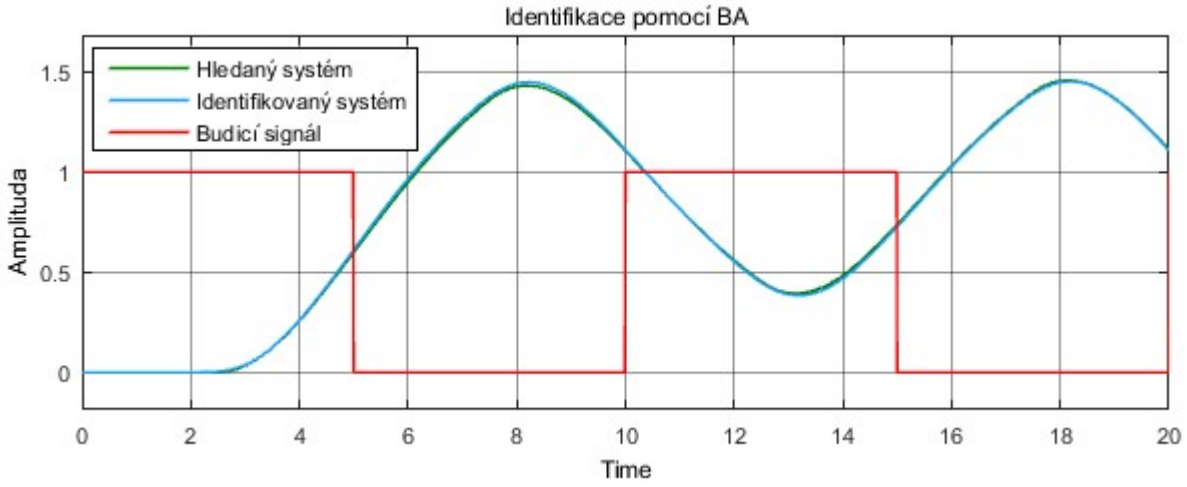
S těmito parametry byl netopýří algoritmus schopný velice rychle konvergovat k optimálnímu řešení. Další úpravy algoritmu již nebyly nutné.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	8	37,4	0,081	4,68
2	10	45,7	0,091	4,57
3	20	87,1	0,088	4,35
4	24	104,8	0,090	4,37
5	7	33,3	0,038	4,76
6	12	54,4	0,034	4,54
7	4	20,5	0,086	5,13
8	25	108,6	0,086	4,34
9	7	33,3	0,088	4,75
10	3	16,0	0,025	5,34
Ø	**12**	**54,1**	**0,071**	**4,68**

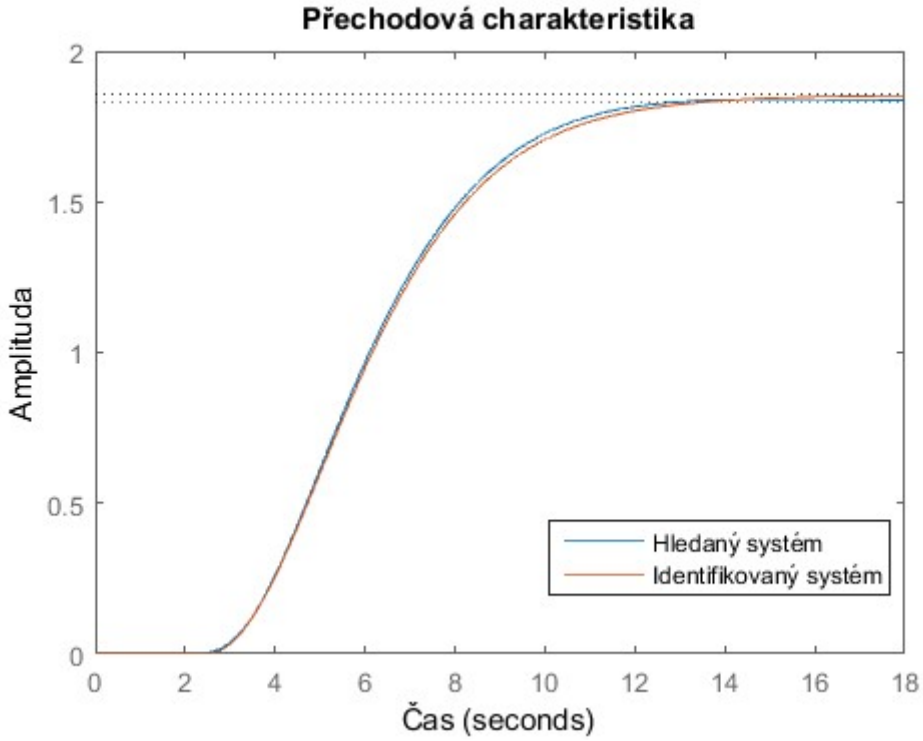
Tabulka 1.: Identifikace nekmitavé soustavy pomocí netopýřího algoritmu

Nejlepší řešení nalezené pomocí netopýřího algoritmu je řešení číslo 10 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}025$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}8548\ \exp(-2{,}5031s)}{(2{,}0063s + 1)(3{,}1892s + \exp(-1{,}3174s))} \tag{22}$$



Obr. 7.: Porovnání odezvy na vstupní signál soustav (21) a (22)



Obr. 8.: Porovnání přechodových charakteristik soustav (21) a (22)

3.2.2. Identifikace algoritmem světlušek

Při identifikaci nekmitavé soustavy pomocí algoritmu světlušek bylo použito následující nastavení parametrů:

$$\alpha_0 = 0{,}9 \qquad \beta_0 = 2 \qquad \beta_{min} = 0{,}4 \qquad \gamma = 1{,}4$$

$$\alpha = \alpha_0 \cdot 0{,}96^t \qquad \beta = (\beta_0 - \beta_{min})\, e^{-\gamma r^2} + \beta_{min}$$

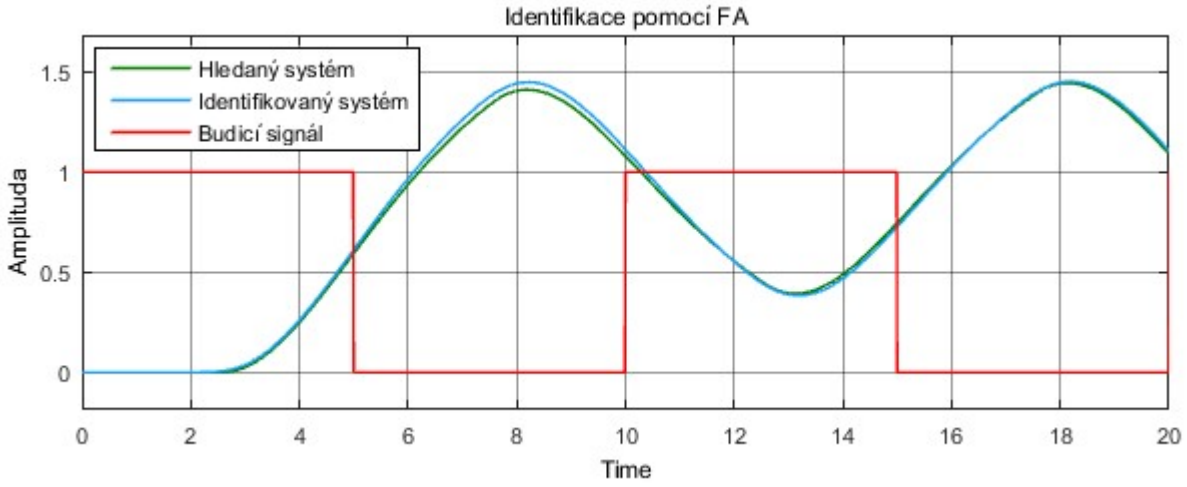
Při použití algoritmu světlušek se ukázalo vhodné použít proměnnou délku náhodného kroku v závislosti na dosažené kvalitě řešení. Čím nižší hodnoty účelové funkce bylo dosaženo, tím kratší byl použit náhodný krok.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	17	75,3	0,097	4,43
2	10	45,1	0,054	4,51
3	12	53,3	0,070	4,44
4	14	62,6	0,095	4,47
5	22	91,5	0,068	4,16
6	13	62,7	0,076	4,82
7	10	45,7	0,090	4,57
8	15	65,6	0,096	4,38
9	9	39,9	0,077	4,43
10	18	78,1	0,086	4,34
Ø	**14**	**62,0**	**0,081**	**4,45**

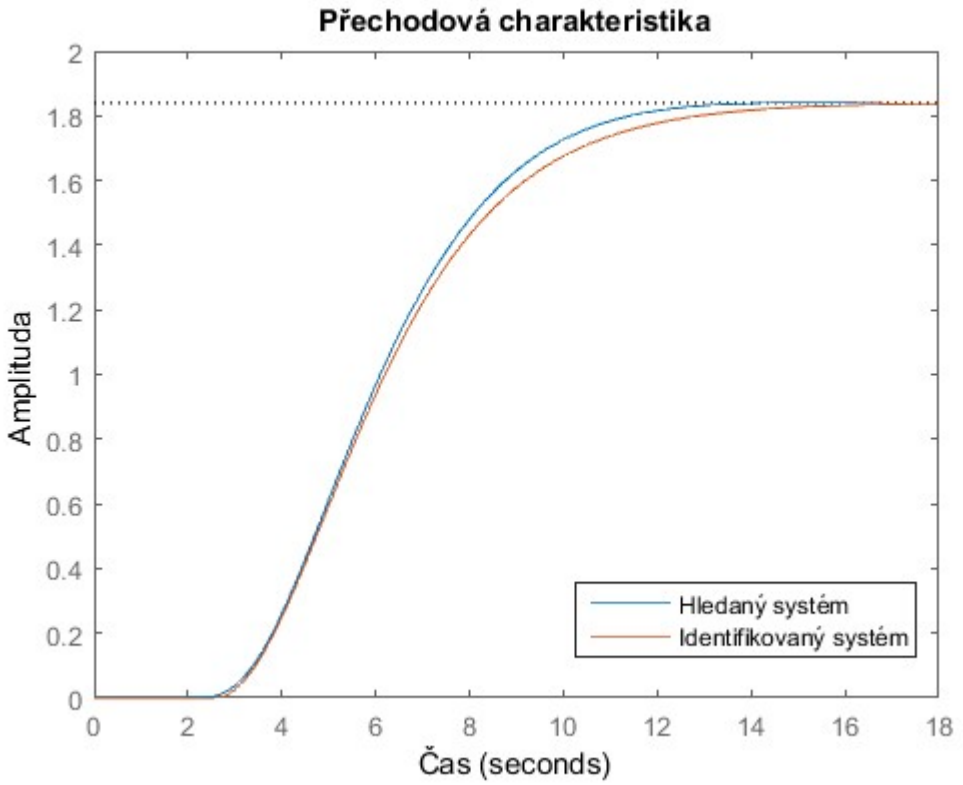
Tabulka 2.: Identifikace nekmitavé soustavy pomocí algoritmu světlušek

Nejlepší řešení nalezené pomocí algoritmu světlušek je řešení číslo 2 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}054$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}8412\ \exp(-2{,}5793s)}{(1{,}9231s + 1)(3{,}0669s + \exp(-1{,}1170s))} \tag{23}$$



Obr. 9.: Porovnání odezvy na vstupní signál soustav (21) a (23)



Obr. 10.: Porovnání přechodových charakteristik soustav (21) a (23)

3.2.3. Identifikace optimalizací hejnem částic

Při identifikaci nekmitavé soustavy pomocí optimalizace hejnem částic bylo použito následující nastavení parametrů:

$$\alpha = 1{,}5 \qquad \beta = 1{,}8$$

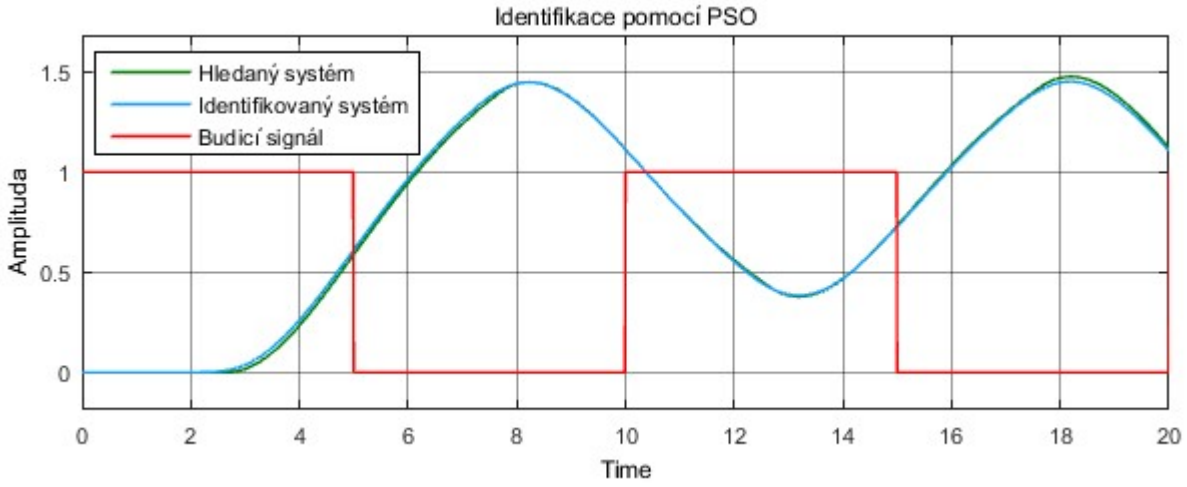
Při identifikaci nekmitavé soustavy se optimalizace hejnem částic ukázala nejméně citlivá na nastavení svých řídících parametrů. Současně u ní nebylo nutné použít úpravu parametrů v závislosti na kvalitě dosavadního optimálního řešení. I přes to algoritmus fungoval rychle a stabilně.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	5	24,6	0,068	4,93
2	11	48,7	0,069	4,43
3	18	77,4	0,064	4,30
4	11	49,0	0,093	4,46
5	14	61,2	0,076	4,37
6	5	24,6	0,067	4,91
7	16	69,6	0,062	4,35
8	18	74,7	0,022	4,15
9	19	78,3	0,088	4,12
10	16	85,0	0,096	5,31
Ø	**13**	**59,3**	**0,071**	**4,53**

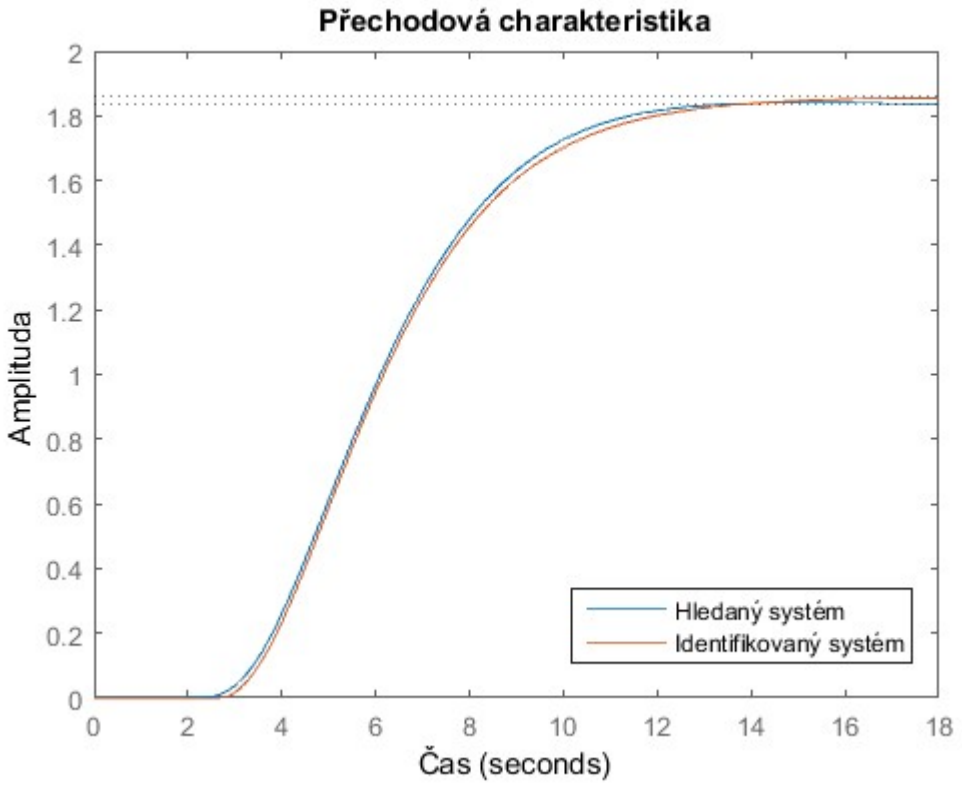
Tabulka 3.: Identifikace nekmitavé soustavy pomocí optimalizace hejnem částic

Nejlepší řešení nalezené pomocí optimalizace hejnem částic je řešení číslo 8 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}022$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}8579\ \exp(-2{,}6720s)}{(1{,}7797s + 1)(3{,}1123s + \exp(-1{,}1335s))} \tag{24}$$



Obr. 11.: Porovnání odezvy na vstupní signál soustav (21) a (24)



Obr. 12.: Porovnání přechodových charakteristik soustav (21) a (24)

3.2.4. Identifikace diferenciální evolucí

Při identifikaci nekmitavé soustavy pomocí diferenciální evoluce bylo použito následující nastavení parametrů:

$$F = 0{,}4 \qquad C_r = 0{,}9$$

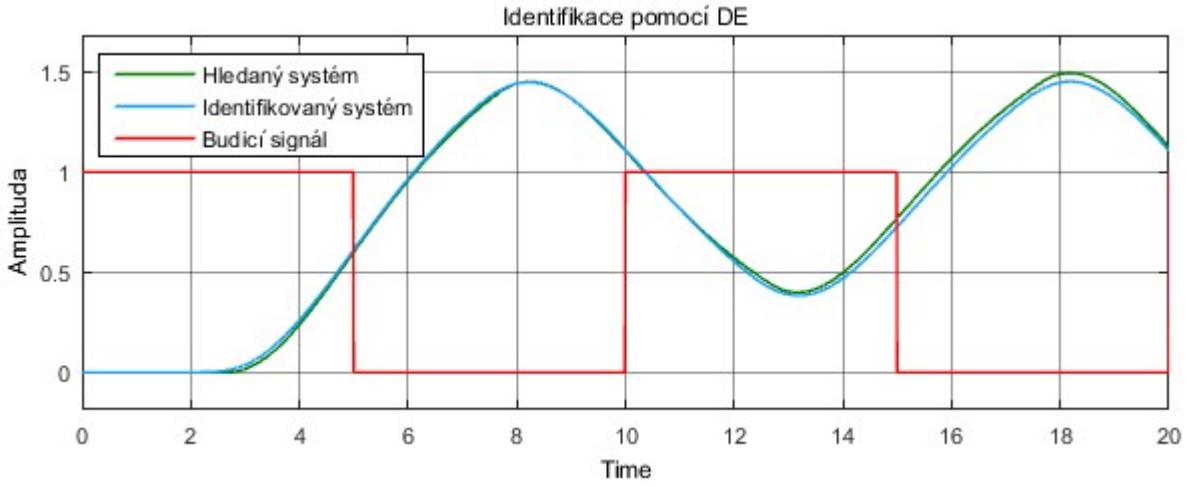
Ani u diferenciální evoluce nebylo třeba upravovat parametry v průběhu běhu výpočtu. Algoritmus stabilně konvergoval k výsledkům s žádanou hodnotou účelové funkce.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	9	39,3	0,058	4,36
2	24	97,8	0,094	4,07
3	16	66,9	0,087	4,18
4	16	67,2	0,078	4,20
5	16	67,4	0,088	4,21
6	26	101,1	0,080	3,89
7	9	39,3	0,095	4,37
8	9	39,4	0,079	4,38
9	24	98,6	0,099	4,11
10	17	71,4	0,041	4,20
Ø	**17**	**68,8**	**0,080**	**4,20**

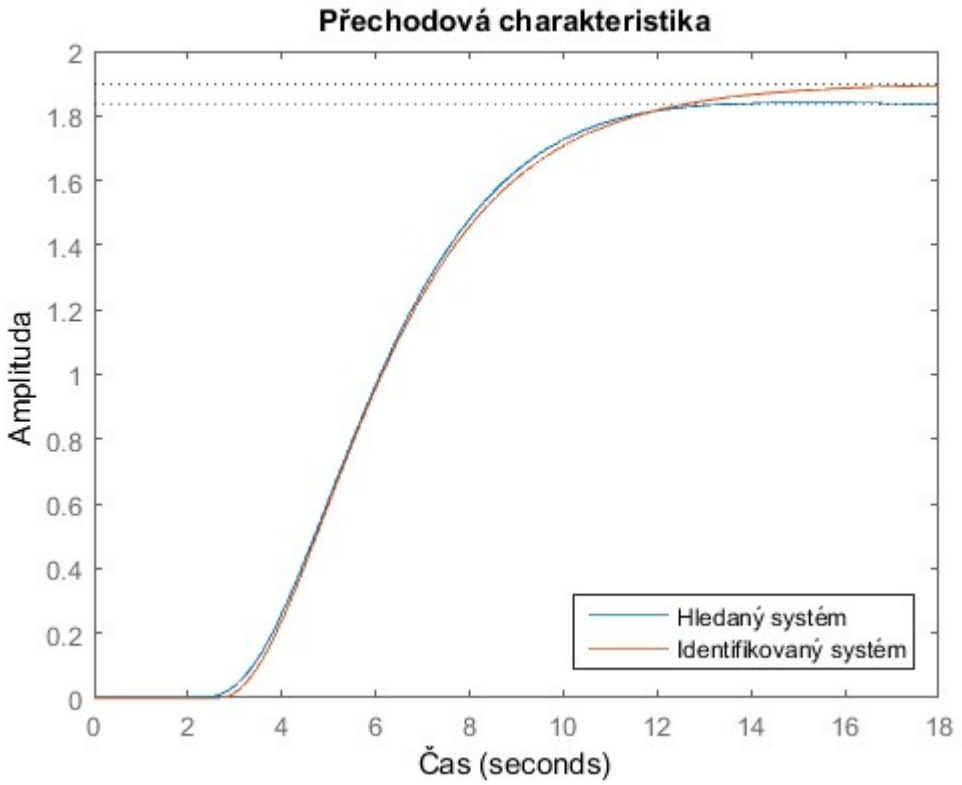
Tabulka 4.: Identifikace nekmitavé soustavy pomocí diferenciální evoluce

Nejlepší řešení nalezené pomocí diferenciální evoluce je řešení číslo 10 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}041$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}900\ \exp(-2{,}6829s)}{(2{,}3057s + 1)(2{,}4849s + \exp(-0{,}9050s))} \tag{25}$$



Obr. 13.: Porovnání odezvy na vstupní signál soustav (21) a (25)



Obr. 14.: Porovnání přechodových charakteristik soustav (21) a (25)

3.2.5. Výsledky identifikace

Při identifikaci nekmitavého systému všechny testované algoritmy opakovaně dosáhly požadované hodnoty účelové funkce $\boldsymbol{f}_{cost}$ i bez nutnosti předčasného ukončení algoritmu z důvodu uvíznutí. V tabulkách 1. – 4. se lze přesvědčit, že průměrná hodnota účelové funkce je u nalezených řešení lepší, než bylo požadováno.

Při porovnání času potřebného k provedení jedné iterace vychází nejrychleji diferenciální evoluce. Algoritmus světlušek a optimalizace hejnem částic dosáhli podobných výsledků přibližně o $0{,}3\ s$ pomalejších než diferenciální evoluce. Netopýří algoritmus byl oproti diferenciální evoluci pomalejší o $0{,}5\ s$.

Porovnáme-li počet iterací potřebných k dosažení výsledku s požadovanou hodnotou účelové funkce (rychlost konvergence), je nejrychlejší netopýří algoritmus, který daný výsledek určily v průměru po 12 iteracích. Optimalizace hejnem částic řešení určila po 13 iteracích a algoritmus světlušek v průměru po 14 iteracích. Diferenciální

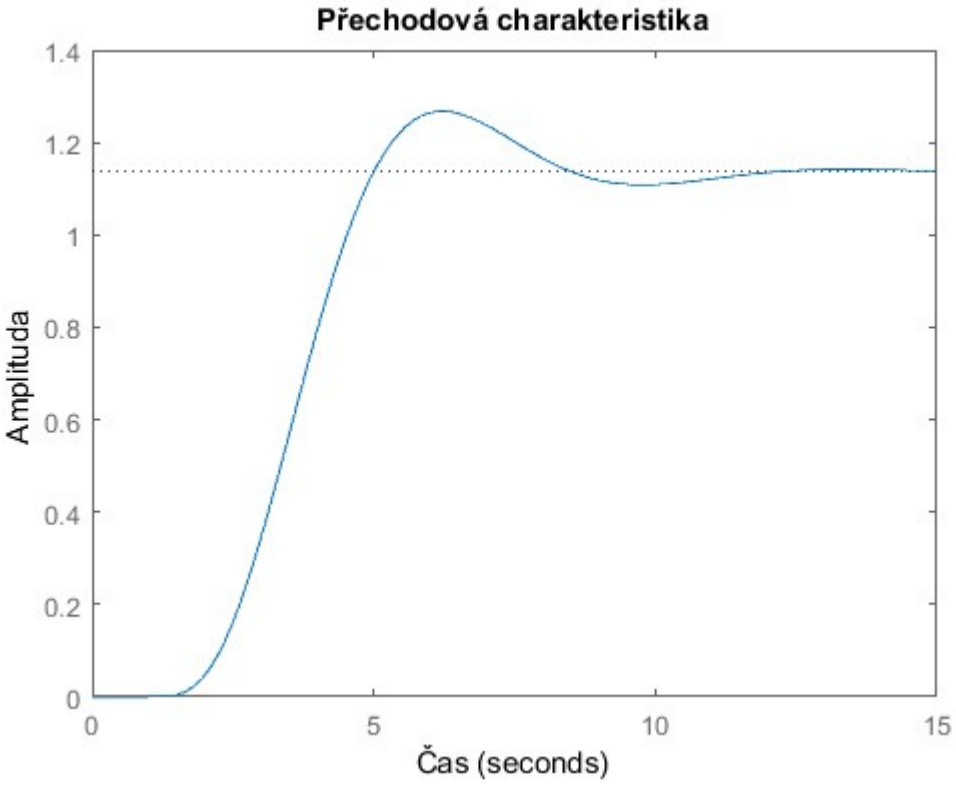
evoluce dosáhla výsledku v průměru po 17 iteracích. Je tedy vidět, že všechny hejnové algoritmy jsou velmi vyrovnané.

Netopýří algoritmus měl sice nejdelší čas na provedení jedné iterace, ale jelikož potřeboval méně iterací k nalezení výsledku, byl celkový čas nejkratší, těsně pod jednu minutu. Optimalizace hejnem částic a algoritmus světlušek dosáhli řešení obvykle po jedné minutě. Identifikace diferenciální evolucí trvala přibližně o $10\ s$ déle. I tady je zjevná vyrovnanost použitých algoritmů.

3.3. Identifikace simulovaného kmitavého systému

Jako kmitavý systém byla k identifikaci použita soustava čtvrtého řádu s dopravním zpožděním na vstupu daná následujícím přenosem:

$$G(s) = \frac{2{,}5\exp(-s)}{s^4 + 4\,s^3 + 6\,s^2 + 5\,s + 2{,}2} \tag{26}$$



Obr. 15.: Přechodová charakteristika soustavy (26)

Pro identifikaci kmitavého systému bylo u všech algoritmů opět použito 40 agentů a vyhodnocovaný úsek byl 20 sekund. Dále byla opět použita požadovaná hodnota účelové funkce $\boldsymbol{f}_{cost} \leq 0{,}1$. Navíc byla do algoritmů přidána podmínka kmitavosti systému, tj. $\tau_y/\tau_2 > 1/e$.

3.3.1. Identifikace netopýřím algoritmem

Pro identifikaci kmitavé soustavy pomocí netopýřího algoritmu se jako nejlepší nastavení volitelných parametrů algoritmu ukázala následující volba:

$$f_{min} = 0 \qquad A^0 = 0{,}25 \qquad \alpha = 0{,}9$$

$$f_{max} = 2 \qquad r^0 = 0{,}5 \qquad \gamma = 0{,}7$$

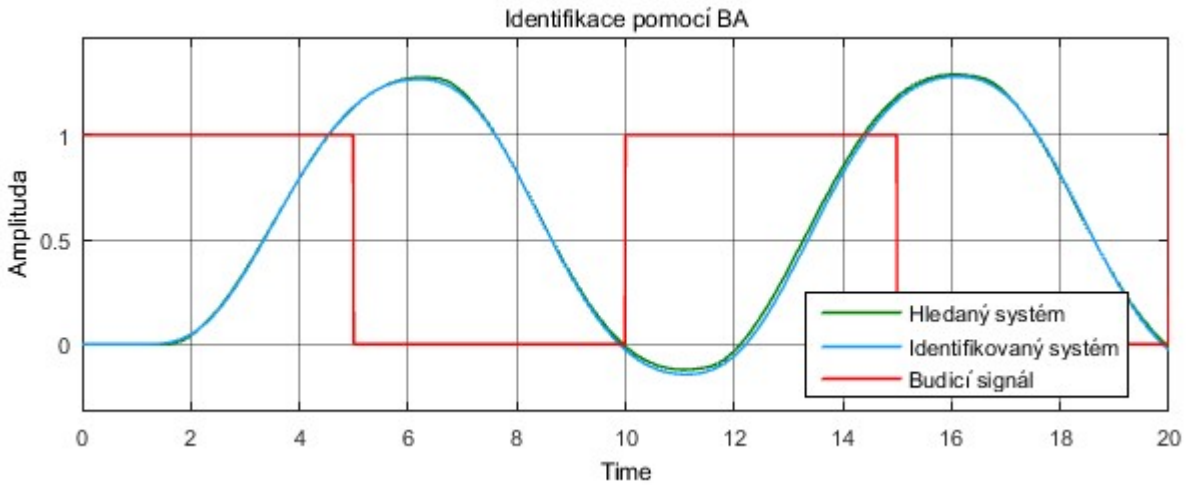
Při identifikaci kmitavé soustavy se u netopýřího algoritmu projevila nejmenší potřeba měnit řídicí parametry algoritmu. Tentokrát se ovšem ukázalo výhodné použít proměnnou délku náhodného kroku v závislosti na kvalitě nejlepšího nalezeného řešení.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	29	96,9	0,091	3,34
2	5	23,3	0,095	4,67
3	15	62,6	0,083	4,17
4	34	136,9	0,075	4,03
5	5	23,8	0,093	4,76
6	19	80,3	0,049	4,22
7	4	20,7	0,072	5,17
8	10	44,2	0,098	4,42
9	22	89,2	0,083	4,05
10	6	27,5	0,088	4,58
Ø	**15**	**60,5**	**0,083**	**4,34**

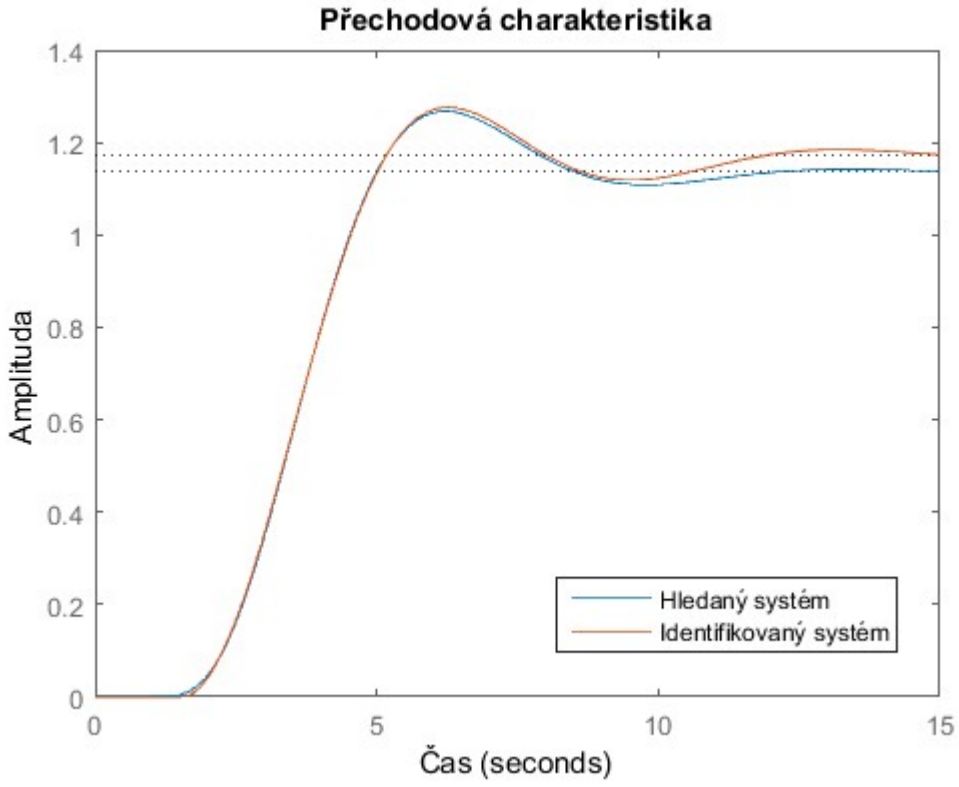
Tabulka 5.: Identifikace kmitavé soustavy pomocí netopýřího algoritmu

Nejlepší řešení nalezené pomocí netopýřího algoritmu je řešení číslo 6 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}049$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}1711\ \exp(-1{,}5417s)}{(1{,}5893s + 1)(1{,}6794s + \exp(-1{,}3642s))} \tag{27}$$



Obr. 16.: Porovnání odezvy na vstupní signál soustav (26) a (27)



Obr. 17.: Porovnání přechodových charakteristik soustav (26) a (27)

3.3.2. Identifikace algoritmem světlušek

Při identifikaci kmitavé soustavy pomocí algoritmu světlušek bylo použito následující nastavení parametrů:

$\alpha_0 = 0{,}8$ $\beta_0 = 2$ $\beta_{min} = 0{,}1$ $\gamma = 1{,}5$

$$\alpha = \alpha_0 \cdot 0{,}97^t \qquad \beta = (\beta_0 - \beta_{min})\, e^{-\gamma r^2} + \beta_{min}$$

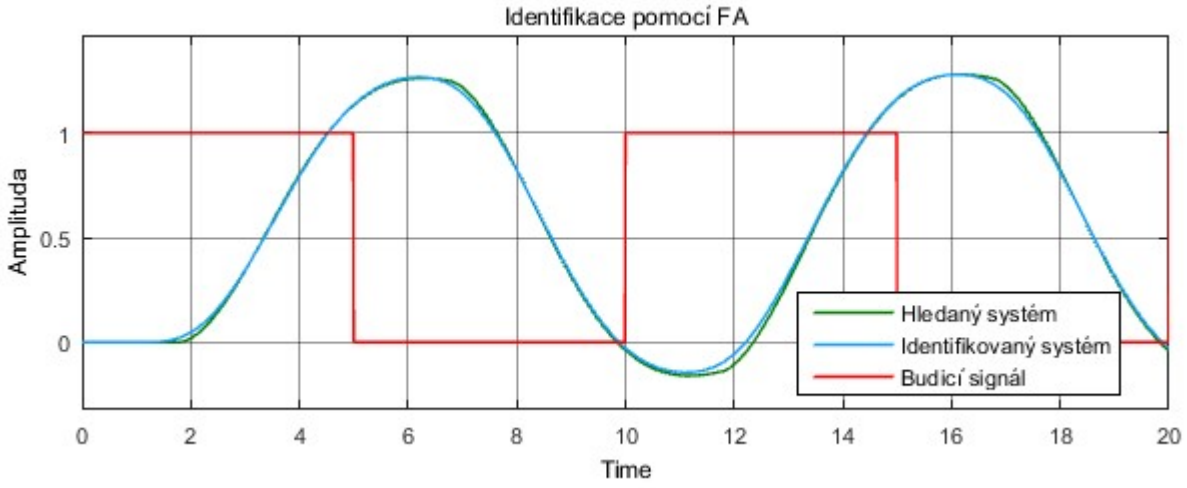
Při identifikaci kmitavé soustavy pomocí algoritmu světlušek byl opět použitý proměnný náhodný krok.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	8	36,9	0,093	4,62
2	14	61,7	0,089	4,41
3	14	61,5	0,060	4,39
4	26	107,7	0,097	4,14
5	17	71,5	0,077	4,21
6	19	79,7	0,065	4,19
7	24	187,2	0,097	7,80
8	12	53,7	0,074	4,48
9	18	78,0	0,100	4,33
10	16	76,3	0,067	4,77
Ø	**17**	**81,4**	**0,082**	**4,73**

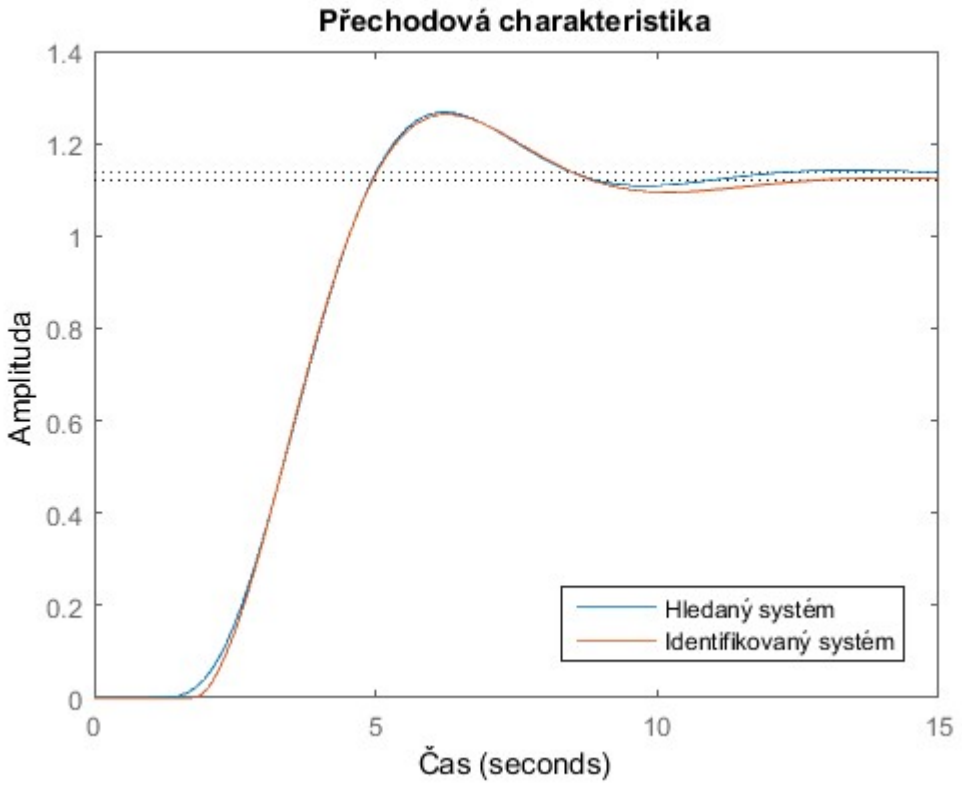
Tabulka 6.: Identifikace kmitavé soustavy pomocí algoritmu světlušek

Nejlepší řešení nalezené pomocí algoritmu světlušek je řešení číslo 3 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}060$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}1202\ \exp(-1{,}7546s)}{(0{,}8176s + 1)(1{,}9897s + \exp(-1{,}3409s))} \tag{28}$$



Obr. 18.: Porovnání odezvy na vstupní signál soustav (26) a (28)



Obr. 19.: Porovnání přechodových charakteristik soustav (26) a (28)

3.3.3. Identifikace optimalizací hejnem částic

Při identifikaci kmitavé soustavy pomocí optimalizace hejnem částic bylo použito následující nastavení parametrů:

$$\alpha = 1{,}2 \qquad\qquad \beta = 1{,}9$$

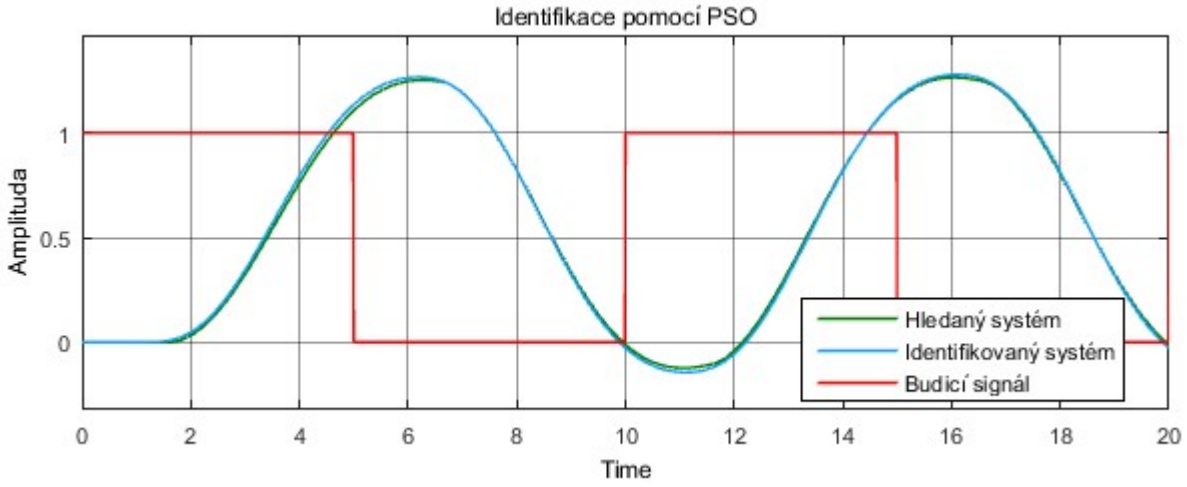
Na rozdíl od identifikace nekmitavé soustavy pomocí optimalizace hejnem částic, se u identifikace kmitavé soustavy projevila určitá citlivost na nastavení řídicích parametrů. Při použití výše uvedených hodnot algoritmus opět rychle a stabilně konvergoval k požadovanému řešení. Tento algoritmus nebylo nutné nijak dál upravovat.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	12	55,3	0,087	4,61
2	7	35,1	0,094	5,02
3	15	68,2	0,059	4,54
4	6	28,5	0,050	4,75
5	21	97,0	0,095	4,62
6	13	62,9	0,082	4,84
7	29	131,8	0,074	4,55
8	27	119,6	0,097	4,43
9	33	144,8	0,096	4,39
10	27	119,1	0,098	4,41
Ø	**19**	**86,2**	**0,083**	**4,61**

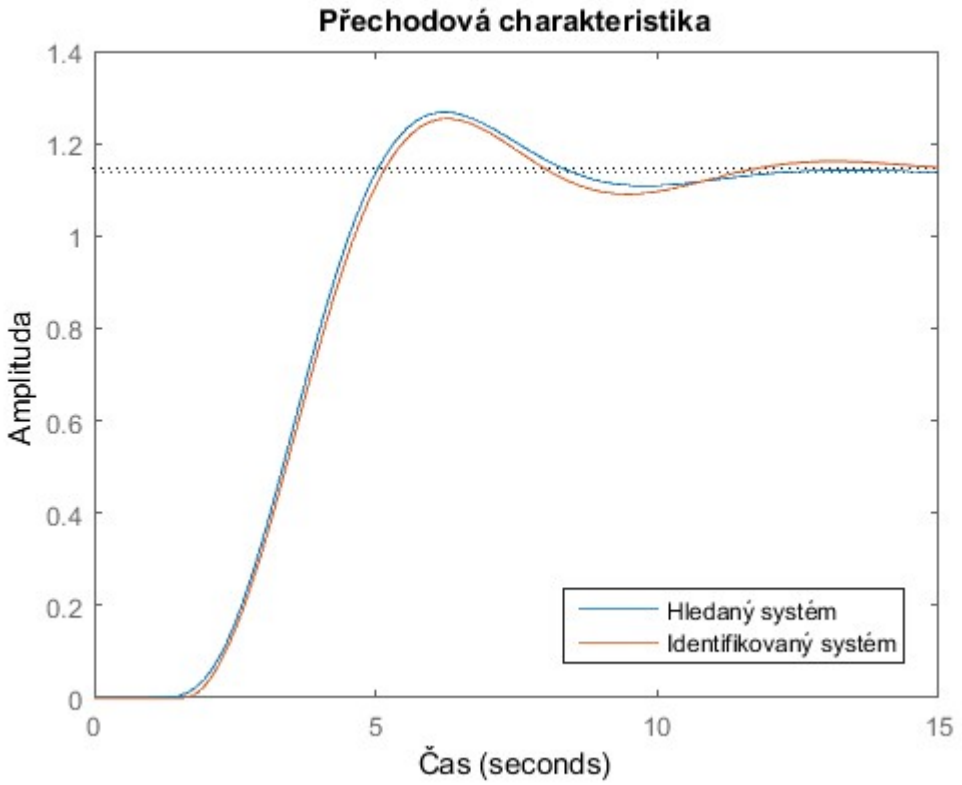
Tabulka 7.: Identifikace kmitavé soustavy pomocí optimalizace hejnem částic

Nejlepší řešení nalezené pomocí optimalizace hejnem částic je řešení číslo 4 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}050$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}1464\ \exp(-1{,}6038s)}{(1{,}5962s + 1)(1{,}6480s + \exp(-1{,}3611s))} \tag{29}$$



Obr. 20.: Porovnání odezvy na vstupní signál soustav (26) a (29)



Obr. 21.: Porovnání přechodových charakteristik soustav (26) a (29)

3.3.4. Identifikace diferenciální evolucí

Při identifikaci kmitavé soustavy pomocí diferenciální evoluce bylo použito následující nastavení parametrů:

$$F = 0{,}4 \qquad C_r = 0{,}9$$

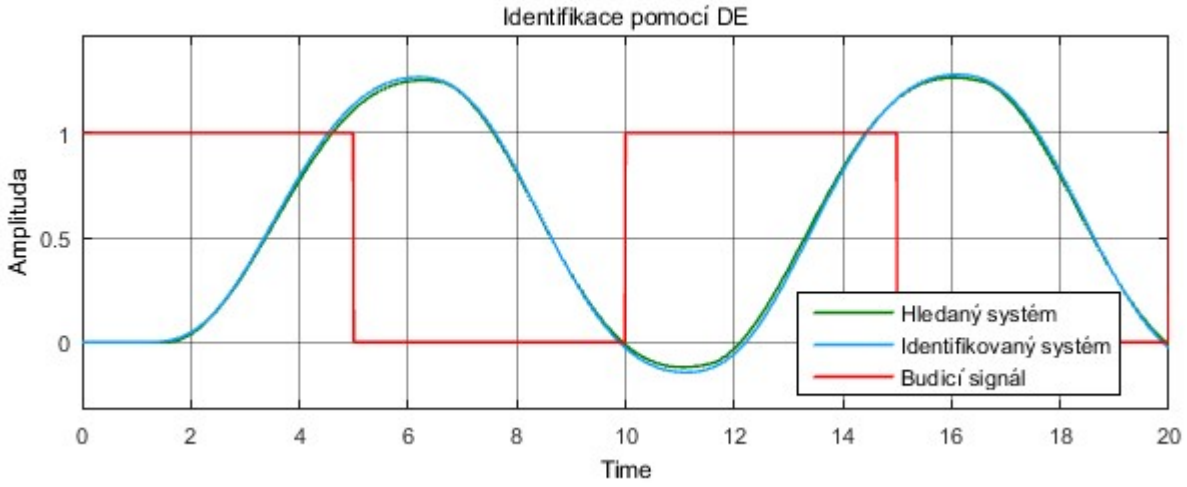
Podobně jako u netopýřího algoritmu nebylo nutné měnit parametry algoritmu a i přes to algoritmus při identifikaci kmitavé soustavy dobře konvergoval k řešení s požadovanou přesností. Parametry nebylo nutné v průběhu identifikace nijak upravovat.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	16	62,8	0,082	3,93
2	8	38,5	0,064	4,81
3	27	119,3	0,100	4,42
4	26	114,9	0,078	4,42
5	12	55,6	0,073	4,64
6	21	93,7	0,094	4,46
7	29	127,6	0,096	4,40
8	16	72,2	0,088	4,51
9	20	89,4	0,089	4,47
10	28	123,4	0,051	4,41
Ø	**20**	**89,8**	**0,081**	**4,45**

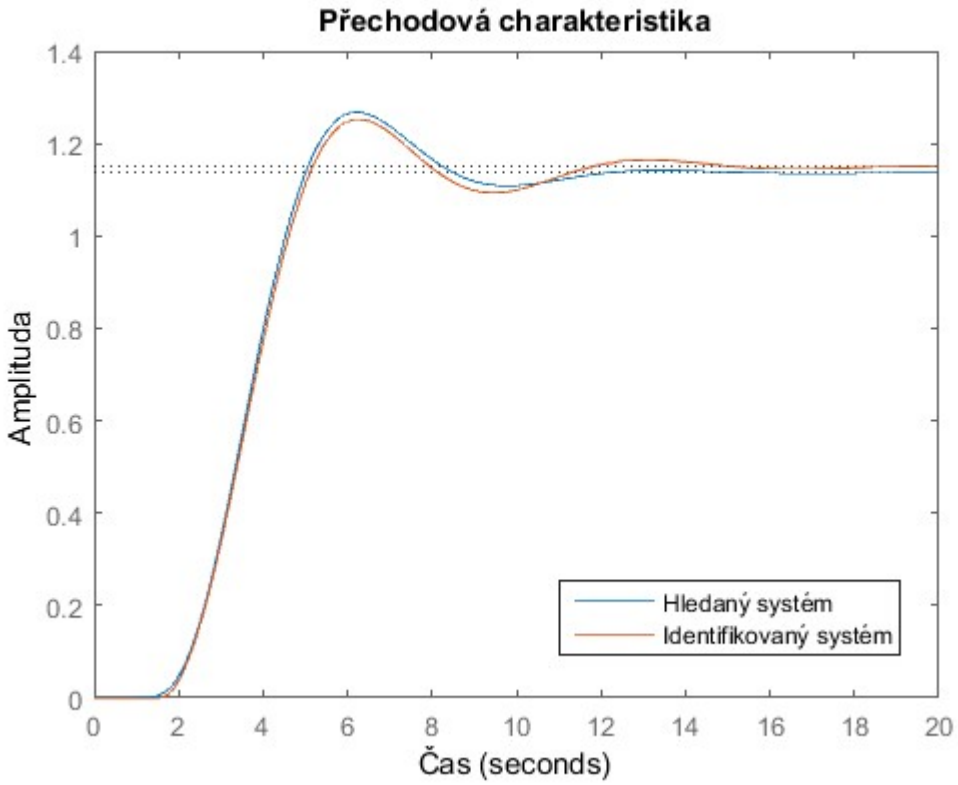
Tabulka 8.: Identifikace kmitavé soustavy pomocí diferenciální evoluce

Nejlepší řešení nalezené pomocí diferenciální evoluce je řešení číslo 10 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}051$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}1497\ \exp(-1{,}5637s)}{(1{,}6278s + 1)(1{,}6503s + \exp(-1{,}3654s))} \tag{30}$$



Obr. 22.: Porovnání odezvy na vstupní signál soustav (26) a (30)



Obr. 23.: Porovnání přechodových charakteristik soustav (29) a (30)

3.3.5. Výsledky identifikace

Stejně jako u identifikace nekmitavého systému, i při identifikaci kmitavého systému všechny testované algoritmy ve všech případech dosáhly řešení s požadovanou přesností účelové funkce $\boldsymbol{f}_{cost}$ aniž by byl výpočet přerušen z důvodu uvíznutí, jak je možné se přesvědčit v tabulkách 5. – 8.

Při porovnání času potřebného k provedení jedné iterace byly v tomto případě algoritmy vyrovnanější. Tentokrát jednu iteraci nejrychleji provedly netopýří algoritmus a diferenciální evoluce s téměř shodnými časy. Algoritmus světlušek a optimalizace hejnem částic potřebovali přibližně o $0{,}3\ s$ delší čas k provedení a vyhodnocení jedné iterace.

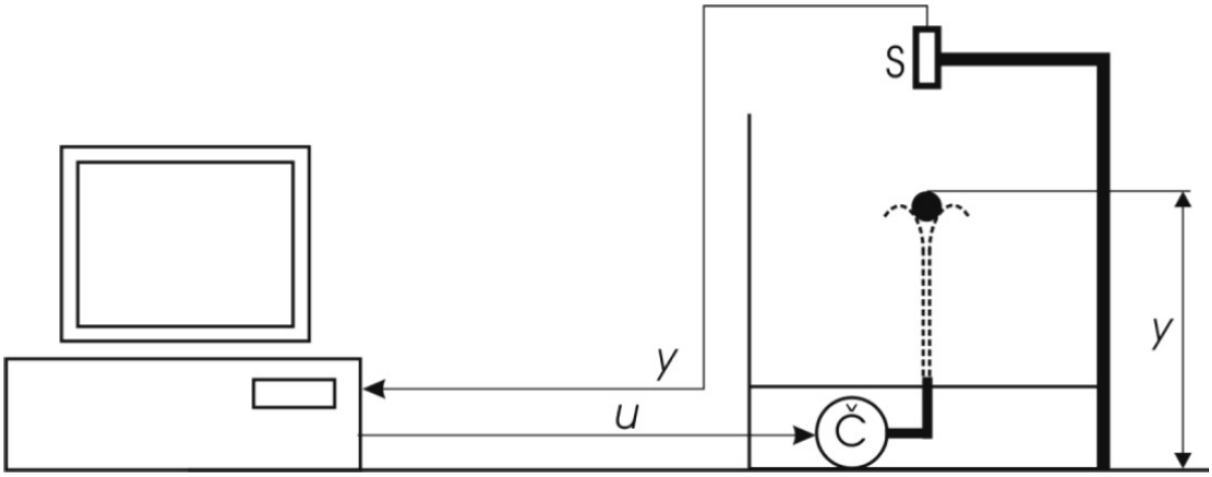
Všechny algoritmy potřebovaly k identifikaci kmitavého systému více iterací, než k identifikaci nekmitavého. U netopýřího algoritmu, algoritmu světlušek a diferenciální evoluce byl nárůst 3 iterace. U optimalizace hejnem částic bylo třeba o 6 iterací více.

Jelikož v tomto případě identifikace byly časy potřebné k provedení jedné iterace téměř totožné, jsou celkové časy potřebné k provedení výpočtu úměrné počtu potřebných iterací. Nejrychlejší byl opět netopýří algoritmus, který identifikaci provedl během 1 minuty. Ostatní použité algoritmy výpočet ukončili do 1,5 minuty.

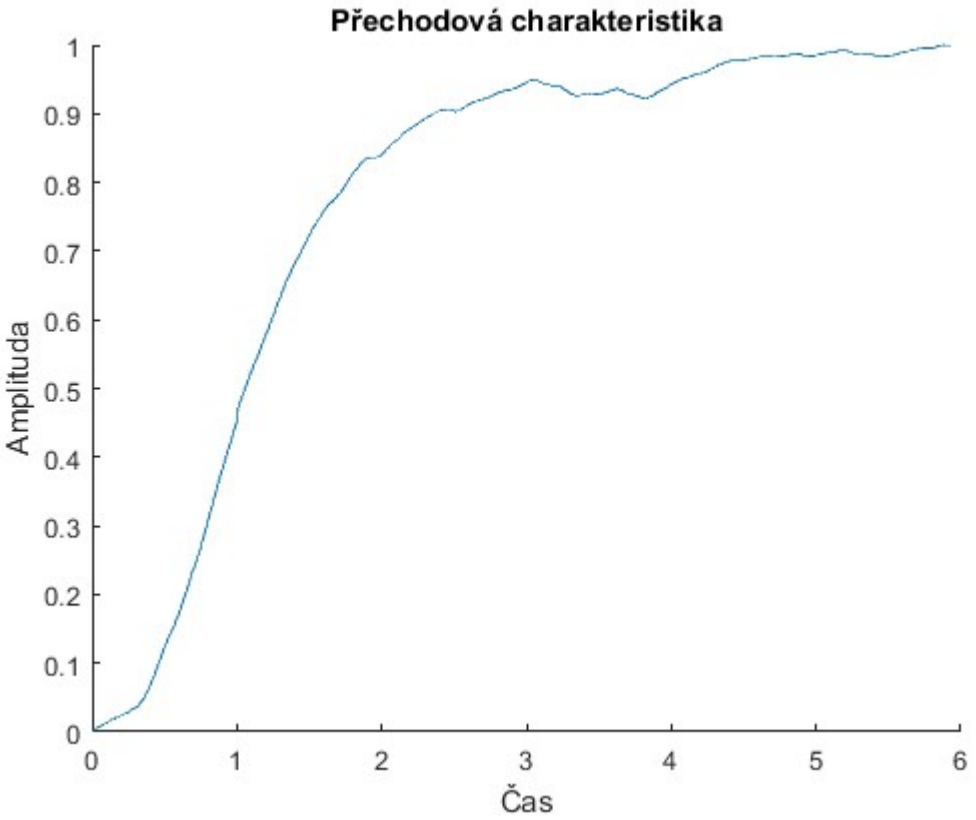
3.4. Identifikace reálného systému

K identifikaci reálného systému byla použita úloha vodní levitace. Jde o nekmitavou soustavu s malými vlivy rušení.

Úloha se skládá z vodní nádrže, ve které je čerpadlo. To nasává vodu a vhání ji do trysky. Průtok vody je ovládán z počítače signálem u. Proud vody z trysky ovlivňuje polohu míčku levitujícího na vodním sloupci. Poloha míčku y je snímána čidlem.



Obr. 24.: Schéma vodní levitace (převzato z [12] str. 1)



Obr. 25.: Přechodová charakteristika systému vodní levitace

Jelikož se míček na vodním sloupci pohybuje částečně nepředvídatelným způsobem, je výstupní signál y zašuměný a průběhy odezvy nejsou pravidelné a hladké jako u simulovaného systému. S přihlédnutím k této skutečnosti je nutné volit požadovanou hodnotu účelové funkce vyšší, než bylo použito u simulovaných soustav. Konkrétně byla zvolena hodnota $\boldsymbol{f}_{cost} \leq 0{,}8$. Ostatní parametry identifikace (tj. počet použitých agentů a délka vyhodnocovaného úseku) byly zachovány.

3.4.1. Identifikace netopýřím algoritmem

Pro identifikaci reálné soustavy pomocí netopýřího algoritmu se jako nejlepší nastavení volitelných parametrů algoritmu ukázala následující volba:

$f_{min} = 0$ $\quad A^0 = 0{,}35$ $\quad \alpha = 0{,}9$

$f_{max} = 1{,}8$ $\quad r^0 = 0{,}8$ $\quad \gamma = 0{,}9$

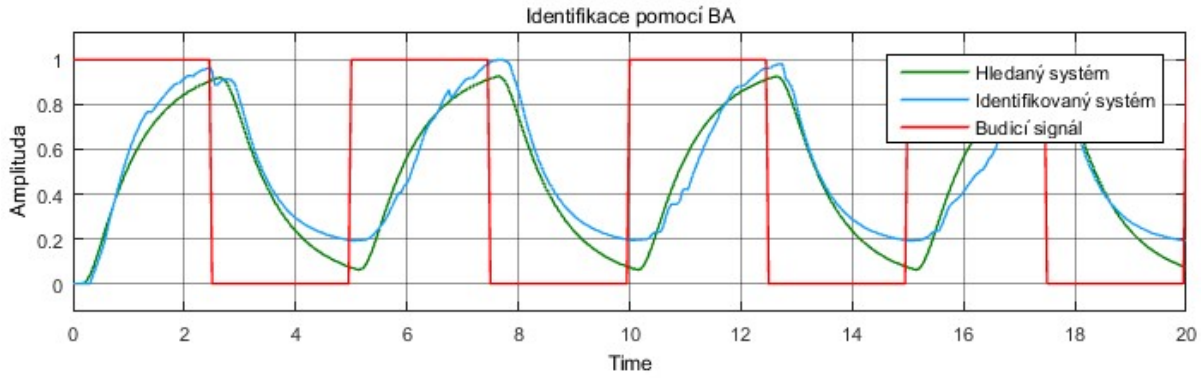
Při identifikaci reálné soustavy nebylo nutné použít proměnnou délku náhodného kroku. S použitím výše uvedených parametrů netopýří algoritmus rychle konvergoval.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	8	14,8	0,792	1,85
2	14	24,2	0,738	1,73
3	10	18,1	0,699	1,81
4	18	30,5	0,575	1,70
5	14	24,5	0,723	1,75
6	19	32,5	0,734	1,71
7	12	21,0	0,766	1,75
8	14	24,5	0,709	1,75
9	25	42,9	0,719	1,71
10	21	36,4	0,751	1,73
Ø	**16**	**26,9**	**0,721**	**1,75**

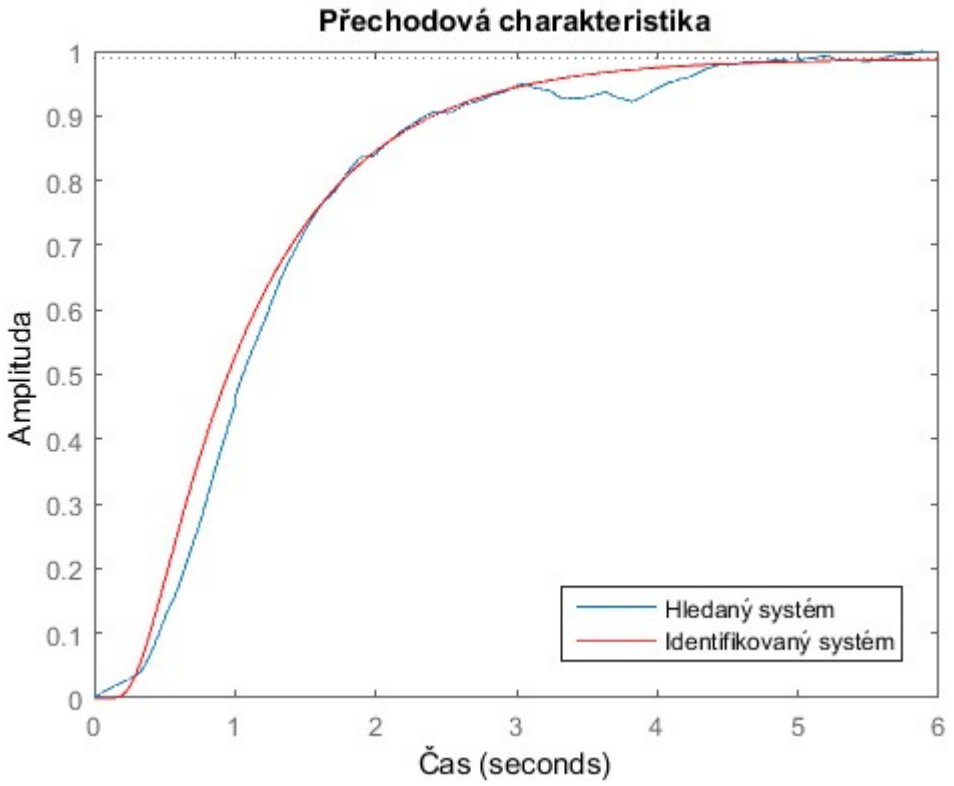
Tabulka 9.: Identifikace reálné soustavy pomocí netopýřího algoritmu

Nejlepší řešení nalezené pomocí netopýřího algoritmu je řešení číslo 4 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}575$, kterému odpovídá anisochronní model:

$$G(s) = \frac{0{,}9880\ \exp(-0{,}1575s)}{(0{,}8513s + 1)(0{,}2867s + \exp(-0{,}1065s))} \tag{31}$$



Obr. 26.: Porovnání odezvy na vstupní signál vodní levitace a soustavy (31)



Obr. 27.: Porovnání přechodových charakteristik vodní levitace a soustavy (31)

3.4.2. Identifikace algoritmem světlušek

Při identifikaci reálné soustavy pomocí algoritmu světlušek bylo použito následující nastavení parametrů:

$$\alpha_0 = 0{,}8 \qquad \beta_0 = 2{,}2 \qquad \beta_{min} = 0{,}5 \qquad \gamma = 1{,}5$$

$$\alpha = \alpha_0 \cdot 0{,}95^t \qquad \beta = (\beta_0 - \beta_{min})\, e^{-\gamma r^2} + \beta_{min}$$

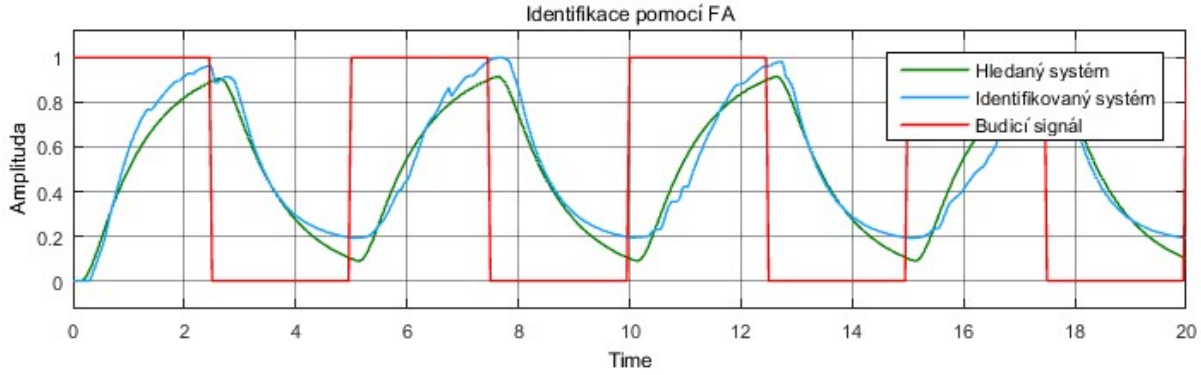
Při identifikaci reálné soustavy algoritmus světlušek neprojevoval přílišnou citlivost na nastavení řídicích parametrů. Zároveň nebylo nutné použít proměnnou délku náhodného kroku. I přes to konvergoval skutečně velmi rychle a byl schopný dosáhnout lepších výsledků než ostatní použité algoritmy.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	6	11,8	0,620	1,96
2	4	8,4	0,644	2,09
3	8	14,9	0,789	1,86
4	10	17,8	0,762	1,78
5	8	14,5	0,497	1,82
6	3	7,0	0,767	2,32
7	7	13,9	0,655	1,99
8	4	8,3	0,632	2,06
9	6	11,8	0,552	1,96
10	4	8,2	0,754	2,06
Ø	**6**	**11,6**	**0,667**	**1,99**

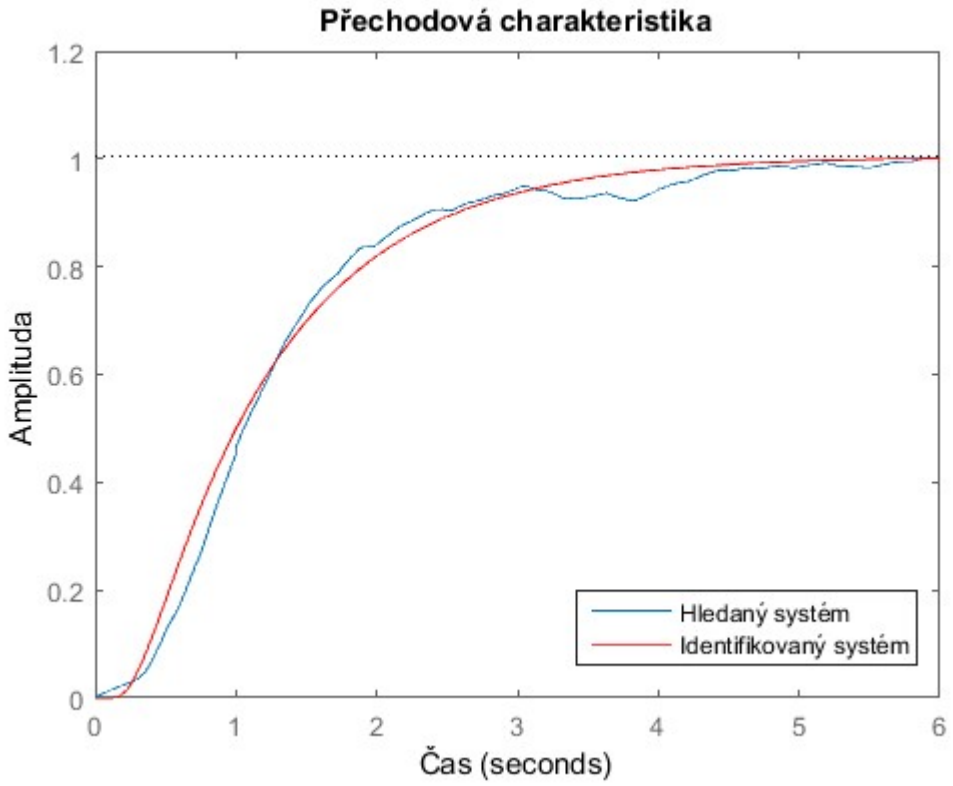
Tabulka 10.: Identifikace reálné soustavy pomocí algoritmu světlušek

Nejlepší řešení nalezené pomocí algoritmu světlušek je řešení číslo 9 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}552$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}0045\ \exp(-0{,}1323s)}{(0{,}9933s + 1)(0{,}2794s + \exp(-0{,}1038s))} \tag{32}$$



Obr. 28.: Porovnání odezvy na vstupní signál vodní levitace a soustavy (32)



Obr. 29.: Porovnání přechodových charakteristik vodní levitace a soustavy (32)

3.4.3. Identifikace optimalizací hejnem částic

Při identifikaci reálné soustavy pomocí optimalizace hejnem částic bylo použito následující nastavení parametrů:

$$\alpha = 1{,}2 \qquad \beta = 1{,}6$$

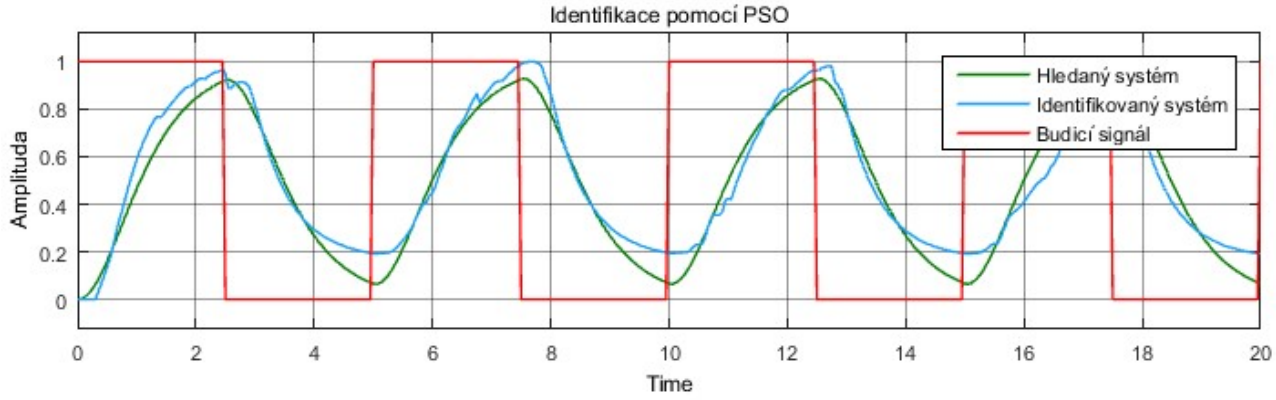
Při identifikaci reálné soustavy optimalizace hejnem částic opět projevila vyšší citlivost na nastavení řídicích parametrů a to především hodnoty β.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	20	34,4	0,601	1,72
2	24	40,8	0,760	1,70
3	19	32,5	0,791	1,71
4	18	30,9	0,742	1,72
5	12	22,0	0,665	1,83
6	24	41,1	0,796	1,71
7	22	37,5	0,599	1,70
8	18	31,4	0,780	1,74
9	22	36,8	0,793	1,67
10	24	40,0	0,677	1,67
Ø	**20**	**34,7**	**0,720**	**1,72**

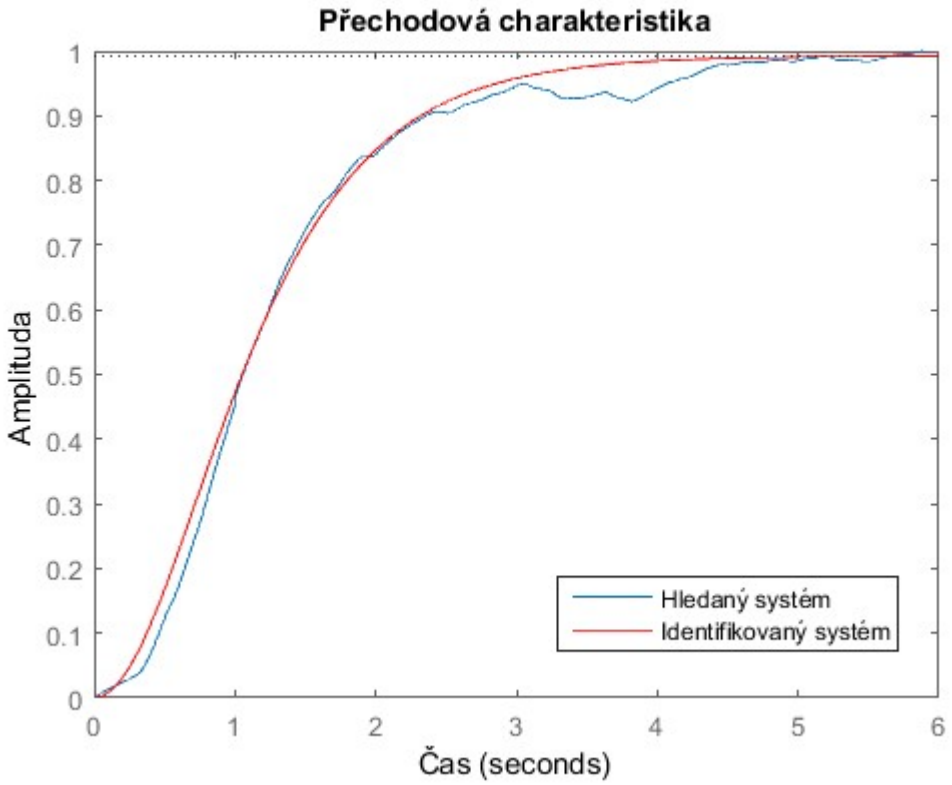
Tabulka 11.: Identifikace reálné soustavy pomocí optimalizace hejnem částic

Nejlepší řešení nalezené pomocí optimalizace hejnem částic je řešení číslo 7 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}599$, kterému odpovídá anisochronní model:

$$G(s) = \frac{0{,}9928\ \exp(-0{,}0004s)}{(0{,}6758s + 1)(0{,}8576s + \exp(-0{,}3187s))} \tag{33}$$



Obr. 30.: Porovnání odezvy na vstupní signál vodní levitace a soustavy (33)



Obr. 31.: Porovnání přechodových charakteristik vodní levitace a soustavy (33)

3.4.4. Identifikace diferenciální evolucí

Při identifikaci reálné soustavy pomocí diferenciální evoluce bylo použito následující nastavení parametrů:

$$F = 0{,}45 \qquad C_r = 0{,}85$$

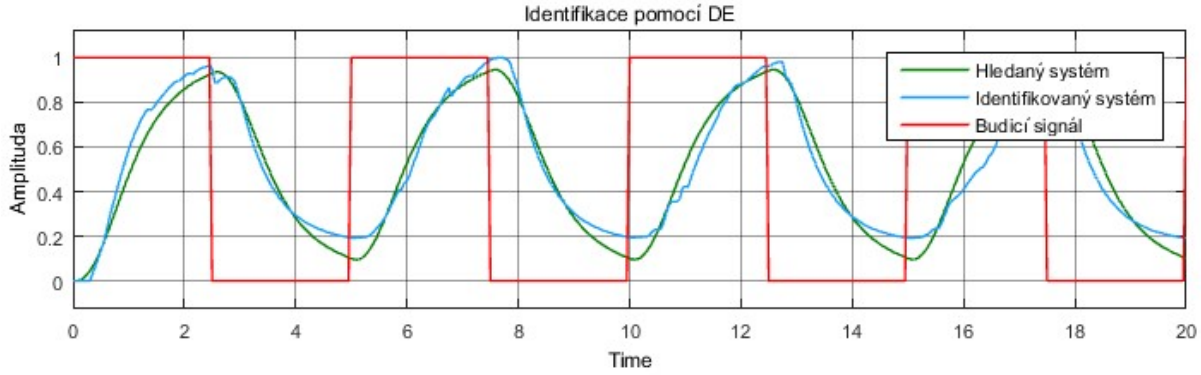
Diferenciální evoluce při identifikaci reálné soustavy nedosahovala tak dobrých výsledků, jako použité hejnové algoritmy. Úprava řídicích parametrů na tuto skutečnost měla jen malý vliv.

n	Počet iterací i	Čas t [s]	Dosažená $\boldsymbol{f}_{cost}$	t/i [s]
1	22	37,0	0,679	1,68
2	29	47,9	0,722	1,65
3	29	48,1	0,731	1,66
4	23	38,3	0,720	1,67
5	28	56,6	0,789	2,02
6	23	38,4	0,741	1,67
7	27	45,1	0,774	1,67
8	29	48,1	0,780	1,66
9	23	38,3	0,742	1,67
10	20	33,6	0,623	1,68
Ø	**25**	**43,1**	**0,731**	**1,70**

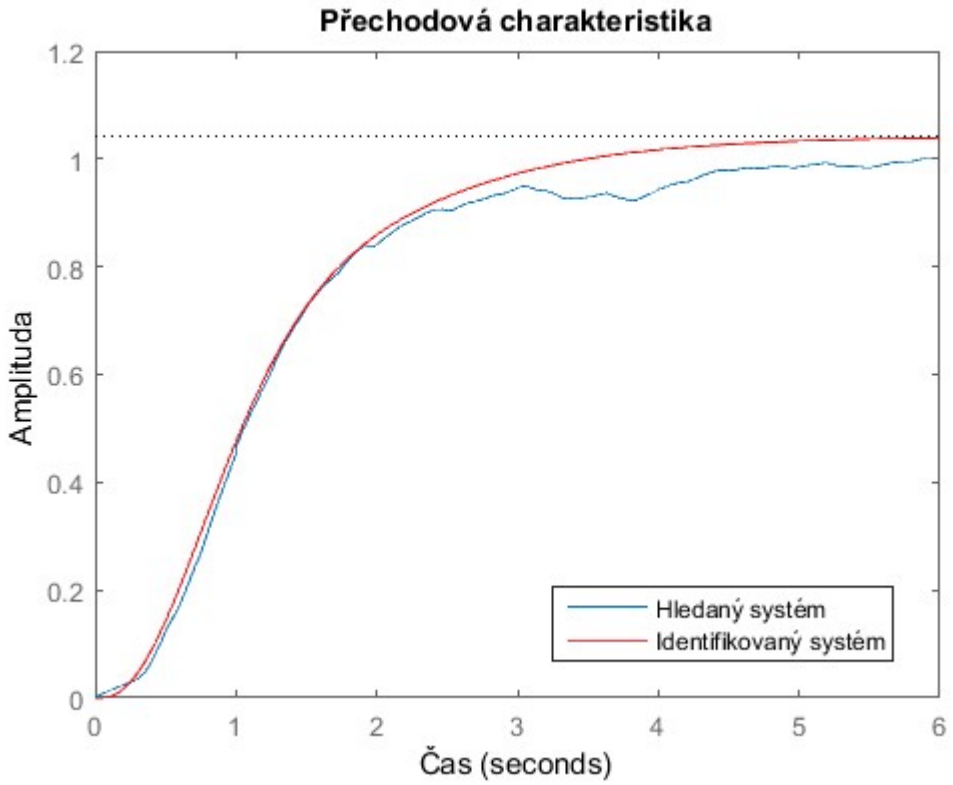
Tabulka 12.: Identifikace reálné soustavy pomocí diferenciální evoluce

Nejlepší řešení nalezené pomocí diferenciální evoluce je řešení číslo 10 s hodnotou účelové funkce $\boldsymbol{f}_{cost} = 0{,}623$, kterému odpovídá anisochronní model:

$$G(s) = \frac{1{,}0423\ \exp(-0{,}0577s)}{(0{,}8848s + 1)(0{,}6207s + \exp(-0{,}3410s))} \tag{34}$$



Obr. 32.: Porovnání odezvy na vstupní signál vodní levitace a soustavy (34)



Obr. 33.: Porovnání přechodových charakteristik vodní levitace a soustavy (34)

3.4.5. Výsledky identifikace

Ani tentokrát u žádného z použitých algoritmů nedošlo k uvíznutí a pokaždé bylo nalezeno řešení s požadovanou hodnotou účelové funkce $\boldsymbol{f}_{cost}$, jak je možné se přesvědčit v tabulkách 9. – 12.

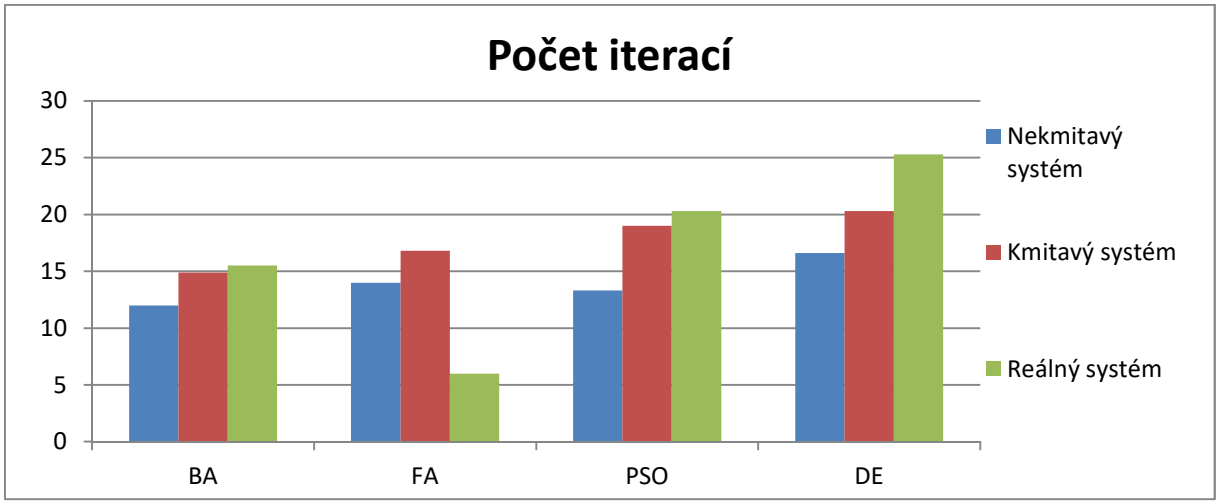
Čas potřebný k provedení jedné iterace byl u identifikace reálné soustavy výrazně zkrácen, jelikož průběh identifikované soustavy nebyl simulován, ale načítán z uložených hodnot. Všechny algoritmy provedly iteraci v čase kratším než $2\,s$. Nejpomalejší byl algoritmus světlušek.

Algoritmus světlušek k nalezení řešení potřeboval nejmenší počet iterací a to i v porovnání s identifikacemi simulovaných soustav. Netopýří algoritmus potřeboval v průměru 16 iterací, což je jenom o jednu víc než při identifikaci kmitavé soustavy. Jde tedy také o poměrně dobrý výsledek. Podobně na tom byla identifikace provedená pomocí optimalizace hejnem části, která u reálné soustavy potřebovala v průměru 20

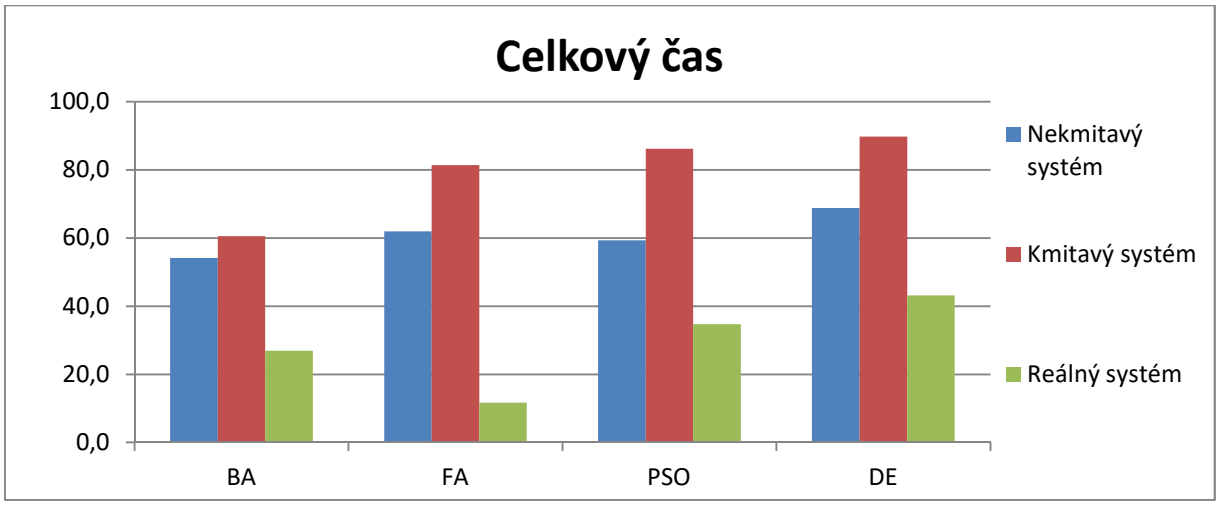
iterací, což je opět pouze o jednu více než při identifikaci kmitavé soustavy. Diferenciální evoluce potřebovala k nalezení řešení s požadovanou kvalitou 25 iterací.

Výrazné zkrácení času potřebného k provedení jedné iterace se projevilo i na celkovém potřebném čase. U algoritmu světlušek se celkový čas ve většině případů pohyboval pod $10\ s$ (ve všech případech pod $20\ s$). Netopýří algoritmus a optimalizace hejnem částic řešili úlohu přibližně $30\ s$, diferenciální evoluce průměrně $43\ s$.

3.5. Porovnání výsledků



Graf 1.: Počet iterací potřebný k dosažení výsledku s požadovanou hodnotou účelové funkce



Graf 2.: Celkový čas potřebný k dosažení výsledku s požadovanou hodnotou účelové funkce

Při uvedeném nastavení všechny algoritmy konvergovali k požadovanému řešení a nebylo třeba běh výpočtu přerušit z důvodu uvíznutí.

Většina algoritmů nejrychleji identifikovala nekmitavý systém a nejpomaleji reálný systém. Výjimkou je algoritmus světlušek, který při identifikaci reálného systému konvergoval výrazně rychleji. Netopýří algoritmus ve všech třech případech dosahoval podobné rychlosti. Diferenciální evoluce ve všech případech dosahovala horších výsledků než použité hejnové algoritmy.

Netopýří algoritmus téměř nevyžadoval úpravy řídicích parametrů. I přes to stabilně dosahoval výborných výsledků ve všech použitých případech identifikace.

Při nastavování algoritmu světlušek bylo zásadní nastavení parametru α. Jeho úpravou bylo možné zrychlit konvergenci, což se projevilo především při identifikaci reálné soustavy.

Optimalizace hejnem částic pracovala lépe, pokud pro nastavení parametrů platil vztah $\alpha < \beta$. Parametr α bylo vhodné volit v intervalu $(1{,}1 - 1{,}5)$, parametr β v intervalu $(1{,}5 - 2)$. Optimalizace nejlépe pracovala při identifikaci simulované nekmitavé soustavy, kde dosáhla výsledků srovnatelných s ostatními hejnovými algoritmy. V ostatních případech pracovala pomaleji než hejnové algoritmy.

Diferenciální evoluce konvergovala nejrychleji po nastavení menších hodnot mutační konstanty $F < 0{,}5$ a vyšších hodnot prahu křížení $C_r \approx 0{,}9$. I přes to ve všech třech testovaných případech byla časově náročnější než hejnové algoritmy, především pak při identifikaci reálné soustavy.

4. Závěr

Pro identifikaci systému byly z hejnových algoritmů vybrány netopýří algoritmus, algoritmus světlušek a optimalizace hejnem částic. Tyto algoritmy byly použity postupně pro identifikaci nekmitavého, kmitavého a reálného systému. Ve všech případech bylo dosaženo požadovaných výsledků a to použitím menšího počtu iteračních kroků než při použití diferenciální evoluce.

U každého algoritmu bylo testováno různé nastavení řídicích parametrů. Nejdříve byly použity hodnoty doporučené v odborné literatuře. Následně byly hodnoty v doporučeném rozmezí upravovány a byl sledován vliv na rychlost konvergence algoritmu. S uvedenými hodnotami algoritmus danou úlohu dlouhodobě řešil rychleji a stabilněji než pro jiné nastavení.

Netopýří algoritmus se projevil pro tento typ úloh jako univerzální nástroj. Ve všech třech případech konvergoval rychle a stabilně i při téměř stejném nastavení parametrů. Jde tedy o vhodný prostředek, pokud máme o identifikované soustavě málo informací.

Algoritmus světlušek ze všech algoritmů nejrychleji identifikoval reálnou soustavu. Nejsnáze se tedy vypořádal se šumy a nepravidelnostmi v průběhu dat. Tento algoritmus velmi dobře pracoval i s kmitavou soustavou. Dá se tedy předpokládat, že by rychle konvergoval i při identifikaci kmitavé reálné soustavy.

Optimalizace hejnem částic se projevila jako pomalejší mezi hejnovými algoritmy, přesto nejde o příliš velký rozdíl. Pro nastavení tohoto algoritmu se ukázal důležitější pohyb částic okolo lokálního optima $\boldsymbol{x}^*$, než okolo globální hodnoty $\boldsymbol{g}^*$.

U diferenciální evoluce bylo důležité nastavení vyšší hodnoty prahu křížení. Pak se rychlost konvergence blížila rychlosti použitých hejnových algoritmů. Srovnatelné kvality ovšem bylo dosaženo pouze při identifikaci simulovaných soustav.

Při identifikaci simulovaných soustav by bylo možné zkrátit výpočetní čas použitím uložených dat průběhu, podobně jako u identifikace reálné soustavy. Tím by čas potřebný k provedení jedné iterace měl odpovídat právě hodnotám u identifikace reálné soustavy.

5. ZDROJE

[1] X.-S. Yang, Nature-inspired optimization algorithms, London: Elsevier, 2014.

[2] R. Eberhart a J. Kennedy, „A new Optimizer Using Particle Swarm Theory," IEEE, Wien, 1995.

[3] D. H. Wolpert a W. G. Macready, „No Free Lunch Theorems for Optimization," IEEE Transactions on Evolutionary Computation, Piscataway, 1997.

[4] S. Nakrani a C. Tovey, „On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers," Pennsylvania, 2004.

[5] X.-S. Yang, „Multiobjective firefly algorithm for continuous optimization," *Engineering with Computers,* 2013.

[6] X.-S. Yang, Natue-Inspired Metaheuristic Algorithms, Bristol: Luniver Press, 2008.

[7] X.-S. Yang a S. Deb, „Cuckoo Search via Lévy flights," v *World Congress on Nature & Biologically Inspired Computing (NaBIC)*, Coimbatore, 2009.

[8] X.-S. Yang, „A New Metaheuristic Bat-Inspired Algorithm," *Nature inspired cooperative strategies for optimization (NICSO 2010),* 2010.

[9] X.-S. Yang, Swarm Intelligence and Bio-Inspired Computation: Theory and Applications, Boston: Elsevier, 2013.

[10] I. Zelinka, Umělá inteligence v problémech globální optimalizace, Praha: BEN - technická literatura, 2002.

[11] P. Zítek a A. Víteček, Návrh řízení podsystémů se zpožděními a nelinearitami, Praha: Vydavatelství ČVUT, 1999.

[12] I. Zelinka, Evoluční výpočetní techniky : principy a aplikace, Praha: BEN - technická literatura, 2009.

[13] „Návody na laboratorní cvičení z automatického řízení," [Online]. Available: http://vlab.fs.cvut.cz/navody/files/F2.pdf. [Přístup získán 5 6 2017].