SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-5414-61615

Tvorba balíka nástrojov pre automatizáciu a riadenie v jazyku Julia

Diplomová práca

2017 **Bc. Natália Mikušová**

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-5414-61615

Tvorba balíka nástrojov pre automatizáciu a riadenie v jazyku Julia

Diplomová práca

Študijný program: Automatizácia a informatizácia v chémii a potravinárstve
Študijný odbor: 5.2.14. automatizácia
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky
Vedúci záverečnej práce: prof. Ing. Miroslav Fikar, DrSc.
Konzultant: Ing. Martin Klaučo

2017 **Bc. Natália Mikušová**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2016/2017
Evidenčné číslo: FCHPT-5414-61615

STU FCHPT

ZADANIE DIPLOMOVEJ PRÁCE

Študentka:	**Bc. Natália Mikušová**
ID študenta:	61615
Študijný program:	automatizácia a informatizácia v chémii a potravinárstve
Študijný odbor:	5.2.14. automatizácia
Vedúci práce:	prof. Ing. Miroslav Fikar, DrSc.
Konzultant:	Ing. Martin Klaučo

Názov práce: **Tvorba balíka nástrojov pre automatizáciu a riadenie v jazyku Julia**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Zadanie práce:

– Prehľad dostupných softvérových balíkov pre prostredie JULIA určených na optimalizáciu a riadenie procesov.
– Implementácia online MPC regulátorov vo výpočtovom prostredí JULIA.
– Implementácia explicitného MPC regulátora vo výpočtovom prostredí JULIA.
– Príprava uceleného softvérového MPC balíka pre prostredie JULIA.
– Príprava jednoduchej dokumentácie v anglickom jazyku na internetových úložiskách GIT alebo BITBUCKET.
– Porovnanie navrhovaných riešení s programom MATLAB.

Rozsah práce: 50

Zoznam odbornej literatúry:

1. Kvasnica, M. *Real-Time Model Predictive Control via Multi-Parametric Programming:Theory and Tools: Theory and Tools.* Saarbrücken : VDM Verlag Dr. Müller, 2009. 231 s. ISBN 978-3-639-20644-9.

Riešenie zadania práce od: 13. 02. 2017

Dátum odovzdania práce: 21. 05. 2017

Bc. Natália Mikušová
študentka

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Rada by som poďakovala vedúcemu mojej diplomovej práce prof. Ing. Miroslavovi Fikarovi, DrSc. a tiež môjmu konzultantovi Ing. Martinovi Klaučovi za ich odborné vedenie, rady, pripomienky a pomoc poskytnutú pri vypracovávaní práce.

Natália Mikušová
Bratislava, 2017

Abstrakt

Cieľom diplomovej práce bolo vytvoriť ucelený softvérový balík, ktorý implementuje implicitné aj explicitné prediktívne riadenie v jazyku Julia. V prvej časti sme poskytli základné informácie o jazyku Julia a prehľad nástrojov určených na optimalizáciu a riadenie, ktoré sú aktuálne jeho súčasťou. Hlavná časť práce sa zaoberala teoretickým návrhom implicitných a explicitných prediktívnych regulátorov. Tu sme ukázali, ako naformulovať optimalizačný problém a vyriešiť ho pomocou kvadratického alebo multiparametrického programovania. V závere sme sumarizovali všetky teoretické poznatky a vytvorili balík *mpc.jl*. Tento balík je voľne dostupný na internetovom úložisku GitHub spolu s jednoduchou dokumentáciou v anglickom jazyku.

Kľúčové slová:

Julia, implicitné prediktívne riadenie, explicitné prediktívne riadenie, kvadratické programovanie, multiparametrické programovanie

Abstract

The objective of master's thesis was to develope Julia software package which implements implicit and explicit model predictive control. In the first section we provided basic information about Julia programming language and overview of tools designed for mathematical optimization and process control that are currently available. The main part of the thesis deals with theoretical design of implicit and explicit predictive controllers. We have shown here how to formulate optimization problem and solve it using quadratic or multiparametric programming. At the end we summarized theoretical knowledge and created package called *mpc.jl*. This package is open-source and available on GitHub along with simple English documentation.

Key Words:

Julia, implicit model predictive control, explicit model predictive control, quadratic programming, multiparametric programming

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Julia, nový programovací jazyk, sa do povedomia verejnosti dostal prvýkrát v roku 2012. Za posledných 5 rokov si získal mnoho priaznivcov a v súčasnosti sa o ňom rozpráva čím ďalej tým viac. V programátorskej komunite je populárny najmä vďaka tomu, že kombinuje to najlepšie z jazykov ako sú C, Matlab alebo Python. Rovnako ako C, aj Julia je voľne stiahnuteľný programovací jazyk s úctyhodným výkonom. Podobností s Matlabom je tiež niekoľko, medzi najvýznamnejšie patrí syntax, dobrá numerická presnosť alebo rozsiahla knižnica matematických funkcií

Keďže Julia je relatívne nový jazyk, neobsahuje zatiaľ toľko nástrojov a balíkov ako Matlab alebo Python. To dáva možnosť užívateľom aktívne sa zapájať do vývoja jazyka a tým napomáhať k jeho zlepšovaniu. Existuje niekoľko balíkov, ktoré sa zaoberajú optimalizáciou a riadením, žiadny z nich však neponúka užívateľom možnosť navrhovať prediktívne regulátory. Preto sa stalo cieľom tejto práce nadviazať na aktuálne riešenia a implementovať implicitné a explicitné prediktívne riadenie do Julie.

Prediktívne riadenie je druh optimálneho riadenia. V praxi je veľmi obľúbené, pretože je vhodné ho použiť pri systémoch s viacerými vstupmi a výstupmi, ktoré majú určité fyzikálne ohraničenia. Navyše vie optimalizovať výkon tak, aby sme pri riadení dosahovali minimálne náklady alebo maximálny zisk. Balík, ktorým do Julie prispejeme, bude vhodný najmä pre akademickú sféru, pretože bude simulačne riešiť problémy prediktívneho riadenia v prostredí, ktoré je k dispozícií zadarmo, narozdiel od vyššie spomínaného Matlabu, kde sa cena licencie šplhá do výšky tisícov eur.

Kapitola 1

Julia

Julia je nový dynamický programovací jazyk určený na numerickú analýzu, vedeckotechnické výpočty a programovanie vo všeobecnosti. Jej vývoj začal ako indicko-americká spolupráca v roku 2009. Cieľom bolo vytvoriť jazyk, ktorý bude plnohodnotný, ale zároveň jednoduchý na používanie. Julia bola verejnosti prvýkrát predstavená v roku 2012 v blogovom príspevku. Autori v ňom uviedli svoje zámery a okrem toho prizvali ďalších nadšencov programovania, aby sa na projekte podieľali spolu s nimi. Vďaka tomu bola o rok neskôr uverejnená prvá oficiálna verzia Julie ako výsledok práce viac ako 100 prispievateľov. Odvtedy komunita významne narástla a odhaduje sa, že dnes má Julia okolo 250 tisíc užívateľov po celom svete. Jazyku Julia je každoročne venovaná konferencia JuliaCon a od roku 2017 existuje na univerzite MIT laboratórium orientované na jej testovanie a zlepšovanie.[1]

Julia bola navrhnutá tak, aby tvorba kódu v nej bola jednoduchá a efektívna, preto jej syntax je veľmi podobná iným, dobre známym jazykom (C) a výpočtovým prostrediam (Matlab). Obsahuje sofistikovaný kompilátor, vďaka ktorému sa zaraďuje medzi pokročilé, veľmi výkonné jazyky s dobrou numerickou presnosťou a možnosťou distribuovaného paralelného vykonávania príkazov. Jej súčasťou je rozsiahla knižnica matematických funkcií a navyše je schopná importovať voľne dostupné knižnice napísané v jazyku C a Fortran. Sú to napr. knižnice pre lineárnu algebru, generovanie náhodných čísel, spracovanie signálov a prácu s reťazcami. Táto vlastnosť Julie poskytuje užívateľom možnosť kombinovať funkcie a knižnice napísané v Julii s funkciami a knižnicami iných jazykov. Jednou z najväčších výhod tohto jazyka je, že je zadarmo a voľne dostupný. Práve preto vznikla komunita

vývojárov, ktorá vo veľkom prispieva rôznymi externými balíkmi a pomáha Julii napredovať. [2]

V posledných rokoch sa Julii podarilo zaujať tiež niektoré významné spoločnosti z finančného sektora. Britský líder v oblasti poisťovníctva Aviva používa Juliu pri vyhodnocovaní rizík investovania a Federal Reserve Bank of New York na vytváranie finančných modelov pri sledovaní ekonomiky Spojených štátov. [1]

Obr. 1.1: Logo jazyka Julia

Autori	Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman
Rok vydania	2012
Aktuálna verzia	0.5.1 (máj 2017)
Licencia	MIT
Podporované OS	Linux, Windows, macOS
Súborová prípona	.jl
Webová stránka	julialang.org

Tabuľka 1.1: Základné informácie o Julii

Ak všetko zhrnieme, toto sú najvýznamnejšie charakteristiky a vlastnosti Julie:[2]

- Julia je zadarmo a voľne dostupná pre všetkých užívateľov.
- Má veľmi dobrý výkon blížiaci sa k staticky kompilovaným jazykom, napríklad C.

- Obsahuje vstavaného správcu balíkov.
- Funkcie jazyka C volá priamo bez využitia špeciálnych nástrojov alebo API.
- Dokáže pracovať s funkciami jazyka Python pomocou balíka *PyCall*.
- Podporuje makrá (podobné ako v jazyku Lisp) a metaprogramovanie.
- Julia je navrhnutá pre paralelizmus a distribuované výpočty.
- Podporuje viacnásobné odosielanie.
- Užívatelia si môžu vytvárať vlastné dátové typy, ktoré sú rovnako rýchle a kompaktné ako preddefinované.
- K dispozícii je efektívna konverzia nie len pre číselné dátové typy.
- Výkonné shell prostredie pre správu procesov.
- Julia obsahuje podporu pre Unicode, ktorá zahŕňa UTF-8 a iné.

Ako bolo spomenuté vyššie, Julia má inovatívny dizajn a kvalitný JIT (just-in-time) kompilátor, preto sa jej výkon často približuje výkonu jazyka C. Výkon Julie sa skúmal veľmi podrobne, bolo vytvorených niekoľko testov, ktoré zisťovali, ako Julia obstojí v porovnaní so známymi jazykmi, ktoré sa často používajú na numerické a vedeckotechnické výpočty. Testovanie fungovalo na jednoduchom princípe. V Julii a vybraných jazykoch bol spustený rovnaký algoritmus, pričom vždy sa zmeral celkový čas, ktorý bol potrebný na vyhodnotenie. Výsledky pre niektoré algoritmy sú uvedené v tabuľke 1.2. Všetky časy sú uvedené relatívne k času v C (C = 1.0). Platí, že čím menšie číslo, tým lepší výkon daný jazyk dosiahol.

	Julia	Fortran	Python	Java	Matlab	R
Fibbonaciho postupnosť	2.11	0.70	77.76	1.21	26.89	533.52
Quicksort triedenie	1.15	1.31	32.89	2.60	4.92	264.54

Tabuľka 1.2: Výsledky testovania výkonu Julie

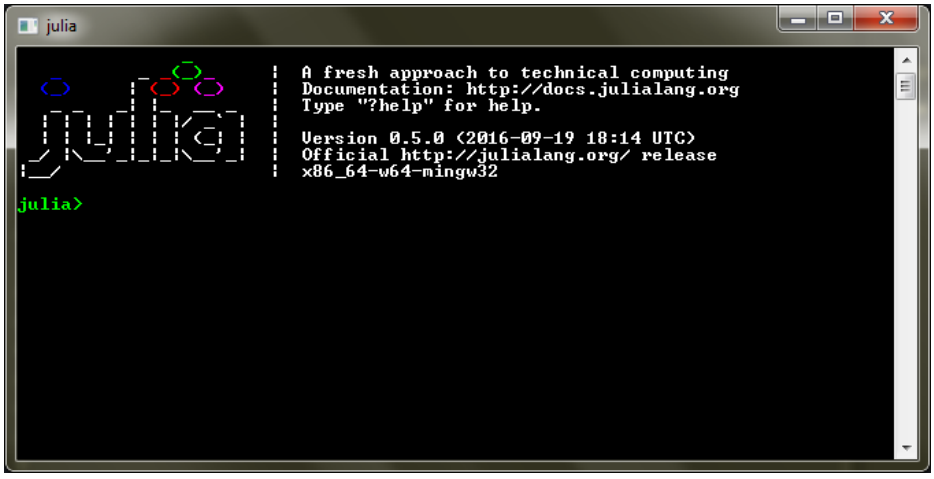
Testovanie výkonnosti skúmalo také vlastnosti jazyka ako je napr. volanie funkcií, úprava reťazcov, triedenie, vyhodnocovanie cyklov, generácia náhodných čísiel, operácie s poliami a iné. Je treba poznamenať, že testovacie algoritmy neboli napísané tak, aby sa dosiahol najlepší možný výkon, ale aby sa otestoval výkon pri špecifických situáciách v každom jazyku.

1.1 Inštalácia

Existuje niekoľko rôznych spôsobov ako spustiť Juliu:[3]

1. Prvým spôsobom je použiť príkazový riadok Julie v termináli.
2. Alternatívou je editor Atom a v ňom integrované vývojové prostredie Juno.
3. Treťou možnosťou je programovať priamo v prehliadači na webovej stránke JuliaBox.com. Ak sa užívateľ rozhodne pre tento spôsob, žiadna inštalácia nie je vyžadovaná, jedinou požiadavkou je prihlásenie sa.

V ďalšej časti sa podrobnejšie budeme venovať prvým dvom bodom, uvedieme jednoduchý návod, v ktorom ukážeme, ako stiahnuť a nainštalovať Juliu a tiež ako v nej začať pracovať.

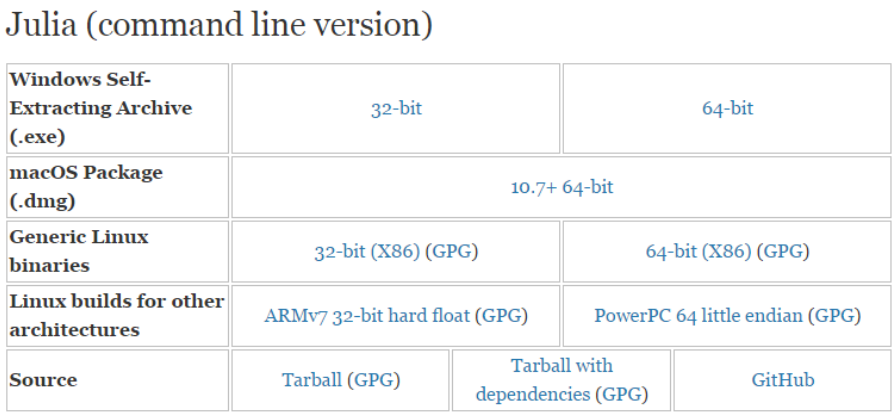


Obr. 1.2: Terminálová verzia Julie

Aby sme mohli začať používať Juliu, je potrebné vykonať tieto kroky:[4]

1. **Stiahnutie Julie:**

   Prvým krokom je stiahnutie Julie (verzia pre príkazový riadok) z lokality http://julialang.org/downloads/. Užívatelia OS Windows majú na výber 2 možnosti – verziu pre 32 a 64-bitový operačný systém (obrázok 1.3).



Obr. 1.3: Verzie Julie pre jednotlivé operačné systémy

2. **Inštalácia Julie:**

   Juliu nainštalujeme tak, že rozbalíme stiahnutý archív, spustíme inštalačný (.exe) súbor a prejdeme cez všetky kroky inštalácie.

3. **Stiahnutie a inštalácia editoru Atom:**

   Editor Atom je možné stiahnuť z webovej stránky https://atom.io/. Inštalácia prebieha rovnako ako v bode 2. Ak bol editor Atom nainštalovaný už predtým, je potrebné sa uistiť, že jeho verzia je 1.7 a vyššia.

4. **Inštalácia Juno IDE:**

   V poslednom kroku otvoríme Atom, prejdeme do nastavení *Settings* a klikneme na záložku *Install*. Po poľa na vyhľadávanie balíkov napíšeme `uber-juno` a potvrdíme klávesou enter. Po nájdení JUNO klikneme

na tlačidlo *Install*. Atom týmto nainštaluje všetky potrebné súčasti a pripraví JUNO a Juliu na okamžité používanie.

Vyššie uvedený návod sa vzťahuje na aktuálnu verziu. Všetky minulé verzie sa dajú stiahnuť na stránke http://julialang.org/downloads/oldreleases.html

1.2 Dokumentácia

Kompletná dokumentácia k jazyku je dostupná na stránke http://docs.julialang.org/en/stable/. V jednotlivých kapitolách sú podrobne rozpísané také témy ako: vytváranie premenných, dátové typy premenných, matematické a logické operátory, práca s reťazcami, vyhodnocovanie podmienok a cyklov, vytváranie funkcií a metód, práca so súbormi a mnohé ďalšie. [5]

Zoznam všetkých publikácii týkajúcich sa Julie a vybrané videonávody sú k dispozícií na http://julialang.org/learning/.

1.3 Správa balíkov

Julia má vstavaného správcu balíkov, pomocou ktorého ju vieme prispôsobiť na špecifické úkony. Pridávaním rôznych rozšírení a užitočných funkcionalít je možné upraviť si Juliu podľa svojich predstáv a ešte viac prehĺbiť niektoré jej schopnosti. [6]

Zoznam registrovaných balíkov a rozšírení, ktoré podporuje aktuálna verzia Julie sa nachádza na webovej stránke http://docs.julialang.org/en/release-0.4/manual/packages/. Všetky funkcie týkajúce sa správy balíkov sú uložené v module Pkg. Tento modul je súčasťou základnej inštalácie Julie.

1.3.1 Pridávanie, odstraňovanie a aktualizovanie balíkov

Funkcia `Pkg.status()` vypíše zoznam, ktorý uvádza názvy a aktuálne verzie všetkých nainštalovaných balíkov. Na začiatku nie sú nainštalované žiadne balíky a výpis vyzerá nasledovne:

```
julia> Pkg.status()
INFO: Initializing package repository /Users/user/.julia/v0.5
INFO: Cloning METADATA from git://github.com/JuliaLang/
      METADATA.jl
No packages installed.
```

Adresár obsahujúci balíky je automatický inicializovaný prvým použitím ľubovoľnej funkcie modulu `Pkg`, vrátane `Pkg.status()`. Keď adresár nie je prázdny, pretože užívateľ nainštaloval niektoré externé balíky, ukáže sa nasledovný výpis:

```
julia> Pkg.status()
Required packages:
 - Distributions                 0.2.8
Additional packages:
 - NumericExtensions             0.2.17
 - Stats                         0.2.6
```

Balíky sú rozdelené do 2 kategórií (*required* a *additional*). Je treba poznamenať, že v tomto zozname nájdeme iba registrované balíky, pretože len tie sú spravované modulom `Pkg`.

Podobná funkcia ako `Pkg.status()` je funkcia `Pkg.installed()`, ktorá zapíše zoznam do premennej:

```
julia> Pkg.installed()
Dict{String,VersionNumber} with 3 entries:
"Distributions"     => v"0.2.8"
"Stats"             => v"0.2.6"
"NumericExtensions" => v"0.2.17"
```

Inštalácia a odstraňovanie balíkov je založené na nasledovnom princípe. Keď užívateľ povie, že chce nainštalovať nový balík, namiesto priamej inštalácie sa v skutočnosti vytvorí požiadavka. Správca balíkov túto požiadavku zapíše do zoznamu požiadaviek a následne ju analyzuje. Vyhodnotí, akú verziu balíka je výhodné nainštalovať a aké iné súčasti sú potrebné, aby požiadavka bola splnená optimálne s minimálnym počtom operácii.

Tento systém je najlepšie vysvetliť na príklade aktualizácie. Uvažujme, že správca balíkov pri vyhodnocovaní požiadavky zistil, že doplnkový balík

bol nainštalovaný iba preto, lebo primárny ho vyžadoval. Ďalej zistil, že nová verzia primárneho balíka už žiadny doplnok nepotrebuje, preto sa pri aktualizácii v skutočnosti vykonalo vymazanie doplnkového balíka a nie jeho opätovné nainštalovanie.

Požiadavky na balíky sú uchovávané v súbore `/.julia/v0.5/REQUIRE`. Existujú tri rôzne spôsoby, ako balíky spravovať:

1. Súbor REQUIRE môžeme editovať manuálne, dopisovať alebo vymazávať z neho požiadavky a potom zavolať funkciu `Pkg.resolve()`, ktorá nainštaluje, aktualizuje alebo vymaže balíky tak, aby sa vyhovelo novým požiadavkám.

2. Druhým spôsobom je použiť funkciu `Pkg.edit()`, ktorá otvorí REQUIRE súbor v niektorom editore a po ukončení jeho úprav automaticky zavolá funkciu `Pkg.resolve()`, ak je to potrebné.

3. Ak chceme v jednom kroku inštalovať alebo vymazať iba jeden balík, najjednoduchšou možnosťou je použiť príkazy `Pkg.add()` a `Pkg.rm()`. Tieto transformujú názov balíka na požiadavku, zapíšu ju do REQUIRE a vykonajú `Pkg.resolve()`, všetko automaticky.

   Príklad použitia funkcie `Pkg.add()` a `Pkg.rm()` je nižšie:

```
julia> Pkg.add("Distributions")
INFO: Cloning cache of Distributions from
      git://github.com/JuliaStats/Distributions.jl.git
INFO: Cloning cache of NumericExtensions from
      git://github.com/lindahua/NumericExtensions.jl.git
INFO: Installing Distributions v0.2.7
INFO: Installing NumericExtensions v0.2.17
INFO: REQUIRE updated.
```

Ako bolo povedané v bode 3, funkcia `Pkg.add()` vezme názov balíka a pridá ho do súboru `/.julia/v0.5/REQUIRE`. Po aktualizácii požiadaviek zavolá funkciu `Pkg.resolve()`, ktorá príde k záveru, že balík musí byť nainštalovaný. Podobne funguje aj `Pkg.rm()`.

```
julia> Pkg.rm("Distributions")
INFO: Removing Distributions v0.2.7
INFO: Removing Stats v0.2.6
INFO: Removing NumericExtensions v0.2.17
INFO: REQUIRE updated.
```

`Pkg.add()` a `Pkg.rm()` je výhodné používať, ak potrebujeme nainštalovať alebo odstrániť jeden, prípadne malé množstvo balíkov. Pri väčšom množstve je odporúčané použiť spôsob vysvetlený v bode 1 a 2, pretože sa tým výrazne zrýchli celý proces.

Pre získanie najnovších verzií balíkov stačí napísať príkaz `Pkg.update()`:

```
julia> Pkg.update()
INFO: Updating METADATA...
INFO: Computing changes...
INFO: Upgrading Distributions: v0.2.8 => v0.2.10
INFO: Upgrading Stats: v0.2.7 => v0.2.8
```

Neregistrované balíky

Všetky vyššie spomenuté funkcie pracujú iba s oficiálnymi balíkmi, ktoré sú zaregistrované v repozitári *METADATA.jl*. Správca balíkov však vie nainštalovať aj neregistrované balíky. Všetky Julia balíky sú v skutočnosti git repozitáre obsahujúce súbory s Julia kódom, preto je možné ich naklonovať. Na inštaláciu takýchto balíkov sa používa príkaz `Pkg.clone(url)`, kde `url` je git adresa, z ktorej bude balík naklonovaný.

```
julia> Pkg.clone("git://example.com/path/to/Package.jl.git")
INFO: Cloning Package from
      git://example.com/path/to/Package.jl.git
Cloning into Package...
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 22 (delta 8), reused 22 (delta 8)
Receiving objects: 100% (22/22), 2.64 KiB, done.
Resolving deltas: 100% (8/8), done.
```

Podľa konvencií všetky Julia repozitáre končia príponou .jl, čo zabraňuje kolidovaniu s repozitármi iných jazykov a zároveň je jednoduché ich nájsť pomocou vyhľadávačov.

Ak neregistrovaný balík vo svojej štruktúre obsahuje REQUIRE súbor, tak pomocou tohto súbora sa zistí, na ktorých registrovaných balíkoch je závislý neregistrovaný, aby mohli byť automaticky nainštalované spolu s ním.

Offline inštalácia

Pre počítače a zariadenia bez internetového pripojenia existuje možnosť nainštalovať balík offline tak, že sa skopíruje adresa balíka pomocou funkcie `Pkg.dir()`.

1.4 Grafické nástroje Julie

Vykresľovanie grafov je v Julii realizované pomocou externých balíkov *PyPlot.jl*, *Gadfly.jl*, *Plots.jl* a iných. [7]

PyPlot využíva balík *PyCall*, vďaka ktorému môžeme v Julii volať rôzne funkcie jazyka Python. Aby bolo možné pracovať s balíkom *PyPlot*, je potrebné mať správne nainštalovaný Python. Inštalácia a príklad použitia sa nachádza nižšie:

```
Pkg.add("PyPlot")
using PyPlot
x = linspace(0,2*pi,1000); y = sin(3*x + 4*cos(2*x))
plot(x, y, color="red", linewidth=2.0, linestyle="--")
```

Gadfly je balík, ktorý implementuje Wickham-Wilkinsonovu metódu vizualizácie dát v Julii. Tento balík je veľmi podobný balíku *ggplot2*, čo je vizualizačný nástroj v štatistickom programovacom jazyku R. Ako je zrejmé z príkladu, inštaluje sa rovnako ako *PyPlot*, ale syntaxou sa výrazne líši.

```
Pkg.add("Gadfly")
using Gadfly
draw(SVG("output.svg", 6inch, 3inch), plot([sin, cos], 0, 25))
```

Plots je grafický balík, ktorý združuje vyššie uvedené balíky (plus niektoré ďalšie) do jedného spoločného rozhrania a dovoľuje medzi nimi prepínať podľa potreby. Inštalácia tohto balíka a príklad jeho použitia je nižšie:

```
Pkg.add("Plots")
using Plots
plotly() # Choose the Plotly.jl backend for web interactivity
plot(rand(5,5),linewidth=2,title="My Plot")
Pkg.add("PyPlot") # Install a different backend
pyplot() # Switch to using the PyPlot.jl backend
plot(rand(5,5),linewidth=2,title="My Plot")
# The same plotting command works
```

Obr. 1.4: Grafické možnosti Julie

Kapitola 2

Optimalizácia v Julii

Jadrom prediktívneho riadenia, ktorému sa venuje táto práca je riešenie optimalizačného problému. Preto sa v tejto kapitole pozrieme na možnosti, ktoré nám Julia ponúka v oblasti matematickej optimalizácie.

Optimalizácia je oblasť matematiky, ktorá sa zaoberá hľadaním najlepšieho možného riešenia problému vzhľadom na zadané kritéria. Pri riešení optimalizačného problému preto minimalizujeme alebo maximalizujeme nejakú funkciu, systematicky vyberáme také hodnoty premenných, aby funkcia nadobudla svoju minimálnu, prípadne maximálnu hodnotu a zároveň neboli porušené dopredu určené kritéria. [8]

Optimalizačný problém vo všeobecnom tvare zapisujeme nasledovne:

$$\min_{x} \quad f(x) \tag{2.1a}$$

$$\text{s.t.} \quad g_i(x) \leq 0, \quad i = 1, \ldots, m \tag{2.1b}$$

$$h_j(x) = 0, \quad j = 1, \ldots, p \tag{2.1c}$$

Prvá časť problému, $f(x) : \mathbb{R}^n \to \mathbb{R}$, je účelová funkcia, ktorú minimalizujeme. Účelová funkcia priradí každej optimalizovanej premennej x jej hodnotu. Druhou časťou problému sú ohraničenia. Poznáme ohraničenia v tvare nerovností $g_i(x) \leq 0, i = 1, \ldots, m$ a ohraničenia v tvare rovností $h_j(x) = 0, j = 1, \ldots, p$. Ohraničenia problému nám určujú oblasť všetkých dovolených hodnôt optimalizovaných premenných.

Optimalizačné problémy môžeme rozdeliť do rôznych kategórií na základe typu účelovej funkcie, ohraničení alebo optimalizovaných premenných. Medzi najznámejšie oblasti optimalizácie patria:

- Konvexné programovanie

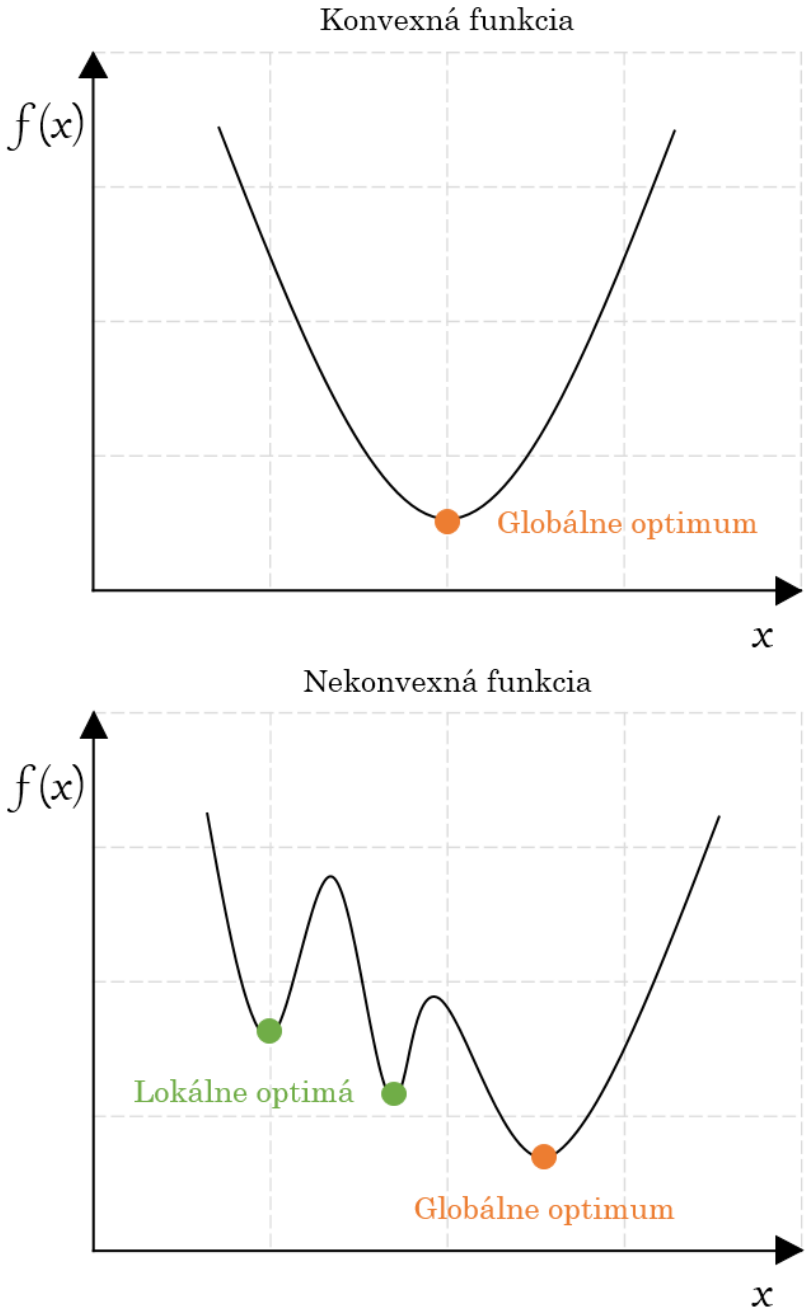
- Lineárne programovanie
- Kvadratické programovanie
- Nelineárne programovanie
- Celočíselné programovanie

Konvexné programovanie študuje optimalizačné problémy, ktoré majú konvexnú účelovú funkciu a konvexné ohraničenia. Rozdiel medzi konvexnou a nekonvexnou funkciou ukazuje obrázok 2.1. Funkciu nazývame konvexnou vtedy, ak sa spojnica medzi ľubovoľnými dvoma bodmi ležiacimi na grafe funkcie nachádza nad grafom, prípadne na ňom. Príklad konvexnej a nekonvexnej množiny reprezentujúcej ohraničenia je na obrázku 2.2. Ak si zvolíme dva rozdielne body A a B z množiny $\mathcal{S}$, tak množina je konvexná vtedy, keď spojnica týchto dvoch bodov celá leží vo vnútri množiny. [9]

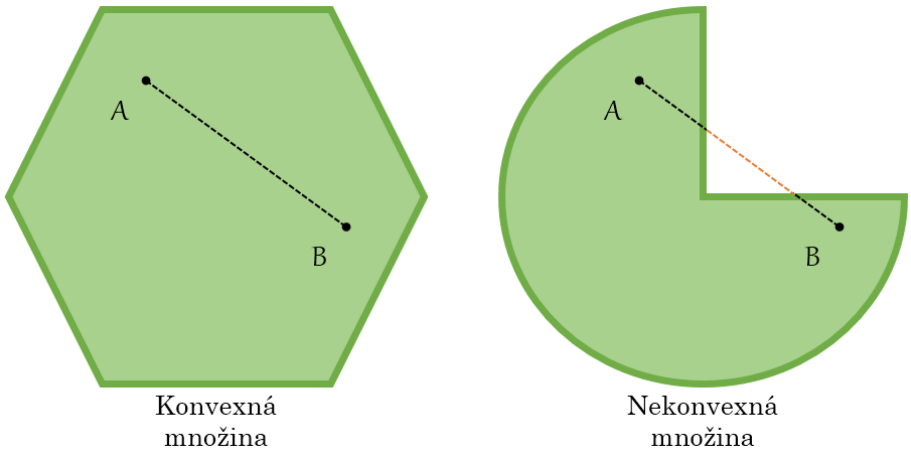
V optimalizácii hrajú dôležitú úlohu najmä striktne konvexné funkcie, pretože majú iba jeden extrém. Ak je optimalizačný problém s konvexnou účelovou funkciou a ohraničeniami riešiteľný, máme istotu, že nájdeme jeho globálne optimum. Preto sa konvexné optimalizačné problémy zaraďujú medzi jednoduché.

Opakom konvexných problémov sú problémy nekonvexné. Tieto problémy majú nekonvexnú účelovú funkciu, nekonvexné ohraničenia, alebo kombináciu oboch. Nájdenie riešenia v takomto prípade je zložité. Zvyčajne treba vynaložiť viac času ako pri konvexnej optimalizácii, navyše nemáme záruku, že sme našli globálne optimum a nie lokálne.

Lineárne programovanie (skrátene LP) je podoblasťou konvexného programovania. V tomto prípade máme lineárnu účelovú funkciu a ohraničenia v tvare lineárnych rovností a nerovností. Kvadratické programovanie (skrátene QP) rieši problémy, v ktorých vystupuje kvadratická účelová funkcia a lineárne ohraničenia v tvare rovností a nerovností. Kvadratický problém tiež môže byť konvexný, ak účelová funkcia spĺňa špecifické požiadavky. Nelineárne programovanie sa zaoberá prípadmi, kedy účelová funkcia aj ohraničenia obsahujú nelineárne časti. V celočíselnom programovaní sú niektoré optimalizované premenné ohraničené takým spôsobom, že môžu nadobúdať iba celočíselné hodnoty. Takéto problémy sú nekonvexné a vo všeobecnosti náročné na vyriešenie.

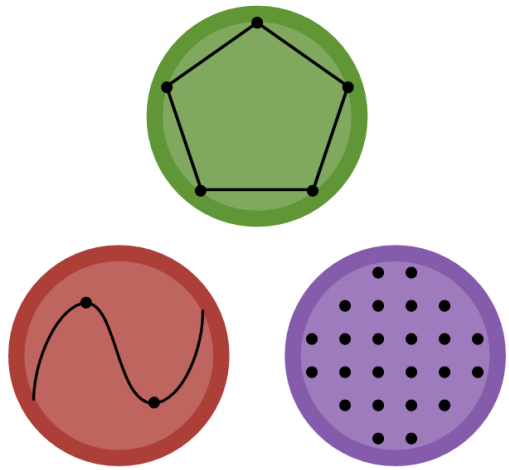


Obr. 2.1: Príklad konvexnej a nekonvexnej funkcie



Obr. 2.2: Príklad konvexnej a nekonvexnej množiny

Ako bolo uvedené v predchádzajúcej kapitole, Julia je rýchly a výkonný dynamický programovací jazyk, ktorý veľmi dobre pracuje s externými knižnicami a aplikuje niektoré pokročilé techniky ako napr. paralelné vykonávanie príkazov, preto je vhodnou voľbou na implementáciu optimalizačných nástrojov.



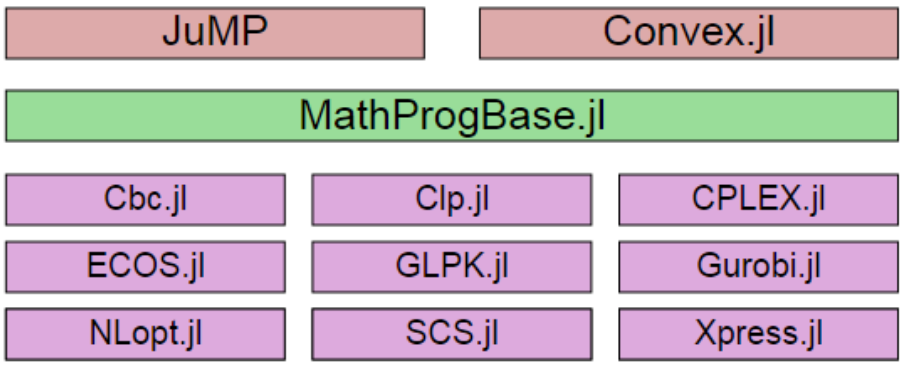
Obr. 2.3: Logo skupiny JuliaOpt

Za optimalizáciu v Julii je zodpovedná GitHub skupina JuliaOpt, ktorá združuje všetky balíky súvisiace s optimalizáciou. Jej cieľom je uľahčiť spo-

luprácu medzi jednotlivými vývojármi a previazať pôvodne nezávislé balíky, aby vznikol ucelený súbor nástrojov pre matematickú optimalizáciu. Táto skupina bola predstavená na konferencii JuliaCon v roku 2015 a jej domovská stránka je `github.com/JuliaOpt`. [10]

Na obrázku 2.4 môžeme vidieť najvýznamnejšie súčasti JuliaOpt. Hore sú ružovou farbou vyznačené modelovacie jazyky. Pomocou nich vie užívateľ jednoducho a intuitívne zadefinovať optimalizačný problém v Julii.

- *JuMP* je algebraický modelovací jazyk určený na lineárne, kvadratické a nelineárne programovanie. *JuMP* generuje modely rovnako rýchlo ako komerčné modelovacie nástroje a podporuje niektoré pokročilé nastavenia solverov.
- *Convex.jl* je rovnako ako *JuMP* modelovací jazyk, ale dá sa použiť iba na problémy konvexného programovania.



Obr. 2.4: Optimalizačné balíky v Julii

Ak máme optimalizačný problém v štandardnom tvare, je výhodné použiť balík *MathProgBase* znázornený zelenou farbou. Pomocou tohto balíka vyriešime optimalizačný problém zavolaním jednej funkcie, nie je potrebné žiadne modelovanie. Fialovou farbou sú označené externé solvery, ktoré možno do Julie importovať.

Tabuľky 2.1 a 2.2 udávajú, ktoré druhy optimalizačných problémov sú podporované jednotlivými modelovacími jazykmi a solvermi.

	JuMP	*Convex*
Lineárne programovanie	✓	✓
Kvadratické programovanie	✓	✓
SOCP/SDP	✓	✓
NLCP	✓	✗
Nekonvexné programovanie	✓	✗
Celočíselné programovanie	✓	✓
Licencia	voľná	voľná

Tabuľka 2.1: Problémy podporované balíkmi *JuMP* a *Convex*

Skratka LP označuje lineárne programovanie, QP kvadratické programovanie. SOCP je taký druh problému, kedy máme lineárnu účelovú funkciu a kvadratické ohraničenia, SDP je semidefinitné programovanie, NLCP sú nelineárne konvexné problémy, IP je celočíselné programovanie a NCP nekonvexné programovanie.

Voľná licencia znamená, že balík sa dá stiahnuť a používať zadarmo. Skratka kom. označuje komerčné solvery. Ak sa pri tejto skratke nachádza číslo 1, solver je komerčný, ale ponúka voľné akademické licencie.

Ľubovoľný z týchto balíkov si užívateľ môže stiahnuť cez správcu balíkov spôsobom opísaným v kapitole 1.

```
julia> Pkg.add("JuMP")
julia> Pkg.add("Clp")
```

Voľne dostupné solvery budú stiahnuté a nainštalované automaticky. Ak chceme použiť externé komerčné solvery, je potrebné ich manuálne stiahnuť ešte predtým ako ich začneme inštalovať v Julii.

Pre pochopenie prediktívneho riadenia je potrebné si osvojiť princípy lineárneho a kvadratického programovania, preto tieto témy bližšie rozoberieme v nasledujúcej časti.

Solver	LP/QP	SOCP SDP	NLCP	NCP	IP	Licencia
CDD	✓	✗	✗	✗	✗	voľná
Clp	✓	✗	✗	✗	✗	voľná
Cbc	✓	✗	✗	✗	✓	voľná
GLPK	✓	✗	✗	✗	✓	voľná
CSDP	✓	✓	✗	✗	✗	voľná
ECOS	✓	✓	✗	✗	✗	voľná
SCS	✓	✓	✗	✗	✗	voľná
CPLEX	✓	✓	✗	✗	✓	kom.[1]
Gurobi	✓	✓	✗	✗	✓	kom.[1]
FICO Xpress	✓	✓	✗	✗	✓	kom.[1]
Mosek	✓	✓	✓	✗	✓	kom.[1]
Parajito	✓	✓	✓	✗	✓	voľná
NLopt	✗	✗	✓	✓	✗	voľná
Ipopt	✓	✗	✓	✓	✗	voľná
Bonmin	✓	✗	✓	✓	✓	voľná
Couenne	✓	✗	✓	✓	✓	voľná
Artelys Knitro	✓	✗	✓	✓	✓	kom.
SCIP	✓	✓	✓	✓	✓	kom.[1]

Tabuľka 2.2: Problémy podporované externými solvermi

2.1 Lineárne programovanie

Lineárne programovanie alebo lineárna optimalizácia je metóda, akou vieme dosiahnuť najlepší možný výsledok (napr. maximálny zisk alebo minimálne náklady) použitím matematického modelu, v ktorom vystupujú iba lineárne vzťahy.[11]

Presnejšie, lineárne programovanie je oblasťou optimalizácie, kde jednáme výlučne s lineárnou účelovou funkciou a lineárnymi ohraničeniami

v tvare rovností a/alebo nerovností. Zlučiteľnou oblasťou, v ktorej bude ležať riešenie problému je polytop (alebo polyhedron), čo je konvexná množina, ktorá vznikne spojením konečného množstva polpriestorov. Každý polpriestor je určený jednou lineárnou nerovnosťou. Účelová funkcia je potom reálna funkcia definovaná na tomto polytope. Algoritmy určené na lineárne programovanie hľadajú taký bod polytopu, kde funkcia má najmenšiu (najväčšiu) hodnotu.

Štandardný tvar lineárneho programu je nižšie:

$$\min_{x} \quad c^{\intercal}x \tag{2.2a}$$

$$\text{s.t.} \quad Ax \leq b \tag{2.2b}$$

$$A_{\text{eq}}x = b_{\text{eq}} \tag{2.2c}$$

kde $x \in \mathbb{R}^n$ reprezentuje vektor optimalizovaných premenných, $c \in \mathbb{R}^n$ je vektor obsahujúci koeficienty účelovej funkcie, $A \in \mathbb{R}^{m \times n}$ je matica ľavých strán ohraničení v tvare nerovností, $b \in \mathbb{R}^m$ je vektor pravých strán ohraničení v tvare nerovností, $A_{\text{eq}} \in \mathbb{R}^{p \times n}$ je matica ľavých strán ohraničení v tvare rovností a $b_{\text{eq}} \in \mathbb{R}^p$ je vektor pravých strán ohraničení v tvare rovností. m je počet ohraničení v tvare nerovností a p v tvare rovností.

Výhoda takéhoto tvaru spočíva v tom, že ho podporuje takmer každý solver určený na lineárne programovanie.

Lineárne programovanie je veľmi známym konceptom, ktorý sa využíva v rôznych oblastiach. Je často používaný vo finančníctve a ekonomike a má svoje uplatnenie aj v priemysle. Oblasti priemyslu, ktoré využívajú lineárne modely sú napr. preprava, energetika, telekomunikácie alebo výroba.

V Julii existuje niekoľko spôsobov, ako vyriešiť lineárny optimalizačný problém. Môžeme ho najskôr namodelovať v jeho prirodzenom tvare pomocou *JuMP* alebo *Convex* a potom ho vyriešiť, alebo ho môžeme skonvertovať do štandardného tvaru a použiť funkciu `linprog` balíka *MathProgBase*.

Optimalizačný balík *MathProgBase* obsahuje jednoduché, ale výkonné funkcie pre lineárne, kvadratické a celočíselné programovanie. Aby sa dal tento balík použiť, je nevyhnutné nainštalovať externý solver, ktorý bude podporovať vyššie uvedené druhy programovania.

2.1.1 linprog

Funkcia `linprog` uvažuje lineárny program v tvare:[12]

$$\begin{aligned} \min \quad & c^\intercal x && \text{(2.3a)} \\ \text{s.t.} \quad & a_i^\intercal x \ \text{sense}_i \ b_i \quad \forall i && \text{(2.3b)} \\ & l \leq x \leq u && \text{(2.3c)} \end{aligned}$$

pričom má nasledujúcu syntax:

```
linprog(c, A, sense, b, l, u, solver)
```

kde

- `c` je vektor reprezentujúci účelovú funkciu problému vždy v tvare minimalizácie
- `A` je matica ohraničení, jej riadky sú vektory $a_i^\intercal$
- `b` je vektor ohraničení
- `sense` je vektor obsahujúci znaky <,> a =
- `l` je vektor dolných hraníc optimalizovaných premenných
- `u` je vektor horných hraníc optimalizovaných premenných
- `solver` je premenná špecifikujúca solver

Argumenty `b`, `sense`, `l` a `u` akceptujú skalár. V prípade, že sú tieto premenné skalárne, aj keď problém na ich mieste vyžaduje vektory, hodnoty sú replikované, aby všetky rozmery vzájomne korešpondovali. Ak nemáme dolné a horné ohraničenia na optimalizované premenné, vložíme hodnoty `-Inf` a `Inf`.

`linprog` má aj svoju skrátenú verziu:

```
linprog(c, A, sense, b, solver) =
    linprog(c, A, sense, b, 0, Inf, solver)
```

Funkcia `linprog` vracia inštanciu nasledovného typu:

```
type LinprogSolution
    status
    objval
    sol
    attrs
end
```

kde `status` je premenná, ktorá nás informuje o tom, či problém bol úspešne vyriešený alebo nie a nadobúda tieto hodnoty:

- : *Optimal*
- : $Infeasible$
- : *Unbounded*
- : $UserLimit$
- : *Error*

Ak bolo nájdené optimálne riešenie, naplnia sa ostatné polia:

- `objval`: optimálna hodnota účelovej funkcie
- `sol`: optimálne riešenie problému
- `attrs`: ďalšie relevantné atribúty ako sú napr. optimálne hodnoty Lagrangeových násobičov

Príklad:

```
using MathProgBase, Clp

sol = linprog([-1,0],[2 1],'<',1.5, ClpSolver())
if sol.status == :Optimal
    println("Optimal objective value is $(sol.objval)")
    println("Optimal solution vector is:
            [$(sol.sol[1]), $(sol.sol[2])]")
else
    println("Error: solution status $(sol.status)")
end
```

2.2 Kvadratické programovanie

V oblasti optimalizácie a matematiky je kvadratické programovanie veľmi známy a študovaný koncept. Kvadratické programovanie je špeciálny typ optimalizačného problému, kedy minimalizujeme kvadratickú účelovú funkciu vzhľadom na lineárne obmedzenia optimalizovaných premenných. Obmedzenia môžu byť v tvare rovností a nerovností. [13]

Vo všeobecnosti môžeme kvadratický optimalizačný problém zadefinovať nasledovne:

$$\min_{x} \quad \frac{1}{2}x^T Px + q^T x + r \tag{2.4a}$$

$$\text{s.t.} \quad Ax \leq b \tag{2.4b}$$

$$A_{\text{eq}}x = b_{\text{eq}} \tag{2.4c}$$

$x \in \mathbb{R}^n$ je vektor optimalizovaných premenných, $P \in \mathbb{R}^{n \times n}$ je štvorcová matica obsahujúca koeficienty kvadratických členov účelovej funkcie, $q \in \mathbb{R}^n$ je vektor obsahujúci koeficienty lineárnych členov, $r \in \mathbb{R}$ je skalár. Matica $A \in \mathbb{R}^{m \times n}$ reprezentuje ľavú stranu ohraničení v tvare nerovností, $b \in \mathbb{R}^m$ je vektor pravých strán ohraničení v tvare nerovností, ,A_{eq} reprezentuje ľavú stranu ohraničení v tvare rovností, b_{eq} pravú stranu ohraničení v tvare rovností, kde m je počet ohraničení v tvare nerovností a p počet ohraničení v tvare rovností.

V praxi sú problémy prediktívneho riadenia často zadefinované vo forme kvadratického programovania. Vyžaduje sa, aby takéto problémy mali jedinečné riešenie – jedno globálne optimum. Dôvodom je, že optimalizujeme akčný zásah a preto nemôžeme dostať viac ako jedno riešenie. Aby problém spĺňal túto požiadavku, matica P musí byť pozitívne definitná. Ak platí $P \succ 0$, problém spadá do oblasti konvexnej optimalizácie a my vždy nájdeme jeho globálne optimum. [14]

Ak kvadratický problém obsahuje len ohraničenia v tvare rovností, je veľmi jednoduché ho vyriešiť. Ak obsahuje tiež nerovnosti, medzi najčastejšie používané algoritmy patria metóda vnútorného bodu a aktívneho setu. [15]

2.2.1 quadprog

Funkcia `quadprog` vie vyriešiť kvadratický problém v tvare:[16]

$$\min \quad \frac{1}{2}x^\intercal Q x + c^\intercal x \tag{2.5a}$$

$$\text{s.t.} \quad a_i^\intercal x \ \text{sense}_i \ b_i \quad \forall i \tag{2.5b}$$

$$l \leq x \leq u \tag{2.5c}$$

pričom má nasledujúcu syntax:

```
quadprog(c, Q, A, sense, b, l, u, solver)
```

kde

- `c` je vektor reprezentujúci účelovú funkciu problému vždy v tvare minimalizácie
- `Q` je hessián
- `A` je matica ohraničení, jej riadky sú vektory $a_i^\intercal$
- `b` je vektor ohraničení
- `sense` je vektor obsahujúci znaky <,> a =
- `l` je vektor dolných hraníc optimalizovaných premenných
- `u` je vektor horných hraníc optimalizovaných premenných
- `solver` je premenná špecifikujúca solver

`quadprog` sa riadi rovnakými pravidlami ako `linprog`. Argumenty `b`, `sense`, `l` a `u` môžu byť skalár. V prípade, že rozmery nesedia, skaláre sú transformované na vektory. Ak nemáme dolné a horné ohraničenia na optimalizované premenné, znovu používame hodnoty `-Inf` a `Inf`.

Funkcia `quadprog` vracia inštanciu nasledovného typu:

```
type QuadprogSolution
    status
    objval
```

```
    sol
    attrs
end
```

pričom polia nadobúdajú rovnaké hodnoty ako pri funkcii `linprog`.

Príklad:

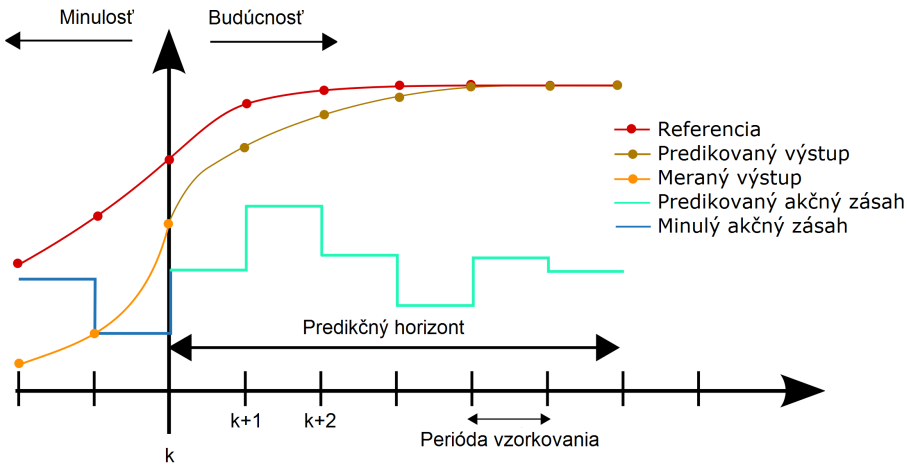
```
using MathProgBase, Ipopt

sol = quadprog([0., 0., 0.],[2. 1. 0.; 1. 2. 1.; 0. 1. 2.],
    [1. 2. 3.; 1. 1. 0.],'>',[4., 1.],-Inf,Inf, IpoptSolver())
if sol.status == :Optimal
    println("Optimal objective value is $(sol.objval)")
    println("Optimal solution vector is: [$(sol.sol[1]),
        $(sol.sol[2]), $(sol.sol[3])]")
else
    println("Error: solution status $(sol.status)")
end
```

Kapitola 3

Prediktívne riadenie

Prediktívne riadenie alebo skrátene MPC (z anglického jazyka Model Predictive Control) je pokročilá metóda procesného riadenia, ktorá sa začala používať v priemysle v 80. rokoch 20. storočia. Využívali ju najmä v chemickom priemysle a energetike. Prediktívne riadenie stavia na znalosti dynamického modelu procesu, najčastejšie je to lineárny empirický model získaný na základe identifikácie reálneho systému. [17] Typickou črtou MPC je, že na základe optimalizácie dokáže predpovedať budúce situácie a podľa týchto predpovedí vhodne jednať. Vďaka tomu sú prediktívne regulátory nadradené PID a LQR regulátorom, ktoré túto vlastnosť nemajú.[18]



Obr. 3.1: Prediktívne riadenie

	MIMO systémy	Ohraničenia	Optimalizácia výkonu
PID	nie	nie	nie
LQR	áno	áno	nie
MPC	áno	áno	áno

Tabuľka 3.1: Porovnanie PID, LQR a MPC regulátorov

Ako bolo uvedené vyššie, prediktívne riadenie je druh optimálneho riadenia. Stratégia takéhoto riadenia spočíva v tom, že predikujeme budúci vývoj stavových, prípadne výstupných veličín na zvolenom horizonte na základe modelu systému a pomocou optimalizácie zisťujeme, aké akčné zásahy treba aplikovať na systém, aby sme dosiahli žiadanú kvalitu riadenia. [14]

Keďže prediktívne riadenie využíva optimalizáciu, čo je vlastne hľadanie najlepšieho možného riešenia problému, vieme ním zabezpečiť také ciele ako sú napr. minimálna spotreba energie alebo maximálny zisk. Narozdiel od PID regulátora si MPC regulátor poradí so systémami s veľkými dopravnými oneskoreniami a systémami vyšších rádov s viacerými vstupmi a výstupmi. Môžeme tiež brať do úvahy fyzikálne ohraničenia systémov alebo iné špecifikácie týkajúce sa riadenia. Toto je možné vďaka tomu, že jadrom prediktívneho riadenia je optimalizačný problém, ktorý vieme naformulovať tak, aby zahŕňal všetky naše požiadavky.

MPC problémy sú často definované vo forme kvadratického programovania. To znamená, že zvyčajne optimalizujeme také problémy, ktoré majú kvadratickú účelovú funkciu a lineárne ohraničenia v tvare rovností a nerovností. Postup pri prediktívnom riadení je nasledovný. V každej perióde vzorkovania riešime optimalizačný problém s aktualizovanými meraniami stavov, čím získame optimálnu sekvenciu akčných zásahov do systému. Následne z tejto sekvencie vyberieme iba prvý akčný zásah a ten aplikujeme na systém. Táto technika sa nazýva metóda posuvného časového horizontu. Predikčný horizont zostáva konštantný počas celého trvania riadenia. Dôvod, prečo aplikujeme len prvý akčný zásah namiesto celej sekvencie je ten, že predikcia vo väčšine prípadov neráta s poruchami, ktoré môžu ovplyvňovať systém.

Aby sme to zhrnuli, pri návrhu prediktívneho regulátora je potrebné prejsť týmito krokmi:

1. Odvodenie matematického modelu procesu
2. Formulácia optimalizačného problému
3. Odmeranie stavových veličín procesu v čase t
4. Riešenie optimalizačného problému, ktorého výsledkom je sekvencia optimálnych akčných zásahov na horizonte N
5. Aplikovanie prvého elementu sekvencie na proces
6. Opakovanie od bodu 3

3.1 Formulácia optimalizačného problému

Nasledujúca časť sa bude zaoberať problematikou formulácie optimalizačného problému. Aby výsledné riadenie systému spĺňalo naše očakávania, treba optimalizačný problém na začiatku správne zostrojiť. V praxi sa stretávame s tým, že každý systém má svoje špecifické vlastnosti, ktorým sa musíme prispôsobiť. Rôzne systémy vyžadujú rôzne prístupy pri riadení, čo znamená, že bude existovať veľké množstvo formulácií. Meniť sa môže druh modelu, ohraničenia a ďalšie iné parametre. V tejto práci sa zameriame na dve základné formulácie – problém regulácie a problém sledovania.

3.1.1 Problém regulácie

Problém regulácie je jednou zo základných formulácií v prediktívnom riadení. Riešením tohto problému získame MPC regulátor, ktorý tlačí systém do jeho počiatku, t.j. do nulového stavu. [14]

Uvažujme, že dynamiku systému opisuje lineárny stavový model v diskrétnom čase. Má nasledovný tvar:

$$x_{k+1} = Ax_k + Bu_k \tag{3.1}$$

Stavové veličiny $x \in \mathbb{R}^{n_x}$ a vstupné veličiny $u \in \mathbb{R}^{n_u}$ systému sú ohraničené:

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U} \tag{3.2}$$

Uvedený model je daný dvoma maticami, systémovou maticou $A \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$ a maticou vstupov $B \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{u}}$, kde n_x označuje počet stavov a n_u počet vstupov.

Ďalej uvažujme kvadratickú účelovú funkciu:

$$\min \sum_{k=0}^{N-1} x_k^\intercal Q_\mathrm{x} x_k + \sum_{k=0}^{N-1} u_k^\intercal Q_\mathrm{u} u_k \tag{3.3}$$

V prvom člene účelovej funkcie penalizujeme vzdialenosť (odchýlku) predikovaných stavov od nuly a v druhom vzdialenosť predikovaných vstupov od nuly na zvolenom predikčnom horizonte N. $Q_\mathrm{x} \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$ a $Q_\mathrm{u} \in \mathbb{R}^{n_\mathrm{u} \times n_\mathrm{u}}$ nazývame váhové matice, pretože určujú, ktorý člen účelovej funkcie má vyššiu a ktorý nižšiu prioritu. Na matice Q_x a Q_u je kladená požiadavka, aby boli pozitívne definitné.

Keď spojíme dohromady model systému, ohraničenia a účelovú funkciu, naformulujeme optimalizačný problém:

$$\begin{aligned}
\min \quad & \sum_{k=0}^{N-1} x_k^\intercal Q_\mathrm{x} x_k + \sum_{k=0}^{N-1} u_k^\intercal Q_\mathrm{u} u_k && \text{(3.4a)}\\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k && \text{(3.4b)}\\
& x_0 = x(t) && \text{(3.4c)}\\
& x_k \in \mathcal{X} && \text{(3.4d)}\\
& u_k \in \mathcal{U} && \text{(3.4e)}\\
& k = 1, \ldots, N-1 && \text{(3.4f)}
\end{aligned}$$

Aby sme mohli vypočítať optimálne akčné zásahy, predikčný model musí byť inicializovaný meraním stavových veličín.

Vo všeobecnosti uvažujeme, že ohraničenia v takomto probléme sú lineárne nerovnosti tvoriace polyhedron:

$$\mathcal{X} = \{x \in \mathbb{R}^{n_\mathrm{x}} | M_\mathrm{x} x \leq m_\mathrm{x}\}, \tag{3.5a}$$

$$\mathcal{U} = \{u \ \in \mathbb{R}^{n_\mathrm{u}} | M_\mathrm{u} u \leq m_\mathrm{u}\} \tag{3.5b}$$

kde $M_\mathrm{x} \in \mathbb{R}^{h_\mathrm{x} \times n_\mathrm{x}}$,$m_\mathrm{x} \in \mathbb{R}^{h_\mathrm{x}}$ $M_\mathrm{u} \in \mathbb{R}^{h_\mathrm{u} \times n_\mathrm{u}}$, $m_\mathrm{u} \in \mathbb{R}^{h_\mathrm{u}}$. h_x a h_u označuje počet polpriestorov, ktoré definujú polyhedron.

V praxi je však bežné, že stavové a vstupné veličiny sú ohraničené zdola

minimálnou hodnotou a zhora maximálnou nasledovne:

$$x_{\min} \leq x_k \leq x_{\max}, \tag{3.6a}$$

$$u_{\min} \leq u_k \leq u_{\max} \tag{3.6b}$$

Keď nahradíme všeobecné ohraničenia za tieto, dostávame problém v tvare:

$$\min \quad \sum_{k=0}^{N-1} x_k^\intercal Q_\mathrm{x} x_k + \sum_{k=0}^{N-1} u_k^\intercal Q_\mathrm{u} u_k \tag{3.7a}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k \tag{3.7b}$$

$$x_0 = x(t) \tag{3.7c}$$

$$x_{\min} \leq x_k \leq x_{\max} \tag{3.7d}$$

$$u_{\min} \leq u_k \leq u_{\max} \tag{3.7e}$$

$$k = 1, \ldots, N-1 \tag{3.7f}$$

3.1.2 Problém sledovania

V priemysle sa často vyžaduje, aby výstupná veličina sledovala nejakú nenulovú referenciu, najlepšie bez trvalej regulačnej odchýlky. Keď chceme navrhnúť prediktívne riadenie pre takýto prípad, musíme v predchádzajúcej formulácii vykonať niekoľko modifikácií.

Znovu budeme uvažovať lineárny stavový model v diskrétnom čase. Problém sledovania však musí okrem stavovej rovnice obsahovať aj výstupnú:

$$x_{k+1} = Ax_k + Bu_k \tag{3.8a}$$

$$y_k = Cx_k + Du_k \tag{3.8b}$$

Tento upravený model definujú 4 matice – matica $A \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$, kde n_x je počet stavov systému, matica $B \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{u}}$, kde n_u je počet vstupov do systému, matica $C \in \mathbb{R}^{n_\mathrm{y} \times n_\mathrm{x}}$, kde n_y je počet výstupov zo systému a matica $D \in \mathbb{R}^{n_\mathrm{y} \times n_\mathrm{u}}$. Premenná $x \in \mathbb{R}^{n_\mathrm{x}}$ reprezentuje stavy, $u \in \mathbb{R}^{n_\mathrm{u}}$ vstupy a $y \in \mathbb{R}^{n_\mathrm{y}}$ výstupy.

Účelovú funkciu zmeníme tiež:

$$\min \quad \sum_{k=0}^{N-1} (y_k - r)^\intercal Q_\mathrm{y} (y_k - r) + \sum_{k=0}^{N-1} \Delta u_k^\intercal Q_\mathrm{u} \Delta u_k \tag{3.9}$$

V nasledovnej formulácii účelovej funkcie penalizujeme regulačnú odchýlku $y_k - r$. Aby výstup presne sledoval referenciu, táto odchýlka musí byť nulová. Ďalej penalizujeme inkrementy akčných zásahov označované ako Δu. Δu je definované ako rozdiel medzi aktuálnou hodnotou akčného zásahu a hodnotou akčného zásahu v minulej perióde vzorkovania, čo sa dá zapísať ako $\Delta u_k = u_k - u_{k-1}$. Ak je výstup na referencii, systém je v ustálenom stave a preto sa akčný zásah nemení. To znamená, že jeho hodnota je rovnaká v každej perióde a teda rozdiel dvoch za sebou nasledujúcich akčných zásahov Δu je nulový. $Q_\mathrm{y} \in \mathbb{R}^{n_\mathrm{y} \times n_\mathrm{y}}$ a $Q_\mathrm{u} \in \mathbb{R}^{n_\mathrm{u} \times n_\mathrm{u}}$ sú váhové matice. Znovu platí požiadavka, že matice Q_y a Q_u musia byť pozitívne definitné.

Keď spojíme model, účelovú funkciu a navyše pridáme všeobecné ohraničenia na optimalizované premenné, vznikne takýto optimalizačný problém.

$$\begin{aligned}
\min \quad & \sum_{k=0}^{N-1} (y_k - r)^\intercal Q_\mathrm{y} (y_k - r) + \sum_{k=0}^{N-1} \Delta u_k^\intercal Q_\mathrm{u} \Delta u_k && \text{(3.10a)} \\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k && \text{(3.10b)} \\
& y_k = Cx_k + Du_k && \text{(3.10c)} \\
& \Delta u_k = u_k - u_{k-1} && \text{(3.10d)} \\
& x_0 = x(t) && \text{(3.10e)} \\
& u_{-1} = u(t-1) && \text{(3.10f)} \\
& u_k \in \mathcal{U} && \text{(3.10g)} \\
& y_k, r_k \in \mathcal{Y} && \text{(3.10h)} \\
& \Delta u_k \in \Delta\mathcal{U} && \text{(3.10i)} \\
& k = 1, \ldots, N-1 && \text{(3.10j)}
\end{aligned}$$

Pri tejto formulácii potrebujeme prístup k akčnému zásahu z predošlej periódy, preto na začiatku musíme inicializovať akčný zásah u_{-1}.

To, že výstup nezanecháva trvalú regulačnú odchýlku spôsobuje člen Δu. Tento člen vystupuje ako diskrétny integrátor.

$$u_k = u_{k-1} + \Delta u_k \leftarrow u_{k+1} = u_k + \Delta u_{k+1} \tag{3.11}$$

Je potrebné si uvedomiť, že formulácia, ktorú sme uviedli nebude odstraňovať trvalú regulačnú odchýlku, ak budeme chcieť riadiť nelineárny MIMO systém. Regulačná odchýlka je odstránená iba vtedy, keď sa model presne zhoduje s reálnym systémom.

Ak všeobecné ohraničenia v predošlom probléme nahradíme za konkrétne, dostaneme konečný tvar problému:

$$\min \quad \sum_{k=0}^{N-1} (y_k - r)^\intercal Q_\mathrm{y} (y_k - r) + \sum_{k=0}^{N-1} \Delta u_k^\intercal Q_\mathrm{u} \Delta u_k \tag{3.12a}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k \tag{3.12b}$$

$$y_k = Cx_k + Du_k \tag{3.12c}$$

$$\Delta u_k = u_k - u_{k-1} \tag{3.12d}$$

$$x_0 = x(t) \tag{3.12e}$$

$$u_{-1} = u(t-1) \tag{3.12f}$$

$$u_{\min} \leq u_k \leq u_{\max} \tag{3.12g}$$

$$\Delta u_{\min} \leq \Delta u_k \leq \Delta u_{\max} \tag{3.12h}$$

$$y_{\min} \leq y_k \leq y_{\max} \tag{3.12i}$$

$$k = 1, \ldots, N-1 \tag{3.12j}$$

3.2 Formulácia kvadratického programu

V predchádzajúcej časti sme naformulovali dva optimalizačné problémy – problém regulácie a problém sledovania. V oboch týchto problémoch vystupuje kvadratická účelová funkcia a lineárne ohraničenia v tvare rovností aj nerovností. To znamená, že aby sme získali optimálne akčné zásahy, potrebujeme použiť solver vhodný na kvadratické programovanie. Všetky dostupné kvadratické solvery však požadujú problém naformulovaný v štandardnom tvare, preto ďalším krokom musí byť reformulácia pôvodných problémov do tohto tvaru.

Existujú dva spôsoby, ako vykonať túto transformáciu. Jedna metóda sa nazýva hustá a druhá riedka formulácia. Hlavným rozdielom medzi týmito dvoma formuláciami je výsledná štruktúra matíc kvadratického programu. Presnejšie, hustá formulácia vedie k nižšiemu počtu optimalizovaných premenných v porovnaní s riedkou, ale na druhej strane dostávame matice s horšou štruktúrou, čo môže viesť k zvýšeniu výpočtovej náročnosti. Náročnosť riešenia problému súvisí najmä s voľbou predikčného horizontu. Hustá formulácia je preferovaná vtedy, keď máme kratší predikčný horizont a riedka naopak vtedy, keď je predikčný horizont dlhší. [19]

Našim účelom lepšie vyhovuje hustá formulácia, preto ju bližšie opíšeme v nasledujúcej časti.

3.2.1 Problém regulácie

V tejto časti je naším cieľom upraviť problém regulácie tak, aby sa dal prepísať do štandardného tvaru kvadratického programovania. [14]

Uvažujme optimalizačný problém v tvare:

$$\min \quad \sum_{k=0}^{N-1} x_k^\intercal Q_\mathrm{x} x_k + \sum_{k=0}^{N-1} u_k^\intercal Q_\mathrm{u} u_k \tag{3.13a}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k \tag{3.13b}$$

$$x_0 = x(t) \tag{3.13c}$$

$$u_{\min} \leq u_k \leq u_{\max} \tag{3.13d}$$

$$x_{\min} \leq x_k \leq x_{\max} \tag{3.13e}$$

$$k = 1, \ldots, N-1 \tag{3.13f}$$

V prvom kroku zadefinujeme nové premenné U a X nasledovne:

$$U = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}, \quad X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{3.14}$$

Ďalej upravíme model systému. Každý stav v nasledujúcej perióde vyjadríme pomocou stavu a vstupu v aktuálnej perióde.

$$x_1 = Ax_0 + Bu_0 \tag{3.15}$$

$$\begin{aligned} x_2 &= Ax_1 + Bu_1 \\ &= A(Ax_0 + Bu_0) + Bu_1 \\ &= A^2x_0 + ABu_0 + Bu_1 \end{aligned} \tag{3.16}$$

$$\begin{aligned} x_3 &= Ax_2 + Bu_2 \\ &= A(A^2x_0 + ABu_0 + Bu_1) + Bu_2 \\ &= A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \end{aligned} \tag{3.17}$$

Keď prvú rovnicu dosadíme do druhej, druhú do tretej atď, zistíme, že, stav v ľubovoľnej perióde vzorkovania vieme zapísať pomocou počiatočnej podmienky a vstupných veličín.

Pôvodný model preto prepíšeme do tvaru:

$$X = \tilde{A}x_0 + \tilde{B}U \tag{3.18}$$

kde

$$\tilde{A} = \begin{bmatrix} I \\ A \\ A^2 \\ A^3 \\ \vdots \\ A^{N-1} \end{bmatrix}, \tilde{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ B & 0 & 0 & 0 & \dots & 0 \\ AB & B & 0 & 0 & \dots & 0 \\ A^2B & AB & B & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ A^{N-2}B & A^{N-3}B & \dots & A^2B & AB & B \end{bmatrix} \tag{3.19}$$

Pomocou nových premenných (3.14) preformulujeme účelovú funkciu:

$$\min \quad X^\intercal \tilde{Q}_\mathrm{x} X + U^\intercal \tilde{Q}_\mathrm{u} U \tag{3.20}$$

kde

$$\tilde{Q_\mathrm{x}} = \begin{bmatrix} Q_\mathrm{x} & 0 & 0 & 0 \\ 0 & Q_\mathrm{x} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_\mathrm{x} \end{bmatrix}, \tilde{Q_\mathrm{u}} = \begin{bmatrix} Q_\mathrm{u} & 0 & 0 & 0 \\ 0 & Q_\mathrm{u} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_\mathrm{u} \end{bmatrix} \tag{3.21}$$

Do účelovej funkcie substituujeme výraz (3.18), čím z problému odstránime ohraničenia v tvare rovností. Toto je veľmi užitočný krok, pretože nie každý kvadratický solver takéto ohraničenia podporuje.

$$\begin{aligned} & X^\intercal \tilde{Q}_\mathrm{x} X + U^\intercal \tilde{Q}_\mathrm{u} U \\ & = (\tilde{A}x_0 + \tilde{B}U)^\intercal \tilde{Q}_\mathrm{x}(\tilde{A}x_0 + \tilde{B}U) + U^\intercal \tilde{Q}_\mathrm{u} U \\ & = x_0^\intercal \tilde{A}^\intercal \tilde{Q}_\mathrm{x} \tilde{A} x_0 + 2x_0^\intercal \tilde{A}^\intercal \tilde{Q}_\mathrm{x} \tilde{B} U + U^\intercal \tilde{B}^\intercal \tilde{Q}_\mathrm{x} \tilde{B} U + U^\intercal \tilde{Q}_\mathrm{u} U \\ & = U^\intercal (\tilde{B}^\intercal \tilde{Q}_\mathrm{x} \tilde{B} + \tilde{Q}_\mathrm{u}) U + 2x_0^\intercal \tilde{A}^\intercal \tilde{Q}_\mathrm{x} \tilde{B} U + x_0^\intercal \tilde{A}^\intercal \tilde{Q}_\mathrm{x} \tilde{A} x_0 \end{aligned} \tag{3.22}$$

Po úprave účelovej funkcie vieme zadefinovať maticu P, vektor q a skalár r.

$$P = 2(\tilde{B}^\intercal \tilde{Q}_\mathrm{x} \tilde{B} + \tilde{Q}_\mathrm{u}), \tag{3.23}$$

$$q = 2x_0^\intercal \tilde{A}^\intercal \tilde{Q}_\mathrm{x} \tilde{B}, \tag{3.24}$$

$$r = x_0^\intercal \tilde{A}^\intercal \tilde{Q}_\mathrm{x} \tilde{A} x_0 \tag{3.25}$$

Pôvodné ohraničenia najskôr rozdelíme:

$$x_{\min} \leq x_k \leq x_{\max} \tag{3.26a}$$
$$u_{\min} \leq u_k \leq u_{\max} \tag{3.26b}$$

$$U \leq U_{\max} \tag{3.27a}$$
$$-U \leq -U_{\min} \tag{3.27b}$$
$$X \leq X_{\max} \tag{3.27c}$$
$$-X \leq -X_{\min} \tag{3.27d}$$

Potom dosadíme rovnicu (3.18) a následne ich upravíme tak, aby členy obsahujúce premenné vystupovali vľavo a konštanty vpravo.

$$U \leq U_{\max} \tag{3.28a}$$
$$-U \leq -U_{\min} \tag{3.28b}$$
$$\tilde{A}x_0 + \tilde{B}U \leq X_{\max} \tag{3.28c}$$
$$-\tilde{A}x_0 + \tilde{B}U \leq -X_{\min} \tag{3.28d}$$

$$U \leq U_{\max} \tag{3.29a}$$
$$-U \leq -U_{\min} \tag{3.29b}$$
$$\tilde{B}U \leq X_{\max} - \tilde{A}x_0 \tag{3.29c}$$
$$-\tilde{B}U \leq -X_{\min} + \tilde{A}x_0 \tag{3.29d}$$

V závere vytvoríme matice A a b.

$$A = \begin{bmatrix} I \\ -I \\ \tilde{B} \\ -\tilde{B} \end{bmatrix}, b = \begin{bmatrix} U_{\max} \\ -U_{\min} \\ X_{\max} - \tilde{A}x_0 \\ -X_{\min} + \tilde{A}x_0 \end{bmatrix} \tag{3.30}$$

3.2.2 Problém sledovania

Pri probléme sledovania budeme postupovať podobne ako pri probléme regulácie.

Najskôr preformulujeme účelovú funkciu:

$$\begin{aligned} \min \quad & \sum_{k=0}^{N-1} (y_k - r)^\intercal Q_\mathrm{y} (y_k - r) + \sum_{k=0}^{N-1} \Delta u_k^\intercal Q_u \Delta u_k \\ & = \min \quad (Y - R)^T \tilde{Q}_\mathrm{y} (Y - R) + \Delta U^T \tilde{Q}_\mathrm{u} \Delta U \end{aligned} \tag{3.31}$$

pričom zadefinujeme:

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix}, R = \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{N-1} \end{bmatrix} \Delta U = \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \Delta u_2 \\ \vdots \\ \Delta u_{N-1} \end{bmatrix} \tag{3.32}$$

$$\tilde{Q}_\mathrm{y} = \begin{bmatrix} Q_\mathrm{y} & 0 & 0 & 0 \\ 0 & Q_\mathrm{y} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_\mathrm{y} \end{bmatrix}, \tilde{Q}_\mathrm{u} = \begin{bmatrix} Q_\mathrm{u} & 0 & 0 & 0 \\ 0 & Q_\mathrm{u} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_\mathrm{u} \end{bmatrix} \tag{3.33}$$

Naším cieľom je vytvoriť problém s jedinou optimalizovanou premennou U,

$$U = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} \tag{3.34}$$

preto si z ohraničení vyjadríme predikciu výstupov Y a diferenciu akčných zásahov ΔU ako funkciu U a tieto vzťahy substituujeme do účelovej funkcie.

Výraz pre Y získame zo stavového modelu nasledovnými matematickými úpravami:

$$x_1 = Ax_0 + Bu_0 \tag{3.35a}$$

$$y_0 = Cx_0 + Du_0 \tag{3.35b}$$

$$\begin{aligned} x_2 &= Ax_1 + Bu_1 \\ &= A(Ax_0 + Bu_0) + Bu_1 \\ &= A^2x_0 + ABu_0 + Bu_1 \end{aligned} \tag{3.36a}$$

$$\begin{aligned} y_1 &= Cx_1 + Du_1 \\ &= C(Ax_0 + Bu_0) + Du_1 \\ &= CAx_0 + CBu_0 + Du_1 \end{aligned} \tag{3.36b}$$

$$\begin{aligned} y_2 &= Cx_2 + Du_2 \\ &= C(A^2x_0 + ABu_0 + Bu_1) + Du_2 \\ &= CA^2x_0 + CABu_0 + CBu_1 + Du_2 \end{aligned} \tag{3.37}$$

Z rovníc si môžeme všimnúť, že ľubovoľný výstup je závislý iba od počiatočného stavu x_0 a vstupov. Preto môžeme pôvodný stavový model prepísať do tvaru:

$$Y = \tilde{C}x_0 + \tilde{D}U \tag{3.38}$$

kde

$$\tilde{C} = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{N-1} \end{bmatrix}, \tilde{D} = \begin{bmatrix} D & 0 & 0 & 0 & 0 & 0 \\ CB & D & 0 & 0 & 0 & 0 \\ CAB & CB & D & 0 & 0 & 0 \\ CA^2B & CAB & CB & D & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ CA^{N-2}B & CA^{N-3}B & \dots & CAB & CB & D \end{bmatrix} \tag{3.39}$$

Na základe definície Δu môžeme vektor ΔU zapísať ako:

$$\Delta U = \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \Delta u_2 \\ \vdots \\ \Delta u_{N-1} \end{bmatrix} = \begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ u_2 - u_1 \\ \vdots \\ u_{N-1} - u_{N-2} \end{bmatrix} \tag{3.40}$$

Podobne ako pri vyjadrovaní Y, aj tu zisťujeme, že ΔU je závislé od počiatočnej podmienky pre akčný zásah a vstupov:

$$\Delta U = \Lambda U + \lambda u_{-1} \tag{3.41}$$

kde

$$\Lambda = \begin{bmatrix} I_{n_\mathrm{u}} & 0 & 0 & 0 & 0 \\ -I_{n_\mathrm{u}} & I_{n_\mathrm{u}} & 0 & 0 & 0 \\ 0 & -I_{n_\mathrm{u}} & I_{n_\mathrm{u}} & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & -I_{n_\mathrm{u}} & I_{n_\mathrm{u}} \end{bmatrix}, \lambda = \begin{bmatrix} -I_{n_\mathrm{u}} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.42}$$

Potom, ako sme zadefinovali všetky matice a vektory, spravíme substitúciu do účelovej funkcie:

$$\begin{aligned} &(Y-R)^T\tilde{Q}_\mathrm{y}(Y-R) \\ &\quad = (\tilde{C}x_0 + \tilde{D}U - R)^T\tilde{Q}_\mathrm{y}(\tilde{C}x_0 + \tilde{D}U - R) \\ &\quad = x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}\tilde{C}x_0 + x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}\tilde{D}U - x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}R \\ &\qquad + U^\intercal\tilde{D}^\intercal\tilde{Q}_\mathrm{y}\tilde{C}x_0 + U^\intercal\tilde{D}^\intercal\tilde{Q}_\mathrm{y}\tilde{D}U - U^\intercal\tilde{D}^\intercal\tilde{Q}_\mathrm{y}R \\ &\qquad - R^\intercal\tilde{Q}_\mathrm{y}\tilde{C}x_0 - R^\intercal\tilde{Q}_\mathrm{y}\tilde{D}U + R^\intercal\tilde{Q}_\mathrm{y}R \\ &\quad = U^\intercal\tilde{D}^\intercal\tilde{Q}_\mathrm{y}\tilde{D}U + (2x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}\tilde{D} - 2R^\intercal\tilde{Q}_\mathrm{y}\tilde{D})U \\ &\qquad + (x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}\tilde{C}x_0 - x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}R - R^\intercal\tilde{Q}_\mathrm{y}\tilde{C}x_0 + R^\intercal\tilde{Q}_\mathrm{y}R) \end{aligned} \tag{3.43}$$

$$\begin{aligned} \Delta U^T\tilde{Q}_\mathrm{u}\Delta U &= (\Lambda U + \lambda u_{-1})^T\tilde{Q}_\mathrm{u}(\Lambda U + \lambda u_{-1}) \\ &= U^\intercal\Lambda^\intercal\tilde{Q}_\mathrm{u}\Lambda U + U^\intercal\Lambda^\intercal\tilde{Q}_\mathrm{u}\lambda u_{-1} + u_{-1}^T\lambda^\intercal\tilde{Q}_\mathrm{u}\Lambda U + u_{-1}^T\lambda^\intercal\tilde{Q}_\mathrm{u}\lambda u_{-1} \\ &= U^\intercal\Lambda^\intercal\tilde{Q}_\mathrm{u}\Lambda U + 2u_{-1}^T\lambda^\intercal\tilde{Q}_\mathrm{u}\Lambda U + u_{-1}^T\lambda^\intercal\tilde{Q}_\mathrm{u}\lambda u_{-1} \end{aligned} \tag{3.44}$$

Sériou všetkých uvedených krokov sme eliminovali ohraničenia v tvare rovností. Ak združíme všetky kvadratické výrazy do jedného, vieme definovať maticu P kvadratického optimalizačného problému.

$$P = \tilde{D}^\intercal\tilde{Q}_\mathrm{y}\tilde{D} + \Lambda^\intercal\tilde{Q}_\mathrm{u}\Lambda \tag{3.45}$$

Rovnako môžeme spojiť aj lineárne a skalárne výrazy, čím dostaneme vzťahy pre q a r.

$$q = (2x_0^\intercal\tilde{C}^\intercal\tilde{Q}_\mathrm{y}\tilde{D} - 2R^\intercal\tilde{Q}_\mathrm{y}\tilde{D} + 2u_{-1}^T\lambda^\intercal\tilde{Q}_\mathrm{u}\Lambda)^\intercal \tag{3.46}$$

$$r = x_0^\intercal \tilde{C}^\intercal \tilde{Q}_\mathrm{y} \tilde{C} x_0 - x_0^\intercal \tilde{C}^\intercal \tilde{Q}_\mathrm{y} R - R^\intercal \tilde{Q}_\mathrm{y} \tilde{C} x_0 + R^\intercal \tilde{Q}_\mathrm{y} R + u_{-1}^T \lambda^\intercal \tilde{Q}_\mathrm{u} \lambda u_{-1} \tag{3.47}$$

Posledným krokom je prepísať pôvodné ohraničenia na vstupy a výstupy do maticovej podoby. Tieto ohraničenia:

$$y_{\min} \leq y_k \leq y_{\max} \tag{3.48a}$$
$$u_{\min} \leq u_k \leq u_{\max} \tag{3.48b}$$
$$\Delta u_{\min} \leq \Delta u_k \leq \Delta u_{\max} \tag{3.48c}$$

môžeme prepísať ako:

$$Y \leq Y_{\max} \tag{3.49a}$$
$$-Y \leq -Y_{\min} \tag{3.49b}$$
$$U \leq U_{\max} \tag{3.49c}$$
$$-U \leq -U_{\min} \tag{3.49d}$$
$$\Delta U \leq \Delta U_{\max} \tag{3.49e}$$
$$-\Delta U \leq -\Delta U_{\min} \tag{3.49f}$$

Rovnako ako predtým, aj sem substituujeme výrazy pre Y a ΔU, aby sme pracovali iba s jednou optimalizovanou premennou U.

$$\tilde{C} x_0 + \tilde{D} U \leq Y_{\max} \tag{3.50a}$$
$$-(\tilde{C} x_0 + \tilde{D} U) \leq -Y_{\min} \tag{3.50b}$$
$$U \leq U_{\max} \tag{3.50c}$$
$$-U \leq -U_{\min} \tag{3.50d}$$
$$\Lambda U + \lambda u_{-1} \leq \Delta U_{\max} \tag{3.50e}$$
$$-(\Lambda U + \lambda u_{-1}) \leq -\Delta U_{\min} \tag{3.50f}$$

Výrazy obsahujúce U necháme na ľavej strane a všetky ostatné presu-

nieme na pravú stranu, aby sme vedeli zadefinovať matice A a b.

$$\begin{aligned} U &\leq U_{\max} && \text{(3.51a)} \\ -U &\leq -U_{\min} && \text{(3.51b)} \\ \tilde{D}U &\leq Y_{\max} - \tilde{C}x_0 && \text{(3.51c)} \\ -\tilde{D}U &\leq -Y_{\min} + \tilde{C}x_0 && \text{(3.51d)} \\ \Lambda U &\leq \Delta U_{\max} - \lambda u_{-1} && \text{(3.51e)} \\ -\Lambda U &\leq -\Delta U_{\min} + \lambda u_{-1} && \text{(3.51f)} \end{aligned}$$

$$A = \begin{bmatrix} I_{Nn_{\mathrm{u}}} \\ -I_{Nn_{\mathrm{u}}} \\ \tilde{D} \\ -\tilde{D} \\ \Lambda \\ -\Lambda \end{bmatrix}, b = \begin{bmatrix} U_{\max} \\ -U_{\min} \\ Y_{\max} - \tilde{C}x_0 \\ -Y_{\min} + \tilde{C}x_0 \\ \Delta U_{\max} - \lambda u_{-1} \\ -\Delta U_{\min} + \lambda u_{-1} \end{bmatrix} \tag{3.52}$$

Týmito úpravami sme získali štandardný tvar kvadratického programu, ktorý môžeme riešiť pomocou vybraného solveru.

Kapitola 4

Explicitné prediktívne riadenie

V dnešnej dobe priemysel čoraz viac siaha po prediktívnom riadení. Cieľom každého výrobného podniku je vyprodukovať čo najvyšší zisk pri čo najnižších nákladoch a zostať tak konkurencieschopný. Pri tejto snahe sa treba vysporiadať s mnohými výzvami, napr. ako čo najefektívnejšie využiť výrobnú kapacitu alebo obmedzené zdroje podniku. Prediktívne riadenie vie na takéto otázky odpovedať, pretože jeho jadro tvorí optimalizácia. [20]

Je zrejmé, že MPC má svoje nepopierateľné klady, avšak spája sa s ním aj niekoľko nevýhôd. Hlavnou nevýhodou klasického prediktívneho riadenia predstaveného v predošlej časti je výpočtová náročnosť. V tradičnom prístupe, nazývame ho aj implicitné prediktívne riadenie, najskôr preformulujeme pôvodný optimalizačný problém na kvadratický program a potom získavame optimálne akčné zásahy online, riešením kvadratického programovania v každej perióde vzorkovania, vždy pre nové počiatočné podmienky, čo sú zvyčajne aktualizované merania stavov, s použitím numerických optimalizačných metód. V skratke to znamená, že potrebujeme solver, ktorý bude pracovať v reálnom čase a neustále dodávať nové sekvencie optimálnych akčných zásahov. Pri tomto prístupe je na softvér aj hardvér kladená obrovská zátaž, pretože solver musí všetky operácie stihnúť za dobu jednej periódy vzorkovania. Ak solver nestihne problém vyriešiť, v najlepšom prípade sa zhorší kvalita riadenia, ale v najhoršom môže dôjsť k vážnej chybe celej prevádzky. Napriek tomu, že optimalizačné algoritmy sa stále zlepšujú, implementovať prediktívne riadenie je aj dnes pomerne náročné. Mnohé podniky sa rozhodnú, že ani za cenu zlepšenia výkonu sa im neoplatí investovať do nového a drahého vybavenia. [19]

Kvôli spomenutej nevýhode klasického prístupu sa nedávno prišlo so stratégiou, ktorá sa nazýva explicitné prediktívne riadenie. Pri tejto metóde sa odbúra výpočtová náročnosť tým, že celý proces optimalizácie prebehne offline, iba raz a na akomkoľvek zariadení. Explicitné prediktívne riadenie je založené na multiparametrickom programovaní, kedy riešime optimalizačný problém pre všetky zlučiteľné počiatočné podmienky. Riešením multiparametrického programovania je explicitná funkcia meniacich sa parametrov systému, najčastejšie stavov.

Keď sa rozhodneme pre explicitné MPC, vyhneme sa opakovanej optimalizácii. Optimálne akčné zásahy budeme získavať pomocou predpočítanej funkcie vždy po novom odmeraní parametrov. Výpočtové zaťaženie je tu značne redukované, pretože získavanie sekvencie akčných zásahov je séria jednoduchých vyhodnotení funkcie. Takáto predpočítaná explicitná funkcia pre veľkú triedu optimalizačných problémov nadobúda tvar PWA (po častiach afinnej) funkcie. Je dobré poznamenať, že oba prístupy vedú k rovnakému výsledku.

Definičný obor PWA funkcie je rozdelený do konečného počtu regiónov, ktoré sa nazývajú aj kritické. Tieto regióny prislúchajú afinným funkciám, ktoré určujú, akú hodnotu bude mať akčný zásah pri daných meraniach stavov.

Podobne ako implicitné prediktívne riadenie, aj explicitné má svoje výhody a nevýhody.

Výhody

- Explicitný regulátor je výsledkom jednorazovej offline optimalizácie. Online implementácia je redukovaná na jednoduché vyhodnocovanie funkcie.
- Explicitné regulátory môžeme implementovať aj na jednoduchom a cenovo dostupnom hardvéri ako je napr. PLC
- Výpočtová náročnosť online fázy nie je veľká, takže explicitné MPC je vhodné aj pre systémy s malou periódou vzorkovania.

Nevýhody

- Na uchovanie všetkých regiónov je potrebné veľké množstvo pamäte, čo sa ukazuje ako problém hlavne pri snahe implementovať explicitné MPC na jednoduchší hardvér.
- Ďalšia limitácia explicitného prediktívneho riadenia je, že parametrické riešenie platí iba pre konkrétny problém, pre jednu voľbu matíc kvadratického programu. Ak sa zmenia parametre problému alebo ohraničenia, rovnaké parametrické riešenie sa už nedá použiť. [14]
- Implementácia explicitného MPC stále môže byť drahá pre veľké optimalizačné problémy. Náročnosť výpočtu multiparametrického problému rýchlo rastie s veľkosťou problému. Zložitosť sa odvíja od predikčného horizontu, čo vo svojej podstate je počet optimalizovaných premenných a ohraničení.

4.1 Multiparametrické programovanie

Explicitné prediktívne riadenie je založené na multiparametrickom programovaní. Vyznačuje sa tým, že optimálne riešenie získavame pre celý rozsah hodnôt parametrov (zlučiteľných počiatočných podmienok). Cieľom je získať explicitnú funkciu, na základe ktorej budeme vedieť vypočítať akčné zásahy. [19]

Budeme uvažovať nasledovný problém:

$$\min_{z} \quad J(z,\theta) \tag{4.1a}$$

$$\text{s.t.} \quad Gz \leq E\theta + w \tag{4.1b}$$

kde $z \in \mathbb{R}^{n_z}$ je vektor optimalizovaných premenných a $\theta \in \mathbb{R}^{n_\theta}$ je vektor parametrov. Ohraničenia sú reprezentované maticami $G \in \mathbb{R}^{n \times n_z}$, $E \in \mathbb{R}^{n \times n_\theta}$ a $w \in \mathbb{R}^n$, kde n je počet ohraničení.

Riešenie multiparametrického programovania sa skladá z:

- zákona riadenia, ktorý môžeme nazvať tiež optimálnym riešením, t.j. $\mu^*(\theta) = \text{argmin } J(z,\theta)\ s.t.\ Gz \leq E\theta + w$, vo forme polytopickej PWA funkcie.

- Účelovej funkcie $J^*(\theta)$
- Polytopickej partície zloženej z polytopov nazývaných regióny, ktorá udáva oblasť všetkých zlučiteľných hodnôt parametrov θ.

$$\Omega = \{\theta \in \mathcal{X} | \exists z : Gz \leq E\theta + w\} \tag{4.2}$$

Na tejto oblasti sú definované funkcie $\mu^*(\theta)$ a $J^*(\theta)$

4.2 Multiparametrické kvadratické programovanie

Namiesto všeobecného problému budeme uvažovať problém kvadratického programovania v nasledovnom tvare:

$$\min_U \quad \frac{1}{2}U^\intercal HU + (q^\intercal\theta + f_\mathrm{U})^\intercal U + \theta Y\theta + f_\theta^T\theta + f \tag{4.3a}$$

$$\text{s.t.} \quad Gz \leq E\theta + w \tag{4.3b}$$

kde $U \in \mathbb{R}^{n_\mathrm{U}}$, $\theta \in \mathbb{R}^{n_\theta}$, $H \in \mathbb{R}^{n_\mathrm{U} \times n_\mathrm{U}}$, $q \in \mathbb{R}^{n_\theta \times n_\mathrm{U}}$, $f_\mathrm{U} \in \mathbb{R}^{n_\mathrm{U}}$, $Y \in \mathbb{R}^{n_\theta \times n_\theta}$, $f_\theta \in \mathbb{R}^{n_\theta}$, $f \in \mathbb{R}$,$G \in \mathbb{R}^{n \times n_\mathrm{U}}$, $E \in \mathbb{R}^{n \times n_\theta}$ a $w \in \mathbb{R}^n$. Pre maticu H platí podmienka $H = H^\intercal \succ 0$.

Riešenie sa bude skladať z nasledujúcich častí:

- zákon riadenia

$$U^*(\theta) = \operatorname{argmin} \quad \frac{1}{2}U^\intercal HU + (q^\intercal\theta + f_\mathrm{U})^\intercal U + \theta Y\theta + f_\theta^T\theta + f \tag{4.4a}$$

$$\text{s.t.} \quad Gz \leq E\theta + w \tag{4.4b}$$

- účelová funkcia $J^*(\theta)$
- polytopická partícia $\Omega = \cup_i \mathcal{R}_i$ pozostávajúca z $i = 1, \ldots, M$ kritických regiónov

4.2.1 Formulácia kvadratického programu

Problém regulácie

Môžeme si všimnúť, že implicitné a explicitné prediktívne riadenie neuvažuje rovnaký tvar optimalizačného problému.

Pri explicitnom MPC budeme používať problém v tvare:

$$\min_{U} \quad \frac{1}{2}U^\intercal HU + (q^\intercal\theta + f_{\mathrm{U}})^\intercal U + \theta Y\theta + f_\theta^T\theta + f \tag{4.5a}$$

$$\text{s.t.} \quad Gz \leq E\theta + w \tag{4.5b}$$

Z rovnice 3.22 vieme zadefinovať 4 členy účelovej funkcie:

$$H = 2(\tilde{B}^\intercal\tilde{Q}_{\mathrm{x}}\tilde{B} + \tilde{Q}_{\mathrm{u}}), \tag{4.6}$$

$$\theta = x_0, \tag{4.7}$$

$$q = 2\tilde{A}^\intercal\tilde{Q}_{\mathrm{x}}\tilde{B}, \tag{4.8}$$

$$Y = \tilde{A}^\intercal\tilde{Q}_{\mathrm{x}}\tilde{A} \tag{4.9}$$

Ostatné členy budú pri probléme regulácie nulové. Ohraničenia rovnako ako predtým rozdelíme a upravíme tak, aby sme vedeli vyformovať matice G, E a w.

$$x_{\min} \leq x_k \leq x_{\max} \tag{4.10a}$$

$$u_{\min} \leq u_k \leq u_{\max} \tag{4.10b}$$

$$U \leq U_{\max} \tag{4.11a}$$

$$-U \leq -U_{\min} \tag{4.11b}$$

$$X \leq X_{\max} \tag{4.11c}$$

$$-X \leq -X_{\min} \tag{4.11d}$$

$$U \leq U_{\max} \tag{4.12a}$$

$$-U \leq -U_{\min} \tag{4.12b}$$

$$\tilde{B}U \leq X_{\max} - \tilde{A}x_0 \tag{4.12c}$$

$$-\tilde{B}U \leq -X_{\min} + \tilde{A}x_0 \tag{4.12d}$$

$$G = \begin{bmatrix} I \\ -I \\ \tilde{B} \\ -\tilde{B} \end{bmatrix}, E = \begin{bmatrix} 0 \\ 0 \\ -\tilde{A} \\ \tilde{A} \end{bmatrix}, w = \begin{bmatrix} U_{\max} \\ -U_{\min} \\ X_{\max} \\ -X_{\min} \end{bmatrix} \tag{4.13}$$

Problém sledovania

V prípade problému sledovania budeme vychádzať z rovníc 3.43 a 3.44. Problém sledovania vyžaduje dve počiatočné podmienky, x_0 pre stavy a u_{-1} pre vstupy, a parameter θ bude obsahovať obe.

$$\theta = \begin{bmatrix} x_0 \\ u_{-1} \end{bmatrix} \tag{4.14}$$

Po zavedení θ vieme zadefinovať ostatné členy účelovej funkcie problému 4.5.

$$H = 2(\tilde{D}^\intercal \tilde{Q}_\mathrm{y} \tilde{D} + \Lambda \tilde{Q}_\mathrm{u} \Lambda), \tag{4.15}$$

$$f = R^\intercal \tilde{Q}_\mathrm{y} R, \tag{4.16}$$

$$Y = \begin{bmatrix} \tilde{C}^\intercal \tilde{Q}_\mathrm{y} \tilde{C} & 0 \\ 0 & \lambda^\intercal \tilde{Q}_\mathrm{u} \lambda \end{bmatrix}, \tag{4.17}$$

$$f_\mathrm{U} = -2\tilde{D}^\intercal \tilde{Q}_\mathrm{y} R, \tag{4.18}$$

$$f_\theta = \begin{bmatrix} -2\tilde{C}^\intercal \tilde{Q}_\mathrm{y} R \\ 0 \end{bmatrix}, \tag{4.19}$$

$$q = \begin{bmatrix} 2\tilde{C}^\intercal \tilde{Q}_\mathrm{y} \tilde{D} \\ 2\lambda^\intercal \tilde{Q}_\mathrm{u} \Lambda \end{bmatrix} \tag{4.20}$$

Štruktúra problému sledovania je zložitejšia ako štruktúra problému regulácie, preto sme tentokrát vedeli vytvoriť všetky členy. Pri ohraničeniach postupujeme rovnako ako predtým.

$$y_\mathrm{min} \leq y_k \leq y_\mathrm{max} \tag{4.21a}$$

$$u_\mathrm{min} \leq u_k \leq u_\mathrm{max} \tag{4.21b}$$

$$\Delta u_\mathrm{min} \leq \Delta u_k \leq \Delta u_\mathrm{max} \tag{4.21c}$$

$$Y \leq Y_\mathrm{max} \tag{4.22a}$$

$$-Y \leq -Y_\mathrm{min} \tag{4.22b}$$

$$U \leq U_\mathrm{max} \tag{4.22c}$$

$$-U \leq -U_\mathrm{min} \tag{4.22d}$$

$$\Delta U \leq \Delta U_\mathrm{max} \tag{4.22e}$$

$$-\Delta U \leq -\Delta U_\mathrm{min} \tag{4.22f}$$

$$
\begin{aligned}
U &\leq U_{\max} && \text{(4.23a)}\\
-U &\leq -U_{\min} && \text{(4.23b)}\\
\tilde{D}U &\leq Y_{\max} - \tilde{C}x_0 && \text{(4.23c)}\\
-\tilde{D}U &\leq -Y_{\min} + \tilde{C}x_0 && \text{(4.23d)}\\
\Lambda U &\leq \Delta U_{\max} - \lambda u_{-1} && \text{(4.23e)}\\
-\Lambda U &\leq -\Delta U_{\min} + \lambda u_{-1} && \text{(4.23f)}
\end{aligned}
$$

$$
G = \begin{bmatrix} I \\ -I \\ \tilde{D} \\ -\tilde{D} \\ \Lambda \\ -\Lambda \end{bmatrix}, E = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -\tilde{C} & 0 \\ \tilde{C} & 0 \\ 0 & -\lambda \\ 0 & \lambda \end{bmatrix}, w = \begin{bmatrix} U_{\max} \\ -U_{\min} \\ Y_{\max} \\ -Y_{\min} \\ \Delta U_{\max} \\ -\Delta U_{\min} \end{bmatrix} \tag{4.24}
$$

4.2.2 Riešenie multiparametrického programovania

Multiparametrické programovanie riešime pomocou Karush-Kuhn-Tuckerových podmienok optimality. Uvažujme problém v tvare:

$$
\begin{aligned}
\min_{U} \quad & \frac{1}{2}U^\intercal HU + (q^\intercal\theta + f_{\mathrm{U}})^\intercal U + \theta Y\theta + f_\theta^T\theta + f && \text{(4.25a)}\\
\text{s.t.} \quad & Gz \leq E\theta + w && \text{(4.25b)}
\end{aligned}
$$

Keď platí $H = H^\intercal \succ 0$, kvadratický program je konvexný a KKT podmienky sú nutné a zároveň postačujúce na to, aby U^* bolo globálne optimum.

V prvom kroku vytvoríme Lagrangián:

$$
\mathcal{L} = \frac{1}{2}U^T HU + (q^\intercal\theta + f_{\mathrm{U}})^\intercal U + \theta^\intercal Y\theta + f_\theta^\intercal\theta + f + \lambda^\intercal(GU - E\theta - w) \tag{4.26}
$$

a naformulujeme KKT podmienky:

$$HU + q^\intercal\theta + f_\mathrm{U} + G^\intercal\lambda = 0 \tag{4.27a}$$
$$GU - E\theta - w \leq 0 \tag{4.27b}$$
$$\lambda \geq 0 \tag{4.27c}$$
$$\lambda^\intercal(GU - E\theta - w) = 0 \tag{4.27d}$$

Prvú rovnicu nazývame podmienka stacionarity, druhá rovnica je primárna zlučiteľnosť, tretia je duálna zlučiteľnosť a posledná sa volá doplnková voľnosť.

Aby sme mohli pokračovať, je dôležité porozumieť aktívnym a neaktívnym ohraničeniam. Uvažujme, že máme ohraničenie v tvare nerovnosti:

$$A_i x \leq b_i \tag{4.28}$$

Ohraničenie je nazývané aktívne v x, ak platí

$$A_i x = b_i \tag{4.29}$$

a neaktívne, ak platí:

$$A_i x < b_i \tag{4.30}$$

To znamená, že ohraničenia v tvare rovností sú vždy aktívne. [21] Teóriu aktívnych (index A) a neaktívnych (index N) ohraničení aplikujeme na KKT podmienky, čím vznikne: [19]

$$HU + q^\intercal\theta + f_\mathrm{U} + G_\mathrm{A}^\intercal\lambda_\mathrm{A} = 0 \tag{4.31a}$$
$$G_\mathrm{A}U - E_\mathrm{A}\theta - w_\mathrm{A} = 0 \tag{4.31b}$$
$$G_\mathrm{N}U - E_\mathrm{N}\theta - w_\mathrm{N} < 0 \tag{4.31c}$$
$$\lambda_\mathrm{A} > 0 \tag{4.31d}$$
$$\lambda_\mathrm{N} = 0 \tag{4.31e}$$

Doplnkovú voľnosť nie je nutné uvažovať. Z podmienky stacionarity si vyjadríme akčný zásah U:

$$HU = -q^\intercal\theta - f_\mathrm{U} - G_\mathrm{A}^\intercal\lambda_\mathrm{A} \tag{4.32}$$
$$U = -H^{-1}(q^\intercal\theta + f_\mathrm{U} + G_\mathrm{A}^\intercal\lambda_\mathrm{A}) \tag{4.33}$$

Výraz (4.33) substituujeme do podmienky primárnej zlučiteľnosti.

$$-G_{\mathrm{A}}H^{-1}(q^{\intercal}\theta + f_{\mathrm{U}} + G_{\mathrm{A}}^{\intercal}\lambda_{\mathrm{A}}) - E_{\mathrm{A}}\theta - w_{\mathrm{A}} = 0 \tag{4.34}$$

$$-G_{\mathrm{A}}H^{-1}q^{\intercal}\theta - G_{\mathrm{A}}H^{-1}f_{\mathrm{U}} - G_{\mathrm{A}}H^{-1}G_{\mathrm{A}}^{\intercal}\lambda_{\mathrm{A}} - E_{\mathrm{A}}\theta - w_{\mathrm{A}} = 0 \tag{4.35}$$

$$G_{\mathrm{A}}H^{-1}G_{\mathrm{A}}^{\intercal}\lambda_{\mathrm{A}} = (-G_{\mathrm{A}}H^{-1}q^{\intercal} - E_{\mathrm{A}})\theta + (-G_{\mathrm{A}}H^{-1}f_{\mathrm{U}} - w_{\mathrm{A}}) \tag{4.36}$$

$$\lambda_{\mathrm{A}} = (G_{\mathrm{A}}H^{-1}G_{\mathrm{A}}^{\intercal})^{-1}(-G_{\mathrm{A}}H^{-1}q^{\intercal} - E_{\mathrm{A}})\theta + (G_{\mathrm{A}}H^{-1}G_{\mathrm{A}}^{\intercal})^{-1}(-G_{\mathrm{A}}H^{-1}f_{\mathrm{U}} - w_{\mathrm{A}}) \tag{4.37}$$

Optimálne Lagrangeove násobiče λ_{A} môžeme kompaktne vyjadriť vo forme:

$$\lambda_{\mathrm{A}} = \alpha_{\lambda}\theta + \beta_{\lambda} \tag{4.38}$$

kde

$$\alpha_{\lambda} = (G_{\mathrm{A}}H^{-1}G_{\mathrm{A}}^{\intercal})^{-1}(-G_{\mathrm{A}}H^{-1}q^{\intercal} - E_{\mathrm{A}}) \tag{4.39}$$

$$\beta_{\lambda} = (G_{\mathrm{A}}H^{-1}G_{\mathrm{A}}^{\intercal})^{-1}(-G_{\mathrm{A}}H^{-1}f_{\mathrm{U}} - w_{\mathrm{A}}) \tag{4.40}$$

Nakoniec spätne dosadíme rovnicu (4.38) do vzťahu (4.33).

$$\begin{aligned} U &= -H^{-1}(q^{\intercal}\theta + f_{\mathrm{U}} + G_{\mathrm{A}}^{\intercal}(\alpha_{\lambda}\theta + \beta_{\lambda})) \\ &= -H^{-1}q^{\intercal}\theta - H^{-1}f_{\mathrm{U}} - H^{-1}G_{\mathrm{A}}^{\intercal}\alpha_{\lambda}\theta - H^{-1}G_{\mathrm{A}}^{\intercal}\beta_{\lambda} \\ &= (-H^{-1}q^{\intercal} - H^{-1}G_{\mathrm{A}}^{\intercal}\alpha_{\lambda})\theta + (-H^{-1}f_{\mathrm{U}} - H^{-1}G_{\mathrm{A}}^{\intercal}\beta_{\lambda}) \end{aligned} \tag{4.41}$$

Podobne ako predtým, aj U si môžeme vyjadriť ako lineárnu funkciu parametrov θ.

$$U = \alpha_{\mathrm{U}}\theta + \beta_{\mathrm{U}} \tag{4.42}$$

kde

$$\alpha_{\mathrm{U}} = (-H^{-1}q^{\intercal} - H^{-1}G_{\mathrm{A}}^{\intercal}\alpha_{\lambda}) \tag{4.43}$$

$$\beta_{\mathrm{U}} = (-H^{-1}f_{\mathrm{U}} - H^{-1}G_{\mathrm{A}}^{\intercal}\beta_{\lambda}) \tag{4.44}$$

V tomto bode vieme zostrojiť kritický región spojením podmienky primárnej zlučiteľnosti pre neaktívne ohraničenia a podmienky duálnej zlučiteľnosti pre aktívne ohraničenia. Do oboch nerovníc (4.31d) a (4.31e) dosadíme vzťah odvodený pre optimálne akčné zásahy (4.42) a optimálne Lagrangeove násobiče (4.38). Okrem toho nahradíme ostré nerovnosti za neostré.

$$G_{\mathrm{N}}U - E_{\mathrm{N}}\theta - w_{\mathrm{N}} \leq 0 \tag{4.45}$$
$$G_{\mathrm{N}}(\alpha_{\mathrm{U}}\theta + \beta_{\mathrm{U}}) - E_{\mathrm{N}}\theta - w_{\mathrm{N}} \leq 0 \tag{4.46}$$
$$(G_{\mathrm{N}}\alpha_{\mathrm{U}} - E_{\mathrm{N}})\theta \leq w_{\mathrm{N}} - G_{\mathrm{N}}\beta_{\mathrm{U}} \tag{4.47}$$

$$\lambda_{\mathrm{A}} \geq 0 \tag{4.48}$$
$$-\lambda_{\mathrm{A}} \leq 0 \tag{4.49}$$
$$-(\alpha_\lambda\theta + \beta_\lambda) \leq 0 \tag{4.50}$$
$$-\alpha_\lambda\theta \leq \beta_\lambda \tag{4.51}$$

Skonštruujeme matice A a b

$$A = \begin{bmatrix} G_{\mathrm{N}}\alpha_{\mathrm{U}} - E_{\mathrm{N}} \\ -\alpha_\lambda \end{bmatrix}, b = \begin{bmatrix} w_{\mathrm{N}} - G_{\mathrm{N}}\beta_{\mathrm{U}} \\ \beta_\lambda \end{bmatrix} \tag{4.52}$$

kritického regiónu $\mathcal{R}$

$$\mathcal{R} = \{\theta \in \mathbb{R}^{n_\theta} | A\theta \leq b\} \tag{4.53}$$

Účelová funkcia $J^*(\theta)$ sa tiež získa dosadením rovnice (4.42).

$$\begin{aligned}
&\min \quad \frac{1}{2}U^T H U + (q^\intercal\theta + f_{\mathrm{U}})^\intercal U + \theta^\intercal Y\theta + f_\theta^\intercal\theta + f \\
&= \min \quad \frac{1}{2}(\alpha_{\mathrm{U}}\theta + \beta_{\mathrm{U}})^\intercal H(\alpha_{\mathrm{U}}\theta + \beta_{\mathrm{U}}) + (q^\intercal\theta + f_{\mathrm{U}})^\intercal(\alpha_{\mathrm{U}}\theta + \beta_{\mathrm{U}}) + \theta^\intercal Y\theta + f_\theta^\intercal\theta + f \\
&= \min \quad \theta^\intercal\left(\frac{1}{2}\alpha_{\mathrm{U}}^\intercal H\alpha_{\mathrm{U}} + q\alpha_{\mathrm{U}} + Y\right)\theta + (\beta_{\mathrm{U}}^\intercal H\alpha_{\mathrm{U}} + f_{\mathrm{U}}^\intercal\alpha_{\mathrm{U}} + \beta_{\mathrm{U}}^\intercal q^\intercal + f_\theta^\intercal)\theta \\
&\quad + \left(\frac{1}{2}\beta_U^\intercal H\beta_U + f_{\mathrm{U}}^\intercal\beta_{\mathrm{U}} + f\right) \\
&= \min \quad \theta^\intercal\alpha_j\theta + \beta_j\theta + \gamma_j
\end{aligned} \tag{4.54}$$

$$\alpha_j = \frac{1}{2}\alpha_{\mathrm{U}}^{\intercal} H \alpha_{\mathrm{U}} + q\alpha_{\mathrm{U}} + Y, \tag{4.55}$$

$$\beta_j = \beta_{\mathrm{U}}^{\intercal} H \alpha_{\mathrm{U}} + f_{\mathrm{U}}^{\intercal}\alpha_{\mathrm{U}} + \beta_{\mathrm{U}}^{\intercal} q^{\intercal} + f_{\theta}^{\intercal}, \tag{4.56}$$

$$\gamma_j = \frac{1}{2}\beta_U^{\intercal} H \beta_U + f_{\mathrm{U}}^{\intercal}\beta_{\mathrm{U}} + f \tag{4.57}$$

Vlastnosti parametrického riešenia

Zákon riadenia

Zákon riadenia $U^*(\theta)$ je spojitá PWA funkcia definovaná na polytopickej partícii Ω.

$$U^*(\theta) = \begin{cases} \alpha_1\theta + \beta_1 & \text{ak } \theta \in \mathcal{R}_1 \\ \vdots & \\ \alpha_L\theta + \beta_L & \text{ak } \theta \in \mathcal{R}_L \end{cases} \tag{4.58}$$

Polytopická partícia

Polytopická partícia je zlučiteľná množina $\Omega \subset \mathbb{R}^{n_\theta}$ rozdelená do M kritických regiónov $\mathcal{R}_i$, $\cup_i \mathcal{R}_i = \Omega$. i-ty región definujeme:

$$\mathcal{R}_i = \{\theta \in \mathbb{R}^{n_\theta} | A_i\theta \leq b_i\} \tag{4.59}$$

s maticami $A_i \in \mathbb{R}^{n_{\mathrm{H}} \times n_\theta}$ a $b_i \in \mathbb{R}^{n_{\mathrm{H}}}$, kde n_{H} je počet polpriestorov určujúcich i-ty región.

Účelová funkcia

V kvadratickom parametrickom probléme je účelová funkcia $J^* : \Omega \in \mathbb{R}$ spojitá a konvexná PWQ (po častiach kvadratická) funkcia definovaná na polytopickej partícii Ω.

$$J^*(\theta) = \begin{cases} \theta\alpha_{j,1}\theta + \beta_{j,1}\theta + \gamma_{j,1} & \text{ak } \theta \in \mathcal{R}_1 \\ \vdots & \\ \theta\alpha_{j,m}\theta + \beta_{j,m}\theta + \gamma_{j,m} & \text{ak } \theta \in \mathcal{R}_L \end{cases} \tag{4.60}$$

4.3 Multiparametrické algoritmy

Cieľom multiparametrických algoritmov je syntetizovať vyššie uvedenú polytopickú partíciu Ω, ktorá je rozdelená do kritických regiónov $\mathcal{R}_i, i = 1, \ldots, M$ a na nej definovaný zákon riadenia $U^*(\theta)$ a účelovú funkciu $J^*(\theta)$. Algoritmus na riešenie multiparametrického programovania môžeme rozdeliť na tieto dve časti:

1. Hľadanie množín aktívnych ohraničení: V tejto fáze sa zameriavame na nájdenie všetkých kombinácií aktívnych ohraničení. Jedna takáto kombinácia sa volá množina aktívnych ohraničení a na jej základe vieme vygenerovať kritický región.

2. Riešenie KKT systému: Potom ako získame zoznam aktívnych množín, pre každú množinu zostrojíme kritický región, čím vznikne polytopická partícia pokrývajúca celú zlučiteľnú oblasť parametrov.

   Už skôr sme uviedli, že ohraničenie:

$$A_i x \leq b_i \tag{4.61}$$

je aktívne v x, ak platí

$$A_i x = b_i \tag{4.62}$$

a neaktívne, ak platí:

$$A_i x < b_i \tag{4.63}$$

Množina aktívnych ohraničení je taká množina, ktorá obsahuje všetky aktívne ohraničenia pre aktuálny bod x. Množina aktívnych ohraničení je dôležitá preto, lebo nám určuje, ktoré ohraničenia ovplyvňujú konečný výsledok optimalizácie.

4.3.1 Enumeračná metóda

Existuje viacero postupov, pomocou ktorých vieme nájsť zoznam množín aktívnych ohraničení. V tejto práci opíšeme metódu enumerácie, ktorá sa v multiparametrickom programovaní začala používať ako úplne prvá. V matematike alebo aj v oblasti informačných technológií pojem enumerácia vo

všeobecnosti znamená postupné a usporiadané vyhodnocovanie všetkých prvkov nejakej množiny. [22] V parametrickom programovaní je to vyhodnocovanie všetkých kombinácií ohraničení problému. Na začiatku vyhlásime každú kombináciu za kandidáta na aktívnu množinu. Potom sa presúvame od jedného kandidáta k druhému a postupne zisťujeme, či naozaj spĺňa podmienky na to byť množinou aktívnych ohraničení. Postup v tejto metóde je veľmi jednoduchý a priamočiary, zakladá sa na tom, že keď prejdeme všetky možnosti, máme istotu kompletného riešenia.

Napriek tejto garancii nebola metóda enumerácie v praxi populárna, pretože je časovo veľmi náročná. Výsledky ukazovali, že iba veľmi malá časť kandidátov sa vyhodnotila ako aktívna množina a viedla ku konštrukcii kritického regiónu. To znamená, že dochádzalo k množstvu zbytočného počítania a vyhodnocovania a teda aj veľkému zaťaženiu softvéru a hardvéru.

V roku 2011 prešla metóda enumerácie významnými modifikáciami. Gupta a kol. ju upravili tak, že sa znížila výpočtová náročnosť a celý proces hľadania množín aktívnych ohraničení prebiehal efektívnejšie. Do metódy zaviedli vyraďovacie kritérium, ktoré zredukovalo počet vyhodnotení jednotlivých kombinácií a zároveň poskytli garanciu, že riešenie bude obsahovať minimálny počet regiónov.

V matematike môžeme zoznam všetkých kombinácií ohraničení $\mathcal{W}$ zapísať nasledovne:

$$\begin{aligned}\mathcal{W} = \{&\mathcal{A}_1 = \{\}, \mathcal{A}_2 = \{1\}, \ldots, \mathcal{A}_{r+1} = \{r\}, \ldots, \\ &\mathcal{A}_{r+2} = \{1,2\}, \ldots, \mathcal{A}_{n_\mathrm{A}} = \{1,2,\ldots,r\}\}\end{aligned}$$

kde r je počet ohraničení a $n_\mathrm{A} = \binom{r}{n_\mathrm{U}}$, pričom n_U je rozmer vektora optimalizovaných premenných. Maximálny počet aktívnych množín je $k = \sum_{i=0}^{n_\mathrm{U}} \binom{r}{i}$.

Kandidátov skúmame pomocou riešenia nasledovného lineárneho programu:

$$\begin{aligned}
\max_{t,U,\theta,\lambda_A} \quad & t && \text{(4.64a)}\\
\text{s.t.} \quad & HU + q^\intercal\theta + G_\mathrm{A}^\intercal\lambda_\mathrm{A} = 0 && \text{(4.64b)}\\
& G_\mathrm{A}U - E_\mathrm{A}\theta - w_\mathrm{A} = 0 && \text{(4.64c)}\\
& t \leq w_\mathrm{N} + E_\mathrm{N}\theta - G_\mathrm{N}U && \text{(4.64d)}\\
& \lambda_\mathrm{A} \geq t && \text{(4.64e)}\\
& t \geq 0 && \text{(4.64f)}
\end{aligned}$$

ktorý má vo forme minimalizácie tvar:

$$\begin{aligned} \min \quad & -t && (4.65a)\\ \text{s.t.} \quad & HU + q^\intercal\theta + G_\mathrm{A}^\intercal\lambda_\mathrm{A} = 0 && (4.65b)\\ & G_\mathrm{A}U - E_\mathrm{A}\theta = w_\mathrm{A} && (4.65c)\\ & t + G_\mathrm{N}U - E_\mathrm{N}\theta \leq w_\mathrm{N} && (4.65d)\\ & t \leq \lambda_\mathrm{A} && (4.65e)\\ & -t \leq 0 && (4.65f) \end{aligned}$$

Aby sme tento problém vedeli riešiť cez niektorý solver určený na lineárne programovanie, prevedieme ho do štandardného tvaru :

$$\begin{aligned} \min \quad & c^\intercal z && (4.66a)\\ \text{s.t.} \quad & Az \leq b && (4.66b)\\ & A_\mathrm{eq}z = b_\mathrm{eq} && (4.66c) \end{aligned}$$

V Julii môžeme použiť napr. funkciu `linprog`.

Zavedieme si jednu optimalizovanú premennú z:

$$z = \begin{bmatrix} t \\ U \\ \theta \\ \lambda_\mathrm{A} \end{bmatrix} \tag{4.67}$$

a podľa (4.66) vytvoríme vektory c,b a b_eq a matice A a A_eq

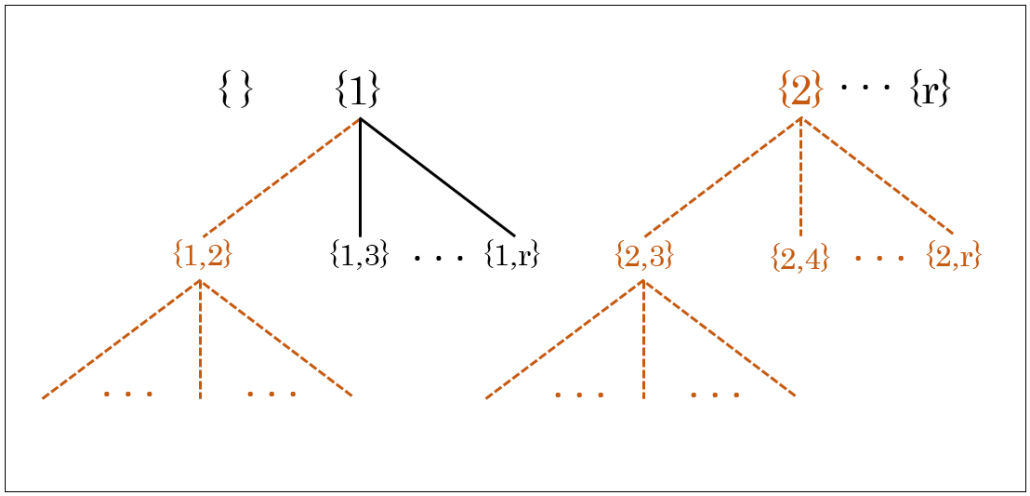
$$c = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.68}$$

$$A = \begin{bmatrix} 1 & G_\mathrm{N} & -E_\mathrm{N} & 0 \\ 1 & 0 & 0 & -I \\ -1 & 0 & 0 & 0 \end{bmatrix}, b = \begin{bmatrix} w_\mathrm{N} \\ 0 \\ 0 \end{bmatrix} \tag{4.69}$$

$$A_\mathrm{eq} = \begin{bmatrix} 0 & H & q^\intercal & G_\mathrm{A}^\intercal \\ 0 & G_\mathrm{A} & -E_\mathrm{A} & 0 \end{bmatrix}, b_\mathrm{eq} = \begin{bmatrix} 0 \\ w_\mathrm{A} \end{bmatrix} \tag{4.70}$$

Navrhovaná technika, ktorá implementuje vyraďovacie kritérium stavia na tom, že lineárny program (4.65), ktorý skúma množinu ohraničení môže byť optimálny, zlučiteľný alebo neriešiteľný. Ak sa problém vyhodnotí ako optimálny, množina ohraničení je aktívna. Ak je problém neriešiteľný, vyriešime ho ešte raz, ale tentokrát nebudeme uvažovať podmienku stacionarity. Ak je nový problém zlučiteľný, teda optimálny bez stacionarity, vyradíme zo zoznamu aktuálnu kombináciu a ak je znovu neriešiteľný, odstránime okrem aktuálnej kombinácie aj všetky ďalšie, ktoré ju obsahujú. Tento postup podstatne znižuje počet kandidátov, ktoré algoritmus musí preskúmať a preto znižuje časovú náročnosť celej metódy.

Celá procedúra je uvedená v algoritme 1.



Obr. 4.1: Vyraďovacie kritérium

Vstup: Zoznam všetkých kandidátov $\mathcal{W}$
Výstup: Zoznam aktívnych množín (optimálnych kandidátov) $\mathcal{W}^*$

Inicializácia: $i \leftarrow 1$, $k \leftarrow \sum_{i=0}^{n_U} \binom{r}{i}$;
while $i \leq k$ **do**
  Zo zoznamu $\mathcal{W}$ vyberieme i-teho kandidáta: $\mathcal{A}_i \leftarrow \mathcal{W}_i$;
  if *matica* $G_{\mathcal{A}_i}$ *má plnú hodnosť* **then**
    Vyriešime LP problém ;
    if *problém je riešiteľný* **then**
      $i \leftarrow i + 1$;
    else
      Vyriešime LP problém bez podmienky stacionarity;
      if *problém je riešiteľný* **then**
        Odstránime aktuálneho kandidáta zo zoznamu:
        $\mathcal{W} \leftarrow \mathcal{W} \setminus \mathcal{A}_i$;
        $k \leftarrow k - 1$;
      else
        Odstránime všetky kombinácie obsahujúce aktuálnu:
        $\mathcal{W} \leftarrow \mathcal{W} \setminus \{*\mathcal{A}_i*\}$;
        $k \leftarrow |\mathcal{W}|$;
      end
    end
  else
    Odstránime aktuálneho kandidáta zo zoznamu:
    $\mathcal{W} \leftarrow \mathcal{W} \setminus \mathcal{A}_i$;
    $k \leftarrow k - 1$;
  end
end
return $\mathcal{W}^*$

Algoritmus 1: Enumeračná metóda

4.4 Online implementácia explicitného prediktívneho riadenia

Potom, ako získame explicitný prediktívny regulátor offline optimalizáciou je potrebné implementovať spätnoväzbové riadenie v uzavretej slučke. [19]

Implicitné prediktívne riadenie je založené na tom, že optimalizačný problém riešime v každej perióde vzorkovania, získavame sekvencie optimálnych akčných zásahov, ale na systém aplikujeme iba prvý prvok sekvencie. Táto technika sa nazýva metóda posuvného časového horizontu a dá sa zhrnúť do týchto bodov:

1. Zmeranie (odhadnutie) aktuálnych stavov $x \in \mathbb{R}^n$
2. Vyriešenie optimalizačného problému pre x a získanie sekvencie akčných zásahov
3. Použitie prvého prvku sekvencie ako vstupu do procesu
4. Opakovanie od bodu 1

Najväčšia výpočtová záťaž je na bode 2. Aby sme zvládli vypočítať optimalizačný problém za dobu jednej periódy a opakovať tento proces stále dookola, potrebujeme výkonný solver a kvalitný hardvér. Implementácia implicitného MPC v priemysle preto vyžaduje značné finančné investície.

Keď sa chceme vyhnúť problematickej online optimalizácii, môžeme si namiesto implicitného prediktívneho riadenia zvoliť explicitné. Explicitný MPC regulátor sa navrhne offline pomocou parametrického programovania a riadenie v uzavretej slučke bude prebiehať nasledovne:

1. Zmeranie (odhadnutie) aktuálnych stavov $x \in \mathbb{R}^n$ a aktualizácia parametra θ
2. Vypočítanie optimálneho akčného zásahu μ^* vyhodnotením funkcie $\mu^*(\theta)$ získanej parametrickým programovaním
3. Aplikovanie akčného zásahu na systém
4. Opakovanie od bodu 1

Bod 2, čo je vyhodnotenie PWA funkcie a získanie akčného zásahu sa nazýva aj Point Location metóda. Je to časovo najnáročnejšia operácia v celej online implementácii. Táto metóda nemá v slovenskom jazyku svoj názov, preto budeme používať anglický.

4.4.1 Point Location metóda

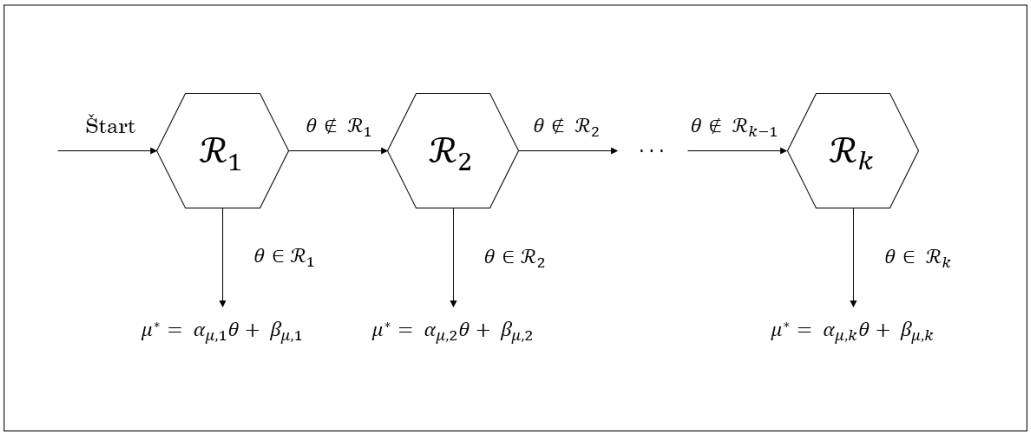
Predpokladajme, že sme vyriešili problém parametrického programovania a ako výsledok máme zákon riadenia vo forme polytopickej PWA funkcie, polytopickú partíciu Ω, ktorá je zložená z kritických regiónov $\Omega = \cup_i \mathcal{R}_i, i = 1, \ldots, M$ a účelovú funkciu $J^*(\theta)$. Aby sme mohli riadiť proces, potrebujeme tieto údaje použiť na výpočet konkrétnych hodnôt akčných zásahov.

Metóda, ktorá sa tomuto venuje sa nazýva Point Location. Jej cieľom je zobrať si riešenie z predchádzajúceho kroku a vypočítať hodnotu akčného zásahu μ^* pre aktuálne merania stavov, t.j. parameter θ. Má jednoduchý algoritmus, ktorý sa dá zhrnúť nasledovne:

1. Podľa hodnoty parametra θ nájdeme v zozname taký región $\mathcal{R}_i$, aby platilo $\theta \in \mathcal{R}_i$.

2. Zo zákona riadenia $\mu^*(\theta)$ prislúchajúcemu regiónu $\mathcal{R}_i$ vypočítame optimálny akčný zásah μ^*

   Metóda je graficky znázornená na obr. 4.2.



Obr. 4.2: Point Location problém

Časovo najnáročnejšia operácia je bod 1, pretože zvyčajne treba prehľadať veľké množstvo regiónov, aby sme našli taký, do ktorého parameter patrí. Čas spracovania teda narastá s počtom regiónov.Naproti tomu druhá operácia je z časového hľadiska zanedbateľná, pretože ako náhle sme zistili správny región, optimálny akčný zásah získame vyhodnotením afinnej funkcie:

$$\mu^* = \alpha_\mu \theta + \beta_\mu \tag{4.71}$$

čo sú jednoduché matematické operácie ako násobenie a sčítavanie dvoch matíc.

Existuje viac metód, ako nájsť správny región. Veľmi často využívaný spôsob je sekvenčné prehľadávanie, kedy prechádzame cez celý zoznam regiónov, t.j. v cykle postupne vyhodnocujeme každý región aby sme našli taký, v ktorom leží aktuálny parameter. Je zjavné, že keď budeme uvažovať každú možnú situáciu, nakoniec nájdeme región spĺňajúci dané kritéria. Algoritmus sekvenčného prehľadávania je nižšie.

Vstup: θ
Výstup: μ^*

```
for j = 1, ..., M do
    if θ ∈ R_j then
        i ← j;
        break;
    end
end
return μ* ← α_{U,i} θ + β_{U,i}
```

Algoritmus 2: Sekvenčné prehľadávanie

Kapitola 5

Balík mpc.jl

V závere sme využili teoretické princípy z predchádzajúcich kapitol a na základe nich sme vytvorili ucelený softvérový balík určený na návrh implicitných a explicitných prediktívnych regulátorov.

Vývoj balíka *mpc.jl* sa začal na jeseň 2016 a stále pokračuje. Aktuálne balík obsahuje 17 funkcií a 3 vlastné dátové typy, pričom všetky súčasti vytvárajú približne 800 riadkov Julia kódu. Pri základnej inštalácii sú užívateľovi exportované okrem dátových typov 4 funkcie. Balík *mpc.jl* je však otvorený systém, to znamená, že na jeho vývoji a zlepšovaní sa môže podieľať každý užívateľ. Nie je problém sprístupniť pôvodne skyté časti, upraviť existujúce funkcie alebo pridať úplne nové. Fáza formulácie problému a fáza jeho riešenia sú v balíku kompletne oddelené, preto je veľmi jednoducho prispôsobiteľný. V prípade potreby si užívateľ môže navrhnúť vlastnú formuláciu problému a na jej vyriešenie použiť nástroje balíka alebo naopak si môže vziať iba naformulovaný problém a vyriešiť ho pomocou svojich vlastných metód.

5.1 Stiahnutie a inštalácia

Balík *mpc.jl* je voľne dostupný na internetovom úložisku GitHub, konkrétne na https://github.com/NMikusova/mpc.jl. Patrí medzi neregistrované balíky, preto je potrebné ho naklonovať pomocou príkazu `Pkg.clone(url)`:

```
julia> Pkg.clone("git://github.com/NMikusova/mpc.jl.git")
INFO: Cloning mpc from
```

```
      git://github.com/NMikusova/mpc.jl.git
INFO: Cloning cache of MathProgBase from
      https://github.com/JuliaOpt/MathProgBase.jl.git
INFO: Cloning cache of Clp from
      https://github.com/JuliaOpt/Clp.jl.git
INFO: Cloning cache of Ipopt from
      https://github.com/JuliaOpt/Ipopt.jl.git
...
INFO: Installing: MathProgBase v0.5.8
INFO: Installing: Clp v0.3.0
INFO: Installing: Ipopt v0.2.6
...
INFO: Complete
```

Okrem vlastných funkcií využíva balík *mpc* tieto nástroje:

- *MathProgBase.jl*

  MathProgBase je optimalizačný nástroj, ktorý poskytuje jednoduché funkcie na riešenie lineárneho, kvadratického a celočíselného programovania. Optimalizačné problémy rieši pomocou externých solverov.

- *Clp.jl*

  Clp (Coin-or linear programming) je externý solver určený na lineárne programovanie napísaný v jazyku C++. Má formu knižnice, ktorú môžeme zavolať v rôznych programovacích jazykoch, vrátane Julie.[23]

- *Ipopt.jl*

  Ipopt alebo Interior Point Optimizer je softvérový balík určený na nelineárnu optimalizáciu spojitých systémov. Jeho nová verzia je napísaná v jazyku C++. Má rovnaký charakter ako *Clp*, tiež ho môžeme používať v rozličných programovacích jazykoch.[24]

- *Combinatorics.jl*

  Combinatorics je balík, ktorý sa zaoberá kombinatorikou a permutáciami.

Všetky vyššie uvedené nástroje sú voľne stiahnuteľné. Balík *mpc* obsahuje vo svojej štruktúre REQUIRE súbor, preto nie je potrebné tieto súčasti inštalovať manuálne.

5.2 Použitie

Balík začíname používať pomocou príkazu `using`, ktorý načíta všetky potrebné funkcie do pamäte.

```
using mpc
```

Balík *mpc* podporuje dve rôzne formulácie optimalizačného problému – problém regulácie a problém sledovania. Obidva optimalizačné problémy vieme vyriešiť pomocou implicitného aj explicitného prediktívneho riadenia.

5.2.1 Problém regulácie

Uvažujeme nasledovný problém regulácie:

$$\min \quad \sum_{k=0}^{N-1} x_k^\intercal Q_\mathrm{x} x_k + \sum_{k=0}^{N-1} u_k^\intercal Q_\mathrm{u} u_k \tag{5.1a}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k \tag{5.1b}$$

$$x_0 = x(t) \tag{5.1c}$$

$$x_\mathrm{min} \leq x_k \leq x_\mathrm{max} \tag{5.1d}$$

$$u_\mathrm{min} \leq u_k \leq u_\mathrm{max} \tag{5.1e}$$

$$k = 1, \ldots, N-1 \tag{5.1f}$$

V prvom kroku si uložíme všetky premenné do štruktúry zavolaním `rmodel`

```
mdl = rmodel(A,B,x0,Qx,Qu,umin,umax,xmin,xmax,N)
```

kde

- `A` a `B` sú matice lineárneho stavového modelu v diskrétnom čase
- `x0` je počiatočná podmienky pre stavy
- `Qx` a `Qu` sú váhové matice
- `umin` je dolné ohraničenie vstupov a `umax` horné ohraničenie vstupov

- `xmin` je dolné ohraničenie stavov a `xmax` horné ohraničenie stavov
- `N` je predikčný horizont

Parametre `xmin` a `xmax` sú nepovinné, to znamená, že je dovolené neuvažovať stavové ohraničenia.V takomto prípade sa `xmin` a `xmax` nastavia ako prázdne množiny:

```
mdl = rmodel(A,B,x0,Qx,Qu,umin,umax,[],[],N)
```

Implicitné prediktívne riadenie

Ak pre problém regulácie navrhujeme implicitný prediktívny regulátor, použijeme funkciu `iregulation`.

```
u,x = iregulation(mdl,t)
```

Funkcia má tieto vstupné a výstupné parametre:

- `mdl` je štruktúra z predchádzajúcej časti
- `t` je čas simulácie
- `u` je vektor alebo matica obsahujúca optimálne akčné zásahy
- `x` je vektor alebo matica stavových veličín

Funkcia `iregulation` v každej perióde skonvertuje pôvodný problém na kvadratický program a ten vyrieši pomocou funkcie `quadprog` z balíka *MathProgBase* a solvera *Ipopt*, čím získa optimálnu sekvenciu akčných zásahov. Vždy vyberie iba prvý akčný zásah a zapíše ho do premennej `u`. Následne aktualizuje stavy a zapíše ich do premennej `x`.

Explicitné MPC

Pri explicitnom prístupe používame funkciu `eregulation`.

```
u,x = eregulation(mdl,t)
```

Vstupné a výstupné parametre sú rovnaké ako predtým.

Funkcia `eregulation` najskôr preformuluje pôvodný optimalizačný problém na problém kvadratického programovania. Nový problém vyrieši pomocou multiparametrického programovania. Enumeračnou metódou získa zoznam všetkých množín aktívnych ohraničení a na základe tohto zoznamu vygeneruje polytopickú partíciu rozdelenú do kritických regiónov a zákony riadenia. Na vytváranie kombinácií ohraničení používame balík *Combinatorics* a na ich vyhodnocovanie pomocou LP problému funkciu *linprog* spadajúcu pod *MathProgBase* a lineárny solver *Clp*. Pri online implementácii sa opakovane používa metóda Point Location, ktorou pre aktuálnu hodnotu parametra vypočítame optimálny akčný zásah.

5.2.2 Problém sledovania

Problém sledovania uvažujeme v tvare:

$$\min \quad \sum_{k=0}^{N-1} (y_k - r)^\intercal Q_\mathrm{y} (y_k - r) + \sum_{k=0}^{N-1} \Delta u_k^\intercal Q_\mathrm{u} \Delta u_k \tag{5.2a}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k \tag{5.2b}$$

$$y_k = Cx_k + Du_k \tag{5.2c}$$

$$\Delta u_k = u_k - u_{k-1} \tag{5.2d}$$

$$x_0 = x(t) \tag{5.2e}$$

$$u_{-1} = u(t-1) \tag{5.2f}$$

$$u_{\min} \leq u_k \leq u_{\max} \tag{5.2g}$$

$$\Delta u_{\min} \leq \Delta u_k \leq \Delta u_{\max} \tag{5.2h}$$

$$y_{\min} \leq y_k \leq y_{\max} \tag{5.2i}$$

$$k = 1, \ldots, N-1 \tag{5.2j}$$

Začneme rovnako ako pri probléme regulácie a uložíme si všetky premenné do štruktúry.

```
mdl = tmodel(A,B,C,D,x0,um1,Qy,Qu,ref,umin,umax,dumin,dumax,
    ymin,ymax,N)
```

- `A`, `B`, `C` a `D` sú matice lineárneho stavového modelu v diskrétnom čase

- `x0` je počiatočná podmienky pre stavy
- `um1` je počiatočná podmienky pre vstupy
- `Qy` a `Qu` sú váhové matice
- `ref` je referencia, ktorú bude sledovať výstup
- `umin` je dolné ohraničenie vstupov a `umax` horné ohraničenie vstupov
- `dumin` je dolné ohraničenie inkrementov vstupov a `dumax` horné ohraničenie inkrementov vstupov
- `ymin` je dolné ohraničenie výstupov a `ymax` horné ohraničenie výstupov
- `N` je predikčný horizont

Nepovinné sú parametre `dumin`, `dumax`, `ymin` a `ymax`.

Implicitné prediktívne riadenie

Keď chceme riadiť výstup na referenciu pomocou implicitného prediktívneho riadenia, použijeme funkciu `itracking`.

```
u,y,x = itracking(mdl,t)
```

Funkcia má tieto vstupné a výstupné parametre:

- `mdl` je štruktúra so vstupnými údajmi
- `t` je čas simulácie
- `u` je vektor alebo matica obsahujúca optimálne akčné zásahy
- `y` je vektor alebo matica výstupných veličín
- `x` je vektor alebo matica stavových veličín

Funkcia `itracking` pracuje rovnako ako funkcia `iregulation`. Jediný rozdiel je v tom, že pri probléme sledovania vytvárame ďalšiu premennú `y` obsahujúcu optimálne výstupy.

Explicitné MPC

Keď navrhujeme explicitný prediktívny regulátor pre problém sledovania, používame funkciu `etracking`.

```
u,y,x = etracking(mdl,t)
```

Vstupné a výstupné parametre sú rovnaké ako v predošlej časti a zároveň metódy a postupy sú totožné s tými, ktoré používa funkcia `eregulation`.

5.3 Testovanie

Posledným krokom tejto práce bolo otestovať balík *mpc*. Testovali sme správanie sa jednotlivých funkcií balíka ako aj jeho celkový výkon. Vyhodnocovali sme také kvalitatívne ukazovatele ako sú časová náročnosť alebo numerická presnosť. Aby sme vedeli výsledky testovania objektívne posúdiť, porovnávali sme Juliu s programovacím jazykom Matlab. Na testovanie sme použili nasledovný príklad:

```
A = [-0.0490 -0.3278;0.2622 0.7376];
B = [0.2622; 0.2099];
Qx = eye(2,2);
Qu = 0.1;
N = 5;
x0 = [-1;3];
umin = -1.5;
umax = 0.5;
t = 20
```

Uvažujeme lineárny stavový model v diskrétnom čase, ktorý opisujú matice A a B. Máme dve stavové veličiny a jednu vstupnú veličinu. Väčšiu váhu priraďujeme stavovým veličinám. Ohraničujeme iba vstupné veličiny. Naším cieľom je navrhnúť taký prediktívny regulátor, ktorý bude regulovať systém do počiatku.

Najskôr sme navrhovali implicitný prediktívny regulátor. Problém regulácie sme vyriešili 9-krát. 3-krát v Julii pomocou funkcie `iregulation` z balíka *mpc* a 6-krát v Matlabe. V Matlabe sme použili dva nástroje – modelovací jazyk YALMIP a funkciu `quadprog`, ktorá je veľmi podobná funkcii

rovnakého názvu v Julii. Pri každom opakovaní sme zmerali celkový čas potrebný na návrh regulátora. Potom sme z jednotlivých časov vypočítali priemerný čas. Časová náročnosť implicitného MPC je znázornená v tabuľke 5.1.

Implicitné MPC			
	Julia	Matlab: YALMIP	Matlab: quadprog
t_1	0.17s	5.47s	0.29s
t_2	0.12s	5.48s	0.30s
t_3	0.15s	5.43s	0.28s
$\bar{t}$	0.15s	5.45s	0.29s

Tabuľka 5.1: Časová náročnosť implicitného prediktívneho riadenia

Z tabuľky 5.1 si môžeme všimnúť, že návrh regulátora trval najkratšie v Julii, avšak funkcia `quadprog` dosahovala porovnateľný výkon. Nevýhoda pri použití funkcie `quadprog` spočíva v tom, že užívateľ si sám musí vygenerovať matice kvadratického programu, pretože táto funkcia slúži iba ako solver. Funkcia `eregulation` v Julii naproti tomu v sebe obsahuje obidva tieto kroky. Návrh regulátora cez YALMIP trval zvyčajne niekoľko sekúnd, napriek tomu sa však tento jazyk teší veľkej popularite. Tým, že YALMIP je modelovací jazyk, užívateľ nepotrebuje vyrábať žiadne matice. Problém zadá v jeho prirodzenom tvare a YALMIP sám ho preloží do štandardného tvaru a automaticky pošle na vyriešenie príslušnému solveru. [25]

Ďalej sme v Julii testovali vplyv predikčného horizontu na časovú náročnosť. Výsledky sú uvedené v tabuľke 5.2. Môžeme pozorovať, že v prípade malého modelu, aký uvažujeme my nehrá predikčný horizont veľkú úlohu. Balík *mpc* zvládol pri ktoromkoľvek predikčnom horizonte navrhnúť prediktívny regulátor do jednej sekundy. Ak by sme uvažovali veľký systém s množstvom ohraničení, predikčný horizont by zásadne začal vplývať na časovú náročnosť.

Implicitné MPC				
	N = 5	N = 10	N = 20	N = 50
t_1	0.166s	0.162s	0.199s	0.474s
t_2	0.123s	0.236s	0.302s	0.537s
t_3	0.157s	0.258s	0.287s	0.513s
$\bar{t}$	0.148s	0.218s	0.262s	0.508s

Tabuľka 5.2: Vplyv predikčného horizontu na časovú náročnosť

Následne sme porovnávali numerickú presnosť v Julii a Matlabe. Optimálne hodnoty vybraných akčných zásahov sú v tabuľke 5.3. Ukazuje sa, že vo väčšine prípadoch sa výsledky z Julie a Matlabu zhodujú na približne 7 desatinných miest.

Implicitné MPC		
	Julia	Matlab
u_1	$-1.4\bar{9}$	-1.50
u_2	-0.790408723	-0.790408718
u_3	-0.293318715	-0.293318720

Tabuľka 5.3: Optimálne hodnoty vybraných akčných zásahov

Druhá fáza testovania bola venovaná explicitnému prediktívnemu riadeniu. V Julii sme problém riešili pomocou funkcie `eregulation` balíka *mpc* a v Matlabe pomocou *mpt toolboxu*, čo je voľne stiahnuteľný balík nástrojov na parametrickú optimalizáciu a prediktívne riadenie. [26]

Explicitné MPC		
	Julia	Matlab: *mpt toolbox*
t_1	1.01s	1.12s
t_2	0.98s	1.17s
t_3	0.99s	1.14s
$\bar{t}$	0.99s	1.14s
počet regiónov	33	30

Tabuľka 5.4: Časová náročnosť explicitného prediktívneho riadenia

Pretože *mpt toolbox* sa pri riešení multiparametrického programovania odlišuje v niektorých postupoch a metódach, štruktúra polytopickej partície a počet regiónov sa môže líšiť. Je to vidno aj z tabuľky 5.4. V Julii obsahuje riešenie nášho príkladu 33 regiónov a v Matlabe iba 30. V prípade, že použijeme kratší predikčný horizont, časová náročnosť v oboch jazykoch je porovnateľná. Avšak, keď v Julii začneme zvyšovať predikčný horizont, časová náročnosť exponenciálne stúpa. Uveďme konkrétny príklad. Ako je vidno na príklade vyššie, momentálne uvažujeme predikčný horizont N = 5 a ohraničujeme iba vstupy systému. To znamená že celkový počet ohraničení je rovný 10 a my musíme dohromady preskúmať vyše tisíc rôznych kombinácii. V tabuľke 5.4 vidíme, že nás to vedie k zostrojeniu 33 regiónov. Keby sme predikčný horizont zväčšili dvojnásobne, počet ohraničení sa tiež zvýši dvojnásobne, ale vyhodnotiť musíme viac ako milión kombinácií. Výsledkom bude zhruba 170 regiónov. Zatiaľ čo *mpt toolbox* si zachováva svoju rýchlosť a riešenie nám poskytne v priebehu sekúnd, Julia na vyhodnotenie potrebuje zopár minút. Je to spôsobené najmä tým, že *mpt toolbox* sa aktívne vyvíja už niekoľko rokov a jeho výkon sa dlhodobo optimalizuje. Balík *mpc* pri explicitnom prediktívnom riadení zatiaľ využíva iba základné techniky, preto je v súčasnej dobe vhodný najmä pre malé systémy a malé predikčné horizonty.

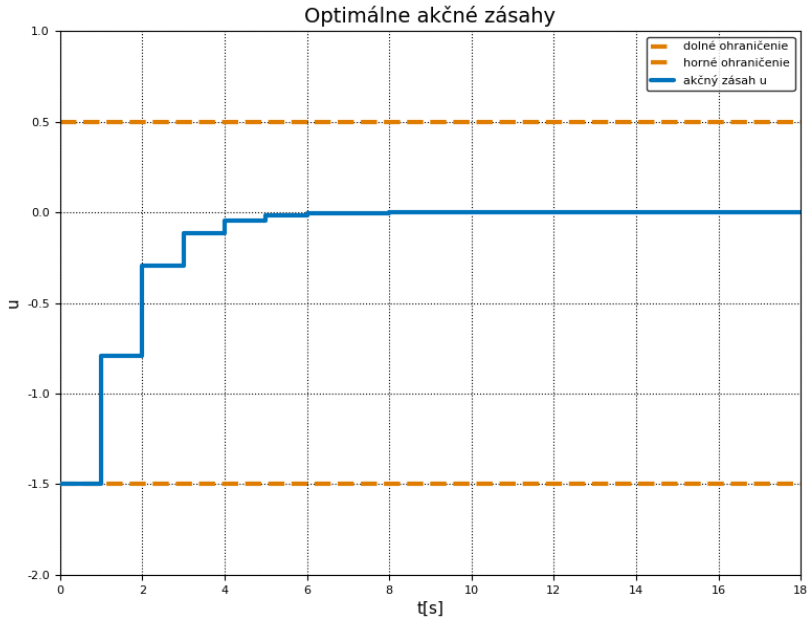
V explicitnom prístupe bolo zaujímave zisťovať časovú náročnosť jednotlivých operácií. Z hľadiska času sme vyhodnotili všetky najdôležitejšie fázy pri návrhu explicitného regulátora a výsledky uviedli v tabuľke 5.5. Potvrdilo sa nám, že najdlhšia operácia je offline optimalizácia, kedy hľadáme riešenie multiparametrického programovania. Zabrala viac ako 96% celko-

vého času. Ostatné operácie sú oproti tejto zanedbateľné. Presvedčili sme sa o tom, že explicitný prístup naozaj výrazne redukuje časovú náročnosť online implementácie.

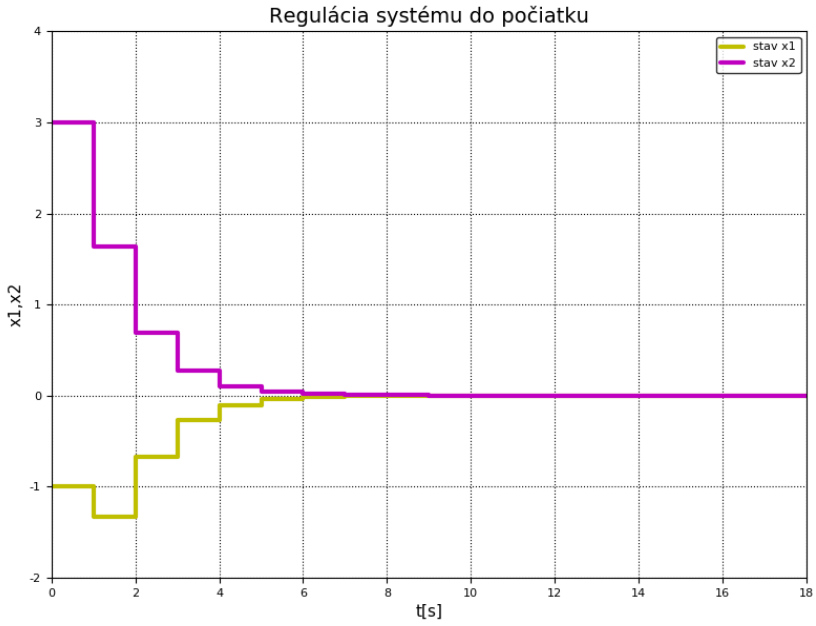
Explicitné MPC					
	t_1	t_2	t_3	$\bar{t}$	% z $\bar{t}$
Vytvorenie modelu	0.00033s	0.00046s	0.00045s	0.00041s	0.04
Formulácia kvadratického programu	0.003s	0.004s	0.006s	0.004s	0.43
Enumeračná metóda	0.85s	0.89s	0.93s	0.89s	96.6
Point Location metóda	0.033s	0.026s	0.022s	0.027s	2.90
Celkový čas	0.886s	0.920s	0.958s	0.921s	100

Tabuľka 5.5: Časová náročnosť jednotlivých operácií v explicitnom MPC

Na záver je dobré spomenúť, že pri oboch prístupoch sme získali rovnaký výsledok s rovnakou numerickou presnosťou t.j. rovnaké optimálne akčné zásahy. Riadenie systému sme znázornili aj graficky pomocou balíka *Plots*.



Obr. 5.1: Optimálne akčné zásahy



Obr. 5.2: Riadenie prediktívnym regulátorom

Záver

Cieľom tejto diplomovej práce bolo vyvinúť softvérový balík určený na prediktívne riadenie v jazyku Julia.

Na začiatku sme zhrnuli všetky najvýznamnejšie vlastnosti a charakteristiky tohto jazyka a potom sme uviedli krátky prehľad dostupných optimalizačných nástrojov. Bolo to najmä z toho dôvodu, že prediktívne riadenie z veľkej časti tvorí práve optimalizácia a tiež kvôli tomu, že niektoré z týchto nástrojov používame v *mpc* balíku.

V hlavnej časti práce sme sa zameriavali na teoretický návrh implicitných a explicitných prediktívnych regulátorov. Uviedli sme dve najčastejšie formulácie optimalizačného problému – problém regulácie a problém sledovania – a ukázali postup, ktorým ich vieme prepísať na problémy kvadratického programovania v štandardnom tvare. Ďalej sme diskutovali samotné riešenie týchto problémov. Pri implicitnom MPC sme vysvetlili metódu posuvného časového horizontu, kedy na kvadratický program opakovane aplikujeme niektorý kvadratický solver, čím získavame optimálne sekvencie akčných zásahov. Explicitné prediktívne riadenie sme riešili pomocou parametrického programovania, ktoré je typické tým, že vezme do úvahy všetky počiatočné podmienky a vygeneruje riešenie pozostávajúce s polytopickej partície, zákonov riadenia a účelovej funkcie. Hlavnou metódou tejto časti, offline optimalizácie, je enumeračná metóda hľadajúca množiny aktívnych ohraničení. Uviedli sme tiež, ako treba použiť toto riešenie na získanie optimálnych akčných zásahov pomocou Point Location metódy.

Nakoniec sme všetky poznatky zužitkovali pri tvorbe balíka *mpc*. Tento balík obsahuje 4 užívateľské funkcie, ktorými vieme navrhnúť implicitný aj explicitný regulátor pre problém regulácie a sledovania. Balík sme otestovali a porovnali ho s podobne ladenými balíkmi Matlabu. Nakoniec sme ho uverejnili na internetovom úložisku GitHub.

Literatúra

[1] Wikipedia. Julia (programming language). Dostupné na internete: https://en.wikipedia.org/wiki/Julia_(programming_language). Posledný prístup: máj, 2017.

[2] Julia. Julia. Dostupné na internete: http://julialang.org/. Posledný prístup: máj, 2017.

[3] Julia. Downloads. Dostupné na internete: https://julialang.org/downloads/. Posledný prístup: máj, 2017.

[4] Juno. Getting Juno. Dostupné na internete: https://github.com/JunoLab/uber-juno/blob/master/setup.md. Posledný prístup: máj, 2017.

[5] Julia. Julia Documentation. Dostupné na internete: https://docs.julialang.org/en/stable/. Posledný prístup: máj, 2017.

[6] Julia. Packages. Dostupné na internete: http://docs.julialang.org/en/release-0.4/manual/packages/. Posledný prístup: máj, 2017.

[7] Julia. Plotting in Julia. Dostupné na internete: https://julialang.org/downloads/plotting.html. Posledný prístup: máj, 2017.

[8] Wikipedia. Mathematical optimization. Dostupné na internete: https://en.wikipedia.org/wiki/Mathematical_optimization. Posledný prístup: máj, 2017.

[9] Boyd, S. a Vandenberghe, L. *Convex Optimization*. Cambridge University Press, New York, USA, 7th vydanie, 2009.

[10] JuliaOpt. What is JuliaOpt? Dostupné na internete: http://www.juliaopt.org/. Posledný prístup: máj, 2017.

[11] Wikipedia. Linear programming. Dostupné na internete: https://en.wikipedia.org/wiki/Linear_programming. Posledný prístup: máj, 2017.

[12] MathProgBase. Linear Programming. Dostupné na internete: http://mathprogbasejl.readthedocs.io/en/latest/linprog.html. Posledný prístup: máj, 2017.

[13] Wikipedia. Quadratic programming. Dostupné na internete: https://en.wikipedia.org/wiki/Quadratic_programming. Posledný prístup: máj, 2017.

[14] Klaučo, M. Predictive Control of Complex Systems. (2015).

[15] Advanced Optimization: on-line course. Dostupné na internete: http://www.kirp.chtf.stuba.sk/moodle/pluginfile.php/60753/mod_resource/content/1/opt2_09.pdf. Posledný prístup: máj 2017.

[16] MathProgBase. Quadratic Programming. Dostupné na internete: http://mathprogbasejl.readthedocs.io/en/latest/quadprog.html. Posledný prístup: máj, 2017.

[17] Wikipedia. Model predictive control. Dostupné na internete: https://en.wikipedia.org/wiki/Model_predictive_control. Posledný prístup: máj, 2017.

[18] Prediktívne riadenie: on-line kurz pre FCHPT. Dostupné na internete: http://www.kirp.chtf.stuba.sk/moodle/course/view.php?id=309. Posledný prístup: máj, 2015.

[19] Holaza, J. *Fast and Memory-Efficient implementation of Model Predictive Control*. Doktorská práca, ÚIAM FCHPT STU v Bratislave, Radlinského 9, 812 37 Bratislava, 24.08.2016 2016.

[20] van den Boom, T., Backx, T., TU Delft, F. o. M. E. a (3mE), M. T. *Model Predictive Control: College SC4060; Course Notes*. TU Delft, 2007.

[21] Wikipedia. Active set method. Dostupné na internete: https://en.wikipedia.org/wiki/Active_set_method. Posledný prístup: máj, 2017.

[22] Wikipedia. Enumeration. Dostupné na internete: https://en.wikipedia.org/wiki/Enumeration. Posledný prístup: máj, 2017.

[23] COIN-OR. Clp. Dostupné na internete: https://projects.coin-or.org/Clp. Posledný prístup: máj, 2017.

[24] COIN-OR. Ipopt. Dostupné na internete: https://projects.coin-or.org/Ipopt. Posledný prístup: máj, 2017.

[25] Löfberg, J. YALMIP : A toolbox for modeling and optimization in MATLAB. *CCA/ISIC/CACSD*. IX 2004.

[26] Herceg, M., Kvasnica, M., Jones, C. a Morari, M. Multi-Parametric Toolbox 3.0. *2013 European Control Conference*, 502–510. 2013.