ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

ZÁVĚREČNÁ
PRÁCE

2017

NIKITA
MAZURENKO

České vysoké učení technické v Praze

Fakulta strojní

Ústav přístrojové a řídící techniky

Obor: Informační a automatizační technika

Řízení systému využitím algoritmu diferenciální evoluce

BAKALÁŘSKÁ PRÁCE

Vypracoval: Nikita Mazurenko

Vedoucí práce: prof. Ing. Milan Hofreiter, CSc.

Rok: 2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mazurenko** Jméno: **Nikita** Osobní číslo: **424084**

Fakulta/ústav: **Fakulta strojní**

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

Studijní program: **Strojírenství**

Studijní obor: **Informační a automatizační technika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Řízení systému využitím algoritmu diferenciální evoluce

Název bakalářské práce anglicky:

System Control Using Differential Evolution Algorithm

Pokyny pro vypracování:

1) Seznámit se s algoritmem diferencální evoluce
2) Naprogramovat v prostředí Matlab algoritmus pro prediktivní řízení využívající diferencální evoluce
3) Simulačně ověřit navržené řízení

Seznam doporučené literatury:

YANG, Xin-She. Nature-inspired metaheuristic algorithms. 2nd ed. Frome,: Luniver Press, 2010, vi, 148 s. ISBN 978-1-905986-28-6.
ZELINKA, Ivan. Umělá inteligence v problémech globální optimalizace, BEN-technická literatura, Praha, 2002

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

Student bere na vědomí, že je povinen 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.

19.04.2017

Datum převzetí zadání | Podpis studenta

Prohlášení

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

V Praze dne . . . . . Podpis . . . . . .

Poděkování

Rád bych poděkoval především panu prof. Ing. Milanu Hofreiterovi CSc. za odborné rady a cenné připomínky, kterými přispěl k vypracování této bakalářské práce.

Dále bych rád poděkoval svojí rodině a přátelům, kteří mi poskytli nezbytnou podporu během celého studia.

Nikita Mazurenko

Název práce:

Řízení systému využitím algoritmu diferenciální evoluce

Autor: Nikita Mazurenko

Obor: Informační a automatizační technika

Druh práce: Bakalářka práce

Vedoucí práce: prof. Ing. Milan Hofreiter, CSc.

Ústav přístrojové a řídící techniky

České vysoké učení technické v Praze

Abstrakt: Bakalářská práce je věnována využití algoritmu diferenciální evoluce pro prediktivní řízení. Navržené a programově realizované prediktivní řízení je v práci simulačně ověřováno na lineární soustavě využitím programového prostředí MATLAB.

Teoretická část práce se věnuje problematice optimalizace a optimalizačních algoritmů. Další kapitoly popisují celou rodinu algoritmů inspirovaných živou přírodou. Větší pozornost je věnována algoritmu diferenciální evoluce, a to včetně jeho historie a principu činností. Poslední kapitola teoretické částí popisuje metodu prediktivního řízení.

Praktická část sestává ze dvou částí. První část se zabývá návrhem algoritmu prediktivního řízení s využitím diferenciální evoluce. Druhá část popisuje testovaní navrženého algoritmu na úloze řízení výšky hladiny v nádrži. Dosažené výsledky jsou doloženy časovými průběhy sledovaných veličin.

Klíčová slova: Optimalizace, prediktivní řízení, evoluční algoritmy, diferenciální evoluce.

Title:

System Control Using Differential Evolution Algorithm

Author: Nikita Mazurenko

Thesis supervisor: prof. Ing. Milan Hofreiter, CSc.

Abstract: The bachelor thesis is devoted to the use differential evolution algorithm for predictive control. The proposed and programmatically executed predictive control is simulated in a linear system using the MATLAB programming environment.

The theoretical part deals with optimization and algorithms optimization. The next chapters focus on the wide range of Nature-Inspired algorithms. More attention is paid to the differential evolution algorithm, including its history and operating principle. The last chapter of the theoretical part describes the method of predictive control.

The practical part consists of two parts. In the first part the design of the predictive control algorithm using differential evolution is characterized. The second part describes the testing of the proposed algorithm for the purpose of level control in the tank. The obtained results are documented by the time courses of the monitored parameters.

Key words: Optimization, model predictive control, evolutionary algorithm, differential evolution

Obsah

Seznam použitých zkratek

EA	Evoluční Algoritmus
CR	Crossover (práh křížení)
CV	Cost Value (účelová funkce)
DE	Diferenciální evoluce
MPC	Model Predictive Control (prediktivní řízení)
NP	Number of Population (velikost populace)

Kapitola 1

Úvod

Optimalizace – problematika, která je předmětem celé řady technických a matematických publikaci, knih a aplikací již desítky až stovky let. Relativně dlouho se problém optimalizace řešil z hlediska klasických matematických aparátů. Nedostatek této metody je ale v tom, že umožňuje nalézání globálních extrémů či řešení pouze pro jednodušší problémy. U složitějších problémů se, ale většinou dá docílit jen lokálních extrémů. Důsledkem tohoto nedostatku je, že určitá metoda optimalizace je použitelná jen pro relativně úzkou oblast problémů. V současných inženýrských problémech se rozšíření oblasti uplatnění optimalizační funkce realizuje definováním argumentů účelové funkce v různých oborech, ale to vede k tomu, že argument v určitých intervalech hodnot může nejen změnit svůj obor, ale i dostat se na různá omezení z hlediska fyzikální a ekonomické realizovatelnosti.

Na konci dvacátého století se tato situace změnila kvůli vzniku a postupnému zdokonalovaní množiny nového typu algoritmů – evolučních algoritmů.

Evoluční algoritmy vznikly jako výsledek pozorovaní a snahy napodobovat přirozeným procesům probíhajícím ve světě živých organizmů, zejména vývoj a přirozený výběr související s populaci. Myšlenka evolučních algoritmů byla navržená na konci šedesátých – počátku sedmdesátých let dvacátého století. Ta byla založená na touze vytvořit a realizovat algoritmus počítačového programu, který bude řešit složité problémy podobným způsobem, jak to dělá příroda – pomocí evolucí. Moderní bibliografie evolučních algoritmu má několik tisíc titulů a jejich počet se stále zvětšuje. Poměrně novým typem evolučního algoritmu je diferenciální evoluce, který byl poprvé použít K.Pricem a R.Stornem v roce 1995.

Úkolem této bakalářské práce je seznámení s algoritmem diferenciální evoluce, návrh algoritmu pro prediktivní řízení a studium tohoto algoritmu z hlediska optimalizace a účinností. Jako příklad pro řešení byla zvolena úloha řízení výšky hladiny v nádrži, protože je poměrně jednoduchá, rozsáhle prostudovaná a má efektivní řešení, které je možné porovnat s výsledky této práce.

Základním nástrojem pro návrh algoritmu diferenciální evoluce pro prediktivní řízení jsem zvolil prostředí MATLAB, protože má mnoho vestavených funkci a panelů nástrojů k řešení problémů programovaní algoritmů. Také v prostředí MATLAB jsem simulačně ověřil navržené řízení.

Kapitola 2

Optimalizace

Optimalizace je proces získávaní „nejlepšího“, pokud je možné měřit a měnit to, co je „dobré“ nebo „špatné“. V praxi si snaží dostat „nejvyšších“ nebo „maximálních“ anebo „nejmenších“ nebo „minimálních“ hodnot. Slovem optimum se proto rozumí „maximální“ nebo „minimální“ v závislosti na podmínkách. Optimum – technický termín, který zahrnuje kvantitativní míru a je silnější slovo než „nejlepší“, které je vhodnější pro každodenní použití [1]. Proto i slovo „optimalizovat“, které znamená dosáhnout optima, je silnější než „vylepšit“. Teorie optimalizace je obor matematiky zahrnující kvantitativní studium optima a metody pro její nalezení. Na druhou stranu optimalizační praxe je sbírka technik, metod, postupů a algoritmů, které lze použít k nalezení optima. Problémy s optimalizací se vyskytují ve většině oborů, jako je inženýrství, fyzika, matematika, ekonomika, správy, obchod, společenské vědy, a dokonce i politika. Typickými oblastmi použití optimalizace v inženýrství jsou modelovaní, charakterizace a návrh zařízení, obvodu a systémů, kontrola procesů a mnoho dalších [1].

Nejdůležitější přístup k optimalizaci je založen na numerických metodách. V tomto přístupu se iterativní numerické postupy používají k vytváření progresivně vylepšených řešení optimalizačního problému, který se začíná počátečním odhadem řešení. Tento proces je ukončen, pokud je splněno některé konvergenční kritérium. Numerické metody lze použít k řešení velmi složitých problémů s optimalizací typu, který nelze vyřešit analyticky.

Před začátkem optimalizačního procesu musí být problém správně definován. Funkční kritérium CV musí být odvozeno z hlediska n parametrů $x_1, x_2, \ldots, x_n$ jako

$$CV = f(x_1, x_2, \ldots, x_n) \tag{2.1}$$

Kritérium CV je skalární veličina, která může přijímat četné formy. Může to být například rozdíl mezi požadovaným výkonem a skutečným výkonem v systému. Proměnné $x_1, x_2, \ldots, x_n$ jsou parametry, které v tomto případě ovlivňují skutečný výkon v systému. Nejzákladnějším cílem optimalizace je nastavit parametry $x_1, x_2, \ldots, x_n$ tak, aby se minimalizovala hodnota CV [1]. Matematicky tento problém lze uvést jako

$$minimize\ CV = f(x_1, x_2, \ldots, x_n) \tag{2.2}$$

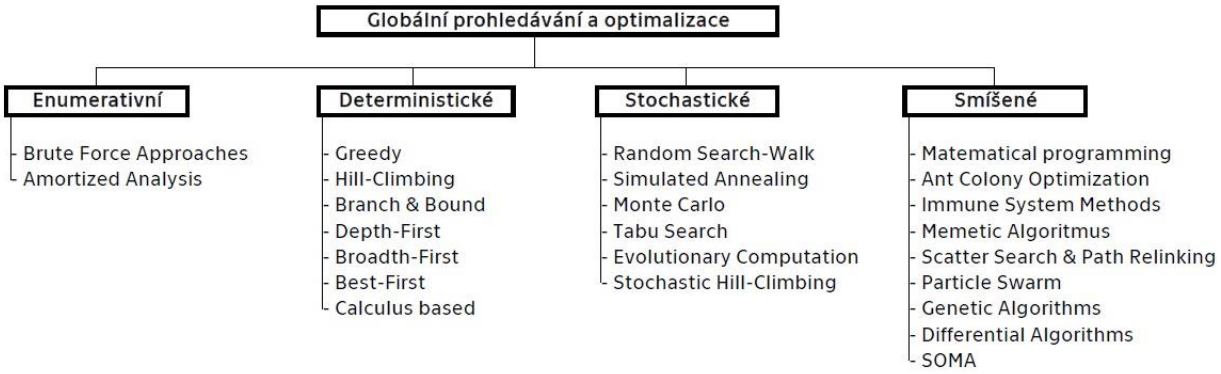
Funkční kritérium CV se obvykle označuje jako účelová (objective function) nebo cenová (cost function) funkce [1].

2.1. Problematika optimalizačních algoritmů

V předchozím textu jsem uvedl, že řešení optimalizačních problémů obvykle vyžaduje praci s argumenty účelové funkce. Definiční obory těchto argumentů mohou být různorodého charakteru jako například celočíselné, reálné, komplexní, diskrétní apod. Může se ale stát, že pro určité subintervaly z povoleného intervalu hodnot může určitý parametr optimalizované funkce nabývat různých typů hodnot (celočíselný, reálný, komplexní, diskrétní apod.). Dalším problémem může být i to, že v rámci optimalizačního procesu budou uplatněny různé hranice a omezení nejen na argumenty dane funkce, ale také i na funkční hodnotu optimalizované funkce. Při využití analytické metody optimalizace je mnohdy řešení těchto problémů možné, ale výrazně komplikované a zdlouhavé. Pro řešení těchto problémů během posledních čtyřiceti let byla vyvinuta celá množina velmi výkonových algoritmů, která se jmenuje „evoluční algoritmy“. Kvůli schopnosti těchto algoritmů elegantně řešit složité problémy, evoluční algoritmy našly široké uplatnění v mnoha inženýrských oborech [2].

Optimalizační algoritmy, které se používají pro hlídaní optimální numerické kombinace parametrů účelové funkce, lze rozdělit několika způsoby. Jeden ze způsobu je rozdělení algoritmů podle principu jejich činnosti, jak je zobrazeno v *Tab. 2.1.*

Tabulka 2.1 - Rozdělení algoritmů podle principu jejich činnosti (převzato z [2])



Každá třída optimalizačních algoritmů představuje obecný způsob řešení daného problému numerickými metodami s různým stupněm složitosti a efektivity. Dále jsou uvedeny jejich vlastnosti a oblasti optimálního použití:

- **Enumerativní.** Cílem enumerativních algoritmů je výpočet všech možných řešení daného problému. Proto jsou vhodné pro řešení optimalizačních úloh, u kterých jsou argumenty účelové funkce diskrétního charakteru a nabývají malého množství hodnot. Pro úspěšné řešení obecných problémů tímto způsobem by bylo zapotřebí nekonečného času [2].
- **Deterministické.** Princip činnosti algoritmu z této třídy je založen pouze na přísných metodách klasické matematiky. Pro dosažení efektivních výsledků algoritmy vyžadují následující předběžné předpoklady [2]:
    - Problém je lineární a konvexní
    - Malý a spojitý prohledávaný prostor možných řešení
    - Účelová funkce má, pokud možno, jeden extrém
    - Mezi parametry „uvnitř" účelové funkce nejsou nelineární interakce.
    - Jsou dostupné parametry typu gradient apod.
    - Problém je definován v analytickém tvaru.

  Výsledkem deterministického algoritmu může být jenom jedno jediné řešení.

- **Stochastické.** Jde o skupinu algoritmů založených na využití náhody. Principem je náhodné hledání hodnot parametrů účelové funkce. Výsledkem je vždy nejlepší řešení, jenž bylo nalezeno během celého náhodného hledání. Obvykle stochastické algoritmy jsou [2]:
    - Pomalé
    - Vhodné jen pro malé prohledávané prostory možných řešení
    - Vhodné pro hrubý odhad.
- **Smíšené.** Tato skupina algoritmů představuje směs stochastických a deterministických metod. Poměrně silnou podmnožinou těchto algoritmů jsou evoluční algoritmy. Algoritmy ze skupiny smíšené mají následující vlastnosti [2]:
    - Jsou robustní. To znamená, že nezávisle na počátečních podmínkách velmi často naleznou kvalitní řešení.
    - Jsou schopné najit kvalitní řešení během relativně malého počtů ohodnocení účelové funkce.
    - Požadavky na předběžnou informace jsou minimální nebo žádné.
    - Není vyžadován analytický popis problému.
    - Schopnost nalézt několik řešení během jednoho spouštění.

Z výše uvedeného textu je vidět, že smíšená metoda optimalizace může být použitá pro problémy bez omezení velikosti prostoru jejich možných řešení. Proto v další částí této bakalářské práci budu podrobněji popisovat smíšené algoritmy. Ze skupiny smíšených algoritmů v poslední době jsou velmi populární evoluční algoritmy. Proto další kapitola této bakalářské práce bude věnovaná evolučním algoritmům.

Kapitola 3

Evoluční algoritmy

Následující kapitola čerpá především ze zdroje „Introduction to Evolutionary Computing“ [3], který se přímo zabývá evolučními algoritmy.

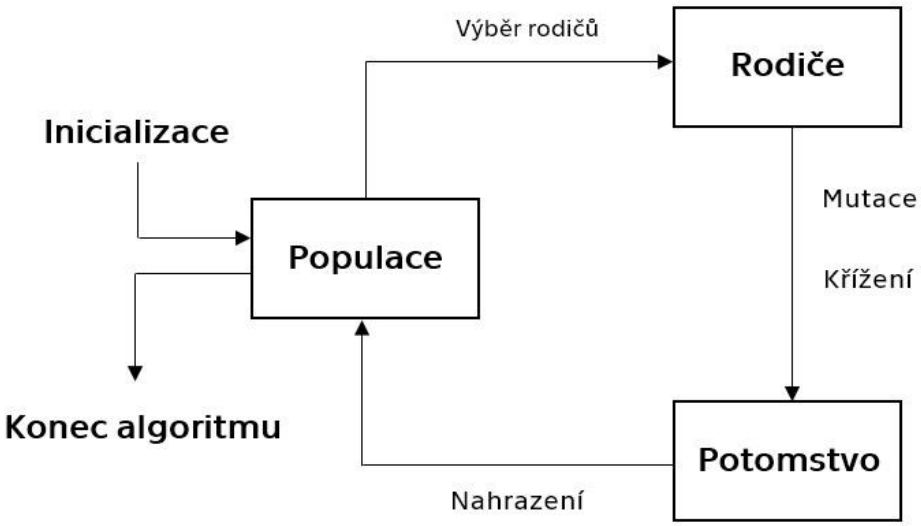
Jak dějiny oboru naznačují, existuje mnoho různých variant Evolučních algoritmů. Společná základní myšlenka, která stojí za všemi těmito technikami, je stejná: vzhledem k populaci jednotlivců působí tlak na životní prostředí přirozeným výběrem (přežití nejschopnějších), což způsobuje zvýšení zdraví obyvatelstva. Vzhledem k tomu, že funkce kvality je maximalizována, můžeme náhodně vytvořit soubor kandidátských řešení, to znamená vzít prvky z oblasti funkce a použít funkci kvality jako abstraktní opatření vhodností – čím vyšší, tím lépe. Na základě této vhodnosti se někteří z lepších kandidátů rozhodli nasadit novou generaci použitím rekombinace a / nebo mutace. Rekombinace je operátor aplikovaný na dva nebo více vybraných kandidátů (tzv. rodiče) a výsledkem je jeden nebo více nových kandidátů (děti). Mutace se vztahuje na jednoho kandidáta a výsledkem je nový kandidát. Provádění rekombinaci a mutace vede k souboru nových kandidátů (potomků), kteří soutěží se starými o místo v nové generaci. Tento proces lze opakovat, dokud nebude nalezen kandidát s dostatečnou kvalitou (řešením) nebo dokud se dosáhne předem stanoveného výpočetního limitu. V tomto procesu existují dvě základní síly, které tvoří základ evolučních systémů:

- Operátory variability (rekombinace a mutace) vytvářejí nezbytnou rozmanitost a tím usnadňují novotu.
- Výběr funguje jako síla tlačící na kvalitu.

Kombinovaná aplikace variace a selekce obecně vede ke zvýšení hodnot vhodnosti v následných populacích. Je snadné vidět takový proces, jako kdyby evoluce optimalizovala nebo alespoň "přiblížila" optimální hodnoty. Alternativně je evoluce

často považována za proces adaptace. Z tohoto hlediska ji není vhodné považovat za objektivní funkci, kterou je třeba optimalizovat, ale jako vyjádření požadavků na ochranu životního prostředí. Přiměřenější splnění těchto požadavků znamená vyšší životaschopnost, která se odráží ve vyšším počtu potomků. Evoluční proces činí obyvatelstvo lépe a lépe přizpůsobené k prostředí.

Uvědomme si, že mnohé složky takového evolučního procesu jsou stochastické. V průběhu výběru mají jednotlivci vyšší šanci být zvoleni než méně vhodní, ale typicky i slabí jednotlivci mají šanci se stát rodičem nebo přežít. Pro rekombinaci jedinců je výběr těch částí, které budou rekombinovány, náhodný. Podobně i pro proces mutace jsou kousky, které budou mutované v kandidátském řešení, náhodně vybrané. Tyto zmutované kousky pak nahrazují ty původně vybrané. Obecná schéma evolučního algoritmu je uvedena na obrázku 3.1.



Obrázek 3.1 - Schéma evolučního algoritmu (převzato z [3])

Je snadné vidět, že toto schéma spadá do kategorie algoritmů generování a testování. Funkce hodnocení představuje heuristický odhad kvality řešení a vyhledávací proces je řízen variací a výběrovými operátory. Evoluční algoritmy (EA) mají řadu vlastností:

- Jsou založeny na populaci, tj. zpracovávají současně celou kolekci kandidátských řešení
- Většinou používají rekombinaci pro mixování informací o více kandidátských řešeních do nového
- Jsou stochastické

3.1. Komponenty evolučních algoritmu

V této části budu podrobně diskutovat o evolučních algoritmech. EA mají řadu komponent, postupů nebo operátorů, které musí být specifikovány za účelem definování konkrétní EA. Nejdůležitější komponenty jsou:

- Reprezentace (definice jednotlivců)
- Vyhodnocovací funkce (nebo fitness funkce)
- Populace
- Mechanismus výběru rodičů
- Variační operátory, mutace a křížení
- Mechanismus výběru přeživších jedinců (nahrazení)

Každá z těchto složek musí být specifikována za účelem definování konkrétního EA. Pro získání běhu algoritmu musí být také definován inicializační postup a podmínka ukončení.

3.1.1. Reprezentace (definice jednotlivců)

Prvním krokem při definování EA je propojení "skutečného světa" se "světem EA", tj. vytvoření mostu mezi původním kontextem problému a problémem, který řeší prostor, ve kterém bude probíhat evoluce. Objekty tvořící možné řešení v původním problémovém kontextu jsou označovány jako fenotypy, přičemž jejich jednotlivé kódování se v rámci EA nazývá genotyp. První konstrukční krok je běžně nazýván reprezentací, neboť spočívá ve specifikaci mapování z fenotypů na soubor genotypů, o kterých se uvádí, že tyto fenotypy reprezentují. Například vzhledem k optimalizačnímu problému na celých číslech by daná sada celých čísel tvořila množinu fenotypů. Pak bychom se mohli rozhodnout, že je budeme zastupovat binárním kódem, a proto by bylo 18 považováno za fenotyp a 10010 za genotyp, který by ho reprezentoval. Je důležité si uvědomit, že fenotypový prostor může být velmi odlišný od genotypového prostoru a že celé evoluční hledání se odehrává v genotypovém prostoru. Řešení – dobrý fenotyp – se získá dekódováním nejlepšího genotypu po ukončení procesu evoluce. Za tímto účelem by mělo usoudit, že optimální řešení problému – fenotyp – je zastoupeno v daném genotypovém prostoru.

Pro označení prvků těchto dvou prostorů se používá mnoho synonym. Na straně původního problémového kontextu se pro určení bodů prostoru možných řešeních používá kandidátské řešení, fenotyp a jednotlivec. Tento prostor se sám běžně nazývá fenotypovým prostorem. Na straně EA, genotypy, chromozomy a opět jednotlivce lze

použít jako body v prostoru, kde se skutečně objeví evoluční vyhledávání. Tento prostor se často nazývá genotypovým prostorem. Také pro prvky jednotlivců existuje mnoho synonymních pojmů. Držitel místa je běžně nazýván proměnnou polohou nebo – v biologicky orientované terminologii – genem. Objekt na takovém místě může být nazýván parametrem nebo hodnotou.

Je třeba poznamenat, že slovo "reprezentace" se používá dvěma různými způsoby. Někdy se jedná o mapování z fenotypového do genotypového prostoru. V tomto smyslu je synonymem pro kódování, např. binární reprezentace nebo binární kódování kandidátských řešení. Inverzní mapování z genotypů na fenotypy se obvykle nazývá dekódováním a je nutné, aby reprezentace byla invertibilní: ke každému genotypu musí být nejvýše jeden odpovídající fenotyp. Slovní reprezentace může být také použita v poněkud jiném smyslu, kde důraz není kladen na samotné mapování, ale na "datovou strukturu" genotypového prostoru [4].

3.1.2. Vyhodnocovací funkce

Úloha vyhodnocovací funkce představuje požadavky na přizpůsobení. Tato funkce formuluje základ pro selekce a tím usnadňuje zlepšení. Také přesněji definuje to, co znamená zlepšení. Z hlediska řešení problémů představuje úkol pro řešení v evolučním kontextu. Technicky je to funkce nebo postup, který přiřazuje měřítko kvality genotypům. Obvykle se tato funkce skládá z měření kvality ve fenotypovém prostoru a inverzní reprezentaci.

Docela často původní problém, který má být vyřešen pomocí EA, je optimalizačním problémem. V tomto případě se název účelová funkce často používá v původní formulaci problémů a vyhodnocovací (fitness) funkce může být identická nebo jednoduchá transformace dané účelové funkce.

3.1.3. Populace

Úlohou populace je držet (reprezentovat) možná řešení. Populace je množina genotypů, která tvoří jednotku evoluce. Jednotlivci jsou statické objekty, které se nemění ani neupravují, je to obyvatelstvo. Vzhledem k zastoupení může být definování populace tak jednoduché, jako určení toho, kolik jednotlivců je v něm, tj. nastavení velikosti populace. V některých sofistikovaných EA populace má další prostorovou strukturu, s distanční mírou nebo sousedským vztahem. V takových případech musí být také definována dodatečná struktura, aby bylo možné zcela určit populaci. Na rozdíl od variačních operátorů, kteří působí na jeden nebo dva rodiče, operátoři výběru (výběr

rodičů a výběr pozůstalých) pracují na úrovni populace. Obecně platí, že zachycují celou současnou populaci a volí se vždy vzhledem k tomu, co máme. Například nejlepší jedinci z dané populace jsou vybrány pro nasazení nové generace nebo nejhorší jednotlivec dané populace je vybrán, aby byl nahrazen novým. U téměř všech aplikací EA je velikost populace konstantní a během evolučního vyhledávání se nemění.

3.1.4. Mechanismus výběru rodičů

Role výběru rodičů je rozlišovat mezi jednotlivci podle jejich kvality, a to zejména umožnit lepší osoby, aby se stali rodiči další generace. Jednotlivec je rodičem, pokud byl vybrán, aby prošel změnami, aby vytvořil potomstvo. Spolu s mechanismem výběru přeživších je výběr rodičů zodpovědný za posílení zlepšení kvality. Rodičovský výběr je typicky pravděpodobnostní. Proto vysoce kvalitní jednotlivci mají vyšší šanci se stát rodiči než ti s nízkou kvalitou. Nicméně jednotlivci s nízkou kvalitou mají často malou, ale pozitivní šanci jinak by celé hledání mohlo být příliš chamtivé a uvízlo v místním optimu.

3.1.5. Variační operátory

Úlohou variačních operátorů je vytvářet nové jedince ze starších. V odpovídajícím fenotypovém prostoru to znamená vytvoření nových kandidátských řešení. U evolučních algoritmu jsou variační operátory rozdělené do dvou typů podle počtu operandů. [3]

3.1.5.1. Mutace

Unární[1] operátor variace se běžně nazývá mutace. Aplikuje se na jeden genotyp a přináší modifikovaného jedince, dítě nebo potomstvo. Operátor mutace je vždy stochastický: jeho výstup – dítě – závisí na výsledcích série náhodných výběrů. Je třeba poznamenat, že libovolný unární operátor nemusí být nezbytně považován za mutaci. Specifický heuristický operátor, který se používá na jednom jedinci, může být nazván jako mutace za to, že je unární. Nicméně obecně má mutace způsobit náhodnou, nestrannou změnu. Z tohoto důvodu by mohlo být vhodnější nevyužívat název mutace pro každý heuristický unární operátor.

3.1.5.2. Křížení

Binární[2] operátor variace se nazývá rekombinace nebo křížení. Jméno naznačuje, že takový operátor sloučí informace z několika původních genotypů do jednoho nebo dvou genotypů, potomků. Podobně jako mutace je rekombinace stochastickým

[1] Operátor je unární, pokud se vztahuje na jeden objekt jako vstup [3]
[2] Operátor je binární, pokud se vztahuje na dva objekty jako vstup [3]

operátorem: volba toho, jaké části každého rodiče jsou kombinovány a způsob, jakým jsou tyto části kombinovány, závisí na náhodě. Zásada rekombinací je jednoduchá – pářením dvou jedinců s různými, ale žádoucími rysy můžeme vytvořit potomky, které kombinují oba tyto rysy [3].

3.1.6. Mechanismus výběru přeživších jedinců (nahrazení)

Úloha výběru přeživších jedinců je podobná rodičovskému výběru, to znamená rozlišovat jednotlivce na základě jejich kvality, ale používá se v jiné fázi evolučního cyklu. V tom je podobný rodičovskému výběru, ale používá se v jiné fázi evolučního cyklu. Mechanismus výběru přeživších se použije, až po vytvoření potomků vybraných rodičů. Velikost populace v evolučních algoritmech je konstantní, a proto musí být vybráno, které osoby budou povoleny v nové generaci. Toto rozhodnutí je obvykle založeno na jejich hodnotách účelové funkce, které upřednostňují osoby s vyšší kvalitou, ačkoli je často používán i pojem věk. Z toho důvodu, na rozdíl od výběru rodičů, který je typicky stochastický, výběr jedinců pro novou generace je často deterministický.

3.1.7. Inicializace

Inicializace je jednoduchá ve většině aplikací EA: První populace je nasazena náhodně vytvořenými jednotlivci. V tomto kroku lze v principu použít heuristiku specifickou pro problém, která se zaměřuje na počáteční populaci s vyšším zdraví. To, zda za to stojí extra výpočetní úsilí nebo ne, je velmi závislé na konkrétní aplikaci. [4]

3.1.8. Konec algoritmu

Pokud jde o vhodnou podmínku ukončení, můžeme rozlišit dva případy. Pokud má problém známou optimální úroveň kondice, pravděpodobně pocházející ze známého optimu dané účelové funkce, mělo by být jako podmínka zastavení použito dosažení této úrovně. Avšak EA jsou stochastické a většinou neexistují žádné záruky k dosažení optimálního stavu, proto by tato podmínka nikdy nenastala a algoritmus se nikdy neskončí. To vyžaduje, aby tato podmínka byla rozšířena o podmínku, která určitě zastaví algoritmus. Příkladem může být podmínka ukončení algoritmu po vypršení maximálního povoleného času, nebo po provedení určitého počtů simulačních cyklů.

Tim je v podstatě popsán cely princip činností evolučních algoritmů. V rámci této bakalářské práce bude použít jeden z nejnovějších evolučních algoritmů – algoritmus diferenciální evoluce. Ale než se tento algoritmus začne probírat, je vhodně se zmínit o oblasti, ve které tento algoritmus bude použit. Proto cílem další kapitoly je seznámit čtenáře s metodou prediktivního řízení.

Kapitola 4

Prediktivní řízení

Tato kapitola je věnována popisu prediktivního řízení. Hlavním zdrojem této kapitoly je [5].

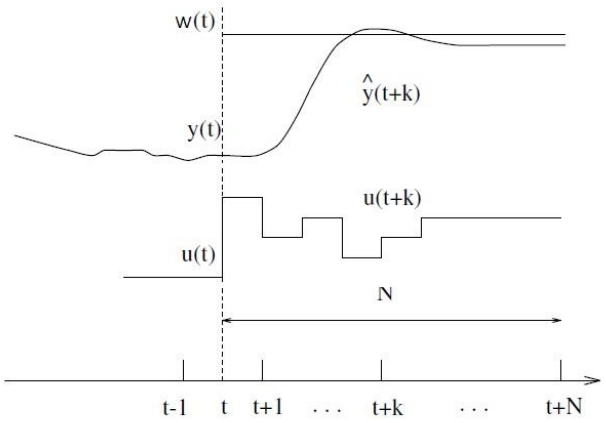
Prediktivní řízení (Model Predictive Control) je pokročilá metoda řízení procesů, která se od 80. let 20. století používá v průmyslových procesech, chemických zařízeních a rafineriích ropy. Termín MPC neurčuje specifickou strategii řízení, ale spíše široký rozsah řídicích metod, které výslovně používají model procesu pro získání řídicího signálu pomocí minimalizace účelové funkce. Tyto metody se vyznačují následujícími společnými znaky:

- Přímé užití modelu k předpovědi výstupu procesu v budoucích časových okamžicích
- Výpočet řídící sekvence minimalizující účelovou funkci
- Posunutí horizontu v každém okamžiku do budoucnosti; opakovaní stejného výpočtu v každém kroku nad aktualizovanými daty

MPC se považuje za moderní způsob řízení, ale tento způsob není dokonalý, proto má několik nevýhod proti klasickému. Mezi nejdůležitější patří nutnost definovaní dobrého matematického modelu, která má velký vliv na kvalitu regulace.

4.1. Princip prediktivního řízení

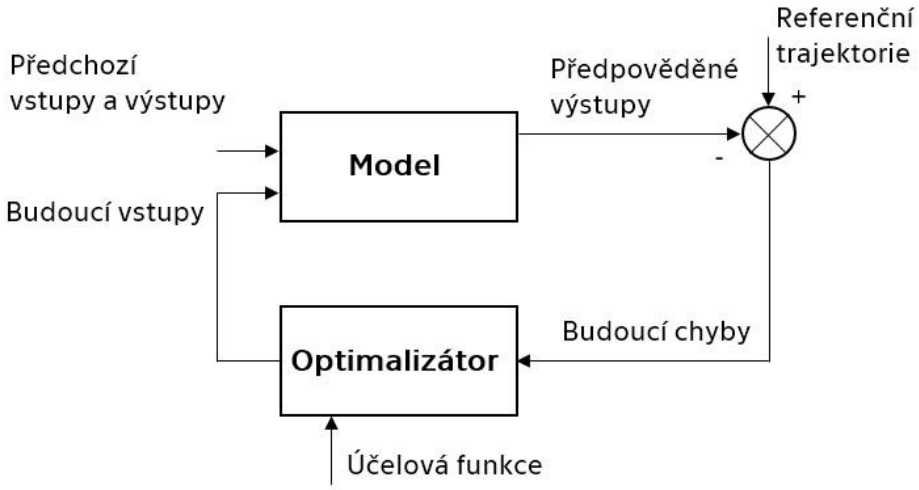
Základní princip prediktivního řízení je charakterizován následující strategií, která je znázorněna na obrázku 4.1:



Obrázek 4.1 - Strategie prediktivního řízení (převzato z [2])

1. Budoucí výstupy pro určený horizont N (predikční horizont) jsou předpovězeny v každém okamžiku t pomocí modelu procesu. Tyto předpokládané výstupy $y\,(t+k)$ pro $k = 1 \ldots N$ závisí na známých hodnotách až do okamžiku t (minulé vstupy a výstupy) a na budoucích řídících signálech $u(t+k)$ pro $k = 0 \ldots N-1$, které mají být odeslány do systému a vypočteny.
2. Sada budoucích řídících signálů se vypočítá pomocí optimalizací určitého kritéria pro udržení procesu co nejblíže požadované referenční trajektorii $w(t+k)$, může to být samotná požadovaná hodnota nebo její blízká aproximace. Toto kritérium je obvykle definováno v podobě funkce kvadratické odchylky mezi předpokládaným průběhem řízené veličiny a jejím požadovaným průběhem.
3. Akční signál $u(t)$ se vysílá do regulovaného procesu, zatímco budoucí určené akční zásahy $u(\tau), pro\ \tau > t$ nejsou realizovány, protože při dalším odběru vzorků je již známá skutečná hodnota řízené veličiny $y(t+1)$ a první krok se opakuje s touto novou hodnotou. Tímto způsobem se vypočítá hodnota $u(t+1)$ s využitím konceptu ustupujícího horizontu.

Pro implementaci této strategie je třeba použít základní strukturu regulátoru, která je zobrazena na obrázku 4.2:



Obrázek 4.2 - Základní struktura regulátoru (převzato z [5])

Model se používá k předpovědi budoucích výstupů zařízení na základě minulých a současných hodnot a na navrhovaných optimálních budoucích akčních veličinách. Model procesu hraje rozhodující roli, protože zvolený model musí být schopen zachytit dynamiku procesu, aby přesně předpovídal budoucí výstupy a byl jednoduše implementovatelný a srozumitelný.

Optimalizátor je další zásadní součástí struktury prediktivního řízení, protože poskytuje akční veličiny (vstupy) do modelu. Tyto vstupy jsou vypočítány s ohledem na účelovou funkci. Je-li tato funkce kvadratická, lze její minimum získat jako lineární funkci minulých vstupů a výstupů a budoucí požadované trajektorii. [2]

Dá se říct, že strategie MPC je velmi podobná strategii řízení, kterou používají řidiči automobilů. Řidič zná požadovanou trajektorii pro konečný horizont řízení a při zohlednění charakteristik vozu rozhodne o tom, které řídící akce zvolit (přidaní plynu, brzdy, otáčení volantem) a do jaké míry je použít, aby se řídil požadovanou trajektorií. V každém okamžiku se provádějí pouze první kontrolní zásahy, podle kterých řidič pozná odezvu auta (změnu trajektorie) a bude mít představu o tom, kterou akční veličinu (např. rychlost nebo poloha volantu) a jak ji musí změnit pro další akční zásah.

Kapitola 5

Diferenciální evoluce

V předchozích kapitolách jsem popisoval různé typy optimalizačních algoritmů a podrobněji jsem psal o perspektivnější rodině algoritmů v dnešní době – evolučních algoritmů. Ale v následujících kapitolách bude použit jenom jeden algoritmus z této oblasti – diferenciální evoluce. Kapitola č. 5 čerpá především ze dvou zdrojů ([2] a [6]), které se zabývají algoritmem DE.

Diferenciální evoluce je metoda, která optimalizuje problém a snaží se zlepšit kandidátské (možné) řešení s ohledem na danou míru kvality. DE patří k metaheurastickým metodám. To jsou metody, které dělají málo nebo žádné předpoklady o optimalizačním problému, což dovoluje metodám vyhledávat velmi velké prostory kandidátských řešení [7]. Nicméně DE jako i celá metaheuristika nezaručuje optimální řešení.

Většinou se DE používá vícerozměrné reálné funkce, ale bez použití gradientu problému, který je třeba optimalizovat. To znamená, že DE nevyžaduje, aby problém byl diferencovatelný, jak je vyžadováno u klasických metod optimalizace. Proto metoda DE může být také použita při optimalizačních problémech, které nejsou spojité, mění se v čase a mají šumy.

Při použití DE je optimalizace daná tím, že se udržuje populace kandidátských řešení a nová řešení vznikají kombinací stávajících řešení podle jednoduchých vzorců a pak se udržuje jen to kandidátské řešení, které je nejlepší nebo nejvíc vhodné pro optimalizační problém. Da se říct, že se tímto způsobem problém optimalizuje jako „černá skříňka“, která pouze poskytuje určitou kvalitu vzhledem ke kandidátskému řešení, proto gradient problému není nutný.

5.1. Historie

V roce 1994 v populárním pro programátory magazínu Dr. Dobb's byl publikován článek o genetickém žíhání [8]. Následkem tohoto příspěvku byla spolupráce autora genetického žíhání Kennetha V. Price s Dr. Rainierem Stornem o využití genetického žíhání pro řešení složitějších problémů. Během tyto práce K. Price začal měnit genetické žíhání z reprezentace binární do dekadické a úměrně tomu operace logické na vektorové. Díky těmto změnám z genetického žíhání vznikl typický algoritmus vhodný pro numerickou optimalizace

Posléze po provedení těchto změn K. Price velmi rychle objevil tzv. diferenciální mutace. Princip diferenciální mutace je v tom, že se tvoří zkušební řešení přičtením rozdílu dvou náhodně zvolených vektorů (jedinců) ke třetímu vektoru z populace. Další změnu navrhl R. Storn a to takovou, že populace potomků vytváří zvláštní populaci, ve které se pak provádí soupeření o místo v nové populaci po naplnění zvláštní populace. Tento nový algoritmus byl pojmenován "diferenciální evoluce".

Protože tato verze evolučního algoritmu byla velmi úspěšná, oba autoři se rozhodli o její publikování [9]. Během práce s touto verzí algoritmu se ukázalo, že DE velmi dobře fungovala na testovacích funkcích, ale pro řešení široké množiny optimalizačních problémů byla nedostačující. Proto vědci vyvinuli další, třetí verzi DE [10], na jejichž základě vznikl novy článek pro magazín Dr. Dobb's. Tímto příspěvkem se DE poprvé představila širšímu publiku na úrovni odborné literatury.

Z důvodu růstu zájmu o nový typ evolučního algoritmu, byla v dalším kroku vytvořena internetová stránka. Tato stránka nejen prezentovala algoritmus, ale i zdrojové texty algoritmu. Také byl publikován další článek o DE. Podle výsledků práce s DE, v tomto článku algoritmus byl klasifikován jako jednoznačně nejlepší algoritmus. [2]

5.2. Parametry diferenciální evoluce

Stejně jako u ostatních evolučních algoritmů, řídicí parametry výrazně ovlivňují kvalitu a činnost diferenciální evoluce. Seznam parametrů a jejich význam je uveden níže:

- $CR \in <0, 1>$ – Práh křížení. Pro případ použití algoritmu pro separabilní funkce je lepší použít hodnoty parametru blízké nule. V opačném případě se

doporučuje nastavit hodnoty blízké jedné. Když hodnota CR bude nastavena na nulu dojde k tomu, že zkušební jedinec bude čistou kopii aktuálního. V tomto případě se mutace zastaví. V případě, když parametr CR bude nastaven na hodnotu jedna, bude zkušební vektor tvořen pouze ze tří náhodně vybraných jedinců, což praktické znamená, že bude probíhat náhodné hledání čísel nežli evoluční algoritmus. Proto je vždycky vhodné volit hodnoty CR odlišné od 0 a 1.

- **NP** $\in <2\mathrm{D}, 100\mathrm{D}>$. To je parametr, který udává velikost populace. Hodnotu tohoto parametru se dá nastavit jenom podle vlastní zkušenosti s tímto algoritmem. Jediné omezení je, že hodnota parametru NP nemá být menší než 4. Při menších velikostech populace, DE přestane fungovat. Nejpoužívanější metoda je, když NP $= 10 \cdot$D.
- **F** $\in <0, 2>$ – Mutační konstanta. Jedná se o poslední řídicí parametr algoritmu, který může nabývat hodnot od 0 do 2.
- **G** (Generace) > 0 – Počet generací, to znamená evolučních cyklů, během nichž se celá populace vyvíjí.
- **D** – Dimenze problému. Parametr, který udává počet argumentů účelové funkce.

Pro názornost, doporučené hodnoty těchto řídicích parametrů jsou uvedené v Tab. 5.1. Zde ale musím uvést, že volbu parametrů ovlivňuje další jev, který se jmenuje stagnace. Tento jev bude popsán v následujícím textu.

Tabulka 5.1- Doporučené hodnoty řídicích parametrů (převzato z [2])

Parametr	Interval	Význam
CR	$< 0, 1 >$	Práh křížení
NP	$< 2\mathrm{D}, 100\mathrm{D} >$	Velikost populace
F	$< 0, 2 >$	Mutační konstanta
Generations	> 0	Počet generací
D	Podle účelové funkce	Počet argumentu účelové funkce

5.2.1. Populace

Základem evolučních algoritmů je práce s populací jedinců. To platí i pro diferenciální evoluce. Pro názornost, populace se da představit jako matice o velikosti NP×D, kde sloupce jsou vektory, představující jednotlivé jedince. Každý jedinec představuje aktuální řešení daného problému. Řešením je množina argumentů účelové funkce (CV – Cost Value) jejíchž optimální číselná kombinace je hledaná. Účelová funkce

udává vhodnost příslušného jedince pro další populace a je spojitá s každým jedincem z populace. Hodnota účelové funkce nese pouze informace o kvalitě určitého jedince, a proto se neúčastní evolučního procesu [11].

Před začátkem evolučního procesu je třeba definovat vzorového jedince (prototyp) pro vygenerovaní počáteční populace. Vzorový jedinec se také používá pro korekci argumentů jedince při překročení hranic prohledávaného prostoru. Pro každou proměnnou konkrétního jedince ve vzorovém jedinci jsou definovány tři parametry. První parametr definuje typ proměnné, například celočíselná, reálná nebo diskrétní. Druhé a třetí parametry definují hranice intervalu, ve kterém se hodnota příslušné proměnné může pohybovat. Příkladem může být {Integer, {Lo, Hi}}, ve kterém je zadán celočíselný typ parametru s dolní hranicí Lo a horní hranicí Hi. Velmi důležité je správné definovaní intervalu pro konkrétní proměnnou, protože v opačném případě může nastat situace, kdy nalezená řešení budou fyzicky nerealizovatelná.

5.2.2. Mutace

Mutace hraje životně důležitou roli jak v klasické evoluční teorii, tak i v evolučních algoritmech. Zvláštnost DE je v tom, že pro tvorbu dalšího potomka je potřeba čtyř rodičů, ale ne dvou jako tomu u ostatních algoritmů. Pro každého jedince z populace se musejí vybrat další tři různí jedinci. Z těchto tří jedinců se vytvoří nový vektor, který se nazývá „šumový vektor". Tento šumový vektor je mutací kombinace tří náhodně zvolených jedinců (rodičů). Mutace se provádí podle vztahu (5.1), ve kterém se rozdíl prvních dvou náhodně vybraných vektoru vynásobí mutační konstantou „F" a přičte se ke třetímu vektoru. [6]

$$v_j = x^G_{r3,j} + F \cdot \left(x^G_{r1,j} - x^G_{r2,j}\right) \tag{5.1}$$

kde

- v_j . . . Šumový vektor
- $x^G_{r1,j}$ . . . První náhodně zvolený jedinec
- $x^G_{r2,j}$ . . . Druhý náhodné zvolený jedinec
- $x^G_{r3,j}$ . . . Třetí náhodně zvolený jedinec
- F . . . Mutační konstanta
- G . . . Pořadové číslo generace
- j . . . Pořadové číslo parametru

5.2.3. Křížení

Pojem křížení u klasických evolučních algoritmů znamená tvorbu nového potomka čili vygenerovaní další pozice na dané hyperploše. DE má další zvláštnost, a to takovou, že proces křížení nastává až po mutaci jedinců. U DE proces křížení spočívá v tom, že se nový jedinec, který se nazývá zkušební, vytváří ze šumového a čtvrtého, doposud nepoužitého vektoru. Pro tvorbu zkušebního vektoru se používá další parametr algoritmu - práh křížení (CR), zmíněný v předchozím textu. V cyklu se postupně vybírají korespondující parametry z aktivního (čtvrtého) a šumového vektoru, například oba první parametry, oba druhé parametry atd. Pro každou vybranou dvojici parametrů je vygenerováno náhodné číslo z intervalu od nuly do jedné. V případě, že náhodně zvolené číslo je menší než hodnota parametru CR, pak do příslušného parametru ve zkušebním vektoru se přesune hodnota parametru ze šumového jedince. V opačném případě do příslušného parametru se přesune hodnota z aktivního jedince. Když se obrátí podmínka pro přiřazování parametru do zkušebního vektoru, výkon algoritmu se nezmění. Po dokončení procesu naplnění hodnot zkušebního vektoru se vypočítá hodnota účelové funkce zkušebního jedince. Tato hodnota se porovnává s hodnotou účelové funkce. Vektor, který má tuto hodnotu optimálnější, se přesune do nové populace. [11]

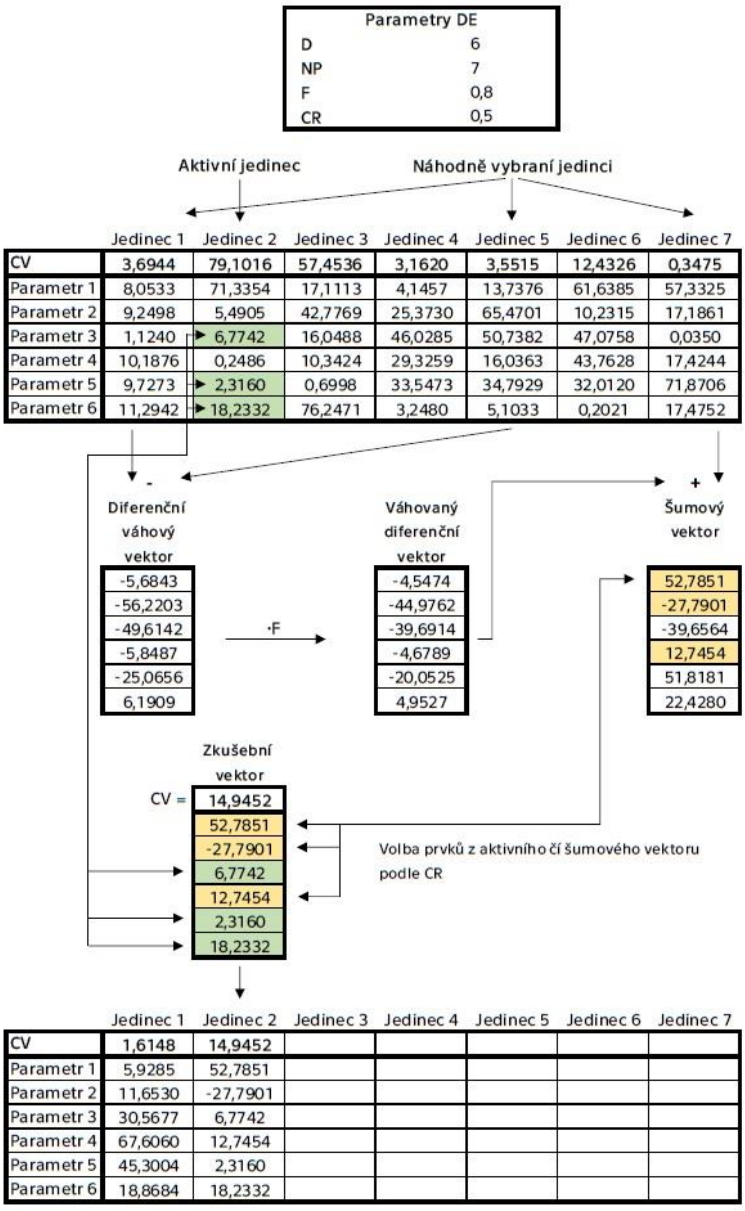
5.2.4. Princip činnosti algoritmu

Cílem algoritmu diferenciální evoluce je během cyklů zvaných „generace“ najít co nejlepší populaci jedinců pro hodnoty účelové funkce, která je spojena s každým jedincem. Každá generace probíhá podle následujících kroku [2]:

1. **Nastavení parametrů** – stanovují se parametry, které určují chod celého algoritmu. Jde o následující parametry: CR – práh křížení, NP – velikost populace, F – mutační konstanta, D – počet argumentů účelové funkce, nebo rozměr jedince. Podrobnější informace o nastavení těchto parametrů byla uvedena v oddílu „Parametry“. V prvním kroku je také nutno nadefinovat prototyp jedince, který jsem popsal v oddílu „Populace“.
2. **Tvorba populace** – probíhá vygenerováním množiny jedinců podle prototypu.
3. **Započetí cyklu generace** – tento cyklus se provádí postupným vybíráním každého jedince až do konce populace. Pro každého jedince se pak provádí evoluční cyklus.
4. **Evoluční cyklus** – cyklus, ve kterém probíhá proces mutace a křížení. Tyto procesy byly popsány v předchozím textu.

5. **Testovaní naplnění ukončovacích parametrů** – algoritmus bude ukončen pouze tehdy, kdy proběhne zadány počet generací, který je zadán uživatelem. Ale uživatel také může vymyslet a naprogramovat vlastní ukončovací podmínky.
6. **Vyhodnoceni** – během každé generace se počítá hodnota účelové funkce, pro kterou se DE používá. Nelepší hodnoty účelové funkce se ukládají do zvláštního vektoru (vektor historie), který znázorňuje průběh evolučního procesu.

Celý evoluční proces tvorby jednoho jedince pro novou populace je schematicky znázorněn na obrázku 5.1.



	Jedinec 1	Jedinec 2	Jedinec 3	Jedinec 4	Jedinec 5	Jedinec 6	Jedinec 7
CV	3,6944	79,1016	57,4536	3,1620	3,5515	12,4326	0,3475
Parametr 1	8,0533	71,3354	17,1113	4,1457	13,7376	61,6385	57,3325
Parametr 2	9,2498	5,4905	42,7769	25,3730	65,4701	10,2315	17,1861
Parametr 3	1,1240	6,7742	16,0488	46,0285	50,7382	47,0758	0,0350
Parametr 4	10,1876	0,2486	10,3424	29,3259	16,0363	43,7628	17,4244
Parametr 5	9,7273	2,3160	0,6998	33,5473	34,7929	32,0120	71,8706
Parametr 6	11,2942	18,2332	76,2471	3,2480	5,1033	0,2021	17,4752

Diferenční váhový vektor	Váhovaný diferenční vektor	Šumový vektor
-5,6843	-4,5474	52,7851
-56,2203	-44,9762	-27,7901
-49,6142	-39,6914	-39,6564
-5,8487	-4,6789	12,7454
-25,0656	-20,0525	51,8181
6,1909	4,9527	22,4280

Zkušební vektor
14,9452
52,7851
-27,7901
6,7742
12,7454
2,3160
18,2332

	Jedinec 1	Jedinec 2	Jedinec 3	Jedinec 4	Jedinec 5	Jedinec 6	Jedinec 7
CV	1,6148	14,9452					
Parametr 1	5,9285	52,7851					
Parametr 2	11,6530	-27,7901					
Parametr 3	30,5677	6,7742					
Parametr 4	67,6060	12,7454					
Parametr 5	45,3004	2,3160					
Parametr 6	18,8684	18,2332					

Obrázek 5.1 – Princip diferenciální evoluce (převzato z [2])

5.3. Stagnace

Pravda je, že ani diferenciální evoluce nemá samé výhody. Mezi tyto nevýhody patří stagnace, která je vlastní tomuto algoritmu. To je jev, při kterém dohází k zastavení minimalizace hodnoty účelové funkce i když nebylo dosaženo globálního extrému. V případě použití evolučních algoritmů dochází ke konvergenci k suboptimálnímu řešení za určitých podmínek:

- Účelová funkce optimalizačního procesu je v lokálním extrému
- Populace není diverzibilní
- Optimalizace probíhá pomalu nebo neprobíhá vůbec

Na rozdíl od těchto obecných příznaků, při použití optimalizačního algoritmu DE, za určitých podmínek může dojít ke stagnaci navzdory faktoru, že:

- Populace není v lokálním extrému
- Populace je pořad diverzibilní
- Vznikají nový jedinci, ale optimalizační proces neprobíhá

Tato situace se nazývá stagnace. Důvod stagnace je takový, že se pomocí mutace dá vygenerovat jenom konečný počet kandidátských řešení. A v případě, že žádné z nich nevylepší hodnotu účelové funkce, pak dojde ke stagnaci. Stagnace závisí na řídicích parametrech DE. Pro snížení rizika vzniku stagnace je třeba zajistit dostatečný počet nových možných řešení (např. nastavit $NP > 20$) a zvýšit jejich schopnost získat místo v nové populaci.

Kapitola 6

Praktická část

Cílem praktické části této bakalářské práce bylo naimplementovat evoluční algoritmus a aplikovat ho na problém prediktivního řízení. Implementovaným algoritmem je DE, který byl popsán v předchozí kapitole č. 5. Tato metoda byla implementována pomocí prostředí MATLAB a zdrojový kód je součástí přiloženého DVD.

Metoda řízení procesu MPC neboli prediktivního řízení, která je popsána v kapitole č. 4 přímo užívá model řízeného procesu. Proto jsem musel na začátku zvolit problém, který budu řešit pomocí navrženého řízení, a model, který bude popisovat daný problém.

6.1. Diskrétní prediktivní řízení hladiny v nádrži

Úloha řízení hladiny v nádrži byla zvolena ze skript „Základy prediktivního řízení" [12], protože je dobře prostudovaná a je poměrně jednoduchá, což dovoluje snadno diskutovat a porovnávat výsledky navřeného řízení.

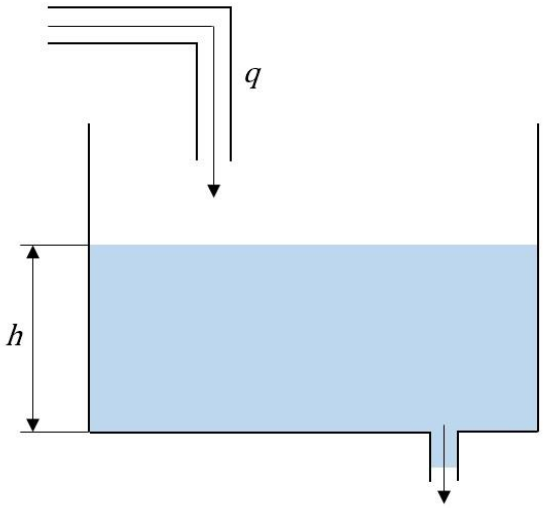
6.1.1. Zadaní

Pro praktickou část této bakalářské práce bylo použito zadaní „Případová studie – diskrétní prediktivní řízení hladiny v nádrži" (viz. [12]):

Vztah mezi vstupním průtokem kapaliny do nádrže a výškou hladiny v nádrži je popsán ve tvaru diskrétní přenosové funkce prvního radu (6.1).

$$\mathrm{Y}(z) = \frac{b_1}{z+a_1}\mathrm{U}(z) \tag{6.1}$$

kde $\mathrm{U}(z)$, $\mathrm{Y}(z)$ jsou z-obrazy vstupu (průtok kapaliny na vstupu do nádrže q) a výstupu (výška hladiny v nádrži h), a a_1, b_1 jsou parametry přenosu systému. Úloha je schematické zobrazena na obrázku 6.1.



Obrázek 6.1- Schematický nákres nádrže s kapalinou (převzato z [12])

Úkolem je převod modelu procesu z podoby diskrétní přenosové funkce do diskrétního stavového tvaru, návrh a následná implementace navrženého algoritmu diskrétního prediktivního řízení s ustupujícím horizontem řízení pro regulaci výšky hladiny v nádrži pomoci vstupního průtoku kapaliny v programovém prostředku MATLAB.

6.1.2. Převod do stavového tvaru

V rámci této bakalářské práce byly použity konkrétní hodnoty parametrů přenosu systému $a_1 = -0{,}9$, $b_1 = 0{,}02$. Tyto parametry jsou určené pro konkrétní systém některou z metod experimentální identifikace. Pro tyto parametry lze obecnou přenosovou funkci (6.1) přepsat ve tvaru (6.2) [12]

$$\mathrm{Y}(z) = \frac{0{,}02}{z-0{,}9}\mathrm{U}(z) \tag{6.2}$$

Pro převod přenosové funkce do stavového tvaru v prostředí MATLAB existuje funkce `tf2ss`. Ale v našem případě máme jednoduchý diskrétní přenos prvního řadu, proto se stavový popis dá sestavit přímo z parametru přenosové funkce:

$$A_m = -a_1; B_m = 1; C_m = b_1; D_m = 0$$

celkově tedy

$$x_m(k+1) = 0{,}9 \cdot x_m(k) + u(k) \tag{6.3}$$

$$y(k) = 0{,}02 \cdot x_m(k) \tag{6.4}$$

kde $u(k) = q(k), y(k) = h(k)$, x_m je stavová proměnná a k - diskrétní čas.

6.1.3. Návrh algoritmu DE

Na začátku návrhu algoritmu je třeba přesně definovat k čemu algoritmus bude sloužit. Hlavním úkolem algoritmu DE pro prediktivní řízení systémů je hledání nejlepších (optimálních) akčních zásahů „u", které vedou výstup řízeného dynamického systému k žádané hodnotě „w". Akční veličiny mohou nabývat různých hodnot ze spojitého nebo diskrétního intervalu. To znamená, že počet možných akčních veličin může být až nekonečně velký. Tento problém může zkomplikovat hledání optimálního akčního zásahu. Dalším následkem může být to, že některé zásahy budou nerealizovatelné z důvodu omezení zařízení systému.

Proto z optimalizačního hlediska je vhodné omezit počet možných hodnot hledaných parametrů a definovat určitý rozsah těchto hodnot. Tento rozsah byl zvolen na intervalu od -10 do 10, který se dá zapsat ve vektorovém tvaru:

$$\mathbf{U_r} = [-10, -5, -2, -1, 0, +1, +2, +5, +10] \tag{6.5}$$

kde hodnoty „$+10$" a „-10" odpovídají největšímu vlivu na akční zásah, hodnoty „$+1$" a „-1" odpovídají nejmenšímu vlivu a hodnota „0" znamená žádnou změnu. To neznamená, že akční zásahy budou přímo nabývat hodnot z vektoru $\mathbf{U_r}$. Tyto hodnoty se přepočítají pro každý případ následujícím způsobem. Uživatel musí definovat hodnoty w_L a w_H, které definují dolní a horní hranice, ve kterých se požadovaná hodnota řízené veličiny bude pohybovat. Pak se vektor možných hodnot vypočítá podle (6.6).

$$\mathbf{U_o} = K_\mathbf{o} \cdot \mathbf{U_r}, \tag{6.6}$$

kde

$$K_o = w_H - w_L \tag{6.7}$$

V našem případě požadovanou hodnotou je výška hladiny v nádrži. Například pro zvolené hodnoty $w_L = 0.2$ a $w_H = 0.6$ bude mít vektor možných hodnot tvar

$$\mathbf{U_o} = [-0.2, -0.1, -0.04, -0.02, 0, +0.02, +0.04, +0.1, +0.2] \tag{6.8}$$

Aby bylo možně dosáhnout s požadovanou přesností žádané hodnoty „w" byly hledány přírůstky akčních zásahů místo absolutních hodnot akčních veličin. To vede k tomu, že daný algoritmus DE bude vypočítávat optimální hodnotu přírůstku akčního zásahu ($\Delta u(k)$). To znamená, že algoritmus určí, o kolik je třeba zmenšit nebo zvětšit

akční veličinu pro získaní nejlepšího (optimálního) výsledku. Pak hodnota akčního zásahu bude stanovena podle vztahu (6.9)

$$u(k) = u(k-1) + \Delta u(k) \tag{6.9}$$

Výběr účelové funkce

Jak bylo uvedeno v předchozím textu, hlavní cíl algoritmu spočívá v dosáhnutí požadované veličiny optimálním způsobem. Jinými slovy algoritmus musí minimalizovat rozdíl mezi požadovanou a skutečnou veličinou. Proto je vhodné použít účelovou funkci ve tvaru (6.10).

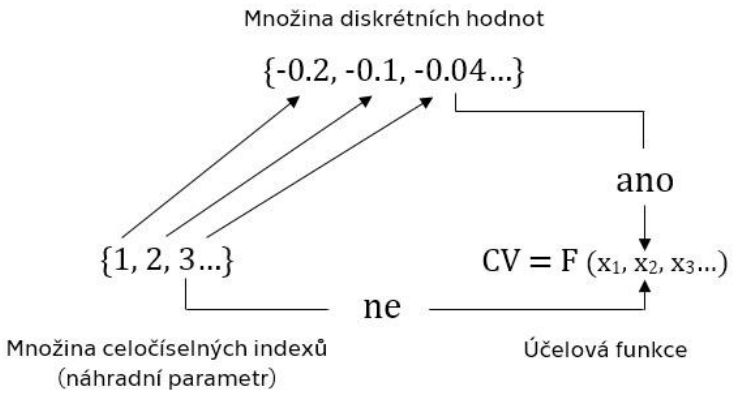
$$CV = \sum_{i=1}^{N} (w(k) - y(k))^2 \tag{6.10}$$

Dimenze problému

Dimenze problému D je jedním z řídicích parametrů DE, který udává počet argumentů účelové funkce. V našem případě argumenty účelové funkce jsou určené hodnoty přírůstků akčních zásahů pro zvoleny horizont. To znamená, že parametr D pořád definuje velikost každého vektoru jedince z populace, ale v tomto případě vektor každého jedince představuje posloupnost možných (kandidátských) přírůstků vstupních hodnot. Z tohoto důvodu v navrženém algoritmu DE parametr D uvádí „délku pohledu do budoucnosti", a tím pádem není ničím jiným něž horizontem predikce. Proto ve vztahu 6.10 místo „N" můžeme přímo používat parametr D.

Definováni prototypu

Na začátku návrhu algoritmu jsem zmínil, že navržený algoritmus bude pracovat jen s omezenou množinou diskrétních hodnot. Objevuje se tady další problém, a to ten, že použitá množina diskrétních hodnot není rovnoměrně rozdělena. Řešení je následující. Pro množinu diskrétních hodnot např. {-0.2, -0.1, -0.04...} se vytvoří celočíselný index pro daný parametr, nabývající hodnot $\{1, 2, 3, 4 \ldots\}$, který nahradí diskrétní parametr. S tímto „falešným" parametrem se pak pracuje jako s normálním celočíselným parametrem, ale při vyhodnocovaní účelové funkce se nedosadí aktuální celočíselné hodnoty, ale hodnoty z původní diskrétní množiny, na které tyto celočíselné parametry (indexy) ukazují [2]. Tento princip je schematicky znázorněn na obrázku 6.2.



Obrázek 6.2 - Princip nahrazení diskrétních parametrů (převzato z [2])

Navržený algoritmus pro prediktivní řízení, podle vektoru $\mathbf{U_r}$, bude používat jenom devět různých diskrétních hodnot. Proto indexy budou nabývat hodnot z intervalu $\langle 1,\ 9\rangle$.

Inicializace algoritmu

Jako ve většině evolučních algoritmů, inicializace algoritmu DE je jednoduchá. První populace je sestavena z jednotlivců, které jsou náhodně vytvořeny podle prototypu. Celý krok definování prototypu a tvorby první populace je realizován v prostředí MATLAB níže uvedeným programovacím kódem:

```
%Nadefinovani prototypu:
    Specimen{1,1}='Integer';
    Specimen{1,2}=[1 9];
%Tvorba prvni populace podle prototypu:
for Naktivni=1:NP
   for Narg=1:D
      Typarg=Specimen{1,1};
      ODDO=Specimen{1,2};
      OD=ODDO(1);
      DO=ODDO(2);
        if Typarg=='Integer'
          Populant(Naktivni,Narg)=unidrnd(DO+1-OD)+OD-1;
        else
          Populant(Naktivni,Narg)=(DO-OD)*rand+OD;
        end;
      %vypocet vstupnich a vystupnich hodnot
      ind = Populant(Naktivni,Narg);
      deltaU=Uo(ind);
      U0 = U0+deltaU;
      xm0 = Am*xm0 + Bm*U0;
      y0  = Cm*xm0;
      %vypocet ucelove funkce kazdeho jedince:
      CVo = CVo + (W - y0)^2;
   end;
   CV(Naktivni) = CVo;
end;
```

Při tvorbě první populace jedinců se taky vypočítávají hodnoty akčních zásahů a hodnoty účelové funkce, které patří příslušnému jedinci. Tyto hodnoty budou postupně optimalizované v další částí algoritmu, která se nazývá evoluční cyklus.

Evoluční cyklus

Evoluční cyklus je nejdůležitější částí celého algoritmu, protože v tomto cyklu se budou vytvářet noví jedinci, nové populace, a to znamená i nové kandidátské řešení s využitím procesů mutace, křížení a výběru přeživších jedinců. Každý s těchto cyklů je podrobně popsán v oddílu č. 5.2.

Jak bylo uvedeno v předchozím textu, každý jedinec z populace má několik parametrů stejného typu. Veškeré tyto parametry představují celočíselné indexy, které patří diskrétním hodnotám přírůstků akčních zásahů. Při práci s celočíselnými hodnotami parametrů mohou nastat další problémy během evolučního cyklu, a hlavně během procesu mutace. Z oddílu č. 5.2.2 je známo, že se mutace provádí podle vztahu X.X. V případě, že hodnota mutační konstanty F nebude celočíselná, budou parametry nabývat reálných ale ne celočíselných hodnot. Řešení tohoto problému je nejjednodušší. Do zkušebního vektoru se nezapíše vypočtená hodnota, ale zapíše se celočíselný index z intervalu $\langle 1,\ 9\rangle$. Větší problém může nastat v situaci, kdy vypočtená hodnota nebude ani přibližně patřit do intervalu $\langle 1,\ 9\rangle$. V tomto případě použití metody dosazení nejbližších indexů bude mít za důsledek to, že se ve zkušebním jedinci nejčastěji budou objevovat krajní hodnoty intervalu. Následkem bude to, že proces hledání kandidátských řešení přestane být náhodným a celý algoritmus DE ztratí svůj význam. Řešení tohoto problému je velmi důležité, ale na druhou stranu je docela jednoduché. Místo této nevhodné vypočtené hodnoty se do zkušebního vektoru dosadí nová náhodně vybraná hodnota (index) z intervalu $\langle 1,\ 9\rangle$. Obě tyto řešení se dají zapsat programovacím kódem:

```
for i = 1:D
   if ZkusV(i)<OD || ZkusV(i)>DO
      ZkusV(i)= unidrnd(DO+1-OD)+OD-1;
   end
      [CIS, numb]= min(abs(Un-ZkusV(i)));
      ZkusV(i)=numb;
      ind = ZkusV(i);
end
```

V dalším kroku je realizován proces křížení. Během křížení se budou právě vytvářet noví (zkušební) jedinci. Každý nový jedinec je dan kombinaci parametrů šumového a aktivního jedince. Pro každého nového jedince se znovu vypočítává hodnota účelové funkce. To znamená, že se pomocí parametrů zkušebního jedinců stanoví hodnoty akčních zásahů a příslušné výstupní veličiny. Účelová funkce právě představuje kvadratickou odchylku výstupní a požadované veličiny. Detailnější informace o tvorbě zkušebních jedinců čtenář může najit v oddílu č. 1.1.1.

Jakmile všichni jedinci z populace projdou přes procesy mutace a křížení bude sestavena další populace jedinců. K tomu se využije mechanizmus výběru přeživších (oddíl č. 3.1.6). Zkušební jedinec, který bude mít optimálnější hodnotu účelové funkce (menší kvadratickou odchylku) než příslušný původní jedinec, přesune se do nove populace. V opačném případě do nove populace se přesune původní jedinec. Realizace mechanizmu výběru přeživších v prostředí MATLAB je následující:

```
for Naktivni=1:NP                    %Postupni vyber aktivniho jedince
  if abs(CV(Naktivni))>abs(CVzkusV) %Porovnani CV zkusebniho s CV aktivniho
      CVP(Naktivni)=CVzkusV;         %Pouze pro docasne uložení
      PopulantP(Naktivni,:)=ZkusV;
        else
      PopulantP(Naktivni,:)=Populant(Naktivni,:);
      CVP(Naktivni)=CV(Naktivni);
  end
end                                  %Konec populace
Populant=PopulantP;                  %Aktualizace nove generace
```

Z praktického důvodu se na konci každé generace ukládají nejlepší parametry jedinců a příslušné hodnoty účelové funkce (CV):

```
[KRIT, kde]=min(CV);
PrubCV(Ngen)=KRIT;
PrubPop(Ngen,:)=Populant(kde,:);
```

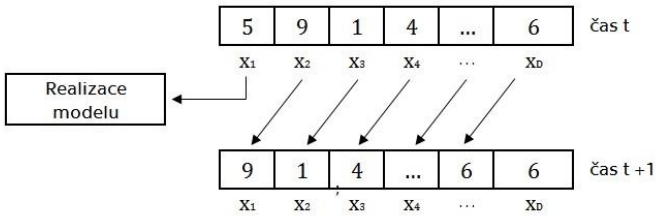
Během simulace v každém kroku diskrétního času se provádí určitý počet evolučních cyklu. Tento počet je dan řídicím parametrem G (oddíl č. 5.2), který udává počet generací. To znamená, že před realizací každého kroku algoritmus DE poskytuje velké množství možných řešení. Toto množství je dáno nejen počtem generací ale i počtem jedinců (NP) v populaci. Proto je docela velká šance, že v každém kroku simulace bude realizován optimální akční zásah. Pravděpodobnost získaní nejlepšího akčního zásahu je tím větší, čím větší bude počet generací a velikost populace. Ale tyto parametry mají omezenou velikost, protože jejich velikost v přímém poměru ovlivňuje čas běhu algoritmu.

Realizace modelu

Z předchozího textu je známo, že výsledkem algoritmu DE je optimální řešení. Řešením je vektor o délce D (řídicí parametr), který představuje posloupnost přírůstků akčních zásahů. V našem případě to znamená, že postupná realizace těchto zásahu změní polohu hladiny v nádrži z počáteční na nějakou určitou polohu. A podle algoritmu DE tato poloha by měla být optimální. Ale pravda je, že se po realizaci prvního akčního zásahu změní poloha hladiny a vznikne další množství možných řešení. Proto je velká šance, že zbylá část aktuálního řešení přestane být optimální. Řešením je realizace jenom prvního parametru (akčního přírůstku) nejlepšího jedince. V tomto případě se pro další krok znovu spustí evoluční cyklus a vypočítá se další optimální řešení. Ve výsledku, v každém kroku bude k dispozici nejlepší (optimální) řešení pro několik dalších kroků, ale zrealizuje se jenom první.

Ale hledaní nových optimálních řešení znovu by bylo nepraktické. Lepším řešením bude, když se posunou všichni parametry nejlepšího jedince o jedno místo (viz obr. 6.3). Z důvodu posunutí parametrů se změní hodnota účelové funkce, kterou je třeba znovu přepočítat. Proces posunutí parametrů v prostředí MATLAB může být realizován jako:

```
for i = 1:D-1
   Populant(:,i)=Populant(:,i+1);
end
```



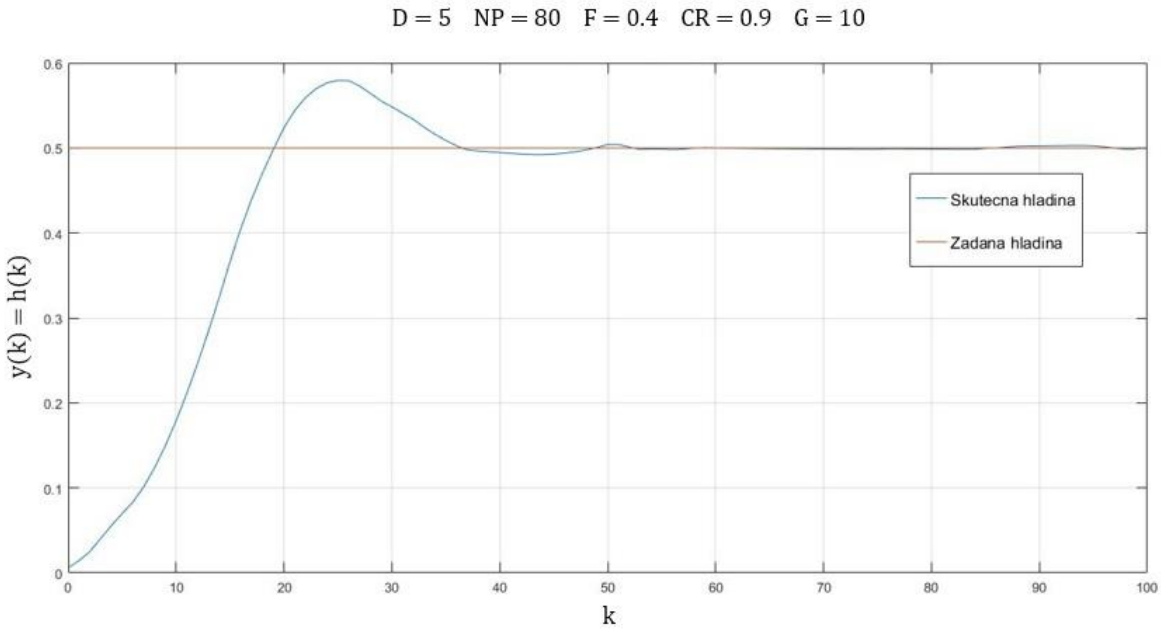
Obrázek 6.3 - Princip posunutí parametrů nejlepšího jedince

Každý z těchto procesů se opakuje v každém kroku simulace. Cely algoritmus DE pro prediktivní řízení byl realizován v prostředí MATLAB podle výše uvedeného návrhu.

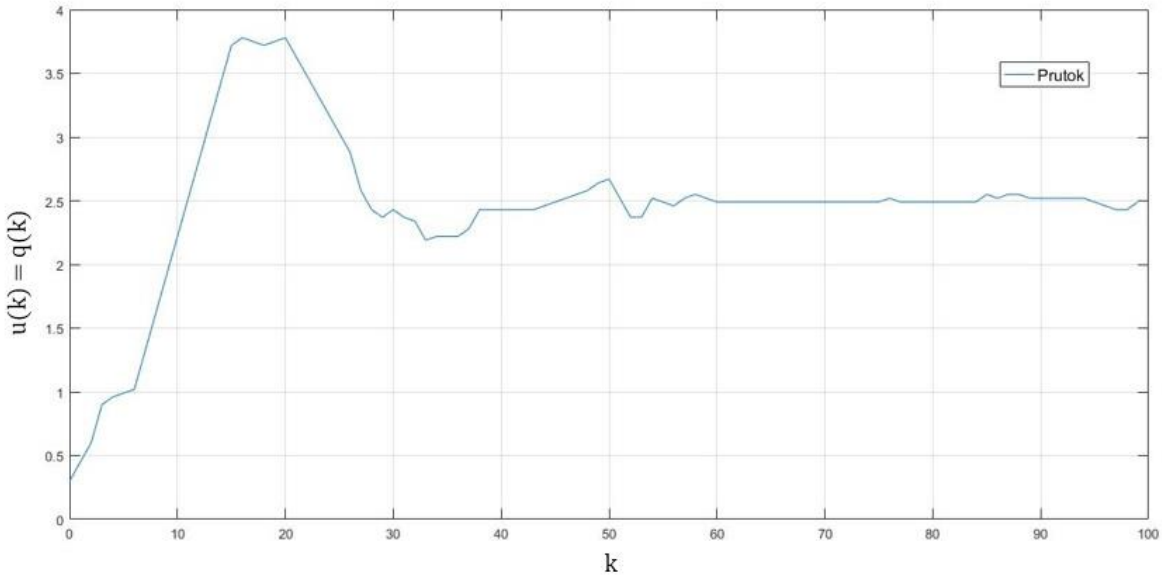
6.1.4. Testování algoritmu

Konstantní výška hladiny

Následující oddíl je věnován simulačnímu ověření navrženého algoritmu DE. Nejdřív simulace byla provedena pro konstantní požadovanou výšku hladiny v nádrži. Průběh regulace a nastavené řídicí parametry jsou zobrazeny na obrázku 6.4. Červená barva zobrazuje požadovanou výšku hladiny a modrá barva – skutečnou. Na dalším obrázku 6.5 je zobrazen průběh akčních zásahu během dané simulace.

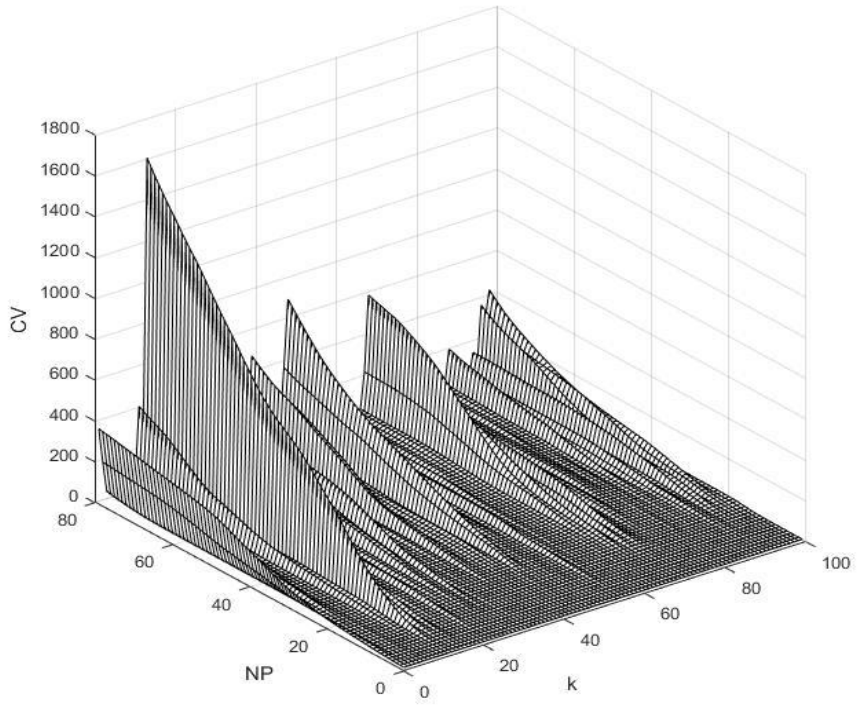


Obrázek 6.4 - Výstupy systému



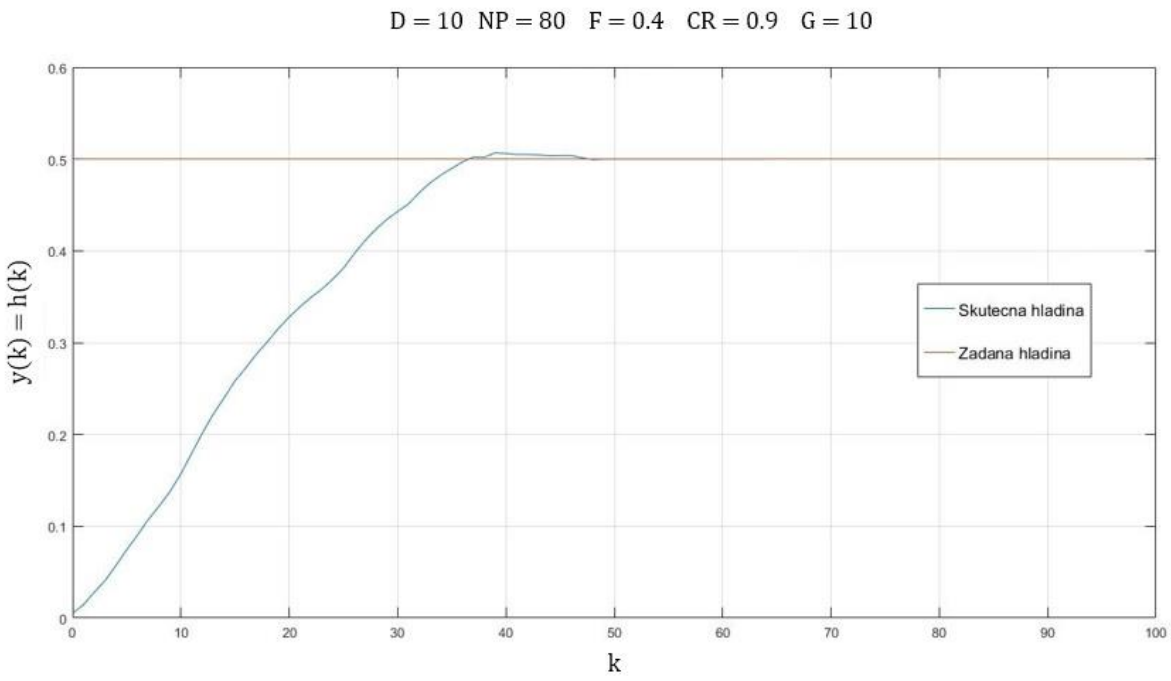
Obrázek 6.5 - Vstupy systému

Velmi důležitou roli při testování algoritmu hraje průběh hodnot účelové funkce každého jedince. Tento průběh dovoluje zkontrolovat, že každý jedinec představuje různé kandidátské řešení s odlišnými hodnotami účelové funkce. Nejlepší jedinci (řešení) jsou umístěny na začátku každé populace, proto je také možné zkontrolovat, že v každém kroku byly realizované optimální (nejlepší) hodnoty akčních zásahů. Grafické zobrazení hodnot účelové funkce během simulace je na obrázku 6.6 (NP – velikost populace, CV – hodnota účelové funkce, k – krok simulace).

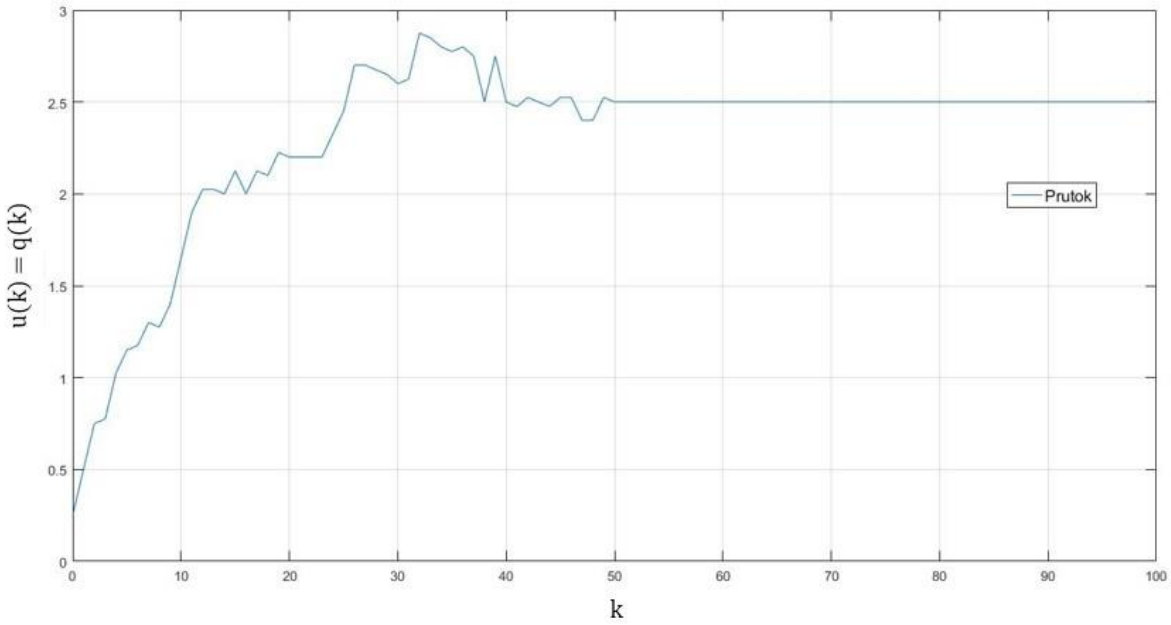


Obrázek 6.6 - Průběh účelové funkce

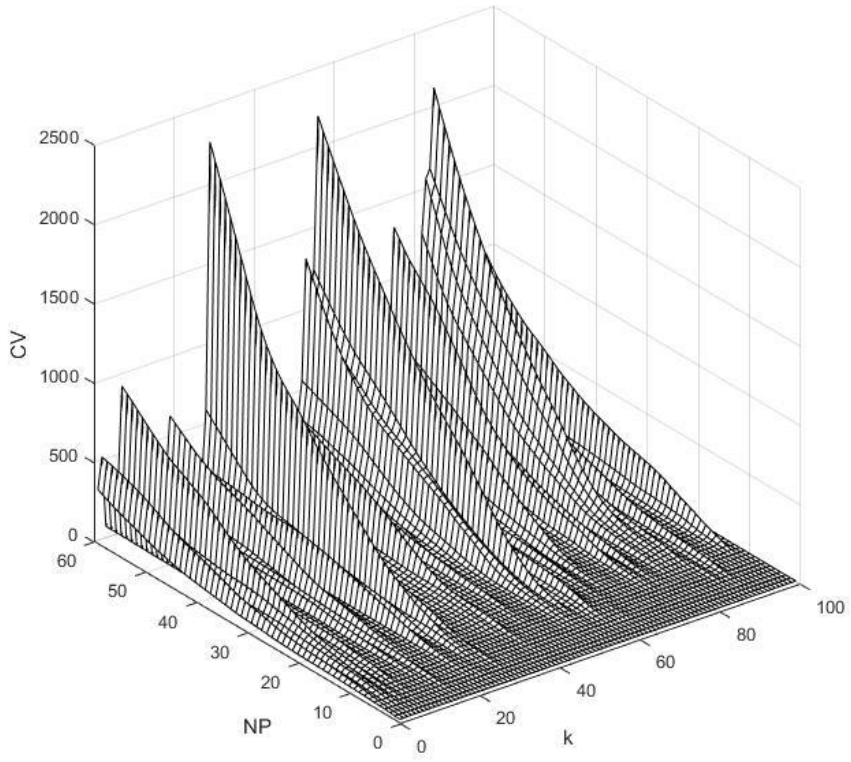
Veškeré řídicí parametry DE ovlivňují chod celého algoritmu řízení. Ale největší vliv na regulační pohod má horizont predikce, to znamená parametr D. Proto v dalším příkladu simulace byly nastavené stejné řídicí parametry, jako v předchozí simulace. Ale parametr D byl zvýšen do hodnoty deset. Jinými slovy byla použita predikce deseti kroků dopředu. Výsledky jsou grafické zobrazený na obrazcích 6.7, 6.8 a průběh účelové funkce je na obrázku 6.9.



Obrázek 6.7 - Výstupy systému



Obrázek 6.8 - Vstupy systému



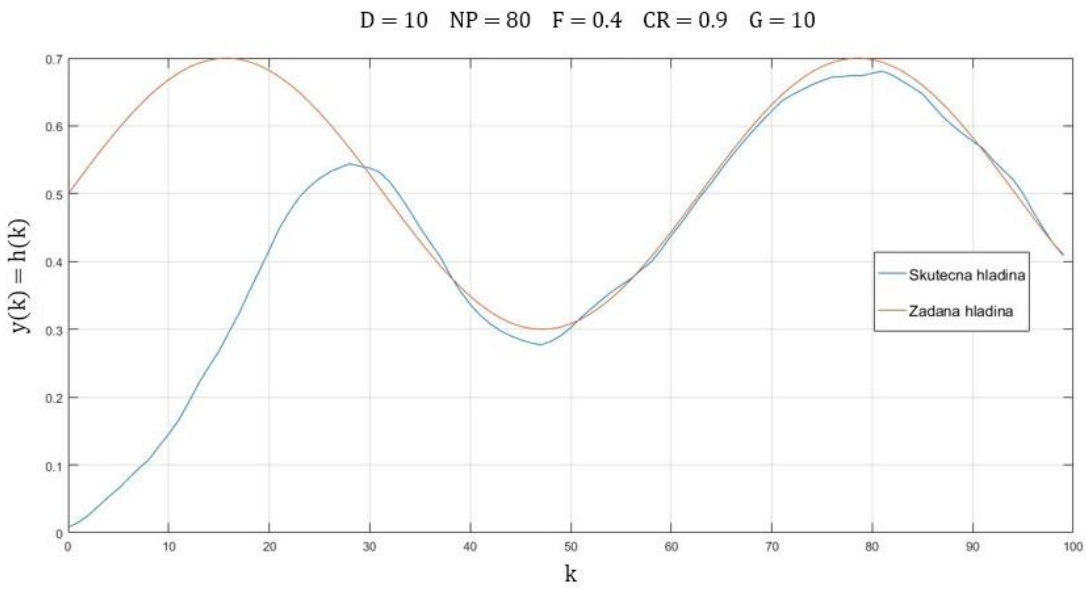
Obrázek 6.9 - Průběh účelové funkce

Při porovnaní obrázků 6.4 a 6.7 je snadno vidět, že při použití stejných řídicích parametrů, ale zároveň delších vektorů řešení (tzn. vetší horizont predikce) můžeme dosáhnout lepší kvality regulace. Další faktor ovlivňující průběh regulace je interval požadovaných hodnot, který určuje hranice, ve kterých se výstupní veličina může pohybovat. Čím větší bude tento interval, tím bude větší rychlost regulace. Ale to znamená i to, že při použití většího intervalu bude minimální hodnota akčního zásahu také větší. Z toho důvodu nebude možné dosáhnout některých přesnějších hodnot výstupní veličiny, která bude kmitat kolem požadované hodnoty. Jinými slovy, kvalita regulace bude horší. O tom, který parametr regulace (rychlost nebo kvalita) je důležitější, rozhoduje uživatel algoritmu podle vlastních požadavků a vlastní zkušenosti.

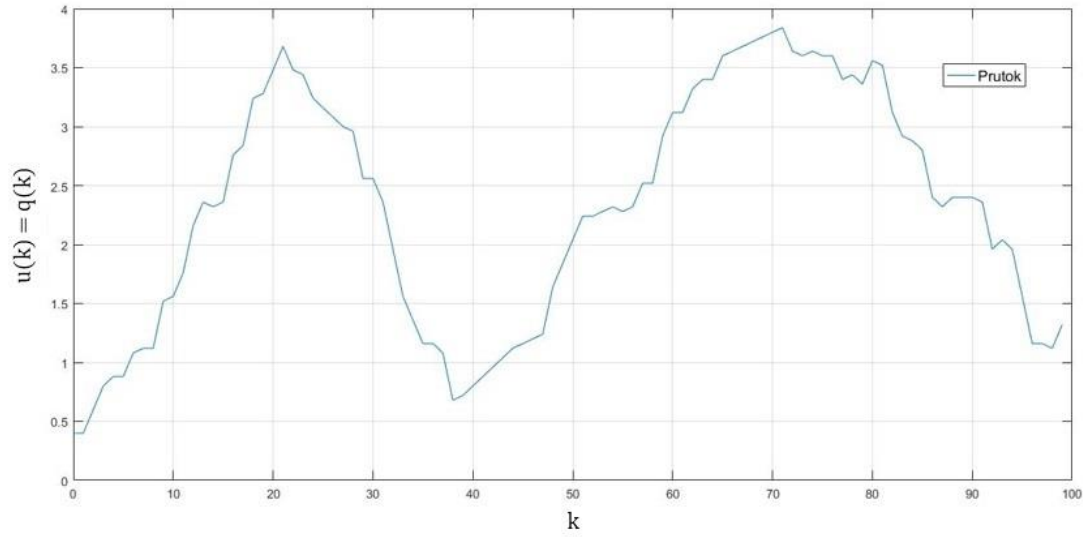
Proměnná výška hladiny

Z předchozího textu je vidět, že navržený algoritmus DE pro prediktivní řízení může být použit pro nastavení výšky hladiny na požadovanou konstantní hodnotu. Proto v dalším kroku byl algoritmus vyzkoušen na složitějším příkladu. V tomto případě požadovaná výška hladiny nebyla zadána přímkou (konstantní hodnotou), ale byla zadána ve tvaru sinusoidy podle (6.11). Výsledky řízení jsou zobrazeny na obrázcích 6.10, 6.11 a 6.12.

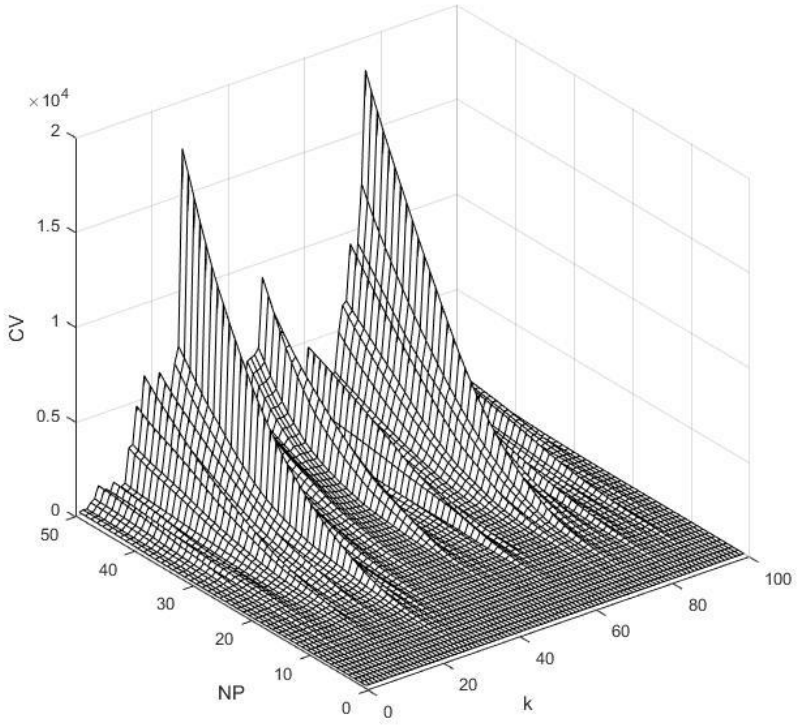
$$H = H_o + 0.2 \cdot \sin(0.1 \cdot k),\ kde\ H_o = 0.5 \tag{6.11}$$



Obrázek 6.10 - Výstupy systému



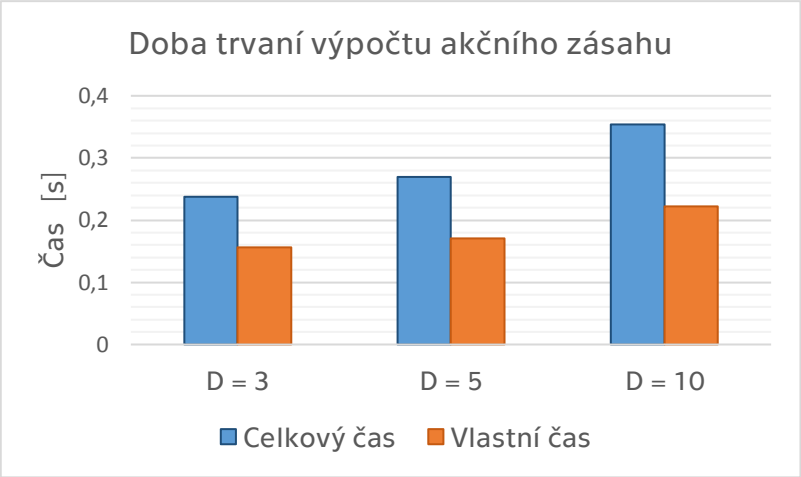
Obrázek 6.11 - Vstupy systému



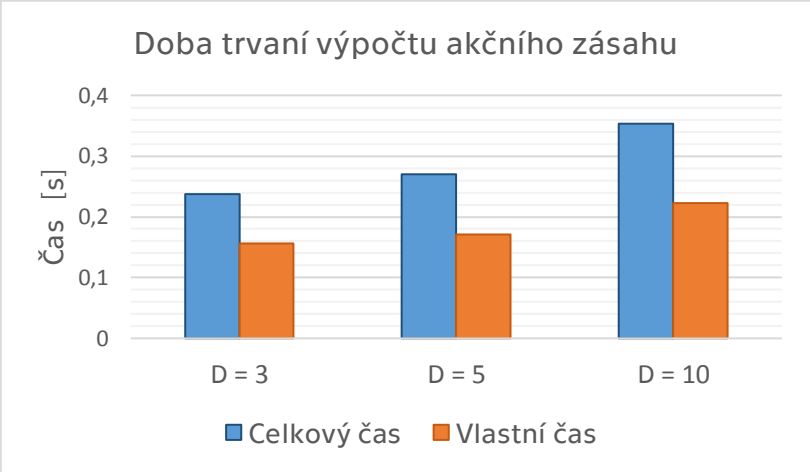
Obrázek 6.12 - Průběh účelové funkce

Vyhodnocení času běhu algoritmu

Cílem navrženého algoritmu DE není jenom prediktivní řízení zadaného modelu (řízení výšky hladiny), ale řízení celé množiny procesů. Všechny procesy však potřebují různou rychlost řízení. Aby bylo možné stanovit množinu procesu, na kterou bude použit navržený algoritmus je třeba zjistit, jak dlouho trvá jeden výpočetní cyklus. Testovaní doby průběhu jednoho cyklu bylo provedeno pro různé délky horizontu predikce. Pro každou délku horizontu predikce byl čas měřen několikrát. Střední hodnoty výsledků testů pro konstantní výšku hladiny jsou na obrázku 6.13 a pro proměnnou výšku hladiny jsou na obrázku 6.14. Modrá barva odpovídá celkovému časovému cyklu. To je čas potřebný nejenom pro výpočet hlavních funkcí, ale i pro výpočet podřízených funkcí. Červenou barvou je zobrazen vlastní čas cyklu.



Obrázek 6.13 – Doba trvaní výpočtu akčního zásahu pro řízení konstantní výšky hladiny



Obrázek 6.14 – Doba trvaní výpočtu akčního zásahu pro řízení proměnné výšky hladiny

6.2. Výsledek práce s algoritmem

Simulační ověření navrženého algoritmu DE bylo provedeno na několika příkladech. Zaprvé byly provedeny simulace řídicího algoritmu pro jednodušší požadovanou funkci. V rámci těchto simulací jsem také diskutoval vliv délky horizontu predikce (parametr D) na chod regulace. Zadruhé bylo navřené řízení vyzkoušeno pro složitější požadovanou funkci ve tvaru sinusoidy.

Podle výsledků uvedených v předchozím oddílu č. 6.1.4 mužů konstatovat, že se ve všech provedených testech algoritmus osvědčil výborně. Během testů jsem ověřil, že se dá velmi snadno řídit model procesu pomocí omezeného počtů hodnot akčních veličin. Vhodnou volbou rozsahu těchto hodnot může uživatel nastavit požadovanou rychlost a kvalitu regulace. Algoritmus DE poměrně rychle hledá optimální řešení

problému a provádí optimalizaci těchto řešení v každém kroku simulace. Tato vlastnost algoritmu se dobře projevuje v případech, kdy požadovaná hodnota výšky hladiny není konstantní. To znamená, že algoritmus DE opravdu nerozhoduje o tom, k čemu slouží dané řešení problému, ale rozhoduje o kvalitě tohoto řešení.

Nejdůležitějším výsledkem provedených testů je, že optimalizační algoritmus DE byl navržen správně a může být použit pro řešení mnoha problémů metodou prediktivního řízení.

6.3. Budoucnost algoritmu

Z výsledků testovaní navrženého algoritmu DE je vidět, že tento algoritmus může být použit pro prediktivní řízení procesu. Ale v rámci praktické části této bakalářské práce byl algoritmus vyzkoušen jenom na poměrně jednoduchém modelu řízení výšky hladiny v nádrži. Proto práce s algoritmem DE má určitě svoji budoucnost. To znamená, že navržený algoritmus může být vyzkoušen na komplikovanějších modelech a procesech. Další mozností pokračovaní práce s navrženým algoritmem může být jeho použití pro nelineární modely. Navržené řízení procesu musí být také ověřeno nejen simulačně, ale i na reálné úloze. Proto může být v budoucnosti zvolena nějaká vhodná laboratorní úloha, která by mohla být řízená pomoci algoritmu DE.

Kapitola 6

Závěr

Úkolem práce je seznámení se s algoritmem diferenciální evoluce, návrh algoritmu pro prediktivní řízení a studium tohoto algoritmu z hlediska optimalizace a účinností.

Základním nástrojem pro návrh algoritmu diferenciální evoluce pro prediktivní řízení bylo zvoleno prostředí MATLAB.

Prvním částí bakalářské práce bylo seznámení s evolučními algoritmy. Tyto algoritmy dnes patří mezi populární optimalizační techniky, jejichž oblasti používaní se neustále rozšiřuji. V návrzích těchto algoritmů je propojená teorie optimalizace s lidskými představami o biologických dějích, které slouží jako předloha optimalizačních postupu. Mezi novinky evolučních algoritmů patří algoritmus diferenciální evoluce, který byl podrobně probrán v rámci této bakalářské práce. Na základě těchto poznatků lze prohlásit, že algoritmus diferenciální evoluce může být úspěšně použít v široké oblasti různých procesů.

V dnešní době velmi populární metoda řízení, která využívá optimalizace, je prediktivní řízení. Principem této metody je na základě výpočtů předpovídat chování řízeného systému. Při použití prediktivního řízení se optimalizují hodnoty akčních zásahu. Proto je velmi důležité, který optimalizační algoritmus bude k tomu sloužit.

Další částí úkolu bakalářské práce bylo navrhnout algoritmus prediktivního řízení s využitím diferenciální evoluce a simulačně ověřit navržené řízení na určitém modelu. Pro návrh a testovaní algoritmu byla zvolená úloha řízení výšky hladiny v nádrži. Navržení algoritmu a jeho simulace byly realizovaný v prostředí MATLAB. Celý zdrojový kód algoritmu využívajícího diferenciální evoluce je součástí přiloženého DVD. Dle mého

názoru algoritmus prediktivního řízení využívající diferenciální evoluce byl navržen úspěšně. Další myšlenkou při návrhu řízení bylo použití diskrétních vstupních hodnot. Tato metoda dovolí použití algoritmu diferenciální evoluce pro méně přesné systémy. Jinými slovy jde o rozšíření oblasti použití navrženého řízení. V každé simulaci algoritmus dokázal poměrně rychle najít optimální hodnoty akčních zásahů. Kvůli tomu dosáhla kvalita regulace dostatečně vysoké úrovně. Během provedených simulací byl taky vyzkoušen vliv řídicích parametrů na chod regulace. Jedním z nejdůležitějších kritérií testovaní průběhu regulace byl čas, potřebný pro vypočet a realizaci jednoho kroku diskrétního času. Tento údaj je velmi důležitý, protože přímo ovlivňuje to, na které procesy může být navržené řízení implementováno. V rámci praktické částí této práce bylo navržené řízení vyzkoušeno nejen pro nastavení výstupní veličiny na požadovanou konstantní hodnotu, ale i pro regulace s proměnnou žádanou hodnotou. Přestože algoritmus má k dispozici omezený počet možných výstupních hodnot, kvalita regulace i pro případ s proměnnou požadovanou hodnotou byla na dostatečné vysoké úrovni.

Úkol bakalářské práce byl splněn.

Práce s navrženým řízení má perspektivu. V budoucnosti by bylo možné testovat algoritmus na komplikovanějších příkladech, například pro prediktivní řízení nelineárních systémů. Dalším velmi důležitým krokem ve vývoji navrženého řízení je jeho použití pro reálné systémy. Z výše uvedených faktů je vidět, že evoluční algoritmus diferenciální evoluce má široké a univerzální použití.

Seznam použité literatury

[1] Andreas Antoniou, Wu-Sheng Lu, Practical Optimization: Algorithms and Engineering Applications, New York: Springer Science+Business Media, LLC, 2007.

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

[3] Eiben A.E., Smith J.E., „Introduction to Evolutionary Computing," [Online]. Available: https://docs.google.com/file/d/0BzYHwqsaxbjQMmIMMlRmNFRsZ2M/edit. [Přístup získán 12 4 2017].

[4] Xinjie Yu, Mitsuo Gen, Introduction to Evolutionary Algorithms, London: Springer, 2010.

[5] CAMACHO E.F., BORDONS C., „Model Predictive Control. 2nd ed.," 2007. [Online]. Available: http://een.iust.ac.ir/profs/Shamaghdari/MPC/Resources. [Přístup získán 12 4 2017].

[6] PRICE K., STORN R., LAMPINEN J., Differential Evolution - A Practical Approach to Global Optimization, Berlin: Springer-Verlag, 2005.

[7] YANG Xin-She, Nature-inspired metaheuristic algorithms. 2nd ed., Frome: Luniver Press, 2010.

[8] PRICE K., „Genetic Annealing," *Dr. Dobb's Journal,* č. 220, pp. 127-132, 1994.

[9] PRICE K., STRON R. , „Minimizing the real functions of the ICEC'96 contest by Differential Evolution," v *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996.

[10] PRICE K., STORN R., „Differential Evolution - A simple evolutionary strategy for fast optimization," *Dr. Dobb's Journal,* č. 264, pp. 18-24, 1997.

[11] STORN R., PRICE K., „Differential Evolution – A Simple and Efficient," *Journal of Global Optimization,* č. 11, p. 341–359, 1997.

[12] MAREŠ J., HRNČIŘÍK P., Základy prediktivního řízení, Praha: Vysoka škola chemiko-technologická v Praze, 2012.

Seznam obrázků

Seznam tabulek