

# Gergely Takács, Martin Gulan

## ZÁKLADY

# PREDIKTÍVNEHO RIADENIA





Gergely Takács, Martin Gulan

# ZÁKLADY PREDIKTÍVNEHO RIADENIA

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
2018

Učebnica predstavuje základnú terminológiu, koncepty a techniky v prediktívnom riadení. Je určená predovšetkým poslucháčom technických odborov s hlbším zameraním na teóriu riadenia, ako sú napríklad špecializácie zamerané na automatizáciu, technickú kybernetiku alebo mechatroniku. Učebnica predstavuje modernú metódu optimálneho spätnoväzbového stavového riadenia známu pod názvom prediktívne riadenie (MPC). Po prehľade terminológie a motivácie učebnica stručne zhrnie princípy tvorby matematických modelov dynamických sústav a odhadu stavu. Ďalej pokračuje odvodením predikčného modelu a účelovej funkcie problému MPC. Potom je predstavený proces hľadania optimálneho akčného zásahu pri zahrnutí procesných obmedzení do formulácie príslušnej optimalizačnej úlohy. Samostatná kapitola je tiež venovaná priblíženiu riešenia problému kvadratického programovania. Po prevedení ukážkových úloh spätnoväzbového riadenia sa záverečná kapitola dotkne problematiky zaručenia stability MPC. Učebnica je pre úplnosť didaktického prístupu bohatou graficky ilustrovaná a doplnená príkladmi a počítačovými simuláciami v prostredí MATLAB a Simulink.

Všetky práva vyhradené. Nijaká časť textu nesmie byť použitá na ďalšie šírenie akoukoľvek formou bez predchádzajúceho súhlasu autorov alebo vydavateľstva.

© doc. Ing. Gergely Takács, PhD., Ing. Martin Gulan

Recenzenti: prof. Ing. Miroslav Fikar, DrSc.

prof. Ing. Štefan Kozák, PhD.

prof. Ing. Boris Rohal-Ilkiv, CSc.

Obálku navrhol: doc. Ing. Gergely Takács, PhD.

Schválila Vedecká rada Strojníckej fakulty STU v Bratislave.

ISBN 978-80-227-4826-1

# Obsah

<b>Obsah</b>	<b>8</b>
<b>Zoznam obrázkov</b>	<b>16</b>
<b>Zoznam tabuliek</b>	<b>17</b>
<b>Finančná podpora</b>	<b>27</b>
<b>Predslov</b>	<b>29</b>
<b>O autoroch</b>	<b>31</b>
<b>Programová podpora</b>	<b>33</b>
<b>1 Úvod do prediktívneho riadenia</b>	<b>35</b>
1.1 Postačujú vždy tradičné metódy riadenia? . . . . .	35
1.2 Základná myšlienka prediktívneho riadenia . . . . .	38
1.2.1 Analógie prediktívneho riadenia . . . . .	39
1.2.2 Klúčové pojmy a koncepty . . . . .	42
1.3 Historický prehľad . . . . .	48
1.3.1 DMC a MAC: riadenie na základe konečnej odozvy . . . . .	49
1.3.2 Zovšeobecnené prediktívne riadenie . . . . .	50
1.3.3 Riadenie s pohyblivým horizontom . . . . .	51
1.3.4 Zjednotenie prístupov prediktívneho riadenia . . . . .	52
1.4 Využitie prediktívneho riadenia . . . . .	53
1.5 Vybrané príklady využitia MPC . . . . .	54
1.5.1 Priemyselné aplikácie . . . . .	54
1.5.2 Akademické aplikácie . . . . .	57
1.6 Budúcnosť prediktívneho riadenia . . . . .	61
1.6.1 Nelineárne MPC . . . . .	61

1.6.2	MPC pre rýchlu dynamiku . . . . .	62
1.6.3	Explicitné MPC . . . . .	63
1.6.4	Vnorené MPC . . . . .	64
1.6.5	Stochastické a robustné MPC . . . . .	65
1.6.6	Ekonomické MPC . . . . .	66
1.6.7	Rozložené MPC . . . . .	66
<b>2</b>	<b>Modelovanie a odhad stavu</b>	<b>69</b>
2.1	Pohybové rovnice a nelineárny stavový model . . . . .	72
2.2	Linearizácia a lineárny stavový model . . . . .	83
2.3	Diskretizácia a diskrétny stavový model . . . . .	100
2.4	Odhad stavu a Kalmanov filter . . . . .	118
2.5	Rozšírený Kalmanov filter . . . . .	138
<b>3</b>	<b>Predikcia</b>	<b>147</b>
3.1	O krok dopredu . . . . .	148
3.2	Predikcia do konca horizontu . . . . .	149
3.3	Autonómny model . . . . .	152
3.4	Predikcia vynútenej odozvy . . . . .	160
3.5	Predikčné matice . . . . .	166
3.6	Koncept pohyblivého horizontu . . . . .	170
3.7	Predikcia daného stavu v budúcnosti . . . . .	170
3.8	Predikcia výstupov . . . . .	172
<b>4</b>	<b>Účelová funkcia</b>	<b>177</b>
4.1	Aká je ideálna účelová funkcia? . . . . .	177
4.1.1	Optimalizačná úloha . . . . .	178
4.1.2	Konvexnosť . . . . .	179
4.1.3	Obmedzenia . . . . .	180
4.1.4	Kvadratické účelové funkcie . . . . .	181
4.1.5	Minimalizácia kvadratickej funkcie . . . . .	184
4.2	Účelová funkcia na konečnom horizonte . . . . .	187
4.2.1	Hodnota účelovej funkcie v budúcnosti . . . . .	188
4.2.2	Odchýlka riadenia ako indikátor kvality . . . . .	188
4.2.3	Stavy v účelovej funkcií . . . . .	190
4.2.4	Vstupy v účelovej funkcií . . . . .	192
4.2.5	Výsledná formulácia funkcie . . . . .	193
4.3	Kompaktný zápis účelovej funkcie . . . . .	195
4.4	Zložená účelová funkcia . . . . .	202
4.4.1	Matica váhovania koncového stavu . . . . .	206

4.4.2	Ljapunova rovnica . . . . .	208
4.4.3	Váhovanie podľa diskrétnej algebraickej Riccatiho rovnice .	210
4.4.4	Hľadanie koncového váhovania pri známom LQ zosilnení .	214
4.5	Zložená účelová funkcia v kompaktnom tvarе . . . . .	220
<b>5</b>	<b>Prediktívne riadenie bez obmedzení</b>	<b>223</b>
5.1	Prediktívne riadenie bez obmedzení . . . . .	224
5.1.1	Vplyv posledného výrazu . . . . .	224
5.1.2	Riešenie problému MPC bez obmedzení . . . . .	228
5.1.3	Riadenie v spätej väzbe . . . . .	230
5.2	Vlastnosti MPC bez obmedzení . . . . .	241
5.2.1	Vzťah LQ a MPC bez obmedzení . . . . .	241
5.2.2	Vplyv dĺžky predikčného horizontu . . . . .	247
5.3	Formulácia obmedzení . . . . .	255
5.3.1	Saturácia vstupov . . . . .	256
5.3.2	Obmedzenia vstupov . . . . .	260
5.3.3	Obmedzenia stavov . . . . .	265
5.3.4	Kombinácia rôznych typov obmedzení . . . . .	270
5.3.5	Iné typy obmedzení . . . . .	273
5.4	Problém prediktívneho riadenia s obmedzeniami . . . . .	274
5.4.1	Riešenie problému MPC s obmedzeniami . . . . .	276
<b>6</b>	<b>Kvadratické programovanie</b>	<b>277</b>
6.1	Problém kvadratického programovania . . . . .	278
6.2	Vrstevnice, gradient a ohraničenia . . . . .	281
6.2.1	Vrstevnice funkcie . . . . .	281
6.2.2	Gradienty a vrstevnice . . . . .	287
6.2.3	Obmedzenia . . . . .	290
6.3	Kvadratické programovanie v MATLABe . . . . .	296
6.3.1	Diagnostika riešiča . . . . .	299
6.4	Metóda aktívnych množín . . . . .	303
6.4.1	Test optimality: Lagrangeove multiplikátory . . . . .	307
6.4.2	Zovšeobecnenie Lagrangeových multiplikátorov . . . . .	312
6.4.3	KKT podmienky pre QP . . . . .	315
6.4.4	Iteratívne zlepšenie riešenia . . . . .	317
6.4.5	QP algoritmus na báze aktívnych množín . . . . .	320
6.5	Metóda vnútorného bodu . . . . .	325

<b>7 Prediktívne riadenie s obmedzeniami</b>	<b>333</b>
7.1 Prediktívne riadenie a kvadratické programovanie . . . . .	334
7.2 Výpočet optimálneho vektora vstupov . . . . .	335
7.3 MPC v uzavretej slučke . . . . .	339
7.4 Sledovanie referenčnej trajektórie . . . . .	343
7.4.1 Referenčné vstupy . . . . .	344
7.4.2 Integračná zložka . . . . .	348
7.5 Prírastky vstupov a ich obmedzenie . . . . .	355
7.6 MPC v prostredí Simulink . . . . .	361
<b>8 Stabilita a riešiteľnosť MPC</b>	<b>383</b>
8.1 Vývoj MPC so zaručenou stabilitou . . . . .	384
8.1.1 Koncové obmedzenia v tvare rovnosti . . . . .	385
8.1.2 Váhovanie koncového stavu . . . . .	386
8.1.3 Cieľové množiny . . . . .	386
8.1.4 Kombinácia cieľovej množiny a váhovania koncového stavu .	387
8.2 Stabilita MPC s nekonečným horizontom . . . . .	388
8.3 Stabilita pomocou koncových obmedzení . . . . .	395
8.4 Maximálna invariantná cieľová množina . . . . .	398
8.5 Problém MPC so zaručenou stabilitou . . . . .	407
<b>Literatúra</b>	<b>417</b>
<b>Zdroj vybraných obrázkov</b>	<b>431</b>
<b>Register</b>	<b>433</b>

# Zoznam obrázkov

1.1	Ilustrácia rozdielu medzi PID a LQ riadením elektromechanického systému „gulôčka na ploche“ . . . . .	36
1.2	Ilustrácia procesných obmedzení. . . . .	37
1.3	Prediktívne riadenie ako analógia šachu. . . . .	39
1.4	Prediktívne riadenie ako analógia šoférovania motorového vozidla. . . . .	40
1.5	Stratégia šoférovania motorového vozidla pomocou PID riadenia. . . . .	41
1.6	Matematický model systému. . . . .	43
1.7	Predikcia (simulácia) správania sa dynamického systému. . . . .	43
1.8	Pohyblivý horizont predikcie v diskrétnom čase. . . . .	44
1.9	Kriteriálna funkcia zhrnie viaceru veličín vplývajúcich na riadenie do jedného skalárneho indikátora kvality. . . . .	44
1.10	Kriteriálna funkcia ako indikátor kvality v zmysle dosiahnutia cieľov riadenia. . . . .	44
1.11	Optimálne akčné zásahy prinášajú najlepšiu možnú kvalitu riadenia pri definovaných podmienkach. . . . .	45
1.12	Obmedzenia a hľadanie optimálnej kvality riadenia. . . . .	46
1.13	Ilustrácia prístavného žeriavu LIEBHERR LHM550 a technológie „Cycoptronic“ na optimálne plánovanie trasy na báze prediktívneho riadenia. . . . .	54
1.14	Ilustrácia schémy energetického riadenia budovy pomocou MPC. . . . .	55
1.15	Ilustrácia schémy adaptívneho tempomatu na báze MPC. . . . .	56
1.16	Ilustrácia závodnej trate pre modely autíčiek v rámci projektu ORCA a porušenie obmedzení v prípade použitia autonómneho riadenia pomocou PI(D) algoritmu. . . . .	58
1.17	Ilustrácia laboratórneho systému „Gulôčka na ploche“. . . . .	59
1.18	Charakter riadenia a dynamické správanie sa pristávajúcej rakety sa podobajú známemu experimentu „gulôčka na ploche“. . . . .	60
1.19	Ilustrácia výkonu porovnatelne naladeného LQ a MPC riadenia systému „gulôčka na ploche“. . . . .	60

1.20	Riadenie musí prebiehať v reálnom čase, pričom akčný zásah musí byť k dispozícii ešte pred koncom daného vzorkovacieho intervalu.	63
1.21	Mikroradič (STML100) v porovnaní s veľkosťou 1 € mince.	65
1.22	Sústredené (centralizované) vs. rozložené (decentralizované) MPC.	67
2.1	Laboratórny model invertovaného kyvadla na vozíku.	72
2.2	Zjednodušená schéma invertovaného kyvadla.	73
2.3	Bloková schéma pre numerickú simuláciu nelineárneho dynamickejho modelu invertovaného kyvadla v prostredí Simulink.	82
2.4	Simulácia volnej a vynútenej odozvy nelineárneho modelu invertovaného kyvadla v prostredí MATLAB a Simulink.	84
2.5	Ilustrácie linearizácie jednoduchej nelineárnej funkcie.	85
2.6	Bloková schéma spojitého LTI stavového modelu.	90
2.7	Simulácia volnej odozvy linearizovaného modelu invertovaného kyvadla.	96
2.8	Porovnanie volnej odozvy získanej simuláciou nelineárneho a linearizovaného modelu invertovaného kyvadla.	97
2.9	Dva ekvivalentné spôsoby interpretácie a simulácie linearizovaného modelu invertovaného kyvadla v prostredí Simulink.	98
2.10	Diskretizácia spojitého systému pomocou ZOH ekvivalentu.	101
2.11	Bloková schéma diskrétneho LTI stavového modelu.	104
2.12	Simulácia vynútenej odozvy diskrétneho linearizovaného modelu invertovaného kyvadla.	108
2.13	Porovnanie vynútenej odozvy diskrétneho modelu invertovaného kyvadla získaného exaktnou diskretizáciou a approximáciou pomocou doprednej Eulerovej metódy.	108
2.14	Simulácia volnej odozvy diskrétneho linearizovaného modelu invertovaného kyvadla s uvažovaním rôznych veľkostí vzorkovacej períody.	109
2.15	Dva ekvivalentné spôsoby interpretácie a simulácie diskrétneho lineárneho modelu invertovaného kyvadla v prostredí Simulink.	110
2.16	Analýza stability pomocou znázornenia vlastných čísel v komplexnej rovine.	116
2.17	Bloková schéma Luenbergerovho pozorovateľa stavu.	119
2.18	Stavové spätnoväzbové riadenie s deterministickým odhadom stavu.	120
2.19	Ilustrácia korekčného a predikčného kroku v rámci jednej iterácie algoritmu Kalmanovho filtra.	122
2.20	Blokový diagram rekurzívneho algoritmu Kalmanovho filtra.	123
2.21	Stavové spätnoväzbové riadenie s optimálnym odhadom stavu pomocou Kalmanovho filtra.	127
2.22	Rudolf E. Kálmán.	128

2.23	Simulácia voľnej odozvy diskrétneho stavového modelu invertovaného kyvadla s odhadom jeho stavu pomocou Kalmanovho filtra. . . . .	135
2.24	Porovnanie odhadu uhlovej výchylky invertovaného kyvadla s rôznoou voľbou kovariančnej matice chyby odhadu. . . . .	135
2.25	Bloková schéma ilustrujúca odhad stavu invertovaného kyvadla pomocou Kalmanovho filtra v prostredí Simulink. . . . .	137
2.26	Podsystém Simulink schémy na Obr. 2.25 implementujúci nelineárny stavový model invertovaného kyvadla. . . . .	137
2.27	Bloková schéma ilustrujúca odhad stavu invertovaného kyvadla pomocou rozšíreného Kalmanovho filtra v prostredí Simulink. . . . .	142
2.28	Simulácia vynútenej odozvy nelineárneho stavového modelu invertovaného kyvadla s odhadom jeho stavu pomocou rozšíreného Kalmanovho filtra. . . . .	143
2.29	Experimentálne výsledky odhadu stavu laboratórneho modelu invertovaného kyvadla pomocou rozšíreného Kalmanovho filtra. . . . .	144
3.1	Veštenie nie je jediný spôsob ako predpovedať budúcnosť. . . . .	148
3.2	Ak poznáme vstup a súčasný stav, na základe modelu vieme vypočítať nasledujúci stav. . . . .	149
3.3	Diskrétny stavový model predstavuje jednokrokovú predikčnú rovniciu. . . . .	149
3.4	Pri spojiteľnom modeli predstavuje stavová rovnica istú analógiu predikcie, pretože derivácia predstavuje smernicu dotyčnice v danom čase. . . . .	150
3.5	Opakovaným (rekurzívnym) dosadzovaním stavov do predikčnej rovnice môžeme predikovať nasledujúce (budúce) stavy. . . . .	150
3.6	Predikcia stavu $n_p$ krokov do budúcnosti. . . . .	151
3.7	Predikovaná voľná odozva autonómneho systému. . . . .	154
3.8	Porovnanie predikcie výchylky voľného konca votknutého nosníka s jej experimentálnym meraním. . . . .	160
3.9	Vytvorenie predikčnej matice $\mathbf{N}$ posunutím blokových stĺpcov smerom dole. . . . .	162
3.10	Porovnanie výsledkov predikcie s lineárnom simuláciou. . . . .	165
3.11	Lietadlo Boeing 777-300ER. . . . .	167
3.12	Predikcia trajektórie odozvy Boeing 777 (o) a skutočné stavy (x). .	169
3.13	Idea pohyblivého horizontu predikcie. . . . .	170
3.14	Lubovoľný predikovaný stav dvojitého integrátora získaný pomocou príslušných blokových riadkov predikčných matíc. . . . .	173
3.15	Násobením transformačnou maticou $\mathbf{L}$ získame predikované výstupy. .	174

4.1	Kriteriálna funkcia prediktívneho riadenia vyjadruje „cenu“, resp. kvalitu riadenia. . . . .	178
4.2	Hľadanie maxima skalárnej funkcie môžeme chápať ako hľadanie minima tej istej funkcie so záporným znamienkom. . . . .	178
4.3	Niekedy nás namiesto funkčnej hodnoty zaujíma hodnota argumentu funkcie v jej minime. . . . .	179
4.4	Nekonvexná funkcia s lokálnym a globálnym extrémom a konvexná funkcia iba s globálnym extrémom. . . . .	180
4.5	Nekonvexná funkcia v 3D: logo MATLABu. . . . .	181
4.6	Optimalizácia bez obmedzení a optimalizácia s obmedzeniami. . .	181
4.7	Kvadratická funkcia skalárnej nezávislej premennej. . . . .	182
4.8	Skalárna kvadratická funkcia závislá od dvoch premenných. . . .	184
4.9	Minimum skalárnej kvadratickej funkcie. . . . .	186
4.10	Minimum kvadratickej funkcie v 3D. . . . .	188
4.11	Odchýlku riadenia nemôžeme priamo používať ako indikátor kvality.	189
4.12	Indexovanie vstupov a stavov v účelovej funkcií. . . . .	194
4.13	Ilustrácia kriteriálnej funkcie na konečnom horizonte <sup>1</sup> . . . . .	203
4.14	Ilustrácia kriteriálnej funkcie na nekonečnom horizonte. . . . .	204
4.15	Jacopo Francesco Riccati. . . . .	207
4.16	Alexander Michajlovič Ljapunov (1857 až 1918). . . . .	209
4.17	Hodnoty prvkov matice dynamiky systému riadeného stavovou spätnou väzbou s ustáleným zosilnením. . . . .	213
4.18	Konvergencia prvkov matice <b>P</b> pri iteratívnej metóde výpočtu Ljapunovej rovnice. . . . .	216
5.1	Hodnota posledného člena kvadratickej funkcie vplýva na hodnotu funkcie, ale nie na hodnotu argumentu v minime. . . . .	225
5.2	Argument minimalizácie kvadratickej funkcie, t. j. $u^*$ , ostáva nezávislý na jej poslednom člene. . . . .	227
5.3	Stavy a vstupy dvojitého integrátora pri MPC riadení bez obmedzení.	234
5.4	Grafická ilustrácia hodnoty účelovej funkcie pri MPC riadení dvojitého integrátora. . . . .	236
5.5	Ak je model systému dokonale známy, predikované stavy ( $\times$ ) sú identické so skutočnými stavmi ( $\circ$ ) v spätej väzbe. . . . .	238
5.6	Stavy v spätej väzbe ( $\circ$ ) nie sú totožné s predikovanými stavmi ( $\times$ ) ak dynamiku systému nepoznáme dokonale. . . . .	239
5.7	Zobrazenie priebehov skutočných stavov a vstupov vo spätej väzbe ( $\circ$ ) a priebehov predikovaných stavov a vstupov ( $\times$ ). . . . .	241
5.8	Štvrtinový model aktívneho (odpruženia) podvozku autobusu. . . .	242

5.9	Spätnoväzbová odozva systému a riadiace vstupy sú v prípade LQ riadenia a MPC bez obmedzení identické. . . . .	245
5.10	Spätnoväzbová odozva systému na vstupy (○) vs. predikcia stavov a vstup (×) pri prediktívnom riadení bez obmedzení a na konečnom horizonte $n_p = 20$ krokov. . . . .	250
5.11	Spätnoväzbová odozva systému na vstupy (—) vs. predikcia stavov a vstupu (---) prediktívneho riadenia bez obmedzení na konečnom horizonte rôznej dĺžky. . . . .	251
5.12	Spätnoväzbová odozva (○) je identická s predikovanou odozvou (×) v prípade riadenia s nekonečným horizontom pomocou zloženého riadiaceho zákona. . . . .	252
5.13	Hodnota účelovej funkcie $J_k$ prediktívneho riadenia bez obmedzení na konečnom horizonte rôznej dĺžky. . . . .	254
5.14	Schematický náčrt elektromechanického systému dvoch zotrvačníkov. . . . .	257
5.15	Porovnanie LQ riadenia bez obmedzení a so saturovaným (orezávaným) LQ riadením. . . . .	259
5.16	Porovnanie LQ riadenia bez obmedzení s konzervatívne naladeným saturovaným LQ riadením . . . . .	260
5.17	Ilustrácia ohraničení absolútnej hodnoty vstupov. . . . .	261
5.18	Schematický náčrt konfigurácie kvadrokoptéry v priestore. . . . .	268
6.1	Rovnaké úrovne atmosferického tlaku vyjadrené pomocou vrstevníc. . . . .	282
6.2	Kvadratická funkcia dvoch nezávislých premenných a jej vrstevnice. . . . .	283
6.3	Plocha kvadratickej funkcie v 3D a jej priemet do 2D – vrstevnice. Symbol „○“ označuje minimum funkcie bez uvažovania obmedzení. . . . .	285
6.4	Vrstevnice kvadratickej funkcie dvoch premenných. Symbol „○“ označuje minimum funkcie bez obmedzení. . . . .	286
6.5	Vzťah vrstevnice a gradientu funkcie. . . . .	287
6.6	Vrstevnica a gradienty funkcie v danej funkčnej hodnote. . . . .	289
6.7	Obmedzenie v tvare rovnosti v priestore dvoch optimalizačných premenných. . . . .	290
6.8	Obmedzenie v tvare nerovnosti v priestore dvoch optimalizačných premenných. . . . .	291
6.9	Jednotlivé obmedzenia v tvare nerovnosti vytvárajú množinu zlúčiteľných riešení QP. . . . .	292
6.10	Určité kombinácie obmedzení vytvárajú obmedzenia, ktoré sú nezlúčiteľné s riešením QP. . . . .	293
6.11	Lineárne obmedzenia znázornené v priemete kvadratickej funkcie. . . . .	294
6.12	Gradient lineárneho obmedzenia v tvare nerovnosti. . . . .	295
6.13	Gradient obmedzenia s indexom $i = 3$ . . . . .	296

6.14	Riešenie minimalizácie QP a zobrazenie jeho argumentu v optime.	298
6.15	Poloha „neobmedzeného“ riešenia problému QP vzhľadom na obmedzenia.	304
6.16	Ilustrácia aktívnych a neaktívnych obmedzení v tvare nerovnosti a rovnosti.	305
6.17	Riešenie ostáva rovnaké aj po odstránení neaktívnych obmedzení.	306
6.18	Joseph-Louis Lagrange (1736 až 1813).	308
6.19	Gradient funkcie a obmedzení sú v optime rovnobežné, pričom majú opačný smer a inú veľkosť.	309
6.20	Gradienty v bode riešenia sú rovnobežné, majú opačný smer a rôzne dĺžky (veľkosť).	312
6.21	Karush, Kuhn a Tucker.	314
6.22	Iterácie načrtnutého QP algoritmu na báze metódy aktívnych množín.	324
6.23	Metóda aktívnych množín po zmene Hessiánu.	324
6.24	Iterácie QP algoritmu na báze metódy vnútorného bodu.	329
6.25	Vnútorná trajekória riešenia a možné numerické nepresnosti.	330
7.1	Formulácia optimalizačného problému MPC je ekvivalentná so všeobecnou formuláciou QP.	335
7.2	Riešenie optimalizačnej úlohy MPC pomocou riešiča „quadprog“.	336
7.3	Optimálna postupnosť budúcich vstupov na riadenie podvozku autobusu v čase $k$ .	338
7.4	Stavy a vstup pri spätnoväzbovom riadení podvozku autobusu pomocou algoritmu MPC s obmedzeniami.	342
7.5	Porovnanie rozdielu výchylky neodpruženej hmoty (nadstavby) voči výchylke odpruženej hmoty (kolesu) vozidla pri prechode cez prekážku s aktívnym riadením podvozku a bez neho.	343
7.6	Schematický náčrt spätnoväzbového riadenia do nulovej referencie s úplným meraním stavu.	343
7.7	Schematický náčrt spätnoväzobného riadenia podľa referenčnej trajektórie pomocou matice stavového príkazu.	344
7.8	Dopravné lietadlo Boeing 777.	345
7.9	Zmena letovej výšky lietadla Boeing 777-300ER.	345
7.10	Prediktívne riadenie letovej výšky lietadla Boeing 777-300ER.	347
7.11	Ilustrácia vstupov a výstupov pri zmene letovej výšky.	348
7.12	Schematický náčrt doprednej väzby na zabezpečenie sledovania žiadanej referenčnej trajektórie pomocou tzv. matice stavového príkazu a s riadiacim členom v ustálenom stave.	349
7.13	Integrácia odchýlky riadenia na sledovanie referenčnej trajektórie.	350
7.14	Ilustrácia zjednodušenej schémy výmenníka tepla.	351

7.15	Sledovanie predpísaného priebehu teploty produktu vo výmenníku tepla pomocou rozšírenia stavovej reprezentácie o integračnú zložku.	353
7.16	Schematický náčrt MPC s rozšírením predikčnej dynamiky na prírastkový tvar.	356
7.17	MPC riadenie výšky podvozku autobusu pomocou výpočtu vstupov v prírastkovom tvere s ohľadom na ich obmedzenia.	358
7.18	Simulácia prediktívneho riadenia aktívneho podvozku autobusu s obmedzeniami na amplitúdu ako aj prírastky vstupu.	361
7.19	Vlastný funkčný blok v Simulinku na základe interpretovaného jazyka MATLABu.	363
7.20	Písanie vlastnej funkcie pre Simulink pomocou jazyku MATLAB.	363
7.21	Podsystém nelineárneho modelu invertovaného kyvadla.	364
7.22	Podsystém na grafické znázornenie výstupov simulácie.	365
7.23	Podsystém na dokreslenie dvoch obmedzení do virtuálneho osciloskopu.	366
7.24	Výsledná bloková schéma spätnoväzbového PID a LQ riadenia kyvadla.	368
7.25	Volná odozva invertovaného kyvadla a neúspešné PID riadenie s automaticky naladenými konštantami.	370
7.26	LQ riadenie invertovaného kyvadla.	371
7.27	Podsystém na online riešenie problému MPC.	373
7.28	Bloková schéma spätnoväzbového riadenia invertovaného kyvadla pomocou prediktívneho riadenia s obmedzeniami.	374
7.29	Prediktívne riadenie invertovaného kyvadla s obmedzením polohy vozíka a akčného zásahu.	376
7.30	Bloková schéma simulácie MPC riadenia invertovaného kyvadla so simuláciou zašumených meraní.	378
7.31	Odporúčaný podsystém na vizualizáciu výsledkov simulácie.	379
7.32	Prediktívne riadenie invertovaného kyvadla s obmedzeniami výchylky vozíka a akčného zásahu pri neúplnej informácii o stave v prítomnosti šumu.	381
8.1	Simulácia riadeného systému z počiatočného stavu $\mathbf{x}_0 = [0.5 \ - 0.5]^T$ — priebeh výstupu, vstupu a účelovej funkcie.	391
8.2	Simulácia riadeného systému z počiatočného stavu $\mathbf{x}_0 = [0.8 \ - 0.8]^T$ — priebeh výstupu a vstupu.	391
8.3	Optimálna predikcia $\tilde{\mathbf{u}}_k$ v čase $k$ a jej zvyšok $\tilde{\mathbf{u}}_{k+1}$ .	394
8.4	Ilustrácia invariantnej cieľovej množiny $\Omega$ spolu so stavovou trajektóriou.	397

8.5	Ilustrácia množiny všetkých riešiteľných, a teda prípustných počiatocných stavov, $S_\Omega$ . . . . .	399
8.6	Ilustrácia maximálnej invariantnej cieľovej množiny $\Pi_{n_c} = \Pi_\infty$ v dvojrozmernom stavovom priestore, ktorá je definovaná ako prienik koncových obmedzení na horizonte $n_c$ . . . . .	401
8.7	Grafické znázornenie koncových obmedzení, ktorých prienik predstavuje množiny $\Pi_n$ , $n = 0, 1, 2, \dots, \infty$ . . . . .	403
8.8	Grafické znázornenie stavových trajektórií: predikovanej v stave $\mathbf{x}(0)$ a skutočnej, simulovanej v uzavretej slučke. . . . .	415

# Zoznam tabuliek

5.1	Vybrané pravidlá derivácie maticových výrazov. . . . .	228
5.2	Zmena zosilnenia riadiaceho zákona a vlastné čísla matice dynamiky v závislosti od predikčného horizontu pre MPC riadenie s konečným horizontom. . . . .	249
6.1	Diagnostické statusy rutiny quadprog(). . . . .	300
8.1	Porovnanie hodnôt účelovej funkcie – získaných predikciou a skutočných, získaných simuláciou v uzavretej slučke – pre rôzne dĺžky predikčného horizontu. . . . .	414



# Matematické značenie

Latinská abeceda:

- 0** Nulová matica príslušného rozmeru.
- 1** Blokový vektor jednotkových matíc  $\mathbf{I}$ .
- a** Množina aktívnych ohraničení.
- $\mathbf{a}_p$**  Množina aktívnych ohraničení v kroku  $p$ .
- $\mathbf{a}^*$**  Množina aktívnych ohraničení pri optimálnom riešení.
- A** Stavová matica prechodu, resp. matica dynamiky diskrétneho stavového modelu.
- $\mathbf{A}_c$**  Matica násobiaca optimalizačnú premennú na ľavej strane obmedzení v tvare nerovnosti  $\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_c$ .  
V Kap. 2 matica dynamiky spojitého stavového modelu.
- $\mathbf{A}_c^i$**   $i$ -tý riadok matice  $\mathbf{A}_c$ .
- $\mathbf{A}_c^a$**  Obmedzenia v aktívnej množine  $\mathbf{a}$ .
- $\mathbf{A}_e$**  Matica násobiaca optimalizačnú premennú na ľavej strane obmedzenia v tvare rovnosti  $\mathbf{A}_e \vec{\mathbf{u}}_k = \mathbf{b}_e$ .
- $\mathbf{b}_c$**  Stípcový vektor na pravej strane obmedzení v tvare nerovnosti  $\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_c$ , resp.  $\mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k$ .
- $\mathbf{b}_c^i$**   $i$ -tý riadok, resp. prvok vektora  $\mathbf{b}_c$ .
- $\mathbf{b}_c^a$**  Obmedzenia v aktívnej množine  $\mathbf{a}$ .
- $\mathbf{b}_e$**  Stípcový vektor na pravej strane obmedzení v tvare rovnosti  $\mathbf{A}_e \vec{\mathbf{u}}_k = \mathbf{b}_e$ .
- $\mathbf{b}_0$**  Matica na pravej strane obmedzení v tvare nerovnosti  $\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k$ .
- B** Matica vstupu diskrétneho stavového modelu.

$\mathbf{B}_c$	Matica vstupu spojitého stavového modelu.
$\mathbf{B}_0$	Matica na pravej strane obmedzení v tvare v tvare nerovnosti $\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k$ , ktorú násobíme aktuálnym stavom.
$\mathbf{C}, \mathbf{C}_c$	Matica výstupu diskrétneho (spojitého) stavového modelu.
$\mathbf{D}, \mathbf{C}_c$	Matica priamej väzby vstupu na výstup diskrétneho (spojitého) stavového modelu.
$e$	Odchýlka riadenia (regulácie), riadiaca (regulačná) odchýlka.
$\mathbf{e}$	Vektor odchýlok riadenia pre jednotlivé výstupy.
$f(\mathbf{u})$	Všeobecná skalárna kvadratická funkcia.
$\mathbf{f}(\mathbf{x}, \mathbf{u})$	Nelineárna funkcia dynamiky.
$g(\mathbf{u})$	Všeobecný tvar ľavej strany lineárnych obmedzení v tvare nerovnosti.
$g_i(\mathbf{u})$	$i$ -té lineárne obmedzenie v tvare nerovnosti.
$\mathbf{g}$	Gradient, pre ktorý platí $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$ .
$\mathbf{G}$	Matica MPC účelovej funkcie, pričom platí $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$ .
$\mathbf{h}(\mathbf{u})$	Všeobecný tvar ľavej strany lineárnych obmedzení v tvare rovnosti.
$\mathbf{h}(\mathbf{x}, \mathbf{u})$	Nelineárna funkcia merania.
$h_j(\mathbf{u})$	$j$ -te lineárne obmedzenie v tvare rovnosti.
$\mathbf{H}$	Hessián, Hessova matica.
$\mathbf{I}$	Jednotková matica, matica s jednotkami na hlavnej diagonále.
$J$ ( $J_k$ )	Hodnota účelovej (kriteriálnej) funkcie (v čase $k$ ).
$k$	Celočíselný diskrétny časový index (vzorka), pričom platí $t = kT_s$ .
$\mathbf{K}$	Zosilnenie stavového riadiaceho zákona, napr. ustálené LQ zosilnenie vrátane znamienka zápornej spätej väzby.
$\mathbf{K}_i$	$i$ -tý blokový riadok zosilnenia $\mathbf{K}$ .
$\mathcal{K}$	Zosilnenie stavového riadiaceho zákona vrátane predikcie v prípade MPC bez obmedzení.
$\mathbf{L}$	Transformačná matica vektora predikovaných stavov $\vec{\mathbf{x}}_k$ na vektor predikovaných výstupov $\vec{\mathbf{y}}_k$ . V Kap. 2 zosilnenie Luenbergerovho pozorovateľa stavu.
$\mathcal{L}$	Lagrangián, Lagrangeova funkcia.
$\mathbf{M}$	Predikčná matica voľnej (autonómnej) odozvy.

$\mathbf{M}_i$	$i$ -tý blokový riadok predikčnej matice $\mathbf{M}$ .
$n_c$	Horizont koncových obmedzení.
$n_e$	Počet obmedzení v tvare rovnosti.
$n_o$	Počet obmedzení v tvare nerovnosti.
$n_p$	Predikčný horizont, resp. dĺžka predikčného horizontu.
$n_u$	Počet vstupov.
$n_x$	Počet stavov.
$n_y$	Počet meraných výstupov.
$\mathbf{N}$	Predikčná matice vynútenej odozvy.
$\mathbf{N}_i$	$i$ -tý blokový riadok predikčnej matice $\mathbf{N}$ .
$\mathbf{N}_x$	Matica stavového príkazu.
$\mathcal{N}(\cdot, \cdot)$	Normálne rozdelenie.
$\mathbf{P}$	Matica váhovania koncového stavu. V Kap. 2 (pri Kalmanovej filtrácii) kovariančná matice chyby odhadu.
$\mathbf{Q}$	Matica váhovania stavov, váhovacia (penalizačná) matice stavov. V Kap. 2 (pri Kalmanovej filtrácii) kovariančná matica procesného šumu.
$r$	Žiadaná hodnota výstupnej veličiny, referencia.
$\mathbf{r}$	Vektor žiadaných hodnôt výstupných veličín, vektor referencií.
$\mathbf{R}$	Matica váhovania vstupov, váhovacia (penalizačná) matice vstupov. V Kap. 2 (pri Kalmanovej filtrácii) kovariančná matica šumu merania.
$\mathcal{R}$	Rozšírená matice váhovania vstupov.
$\mathbb{R}$	Množina reálnych čísel.
$S_\Omega$	Množina zlučiteľných stavov.
$t$	Spojity čas.
$T_s$	Periódna vzorkovanie.
$\mathbf{u}$	Zovšeobecnená optimalizačná premenná.
$\mathbf{u}_p$	Čiastočný výsledok v kroku $p$ .
$\Delta \mathbf{u}_p$	Prírastok čiastočného výsledku v kroku $p$ .

$\mathbf{u}^*$	Minimum všeobecnej kvadratickej funkcie s obmedzeniami.
$\mathbf{u}^\circ$	Minimum všeobecnej kvadratickej funkcie bez obmedzení.
$\mathbf{u}_k^*$	Optimálny vektor vstupov v čase $k$ .
$\mathbf{u}_k$	Vektor vstupov v čase $k$ .
$\bar{u}$	Horné obmedzenie amplitúdy vstupu (skalár).
$\underline{u}$	Dolné obmedzenie amplitúdy vstupu (skalár).
$\bar{\mathbf{u}}$	Horné obmedzenie amplitúdy vstupov (vektor).
	V Kap. 2 pracovný (linearizačný) bod v zmysle vstupu.
$\underline{\mathbf{u}}$	Dolné obmedzenie amplitúdy vstupov (vektor).
$\overrightarrow{\mathbf{u}}_k$	Postupnosť (sekvencia) budúcich vektorov vstupu vypočítaná v čase $k$ .
$\tilde{\mathbf{u}}_{k+1}$	Chvost postupnosti optimálnych vstupov.
$\mathbf{v}_k$	Vektor šumu merania v čase $k$ .
$V(\mathbf{x}_k)$	Diskrétna Ljapunova funkcia.
$V(\mathbf{x}(t))$	Spojité Ljapunova funkcia.
$\mathbf{w}_k$	Vektor procesného šumu v čase $k$ .
$\mathbf{x}_e$	Odhýlka medzi vektorom skutočných a žiadanych stavov, resp. odchýlka riadenia v zmysle stavov, t. j. $\mathbf{x}_e = \mathbf{x}_k - \mathbf{x}_r$ .
$\mathbf{x}_k$	Stavový vektor v čase $k$ .
$\mathbf{x}_k^I$	Stavový vektor integrálu odchýlky riadenia v čase $k$ .
$\mathbf{x}_r$	Referenčný, resp. žiadany stav.
$\bar{\mathbf{x}}$	Pracovný (linearizačný) bod v zmysle stavu.
$\hat{\mathbf{x}}_k$	Odhad stavového vektora v čase $k$ .
$\mathbf{x}_k$	Rozšírený stavový vektor v čase $k$ .
$\overrightarrow{\mathbf{x}}_k$	Postupnosť budúcich (predikovaných) vektorov stavu v čase $k$ .
$\mathbf{y}_k$	Vektor výstupov v čase $k$ .
$\overrightarrow{\mathbf{y}}_k$	Postupnosť budúcich (predikovaných) vektorov výstupu v čase $k$ .
$\mathbb{Z}$	Množina celých čísiel.

Grécka abeceda:

$\alpha$	Faktor skrátenia kroku pri hľadaní po priamke.
$\lambda$	Vektor Lagrangeovych multiplikátorov (násobičov) $\lambda_i$ pre obmedzenia v tvare nerovnosti. V Kap. 2 vektor vlastných čísel matice dynamiky.
$\nu$	Vektor Lagrangeovych multiplikátorov (násobičov) $\nu_j$ pre obmedzenia v tvare rovnosti.
$\Omega$	Cieľová množina.
$\Pi_\infty$	Maximálna invariantná cieľová množina.
$\Psi$	Dynamika uzavretého riadiaceho obvodu, t. j. $\Psi = (\mathbf{A} + \mathbf{B}\mathbf{K})$ .

Matematické operácie:

$\arg$	Argument (funkcie, minimalizácie).
$\det(\mathbf{X})$	Determinant matice $\mathbf{X}$ .
$\text{diag}$	Diagonálna (matica).
$\max$	Maximalizácia (funkcie).
$\min$	Minimalizácia (funkcie).
$\text{rank}(\mathbf{X})$	Hodnosť matice $\mathbf{X}$ .
$\text{tr}(\mathbf{X})$	Stopa matice $\mathbf{X}$ .
$\mathbf{X}^T$	Transpozícia matice $\mathbf{X}$ .
$\mathbf{X} = \mathbf{X}^T$	Symetrická matica.
$\mathbf{X}^{-1}$	Inverzia matice $\mathbf{X}$ .
$\mathbf{X} \succ 0$	Kladná (pozitívna) definitnosť matice.
$\mathbf{X} \succeq 0$	Kladná (pozitívna) semidefinitnosť matice.

Niektoré špecifické značenia sú definované lokálne v jednotlivých kapitolách.



*Janke*  
*a*  
*Janke*



Autori ďakujú Agentúre na podporu výskumu a vývoja (APVV) za finančnú podporu pri riešení projektu, v rámci ktorého vznikla táto publikácia: grant č.

**APVV-14-0399.**



Autori ďakujú agentúre KEGA Ministerstva školstva, vedy, výskumu a športu Slovenskej Republiky za finančnú podporu pri riešení projektu, v rámci ktorého vznikla táto publikácia: grant č. **005STU-4/2018**



MINISTERSTVO ŠKOLSTVA,  
VEDY, VÝSKUMU A ŠPORTU  
SLOVENSKEJ REPUBLIKY

Autori ďakujú Európskemu fondu regionálneho rozvoja za finančnú podporu v rámci Operačného programu Výskum a vývoj a Operačného programu Výskum a inovácie pri riešení projektu, v rámci ktorého vznikla táto publikácia: grant č. **ITMS-26240220084**, „Univerzitný vedecký park STU Bratislava“ a grant č. **ITMS-313021D243**, „Univerzitný vedecký park STU Bratislava–II. fáza“.



**EURÓPSKA ÚNIA**  
Európsky fond regionálneho rozvoja





# Predslov

Učebnica *Základy prediktívneho riadenia* je určená predovšetkým poslucháčom druhého ročníka inžinierskeho štúdia študijného programu Automatizácia a informatizácia strojov a procesov na Strojníckej fakulte Slovenskej Technickej Univerzity v Bratislave. Bola napísaná pre potreby a v duchu povinného predmetu Teória automatického riadenia III a svojim obsahom aj úzko nadväzuje na predmety Teória automatického riadenia I a II, Identifikácia sústav, Simulácia a optimalizácia a ďalšie.

Študenti druhého stupňa vysokoškolského štúdia, ktorí pracujú na svojich diplomových prácach, môžu učebnicu využívať na realizáciu algoritmov prediktívnej regulácie vo svojich záverečných projektoch. Samozrejme, učebnica môže byť prospešná aj pre tých študentov automatického riadenia alebo mechatroniky na doktorandskom stupni vysokoškolského štúdia, ktorí nemali predchádzajúci kontakt s myšlienkou prediktívneho riadenia. Hoci spôsob výkladu problematiky ako aj volba obsahu publikácie predurčuje knihu najmä pre začiatočníkov v tejto oblasti automatického riadenia, učebnica môže byť užitočná aj na oprášenie vedomostí odborníkov.

V rozsahu učebnice, ktorá podľa našich skúseností pokrýva plný semester prednášok, bolo možné uviesť len úvodné poznatky z takej rozsiahlej oblasti, ako je súčasná teória prediktívneho a optimálneho riadenia. Sústredili sme sa najmä na formuláciu algoritmu prediktívneho riadenia, ktorú dnes môžeme považovať už za úplne štandardnú. V tejto učebnici preto predpokladáme model predikcie založený na stavovej reprezentácii, zloženú účelovú funkciu a formuláciu s procesnými obmedzeniami. Učebnica predstavuje základnú terminológiu, postupy a techniky v prediktívnom riadení. Tomuto predchádza nevyhnutný úvod do problematiky stavového modelovania, linearizácie, diskretizácie a odhadu stavu, ktorá v dostupnej študijnnej literatúre na danom študijnom programe nie je v súčasnosti dostatočne pokrytá. Na záver kniha uvádza spôsob garancie stability a riešiteľnosti problému prediktívneho riadenia.

Nové poznatky sú bohatu graficky ilustrované, doplnené príkladmi a počíta-

čovými simuláciami pre úplnosť didaktického prístupu. Vo výklade sme sa snažili predísť zbytočnej komplikácii niektorých pojmov a sústrediť sa skôr na pedagogickú kvalitu textu vychádzajúc z našich skúseností z prednášok a cvičení. Učebnica sa môže pre expertného čitateľa javiť v niektorých častiach až príliš podrobná, respektíve matematicky nedostatočne rigorózna, ale našim cieľom bolo vždy zachovať didaktickú kvalitu a brať do úvahy pripravenosť a chápanie reálnych študentov. Motiváciou pri určení hĺbky a štýlu prezentácie jednotlivých kapitol preto neboli hypotetické očakávania o doteraz nadobudnutných základoch a existujúcich vedomostiah poslucháčov, ale skôr adaptácia na realitu súčasného univerzitného vzdelávania. Ako hovorí názov učebnice, jej cieľom je najmä poskytnúť pevné základy čitateľovi, aby v budúcnosti mohol sebavedome otvoriť aj texty s hlbším záberom tematiky a v konečnom dôsledku tak realizovať moderné interpretácie algoritmov prediktívneho riadenia. Ďalšie vybrané formulácie prediktívneho riadenia (explicitné, adaptívne, robustné, nelineárne, ...) a najmä jeho praktická implementácia budú predmetom potenciálneho pokračovania tejto učebnice.

Veríme, že písanie učebnice nekončí ani po odovzdaní rukopisu do tlače. Preto autori budú vďační každému čitateľovi, ktorý bude ochotný vyjadriť svoje kritické poznatky k tejto učebnici, prípadne upozorní na chyby a omyly, ktoré sa môžu v texte nachádzať.

Na záver chceme vyjadriť vďaku za cenné pripomienky, konštruktívnu kritiku a užitočné pripomienky k textu učebnice recenzentom rukopisu, prof. Ing. Miroslavovi Fikarovi, DrSc., prof. Ing. Štefanovi Kozákovi, PhD., prof. Ing. Borisovi Roháč-Ilkivovi, CSc., a tiež ďalším čitateľom rukopisu, Ing. Lorántovi Bugárovi, Ing. Petrovi Minarčíkovi a Bc. Gáborovi Penzingerovi. Za dodatočnú jazykovú korektúru ďakujeme aj vydavateľstvom prideleným korektorom.

Bratislava, august 2018.

Gergely Takács a  
Martin Gulan

# O autoroch

## Gergely Takács

V roku 2006 ukončil štúdium na Strojníckej fakulte (SjF) Slovenskej Technickej Univerzity v Bratislave (STU v Bratislave) v odbore Mechatronika, a následne počítačoval v internom doktorandskom štúdiu na rovnakom odbore. V rokoch 2006 až 2008 pôsobil pri viacerých príležitostiach ako hostujúci výskumník na Oxfordskej Univerzite, pracujúc v oblasti výpočtovo efektívneho prediktívneho riadenia. Po ukončení doktorandského štúdia nastúpil na Ústav automatizácie, merania a aplikovanej informatiky (ÚAMAI) ako samostatný výskumný pracovník. V roku 2015 prešiel habilitačným konaním a v súčasnosti pôsobí ako docent v študijnom odbore Mechatronika, v kolektíve ÚAMAI SjF STU v Bratislave.

Jeho odborný záujem je zameraný najmä na prediktívne riadenie a súčasný odhad stavov a parametrov, s dôrazom na praktickú realizáciu spomenutých technológií. Vo výskume sa venuje aj aktívnomu tlmeniu vibrácií, riadeniu smart materiálov a iných mechatronických systémov s rýchloou dynamikou. Pedagogicky sa podieľa na zabezpečení predmetov Teória automatického riadenia III, Mikroprocesorová technika, Mikropočítačová technika a venuje sa vedeniu bakalárskych, diplomových a dizertačných prác. Okrem iných významných publikácií je hlavným autorom zahraničnej vedeckej monografie Model Predictive Vibration Control vydanej Springer Verlag, London. Ako zodpovedný riešiteľ a spoluriešiteľ sa zúčastnil rôznych slovenských a zahraničných projektov, napr. APVV, VEGA, KEGA, ESF, NIL, interných grantov STU v Bratislave, a iných.

Osobne prezentoval výsledky svojho výskumu po celom svete, vrátane miest ako Oxford, Cambridge, New York, Bangkok, Peking, Kapské mesto, Hong Kong, Hiroshima a iných. Dvakrát získal ocenenie za najlepší príspevok a konferenčnú prezentáciu. Je aktívnym recenzentom pre niekoľko významných časopisov vo svojom odbore, vrátane Mechatronics, Journal of Vibration and Control, Journal of Sound and Vibration a ďalších. Je členom organizácie The Institute of Electrical and Electronics Engineers (IEEE).

## **Martin Gulan**

Bakalárske, inžinierske a doktorandské štúdium ukončil na Strojníckej fakulte STU v Bratislave v odboroch Aplikovaná mechanika a mechatronika (2010), Aplikovaná mechanika (2012), a Mechatronika (2015). V roku 2011 pôsobil ako hostujúci výskumník v Advanced Micro and Nanosystems Laboratory, Department of Aerospace and Mechanical Engineering na University of Arizona, a v roku 2014 v Laboratory of Signals and Systems na univerzite CentraleSupélec. Po ukončení doktorandského štúdia nastúpil ako postdoktorand STU na Ústav automatizácie, merania a aplikovanej informatiky, kde od roku 2018 pôsobí na mieste odborného asistenta v študijnom odbore Automatizácia.

Jeho odborný záujem je zameraný najmä na oblast návrhu a výpočtovo efektívnej implementácie prediktívneho riadenia, a vniorenej optimalizácie v reálnom čase, s praktickým využitím v rôznych mechatronických aplikáciach. Pedagogicky sa podiela na zabezpečení predmetov Teória automatického riadenia II–III, Identifikácia, Softvérové technológie, a venuje sa vedeniu diplomových a bakalárskych prác. Ako spoluriešiteľ a zodpovedný riešiteľ sa zúčastnil rôznych slovenských a zahraničných projektov, napr. APVV, VEGA, KEGA, NIL, interných grantov STU v Bratislave a iných. Získal tiež Cenu Zväzu automobilového priemyslu Slovenskej republiky za vedenie najlepšej diplomovej práce v roku 2017.

# Programová podpora

Súčasťou tejto učebnice je aj séria príkladov na ilustráciu vysvetlovaných koncepciev a programová realizácia ich riešení.

Programy boli napísané v skriptovacom jazyku programového prostredia MATLAB spoločnosti The MathWorks. Príklady boli napísané a testované vo verzii softvéru MATLAB 2018a (R2018a), pričom nepredpokladáme značné zmeny vo funkciaľite využitých príkazov ani v budúcnosti. Softvér MATLAB bol v čase písania učebnice bezplatne dostupný<sup>2</sup> pre študentov a zamestnancov STU v Bratislave v rámci licencie Total Academic Headcount (TAH).

Súbory zdrojového kódu sú roztriedené na základe jednotlivých kapitol knihy. Názvy súborov sú uvedené v texte, najmä v príkladoch. Súbor zdrojových kódov je súčasťou softvérového úložiska na sledovanie verzií Git a je stiahnutelný z adresy:

<http://zpr.ergelytakacs.com/>

V prípade, že čitateľ nájde chybu v zdrojovom kóde, je vítaný navrhnúť zmeny priamo v prostredí portálu GitHub prostredníctvom požiadavky na tzv. Začlenenie kódú<sup>34</sup>. Alternatívne radi akceptujeme nové nápady alebo upozornenia na chyby aj prostredníctvom fóra v časti Problémy<sup>56</sup>.

---

<sup>2</sup><https://www.stuba.sk/matlab/>

<sup>3</sup>angl. Pull requests

<sup>4</sup><https://github.com/ergelytakacs/Zaklady-prediktivneho-riadenia/pulls>

<sup>5</sup>angl. Issues

<sup>6</sup><https://github.com/ergelytakacs/Zaklady-prediktivneho-riadenia/issues>



# Kapitola 1

## Úvod do problematiky prediktívneho riadenia

### 1.1 Postačujú vždy tradičné metódy riadenia?

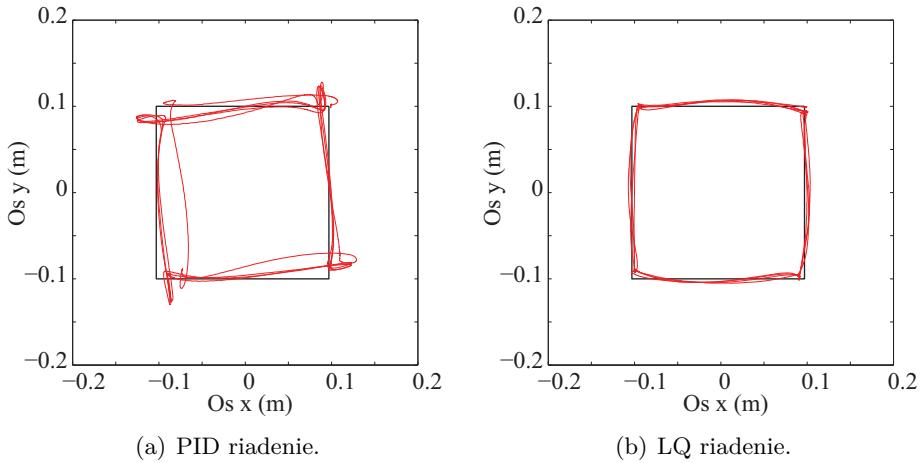
Proporcionálno-integračno-derivačné<sup>1</sup> (PID) riadenie môžno považovať za jednu z najdôležitejších metód automatického riadenia. Majú jednoduchú matematickú štruktúru, sú ľahko a ľacne implementovateľné, široko využívané a akceptované v priemysle. Klasické techniky automatického riadenia, ako napríklad metódu umiestnenia pôlov<sup>2</sup> a mnoho iných dobre známych stratégii taktiež môžeme klasifikovať medzi klasické metódy riadenia. Pre mnohé systémy a procesy splnenie požiadaviek kvality regulačného pochodu nevyžaduje náročné riešenia, napríklad pre jednoduché lineárne systémy, systémy prvého rádu, atď. V týchto prípadoch úplne postačujú jednoduché, napríklad spomenuté klasické metódy riadenia. Často sa však stretнемo so systémami, pri ktorých vystihnutie ich dynamických vlastností vedie na zložitejší matematický opis systému. V takýchto prípadoch sú potrebné zložitejšie riadiace zákony, keďže použitie jednoduchých regulátorov je vo všeobecnosti veľmi obtiažne (musia byť veľmi dobre naladené), dokonca je niekedy až nemožné zabezpečiť danú kvalitu riadenia.

Riadiace algoritmy, ktoré sú založené na princípe optimality riadenia, majú už väčšinou lepšiu kvalitu riadenia ako už spomenuté tradičné algoritmy založené iba na manipulácii s prenosom. To znamená, že tieto algoritmy sledujú predpísanú žiadanosť hodnotu precíznejšie alebo reagujú na vonkajšiu poruchu rýchlejšie. Dokážu si tiež poradiť s mnohorozmernými, neautonómnymi systémami so vzájomnými väzbami medzi vstupnými a výstupnými veličinami. Do tejto triedy môžeme za-

---

<sup>1</sup>angl. proportional, integral, derivative

<sup>2</sup>angl. pole placement



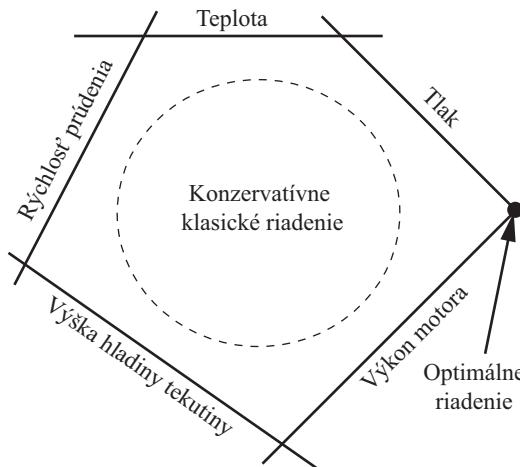
Obr. 1.1. Ilustrácia rozdielu medzi PID a LQ riadením systému „gulôčka na ploche“, kde systém má za úlohu sledovať predpísanú trajektóriu – štvorec [141].

radiť všeobecne známe lineárne kvadratické (LQ)<sup>3</sup> riadenie, ktoré je založené na neohraničenej minimalizácii kvadratickej účelovej funkcie na nekonečnom horizonte. Ďalej tu môžeme spomenúť stavové riadenie na základe  $H_2$  (H-dva) normy,  $H_\infty$  (H-nekonečno) normy a ďalšie. Stavové riadenie s pevnou spätnou väzbou (t. j. s ustáleným zosilnením) je stále široko akceptované a pomerne jednoduché, avšak už môže vyžadovať algoritmické prostriedky za účelom odhadu aktuálneho stavu z merania, ako napríklad Kalmanov filter.

Rozdiel medzi klasickým riadením a takzvaným „moderným“ riadením zná zornia priebehy na Obr. 1.1. Cieľom je sledovať predpísanú štvorcovú trajektóriu mechanickým objektom, napríklad gulôčkou. Na Obr. 1.1(a) vidíme dosiahnuté výsledky pomocou PID stratégie (klasické riadenie), kým na Obr. 1.1(b) vidíme oveľa presnejšiu trajektóriu dosiahnutú pomocou LQ metódy stavového (moder-ného) riadenia.

Takéto riadiace stratégie neobsahujú explicitne stanovené ciele alebo obme- dzenia riadenia. Kvalita riadenia tradičných metód je nekonzistentná, regulačné slučky sa nesprávajú robustne voči zmenám a poruchám a navyše sa sústredia na výkon lokálneho riadenia [91]. To znamená, že tieto metódy riadenia hlavne zabezpečia splnenie krátkodobých cieľov riadenia, čo je žiaľ stratégia, ktorá sa môže v budúcnosti vypomstíť [3]. Väčšina tradičných prístupov riadenia sleduje filozofiu „Vyhrať bitku ale prehrať vojnu“ [3].

<sup>3</sup>angl. linear quadratic



Obr. 1.2. Ilustrácia procesných obmedzení.

Každý reálny systém má svoje praktické obmedzenia. Procesné obmedzenia vyplývajú z fyzikálnych vlastností riadeného systému a jeho časti. Senzory majú konečný rozsah merania, akčné členy neposkytujú nekonečný výkon a riadený systém neznesie extrémne veľké vstupy. Riadený proces môže vyžadovať tiež rôzne bezpečnostné ohraničenia, keďže prevádzka má svoje technické limity, alebo často vplývajú na proces riadenia aj ekonomické faktory.

Každý riadený systém má procesné obmedzenia, ktoré musíme pri riadení rešpektovať.



Žiaľ, často najlepší výkon riadenia dostaneme práve keď systém tlačíme na hranice možností, teda inými slovami prevádzkujeme blízko ohraničení. Tradičné riadiace stratégie ako PID vôbec neuvažujú s procesnými obmedzeniami, a práve preto sú nastavené konzervatívne a pracujú kompromisne. Vzhľadom na kompromisné ladenie sa nesprávajú príliš agresívne, a teda je menšia šanca, aby sa mohli porušiť rôzne praktické obmedzenia. Nevyužívajú však ani všetky možnosti na dosiahnutie optimálneho výsledku. Táto myšlienka je znázornená na Obr. 1.2. V strede vidíme konzervatívnu pracovnú obálku tradičných riadiacich algoritmov. Čiary predstavujú rôzne, často aj navzájom protichodné procesné obmedzenia. Procesné obmedzenia musíme rešpektovať, avšak najlepšie výsledky môžeme dosiať, ak sme blízko ich prieniku.

Výhody a nevýhody tradičných riadiacich algoritmov môžeme zhrnúť do nasledujúcich bodov:

**Výhody:**

- Jednoduchosť stratégie.
- Výpočtovo lacná a nekomplikovaná počítacová implementácia.
- Možnosť nastavovania parametrov rôznymi technikami automatického ladenia.

**Nevýhody:**

- Neobsahujú explicitne, t.j. v rámci kriteriálnej funkcie stanovené ciele riadenia.
- Neuvažujú s obmedzeniami akčných členov alebo procesu.
- Nesprávajú sa robustne voči náhodným zmenám a poruchám.
- Sústredia sa na efekt riadenia na lokálnej úrovni.
- Sústredia sa najmä na splnenie krátkodobých cieľov riadenia

Metódy ako PID riadenie alebo LQ riadenie sú široko využívané v priemysle, avšak z predošej diskusie je zrejmé, že nie sú dokonalé. Formulujme si preto nové výzvy pre lepšie riadenie: Čo sa stane ak vstup (riadiaci, resp. akčný zásah) v danom čase spôsobí problém v budúcnosti? Môžeme mať ešte lepší riadiaci výkon ako poskytuje LQ riadenie a podobné optimálne algoritmy riadenia? Ako môžeme dodržať splnenie rôznych procesných obmedzení, ako napríklad bezpečnostné ohraničenia, fyzické a technické obmedzenia prevádzky akčných členov a strojov, alebo ekonomicke faktory?

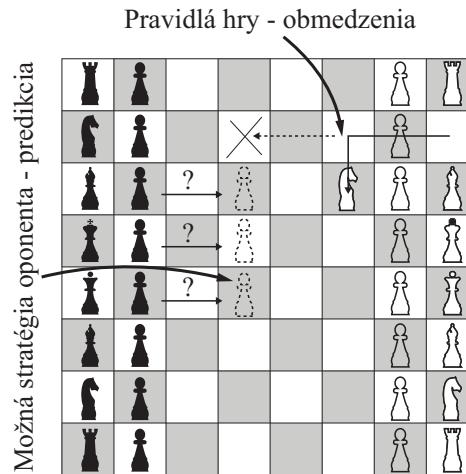
## 1.2 Základná myšlienka prediktívneho riadenia

Základná myšlienka prediktívneho riadenia je prepočítavanie najlepšieho možného – optimálneho riadiaceho zákona počas procesu riadenia. Predstavme si situáciu, že vieme automaticky a optimálne vypočítať akčný zásah, vždy podľa aktuálnej situácie.

Stratégia riadenia známa pod menom prediktívne riadenie<sup>4</sup>, resp. prediktívne riadenie na základe modelu (MPC)<sup>5</sup> robí presne to: prepočíta optimálny riadiaci zákon počas riadenia znova a znova v rámci jednej períody vzorkovania. Jej hlavnou výhodou je, že v prítomnosti procesných obmedzení má lepší výkon ako napríklad LQ riadenie. Na druhú stranu je ale zložitejšie na pochopenie ako aj na praktickú implementáciu. Prediktívne riadenie si poradí aj so systémami, ktoré sú iba ľahko riaditeľné pomocou klasických metód ako je PID riadenie, vrátane dopravných oneskorení v odozve a dynamiky vyššieho rádu.

<sup>4</sup>angl. predictive control

<sup>5</sup>angl. model predictive control

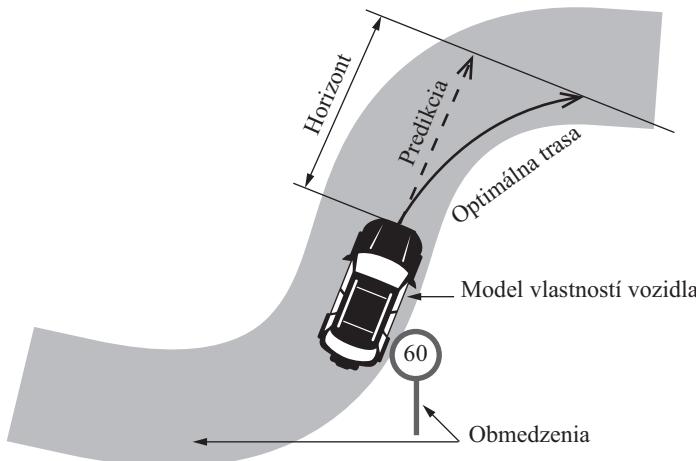


Obr. 1.3. Prediktívne riadenie ako analógia šachu.

### 1.2.1 Analógie prediktívneho riadenia

Prediktívne riadenie môžeme interpretovať napríklad analógiou so šachom [10]. Cieľom v šachu je dosiahnuť najlepší možný výsledok na základe očakávania (predvídania) tajov svojho oponenta pri dodržiavaní daných pravidiel a obmedzení hry. V prediktívnom riadení sa snažíme dosiahnuť najlepší možný riadiaci vstup predvídaním správania sa dynamiky riadeného systému a dodržiavaním procesných obmedzení riadeného systému.

Táto analózia je znázornená na Obr. 1.3. Dlhodobý cieľ každého hráča je výhra, čo je jasne definovaný a kvantifikateľný účel hry. Pri šachu sa hráči snažia predvítať budúce tajy oponenta, čo je istá forma *predikcie*. Na predpoveď budúcich tajov oponenta hráč vždy používa vedomosti, ako je napríklad odpozorovaná obvyklá stratégia protivníka, a tým si vytvára vnútorný *model* jeho možného správania sa. Protivníkom v tejto interpretácii prediktívneho riadenia je porucha pôsobiaca na systém. Predpoveď budúcej stratégie oponenta hráč pripravuje niekoľko tajov vopred – môžeme teda povedať, že predikcia prebieha na určitom *horizonte* do budúcnosti. Predpokladané tajy oponenta ale nemusia byť vždy totožné s realitou, a preto hráč musí vždy znova a znova prehodnotiť aktuálny *stav* hry a nanovo predpovedať budúce tajy oponenta. Inými slovami, horizont predikcie posúva vždy o krok ďalej. Každý hráč musí ale rešpektovať pravidlá hry, ktoré predstavujú *obmedzenia*. V prípade šachu môžeme považovať za obmedzenie to, že napríklad jazdec (kôň) sa môže na šachovnici pohybovať iba v tvare písmena „L“ a jeho zrkadlových obrazov.



Obr. 1.4. Prediktívne riadenie ako analógia šoférovania motorového vozidla.

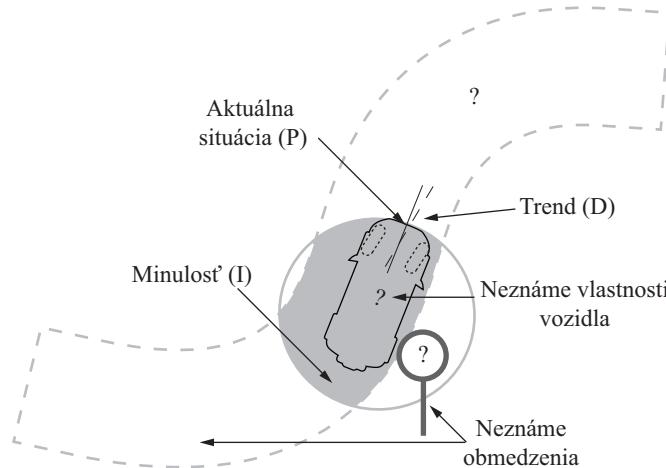
Ďalšou dobrou analógiou prediktívneho riadenia z každodenného života je aj šoférovanie vozidla (Obr. 1.4) [10]. Treba si uvedomiť, že vždy šoférujeme podľa určitých *kritérií* a musíme si vybrať často z viacerých protichodných cieľov. Samotná úloha dostať sa z bodu A do bodu B občas nestačí. Kritériom *optimálneho* šoférovania pritom môže byť napríklad aj šetrenie paliva, na základe čoho by sme mali pridávať plyn pomaly. Mohli by sme si zvoliť aj kritérium, ktorým sa chceme dostať do nášho cieľa v čo najkratšom čase, respektíve môžeme zvoliť aj najkratšiu cestu. Všetky tieto kritériá optimality sú pritom jasne kvantifikovateľné a meraťelné.

Pri šoférovani využívame mentálny *model* vozidla, ktorý vystihuje správanie sa vozidla na cestnej komunikácii. Tento model vozidla opisuje určité vlastnosti, napríklad brzdenie, zrýchlenie, či správanie sa vozidla v zákrute pri vyššej rýchlosťi. Športové auto bude preto úplne iné vlastnosti v zákrute ako kamión.

Pri šoférovani tiež vykonávame určitú formu *predikcie*, pretože neustále predvídame budúce dianie na základe spomenutého vnútorného modelu. Keď napríklad vidíme, že sa blíži zákruta, tak na základe tohto modelu predpokladáme, že by sme opustili vozovku, ak by sme neubrali z rýchlosťi.

Budúce dianie pri šoférovani predvídame v rámci určitého *horizontu*. Horizont môžeme pochopiť ako fyzický úsek cesty, ktorú vidíme, ale aj ako časový úsek plánovania do budúcnosti. Dĺžka tohto horizontu závisí od viditeľnosti, ale aj od praktických úvah: stačí napríklad vedieť čo sa deje pred nami na ceste niekoľko sto metrov a nepotrebujeme napríklad vedieť, že o 100 km nasleduje ostrá zákruta.

Celú našu cestu tiež nenaplánujeme v momente keď si sadneme do auta, ale



Obr. 1.5. Stratégia šoférovania motorového vozidla pomocou PID riadenia.

vždy prehodnotíme situáciu podľa aktuálneho *stavu*. Ak vidíme pred sebou prekážku na ceste, napríklad padnutý strom, musíme prehodnotiť situáciu. Preto sa naša stratégia riadenia, tak ako spomínaný horizont predikcie postupne posúva dopredu v čase a v priestore.

Ďalším dôležitým faktorom pri šoférovaní automobilu je, že vozidlo a samotná činnosť šoférovania podlieha rôznym *obmedzeniam*. Nemôžeme napríklad nechcene vyjsť z cestnej komunikácie, musíme dodržať maximálnu povolenú rýchlosť alebo nemôžeme zrýchliť z nuly na 100 km/h za pol sekundy. Naša stratégia riadenia teda vždy podlieha určitým praktickým obmedzeniam.

Skúsme sa teraz pohrať s myšlienou, ako by vyzeralo PID riadenie automobilu. Ak by sme šoférovali s logikou PID, na orientáciu by sme používali iba spätné zrkadlá. Inak povedané, nevedeli by sme čo sa deje v budúcnosti – pred nami, a mali sme informáciu iba o tom, či všetko bolo v poriadku v minulosti – za nami. Ak použijeme terminológiu známu z MPC, môžeme tvrdiť, že nám chýba predikcia.

Bizarná myšlienka šoférovania pomocou PID algoritmu má aj ďalšie zaujímé hypotetické dôsledky (Obr. 1.5). Kedže neuvažujeme s mentálnym modelom vozidla, nevieme ako sa správa naše vozidlo, či má zrýchlenie vesmírnej rakety alebo vozíka, respektívne nevieme na akej dráhe môžeme zabrzdíť. Neuvažujeme tiež ani s obmedzeniami, preto nás pri hypotetickom PID šoférovaní nezaujíma či zničíme motor, rešpektujeme predpisy, alebo či vyletíme z cesty. Pri hypotetickom PID riadení by sme mali informáciu iba o aktuálnej chybe riadenia, proporcionálna zložka teda zodpovedá aktuálnej situácii. Mali by sme informáciu aj o priebehu

našej cesty z minulosti, to znamená, že jazdíme dopredu, ale stále sa pozéráme iba do spätného zrkadla. Toto by zodpovedalo integračnej zložke PID riadenia. Jediná informácia, ktorá ukazuje do budúcnosti v PID riadení, zodpovedá derivačnej zložke, ktorá naznačuje aspoň trend vývoja regulačnej odchýlky. V prípade hypotetického PID šoférovania to by mohlo byť napríklad smerovanie kolies. Ďalej by sme nemali žiadne iné podmienky na to ako sa dostaneme do cieľa a nevadilo by ako rýchlo sa tam dostaneme, resp. koľko paliva využijeme.

Aj pri týchto analógiách sme opakovali niekoľko kľúčových pojmov, a to:

- model a stav,
- predikcia a horizont,
- účel a optimalita a
- obmedzenia.

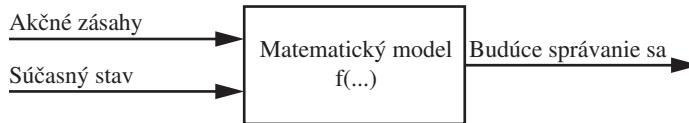
### 1.2.2 Kľúčové pojmy a koncepty

Z predošej diskusie môžeme vyzdvihnuť niektoré kľúčové pojmy a koncepty, ktoré budú detailnejšie analyzované v ďalších kapitolách. Pojmy s ktorými sme sa opakovane stretávali v analógii šachu či šoférovania boli model, predikcia, horizont, účel a optimalita riadenia a obmedzenia. Teraz by sme sa mali posunúť zo sveta analógií a zaradiť si tieto pojmy do širšieho aspektu teórie automatického riadenia.

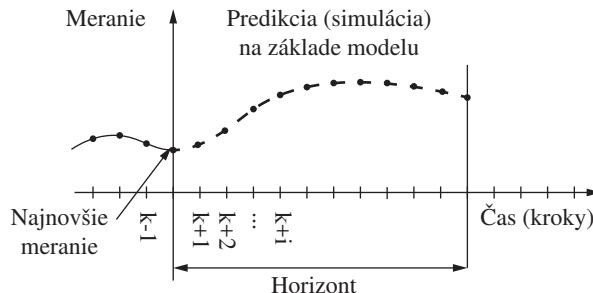
Základom prediktívneho riadenia je znalosť matematického modelu. Model riadeného procesu matematicky opisuje dynamické správanie sa systému a dáva do súvislosti časovo závislú reakciu systému na vstupy. Inými slovami, model jednoznačne vyjadruje cez matematickú funkciu  $f(\cdot)$  aké výstupy môžeme očakávať na snímačoch po aplikácií určitých akčných zásahov (Obr. 1.6). Model môžeme získať pomocou rôznych prístupov, napríklad na základe matematicko-fyzikálnej analýzy alebo experimentálnej identifikácie sústav. Disciplína, ktorá sa zaoberá formuláciou modelov dynamických systémov v automatickom riadení, sa nazýva identifikácia sústav<sup>6</sup>. Kedže stratégia prediktívneho riadenia je založená na modeli, základom kvalitného priebehu riadenia je dobrý model. Základy matematického modelovania dynamických dejov si zopakujeme v nasledujúcej kapitole, teda v Kap. 2.

Proces predikcie môžeme jednoducho považovať za počítačovú simuláciu (Obr. 1.7). Táto simulácia poskytuje časovo závislé správanie sa výstupov, prípadne vnútorných stavov systému, ako reakciu na dané vstupy. Predikciu môžeme realizovať pomocou matematického modelu, určitých informácií zo súčasnosti a na základe známej stratégie určenia vstupov.

<sup>6</sup>angl. system identification



Obr. 1.6. Matematický model systému.

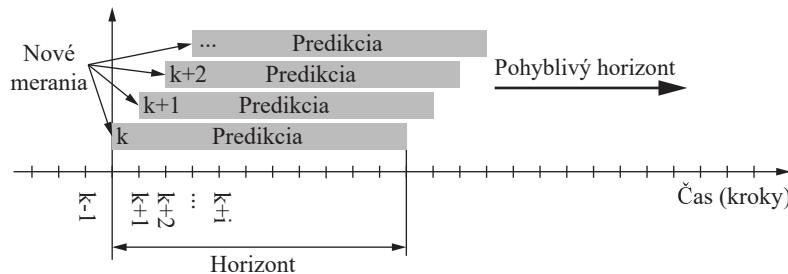


Obr. 1.7. Predikcia (simulácia) správania sa dynamického systému.

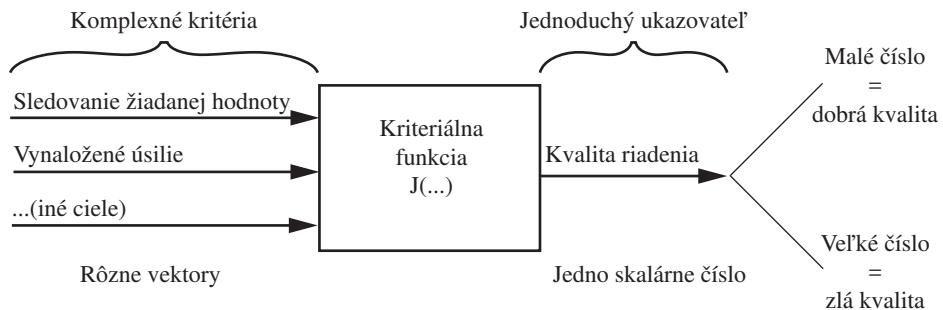
Predikciu (teda našu simuláciu) budeme počítať iba do istého konečného času do budúcnosti. Kedže moderné algoritmy riadenia sú realizované číslicovo na počítačoch, plynutie času taktiež musíme pochopiť diskrétnie. Spojitý čas  $t$  je rovnomerne rozdelený na úseky dĺžky  $T_s$  a potom všetky úkony riadenia, ako je meranie, výpočet a realizácia akčného zásahu sú vykonávané v diskrétnych časových okamihoch  $k$ , t. j.  $t = kT_s$ . Celé číslo, ktoré vyjadruje v akom trvaní do budúcnosti predikujeme, budeme nazývať horizontom predikcie. Predikciu si bližšie predstavíme v Kap. 3.

Je dôležité vedieť, že tento horizont nie je statický. V každej časovej vzorke  $k$  ho totiž posúvame o krok ďalej. Tomu hovoríme, že máme posuvný alebo tiež pohyblivý horizont, a preto občas stratégiu prediktívneho riadenia voláme tiež algoritmus riadenia s posuvným, resp. pohyblivým horizontom. Koncept pohyblivého horizontu je znázornený na Obr. 1.8.

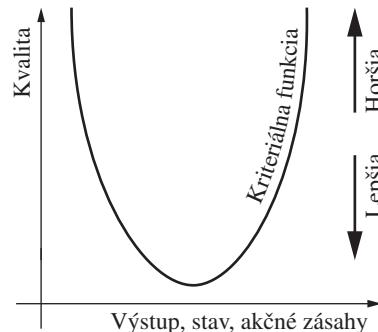
Automatické riadenie procesu alebo systému má zvyčajne viacero cieľov, ktoré môžu byť navzájom aj protichodné. Najčastejšie musíme zabezpečiť, aby výstupy, respektíve vnútorné stavy dynamického systému sledovali predpísaný profil, t. j. naše žiadané hodnoty (referencie). Ďalej je napríklad tiež dôležité aké úsilie musíme vynaložiť pri riadení, a preto samozrejme zohľadňujeme aj vplyv akčných zásahov. Podobné, pomerne abstraktné kritériá musíme jasne a jednoznačne vyjadriť cez matematickú funkciu  $J(\cdot)$  (Obr. 1.9), ktorá svojou hodnotou naznačuje ako blízko, resp. ďaleko sme od dosiahnutia definovaných cieľov riadenia (Obr. 1.10). Takúto funkciu nazývame kriteriálnou funkciou. Kriteriálnu funkciu mô-



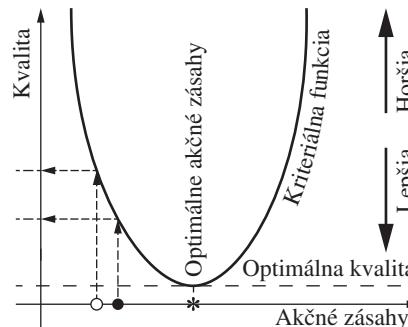
Obr. 1.8. Pohyblivý horizont predikcie v diskrétnom čase.



Obr. 1.9. Kriteriálna funkcia zhrnie viacero veličín vplývajúcich na riadenie do jedného skalárneho indikátora kvality.



Obr. 1.10. Kriteriálna funkcia ako indikátor kvality v zmysle dosiahnutia cieľov riadenia.

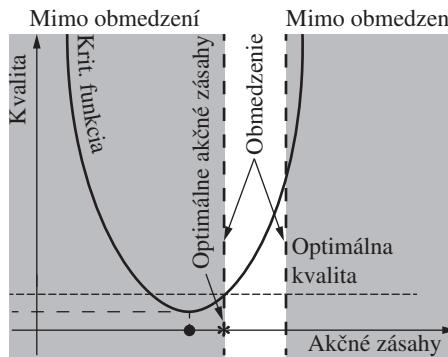


Obr. 1.11. Optimálne akčné zásahy prinášajú najlepšiu možnú kvalitu riadenia pri definovaných podmienkach.

žeme napríklad formulovať tak, aby nižšia hodnota indikovala kvalitnejšie riadenie a naopak vyššia hodnota menej kvalitné riadenie. Ďalej, keďže vieme predikovať vplyv vstupov na dynamiku riadeného systému, vieme aj vypočítať hodnotu kriteriálnej funkcie nie len pre nasledujúci akčný zásah, ale až do konca predikčného horizontu. Vieme teda porovnávať ako rozličné budúce postupnosti akčných zásahov vplývajú na kvalitu riadenia. Zostavenie kriteriálnej funkcie si detailne objasníme v Kap. 4.

V ideálnom prípade dosiahneme najlepšiu možnú kvalitu riadenia, a už vieme, že „dobrá kvalita riadenia“ je definovaná na základe hodnoty kriteriálnej funkcie. Inými slovami, riadenie, ktoré najviac vyhovuje daným podmienkam a definovej kriteriálnej funkcie je optimálne, a keď to vieme zabezpečiť cez daný algoritmus riadenia, hovoríme o optimálnom riadení. Kvalitu vyjadríme cez matematickú funkciu, a teda môžeme hovoriť o optimalizácii aj z matematického hľadiska, pretože hľadáme extrém – minimum alebo maximum – funkcie. Našou úlohou je potom nájsť minimum kriteriálnej funkcie. Keďže hodnota funkcie závisí aj od budúcich akčných zásahov, potom nájdeme aj optimálnu sériu vstupov do riadeného systému.

Táto myšlienka je ilustrovaná aj na Obr. 1.11, kde akýkoľvek iný akčný zásah ako optimálny, t. j. zodpovedajúci minimu kriteriálnej funkcie, by spôsobil horšiu kvalitu riadenia. Akčné zásahy označené plným krúžkom majú za následok lepšiu kvalitu riadenia ako akčné zásahy označené prázdnym krúžkom. Existuje však ešte lepšia kvalita riadenia? Áno, najlepšiu možnú kvalitu riadenia dostaneme keď používame akčné zásahy v bode označenom hviezdičkou. Naša intuitívna otázka, ktorú musí samotný navrhnutý algoritmus zodpovedať je: aké akčné zásahy tvoria najlepšiu možnú kvalitu riadenia? Tento proces musíme zopakovať v každej novej vzorke a prehodnotiť stratégiu na základe najnovších možných meraní. Kriteriálnu



Obr. 1.12. Obmedzenia a hľadanie optimálnej kvality riadenia.

funkciu budeme prvýkrát minimalizovať v neprítomnosti obmedzení v Kap. 5 a vysvetlíme si aj spôsob tvorby obmedzení.

Nemôžeme však používať akčné zásahy v nekonečnom rozsahu, pretože každý akčný člen má svoje obmedzenia. V istých prípadoch musíme dodržať aj technologicke alebo bezpečnostné limity na výstupe riadeného systému, respektíve vo vnútorných stavoch systému. Tieto limity budeme spoločne nazývať ako procesné obmedzenia, respektíve iba obmedzenia. Kľúčovým problémom pri tvorbe riadiaceho algoritmu bude, že musíme dodržať aby obmedzenia boli platné, a to nielen pre nasledujúcu časovú vzorku, ale pre celý horizont. Dodržať to môžeme tak, že s obmedzeniami musíme počítať už pri procese matematickej optimalizácie, a tak hovoríme optimalizácie s obmedzeniami. Optimalizáciu kvadratickej funkcie v prítomnosti obmedzení si preberieme v Kap. 6.

Idea optimalizácie s obmedzeniami<sup>7</sup> je graficky znázornená na Obr. 1.12. Je zrejmé, že najlepšiu možnú – optimálnu kvalitu riadenia by sme dosiahli pri použití akčných zásahov z bodu označenom plným krúžkom. Máme však aj obmedzenia, ktoré určujú priestor dovolených akčných zásahov a všetky zásahy mimo tohto priestoru v našom prípade porušujú obmedzenia. Teda optimálnu kvalitu aj s dodržaním obmedzení dostaneme, ak použijeme akčný zásah presne na hranici obmedzení, označený hviezdičkou. Tento príklad pritom vysvetluje našu motiváciu využiť prediktívne riadenie s obmedzeniami, ako to bolo ilustrované v predošej diskusii a na Obr. 1.2 – najlepšiu kvalitu často dosiahneme len tak, že daný systém riadime na hranici možností, teda práve na obmedzeniach. Ucelený algoritmus prediktívneho riadenia s obmedzeniami si predstavíme v Kap. 7.

Myšlienka riadenia pomocou optimalizácie v prítomnosti obmedzení prináša nielen zlepšenie kvality, ale aj nové výzvy v zabezpečení stability riadenia. Kým

<sup>7</sup>angl. constrained optimization

pri klasických riadiacich prístupoch, ako je napríklad PID riadenie, môžeme vopred vyšetriť stabilitu navrhnutého uzavretého spätnoväzbového obvodu, podobný postup už pri prediktívnom riadení s obmedzeniami nie je možný. Zoberme si ako príklad spätnoväzbový riadiaci obvod, kde môžeme jednoducho matematicky opísať správanie sa uzavretého obvodu a potom vyšetriť umiestnenie pôlov prenosu. Takto môžeme zistif, či náš algoritmický návrh zabezpečí stabilnú odozvu pre každý možný vstup do systému a takto konštatovať, že logicky nie je možné, aspoň pre model obvodu, nestabilné správanie sa. Pri prediktívnom riadení s obmedzeniami takéto vyšetrenie algoritmu nie je možné z rôznych dôvodov, ktoré budeme podrobne analyzovať v ďalšej diskusii. Na úvod si stačí uvedomiť, že vyšetrenie a zabezpečenie stability prediktívneho riadenia nie je triviálna úloha. Stabilitu prediktívneho riadenia si prešetrimo v Kap. 8, kde predstavíme aj riešenie na zabezpečenie stabilného regulačného procesu.

Klúčové pojmy v prediktívnom riadení môžeme zhrnúť takto – prediktívne riadenie:

- používa matematický model;
- predikuje dynamickú odozvu systému, pričom predikcia
  - je vykonaná na danom horizonte do budúcnosti,
  - závisí od modelu, súčasného stavu a budúcich akčných zásahov,
  - je zopakovaná v každom intervale vzorkovania na základe nových dostupných meraní;
- presne definuje kvalitu riadenia, pričom vynaložené úsilie vyjadríme numericky, cez tzv. kriteriálnu funkciu;
- hľadá sériu akčných zásahov, ktoré najviac vyhovujú podmienkam, ktoré sú definované kriteriálnou funkciu a obmedzeniami, teda sú optimálne;
- je zostavená na základe matematickej optimalizácie, čo vykonáme hľadaním najnižšieho extrému – minimalizáciou kriteriálnej funkcie, pričom
  - tento proces je zopakovaný v každom vzorkovacom intervale,
  - čím je hodnota kriteriálnej funkcie nižšia, tým lepšie riadenie máme,
  - a základné poslanie optimalizácie je odpovedať na otázku „*Aká postupnosť budúcich vstupov produkuje najmenšiu hodnotu kriteriálnej funkcie?*“
- procesné obmedzenia vymedzia určité limity riadenia, napríklad na akčné zásahy,

- obmedzenia sú zahrnuté do procesu matematickej optimalizácie, preto hovoríme o optimalizácii s obmedzeniami a
- vyšetrenie a zabezpečenie stability prediktívneho riadenia nie je až taká triviálna úloha ako to je pri klasickom riadení.

### 1.3 Historický prehľad

Niektoré univerzálne vedecké myšlienky z matematiky a geometrie pozná ľudstvo už tisíce rokov. Oproti tomu oblasť techniky známa ako automatické riadenie nemá až takú dlhú a bohatú história. Prvé príklady matematického ošetrenia vlastností systémov so spätnou väzbou – samozrejme hovoríme o mechanických, respektíve pneumatických – siahajú len o niečo viac ako sto rokov do dejín automatizácie. Revolúciou v riadení bolo neskôr rozšírenie číslicových výpočtových zariadení, už môžeme hovoriť o modernom riadení. Prechod od tzv. klasických metód riadenia k modernej teórii riadenia bol však spojený s vývojom stavovej teórie riadenia. Príkladom metódy klasického riadenia je napríklad PID riadenie, ktoré pred rozšírením číslicovej výpočtovej techniky bolo implementované v čisto spojitej analógovej podobe; napríklad pomocou operačných zosilňovačov. Príchod digitálnych počítačov vytvoril jasné hranice: teória riadenia musela prejsť na diskrétné systémy a procesy. Spolu s tým objavili aj prvé „moderné“ algoritmy riadenia, ktoré do istej miery využili aj koncept optimality, napríklad LQ algoritmus riadenia.

Prediktívne riadenie môžeme v istom zmysle zaradiť do ďalšej generácie metód riadenia. Tak, ako to všeobecne platí pre vývoj techniky, idea prediktívneho riadenia nevznikla vo vzduchoprázdnom priestore, ale úzko nadviazala na potreby priemyslu a zrýchlenej evolúcie výpočtovej techniky.

Na jednej strane postupné zvýšenie výpočtového výkonu a zníženie ceny počítačov umožnilo praktickú realizáciu takých riadiacich stratégii, ktoré by predtým boli pamäťovo alebo výpočtovo príliš náročné. Inak povedané, počítač by nedokončil prepočty potrebné na výpočet akčného zásahu pred uplynutím vzorkovacej periody. Istý prelom v pomere ceny a výkonu výpočtových zariadení nastal na konci 70. a začiatku 80. rokov, keď sa objavili prvé počítače pri riadení zložitých procesov, ako napríklad v petrochemickom priemysle.

Na druhej strane na vývoj prediktívneho riadenia vplývali najmä očakávania priemyslu. V záujme každej obchodnej operácie je generovať profit a zlepšenie riadiacej technológie je jedna z možností ako zvyšovať výnosy. Odborníci v procesnom riadení si boli už dlho vedomí toho, že ak chcú zvýšiť kvalitu riadenia, musia používať menej konzervatívne stratégie a dostať sa do prieniku technologických obmedzení. Integrácia procesných obmedzení bola klúčová pri zlepšení výkonu riadenia. Ďalšou motiváciou bolo tiež podržať parametre produkcie alebo

procesu na nezmenenej úrovni a využiť pritom menej energie - ktoré samozrejme môžeme priamo vyjadriť ako finančné úspory. Prediktívne riadenie, ako to bolo predstavené v Kap. 1.2.2, by presne vyriešilo potreby priemyslu: zabezpečilo by optimálny výkon pri dodržaní obmedzení. Prediktívne riadenie je jediný známy spôsob riadenia, ktorý účinne zvládne procesné obmedzenia explicitne a na úrovni algoritmu [101].

Môžeme teda povedať, že motivácia využívať algoritmus prediktívneho riadenia bola jasná, no výpočtovo vykonať optimalizáciu s obmedzeniami v reálnom čase v tej dobe nebolo triviálne. Na jednej strane to vyžadovalo lepšie počítače, na druhej strane bolo možné iba pre procesy s pomalou dynamikou, kde vzorkovacie intervaly boli na úrovni desiatok minút.

Ideálnym prierezom pre implementáciu tejto novej stratégie bol petrochemický priemysel na začiatku 80. rokov minulého storočia, keď sa objavili prvé príklady použitia prediktívneho riadenia v ostrej prevádzke. V petrochemickom priemysle existuje mnoho pomalých procesov, ktoré ponechávajú dostatok času na prepočítanie akčného zásahu. Ďalším dôvodom na rozvinutie prediktívneho riadenia v petrochemickom priemysle je fakt, že aj malé zvýšenie kvality riadenia môže prinášať obrovské úspory na energiách. Predstavme si, že zlepšíme riadenie zložitého petrochemického procesu iba o 1 %. Pri úspore energií to zdanlivo malé zlepšenie pritom môže mať obrovský finančný dopad.

Prediktívne riadenie prešlo procesom postupného vývoja s občasnými veľkými skokmi aj slepými uličkami. Spoločná nomenklatúra zo začiatku ani neexistovala a určite sme nemohli hovoriť o prediktívnom riadení. Niektoré algoritmické formulácie sa najprv vyvinuli nezávisle a na prvý pohľad predstavovali rôzne varianty základnej myšlienky prediktívneho riadenia. Neskôr sa preukázala súvislosť medzi rôznymi stratégiami a dnes už ich poznáme pod spoločným názvom prediktívne riadenie [10]. Na to, aby sme si vedeli umiestniť riadiacu stratégiu, ktorú budeme predstavovať v ďalších kapitolách do histórie vývoja tejto metódy, stručne si ďalej predstavíme niektoré formulácie, ktoré už dnes môžeme považovať za historické.

### 1.3.1 DMC a MAC: riadenie na základe konečnej odozvy

Riadenie s maticou dynamiky<sup>8</sup> (DMC) [37] je založené na modeloch s konečnou prechodovou odozvou<sup>9</sup> (FSR). Algoritmus bol vyvinutý firmou Shell v roku 1979, pričom jeho hlavnou motiváciou bolo najmä riadenie systémov s viacerými vstupmi a výstupmi<sup>10</sup> (MIMO) aj v prítomnosti obmedzení [10]. V roku 1997 už existovalo niekoľko komerčných produktov na základe tohto prístupu [123].

<sup>8</sup>angl. dynamic matrix control (DMC)

<sup>9</sup>angl. finite step response (FSR)

<sup>10</sup>angl. multiple input, multiple output (MIMO)

Model, ktorý prístup DMC uvažuje, je vhodný pre procesný priemysel, pretože je jednoducho identifikovateľný aj pochopiteľný: je to vlastne iba odozva systému na skokovú zmenu, t. j. prechodová odozva. Model tvorí konečná sekvencia hodnôt odozvy na testovací signál. Takéto modely sú ľahko identifikovateľné najmä v prípade priebehov so stabilnou a pomerne nekomplikovanou dynamikou. Príkladom môžu byť tepelné systémy, kde stabilnú odozvu prvého rádu môžeme vyjadriť elementárnym modelom na základe konečnej odozvy. Na druhej strane, ak je systém v otvorenej slučke, t. j. bez riadenia, nestabilný, modely na základe konečnej odozvy nie je možné už identifikovať.

Kým z pohľadu vývoja stratégie prediktívneho riadenia môžeme DMC považovať za súčasť histórie, stále je využívané v priemysle; dokonca DMC je zahrnuté aj do komerčného softvérového produktu a riešenia firmy Aspen Technology (AspenTech) [6]! Metóda DMC je dnes hlavne používaná v aplikáciach s veľkým množstvom vstupov a výstupov, ako aj obmedzení [10].

Kvôli požiadavke na jednoduchú tvorbu modelov sa objavila na svete aj ďalšia formulácia známa ako algoritmicke riadenie na základe modelu<sup>11</sup> (MAC), ktorá využívala modely na základe konečnej impulznej odozvy (FIR)<sup>12</sup>. Podobne ako to je v prípade DMC, na tvorbu modelov stačí systém vybudovať jednoduchým testovacím signálom, v tomto prípade pomocou impulznej zmeny a uložiť prislúchajúcu odozvu v jednotlivých vzorkách. Veľkou výhodou prístupov MAC, respektíve DMC bolo aj to, že kvalita riadenia nebola príliš citlivá na presnosť modelovania.

Z princípu algoritmickeho riadenia na základe modelu boli odvodené aj ďalšie, veľmi podobné riadiace stratégie, ktoré mali svoje vlastné názvy [73, 74] – čiastočne aj z dôvodu firemnej politiky a ochrany duševného vlastníctva – ako napríklad prístup „identifikácie a ovládania“<sup>13</sup> (IDCOM) alebo metóda „heuristické prediktívne riadenie na základe modelu“<sup>14</sup> (MPHC) [128].

### 1.3.2 Zovšeobecnené prediktívne riadenie

Ďalším nezávislým smerom vývoja prediktívneho riadenia bolo v minulosti aj tzv. zovšeobecnené prediktívne riadenie<sup>15</sup> (GPC) [32, 33] a jeho rôzne varianty, ktoré boli vyvíjané v oblasti adaptívneho riadenia. Základným kameňom tohto prístupu bolo zavedenie zvlášť horizontu riadenia a horizontu predikcie. Ako sme už spomenuli, bolo rozšírené aj na adaptívne riadenie, ktoré zvyklo trپet rôznymi nedostatkami, ako napríklad citlivosť na nepresnosť modelu.

<sup>11</sup>angl. model algorithmic control (MAC)

<sup>12</sup>angl. finite impulse response (FIR)

<sup>13</sup>angl. identification and command (IDCOM)

<sup>14</sup>angl. model predictive heuristic control (MPHC)

<sup>15</sup>angl. generalized matrix control (GPC)

Podobne ako pri metódach prediktívneho riadenia DMC a MAC, aj pri zovšeobecnene prediktívnom riadení môžeme hovoriť o rôznych variantoch algoritmu, ktoré nakoniec vyjadrili identický koncept riadenia. Ako príklad môžeme uviesť koncepty ako je [74, 73] „adaptívne riadenie s rozšíreným horizontom“<sup>16</sup> (EHAC) [146], „samoadaptívne riadenie s rozšírenou predikciou“<sup>17</sup> (EPSAC) [39], alebo „prediktívne riadenie s dlhým dosahom“<sup>18</sup> (LRPC) [35]. Po podrobnejšej analýze je zrejmé, že tieto prístupy sú veľmi blízke metódam DMC a MAC.

Na rozdiel od DMC a MAC, metódy GPC už nevyužívajú jednoduché konečné odozvy, FIR alebo FSR, ale takzvané autoregresné modely s klzavým priemerom<sup>19</sup> (ARMA) alebo riadené autoregresné modely s integrovaným klzavým priemerom<sup>20</sup> (CARIMA). Takéto modely berú do úvahy aj náhodné, tzv. stochasticke zložky časových radov a sú pomerne ľahko identifikovateľné pomocou niekoľkých dobre známych algoritmov. Napriek tomu už nie sú až také intuitívne ako FIR alebo FSR modely. Využitie modelov ARMA a CARIMA zdedil GPC hlavne z adaptívneho riadenia, kde sú takéto formulácie dynamiky dejov často využívané.

Metóda zovšeobecnene prediktívneho riadenia na začiatku neuvažovala vo svojej formulácii ani procesné obmedzenia. GPC bola naviac pôvodne založená na modeloch s jedným vstupom a jedným výstupom<sup>21</sup> (SISO), čo nie je vhodné pre potreby procesného priemyslu. Neskôr sa objavilo aj rozšírenie GPC pre systémy s viacerými vstupmi a viacerými výstupmi [132, 79], ale jeho praktická realizácia bola príliš zložitá na široké uplatnenie v praxi.

### 1.3.3 Riadenie s pohyblivým horizontom

Tretím smerom, respektíve formuláciou, ktorú nemôžeme z historického prehľadu vyniechať, je metóda riadenia s pohyblivým horizontom<sup>22</sup> (RHC) [88, 89, 86], ktorá bola na začiatku stvorená v rámci akademického výskumu [74] a priemyslom bola prijatá iba neskôr. Základom RHC je LQ riadenie, ktoré na výpočet optimálnych vstupov minimalizuje kriteriálnu funkciu a využíva stavovú reprezentáciu systému ako matematický model dynamiky dejov. Spočiatku neboli uvažované procesné obmedzenia a formulácia RHC, vhodná nielen pre reguláciu, ale aj na sledovanie žiadanej hodnoty, bola taktiež založená iba neskôr.

<sup>16</sup>angl. extended horizon adaptive control (EHAC)

<sup>17</sup>angl. extended prediction self-adaptive control (EPSAC)

<sup>18</sup>angl. long range predictive control (LRPC)

<sup>19</sup>angl. autoregresive moving average (ARMA)

<sup>20</sup>angl. controlled autoregresive integrated moving average (CARIMA)

<sup>21</sup>angl. single input, single output (SISO)

<sup>22</sup>angl. receding horizon control (RHC)

### 1.3.4 Zjednotenie prístupov prediktívneho riadenia

Na začiatku boli vyššie uvedené smery vývoja teórie prediktívneho riadenia od seba úplne nezávislé. Neskoršie, na začiatku 90. rokov minulého storočia, sa objavili prvé analýzy vzájomného vzťahu rôznych formulácií prediktívneho riadenia [15, 135, 87, 118, 147]. Postupne bolo zistené, že metóda riadenia pohyblivým horizontom opisuje koncept prediktívneho riadenia všeobecne a ostatné prístupy ako DMC, MAC alebo GPC sú iba jeho špeciálnymi prípadmi. Ďalej bolo zistené, že stavová reprezentácia dynamiky riadenej systému, ktorú uvažuje RHC, je najvhodnejší typ modelu z hľadiska univerzálnosti; napriek tomu jednoduchšie modely sú stále atraktívne z pohľadu ich výpočtovej nenáročnosti. Všetky teoretické rozšírenia a zlepšenia, ktoré boli pôvodne vytvorené pre špecifickú historickú formuláciu, sú pritom univerzálne platné aj pre ostatné verzie prediktívneho riadenia.



Rozdiely medzi historickými formuláciami sú nepodstatné a dnes už uvažujeme iba zjednotený koncept prediktívneho riadenia.

Dnes si už pod pojmom prediktívne riadenie, resp. prediktívne riadenie na základe modelu (MPC) predstavujeme zjednotený a všeobecný prístup automatického riadenia. Historické pomenovanie podoby algoritmov MPC však stále žije najmä kvôli konzervativizmu priemyslu. Napriek tomu, rozdiely medzi spomínanými metódami sú nepodstatné a založené iba na type použitých matematických modelov. Keďže z teoretického hľadiska je stavová reprezentácia najviac vhodná pre všeobecnenú diskusiu, budeme ďalej v tejto učebnici uvažovať výlučne stavové modely.



Aby sme zachovali všeobecnosť metódy, budeme ďalej na opis dynamiky systému uvažovať výlučne stavové modely.

Posledným dôležitým miľníkom vo vývoji prediktívneho riadenia bola formalizácia podmienok stability na začiatku nového tisícočia [107]. Dnes už preto môžeme vytvoriť prediktívny regulátor, ktorý má stabilitu riadenia v uzavretej slučke zaručenú.

## 1.4 Využitie prediktívneho riadenia

Prvé varianty prediktívnych algoritmov boli využívané prakticky výlučne v petro-chemickom priemysle. Ako to už bolo spomenuté, samotná metóda bola vyvinutá čiastočne kvôli potrebám procesného priemyslu, kde zvýšenie kvality riadenia systémov s mnohými vstupmi a výstupmi a v prítomnosti obmedzení bolo primárnou motiváciou.

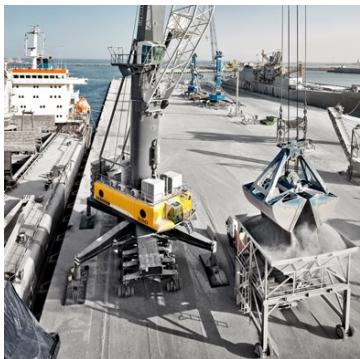
Dnes sa už dôsledkom vývoja výpočtových prostriedkov MPC posúva aj do iných aplikácií s rýchlo dynamikou, ako sú napríklad rôzne mechatronické aplikácie, robotika, automobily, lietadlá, vesmírne zariadenia, medicína atď. Prediktívne riadenie je tak využívané aj v netradičných aplikáciach, ako je napríklad optimalizácia spotreby v inteligentných elektrických rozvodov<sup>23</sup> alebo aj na optimalizáciu priemyselného rybolovu.

Po troch desaťročiach je prediktívne riadenie pre určité praktické priemyselné aplikácie už pomerne vyspelé a nepotrebuje z teoretického hľadiska zásadný vývoj; žiaduce je skôr zlepšenie a zjednodušenie využitelnosti a implementácie algoritmov [49]. Funkcie ktoré sú dôležité pre priemyselné aplikácie, umožňujú implementáciu a ladenie algoritmov prediktívneho riadenia pomocou menej špecializovaných pracovníkov, ďalej ponúkajú automatickú identifikáciu modelov, monitorovanie a diagnostiku priebehov, alebo zjednodušujú údržbu [49]. Zlepšenie kvality riadenia zložitých systémov s mnohými vstupmi a výstupmi a obmedzeniami dosiahnuté pomocou MPC je už samozrejmostou – firmy už vyžadujú intuitívnu a jednoduchú realizáciu, ako aj bezproblémovú údržbu s minimálnym počtom zásahov do chodu procesu.

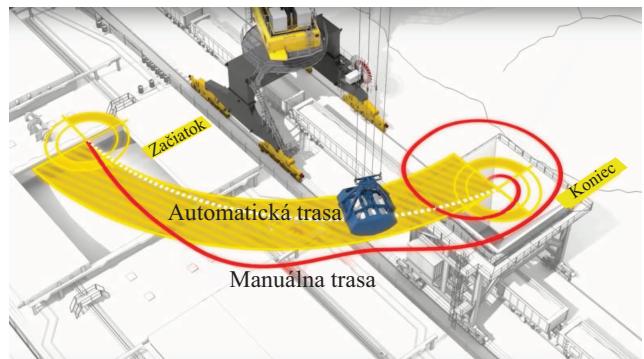
Napriek tomu, že ambíciou tejto knihy bude naučiť čitateľa podrobnosti návrhu MPC, existujú aj hotové softvérové riešenia pre koncových užívateľov na báze algoritmu prediktívneho riadenia. Tieto nástroje sľubujú jednoduché zavedenie algoritmu MPC do priemyselných procesov. Medzi firmy, ktoré ponúkajú softvér na MPC riadenie patria napríklad Honeywell, ABB, Invensys, Rockwell, AspenTech, atď. Napríklad v promočných materiáloch softvéru DMCPplus, ktorý využíva riadenie s maticou dynamiky, firma AspenTech sľubuje zvýšenie produktivity výroby o 5 % a redukciu spotreby energií o 10 % [6].

---

<sup>23</sup>angl. smart grid



(a) Žeriav Liebherr LHM550 [93].



(b) Minimalizácia výkyvu nákladu [92].

Obr. 1.13. Ilustrácia prístavného žeriavu LIEBHERR LHM550 a technológie „Cycoptronic“ na optimálne plánovanie trasy na báze prediktívneho riadenia.

## 1.5 Vybrané príklady využitia MPC

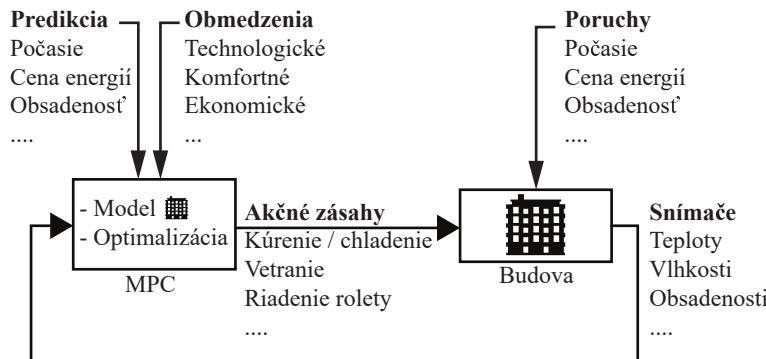
### 1.5.1 Priemyselné aplikácie

V nasledujúcich sekciách uvádzame niekoľko vybraných príkladov využitia prediktívneho riadenia v niektorých praktických aplikáciách. Hoci vo využití MPC stále dominuje procesný priemysel, ukážeme si aj niektoré zaujímavé progresívne oblasti využitia tejto stratégie riadenia.

#### Žeriavy

MPC je využívané na stabilizáciu nákladu v rôznych žeriavoch a zdvíhacích zariadeniach [4, 66]. Pri rýchlych posunoch sa bremeno spojené s otočným ramenom pomocou lana môže kývať a oscilovať, čím sa spomaľuje práca a potenciálne ohrozenie bezpečnosť, ak žeriavnik nie dostatočne skúsený.

Mobilné prístavné žeriavy série LHM spoločnosti Liebherr ponúkajú voliteľnú technológiu „Cycoptronic“ na stabilizáciu nákladu [94]. Marketingový názov pritom skrýva stratégiu prediktívneho riadenia. Náklon lana je meraný gyroskopom, respektíve zrýchlenie lana pomocou akcelerometra. Merania sú pritom rekonštrúované a zlúčené pomocou pozorovateľa. Na základe dynamiky systému – ktorá je podobná dynamike matematického kyvadla – sa predikuje poloha nákladu a žeriav sám koriguje nechcené výkyvy bremena (Obr. 1.13).



Obr. 1.14. Ilustrácia schémy energetického riadenia budovy pomocou MPC.

V záujme ochrany duševného vlastníctva si množstvo firiem volí marketingový názov na základe niektoj vyspej funkcionality svojho produktu, ktorý inak využíva algoritmus prediktívneho riadenia.

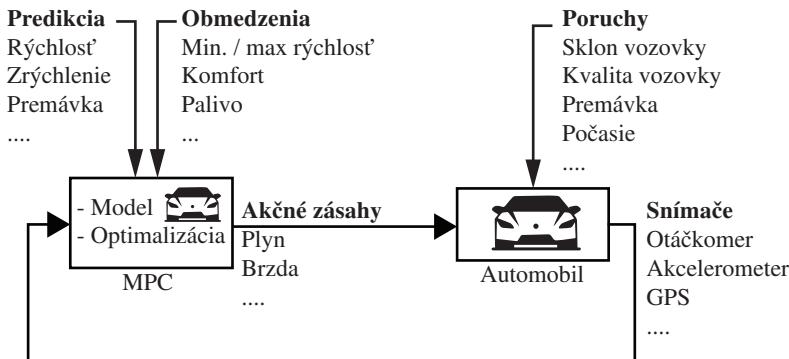


### Energetické riadenie budov

Skoro 40 % spotreby energie globálne využijeme v rôznych budovách [20]. Úspory energie môžeme okrem mechanických úprav (zateplenie a pod.) zvýsiť investíciou do hardvéru, ako sú modernejšie čerpadlá, výmenníky, klimatizácie, atď. Žiaľ, takéto zmeny sú finančne náročné. Alternatívnym spôsobom úspory energie, a teda aj spôsobom šetrenia životného prostredia, je použitie lepších riadiacich stratégií. Modernejšie algoritmy riadenia môžu znamenať značné ušetrenie finančných nákladov bez nutnosti obrovskej investície do samotnej infraštruktúry budovy. Prediktívne riadenie môžeme využiť na energetické riadenie budovy [20, 99, 100] tým, že:

- modelujeme vonkajšie zmeny počasia a obsadenosť budovy,
- využijeme akumuláciu tepla v budove
- môžeme predikovať cenu a spotrebu elektrickej energie,
- obmedzenia: min./max. teplota, osvetlenie atď.

Energetické riadenie vykurovania a chladenia budovy je schematicky znázornené na Obr. 1.14.



Obr. 1.15. Ilustrácia schémy adaptívneho tempomatu na báze MPC.

### Vylepšený tempomat a autonómne vozidlá

Príkladom využitia MPC je napríklad realizácia adaptívneho regulátora rýchlosťi vozidla – adaptívneho „tempomatu“<sup>24</sup> [20].

Základom tejto myšlienky je rozšírenie známej funkcie tempomatu v modernom automobile predpovedou hustoty cestnej premávky. Riadiaca jednotka dostáva informácie o zápchach, ktoré dynamicky vznikajú počas jazdy. Cieľom je pri rešpektovaní okolitej premávky nastaviť žiadanú rýchlosť vozidla tak, aby sa reálna rýchlosť vozidla pohybovala medzi maximálnou a minimálnou dovolenou hodnotou a pritom spotreba paliva bola minimálna. Môžeme si predstaviť situáciu, že na plánovanej trase je o niekoľko sto kilometrov premávka spomalená dôvodom nehody. Prečo by potom vozidlo malo jazdiť plnou rýchlosťou? Spomalením auta tak môžeme ušetriť na palive a vyhnúť sa tiež čakaniu v zápche.

Algoritmus prediktívneho riadenia je priam stvorený na podobnú úlohu (Obr. 1.15). Riadiaci systém predikuje rýchlosť vozidla na základe dynamického modelu, prípadne do predikcie skombinuje sklon cestnej komunikácie alebo vplyv počasia. Systém dodržiava procesné obmedzenia ako sú napríklad predvolené maximálne a minimálne rýchlosťi a optimálne nastavuje žiadanú rýchlosť podľa rýchlosťi sámotej premávky. Takýto systém pritom opäť minimalizuje matematickú funkciu, ktorá vyjadri (hlavne) spotrebu vozidla a podlieha obmedzeniam.

Známe firmy ako Volvo, Ford alebo Hyundai využívajú systémy na autonómne šoférovanie auta pomocou algoritmu prediktívneho riadenia [57, 46]. Iné firmy, v poslednej dobe najmä Tesla, zase nepublikujú svoje výsledky v akademických časopisoch, aby chránili svoju technológiu; no môžeme predpokladať, že na tvorbu softvéru používajú základnú myšlienku ohraničenej optimalizácie v reálnom čase.

<sup>24</sup>angl. automatic cruise control, (ACC)

Riešenia využívajú model vozidla a predikciu trasy pri rešpektovaní okolitej premávky. Algoritmus potom riadi brzdy aj volant, a tak sa vozidlo pohybuje bez ľudského zásahu alebo s minimálnym zásahom šoféra. Experimentálne výsledky svedčia o tom, že schopnosti autonómneho šoférovania v niektorých prípadoch prevyšujú možnosti ľudí: pri autonómnom šoférovaní na ľade boli dokonca dosiahnuté rýchlosť nad 70 km/h [46].

### 1.5.2 Akademické aplikácie

V nasledujúcich sekciách uvádzame niekoľko zaujímavých aplikácií stratégie prediktívneho riadenia v štádiu akademického výskumu. Kým niektoré štúdie majú charakter, kde neskôr sie potenciálne využitie pre reálne systémy je zrejmé, iné skôr z pohľadu overenia funkčnosti algoritmov používajú laboratórne zariadenia. Takéto „hračky“ umožnia vývoj a porovnávanie riadiacich metód a napriek tomu, že v komerčnom slova zmysle nepredstavujú žiadnu reálnu aplikáciu, majú dynamický charakter, ktorý je vo všeobecnosti veľkou výzvou pre riadenie.

Akademické aplikácie prediktívneho riadenia mnohokrát využívajú abstraktné systémy, ktoré nemajú bezprostredné reálne, respektíve komerčné využitie, ale svojou komplexnou dynamikou slúžia najmä na vývoj a zdokonalenie riadiacich stratégii.



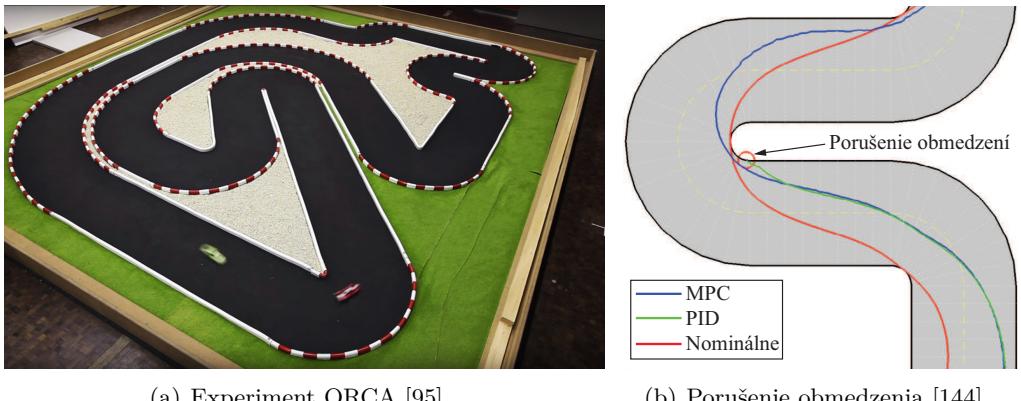
#### Preteky s autíčkami

Veľmi zaujímavé sú tiež preteky s RC<sup>25</sup> autíčkami riadenými v reálnom čase, ktoré pomocou prediktívneho riadenia realizuje projekt ORCA (Optimal RC Racing) na univerzite ETH Zürich. Projekt používa modifikované autá na diaľkové ovládanie z takzvanej kategórie „dnano“ v mierke 1:43. Jedno takéto autíčko váži iba 50 g a zahrňa v sebe mikroradič z rady ARM Cortex M4 [96, 144].

Modely vozidiel sú ovládané na hrane ich fyzikálnych možností, čo znamená, že sily na hnacích kolesách sú plne nasýtené – inými slovami *driftujú* v zákrute. Dosahujú rýchlosť 3 m/s, čo možno nezníe ako veľa, ale pri mierke 1:43 to predstavuje viac ako 450 km/h [96, 145]!

Autá nemajú vlastné senzory – ich polohu sleduje a vyhodnocuje infračervená kamera umiestnená nad závodnou dráhou. Samotné auto neprepočíta akčné zásahy „on board“, algoritmus prediktívneho riadenia namiesto toho beží na centrálnom počítači, ktorý potom len posiela akčné zásahy bezdrôtovo. Cieľom je

<sup>25</sup>angl. remote control (na diaľkové ovládanie)



Obr. 1.16. Ilustrácia závodnej trate pre modely autíčiek v rámci projektu ORCA a porušenie obmedzení (opustenie trate) cez autonómne riadenie pomocou PI(D) algoritmu.

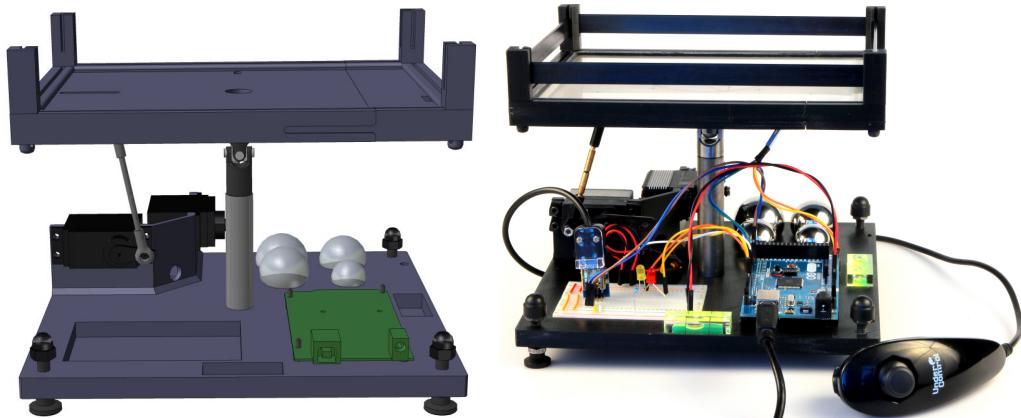
prejsť závodnú dráhu čo najrýchlejšie a dodržať pritom obmedzenia: zostať na danej dráhe a nedostať sa do kolízie so súpermi [96].

Pri úlohe sledovania predpísanej dráhy sa jasne ukázali výhody stratégie prediktívneho riadenia oproti PI riadeniu. Kým pri PI riadení auto viackrát v zákrute narazilo do mantinelu, respektívne vyletelo z dráhy, MPC pomohlo dosiahnuť optimálne výsledky a prejsť dráhu za časy, ktoré by nebolo možné dosiahnuť pomocou manuálneho riadenia alebo iných metód riadenia [145, 144].

### Gulôčka na ploche

Predstavme si plochu, ktorá môže byť naklonená pomocou servomotorov na žiadany uhol. Na ploche je voľne položená gulôčka; jej polohu môžeme sledovať pomocou dotykovej plochy, respektívne cez kamerový systém (Obr. 1.17). Takýto systém je často využívaný v laboratóriách automatického riadenia, pretože sledovanie predpísanej trajektórie na ploche gulôčkou nie je triviálnou úlohou [63].

Keď plochu trochu nakloníme, tak gulôčka v absencii fyzických zábran spadne na zem; čo je skvelý príklad systému, ktorý je v otvorennej slučke – to znamená bez riadenia – nestabilný. Navyše si treba uvedomiť, že gulôčka nemá vlastný pohon, a môžeme ju teda riadiť iba cez naklonenie plochy. Akonáhle hovoríme o súvislosti naklonenia plochy a dynamiky gulôčky, do matematickej reprezentácie vstupujú trigonometrické transformácie vstupu, a preto bude model nelineárny. Aby tieto fažkosti nestačili, gulôčka sa pohybuje na ploche rýchlo, a preto musíme



Obr. 1.17. Ilustrácia laboratórneho systému „Guľôčka na ploche“ [63].

uvažovať rýchle vzorkovanie. Je zrejmé, že aplikácia prediktívneho riadenia na podobné akademické „hračky“ nie je jednoduchá a môžeme pomocou nich efektívne zdokonalovať a porovnávať riadiace stratégie.

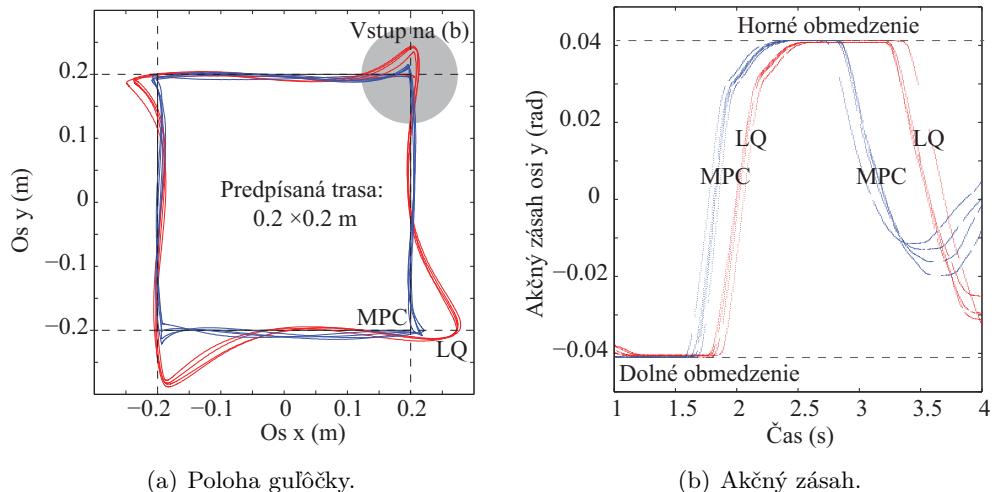
Ak by ste si mysleli, že riadenie systému guľôčka na ploche nemá reálne reálne využitie, stačí uvažovať až neuveriteľný manéver pristátia postranných a centrálnych rakiet série Falcon, spoločnosti SpaceX (Obr. 1.18). Stačí použiť trochu fantázie a inžinierskej intuúcie a hned vidíme, že z pohľadu riadenia sa tieto problémy podobajú.

K riadeniu systému môžeme pristúpiť tak, že pôvodný nelineárny systém linearizujeme pre malé uhly a dostaneme systém štvrtého rádu s dvoma vstupmi, jeden pre každú os [20]. Obmedzenia môžu byť rôznorodé: napríklad signál do motorov (vstupné), uhol naklonenia plochy (stavové), alebo dokonca aj limitovaný rozmer plochy.

Obr. 1.19 ilustruje rozdiel medzi LQ metódou a MPC riadením v úlohe sledovania predpísanej trajektórie guľôčky na ploche [141], pri porovnatelnom nastavení oboch regulátorov. Guľôčka musí pri experimente sledovať žiadanú trajektóriu v tvare  $0.2 \times 0.2$  m veľkého štvorca na ploche. Akčné zásahy sú naklonenia plochy v smere x-ovej osi a y-ovej osi. Na Obr. 1.19(a) vidíme skutočnú dosiahnutú trasu, kde je jednoznačné, že víťazom je stratégia prediktívneho riadenia, ktorá lepšie sleduje predpísanú trajektóriu ako LQ. Obr. 1.19(b) dáva akčný zásah na y-ovej osi, keď guľôčka prechádza cez pravý horný roh trajektórie. Vidíme, že naklonenie plochy je obmedzené na  $0.04$  rad ( $2.3^\circ$ ). Z uvedených priebehov je jasné, že účinkov MPC stratégie na vstupy sa objaví o približne  $0.2$  s skoršie ako v prípade LQ, pretože prediktívne riadenie „vidí“ nasledujúce zmeny na horizonte predikcie.



Obr. 1.18. Charakter riadenia a dynamické správanie sa pristávajúcej rakety sa podobajú známemu experimentu „gulôčka na ploche“ [136].



Obr. 1.19. Ilustrácia výkonu porovnateľne naladeného LQ a MPC riadenia systému „gulôčka na ploche“. (a) znázorňuje reálnu trajektóriu guľôčky, (b) príslušný akčný zásah [141].

## Kvadrokoptéry

Autonómny let kvadrokoptér je taktiež typická oblast nasadenia vyspelých prediktívnych algoritmov [20]. Kvadrokoptéry majú pomerne veľký výkon oproti svojej váhe, sú agilné a vďaka rýchlej rotačnej dynamike môžu vykonať aj skoro okamžité posuvné pohyby. Akčné členy sú motory poháňajúce štyri alebo viac vrtuľ, pričom akčné zásahy vykonáme pomocou zmeny otáčok motorov. Cieľom riadenia môže byť autonómne sa vznášať vo vzduchu, sledovať preddefinovanú trajektóriu, či vykonávať rôzne zložité manévre [20].

Kvadrokoptéra je nelineárny systém so šiestimi stupňami voľnosti, t. j. poloha a orientácia v 3D. Pre takéto systémy môžeme definovať rôzne obmedzenia: napríklad obmedzený tah motorov, zmenu rýchlosťi alebo hranice letového priestoru. Riadenie letu všeobecne zahŕňa mnoho výziev, ako napríklad rýchlosť a nelineárnu dynamiku, prepojenosť vnútorných stavov modelu a efektu akčných zásahov, respektíve nestabilný systém v otvorennej slučke.

## 1.6 Budúcnosť prediktívneho riadenia

Prediktívne riadenie je stále intenzívne skúmaná metóda riadenia. Dodnes nemôžeme tvrdiť, že ide o nemennú špecializáciu v rámci automatického riadenia. Niektoré otázky stále nie sú úplne prešetrené a vyžadujú ďalší intenzívny výskum. Výskum prebieha prevažne v akademickom svete a často možno pôsobí aj abstraktne, no motiváciu vždy čerpá z praktických výziev a problémov v priesmyse.

V budúcnosti sa musí zdokonaliť najmä prediktívne riadenie:

- nelineárnych systémov,
- systémov s rýchloou dynamikou,
- na vnorených mikroradiičoch,
- systémov s neurčitosťami,
- ekonomicky kritických procesov,
- rozložených systémov atď.

Ďalej si preto stručne predstavíme spomenuté výzvy pre ďalšiu budúcnosť algoritmov prediktívneho riadenia.

### 1.6.1 Nelineárne MPC

Najjednoduchšou a najčastejšou matematickou reprezentáciou dynamických dejov sú lineárne modely. Linearita pre našu praktickú definíciu znamená, že výstupy

modelu sú lineárhou funkciou vstupov, teda podliehajú vlastnosti superpozície a homogeneity. Superpozícia znamená, že sčítaním dvoch výstupov modelu s dvoma rôznymi vstupmi dostaneme taký istý výsledok ako keby sme vypočítali výstup kombináciou vstupov. Homogenita znamená, že ak vstupy zväčšíme (zmenšíme), tak výstupy sa taktiež zväčšia (zmenšia) v rovnakom pomere.

V mnohých prípadoch môžeme predpokladať, že lineárne modely postačujú na predikciu, no niektoré dynamické dej sú už zložitejšie a vyžadujú prirodzene nelineárne modely. Stačí ak matematický opis obsahuje násobenie jedného stavu s druhým, trigonometrické funkcie, alebo nespojitosť a už máme nelineárny model. V takomto prípade môžeme voliť stratégiu, že ho linearizujeme v blízkosti pracovného bodu, alebo používame po častiach lineárny model ktorý definuje lineárny ekvivalent pôvodného modelu v istých oblastiach pracovného režimu systému.

Takéto stratégie ale majú svoje obmedzenia a v niektorých prípadoch sme nútene využívať dynamické modely, ktoré sú prirodzene nelineárne. V takomto prípade však značne skomplikujeme návrh prediktívneho riadenia a môžeme hovoriť o nelineárnom MPC (NMPC)<sup>26</sup>.

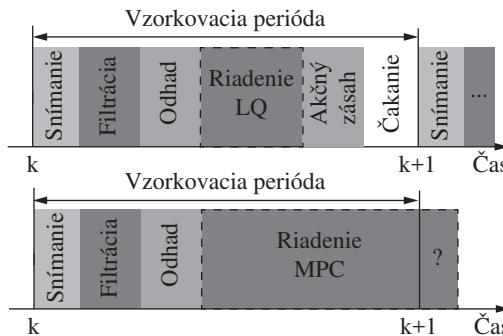
Hlavný problém spočíva v tom, že využitie nelineárneho predikčného modelu často vedie na nekonvexný optimalizačný problém. Riešenie takéhoto problému je oveľa náročnejšie ako riešenie konvexného problému, trvá viac času a vyžaduje zložitejší matematický aparát, ktorý musí dané hardvérové zariadenie zároveň zvládnuť. Ďalej, dosiahnutie použiteľného riešenia vôbec nie je garantované, pretože algoritmus môže naraziť aj na lokálne riešenia a ukončiť optimalizáciu na nesprávnom mieste. Veľké problémy prináša aj otázka stability riadiacej slučky, pretože zaručenie stability sa stáva netriviálnou úlohou.

### 1.6.2 MPC pre rýchlu dynamiku

Pre riadené systémy s pomalou dynamiku, ako napríklad určité petrochemické procesy, potrebujeme pomalé vzorkovanie, niekedy aj na úrovni desiatok minút alebo dokonca hodín. V každej číslicovej implementácii spätnovázobného riadenia je základnou otázkou, či počítač dokáže vyhodnotiť všetky úkony potrebné na výpočet akčného zásahu v reálnom, na tom určenom čase.

V diskrétnom čase musíme ešte pred ukončením danej vzorkovacej periody získať merania zo snímačov, prípadne odfiltrovať šum, odhadnúť stav, vypočítať vstupy a poslat ich na akčné členy (Obr. 1.20). Kým úlohy snímania alebo poslania vstupov na akčné členy sú rovnaké bez ohľadu na zvolenú riadiacu stratégiu, čas potrebný na výpočet akčného zásahu pri MPC je kvôli riešeniu ohraničenej optimalizačnej úlohy vo všeobecnosti rádovo dlhší ako napríklad pri nominálnom LQ riadení.

<sup>26</sup>angl. nonlinear MPC (NMPC)



Obr. 1.20. Riadenie musí prebiehať v reálnom čase, pričom akčný zásah musí byť k dispozícii ešte pred koncom daného vzorkovacieho intervalu.

Výzvou pre budúcnosť je preto tiež riadenie systémov, ktoré majú rýchlu dynamiku, a teda potrebujú veľmi krátke vzorkovacie intervaly. Procesy prebiehajúce v motore automobilu, letová dynamika lietadla alebo vrtuľníka, vibračné a akustické javy potrebujú vzorkovanie na úrovni pod tisícinou sekundy; inými slovami frekvencia vzorkovania, a teda aj výpočtov dosahuje desiatky kHz.

Podstatnou otázkou potom je, či výpočtové zariadenie dokáže prepočítať algoritmus v reálnom čase. Pre spomenuté systémy je dnes problém implementovať aj MPC s malým lineárny modelom, nehovoriac o rádovo komplexnejších lineárnych či nelineárnych modeloch. Dnes je preto skoro nemysliteľné uvažovať ešte zložitejšie logické usporiadania MPC ako sú napríklad adaptívne alebo robustné verzie v reálnom čase.

Tieto výzvy samozrejme nie sú neriešiteľné. Vplyvom vývoja výpočtovej techniky máme čoraz výkonnejšie prostriedky automatického riadenia, vrátane rýchlejších procesorov a väčšej pamäti. Mooreov zákon nie je jedinou formou záchrany, pretože existuje mnoho vynaliezavých teoretických a matematických riešení základného problému MPC, ktoré umožňujú tlačiť hranice tohto prístupu ďaleko za dnes dostupné možnosti.

### 1.6.3 Explicitné MPC

Ako si ukážeme v nasledujúcich kapitolách, akčný zásah pomocou prediktívneho riadenia tradične získavame riešením príslušnej optimalizačnej úlohy s obmedzeniami v každom vzorkovacom intervale pre aktuálnu nameranú alebo odhadnutú hodnotu stavu. Riešenie tejto úlohy sa teda v reálnom čase priebežne, t. j. online opakuje. Je však tiež dobre známe, že daný optimalizačný problém možno vyriešiť aj vopred, t. j. offline, pre všetky stavy splňajúce definované obmedzenia.

Tento prístup prediktívneho riadenia je nazývaný explicitné MPC, pričom prístup priebežného riešenia MPC problému zvykneme nazývať tiež implicitné MPC.

Explicitné MPC na riešenie daného optimalizačného problému používa techniku parametrického, resp. multiparametrického programovania [11, 13]. Riešením je tzv. partícia, t. j. rozdelenie stavového priestoru na určitý počet polyedrických regiónov a prisľúchajúce optimálne vstupy v tvare tzv. po častiach afinnej (PWA, zovšeobecnenie lineárnej) funkcie<sup>27</sup>. Kedže riešenie v uvedenom tvare je získané offline, implementácia v reálnom čase pozostáva z vyhľadania regiónu, v ktorom sa nachádza aktuálny stav systému a vyčíslenia príslušnej afinnej funkcie pre daný stav. Výsledkom je optimálny akčný zásah, ktorý je aplikovaný na riadený systém. Je zrejmé, že implementácia explicitného MPC je vo všeobecnosti oveľa jednoduchšia ako výpočtovo náročné riešenie implicitného MPC problému.

Z uvedeného je však tiež zrejmé, že na pohľad jednoduché vyhľadávanie stavu v rámci stavovej partície môže byť časovo náročné v prípade systemov vyšších rádov a partícií s rádovo tisíckami polyedrických regiónov a zároveň môže mať aj veľké požiadavky na pamäť na uloženie riešenia na danom hardvéri. Z tohto dôvodu sa výskum v tejto oblasti aktívne venuje vývoju techník pre časovo a pamäťovo efektívnu implementáciu explicitného MPC v reálnom čase [83, 84, 61].

#### 1.6.4 Vnorené MPC

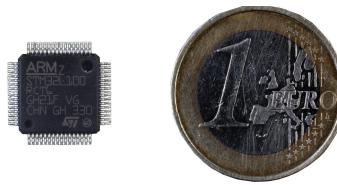
Prediktívne riadenie je prevažne aplikované na veľké a komplexné systémy, kde riadiace vstupy sú určované v reálnom čase na výpočtových zariadeniach, ktoré sú porovnatelne veľké a výkonné ako stolové počítače. V takomto prostredí nie je dôležitým faktorom cena výpočtového zariadenia, ani spotreba elektrickej energie alebo fyzické rozmery či hmotnosť hardvéru.

V poslednom čase však môžeme vidieť čoraz viac rôznych aplikácií pre MPC, ktoré sú fyzicky menšie ako procesy v priemysle. Pre takéto využitie musíme nahradíť počítače v tradičnom zmysle za miniatúrne výpočtové zariadenia, takzvané mikroradiče, ktoré v sebe zahŕňajú všetky potrebné komponenty na výpočet algoritmu riadenia. Mikroradiče sú pritom hlboko integrované, inými slovami *vnorené*<sup>28</sup> do riadeného systému. Na rozdiel od PC, mikroradiče sú rádovo lacnejšie, malé a majú malú spotrebu elektrickej energie (Obr. 1.21).

Značnou nevýhodou mikroradičov je ale ich obmedzený výpočtový výkon, malá operačná pamäť a málo miesta na uloženie algoritmu riadenia. Toto, kombinované so systémami, ktoré majú rýchlu dynamiku, predstavuje značné implementačné ťažkosti. Otázkou potom je, či vôbec vieme nahrať a skladovať algoritmus predik-

<sup>27</sup>angl. piecewise affine (PWA) function

<sup>28</sup>angl. embedded



Obr. 1.21. Mikroradič (STML100) v porovnaní s veľkosťou 1 € mince.

tívneho riadenia vo flash pamäti vnoreného zariadenia, ktorý stojí približne 1 €? Má takéto zariadenie dostatočne veľkú operačnú pamäť (RAM) na to, aby vedelo manipulovať s premennými? A ešte väznejšou dilemom je, či zariadenie vôbec stihne ukončiť výpočet akčného zásahu v rámci jedného vzorkovacieho intervalu.

Samozrejme, zvýšenie výpočtovej kapacity hardvéru je iba jedným z možných riešení. Niektoré vnorené aplikácie môžu obetovať malú mieru riadiaceho výkonu MPC v záujme získania určitých zjednodušení v algoritme, ktoré zrýchlia výpočty, respektíve zmenšia pamäťové nároky [105]. Riešenie základného optimalizačného problému MPC má však v sebe potenciál na určité matematické ale aj programátocké triky na zníženie výpočtového výkonu. Pri implementácii MPC na vnorené systémy pritom musíme kombinovať nielen svoje vedomosti z teórie riadenia či matematiky, ale môžeme tiež prešetriť napríklad aj vplyv numerickej reprezentácie čísel či dĺžky slova na daný problém [138, 78].

### 1.6.5 Stochastické a robustné MPC

Formalizáciou podmienok stability na lineárne prediktívne riadenie v roku 2000 [107] metóda prediktívneho riadenia dorástla na úroveň technologickej vyspelosti. Pri tvorbe stratégie riadenia pripustíme, že algoritmus platí hlavne pre takzvaný *nominálny* systém, inými slovami, väčšinou predpokladáme, že náš model je (takmer) dokonalý a pri jeho tvorbe nerátame s neistotami. Naviac predpokladáme, že náš riadený systém a samotný obvod riadenia je *deterministický*, t. j. ak používame dostatočne dokonalý model dynamiky, môžeme aj dokonale predpovedať ako systém reaguje na určité vstupy. V skutočnosti, žiaľ, vstupuje do jeho návrhu veľa neistôt a tak by sme mali sformulovať algoritmy riadenia pre *stochastické* systémy, takzvané stochastické MPC [148]. V súčasnosti môžeme konštatovať, že kým teória a prax lineárneho prediktívneho riadenia sú dobre vyvinuté, ponúkané riešenia na stochastické MPC už nie sú také kompletné ani zjednotené [105].

Neistoty sú všade okolo nás a v rámci MPC prichádzajú do úvahy najmä pri modelovaní systému, ďalej sú prítomné pri meraní veličín výstupov cez snímače a samozrejme aj pri odhade neznámych vnútorných dynamických stavov. Všetky spomenuté deje sú náhodné, t. j. prinášajú do riadenia stochastický faktor.

Jeden z možných prístupov je, že používame *robustné* MPC, kde sa pripravujeme na riadenie systému v najhoršej možnej situácii. Napríklad, určíme dovolenú toleranciu parametrov nášho predikčného modelu a ladíme riadenie tak, aby sme dosiahli uspokojivú kvalitu v celom rozsahu. Tento prístup je žiaľ veľmi pesimistický a konzervatívny. Táto stratégia nevyužíva svoje možnosti naplno, a výkon riadenia je potom značne degradovaný.

Pri stochastickom MPC sú obmedzeniadané ako pravdepodobnostné veličiny, a ich (malé) porušenie je dokonca aj dovolené [148]. Tým, že štatistické variácie rôznych veličín vstupujú do formulácie explicitne, môžeme dosiahnuť lepší riadiaci výkon. Prístup stochastického prediktívneho riadenia je momentálne použiteľný najmä pre určité triedy systémov a vybrané typy neistôt. Vyšetrenie a garancia stability pri stochastickom MPC je už značne komplikované.

### 1.6.6 Ekonomické MPC

Kriteriálna funkcia MPC je najčastejšie tvorená tak, aby zabezpečila dobrú kvalitu riadenia, t. j. presné a rýchle sledovanie žiadanej hodnoty. Keďže toto má vplyv aj na ekonomiku riadenia, v poslednej dobe sa objavili aj štúdie nového smerovania v odbore, takzvaného *ekonomickeho MPC*<sup>29</sup> [127], v ktorom primárnym cieľom je maximalizácia profitu z ekonomickeho hladiska.

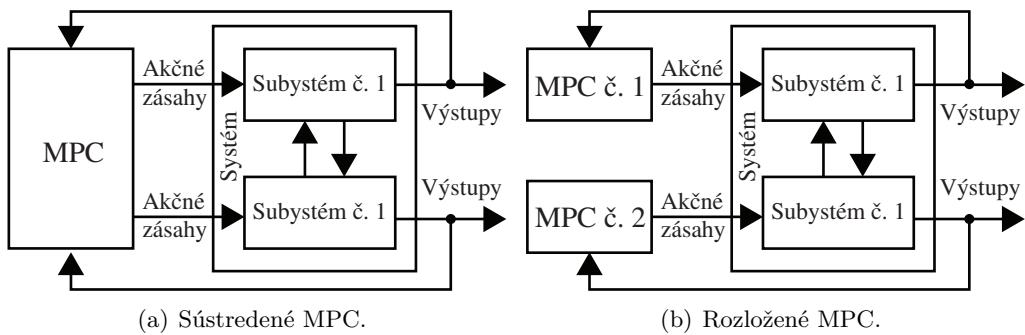
V ekonomickom MPC vstupujú do optimalizačného procesu jasne sformulované ekonomicke očakávania, a preto je dovolené aj menej presné sledovanie žiadanej hodnoty v záujme zvyšovania profitu. Väčšinou pritom ide o kombináciu nadriadenej kriteriálnej funkcie, ktorá vyjadruje ekonomicke faktory riadenia a podriadenej kriteriálnej funkcie, ktorá vyjadruje kvalitu sledovania žiadanej hodnoty.

### 1.6.7 Rozložené MPC

Môžeme si tiež predstaviť niekoľko aplikácií, kde centralizované riadenie, resp. hierarchické riadenie jednoducho nie je prakticky možné. Takéto komplexné systémy majú väčší počet členov, ktoré vyžadujú riadenie a majú svoje lokálne ciele, ale sú rozložené v priestore a majú stanovené aj globálne ciele riadenia (Obr. 1.22). Príkladom môžu byť systémy riadenia dopravy, distribučné siete elektrickej energie alebo aj obrovské výrobné prevádzky [105]. V takomto prípade môžeme hovoriť o rozloženom MPC<sup>30</sup> (DMPC), kde vznikajú špecifické technické výzvy, ako napríklad na ošetrenie komunikačných oneskorení, synchronizácie vzorkovania, zberu dát alebo modelovania [105, 31].

<sup>29</sup>angl. economic MPC

<sup>30</sup>angl. distributed MPC (DMPC)



Obr. 1.22. Sústredené (centralizované) vs. rozložené (decentralizované) MPC.

Základnou podmienkou pri rozloženom MPC je schopnosť rozložiť pôvodný komplexný systém na menšie časti, ktoré môžu mať pri nutnosti splnenia globálnych cieľov riadenia aj svoje lokálne ciele. Viac menších riadiacich slučiek MPC môžeme pritom výpočtovo ľahšie realizovať, pretože prakticky sme paralelizovali pôvodný problém na menšie časti ktoré môžu bežať súčasne. Celkový systém bude robustnejší, pretože ak vypadne obmedzený počet členov zo systému, celkové riadenie stále môže pokračovať a splňať ciele. Ďalej, pri rozloženom MPC môžeme rozširovať systém pridaním ďalších členov a zjednodušíme aj údržbu.



## Kapitola 2

# Modelovanie a odhad stavu

Ako už bolo spomenuté v prvej kapitole, základnou stavebnou jednotkou prediktívneho riadenia (z angl. *model predictive control*) – keďže patrí medzi techniky syntézy riadenia založené na modeli<sup>1</sup> – je model riadeného systému alebo procesu. Je zrejmé, že na rozdiel od iných, klasických metód riadenia, ako je napríklad PID, znalosť dynamiky modelovaného systému dáva regulátoru vlastnosť predikcie, čiže schopnosti predvídať budúci vývoj tzv. stavu systému a tak možnosť naň pôsobiť adekvátnie vypočítanými akčnými zásahmi.

„Model je vyjadrenie podstatných vlastností existujúceho alebo konštruovaného objektu, ktoré opisuje znalosti o tomto objekte v použiteľnej forme.“ [45]  
„Každý dobrý regulátor systému musí byť modelom (musí obsahovať model) tohto systému.“ [5]



Aj v zmysle vyššie uvedenej definície a známeho citátu je zrejmé, že logickým predpokladom dobre navrhnutého, nielen prediktívneho regulátora je formulácia kvalitného matematického modelu. Je tiež známe, že problematikou zostavovania matematických modelov dynamických systémov a procesov sa podrobne zaoberá konkrétna technická špecializácia – *identifikácia sústav*<sup>2</sup> [137]. Motiváciou tejto kapitoly preto nie je poskytnúť obšírny prehľad a analýzu rôznych metód identifikácie, ale zamerať sa skôr na získanie základných poznatkov a zručností potrebných pre odvodenie vhodnej formy stavového modelu, ktorý, ako už vieme, je východiskom pri návrhu moderných metód riadenia, akou je prediktívne riadenie.

Na zopakovanie, pri identifikácii sústav z hľadiska spôsobu tvorby matematického modelu rozlišujeme dva základné a odlišné prístupy [137]. Prvým z nich

<sup>1</sup> angl. model based control techniques

<sup>2</sup> angl. system identification

je tzv. *analytický* prístup, často nazývaný tiež *white-box* „identifikácia“, respektíve modelovanie, pri ktorom matematický model zostavujeme na základe matematicko-fyzikálnej analýzy, t. j. pomocou fyzikálnych a iných základných princípov<sup>3</sup> opisujeme javy prebiehajúce v reálnom objekte a získavame tak vzájomné vzťahy medzi sledovanými veličinami. Tieto, z teórie vychádzajúce znalosti nazývame tiež *apriórne* informácie (z lat. *a priori* – vopred (známe)) a majú rozhodujúci význam pri stanovení štruktúry modelu. Výhodou takto získaného – *štrukturálneho modelu* vo forme tzv. *bielej skrinky*<sup>4</sup> je zrejmá súvislosť medzi parametrami modelu a fyzikálnymi parametrami objektu a jeho dynamickými vlastnosťami. Na druhej strane jeho štruktúra môže byť neúmerne komplikovaná a v tom prípade jeho analýza mimoriadne obtiažna; preto treba hľadať rozumný kompromis medzi zložitosťou a použitelnosťou na určený cieľ. Iným – empirickým prístupom k identifikácii je *experimentálna*, nazývaná tiež *black-box* identifikácia, pri ktorej na formuláciu modelu využívame tzv. *aposteriórne* informácie, získané o vyšetrovanom objekte v priebehu jeho pozorovania, resp. experimentovania s ním. Získaný model síce vystihuje vstupno-výstupné správanie sa objektu, avšak neumožňuje pohľad do jeho vnútornnej štruktúry – zvykneme preto často hovoriť o modeli vo forme tzv. čiernej skrinky<sup>5</sup>. Na druhej strane však, v porovnaní s analytickým prístupom, môžeme do modelu zahrnúť vplyv náhodných chýb (šumu), ktorými sú často zaťažené merané veličiny. Nevýhodou však môže byť fakt, že vyšetrovaný objekt musí byť prístupný experimentu a veličiny merateľné.

Na základe uvedeného sa v mnohých prípadoch, ak je to možné, ako ideálna javí kombinácia oboch spomínaných prístupov. Častý býva preto veľmi efektívny hybridný – tzv. *analyticko-experimentálny* prístup, nazývaný tiež *grey-box* identifikácia, keď sa matematicko-fyzikálnou analýzou získaný model objektu porovnáva s reálnym objektom porovnávaním experimentálne získanými dátami s dátami získanými numerickou simuláciou matematického modelu. Vhodnou kombináciou apriórnych a aposteriórnych informácií tak máme okrem hlbšieho preniknutia do štruktúry objektu možnosť náš model vo forme sivej skrinky<sup>6</sup> na základe experimentu upresňovať a korigovať.

Vzhľadom na apriórne informácie a zámery riadenia musíme posúdiť, či je skúmaný objekt vhodné charakterizovať, a teda matematicky modelovať, ako objekt:

- deterministický alebo stochastický,
- so sústredenými alebo rozloženými parametrami,
- lineárny alebo nelineárny,

<sup>3</sup>angl. first principles

<sup>4</sup>angl. white box

<sup>5</sup>angl. black box

<sup>6</sup>angl. grey box

- 
- časovo nepremenlivý alebo časovo premenlivý,
  - spojity alebo diskrétny,
  - s premenlivými alebo nepremenlivými parametrami atď.

Takýmto spôsobom si môžeme určiť druh matematického modelu a jeho počiaťočné štruktúru, ktoré sa môžu po preskúmaní aposteriórnych informácií potvrdiť, prípadne zmeniť. Uvedené vlastnosti skúmaného objektu rovnako platia aj pre štruktúru stavového modelu, ktorý je pre nás východiskový z pohľadu prediktívneho riadenia. Je teda zrejmé, že stavový model je pomerne obšírny pojem.

Vzhľadom na vyššie uvedené, rôzne matematické reprezentácie vyvstáva otázka: Aký je *dobrý* stavový model pre potreby prediktívneho riadenia? Odpoveď na ňu nie je triviálna. V prvom rade ten, kto model formuluje, by mal vziať do úvahy dostupné apriórne a aposteriórne informácie, ako aj zámery riadenia. V prípade pokročilej metódy riadenia, akou je MPC, platí zásada, že model by mal byť:

- dostatočne podrobny, aby zachytil podstatné dynamické vlastnosti riadenej sústavy,
- dostatočne jednoduchý, aby podliehajúca optimalizačná úloha bola vykonalelná v reálnom čase.

Žiaľ, tieto požiadavky sú vo všeobecnosti často protichodné a pri návrhu štruktúry modelu musíme preto opatrne zvážiť ciele riadenia a naše možnosti.

Ďalej v tejto kapitole sa budeme venovať vysvetleniu nevyhnutných operácií, po pochopení ktorých budeme schopní pre zadaný nelineárny dynamický systém sformulovať jeho *diskrétny lineárny stavový model*. Tento bude v ďalších kapitolách východiskom pre návrh *lineárneho prediktívneho riadenia* v jeho nominálnom tvare. Zvládnutie tohto konceptu zároveň čitateľa predurčí na štúdium pokročilých techník prediktívneho riadenia, ako napríklad robustné, stochasticke alebo adaptívne MPC, pri ktorých bude môcť rovnako využiť a rozšíriť si nadobudnuté základné zručnosti z modelovania. Poznamenajme tiež, že z didaktických dôvodov sa ďalej zameriame na modelovanie na základe matematicko-fyzikálnej analýzy daného systému alebo procesu, vedúce obvykle na minimálnu stavovú realizáciu.

Ako vieme, predpokladom úspešnej implementácie stavového, ako napr. prediktívneho riadenia, je znalosť úplného stavu systému. Kedže v prípade reálnych systémov väčšinou nie sú všetky stavy merané alebo prístupné meraniu, bude záver tejto kapitoly venovaný podstatným základom dôležitej a praktickej súčasti stavového riadenia – problematike *odhadu stavu*<sup>7</sup>. V rozsahu nevyhnutnom z pohľadu riadenia si preto vysvetlíme aspekty návrhu *pozorovateľa stavu*<sup>8</sup>; konkrétnie

---

<sup>7</sup>angl. state estimation

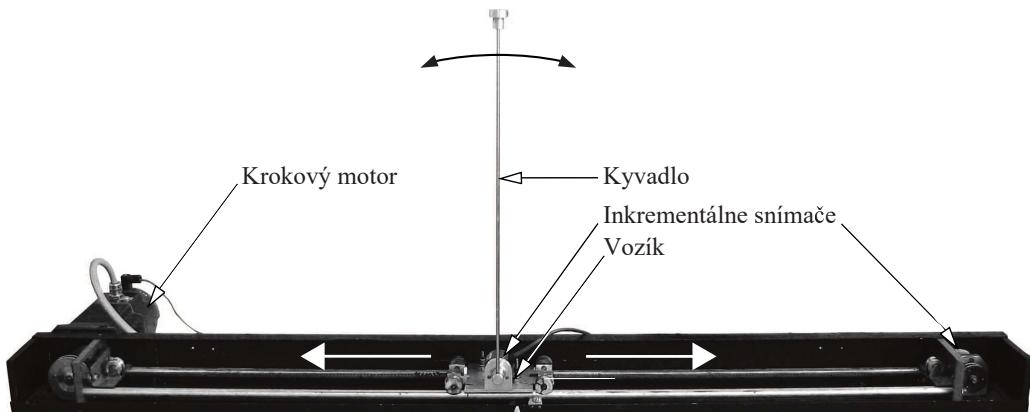
<sup>8</sup>angl. state observer

pomocou známeho *Kalmanovho filtro*. Kedže aj nelineárne systémy opísané lineárizovaným stavovým modelom sa v zmysle svojho účelu často môžu dostať aj do oblasti mimo okolia pracovného bodu, v ktorom zvyčajne navrhujeme lokálny lineárny regulátor, stručne si predstavíme aj variant Kalmanovho filtra pre nelineárne systémy – *rozšírený Kalmanov filter* a ilustrujeme tiež jeho praktické použitie.

## 2.1 Pohybové rovnice a nelineárny stavový model

Ako je všeobecne známe, drvivá väčšina reálnych systémov a procesov má prirodzene nelineárny charakter. Patria medzi ne aj pre automatizáciu typické robotické systémy, vrtuľníky, kvadrokoptéry a rôzne iné mechatronické systémy.

Jedným z oblúbených mechatronických systémov, ktoré nájdeme takmer v každom laboratóriu automatického riadenia, je tzv. invertované kyvadlo znázornené na Obr. 2.1, ktoré pozostáva z kyvadla s jedným koncom rotačne upevneným na pohyblivom vozíku, resp. lineárnom vedení. Invertované kyvadlo sa s obľubou využíva vo výuke teórie automatického riadenia a pri návrhu a testovaní riadiacich algoritmov. Typickou úlohou riadenia v tomto prípade je pomocou horizontálneho pohybu vozíka stabilizovať kyvadlo v jeho vztýčenej – hornej, nestabilnej rovnovážnej polohe<sup>9</sup>. Bez pôsobenia silového účinku by kyvadlo po vychýlení z tejto polohy padlo a ustáliло sa vo volne visiacej – dolnej, stabilnej rovnovážnej polohe.



Obr. 2.1. Laboratórny model invertovaného kyvadla na vozíku.

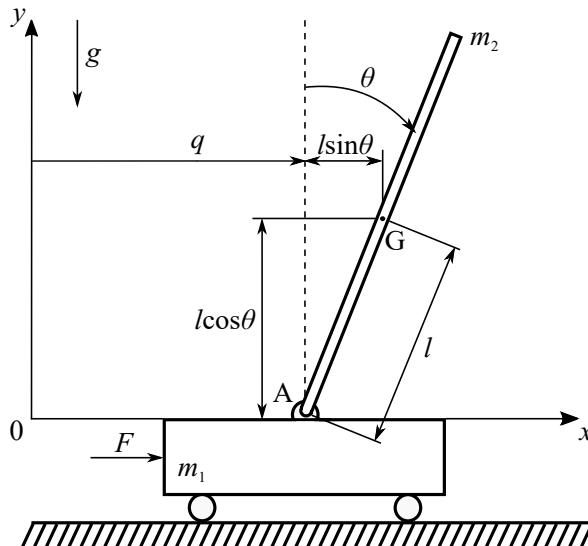
<sup>9</sup>Úlohou riadenia samozrejme nebýva len stabilizácia kyvadla v jeho hornej, nestabilnej rovnovážnej polohe, ale aj jeho vyšvihnutie do jej okolia z dolnej, stabilnej rovnovážnej polohy; prípadne následná zmena referenčnej polohy vozíka pri stálej stabilizácii kyvadla. Podobne v súčasnosti sú pomerne bežné konfigurácie s dvojitým, resp. trojítym kyvadlom; pozri napr. [1, 2].

Okrem toho, že je tento systém nestabilný v otvorennej slučke, má aj ďalšie vlastnosti, ktoré predstavujú výzvu z pohľadu návrhu riadenia. Týmito sú jeho rýchla a silne nelineárna dynamika, a tiež vlastnosť tzv. *čiastočného ovládania* [47], čo znamená, že systém má menej akčných členov ako stupňov voľnosti. Jednoduché invertované kyvadlo (pozri Obr. 2.1) má dva stupne voľnosti (translačný pohyb vozíka a rotačný pohyb kyvadla) a zároveň jeden akčný člen ovládajúci vozík kyvadla. Praktickým príkladom, ktorý sa často dáva do priamej súvislosti so stabilizačiou invertovaného kyvadla je napríklad riadenie polohy rakety pri jej vertikálnom vzlete a mnohé ďalšie systémy, ako napríklad humanoidné roboty, ktoré taktiež majú svoje ťažisko nad pôsobiskom akčnej sily.

Zamerajme sa teraz na nelineárnu dynamiku, ktorá je prirodzenou vlastnosťou reálnych systémov, a teda aj ich matematických modelov. Nelineárny model, založený na nelineárnej diferenciálnej rovnici v sebe zahŕňa nelineárne matematické operácie, ako sú napríklad trigonometrické funkcie, nespojitosťi, saturácia, alebo aj vzájomné násobenie, delenie a umocňovanie premenných.

Ilustrujme si koncept a odvodenie nelineárneho dynamického modelu na príklade systému invertovaného kyvadla. Tento si ďalej transformujeme do tzv. stavopriestorovej reprezentácie, ktorá bude pre nás neskôr dôležitá pre návrh stavového riadenia, akým je aj prediktívne riadenie.

**Príklad 2.1.** Odvodte matematický model – pohybové rovnice invertovaného kyvadla schematicky znázorneného na Obr. 2.2.



Obr. 2.2. Zjednodušená schéma invertovaného kyvadla.

Pre jednoduchosť odvodenia matematického modelu uvažujme štandardné predpoklady, t. j. žiadne trenie ani disipatívne sily a zadefinujme si nasledujúce dynamické veličiny a fyzikálne parametre (pozri Obr. 2.2):

$q, \dot{q}, \ddot{q}$	[m, $\text{ms}^{-1}$ , $\text{ms}^{-2}$ ]	- aktuálna poloha, rýchlosť a zrýchlenie vozíka,
$\theta, \dot{\theta}, \ddot{\theta}$	[rad, $\text{rads}^{-1}$ , $\text{rads}^{-2}$ ]	- aktuálna uhlová výchylka, rýchlosť a zrýchlenie kyvadla,
$F$	[N]	- vstupná sila aplikovaná na vozík,
$m_1$	[kg]	- hmotnosť vozíka,
$m_2$	[kg]	- hmotnosť kyvadla,
$l$	[m]	- vzdialenosť tažiska kyvadla od osi jeho rotácie,
$I$	[ $\text{kgm}^2$ ]	- moment zotrvačnosti kyvadla vzhľadom na jeho tažisko,
$g$	[ $\text{ms}^{-2}$ ]	- tiažové zrýchlenie.

Z Obr. 2.2 je ďalej zrejmé, že pre súradnice tažiska kyvadla, G, platí nasledovné:

$$x_G = q + l \sin \theta,$$

$$y_G = l \cos \theta.$$

Použitím druhého Newtonovho zákona [114], t. j. zákona sily, ktorý zjednodušene hovorí, že časová zmena hybnosti sa rovná výslednej pôsobiacej sile, v smere osi  $x$  dostávame rovnicu:

$$\begin{aligned} F &= m_1 \frac{d^2 q}{dt^2} + m_2 \frac{d^2 x_G}{dt^2} \\ &= m_1 \frac{d^2 q}{dt^2} + m_2 \frac{d^2}{dt^2} (q + l \sin \theta) \\ &= m_1 \ddot{q} + m_2 \left( \ddot{q} + l \left( \frac{d}{dt} \cos \theta \dot{\theta} \right) \right) \\ &= m_1 \ddot{q} + m_2 \ddot{q} + m_2 l \left( \cos \theta \ddot{\theta} - \sin \theta \dot{\theta}^2 \right) \\ &= (m_1 + m_2) \ddot{q} + m_2 l \ddot{\theta} \cos \theta - m_2 l \dot{\theta}^2 \sin \theta, \end{aligned} \quad (2.1)$$

ktorá opisuje dynamiku translačného pohybu vozíka spriahnutého s kyvadlom.

Analyzujme podobne rotačný pohyb kyvadla. V tomto prípade druhý Newtonov zákon hovorí, že časová zmena momentu hybnosti sa rovná výslednému momentu súčinu pôsobiacich na systém, na základe čoho dostávame rovnicu:

$$(I + m_2 l^2) \ddot{\theta} = m_2 g l \sin \theta - m_2 l \ddot{q} \cos \theta, \quad (2.2)$$

ktorá opisuje dynamiku rotačného pohybu kyvadla. Výraz na ľavej strane Rov. (2.2) predstavuje časovú deriváciu momentu hybnosti kyvadla vzhľadom na os jeho

rotácie,  $A$ , pre ktorý platí  $M_A = (I + m_2 l^2) \dot{\theta}$ , a v rámci ktorého  $I_A = I + m_2 l^2$  s využitím Steinerovej vety predstavuje moment zotrvačnosti kyvadla vzhľadom na os rotácie,  $A$ . Pravá strana Rov. (2.2) zohľadňuje moment dvoch pôsobiacich síl, a síce moment od tiažovej sily<sup>10</sup> pôsobiacej na kyvadlo,  $m_2 g l \sin \theta$ , a moment od zotrvačnej sily vozíka,  $-m_2 l \dot{q} \cos \theta$ .

Nelineárne pohybové rovnice (2.1) a (2.2) plne opisujú dynamické správanie sa systému invertovaného kyvadla. Ako môžeme vidieť, uvedený postup získania pohybových rovníc môže byť pre systém s viac stupňami voľnosti pohybu značne namáhavý a neprehľadný.

Ukážme si ešte iný, veľmi praktický a kompaktný spôsob odvodenia pohybových rovníc, ktorý sa s obľubou využíva pri modelovaní aj zložitejších mechanických systémov – pomocou tzv. Euler-Lagrangeovej<sup>11</sup>, rovnice, resp. rovníc.

Ako prvý krok je potrebné si najprv určiť kinetické a pohybové energie, ktoré sa používajú na výpočet tzv. Lagrangeovej funkcie. Pre kinetickú energiu vozíka platí  $K_1 = \frac{m_1 \dot{q}^2}{2}$ , a pre kinetickú energiu kyvadla platí  $K_2 = \frac{m_2 \dot{x}_G^2}{2} + \frac{m_2 \dot{y}_G^2}{2} + \frac{I \dot{\theta}^2}{2}$ . Po dosadení súradníc tiažiska,  $(x_G, y_G)$  a sčítaní získavame celkovú kinetickú energiu systému v nasledujúcom tvaru:

$$K = K_1 + K_2 = \frac{1}{2} (m_1 + m_2) \dot{q}^2 + m_2 l \dot{q} \dot{\theta} \cos \theta + \frac{1}{2} (I + m_2 l^2) \dot{\theta}^2.$$

Celkovú potenciálnu energiu systému, ku ktorej v tomto prípade prispieva len energia kyvadla  $P_1$ , podobne určíme ako

$$P = P_1 = m_2 g l (\cos \theta - 1).$$

Lagrangeova funkcia, tiež známa ako Lagrangeán, je formálne definovaná ako rozdiel celkovej kinetickej a potenciálnej energie systému, a teda nadobudne tvar:

$$\begin{aligned} \mathcal{L} &= K - P \\ &= \frac{1}{2} (m_1 + m_2) \dot{q}^2 + m_2 l \dot{q} \dot{\theta} \cos \theta + \frac{1}{2} (I + m_2 l^2) \dot{\theta}^2 - m_2 g l (\cos \theta - 1). \end{aligned} \quad (2.3)$$

Samotné pohybové rovnice je možné odvodiť z Lagrangeových rovníc v tvaru<sup>12</sup>

$$\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) - \frac{\partial \mathcal{L}}{\partial q} = \tau, \quad (2.4)$$

<sup>10</sup>Tiažová sila je vo všeobecnosti výslednica gravitačnej a odstredivej sily.

<sup>11</sup>Navrhnuté známymi matematikmi L. Eulerom a J.-L. Lagrangeom v 18. storočí. Ich využitie je známe z variačného počtu [50] a klasickej mechaniky [55]. Nazývané tiež, najmä v našej literatúre, Lagrangeove rovnice druhého druhu [23, 116].

<sup>12</sup>Ak v systéme uvažujeme aj prítomnosť viskózneho tlmenia, môžeme ju zohľadniť pridaním výrazu  $+\frac{\partial \mathcal{R}}{\partial \dot{q}}$  na ľavú stranu Rov. (2.4), kde  $\mathcal{R}$  je tzv. Rayleighova disipatívna funkcia [60].

kde  $\mathbf{q} = [q_1 \dots q_n]^T$  predstavuje vektor tzv. zovšeobecnených premenných a  $\boldsymbol{\tau} = [\tau_1 \dots \tau_n]^T$  predstavuje vektor vonkajších síl pôsobiacich na systém, pričom  $n$  označuje počet stupňov voľnosti pohybu systému. V našom prípade sú zovšeobecnenými premennými  $q$  a  $\theta$ , a teda  $\mathbf{q} = [q_1 \ q_2]^T = [q \ \theta]^T$  a  $\boldsymbol{\tau} = [\tau_1 \ \tau_2]^T = [F \ 0]^T$ .

Pri zostavovaní pohybových rovníc si môžeme pomôcť tak, že odvodený Lagrangian z Rov. (2.3) postupne dosadzujeme do výrazov na ľavej strane Rov. (2.4), ktoré potom upravíme pre jednotlivé zovšeobecnené premenné  $\mathbf{q} = [q \ \theta]^T$ , čím dostávame:

$$\begin{aligned} \frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}} \right) &= \frac{d}{dt} \left( (m_1 + m_2) \dot{q} + m_2 l \dot{\theta} \cos \theta \right) \\ &= (m_1 + m_2) \ddot{q} + m_2 l \ddot{\theta} \cos \theta - m_2 l \dot{\theta}^2 \sin \theta, \\ \frac{\partial \mathcal{L}}{\partial q} &= 0, \\ \frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{\theta}} \right) &= \frac{d}{dt} \left( m_2 l \dot{q} \cos \theta + (I + m_2 l^2) \dot{\theta} \right) \\ &= m_2 l \ddot{q} \cos \theta - m_2 l \dot{q} \dot{\theta} \sin \theta + (I + m_2 l^2) \ddot{\theta}, \\ \frac{\partial \mathcal{L}}{\partial \theta} &= m_2 g l \sin \theta - m_2 l \dot{q} \dot{\theta} \sin \theta. \end{aligned}$$

Dosadením takto upravených výrazov späť do systému Euler-Lagrangeovych rovníc (2.4) získavame pohybové rovnice:

$$(m_1 + m_2) \ddot{q} + m_2 l \ddot{\theta} \cos \theta - m_2 l \dot{\theta}^2 \sin \theta = F,$$

$$(I + m_2 l^2) \ddot{\theta} + m_2 l \ddot{q} \cos \theta - m_2 g l \sin \theta = 0,$$

ktoré sú identické s Rov. (2.1)–(2.2).

Pre ich ďalšie použitie zavedieme zo strategických dôvodov štandardné praktické úpravy – uvažujme, že moment zotrvačnosti kvyadla vzhľadom na jeho ľažisko je zanedbateľný, t. j.  $I = 0$ , a uvažujme trenie v rotačnom uložení kvyadla pôsobiacie proti jeho pohybu, dané koeficientom viskózneho tlmenia  $b_\theta$  [ $\text{ms}^{-1}$ ]. Po úpravách nadobúdajú pohybové rovnice invertovaného kvyadla nasledujúci výsledný tvar:

$$(m_1 + m_2) \ddot{q} + m_2 l \ddot{\theta} \cos \theta - m_2 l \dot{\theta}^2 \sin \theta = F, \quad (2.5a)$$

$$l \ddot{\theta} + \ddot{q} \cos \theta - g \sin \theta = -b_\theta \dot{\theta}. \quad (2.5b)$$

□

Pohybové rovnice (2.5a)–(2.5b) sú z pohľadu štruktúry nelineárne diferenciálne rovnice s konštantnými koeficientmi, ktoré vo všeobecnosti predstavujú tzv. *vonkajší opis* systému, t. j. vyjadrujú vzťah medzi vstupnou veličinou  $F$  a výstupnými veličinami  $q$  a  $\theta$ . Hovoríme tiež, že tvoria tzv. vstupno-výstupný model systému.

Ukážme si ďalej ako možno odvodené pohybové rovnice transformovať na tzv. *vnútorný – stavový* opis systému ich prepisom do stavového priestoru, zavedením *stavov* systému. Stavová reprezentácia podáva významne väčšie množstvo informácií o správaní sa daného systému než opis vonkajší (napr. prenosová funkcia, diferenciálna rovnica a pod.), a aj to je jeden z mnohých dôvodov prečo sa používa pri návrhu moderných, pokročilých metód riadenia, akou je napríklad MPC.

**Príklad 2.2.** Transformujte pohybové rovnice (2.5) invertovaného kyvadla do stavového priestoru a odvodte nelineárny stavový model.

Začneme definovaním stavových premenných, ktoré zaužívaným spôsobom označujeme písmenom  $x$ . Keďže systém invertovaného kyvadla je opísaný dvoma diferenciálnymi rovnicami druhého rádu (2.5a) a (2.5b), budeme potrebovať štyri stavy, ktoré si zvolíme takto:

$$x_1 = q, \quad x_2 = \dot{q}, \quad x_3 = \theta \quad \text{a} \quad x_4 = \dot{\theta}.$$

Ak si jednotlivé stavové veličiny zderivujeme v čase, dostávame na základe vzťahov medzi jednotlivými stavmi nasledujúci systém rovnic:

$$\begin{aligned} \dot{x}_1 &= \dot{q} = x_2, \\ \dot{x}_2 &= \ddot{q} = \frac{m_2 l}{m_1 + m_2} (x_4^2 \sin x_3 - \dot{x}_4 \cos x_3) + \frac{F}{m_1 + m_2}, \\ \dot{x}_3 &= \dot{\theta} = x_4, \\ \dot{x}_4 &= \ddot{\theta} = \frac{1}{l} (-\dot{x}_2 \cos x_3 - b_\theta x_4 + g \sin x_3), \end{aligned} \tag{2.6}$$

kde sme na pravú stranu druhej a štvrtnej rovnice za  $\ddot{q}$  a  $\ddot{\theta}$  dosadili samotné pohybové rovnice (2.5a)–(2.5b), upravené a vyjadrené v zmysle stavových premenných. Všimnime si, že tieto rovnicu obsahujú tzv. algebraickú slučku, pretože  $\dot{x}_2$  závisí od  $\dot{x}_4$ , a naopak,  $\dot{x}_4$  závisí od  $\dot{x}_2$ . Aby sme ich mohli softvérovo implementovať, túto slučku odstránime vzájomným dosadením príslušných rovnic jednej do druhej, a po ďalších úpravách dostávame nelineárne stavové rovnice v tvare:

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= \frac{-m_2 g \sin x_3 \cos x_3 + m_2 l x_4^2 \sin x_3 + b_\theta m_2 x_4 \cos x_3 + F}{m_1 + (1 - \cos^2 x_3) m_2}, \\ \dot{x}_3 &= x_4, \\ \dot{x}_4 &= \frac{(m_1 + m_2) (g \sin x_3 - b_\theta x_4) - (m_2 l x_4^2 \sin x_3 + F) \cos x_3}{l (m_1 + (1 - \cos^2 x_3) m_2)}. \end{aligned} \tag{2.7}$$

Všimnime si, že prepisom do stavového priestoru sme z pôvodných 2 diferenciálnych pohybových rovnic 2. rádu získali 4 diferenciálne stavové rovnice 1. rádu.

Všeobecnejší tvar spojitej nelineárnej stavovopriestorovej reprezentácie systému vyzerá takto<sup>13</sup>:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \quad (2.8a)$$

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t), \mathbf{u}(t), t), \quad (2.8b)$$

kde  $\mathbf{x} \in \mathbb{R}^{n_x}$  predstavuje *stavový vektor* (vektor stavov),  $\mathbf{u} \in \mathbb{R}^{n_u}$  je vektor vstupov a  $\mathbf{y} \in \mathbb{R}^{n_y}$  je vektor (meraných) výstupov systému. Rov. (2.8a) sa nazýva (nelineárna) *stavová rovnica* alebo tzv. *rovnica dynamiky*, kde vektorová funkcia  $\mathbf{f}(\cdot)$  je nelineárnu kombináciu stavov a vstupov systému a opisuje teda vývoj jeho stavu v čase. Rov. (2.8b) sa nazýva *rovnica výstupu* alebo *rovnica merania*, a pomocou vo všeobecnosti nelineárnej vektorovej funkcie  $\mathbf{h}(\cdot)$  vyjadruje závislosť výstupu systému na jeho aktuálnom stave a vstupe v čase. Ak teda  $\dot{\mathbf{x}}(t)$  a  $\mathbf{y}(t)$  nie sú lineárnymi kombináciami  $\mathbf{x}(t)$  a  $\mathbf{u}(t)$ , potom model označujeme ako nelineárny. Rov. (2.8a) a (2.8b) sú spoločne známe ako *spojitý nelineárny stavový model*.

Všimnime si tiež, že funkcie  $\mathbf{f}$  a  $\mathbf{h}$  v rovniciach stavového modelu (2.8a)–(2.8b) obsahujú explicitnú závislosť na čase (tretí argument  $t$ ). Nazývame ho preto *časovo premenlivý*, resp. *časovo variantný*, čo sa najčastejšie prejavuje zmenou jeho parametrov v čase<sup>14</sup>. Tento fakt zvyčajne vedie k potrebe ich priebežnej identifikácie a ďalej k tzv. adaptívnomu riadeniu. Ak sa parametre modelu v čase nemenia, hovoríme o tzv. *časovo nepremenlivom*, resp. invariantom stavovom modeli.

Ak v prípade invertovaného kyvadla uvažujeme meranie výchylky vozíka,  $q$ , a uhlovej výchylky kyvadla,  $\theta$ , pričom vstup si označíme ako  $u = F$ , bude mať nás spojity, nelineárny časovo-invariantný stavový model tvar<sup>15</sup>:

$$\underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix}}_{\dot{\mathbf{x}}} = \underbrace{\begin{bmatrix} x_2 \\ -m_2 g \sin x_3 \cos x_3 + m_2 l x_4^2 \sin x_3 + b_\theta m_2 x_4 \cos x_3 + u \\ m_1 + (1 - \cos^2 x_3) m_2 \\ (m_1 + m_2) (g \sin x_3 - b_\theta x_4) - (m_2 l x_4^2 \sin x_3 + u) \cos x_3 \end{bmatrix}}_{\mathbf{f}(\mathbf{x}, u)}, \quad (2.9a)$$

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} x_1 \\ x_3 \end{bmatrix}}_{\mathbf{h}(\mathbf{x})}. \quad (2.9b)$$

□

<sup>13</sup>Zámerne uvádzame aj závislosť dynamických veličín na spojitej čase,  $\cdot(t)$ , ktorú ďalej pre prehľadnosť nebudeme uvádzat.

<sup>14</sup>V prípade kyvadla napríklad, ak by sa jeho dĺžka alebo hmotnosť menila v čase, a podobne. V praktickom prípade vzlietajúcej rakety sa mení jej hmotnosť v dôsledku spalovania paliva.

<sup>15</sup>Všimnime si, že rovnica výstupu je v tomto prípade lineárna a nezávisí od vstupu systému. Toto by už však neplatilo ak by sme uvažovali napr. meranie zrýchlení  $\ddot{q} = \dot{x}_2 = \dots, \dot{\theta} = \dot{x}_4 = \dots$

Pripomeňme ešte, že (2.8a) predstavuje sústavu diferenciálnych rovníc prvého rádu, zatiaľ čo (2.8b) predstavuje sústavu algebraických rovníc. Ako tiež vidíme na našom príklade, stavové rovnice (2.9a), konkrétnie druhá a štvrtá, opisujú silne nelineárnu dynamiku systému. Nelinearity sú tu dôsledkom vzájomného násobenia a delenia stavov a vstupu systému, ich trigonometrických funkcií a mocnín.

Medzi výhody *vnútornej – stavovopriestorovej reprezentácie* systému oproti iným, tradičným formám vonkajšieho opisu systému, patrí, že:

- umožňuje prístup k vnútornému stavu systému;
- umožňuje zohľadniť počiatočné podmienky;
- poskytuje riešenie priamo v časovej oblasti;
- transformuje diferenciálnu rovnicu  $n_x$ -tého rádu na  $n_x$  diferenciálnych rovníc prvého rádu<sup>a</sup>, ktorých numerické riešenie je jednoduchšie;
- vzhľadom na svoju štruktúru je jednoduchá na implementáciu;
- je použiteľná aj na zložité systémy (nielen mechanické, ale aj elektrické, biologické, ekonomické, spoločenské a rôzne iné systémy);
- je použiteľná aj na systémy s viacerými vstupmi a výstupmi;
- je použiteľná aj na nelineárne systémy;
- je použiteľná aj na časovo premenlivé systémy;
- je východiskom pre návrh a analýzu moderných metód riadenia.

*Stav* dynamického systému v čase  $t_0$ ,  $\mathbf{x}(t_0)$ , predstavuje minimálne množstvo informácií v čase  $t_0$ , ktoré spoločne so vstupom  $\mathbf{u}(t)$ ,  $t \geq t_0$  jedinečne určuje správanie sa systému v čase  $t \geq t_0$ .

*Stavový model* je matematickým opisom dynamického systému v zmysle minimálneho počtu stavových premenných  $x_i(t)$ ,  $i = 1, \dots, n_x$ , ktoré sú spolu so znalosťou týchto premenných v počiatočnom čase  $t_0$  a vstupov systému v čase  $t \geq t_0$  postačujúce na predikciu budúceho stavu a výstupov systému v ľubovoľnom čase  $t > t_0$ . Práve vlastnosť predikcie stavového modelu je podstatná pre návrh prediktívneho riadenia.

*Stavový priestor* je  $n_x$ -rozmerný priestor, ktorého súradnicové osi sú definované v zmysle jednotlivých stavov  $x_1, \dots, x_{n_x}$ ;  $n_x$  označuje *rád modelu*.

<sup>a</sup>Vo všeobecnosti transformuje systém  $m$  diferenciálnych rovníc  $n$ -tého rádu na  $n_x = m \cdot n$  diferenciálnych rovníc prvého rádu.



Kedže sme si už predstavili nelineárny stavový model, ukážme si na príklade invertovaného kyvadla ako môžeme numericky odsimulovať jeho dynamickú odozvu.

**Príklad 2.3.** Vykonajte numerickú simuláciu nelineárnych stavových rovníc (2.9a) získaných na základe pohybových rovníc (2.5a)–(2.5b) invertovaného kyvadla.

Dynamické správanie sa nášho systému môžeme vyšetrovať z dvoch pohľadov – na základe tzv. *voľnej odozvy*<sup>16</sup> alebo vynútenej odozvy<sup>17</sup> systému. Uvažujme teraz prvý prípad, keď k vybudeniu pohybu systému dochádza iba v dôsledku počiatočnej podmienky, a teda bez pôsobenia vonkajšej budiacej sily, t. j.  $u = F = 0$ . Uvažujme preto, že kyvadlo sa na začiatku nachádza v nenulovej počiatočnej polohe, vychýlené o  $30^\circ$  od jeho nestabilnej rovnovážnej polohy. Tejto konfigurácii prislúcha stav  $\mathbf{x}_0 = [q_0 \dot{q}_0 \theta_0 \dot{\theta}_0]^T = [0 \ 0 \ \frac{\pi}{6} \ 0]^T$ .

Na vykonanie numerickej simulácie voľnej odozvy invertovaného kyvadla možno efektívne využiť skriptové prostredie programu MATLAB. Začnime zadefinovaním parametrov a počiatočného stavu systému, ako aj trvania a kroku simulácie:

Zdrojový kód 2.1. simnelinkyvadlo.m

```
m1=0.4; % Hmotnosť vozika [kg]
m2=0.08; % Hmotnosť kyvadla [kg]
l=0.2; % Vzdialenosť taziska od osi rotacie [m]
g=9.81; % Tiaze zrychlenie [m/s2]
b=0.1; % Koeficient trenia [m/s]
cas=0:0.01:10; % Trvanie a krok simulacie [s]
x0=[0;0;pi/6;0];% Počiatocne podmienky (hodnoty stavov)
```

Pred samotnou simuláciu si vo vlastnom skripte *nelinkyvadlo.m* vytvoríme rovnomennú funkciu *nelinkyvadlo()*, pomocou ktorej kompaktne zapíšeme nelineárne stavové rovnice (2.9a). Jej argumentom sú deklarované fyzikálne parametre systému, pričom v našom prípade voľnej odozvy vstupnú silu neuvažujeme.

Zdrojový kód 2.2. nelinkyvadlo.m

```
function dxdt = nelinkyvadlo(t,x,m1,m2,l,g,b)
dxdt = [x(2);
        (-m2*g*sin(x(3))*cos(x(3))+m2*l*(x(4))^2)*sin(x
        (3))+b*m2*x(4)*cos(x(3)))/(m1+(1-(cos(x(3)))
        ^2)*m2);
x(4);
```

<sup>16</sup>angl. free/zero-input response

<sup>17</sup>angl. forced/zero-state response

$$((m1+m2)*(g*\sin(x(3))-b*x(4))-(m2*l*((x(4))^2)*\sin(x(3)))*\cos(x(3)))/(l*(m1+(1-\cos(x(3)))^2)*m2))];$$

Simuláciu takto implementovaného stavového modelu vykonáme po návrate do skriptu *nelinkyvadlo.m* nasledujúcim príkazom:

```
[t, x] =ode45(@(t, x) nelinkyvadlo(t, x, m1, m2, l, g, b), cas, x0);
```

kde `ode45`<sup>18</sup> predstavuje riešič, ktorý využitím Runge-Kutta metódy 4/5. rádu numericky integruje systém obyčajných diferenciálnych rovníc<sup>19</sup> (ODE), ktoré sú v našom prípade dané funkciou `nelinkyvadlo()`. Ďalšími vstupnými argumentmi sú simulačný čas a počiatočný stav, a výstupom vektor riešení `x` v čase `t`.

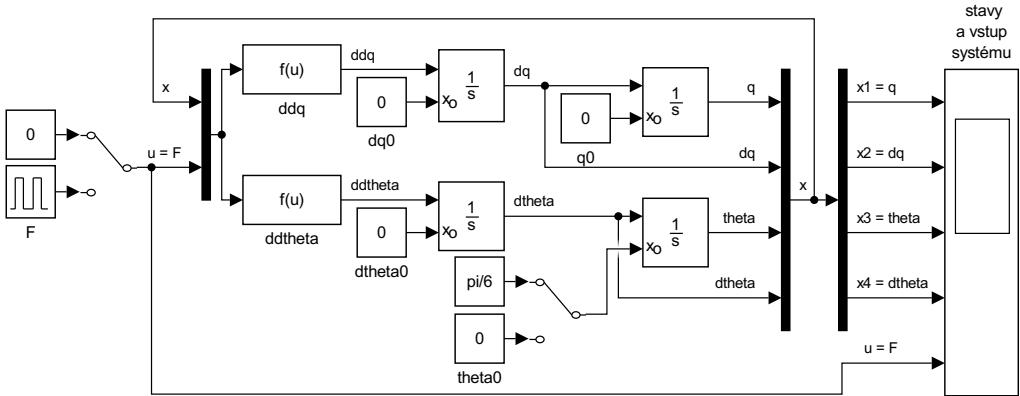
Získané výsledky si môžeme graficky znázorniť tak, že pomocou nasledujúcich príkazov vykreslíme v samostatných oknách časové priebehy jednotlivých stavov systému a pre ilustráciu aj nulový vstup, pridáme mriežku a označíme osi grafov.

```
figure('color', 'w') % Otvorenie grafickeho okna
subplot(5,1,1) % Prvy z podgrafov
plot(t,x(:,1)); grid on; % Posunutie vozika + mriezka
ylabel('x_1 (m)'); % Oznacenie osi y
subplot(5,1,2)
plot(t,x(:,2)); grid on; % Rychlosť vozika
ylabel('x_2 (ms^{-1})');
subplot(5,1,3)
plot(t,x(:,3)); grid on; % Uhlova vychylka kyvadla
ylabel('x_3 (rad)');
subplot(5,1,4)
plot(t,x(:,4)); grid on; % Uhlova rychlosť kyvadla
ylabel('x_4 (rads^{-1})');
subplot(5,1,5)
plot(t,zeros(length(t),1)); grid on; % Sila apl. na vozik
xlabel('t (s)'); ylabel('u (N)'); % Oznacenie osi x a y
```

Grafické výsledky si zámerne znázorníme neskôr. Ešte predtým skúsme uvažovať situáciu, keď pohyb systému nastane v dôsledku pôsobiacej sily, t. j. vynútenú odozvu systému. Kedže simulaovať ju pomocou skriptu by bolo zbytočne komplikované, ukážme si ako ju možno efektívne zrealizovať v blokovom grafickom prostredí programu MATLAB–Simulink.

<sup>18</sup>ode45 predstavuje pomerne všeobecný riešič obyčajných diferenciálnych rovníc (ODE), ktorý sa odporúča vyskúšať ako prvý pri väčšine problémov [104]. Ak je daný problém ‘tuhý’ (angl. stiff) alebo vyžaduje vyššiu presnosť, môžu byť na jeho riešenie vhodnejšie iné riešiče; viď [102, 133].

<sup>19</sup>angl. ordinary differential equation (ODE)



Obr. 2.3. Bloková schéma pre numerickú simuláciu nelineárneho dynamického modelu invertovaného kvyadla v prostredí Simulink.

Začnime tým, že zadaním príkazu `simulink` v pracovnom okne (*Command Window*) MATLABu a zvolením *Blank Model* si otvoríme prázdné okno pre návrh blokovej schémy. Ešte pred jej vytváraním si upresníme nastavenia simulácie, ktoré otvoríme v ponuke menu *Simulation→Model Configuration Parameters*. V záložke riešiča (*Solver*) si pod hlavičkou *Simulation time* špecifikujeme simulačný čas, pod hlavičkou *Solver options* zvolíme typ riešiča (*Type*) s pevným krokom (*Fixed step*) a riešič ako *ode4* (*Runge-Kutta*). V *Additional options* si špecifikujeme veľkosť kroku integrácie (*Fixed-step size*) identicky ako v skripte, t. j. *0.01* (s). Alternatívne si môžeme zvoliť aj riešič s premenlivým krokom (*Variable-step*) ako *ode45* (*Dormand-Prince*) a ďalej špecifikovať minimálnu a maximálnu dĺžku kroku. Ďalej môžeme pristúpiť k tvorbe blokovej schémy znázornenej na Obr. 2.3. Jednotlivé bloky si priebežne vyberáme z knižnice (*Library Browser*, *Ctrl+Shift+L*). Začneme so stavovým modelom (2.9a), ktorého druhú a štvrtú rovnicu si môžeme jednoducho implementovať pomocou blokov *Fcn* z triedy *User-Defined Functions*. Do týchto blokov si priamo prepíšeme pravé strany týchto rovníc, pričom stavu a vstupu systému si označíme ako  $u(1)$  až  $u(5)$ , t. j. ako zložky abstraktného vektora  $u$ , ktorý predstavuje vstup do bloku *Fcn*. Keďže fyzikálne výstupy z týchto blokov sú zrýchlenie vozíka  $\ddot{q}$  a uhlové zrýchlenie kvyadla, musíme každý z nich v rámci simulácie dvakrát zintegrovať ( $\ddot{q} \rightarrow \dot{q} \rightarrow q$ ,  $\ddot{\theta} \rightarrow \dot{\theta} \rightarrow \theta$ ), na čo využijeme bloky *Integrator*<sup>20</sup>, v ktorých môžeme zadať počiatočnú podmienku bud priamo alebo externe pomocou blokov *Constant*. Výstupy z integrátorov ďalej pomocou *Mux*

<sup>20</sup>Dvojitú integráciu možno vykonať aj naraz pomocou bloku *Integrator, Second-Order*, na ktorého vstupe možno zadať napríklad v prípade vozíka  $\ddot{q}, \dot{q}_0, q_0$  a na výstupe získať  $\dot{q}, q$ .

bloku zoradíme do stavového vektora  $\mathbf{x} = [x_1 \ x_2 \ x_3 \ x_4]^T = [q \ \dot{q} \ \theta \ \dot{\theta}]^T$ , ktorý spolu s našim vstupom  $u = F$  priviedieme na vstup *Fcn* blokov. Podobne si stavové spolu so vstupnou veličinou priviedieme na vstupy bloku *Scope* z triedy *Sinks*, v ktorom si ich časové priebehy môžeme znázorniť v rovnakej podobe ako v skripte. Tento blok tiež ponúka možnosť uloženia získaných priebehov do pracovného priestoru (*Workspace*) MATLABu (*View*→*Configuration Properties*→*Logging*→ *Log data to workspace*,  *Limit data points to last...*).

Všimnime si, že bloková schéma na Obr. 2.3 pomocou bloku *Manual Switch* umožňuje vykonať simuláciu voľnej odozvy pri uvažovaní  $\theta_0 = \pi/6$  a  $F = 0$ , ako aj vynútenej odozvy kedž  $\theta_0 = 0$  a silu pôsobiacu na vozík si *Pulse Generator* vygenerujeme ako impulz o veľkosti  $F = -1$  aplikovaný na začiatku simulácie.

Simuláciu modelu spustíme tlačidlom *Run* (resp. *Ctrl+T*). Priebehy jednotlivých sledovaných veličín môžeme do grafov získaných pomocou skriptu dokresliť po ich podržaní pomocou príkazov *hold on*, a zobrazíť ich tak ako na Obr. 2.4.

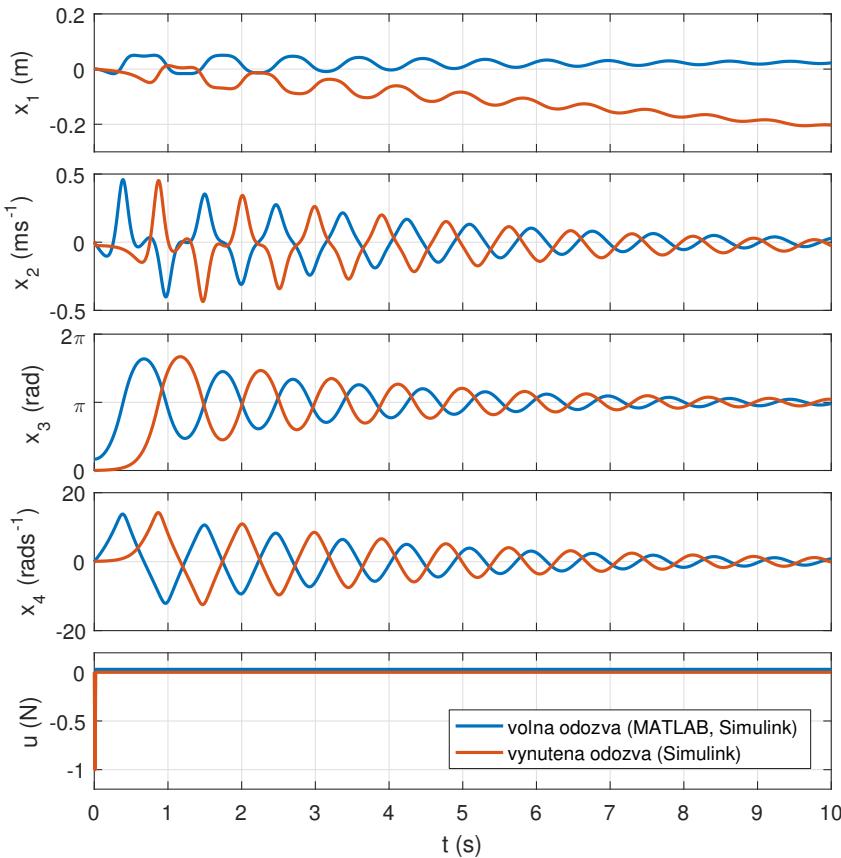
Na Obr. 2.4 sú modrou farbou znázornené priebehy stavových a vstupnej veličiny získané simuláciou voľnej odozvy – s identickými priebehmi získanými pomocou skriptu a blokovej schémy – a červenou farbou vynútená odozva simulovaná pomocou blokovej schémy. Ako vidíme, v oboch prípadoch kyvadlo po vychýlení z hornej, nestabilnej rovnovážnej polohy padne a kedž uvažujeme trenie v jeho rotačnom uložení, tak tlmením kmitaním osciluje okolo dolnej, stabilnej rovnovážnej polohy až sa v nej ustáli. Podobne sa vozík vplyvom pohybu kyvadla, resp. aplikovaného silovú impulzu posunie do novej, vychýlenej polohy.

Poznamenajme ešte, že rovnicu výstupu (2.8a) nie je potrebné implementovať, kedž v simulácii sú všetky stavy systému známe.  $\square$

## 2.2 Linearizácia a lineárny stavový model

Ako je všeobecne známe, takmer všetky reálne systémy a procesy majú nelineárnu dynamiku, kedž diferenciálne rovnice opisujúce správanie sa systému sú nelineárne. Väčšina teórií, ktoré boli vyvinuté, sa však práve naopak sústredí na lineárne systémy. Poznáme tiež mnoho nelineárnych systémov, ktoré pracujú len v špecifických pracovných režimoch, resp. je pre nás zaujímavé ich správanie sa iba v určitých režimoch, v ktorých ich chceme riadiť. Prirodzene teda prichádza do úvahy nasledujúca otázka: „Za akých okolností sa môžeme na nelineárny systém pozerat ako na lineárny systém?“

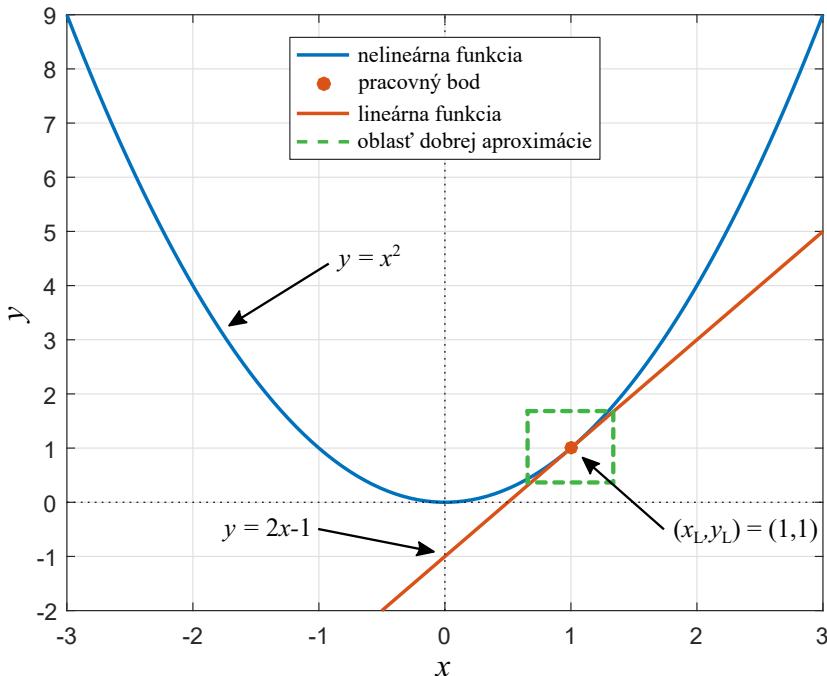
Našťastie je tiež známe, že aj keď drívavá väčšina fyzikálnych systémov zahŕňa nelinearity, často môže byť ich správanie sa v rámci určitého rozsahu pracovných podmienok dostatočne dobre approximované správaním sa lineárneho modelu. Príslušná technika získavania lineárnej approximácie sa nazýva *linearizácia*. V tejto



Obr. 2.4. Simulácia voľnej a vynútenej odozvy nelineárneho modelu invertovaného kyvadla v prostredí MATLAB a Simulink.

podkapitole si ukážeme ako môžeme daný nelineárny systém linearizovať v určitom *pracovnom bode*, nazývanom aj rovnovážny bod.

Linearizácia nelineárnych modelov hrá dôležitú úlohu v praktickom návrhu riadenia. Princíp linearizácie nelineárnych rovníc predpokladá, že príslušné veličiny sa menia len málo, a teda ich odchýlky od pracovného bodu, resp. ustáleného stavu sú malé. Výsledkom linearizácie je tzv. *linearizovaný model*, ktorého správanie sa v určitom malom okolí pracovného bodu rozumne aproximuje správanie sa pôvodného nelineárneho modelu. Linearizované, alebo lineárne modely sú spravidla jednoduchšie na pochopenie a umožňujú použitie lineárnych metód riadenia. Spomeňme si, že klasické metódy návrhu a analýzy riadiacich systémov vyžadujú lineárne, časovo invariantné modely.



Obr. 2.5. Ilustrácie linearizácie jednoduchej nelineárnej funkcie.

My sa ďalej zameriame na linearizáciu (spojitého) nelineárneho stavového modelu, pomocou ktorej získame (spojity) *linearizovaný stavový model*, ktorý bude prvým predpokladom pre návrh lineárneho prediktívneho riadenia.

Ilustrujme si koncept linearizácie na príklade jednoduchej nelineárnej – kvadratickej funkcie  $y = x^2$ , ktorá je znázornená na Obr. 2.5 modrou farbou. Je zrejmé, že táto funkcia je nelineárna v jej nezávislej premennej  $x$ . Uvažujme preto, že ju chceme lineárne aproximovať v pracovnom, tzv. linearizačnom bode  $(\bar{x}, \bar{u}) = (1, 1)$ . Linearizáciou funkcie  $y = x^2$  v bode  $(1, 1)$  dostávame<sup>21</sup> lineárnu funkciu  $y = 2x - 1$  (vyznačenú na obrázku červenou farbou), ktorá je z geometrického pohľadu dotyčnicou k pôvodnej nelineárnej funkcií v linearizačnom bode  $(1, 1)$ , a teda predstavuje sklon funkcie v tomto bode.

Ako je zrejmé z Obr. 2.5, v určitom blízkom okolí pracovného bodu (vyznačenom zelenou farbou) je lineárna funkcia  $y = 2x - 1$  dobrou aproximáciou nelineárnej funkcie  $y = x^2$ , zatiaľ čo mimo tohto okolia je táto aproximácia už nepresná,

<sup>21</sup>Pripomeňme si, že matematicky toto vykonáme parciálnou deriváciou funkcie  $y = x^2$  podľa nezávislej premennej  $x$ , t. j.  $y' = \frac{\partial x^2}{\partial x} = 2x + c$ , kde po dosadení linearizačného bodu  $(\bar{x}, \bar{u}) = (1, 1)$  za  $(x, y)$  dostávame  $c = -1$ , a teda lineárnu approximáciu v  $(\bar{x}, \bar{u})$  predstavuje funkcia  $y = 2x - 1$ .

resp. nepostačujúca. Táto oblasť platnosti dobrej aproximácie samozrejme závisí od daného nelineárneho modelu. Je zrejmé, že podobným spôsobom by sme postupovali aj v prípade viacozmerných, vektorových funkcií, keď parciálne deriváciu vykonávame podľa jednotlivých nezávislých premenných.

Zovšeobecníme si teraz koncept linearizácie na dynamické systémy, ktoré budeme uvažovať vo forme spojitého nelineárneho časovo nepremenlivého (invariantného) stavového modelu v nasledujúcom tvare<sup>22</sup>:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \quad \begin{cases} \dot{x}_1 = f_1(x_1, \dots, x_{n_x}, u_1, \dots, u_{n_u}), \\ \vdots \\ \dot{x}_{n_x} = f_{n_x}(x_1, \dots, x_{n_x}, u_{11}, \dots, u_{n_u}), \end{cases} \quad (2.10a)$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u}) \quad \begin{cases} y_1 = h_1(x_1, \dots, x_{n_x}, u_1, \dots, u_{n_u}), \\ \vdots \\ y_{n_y} = h_{n_y}(x_1, \dots, x_{n_x}, u_1, \dots, u_{n_u}). \end{cases} \quad (2.10b)$$

Kedže nás zaujíma linearizácia dynamiky systému v okolí určitého pracovného bodu, zavedme si najprv nové súradnice

$$\Delta \mathbf{x} = \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_{n_x} \end{bmatrix} = \begin{bmatrix} x_1 - \bar{x}_1 \\ \vdots \\ x_{n_x} - \bar{x}_{n_x} \end{bmatrix} = \mathbf{x} - \bar{\mathbf{x}},$$

a podobne pre  $\Delta \mathbf{u} = \mathbf{u} - \bar{\mathbf{u}}$  a  $\Delta \mathbf{y} = \mathbf{y} - \mathbf{h}(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ , ktoré predstavujú *odchylinky* stavových, vstupných a výstupných veličín od ich príslušných rovnovážnych hodnôt, resp. nominálnych trajektórií  $\bar{\mathbf{x}}$ ,  $\bar{\mathbf{u}}$  a  $\bar{\mathbf{y}} = \mathbf{h}(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ , ktoré budú v nasledujúcom predstavovať náš pracovný bod.

Nelineárne diferenciálne stavové rovnice vo vektorovom tvare (2.10a) môžeme zapísat pomocou rozvoja do Taylorovho radu v bode  $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$  v nasledujúcom tvare:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \bigg|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \bigg|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} (\mathbf{u} - \bar{\mathbf{u}}) + \text{výrazy vyšších rádov.} \quad (2.11)$$

Lineárnu aproximáciu nelineárnej funkcie  $\mathbf{f}(\mathbf{x}, \mathbf{u})$  získame tak, že v jej Taylorovom rozvoji na pravej strane Rov. (2.11) zanedbáme výrazy druhého a vyšších rádov, t. j. ponecháme len lineárne členy, čím dostávame:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) \approx \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \bigg|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \bigg|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} (\mathbf{u} - \bar{\mathbf{u}}). \quad (2.12)$$

<sup>22</sup>Značenie závislosti stavových, vstupných a výstupných veličín na spojitej čase, t. j.  $\cdot(t)$ , pre jednoduchosť výkladu ďalej zanedbáme.

Nelineárnu funkciu  $\mathbf{f}(\mathbf{x}, \mathbf{u})$  na ľavej strane (2.12), pre ktorú z rovnice (2.10a) platí  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ , môžeme v zmysle odchýlok  $\Delta$  nahradiť výrazom  $\dot{\mathbf{x}} + \Delta \dot{\mathbf{x}}$ . Zároveň, ak je pre nás lineárna aproximácia (2.12) dostatočne presná ( $\approx$  formálne nahradíme  $=$ ), dostávame nasledujúcu rovnicu:

$$\dot{\mathbf{x}} + \Delta \dot{\mathbf{x}} = \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} (\mathbf{x} - \bar{\mathbf{x}}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} (\mathbf{u} - \bar{\mathbf{u}}). \quad (2.13)$$

Vzhľadom na to, že v pracovnom bode platí  $\dot{\mathbf{x}} = \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ , môžeme príslušné výrazy  $\dot{\mathbf{x}}$  a  $\mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}})$  z ľavej a pravej strany Rov. (2.13) odstrániť. Potom po substitúcii  $\Delta \mathbf{x} = (\mathbf{x} - \bar{\mathbf{x}})$  a  $\Delta \mathbf{u} = (\mathbf{u} - \bar{\mathbf{u}})$  dostávame rovnicu v tvare:

$$\Delta \dot{\mathbf{x}} = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \Delta \mathbf{x} + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \Delta \mathbf{u}, \quad (2.14)$$

ktorá predstavuje tzv. Jacobiho linearizáciu nelineárnej stavovej rovnice (2.10a) v pracovnom bode  $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ .

Tento postup možno analogicky použiť na linearizáciu vo všeobecnosti nelineárnej algebraickej rovnice výstupu (2.10b), čím získame:

$$\Delta \mathbf{y} = \frac{\partial \mathbf{h}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \Delta \mathbf{x} + \frac{\partial \mathbf{h}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \Delta \mathbf{u}. \quad (2.15)$$

Kedže  $\mathbf{f}(\mathbf{x}, \mathbf{u})$  a  $\mathbf{h}(\mathbf{x}, \mathbf{u})$  v linearizovaných Rov. (2.14) a (2.15) sú vo všeobecnosti vektorové funkcie, ich parciálne derivácie podľa vektorových premenných  $\mathbf{x}$  a  $\mathbf{u}$ , vyhodnotené v pracovnom bode  $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ , sú matice<sup>23</sup>, tradične značené ako<sup>24</sup>:

$$\begin{aligned} \mathbf{A}_c &= \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \dots & \frac{\partial f_1}{\partial x_{n_x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_x}}{\partial x_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \dots & \frac{\partial f_{n_x}}{\partial x_{n_x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n_x} \\ a_{21} & a_{22} & \dots & a_{2n_x} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_x 1} & a_{n_x 2} & \dots & a_{n_x n_x} \end{bmatrix}, \end{aligned} \quad (2.16a)$$

<sup>23</sup>V matematike maticu prvých derivácií vektorovej funkcie nazývame Jacobiho matica alebo tzv. Jakobián. Preto uvedenej linearizácií hovoríme aj Jacobiho linearizácia.

<sup>24</sup>Dolný index  $\cdot_c$  pri maticiach  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$  a  $\mathbf{D}$  používame na explicitné zdôraznenie spojitého (angl. continuous) modelu.

$$\mathbf{B}_c = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \cdots & \frac{\partial f_1}{\partial u_{n_u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_x}}{\partial u_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \cdots & \frac{\partial f_{n_x}}{\partial u_{n_u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \end{bmatrix} = \dots, \quad (2.16b)$$

$$\mathbf{C}_c = \frac{\partial \mathbf{h}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \cdots & \frac{\partial h_1}{\partial x_{n_x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{n_y}}{\partial x_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \cdots & \frac{\partial h_{n_y}}{\partial x_{n_x}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \end{bmatrix} = \dots, \quad (2.16c)$$

$$\mathbf{D}_c = \frac{\partial \mathbf{h}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \cdots & \frac{\partial h_1}{\partial u_{n_u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{n_y}}{\partial u_1} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} & \cdots & \frac{\partial h_{n_y}}{\partial u_{n_u}} \Big|_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \end{bmatrix} = \dots. \quad (2.16d)$$

S využitím vyššie definovaných matíc môžeme Rov. (2.14) a (2.15) prepísat do nasledujúceho tvaru, predstavujúceho spojity linearizovaný časovo nepremenlivý (invariantný) stavový model:

$$\Delta \dot{\mathbf{x}} = \mathbf{A}_c \Delta \mathbf{x} + \mathbf{B}_c \Delta \mathbf{u} \quad \left\{ \begin{array}{l} \Delta \dot{x}_1 = a_{11} \Delta x_1 + \cdots + a_{1n_x} \Delta x_{n_x} \\ \quad + b_{11} \Delta u_1 + \cdots + b_{1n_u} \Delta u_{n_u}, \\ \vdots \\ \Delta \dot{x}_{n_x} = a_{n_x 1} \Delta x_1 + \cdots + a_{n_x n_x} \Delta x_{n_x} \\ \quad + b_{n_x 1} \Delta u_1 + \cdots + b_{n_x n_u} \Delta u_{n_u}, \end{array} \right. \quad (2.17a)$$

$$\Delta \mathbf{y} = \mathbf{C}_c \Delta \mathbf{x} + \mathbf{D}_c \Delta \mathbf{u} \quad \left\{ \begin{array}{l} \Delta y_1 = c_{11} \Delta x_1 + \cdots + c_{1n_x} \Delta x_{n_x} \\ \quad + d_{11} \Delta u_1 + \cdots + d_{1n_u} \Delta u_{n_u}, \\ \vdots \\ \Delta y_{n_y} = c_{n_y 1} \Delta x_1 + \cdots + c_{n_y n_x} \Delta x_{n_x} \\ \quad + d_{n_y 1} \Delta u_1 + \cdots + d_{n_y n_u} \Delta u_{n_u}, \end{array} \right. \quad (2.17b)$$

ktorého Rov. (2.17a)–(2.17b) predstavujú linearizácie nelineárnych rovníc (2.10a)–(2.10b). Je teda zrejmé, že  $\Delta \dot{\mathbf{x}}$  a  $\Delta \mathbf{y}$  sú lineárnymi kombináciami odchýlok stavov a vstupov systému, a aproximácia je tým presnejšia, čím sú odchýlky  $\Delta \mathbf{x}$  a  $\Delta \mathbf{u}$  menšie. Matica  $\mathbf{A}_c \in \mathbb{R}^{n_x \times n_x}$  sa nazýva matica dynamiky<sup>25</sup>,  $\mathbf{B}_c \in \mathbb{R}^{n_x \times n_u}$  pred-

<sup>25</sup> angl. system matrix

stavuje maticu vstupu, a  $\mathbf{C}_c \in \mathbb{R}^{n_y \times n_x}$  je matica výstupu.  $\mathbf{D}_c \in \mathbb{R}^{n_y \times n_u}$  je matica priamej väzby výstupu na vstupu<sup>26</sup>, a jej vplyv je pri väčšine systémov z dynamického hľadiska zanedbateľný a často sa považuje za nulový. Poznamenajme, že v prípade systému s jedným vstupom a jedným výstupom<sup>27</sup> sa matica  $\mathbf{B}_c$  zredukuje na slípcový vektor,  $\mathbf{C}_c$  na riadkový vektor a  $\mathbf{D}_c$  na skalár.

Všimnime si tiež, že výsledkom parciálnych derivácií v (2.16a)–(2.16d) sú konštanty, a teda koeficienty matíc  $\mathbf{A}_c$ ,  $\mathbf{B}_c$ ,  $\mathbf{C}_c$  a  $\mathbf{D}_c$  v odvodenom tvare sa nemenia v čase. Ak je však pôvodný nelineárny stavový premenlivý v čase, resp. s premenlivými parametrami, budú aj koeficienty uvedených matíc linearizovaného modelu vo všeobecnosti funkciami spojitého času, čo ďalej viedie na linearizovaný, časovo premenlivý<sup>28</sup> (variantný) stavový model v tvare:

$$\Delta \dot{\mathbf{x}} = \mathbf{A}_c(t) \Delta \mathbf{x} + \mathbf{B}_c(t) \Delta \mathbf{u}, \quad (2.18a)$$

$$\Delta \mathbf{y} = \mathbf{C}_c(t) \Delta \mathbf{x} + \mathbf{D}_c(t) \Delta \mathbf{u}. \quad (2.18b)$$

Aby sme zjednodušenili výklad v nasledujúcich kapitolách, zanedbáme v prípade linearizovaných systémov značenie odchýlok  $\Delta$  v (2.17a)–(2.17b), a teda linearizovaný, časovo invariantný stavový model budeme zapisovať v nasledujúcom, bežne používanom zjednodušenom tvare:

$$\dot{\mathbf{x}} = \mathbf{A}_c \mathbf{x} + \mathbf{B}_c \mathbf{u}, \quad (2.19a)$$

$$\mathbf{y} = \mathbf{C}_c \mathbf{x} + \mathbf{D}_c \mathbf{u}. \quad (2.19b)$$

Pripomeňme však, že z príne matematického pohľadu je takýto zápis možný iba v prípade, že ide o prirodzené lineárny systém, alebo v prípade linearizovaného modelu alebo systému, ak  $\bar{\mathbf{x}} = \mathbf{0}$ ,  $\bar{\mathbf{u}} = \mathbf{0}$  a  $\mathbf{h}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbf{0}$ , kedy  $\Delta \mathbf{x} = \mathbf{x}$ ,  $\Delta \mathbf{u} = \mathbf{u}$  a  $\Delta \mathbf{y} = \mathbf{y}$ .

Spomeňme si ešte mimoriadnu dôležitosť matice dynamiky  $\mathbf{A}_c$ , ktorá je nositeľom dynamických vlastností systému, a môže byť použitá na získanie všeobecného riešenia stavovej rovnice spojitého lineárneho (časovo invariantného) modelu v nasledujúcom tvare [129]:

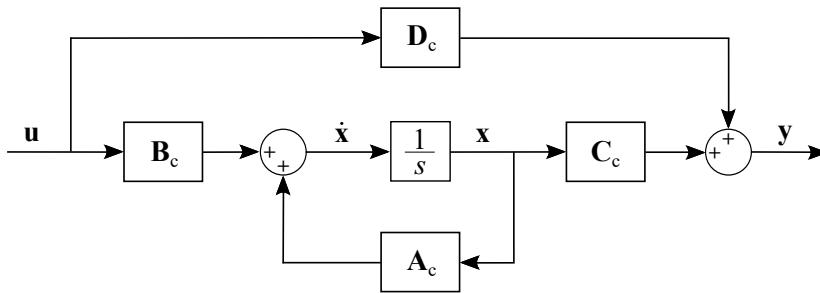
$$\mathbf{x}(t) = e^{\mathbf{A}_c(t-t_0)} \mathbf{x}(t_0) + \int_{t_0}^t e^{\mathbf{A}_c(t-\tau)} \mathbf{B}_c \mathbf{u}(\tau) d\tau,$$

kde prvý výraz na pravej strane predstavuje voľnú odozvu a druhý výraz vynútenú odozvu systému, pričom  $\mathbf{x}(t_0)$  je počiatočný stav systému v čase  $t_0$ . Matica dynamiky  $\mathbf{A}_c$  je preto, okrem iného, dôležitá pri analýze stability a spolu s maticami  $\mathbf{B}_c$  a  $\mathbf{C}_c$  aj riaditeľnosti a pozorovateľnosti lineárnych systémov.

<sup>26</sup>angl. feedthrough matrix

<sup>27</sup>angl. single-input single-output (SISO) system

<sup>28</sup>angl. time-variant (model)



Obr. 2.6. Bloková schéma spojitého LTI stavového modelu<sup>30</sup>.

Spojité lineárna, resp. linearizovaná časovo invariantná<sup>29</sup> (ďalej len LTI) stavovopriestorová reprezentáciu systému vo formálne zjednodušenom tvare podľa (2.19) je znázornená v blokovej schéme na Obr. 2.6.



Zapamäťajme si, že diferenciálne stavové rovnice (2.10a) a (2.17a) opisujú dynamiku toho istého systému. Rozdielom však je, že Rov. (2.17a):

- je len aproximáciou (2.10a) v danom pracovnom bode;
- používa odchýlkové premenné;
- je lineárna s konštantnými koeficientmi.

Ako sme si ukázali, Jacobiho linearizáciou nelineárneho modelu môžeme získať linearizovaný model, ktorý vieme rozumne aproximovať správanie sa systému v blízkom okolí daného pracovného bodu. Lineárny stavový model je tiež východiskom pre návrh moderných metód riadenia, akou je napríklad lineárne prediktívne riadenie. Ak navrhнемe regulátor, ktorý je schopný efektívne riadiť odchýlky stavov, bude pracovať spoločne s tým, že systém nachádzať v blízkom okolí pracovného bodu. V prípade, že systém chceme riadiť vo viacerých pracovných režimoch, podobne môžeme navrhnúť tzv. siet linearizovaných modelov a na základe nich lokálne lineárne regulátory, resp. adaptívny regulátor.

Zapamäťajme si tiež, že pôvodný nelineárny model nestráca svoj význam, a okrem situácií, keď je jeho použitie nevyhnutné, ho tiež často používame napríklad v numerických simuláciách lineárneho riadenia na simuláciu dynamiky sa reálneho systému.

<sup>29</sup>angl. linear time-invariant (LTI)

<sup>30</sup>Symbol  $\frac{1}{s}$  sa v oblasti riadenia často používa na označenie integrácie. V Laplaceovej  $s$ -oblasti platí, že:  $\mathcal{L}\{\int f(t)dt\} = \frac{1}{s}F(s)$ ; v našom prípade:  $\mathcal{L}\{x(t)\} = \mathcal{L}\{\int \dot{x}(t)dt\} = \frac{1}{s}sX(s) = \frac{1}{s}\mathcal{L}\{\dot{x}(t)\}$ .

Pripomeňme si, že ak uvažujeme už lineárny systém, opísaný napríklad lineárrou diferenciálnou rovnicou s konštantnými koeficientmi (pri  $n = m$ ) v tvare

$$\frac{d^n y}{dt^n} + a_1 \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_{n-1} \frac{dy}{dt} + a_n y = b_0 \frac{d^n u}{dt^n} + b_1 \frac{d^{n-1} u}{dt^{n-1}} + \cdots + b_{n-1} \frac{du}{dt} + b_n u,$$

kde  $u$  označuje vstup a  $y$  výstup, možno LTI stavový model získať priamo z tej pomociou už spomínaného prepisu do stavového priestoru. K jednému vstupno-výstupnému modelu vieme teoreticky priradiť nekonečne veľa stavových modelov. Medzi najznámejšie formy lineárnej stavovopriestorovej reprezentácie z pohľadu návrhu regulátora, resp. pozorovateľa stavu patria *kanonický tvar riaditeľnosti*:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}}_{\mathbf{A}_{c,riad}} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{B}_{c,riad}} u, \quad (2.20)$$

$$y = \underbrace{\begin{bmatrix} b_n - a_n b_0 & b_{n-1} - a_{n-1} b_0 & \cdots & b_1 - a_1 b_0 \end{bmatrix}}_{\mathbf{C}_{c,riad}} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \underbrace{b_0}_{\mathbf{D}_{c,riad}} u,$$

a *kanonický tvar pozorovateľnosti* stavového modelu:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 & -a_n \\ 1 & 0 & \cdots & 0 & -a_{n-1} \\ 0 & 1 & \cdots & 0 & -a_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_1 \end{bmatrix}}_{\mathbf{A}_{c,poz}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \underbrace{\begin{bmatrix} b_n - a_n b_0 \\ b_{n-1} - a_{n-1} b_0 \\ b_{n-2} - a_{n-2} b_0 \\ \vdots \\ b_1 - a_1 b_0 \end{bmatrix}}_{\mathbf{B}_{c,poz}} u, \quad (2.21)$$

$$y = \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix}}_{\mathbf{C}_{c,poz}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \underbrace{b_0}_{\mathbf{D}_{c,poz}} u,$$

pričom platí, že  $\mathbf{A}_{c,obs} = \mathbf{A}_{c,riad}^T$ ,  $\mathbf{B}_{c,obs} = \mathbf{C}_{c,riad}^T$ ,  $\mathbf{C}_{c,obs} = \mathbf{B}_{c,riad}^T$  a  $\mathbf{D}_{c,obs} = \mathbf{D}_{c,riad}$ .

Typickú štruktúru matíc uvedených kanonických tvarov LTI stavového modelu si môžeme porovnať s vo všeobecnosti plnými maticami získanými linearizáciou nelineárneho stavového modelu; pozri napr. (2.16a).

Ukážme si ďalej na príklade invertovaného kyvadla, ako môžeme linearizáciu systému vykonať numericky.

**Príklad 2.4.** Odvodte linearizovaný model invertovaného kyvadla na vozíku opísaného nelineárny časovo invariantným stavovým modelom (2.9). Ako pracovný bod uvažujte jeho stabilnú rovnovážnu polohu.

Zo schémy na Obr. 2.2 je zrejmé, že ak sa systém nachádza v pokoji, je jeho fyzikálna konfigurácia s kyvadlom visiacim v dolnej, stabilnej rovnovážnej polohe, ktorej zodpovedá pracovný bod opísaný stavom  $\bar{\mathbf{x}} = [\bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4]^T = [0 0 \pi 0]^T$  a vstupom  $\bar{u} = 0$ .

Matice príslušného linearizovaného stavového modelu získame Jacobiho linearizáciou, pozri (2.16), v tvare:

$$\begin{aligned}
 \mathbf{A}_c &= \frac{\partial \mathbf{f}(\mathbf{x}, u)}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}, \bar{u}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} \Big|_{\bar{\mathbf{x}}, \bar{u}} & \cdots & \frac{\partial f_1}{\partial x_4} \Big|_{\bar{\mathbf{x}}, \bar{u}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_4}{\partial x_1} \Big|_{\bar{\mathbf{x}}, \bar{u}} & \cdots & \frac{\partial f_4}{\partial x_4} \Big|_{\bar{\mathbf{x}}, \bar{u}} \end{bmatrix}, \\
 \mathbf{B}_c &= \frac{\partial \mathbf{f}(\mathbf{x}, u)}{\partial u} \Big|_{\bar{\mathbf{x}}, \bar{u}} = \begin{bmatrix} \frac{\partial f_1}{\partial u} \Big|_{\bar{\mathbf{x}}, \bar{u}} \\ \vdots \\ \frac{\partial f_4}{\partial u} \Big|_{\bar{\mathbf{x}}, \bar{u}} \end{bmatrix}, \\
 \mathbf{C}_c &= \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} \Big|_{\bar{\mathbf{x}}} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} \Big|_{\bar{\mathbf{x}}} & \cdots & \frac{\partial h_1}{\partial x_4} \Big|_{\bar{\mathbf{x}}} \\ \frac{\partial h_2}{\partial x_1} \Big|_{\bar{\mathbf{x}}} & \cdots & \frac{\partial h_2}{\partial x_4} \Big|_{\bar{\mathbf{x}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} \Big|_{\bar{\mathbf{x}}} & \cdots & \frac{\partial h_n}{\partial x_4} \Big|_{\bar{\mathbf{x}}} \end{bmatrix}, \\
 \mathbf{D}_c &= \frac{\partial \mathbf{h}(\mathbf{x})}{\partial u} \Big|_{\bar{\mathbf{x}}} = \begin{bmatrix} \frac{\partial h_1}{\partial u} \Big|_{\bar{\mathbf{x}}, \bar{u}} \\ \frac{\partial h_2}{\partial u} \Big|_{\bar{\mathbf{x}}, \bar{u}} \\ \vdots \\ \frac{\partial h_n}{\partial u} \Big|_{\bar{\mathbf{x}}, \bar{u}} \end{bmatrix}.
 \end{aligned} \tag{2.22}$$

Parciálne derivácie potrebné pre výpočet týchto matíc môžeme efektívne vykonať a numericky vyčísliť v prostredí programu MATLAB, ako si ukážeme ďalej. Predpokladáme samozrejme, že funkcie  $\mathbf{f}$  a  $\mathbf{h}$  sú diferencovateľné.

Ešte predtým si tiež všimnime, že matice  $\mathbf{B}_c$  a  $\mathbf{D}_c$  sa v tomto prípade redukujú na stĺpcové vektory, keďže náš model má iba jeden vstup. Matice  $\mathbf{C}_c$  a  $\mathbf{D}_c$  by sme tiež vedeli ľahko vypočítať aj ručne, keďže rovnica výstupu je lineárna a  $\mathbf{h}(\mathbf{x})$  nie je funkciou vstupu.

Ďalej uvedený skript *linearizacia.m* si formálne uvedieme vyčistením pracovného priestoru a okna MATLABu, načítaním parametrov systému (pozri Zdrojový kód 2.1) a ďalej pokračujeme definovaním tzv. symbolických premenných<sup>31</sup>:

Zdrojový kód 2.3. linearizacia.m

```
clear; clc
load param.mat % Nacitanie parametrov
syms dxdt y x1 x2 x3 x4 u % Symbolicke premenne
```

Nasleduje implementácia stavovej rovnice (2.9a) a rovnice výstupu (2.9b) vo vektorovom tvare:

```
dxdt=[x2; % Stavova rovnica
       (-m2*g*sin(x3)*cos(x3)+m2*l*((x4)^2)*sin(x3)+b*m2*
       x4*cos(x3)+u) / (m1+((sin(x3))^2)*m2);
       x4;
       ((m1+m2)*(g*sin(x3)-b*x4)-(m2*l*((x4)^2)*sin(x3)+u*
       )*cos(x3)) / (l*(m1+((sin(x3))^2)*m2))];
y=[x1; % Rovnica vystupu
   x3];
```

Tieto rovnice ďalej linearizujeme<sup>32</sup> tak, ako je uvedené v (2.22), na čo použijeme príkaz *jacobian()*, ktorý vypočíta Jacobiho matice funkcií *f* a *h* podľa *x* a *u*.

```
Alin=jacobian(dxdt,[x1 x2 x3 x4]);
Blin=jacobian(dxdt,u);
Clin=jacobian(y,[x1 x2 x3 x4]);
Dlin=jacobian(y,u);
```

Získané Jacobiho matice však obsahujú symbolické premenné a množstvo zložitých matematických výrazov, ktoré sú výsledkom parciálnych derivácií daných funkcií podľa týchto premenných. Preto, aby sme získali matice linearizovaného stavového modelu, musíme Jakobiány ešte vyhodnotiť v pracovnom bode  $(\bar{x}, \bar{u})$ , t.j.

```
xlin=[0 0 pi 0];
ulin=0;
```

čo vykonáme príkazom *subs()*, ktorý nahradí symbolické premenné, *x* a *u*, v jednotlivých Jakobiánoch uvažovaným pracovným bodom daným stavmi  $\bar{x}$  a vstupom  $\bar{u}$ , a výstup ktorého ďalej numericky zjednodušíme príkazom *eval()*:

<sup>31</sup>Použitie symbolických premenných vo výpočtoch implikuje, že ich MATLAB musí vykonať analyticky, nie numericky. V našom prípade ide o operáciu parciálnej derivácie.

<sup>32</sup>V podstate formálne linearizujeme len nelineárnu stavovú rovnicu, keďže rovnica výstupu je sama o sebe lineárna.

```

Ac=eval(subs(Alin, [x1 x2 x3 x4 u], [xlin ulin]));
Bc=eval(subs(Blin, [x1 x2 x3 x4 u], [xlin ulin]));
Cc=eval(subs(Clin, [x1 x2 x3 x4], xlin));
Dc=eval(subs(Dlin, [x1 x2 x3 x4], xlin));
save('linearnySM', 'Ac', 'Bc', 'Cc', 'Dc') % Ulozenie matic

```

Výsledkom tejto procedúry sú matice  $\mathbf{A}_c$ ,  $\mathbf{B}_c$ ,  $\mathbf{C}_c$  a  $\mathbf{D}_c$  nižšie uvedeného spojitého časovo invariantného stavového modelu, získaného linearizáciou (2.9) v stabilnej rovnovážnej polohe (ozn. ' $\downarrow$ ') danej pracovným bodom  $(\bar{\mathbf{x}}^T, \bar{u}) = (0 \ 0 \ \pi \ 0 \ 0)$ :

$$\Delta \dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1.962 & -0.02 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -58.86 & -0.6 \end{bmatrix}}_{\mathbf{A}_c^\downarrow} \Delta \mathbf{x} + \underbrace{\begin{bmatrix} 0 \\ 2.5 \\ 0 \\ 12.5 \end{bmatrix}}_{\mathbf{B}_c^\downarrow} \Delta u, \quad (2.23a)$$

$$\Delta \mathbf{y} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{C}_c} \Delta \mathbf{x} + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\mathbf{D}_c} \Delta u. \quad (2.23b)$$

Určitý náhľad do štruktúry modelu by sme získali, ak by sme si jeho fyzikálne parametre zahrnuli medzi symbolické premenné (`syms`) a pri vyčíslení matíc modelu dosadili (`subs`) len pracovný bod a nie ich hodnoty, čím by sme získali:

$$\mathbf{A}_c^\downarrow = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{m_2 g}{m_1} & \frac{m_2 b}{m_1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g(m_1+m_2)}{m_1 l} & -\frac{b(m_1+m_2)}{m_1 l} \end{bmatrix}, \quad \mathbf{B}_c^\downarrow = \begin{bmatrix} 0 \\ \frac{1}{m_1} \\ 0 \\ -\frac{1}{m_1 l} \end{bmatrix}. \quad (2.24)$$

Ak by sme ako pracovný bod uvažovali hornú, nestabilnú rovnovážnu polohu (' $\uparrow$ '), linearizovaná rovnica dynamiky by pre porovnanie mala nasledujúci tvar<sup>33</sup>:

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1.962 & 0.02 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 58.86 & -0.6 \end{bmatrix}}_{\mathbf{A}_c^\uparrow} \mathbf{x} + \underbrace{\begin{bmatrix} 0 \\ 2.5 \\ 0 \\ -12.5 \end{bmatrix}}_{\mathbf{B}_c^\uparrow} u. \quad (2.25)$$

□

<sup>33</sup>V tomto prípade môžeme použiť jej formálne zjednodušený tvar (2.19a), keďže  $\bar{\mathbf{x}} = \mathbf{0}$  a  $\bar{u} = 0$ , a teda  $\Delta \mathbf{x} = \mathbf{x}$  a  $\Delta u = u$ .

Podobne ako v prípade nelineárneho stavového modelu invertovaného kyvadla, odsimulujme si správanie sa linearizovaného stavového modelu tohto systému.

**Príklad 2.5.** Vykonajte numerickú simuláciu LTI stavového modelu (2.23) získaného linearizáciou nelineárneho stavového modelu (2.9) invertovaného kyvadla v jeho stabilnej rovnovážnej polohe.

Príklad riešime v skripte, kde začneme načítaním získaných matíc linearizovaného stavového modelu (pozri Zdrojový kód 2.3) a vytvorením objektu stavového modelu pomocou príkazu `ss()`.

Zdrojový kód 2.4. simlinkyvadlo.m

```
clear; clc; close all
load linearnySM          % Nacitanie matic stav. modelu
sys=ss(Ac,Bc,Cc,Dc);    % Vytvor. objektu stav. modelu
```

Pokračujeme zadefinovaním simulačného času a vektoru počiatočných stavov systému  $\Delta\mathbf{x}_0$ , ktoré sú v odchýlkovom tvaru (Rov. (2.9a)). Ak teda uvažujeme počiatočnú výchylku kyvadla napr.  $30^\circ$  od jeho stabilnej rovnovážnej polohy, t. j.  $\frac{5}{6}\pi$  v zmysle Obr. 2.2, platí, že  $\Delta x_{3,0} = x_{3,0} - \bar{x}_3 = \frac{5}{6}\pi - \pi = -\frac{\pi}{6}$ .

```
cas=0:0.01:10;           % Trvanie a krok simulacie
dx0=[0;0; -pi/6; 0];    % Pociatocny stav systemu
```

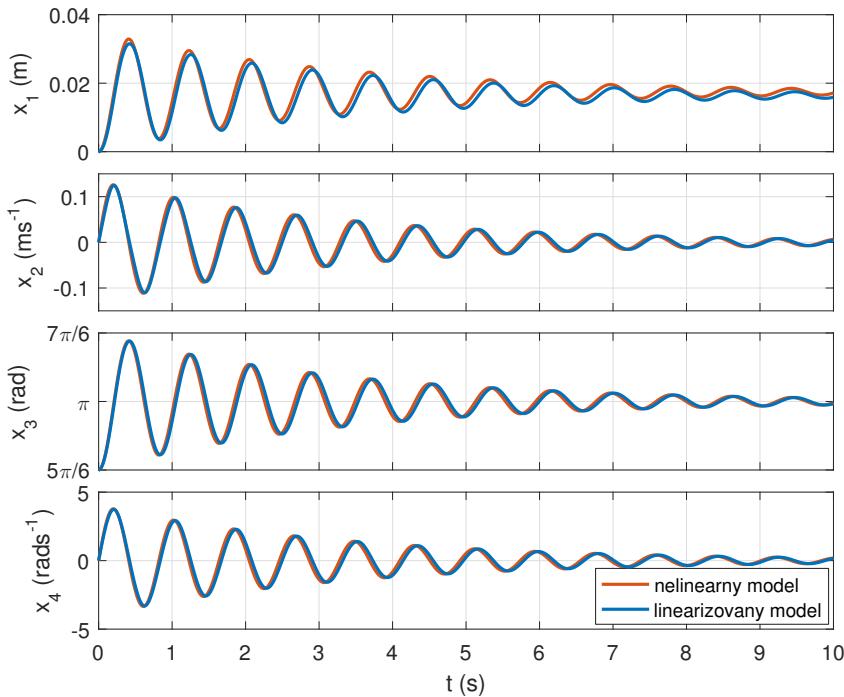
Ak uvažujeme simuláciu voľnej odozvy modelu, môžeme Rov. (2.9a) numericky integrovať podobne ako v prípade nelineárneho modelu (pozri Zdrojový kód 2.1). My ale namiesto toho efektívne použíme príkaz `initial()`, ktorý vypočíta voľnú odozvu systému `sys()` na počiatočný stav `dx0` (t. j. bez uvažovania vstupu):

```
[dy,t,dx]=initial(sys,dx0,cas); % Simulacia volnej odozvy
```

a ktorého výstupom je matica simulovaných (odchýliek) stavov  $\mathbf{x}$  0 ako aj uvažovaných výstupov  $\mathbf{y}$  systému. Pred vykreslením získaných priebehov veličín si tieto, konkrétnie uhlovú výchylku kyvadla, z praktických dôvodov transformujme z odchýlkového tvaru späť do pôvodných súradníc ( $\Delta x_3 = x_3 - \bar{x}_3 \Rightarrow x_3 = \Delta x_3 + \bar{x}_3$ ):

```
x=dx;                      % Transf. súradnic na x a y
x(:,3)=dx(:,3)+pi;          % ...manualne
y=(Cc*x')';                % ...pomocou matice vystupu
```

Výsledky simulácie voľnej odozvy linearizovaného modelu si na záver vykreslíme identickým spôsobom ako v Zdrojovom kóde 2.1. Na Obr. 2.7 sú modrou farbou znázornené priebehy stavových veličín v čase. Tieto sú pre porovnanie doplnené aj o priebehy stavov získané simuláciou plne nelineárneho modelu (2.9), znázornené červenou farbou.

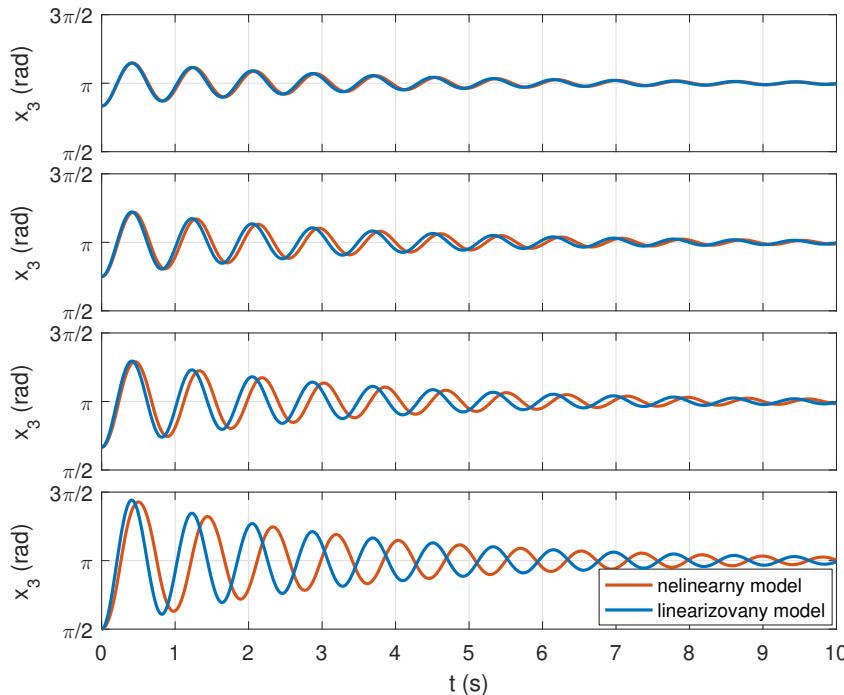


Obr. 2.7. Simulácia voľnej odozvy linearizovaného modelu invertovaného kyvadla.

Z Obr. 2.7 – s dôrazom na priebeh tretieho stavu, t. j. uhlovej výchylky kyvadla – je zrejmé, že model získaný linearizáciou v okolí dolnej, stabilnej rovnovážnej polohy ( $\bar{x}_3 = \pi$ ) v okolí  $\pm 30^\circ$  ( $\pm \frac{\pi}{6}$ ) veľmi dobre approximuje správanie sa nelineárneho modelu systému. Táto approximácia je samozrejme tým lepšia, v čím bližšom okolí pracovného bodu  $(\bar{x}^T, \bar{u}) = (0 \ 0 \ \pi \ 0 \ 0)$  sa systém pohybuje. Tento fakt je ilustrovaný na Obr. 2.8, ktorý na príklade uhlovej výchylky kyvadla znázorňuje efekt jej počiatočnej hodnoty na simuláciu voľnej odozvy systému. Ako vidíme, čím „dalej“ od pracovného bodu simuláciu inicializujeme, tým menej linearizovaný model reprezentuje skutočnú dynamiku systému.

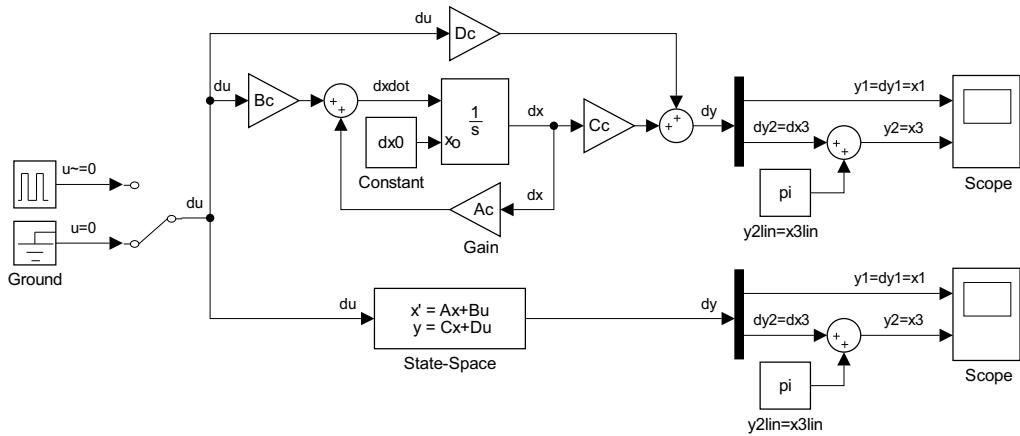
Podobným spôsobom by sme mohli simulať aj vynútenú odozvu linearizovaného, resp. lineárneho modelu nášho systému; napríklad jeho prechodovú odozvu (príkaz `step()`), impulznú odozvu (`impulse()`), alebo aj odozvu na všeobecný vstupný signál (`lsim()`).

Poznamenajme, že simulať by sme mohli aj model linearizovaný vzladom na hornú, nestabilnú rovnovážnu polohu systému (pozri model (2.25)), ale jeho odozva by však bez použitia regulátora, ktorý by systém v tejto polohe stabilizoval, veľmi rýchlo divergovala a simulácia by tak nemala žiadnu výpovednú hodnotu.



Obr. 2.8. Porovnanie voľnej odozvy (v zmysle uhlovej výchylky kyvadla) získanej simuláciou nelineárneho a linearizovaného modelu s uvažovaním rôznych počatočných výchyliek kyvadla od jeho pracovnej – stabilnej rovnovážnej polohy, v poradí ( $\Delta x_{3,0} = x_{3,0} - \pi =$ )  $-\frac{\pi}{6}$ ,  $-\frac{\pi}{4}$ ,  $-\frac{\pi}{3}$  a  $-\frac{\pi}{2}$ .

Pre úplnosť, správanie sa nášho linearizovaného modelu invertovaného kyvadla môžeme tiež veľmi jednoducho numericky odsimulovať v prostredí Simulink. Obr. 2.9 znázorňuje dva spôsoby implementácie linearizovaného, resp. lineárneho spojitého stavového modelu. Horná časť blokovej schémy ilustruje intuitívny postup vyskladania modelu pomocou blokovej interpretácie rovnice dynamiky a rovnice výstupu tak, ako sme si ukázali na Obr. 2.6. Značne jednoduchšie je však použiť preddefinovaný blok *State-Space*, ktorý nájdeme v knižnici Simulinku v triede *Continuous*. Po jeho rozkliknutí si môžeme jednoducho nastaviť parametre stavového modelu, t. j. matice stavového modelu a vektor počiatočných stavov. Použijeme na to premenné, ktoré máme (po spustení skriptu *simlinkyvadlo.m*) uložené v pracovnom priestore MATLABu. Rozdielom oproti prvému spôsobu je, že blok *State-Space* dáva na výstupe rovnako ako reálny systém výstupnú veličinu, t. j. stav je jeho vnútornou premennou. Plný stavový vektor môžeme získať jednoduchým trikom – tak, že namiesto namiesto získaných matíc  $\mathbf{C}_c$  a  $\mathbf{D}_c$  použijeme ma-



Obr. 2.9. Dva ekvivalentné spôsoby interpretácie a simulácie linearizovaného modelu invertovaného kyvadla pomocou blokovej schémy v prostredí Simulink.

tice `eye (length (Ac))` (jednotková matica) a `zeros (length (Ac), 1)` (nulový vektor), t. j. uvažujeme, že všetky stavy systému sú merané. Ak nám však postačuje vykresľovať samotné výstupy systému (prvý a tretí stav), tak postupujeme ako na Obr. 2.9. Posledným krokom pred vykreslením je ich už spomínaná spätná transformácia z odchýlkového tvaru; v tomto prípade prakticky len uhlovej výchylky kyvadla. V prípade voľnej odozvy môžeme nulový vstup interpretovať pomocou bloku *Ground* (ekvivalent k *Constant = 0*); v prípade vynútenej odozvy si z knižnice vyberieme niektorý z blokov z triedy *Sources*. Získaná simulovaná odozva bude samozrejme identická ako tá, získaná pomocou skriptu.

Poznamenajme, že v rovnakej blokovej schéme môžeme simulovať aj pôvodný—spojitý nelineárny stavový model. Stačí, ak napríklad do schémy na Obr. 2.9 vložíme blok *Subsystem*, do ktorého prekopírujeme nelineárny dynamický model zo schémy na Obr. 2.3 a na jeho vstupný port pripojíme spoločný vstupný signál. Tákymto spôsobom môžeme alternatívne získať priebehy na Obr. 2.8 porovnávajúce správanie sa jednotlivých modelov v okolí pracovného bodu.

□



Ak pri návrhu lineárneho, napríklad prediktívneho regulátora vychádzame z linearizovaného modelu systému, dbajme pri jeho odvodení na rozumnú voľbu pracovného bodu a jeho správanie sa overme pomocou simulácie.

Na záver tejto podkapitoly si pripomeňme, že štruktúru spojitého lineárneho stavového modelu, napríklad modelu systému invertovaného kyvadla (2.24), získaného jeho linearizáciou v jeho dolnej, stabilnej rovnovážnej polohe, by sme mohli získať aj *analyticko-experimentálnym*, tzv. grey-box prístupom k identifikácii.

Tento prístup je obzvlášť zaujímavý ak si nie sme istí určitými parametrami daného modelu; napríklad ak veľkosť daného parametra bola len odhadnutá alebo približne určená analyticky. V prípade modelu invertovaného kyvadla, (2.24), by sme vo všeobecnosti za neznámy alebo neurčitý parameter mohli považovať napríklad koeficient viskózneho tlmenia  $b$ . Kedže poznáme matematickú štruktúru daného dynamického modelu (apriórne informácie), pomocou grey-box identifikácie môžeme na základe dát (aposteriárnych informácií) získaných počas pozorovania alebo experimentovania s reálnym systémom veľkosť daného parametra odhadnúť, a tak upresniť štruktúru modelu. Kvalitný odhad daného parametra samozrejme predpokladá experimentálne vstupno-výstupné dátá s dostatočne informatívou hodnotou. Toto v praxi dosiahneme voľbou vhodného budiaceho signálu (napr. náhodný, pseudonáhodný, periodický a iné signály), ktorý musí dostatočne vybudíť dynamiku systému v požadovanom okolí pracovného bodu. Správanie sa získaného grey-box modelu (model vo forme sivej skrinky) je ďalej vhodné simulačne overiť porovnaním jeho dynamickej odozvy s odozvou reálneho systému.

Všetky spomínané úkony možno efektívne realizovať pomocou funkcií a blokov knižnice System Identification Toolbox v prostredí programu MATLAB, resp. Simulink. Pre odhad parametrov lineárnych, resp. nelineárnych grey-box modelov táto ponúka viaceré efektívne funkcie, ako napr. `greyest()` a `pem()`, resp. `nlgreyest`. Veľmi oblúbený je spomínaný PEM<sup>34</sup> algoritmus využívajúci numerickú minimalizáciu chyby predikcie, t. j. rozdielu medzi nameraným výstupom a predikovaným výstupom modelu, na aktualizáciu (upresnenie) parametra/parametrov „počiatočného modelu“ [97]. Pre viac informácií pozri [103].

Poznamenajme, že náš lineárny stavový model by sme mohli získať aj výlučne *experimentálnym*, takzvaným black-box prístupom k identifikácii. V tomto prípade je pre nás štruktúra modelu neznáma (pričom jeho rád je voliteľný) a numericky ju získame pomocou zvoleného algoritmu odhadu prispôsobovaním nameraných dát. V MATLABe nato môžeme použiť napríklad funkcie `ssest()` alebo `n4sid()` (podpriestorová metóda identifikácie [97]). Nevýhodou v tomto prípade je však nevyhnutná strata súvislosti s fyzikálnym systémom (model vo forme čiernej skrinky), t. j. jednotlivé stavy systému nemajú už svoj fyzikálny význam. V prípade invertovaného kyvadla, ktorého dynamický model je pomerne jednoducho odvoditeľný, bez náročne modelovateľných dynamických javov a approximácií, nemá použitie iba samotnej experimentálnej identifikácie význam.

<sup>34</sup>angl. prediction error method (PEM)

## 2.3 Diskretizácia a diskrétny stavový model

Vo všeobecnosti rozlišujeme dva základné typy tzv. diskrétnych systémov:

- systémy, ktoré sú dôsledkom vzorkovania výstupu spojitéh systémov iba v určitých diskrétnych časových okamihoch; a
- systémy s prirodzene diskrétnym charakterom, ktorých stavy sú definované iba v diskrétnych časových okamihoch a čo sa deje medzi nimi nás nezaujíma.

Nás samozrejme bude ďalej zaujímať prvý z nich, t. j. spojité dynamické systémy, ktorých výstupom sú vzorkované dátá. Ako je totiž všeobecne známe, väčšina reálnych systémov alebo procesov má spojité dynamiku, či už sú to napríklad chemické reakcie, trajektórie letu raket, elektrárne, topenie ľadovcov, alebo pre nás zaujímavé mechatronické systémy. Počítače však v súčasnosti pracujú takmer výlučne diskrétnie. Preto, ak chceme simulovať spojité systém diskrétnym zariadením, potrebujeme určitú metodiku na konverziu spojitého modelu na diskrétny. Proces transformácie spojitého (analógového) systému tak, aby bol použiteľný diskrétny (hovoríme tiež číslicovo, resp. digitálne) pracujúcim hardvérom sa nazýva *diskretizácia*. Diskretizácia je taktiež používaná v riadení, keďže na to aby sme navrhli diskrétny regulátor potrebujeme poznáť diskrétny model systému, ktorý je sám o sebe obyčajne opísaný pomocou diferenciálnych rovníc.

Ak teda zamýšľame použiť už odvodený spojité lineárny, alebo linearizovaný model pre návrh lineárneho prediktívneho regulátora, ukážme si spôsob akým ho prevedieme na jeho diskrétnu formu, t. j. *diskrétny lineárny stavový model*<sup>35</sup>.

Keďže prediktívny regulátor bude v nasledujúcich kapitolách odvodený v diskrétnej forme, výstupy zo systému (merania), ktoré sú vstupom pre riadiaci systém, musia byť *vzorkované*. Ak daný vzorkovač pracuje so *vzorkovacou* períodou  $T_s$ , označme si vzorkované hodnoty meraných veličín ako

$$\mathbf{y}(k) = \mathbf{y}(t_k), \quad t_k = kT_s, \quad k = 0, 1, 2, 3, \dots$$

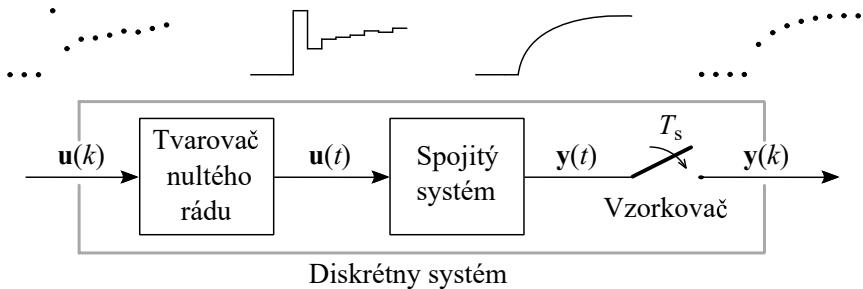
Výstup z regulátora bude nadobúdať diskrétné hodnoty, ktoré si označme ako  $\mathbf{u}(k)$ . Samotný vstup do riadeného systému je zvyčajne získaný spracovaním  $\mathbf{u}(k)$  tzv. *tvarovačom nultého rádu*<sup>36</sup> (ZOH), ktorý zabezpečí, že vstup medzi jednotlivými vzorkovacími okamihmi je podržaný, t. j.

$$\mathbf{u}(t) = \mathbf{u}(k), \quad t_k < t < t_{k+1},$$

je konštantný [51]. Predpokladáme, že vzorkovanie sa realizuje pravidelne a períoda vzorkovania je daná vzťahom  $t_{k+1} - t_k = T_s$ . Uvedený diskrétny, tzv. ZOH ekvivalent je ilustrovaný na Obr. 2.10.

<sup>35</sup>angl. discrete-time linear state-space model

<sup>36</sup>angl. zero-order hold (ZOH)



Obr. 2.10. Diskretizácia spojitého systému pomocou ZOH ekvivalentu<sup>37</sup>.

Ako sme už načrtli, uvažujme, že nás systém je opísaný spojitým, lineárnym časovo invariantným stavovým modelom v nasledujúcom tvaru<sup>38</sup>:

$$\dot{\mathbf{x}}(t) = \mathbf{A}_c \mathbf{x}(t) + \mathbf{B}_c \mathbf{u}(t), \quad (2.26a)$$

$$\mathbf{y}(t) = \mathbf{C}_c \mathbf{x}(t) + \mathbf{D}_c \mathbf{u}(t). \quad (2.26b)$$

Poznamenajme najprv, že vzorkované hodnoty výstupu  $\mathbf{y}$  splňajú nasledovné:

$$\mathbf{y}(k) = \mathbf{y}(t_k) = \mathbf{C}_c \mathbf{x}(t_k).$$

Ako už vieme, riešenie stavovej rovnice (2.26a) má tvar:

$$\mathbf{x}(t) = e^{\mathbf{A}_c(t-t_0)} \mathbf{x}(t_0) + \int_{t_0}^t e^{\mathbf{A}_c(t-\tau)} \mathbf{B}_c \mathbf{u}(\tau) d\tau.$$

Uvažujme, že  $t_0 = t_k$  a  $t = t_{k+1}$ , a po úprave dostávame:

$$\begin{aligned} \mathbf{x}(t_{k+1}) &= e^{\mathbf{A}_c T_s} \mathbf{x}(t_k) + \int_{t_k}^{t_{k+1}} e^{\mathbf{A}_c(t_{k+1}-\tau)} \mathbf{B}_c \mathbf{u}(\tau) d\tau, \\ &= e^{\mathbf{A}_c T_s} \mathbf{x}(t_k) + \int_{t_k}^{t_{k+1}} e^{\mathbf{A}_c(t_{k+1}-\tau)} \mathbf{B}_c d\tau \mathbf{u}(k). \end{aligned} \quad (2.27)$$

Ak v druhom člene na pravej strane Rov. (2.27) zavedieme substitúciu  $\tau' = t_{k+1} - \tau$ , bude pre tento platí:

$$\int_{t_k}^{t_{k+1}} e^{\mathbf{A}_c(t_{k+1}-\tau)} \mathbf{B}_c d\tau = \int_{T_s}^0 e^{\mathbf{A}_c \tau'} (-d\tau') \mathbf{B}_c = \int_0^{T_s} e^{\mathbf{A}_c \tau} d\tau \mathbf{B}_c.$$

<sup>37</sup>Uvedená bloková schéma je formálne zjednodušená. V praxi je tvarovač súčasťou digitálno-analógového (D/A) prevodníka a vzorkovač súčasťou analógovo-digitálneho (A/D) prevodníka. A/D konverzia pozostáva z operácií vzorkovania, kvantovania a kódovania, ktorými daný signál diskretizujeme. D/A konverzia naopak prevádzka číslicový signál na spojity. ZOH predstavuje matematický model praktickej rekonštrukcie signálu tak, ako ju vykonáva bežný D/A prevodník.

<sup>38</sup>Všimnime si, že pre jednoduchosť uvažujeme štandardný tvar spojitého LTI modelu. V prípade, že ide o linearizovaný model, je potrebné uvažovať jeho odchýlkový tvar (2.19).

Po uvedenej úprave stavovej rovnice a pri platnosti značenia  $t_k = kT_s$  (vzorkovacie okamihy  $t_k$  sú celočíselnými násobkami vzorkovacej periódy  $T_s$ ) dostávame diskrétny LTI stavový model vo formálne zjednodušenom tvare:

$$\mathbf{x}(k+1) = \mathbf{Ax}(k) + \mathbf{Bu}(k), \quad (2.28a)$$

$$\mathbf{y}(k) = \mathbf{Cx}(k) + \mathbf{Du}(k), \quad (2.28b)$$

kde pre jednotlivé jeho matice platí, že:

$$\mathbf{A} = e^{\mathbf{A}_c T_s}, \quad (2.29a)$$

$$\mathbf{B} = \left( \int_0^{T_s} e^{\mathbf{A}\tau} d\tau \right) \mathbf{B}_c = \dots = \mathbf{A}_c^{-1} (\mathbf{A} - \mathbf{I}) \mathbf{B}_c, \text{ ak } \det(\mathbf{A}_c) \neq 0, \quad (2.29b)$$

$$\mathbf{C} = \mathbf{C}_c, \quad (2.29c)$$

$$\mathbf{D} = \mathbf{D}_c. \quad (2.29d)$$

Všimnime si, že na rozdiel od matíc  $\mathbf{A}_c$  a  $\mathbf{B}_c$ , matice  $\mathbf{C}_c$  a  $\mathbf{D}_c$  sa po diskretizácii nemenia, keďže rovnica výstupu nie je dynamickým vzťahom. Z matíc diskrétneho stavového modelu (2.28) si zdôrazníme najmä dôležitosť jeho (vždy nesingulárnej) matice dynamiky  $\mathbf{A}$ , presnejšie nazývanej *stavová matica prechodu*<sup>39</sup>, ktorej súčinom so stavovým vektorom  $\mathbf{x}$  v kroku  $k$  v prípade autonómneho systému (t.j. systému bez vstupov) dostávame stavový vektor v nasledujúcom kroku  $k+1$ :

$$\mathbf{x}(k+1) = \mathbf{Ax}(k).$$

Stavovú maticu prechodu  $\mathbf{A}$  získame zo stavovej matice spojitého LTI stavového modelu,  $\mathbf{A}_c$ , diskretizáciou podľa vzťahu  $\mathbf{A} = e^{\mathbf{A}_c T_s}$ . Vzťah medzi nimi vyjadruje výraz  $e^{\mathbf{A}_c T_s}$ , ktorý predstavuje tzv. maticovú exponenciálnu funkciu (angl. matrix exponential). Túto možno alternatívne reprezentovať pomocou jej rozvoja do konvergujúceho mocninového radu v tvare:

$$e^{\mathbf{A} T_s} = \mathbf{I} + T_s \mathbf{A}_c + \frac{T_s^2}{2!} \mathbf{A}_c^2 + \dots = \sum_{i=0}^{\infty} \frac{1}{i!} T_s^i \mathbf{A}_c^i, \quad (2.30)$$

kde  $\mathbf{I}$  označuje jednotkovú maticu s rovnakými rozmermi ako matice  $\mathbf{A}$ . Je zrejmé, že čím viac členov tohto rozvoja uvažujeme, tým bude diskrétna approximácia spojitej dynamiky presnejšia. Hovoríme o tzv. ráde approximácie.

Poznamenajme, že ak by sme na ľavej strane rovnice dynamiky (2.26a) spojitého LTI stavového modelu nahradili deriváciu stavu jej diskrétnou approximáciou v zmysle *doprednej Eulerovej* metódy, t.j.

$$\dot{\mathbf{x}}(t) = \frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{T_s}, \quad (2.31)$$

<sup>39</sup>angl. state-transition matrix

$\mathbf{x}(t)$  a  $\mathbf{u}(t)$  ich diskrétnymi ekvivalentmi, dostávame:

$$\frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{T_s} = \mathbf{A}_c \mathbf{x}(k) + \mathbf{B}_c \mathbf{u}(k),$$

a po úprave:

$$\mathbf{x}(k+1) = \underbrace{(\mathbf{I} + T_s \mathbf{A}_c)}_{\mathbf{A}} \mathbf{x}(k) + \underbrace{T_s \mathbf{B}_c}_{\mathbf{B}} \mathbf{u}(k), \quad (2.32)$$

čo predstavuje rovnicu dynamiky diskretizovanú pomocou Eulerovej approximácie. Všimnime si, že výraz  $(\mathbf{I} + T_s \mathbf{A}_c)$  predstavuje prvé dva členy mocninového rozvoja (2.30), t. j. exaktnej diskretizácie matice dynamiky (2.29a) a podobne výraz  $T_s \mathbf{B}_c$  by sme dostali ako prvý člen rozvoja  $\left(\int_0^{T_s} e^{\mathbf{A}\tau} d\tau\right) \mathbf{B}_c$  v rámci exaktnej diskretizácie matice vstupu (2.29b). V prípade Eulerovej integrácie teda hovoríme o approximácii prvého rádu.

Eulerova metóda však vyžaduje aby vzorkovacia perióda  $T_s$  bola malá, zatiaľ čo exaktná diskretizácia platí pre akúkoľvek veľkosť  $T_s$ , a preto nie je vo všeobecnosti vhodná (dostatočne presná) z pohľadu ďalšieho návrhu riadenia.

Poznamenajme tiež, že uvedená diskretizácia spojitého stavového modelu platí rovnako ak ide o model linearizovaný, pozri (2.17), alebo časovo premenlivý, (2.18). Načrtomme si ako by sme postupovali, ak by predmetom diskretizácie boli priamo nelineárne diferenciálne stavové rovnice  $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$ . Pre simuláciu takého modelu platí, že ak poznáme stavový vektor v určitom časovom okamihu,  $t_k$ , môže byť tento vypočítaný pre neskorší časový okamih,  $t_k + T_s$ , pomocou Taylorovho radu:

$$\mathbf{x}(t_k + T_s) = \mathbf{x}(t_k) + \frac{d\mathbf{x}(t_k)}{dt} T_s + \frac{d^2\mathbf{x}(t_k)}{dt^2} \frac{T_s^2}{2!} + \dots,$$

kde na výpočet prvej derivácie použijeme stavový model systému,  $\dot{\mathbf{x}}(t_k) = \mathbf{f}(\cdot, \cdot, t_k)$ :

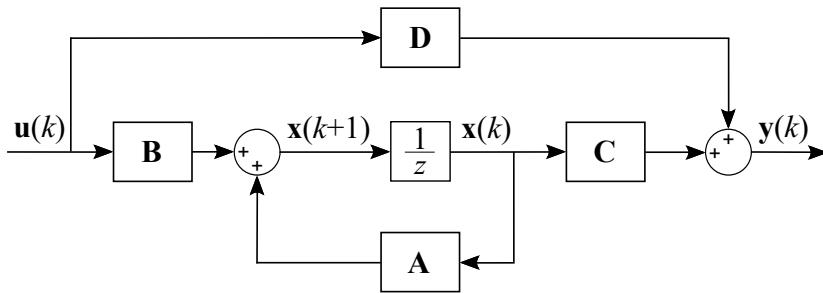
$$\mathbf{x}(t_k + T_s) = \mathbf{x}(t_k) + \mathbf{f}(t_k) T_s + \frac{d\mathbf{f}(t_k)}{dt} \frac{T_s^2}{2!} + \dots. \quad (2.33)$$

Poznáme rôzne metódy numerickej integrácie, ktoré sa líšia v numerických approximáciách derivácií funkcie  $\mathbf{f}$ . Je zrejmé, že v prípade najjednoduchšej – Eulerovej – approximácie prvého rádu a formálnej úprave dostávame:

$$\frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{T_s} = \mathbf{f}(k). \quad (2.34)$$

Ako vidíme, ak v diferenčnej rovnici (2.34) nahradíme nelineárnu funkciu  $\mathbf{f}(k)$  lineárnu, resp. linearizovanou dynamikou, získame identickú diskrétnu approximáciu ako v (2.31). To isté platí pre approximácie vyšších rádov.

Diskrétna lineárna, resp. linearizovaná časovo invariantná stavovopriestorová reprezentácia 2.28 systému je znázornená v blokovej schéme na Obr. 2.11.

Obr. 2.11. Bloková schéma diskrétneho LTI stavového modelu<sup>40</sup>.

Pripomeňme ešte, že diskrétny lineárny stavový model (2.28) môžeme získať aj priamo z lineárnej diferenčnej rovnice  $n$ -tého rádu v nasledujúcom tvare:

$$\begin{aligned} a_n y(k-n) + a_{n-1} y(k-n+1) + \cdots + a_1 y(k-1) + y(k) = \\ = b_m u(k-m) + \cdots + b_1 u(k-1) + b_0 u(k), \quad n \geq m, \end{aligned}$$

a to tradične napríklad v jeho riaditeľnom alebo pozorovateľnom kanonickom tvare so štruktúrou príslušných matíc identickou ako v (2.20), resp. (2.21).



Stavová matica prechodu **A** (dalej len matica dynamiky diskrétneho stavového modelu) je rozhodujúca pre nájdenie všeobecného riešenia lineárnych stavových rovníc v tvare (2.26a). Na jej získanie je kľúčový výpočet maticovej exponenciálnej funkcie  $e^{\mathbf{A}cT_s}$ , ktorý vykonávame pomocou:

- rozvoja do mocninového radu;
- späťnej Laplaceovej transformácie (symbolicky);
- vlastných čísel a vlastných vektorov;
- Padého approximácie;
- spektrálneho rozkladu;
- príkazu `expm()` v MATLABe;
- ... pozri [111].

Efektívny spôsob exaktnej diskretizácie spojitých modelov dynamických systémov v prostredí MATLAB je použitie funkcie `c2d()`.

<sup>40</sup>  $\frac{1}{z} = z^{-1}$  je tzv. operátor spätného posunutia a predstavuje diskrétnu analógiu integrácie. V diskrétej  $z$ -oblasti (pri 0 p.p.) platí, že:  $z^{-1}f(k) = f(k-1)$ ; v našom prípade  $z^{-1}x(k+1) = x(k)$ .

Ilustrujme si teraz správanie sa diskretizovaného modelu opäť na príklade systému invertovaného kyvadla.

**Príklad 2.6.** Nájdite diskrétnu approximáciu spojitého stavového modelu (2.23) získaného linearizáciou dynamiky invertovaného kyvadla v jeho stabilnej rovnovážnej polohe a vykonajte jeho numerickú simuláciu.

Túto úlohu môžeme opäť efektívne vykonať v prostredí MATLAB. Vytvorime si skript *simdiskkyvadlo.m*, a začnime načítaním matíc spojitého lineárneho stavového modelu, platného v okolí pracovného bodu  $(\bar{x}^T, \bar{u}) = (0 \ 0 \ \pi \ 0 \ 0)$  a ďalej vytvorením objektu stavového modelu:

Zdrojový kód 2.5. *simdiskkyvadlo.m*

```
clear; clc; %close all
load linearnySM % Nacitanie matic stav. modelu
sys=ss(Ac,Bc,Cc,Dc); % Vytvor. objektu stav. modelu
```

Pred jeho diskretizáciou si najprv definujeme vzorkovaciu períodu  $T_s$  – uvažujme veľkosť 0.01 s, resp. 10 ms. Samotnú diskretizáciu vykonáme efektívne pomocou príkazu *c2d()* (z angl. continuous-to-discrete), ktorého argumentmi sú objekt spojitého stavového modelu, vzorkovacia períoda, a metóda diskretizácia, kde uvažujeme, že hodnota vstupného signálu je počas trvania vzorky držaná na konštantnej úrovni – t.j. tvarovač nultého rádu.

```
Ts=0.01; % Vzorkovacia perioda [s]
sysd=c2d(sys,Ts,'zoh'); % Diskretizacia stav. modelu
```

Výsledkom je ZOH diskretizácia systému *sys()* podľa (2.29) na úrovni strojovej presnosti. Jednotlivé matice diskrétneho stavového modelu získame ich vyčítaním zo získaného objektu *sysd()* štýlom  $A = sysd.A$ ,  $B = \dots$  atď.; kompaktnejším je ale použitie príkazu *ssdata()* v tvare:

```
[A,B,C,D]=ssdata(sysd); % Vycit. matic disk. stav. modelu
```

Po dosadení získaných matíc do (2.17) dostávame diskrétnu approximáciu linearizovaného modelu (2.23) v tvare<sup>41</sup>

$$\Delta \mathbf{x}(k+1) = \underbrace{\begin{bmatrix} 1.0000 & 0.0100 & -0.0001 & -0.0000 \\ 0 & 1.0000 & -0.0195 & -0.0003 \\ 0 & 0 & 0.9971 & 0.0100 \\ 0 & 0 & -0.5863 & 0.9911 \end{bmatrix}}_{\mathbf{A}^{\downarrow}} \mathbf{x}(k) + \underbrace{\begin{bmatrix} 0.0001 \\ 0.0250 \\ 0.0006 \\ 0.1245 \end{bmatrix}}_{\mathbf{B}^{\downarrow}} \Delta u(k), \quad (2.35a)$$

<sup>41</sup> Uvedený formát zobrazovania numerického výstupu so 4 číslicami za desatinou čiarkou je pre MATLAB základný. Zadaním príkazu *format long* v pracovnom okne MATLABu zobrazíme až 15 číslic za desatinou čiarkou. Krátky formát opäť dostaneme príkazom *format short*.

$$\Delta \mathbf{y}(k) = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{C}} \Delta \mathbf{x}(k) + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\mathbf{D}} \Delta u(k). \quad (2.35b)$$

Všimnime si, že štruktúra matice dynamiky a matice vstupu spojitého stavového modelu (2.23) sa po diskretizácii zmenila v zmysle mocninového rozvoja exponenciálnej maticovej funkcie  $e^{\mathbf{A}_c T_s}$  (2.30) vystupujúcej vo vzťahoch (2.29a)–(2.29b). Poznamejme tiež, že matice  $\mathbf{A}$  by sme podľa (2.29a) dostali v identickom tvare pomocou príkazu  $\mathbf{A}=\text{expm}(\mathbf{A}_c \star T_s)$ , zatiaľ čo na výpočet matice  $\mathbf{B}$  by sme nemohli použiť vzťah  $\mathbf{B} = \mathbf{A}_c^{-1} (\mathbf{A} - \mathbf{I}) \mathbf{B}_c$ , keďže matice  $\mathbf{A}_c$  je singulárna ( $\det(\mathbf{A}_c) = 0$ ), a teda museli integrál rozvoja v (2.29b) vypočítať iteráčne.

Ked už máme odvodený potrebný diskrétny stavový model, môžeme dalej pristúpiť k jeho numerickej simulácii. Uvažujme pre zmenu jeho vynúteného odozvu, konkrétnie odozvu na diskrétny impulz predstavujúci pôsobenie sily o veľkosti 50 N v trvaní jednej vzorky. Začnime tradične definovaním simulačného času a počiatočného stavu systému s kyvadlom v pokoji, t. j. jeho dolnej, stabilnej rovnovážnej polohe danej, ktorý je daný odchýlkou  $\Delta \mathbf{x}_0 = [0 \ 0 \ 0]^T$  od pracovného bodu.

<code>T=10;</code>	% Konecny cas simulacie
<code>cas=0:Ts:T;</code>	% Trvanie a krok simulacie
<code>N=T/Ts;</code>	% Pocet krokov simulacie
<code>dx0=[0;0;0];</code>	% Pociatocny stav systemu

Pokračujme predbežným vytvorením, tzv. prealokáciou potrebných štruktúr pre uloženie simulovaných stavov a výstupov systému a zápisom počiatočného stavu a vstupného impulzu do prvých stĺpcov prealokovaných matíc na ukladanie stavov a vstupov v čase.

<code>[nx,nu]=size(B);</code>	% Pocet stavov a vstupov
<code>ny=size(C, 1);</code>	% Pocet vystupov
<code>dx=zeros(nx,N); % Prealok. matice pre ukladanie stavov</code>	
<code>du=zeros(nu,N); % Prealokacia vektoru vstupov</code>	
<code>dy=zeros(ny,N); % Prealok. matice pre ukladanie vystupov</code>	
<code>dx (:,1)=dx0; % Zapis pociatocneho stavu do matice dx</code>	
<code>du (:,1)=50; % Definov. impulzu na zaciatku simulacie</code>	

Všimnime si tiež, že kód sa snažíme vždy v čo najväčšej miere parametrizovať, t. j. rôzne pomocné premenné ( $N$ ,  $n_x$ , ...) si vycítavame z nami zadaných základných parametrov systému a simulácie.

Samotnú simuláciu diskrétneho lineárneho stavového modelu (2.35) pre lepšiu názornosť vykonáme jeho implementáciou v slučke – cykle, kde využijeme fakt, že diferenčná rovnica dynamiky (2.35a) je v podstate tzv. jednokrokový *prediktor*.

```

for k=1:N
    dx (:, k+1)=A*dx (:, k)+B*du (:, k) ;
    dy (:, k)=C*dx (:, k)+D*du (:, k) ;
end

```

Všimnime si, že hodnoty simulovaných stavov sa v danom kroku  $k$  zapíšu do  $(k+1)$ -ho stĺpca predvytvorenej matice  $\mathbf{dx}$ , odkiaľ sa v nasledujúcom kroku (t. j. iterácií cyklu) opäť vycítajú a použijú. Keďže pracujeme s už linearizovaným modelom v okolí stabilnej rovnovážnej polohy kyvadla, získané priebehy stavov a výstupov si ešte transformujeme z odchýlkového tvaru späť do pôvodných súradníč:

```

u=du; x=dx; % Transf. súradnic z du/dx na u/x
x (3, :)=x (3, :) +pi; % ...
y=C*x; % Transf. y pomocou matice výstupu
save sysd % Ulož. diskretného stav. modelu

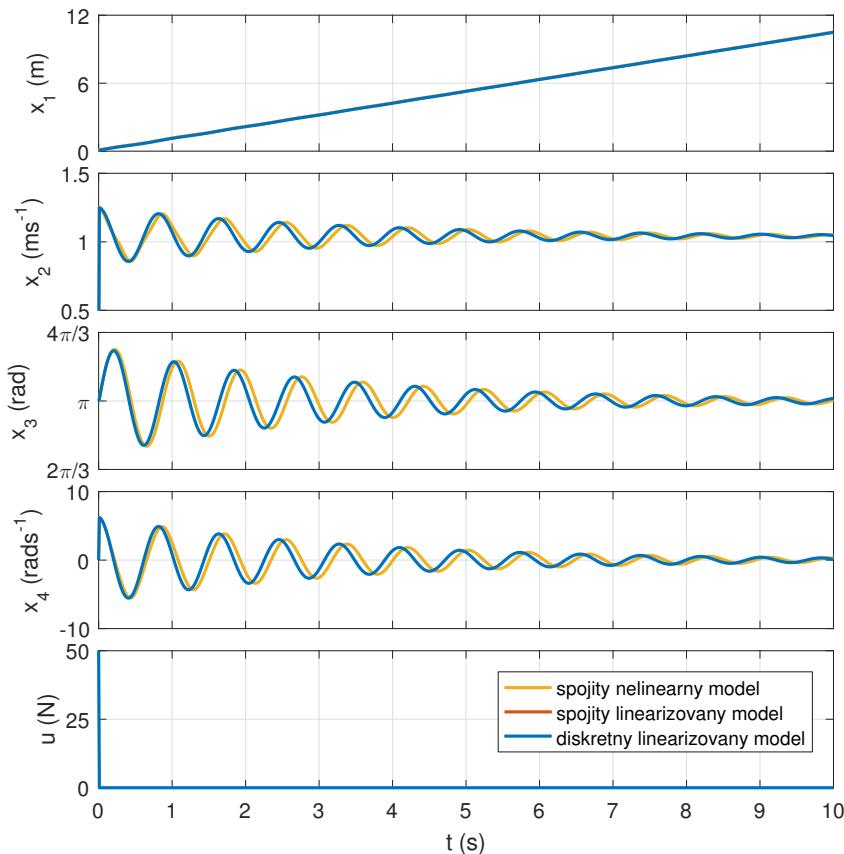
```

Časové priebehy stavov a vstupu (príp. výstupu) si opäť znázorníme analogicky ako v Zdrojovom kóde 2.1 s tým, že dané premenné sú v tomto prípade uložené v riadkoch a nie stĺpcach matíc  $\mathbf{x}$ , resp.  $\mathbf{y}$ .

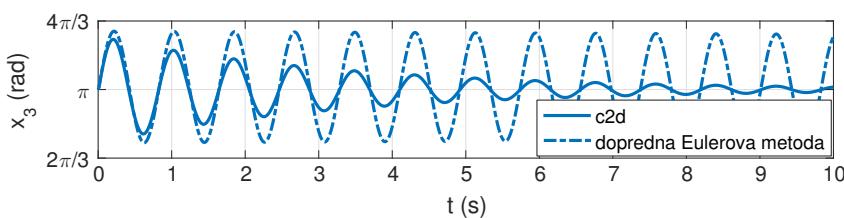
Získaná odozva diskretizovaného modelu je graficky znázornená modrou farbou na Obr. 2.12, kde je doplnená aj o odozvu spojitého nelineárneho a linearizovaného modelu, z ktorých sme vychádzali pri jeho odvodení. Všimnime si, že kyvadlo sa po aplikácii daného impulzu na začiatku simulácie vychýli až o  $40^\circ$  od jeho stabilnej rovnovážnej polohy. Táto výchylka už z pohľadu platnosti linearizovaného modelu nie je zanedbatelná, čo sa prejaví mierne nepresnou odozvou v porovnaní s odozvou plne nelineárneho modelu (2.9a) znázornenou žltou farbou. Podstatné pre nás je však, že odozva spojitého a diskretného linearizovaného modelu je totožná, t. j. na Obr. 2.12 sú červené priebehy identicky prekryté modrými.

Oprávnene nás tiež môže zaujímať aký vplyv má na dynamickú odozvu nášho systému presnosť diskretizácie, v zmysle presnosti aproximácie mocninového rozvoja maticovej exponenciálnej funkcie (2.30). Toto je ilustrované na Obr. 2.13, kde na príklade výchylky kyvadla vidíme, že ak uvažujeme aproximáciu prvého rádu, t. j. (napríklad doprednú) Eulerovu metódu, simulácia získaného diskretného modelu v tvare (2.32) dáva v porovnaní s (2.35) veľmi nepresnú dynamickú odozvu. Túto skutočnosť možno sčasti vykompenzovať, ak by sme uvažovali rýchlejšie vzorkovanie, t. j. menšiu vzorkovaciu períodu  $T_s$ . Praktické používanie Eulerovej metódy pre diskretizáciu, resp. numerickú integráciu (*ode1*) spojitých modelov sa však vo všeobecnosti neodporúča.

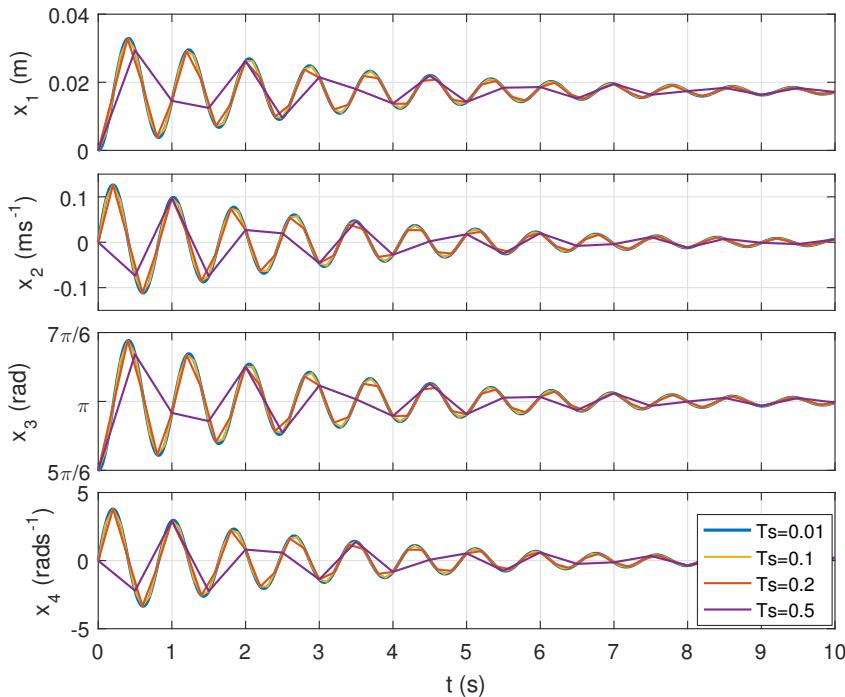
Pripomeňme ešte, že identickú odozvu na všeobecný signál  $du$  by sme jednoduchšie ako manuálnou implementáciou modelu v slučke získali pomocou príkazu `[dy, t, dx]=lsim(sysd, du, cas)`, prípadne `impulse()` (jednotkový impulz).



Obr. 2.12. Simulácia vynútenej odozvy diskrétneho linearizovaného modelu invertovaného kyvadla.



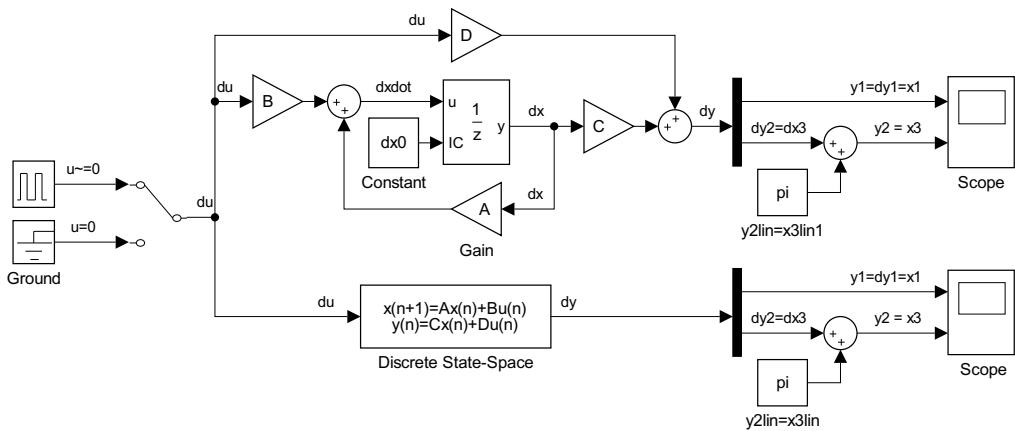
Obr. 2.13. Porovnanie vynútenej odozvy (v zmysle uhlovej výchylky kyvadla) diskrétneho modelu získaného presnou diskretizáciou pomocou príkazu `c2d()` (plná čiara) a approximáciou pomocou doprednej Eulerovej metódy (prerušovaná čiara).



Obr. 2.14. Simulácia voľnej odozvy diskrétneho linearizovaného modelu invertovaného kyvadla s uvažovaním rôznych veľkostí vzorkovacej periódy.

Pre úplnosť si ešte ilustrujme aký má na simuláciu diskrétneho modelu (2.35) systému invertovaného kyvadla vplyv rôzna veľkosť vzorkovacej periódy  $T_s$ . Na Obr. 2.14 pre lepšiu názornosť porovnávania uvažujeme voľnú odozvu modelu (2.35) s počiatočnou podmienkou  $\Delta\mathbf{x}_0 = [0 \ 0 \ -\frac{\pi}{6} \ 0]^T$ , t. j. s výchylkou kyvadla  $30^\circ$  od jeho stabilnej rovnovážnej polohy. Simulované modely boli získané presnou diskretilzáciou s využitím tvarovača nultého rádu podľa (2.29) pomocou príkazu `c2d()`.

Odozva modelu získaná pri jeho diskretilzácií a vzorkovaní s periódou 0.01 s, resp. frekvenciou 100 Hz (modrá farba) je v podstate identická s voľnou odozvou spojitého modelu znázornenou na Obr. 2.7. Na druhej strane pri uvažovaní 10 až 20-krát pomalsieho vzorkovania ( $T_s = 0.1$  s, 0.2 s, žltá a červená farba) viditeľne strácame dôležité údaje o dynamickom správaní sa systému, najmä v oblasti amplitúd kmitania. Toto je v prípade rýchleho dynamického systému ako je invertované kyvadlo neprípustné, a stavové riadenie navrhnuté na základe príslušných diskrétnych modelov by potom dostatočne nereflektovalo dynamiku reálneho systému a strácalo tak na kvalite. Diskrétny model a jeho odozva získané s uvažovaním vzorkovacej frekvencie 2 Hz už samozrejme nemajú žiadnu výpovednú hodnotu.



Obr. 2.15. Dva ekvivalentné spôsoby interpretácie a simulácie diskrétneho lineárneho modelu invertovaného kyvadla pomocou blokovej schémy v Simulinku.

Podotýkame, že lineárne úseky medzi jednotlivými vzorkami na Obr. 2.14 sú iba dôsledkom použitia príkazu `plot()`, ktorý týmto spôsobom spája diskrétne údaje získané simuláciou.

Tak, ako v prípade spojitého linearizovaného modelu, dynamické správanie sa jeho diskretizovanej formy možno tiež efektívne odsimulovať v prostredí MATLAB-Simulink. V hornej časti Obr. 2.15 je diskrétny stavový model zostavený intuitívnym vyskladaním Rov. (2.28) podľa schémy 2.11 (tu v odchýlkovom tvare), a to pomocou blokov *Gain*<sup>42</sup> a *Sum*. Simuláciu vývoja stavu podľa diskrétnej stavovej rovnice (2.28a) zabezpečíme pomocou bloku jednotkového oneskorenia, resp. jeho variantu *Unit Delay External IC* s možnosťou externého zadania počiatočnej podmienky – stavu. Na obrázku je rovnako ilustrovaný praktickejší spôsob vytvorenia diskrétneho stavového modelu – priamo na to určeným blokom *Discrete State-Space*, ktorého parametrami sú jeho matice, počiatočná hodnota stavového vektora a vzorkovacia períoda.

Ešte pred spustením samotnej simulácie je potrebné diskrétny charakter modelu zohľadniť v jej nastaveniach. V ponuke menu *Simulation→Model Configuration Parameters*, v záložke riešiča (*Solver*) nastavíme pevný krok (*Fixed-step*) a namiesto konkrétnego riešiča vyberieme možnosť *discrete (no continuous states)*. Ak je bloková schéma zostavená správne, získané výsledky budú zhodné s tými, ktoré sme získali pomocou skriptu *simdiskkyvadlo.m*.

□

<sup>42</sup>Bloky *Gain* musia byť pred použitím nastavené na vektorové násobenie, t.j. *Matrix (K\*u)*, pretože v rovnicach stavového modelu násobíme matice s vektormi.

Ako je zrejmé, vhodná voľba vzorkovacej períody, respektíve vzorkovacej frekvencie je veľmi dôležitá. Príliš nízka vzorkovacia frekvencia môže mať za dôsledok stratu veľkého množstva informácií o dynamike riadeného systému, ktorá tak znehodnotí správanie sa navrhnutého regulátora. Naopak, príliš rýchle vzorkovanie zvyšuje záťaž na procesor a môže tiež viesť na diskrétnu reprezentáciu so zlou numerickou podmienenosťou.



V prípade kmitavých systémov býva voľba vzorkovacej períody často vzťahovaná na dominantnú frekvenciu kmitania. Pri tlmených systémoch sa voľba vzorkovacej períody zvyčajne vzťahuje na časovú konštantu. Vo všeobecnosti tiež býva dobrým prvým nástrelom zvoliť vzorkovaciu períodu ako desatinu doby nábehu neriadeného systému.

Zapamätajme si, že linearizácia a diskretizácia sú dva nezávislé koncepty. Ich spoločnou črtou je, že oba predstavujú určitú formu aproximácie dynamického systému. Líšia sa v zásade v tom, že linearizácia predstavuje lineárnu aproximáciu nelineárnej dynamiky v priestore stavov a vstupov, zatiaľ čo diskretizácia predstavuje diskrétnu aproximáciu spojitej dynamiky v časovej oblasti.



Ak uvažujeme spojity nelineárny dynamický systém, pre ktorý chceme navrhnúť lokálny, diskrétny lineárny stavový regulátor, musíme pri zostavovaní potrebného diskrétneho linearizovaného stavového modelu – t.j. odvodenie pohybových rovníc, linearizáciu a diskretizáciu – dbať pri jednotlivých týchto aproximáciách reálneho systému na opatrnosť, aby výsledný model bol rozumným kompromisom medzi presnosťou a použiteľnosťou pre daný účel.

Venujme ďalej pozornosť niektorým dôležitým vlastnostiam systému opísaného diskrétnym lineárnym stavovým modelom (2.28), a to jeho stabilite, riaditeľnosti a pozorovateľnosti. Odvodenie príslušných podmienok pre ich platnosť je analogické ako v prípade spojitých systémov, kde sme si ich však z praktických dôvodov neuvádzali. Tieto vlastnosti sa však po diskretizácii modelu môžu zmeniť.

Azda najdôležitejšou z nich je *stabilita*. Kedže voľná odozva systému  $\mathbf{x}(k+1) = \mathbf{Ax}(k) + \mathbf{Bu}(k)$  je  $\mathbf{x}(k) = \mathbf{A}^k \mathbf{x}_0$ , jeho stabilita bude závisieť iba od matice  $\mathbf{A}$ . Uvažujme autonómny diskrétny lineárny systém:

$$\mathbf{x}(k+1) = \mathbf{Ax}(k). \quad (2.36)$$

Tento systém je *asymptoticky stabilný* vtedy a len vtedy, ak pre danú symetrickú kladne definitnú maticu  $\mathbf{Q}$  existuje symterická kladne definitná matica  $\mathbf{P}$ , ktorá je jediným riešením rovnice<sup>43</sup>

$$\mathbf{A}^T \mathbf{P} \mathbf{A} - \mathbf{P} = -\mathbf{Q}. \quad (2.37)$$

Dôkaz, že toto tvrdenie je postačujúce, je veľmi jednoduchý [110]. Predpokladajme, že existuje funkcia<sup>44</sup>

$$V(\mathbf{x}) = \mathbf{x}^T \mathbf{P} \mathbf{x},$$

pre ktorú, ak matica  $\mathbf{P}$  je kladne definitná, platí:

$$\begin{aligned} V(\mathbf{x}) &> 0 \quad \text{pre } \mathbf{x} \neq \mathbf{0}, \\ V(\mathbf{0}) &= 0, \\ V(\mathbf{x}) &\rightarrow \infty \quad \text{pre } \|\mathbf{x}\| \rightarrow \infty, \end{aligned}$$

a následne:

$$\begin{aligned} \Delta V(\mathbf{x}) &= \mathbf{x}^T (k+1) \mathbf{P} \mathbf{x} (k+1) - \mathbf{x}^T (k) \mathbf{P} \mathbf{x} (k) \\ &= \mathbf{x}^T (k) \left( \mathbf{A}^T \mathbf{P} \mathbf{A} - \mathbf{P} \right) \mathbf{x} (k) \\ &= -\mathbf{x}^T (k) \mathbf{Q} \mathbf{x} (k), \end{aligned}$$

z čoho vyplýva, že:

$$\Delta V(\mathbf{x}) < 0 \quad \text{pre } \mathbf{x} \neq \mathbf{0},$$

pretože matica  $\mathbf{Q}$  je kladne definitná. Týmto je dôkaz ukončený.

Alternatívnym, ale veľmi praktickým spôsobom určenia stability systému (2.36) je metóda vychádzajúca z analýzy vlastných čísel matice dynamiky systému. Označme si vlastné čísla matice  $\mathbf{A} \in \mathbb{R}^{n_x \times n_x}$  ako  $\lambda_1, \dots, \lambda_m$ ,  $m \leq n_x$ . Hovoríme, že systém (2.36) je:

- *asymptoticky stabilný* vtedy a len vtedy, ak  $|\lambda_i| < 1$ ,  $\forall i = 1, \dots, m$ ;
- *na hranici stability*, ak  $|\lambda_i| \leq 1$ ,  $\forall i = 1, \dots, m$ , a vlastné čísla s vektorom 1 majú rovnakú algebraickú a geometrickú násobnosť<sup>45</sup>;
- *nestabilný* ak  $\exists i$  také, že  $|\lambda_i| > 1$ .

<sup>43</sup>Známa ako diskrétna Ljapunovova rovnica; je špeciálnym prípadom tzv. Steinovej rovnice.

<sup>44</sup>V tomto kontexte nazývaná diskrétna (kvadratická) Ljapunovova funkcia.

<sup>45</sup>Algebraická násobnosť  $\lambda_i$  je daná príslušným počtom rovnakých koreňov  $\lambda_i$  charakteristickeho polynómu matice  $\mathbf{A}$ , t. j.  $\det(\lambda \mathbf{I} - \mathbf{A})$ . Geometrická násobnosť  $\lambda_i$  je daná počtom lineárne nezávislých vlastných vektorov  $v_i$ , pre ktoré platí  $\mathbf{A}v_i = \lambda_i v_i$ . Geometrická násobnosť vlastných čísel nikdy nemôže byť väčšia ako ich algebraická násobnosť.

Dôkaz možno nájsť napríklad v [14]. Platí teda, že systém (2.36) je stabilný vtedy a len vtedy, ak všetky vlastné čísla matice  $\mathbf{A}$  sú menšie ako 1, t. j. ležia vo vnútri jednotkovej kružnice. Touto podmienkou je ekvivalentne podmienená aj existencia kladne definitnej matice  $\mathbf{P}$  ako riešenia diskrétnej Ljapunovovej rovnice (2.37). Je teda zrejmé, že vlastnosť stability diskrétneho lineárneho systému závisí iba od veľkosti<sup>46</sup> vlastných čísel matice dynamiky. Spomeňme si, že tá istá podmienka stability platí aj pre póly  $z_i$  diskrétneho prenosu  $G(z)$ , ktoré sú totožné s vlastnými číslami  $\lambda_i$  matice dynamiky  $\mathbf{A}$  stavového modelu daného systému<sup>47</sup>.

Poznamenajme, že v prípade spojitých systémov sa pri analýze stability postupuje analogicky. Podmienka stability tu hovorí, že vlastné čísla matice  $\mathbf{A}_c$  (alebo ekvivalentne póly prenosu  $G(s)$ ) sa musia nachádzať v ľavej polrovine komplexnej  $s$ -roviny<sup>48</sup>, resp. ich reálna časť musí byť záporná, t. j.  $\operatorname{Re}(\lambda_i) < 0$ .

Pojmu stabilita spojitých systémov sa hovorí taktiež analytická stabilita. V kontexte numerickej simulácie spojitých systémov sa zvykne hovoriť o tzv. numerickej stabiliti, a to najmä v spojitosi s metódami numerickej integrácie, ktorými approximujeme rozvoj (2.33) dynamiky daného systému v čase [28]. Je preto zrejmé, že okrem rádu jeho approximácie, resp. rádu integračnej metódy, na numerickú stabilitu zásadne vplýva veľkosť vzorkovacej periody  $T_s$ . Spomeňme si, že aj v rámci Pr. 2.3 a Pr. 2.5 sme na numerickú simuláciu spojitého nelineárneho a linearizovaného stavového modelu použili numerickú integráciu, konkrétnie Runge-Kutta metódu štvrtého rádu (riešič *ode45*) s dĺžkou kroku 0.01 s<sup>49</sup>. Ako ilustroval Pr. 2.6, pri testovaní integračných metód nižších rádov, prípadne väčších dĺžok kroku, s najväčšou pravdepodobnosťou narazíme na numerickú nestabilitu simulácie.

Uvedeným spôsobom analyzujeme stabilitu diskrétneho systému v otvorennej slučke. Pokiaľ slučku uzatvoríme napríklad stavovým regulátorom so spätnoväzbovým riadiacim zákonom  $\mathbf{u}(k) = -\mathbf{K}\mathbf{x}(k)$ , kde  $\mathbf{K}$  je zosilnenie regulátora a tento dosadíme do stavovej rovnice, dostávame:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) - \mathbf{B}\mathbf{K}\mathbf{x}(k) = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}(k) = \Psi\mathbf{x}(k).$$

Stabilitu uzavretého regulačného obvodu analyzujeme analogicky, ale na základe vlastných čísel tzv. stavovej matice, resp. matice dynamiky uzavretého regulačného obvodu  $\Psi = \mathbf{A} - \mathbf{B}\mathbf{K}$ .



<sup>46</sup>Vlastné čísla  $\lambda_i$  sú vo všeobecnosti komplexné čísla v tvare  $\lambda = a + bi$ , pre veľkosť ktorých platí:  $|\lambda| = \sqrt{a^2 + b^2}$ .

<sup>47</sup>Toto platí iba v prípade riaditeľného a pozorovateľného systému.

<sup>48</sup>Pripomeňme si, že presné zobrazovanie z  $s$ -roviny do  $z$ -roviny je dané vzťahom  $z = e^{sT_s}$ .

<sup>49</sup>Dĺžka kroku tu predstavuje vzorkovaciu periódou simulácie.

Ďalšie dve dôležité vlastnosti stavových modelov, ktorým je potrebné venovať pozornosť ešte pred návrhom regulátora, sú riaditeľnosť a pozorovateľnosť.

Uvažujme dynamický systém opísaný diskrétnym LTI stavovým modelom:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k), \quad (2.38a)$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k), \quad (2.38b)$$

kde  $\mathbf{A} \in \mathbb{R}^{n_x \times n_x}$ ,  $\mathbf{B} \in \mathbb{R}^{n_x \times n_u}$ ,  $\mathbf{C} \in \mathbb{R}^{n_y \times n_x}$  a maticu  $\mathbf{D}$  neuvažujeme.

Úplná riaditeľnosť (zjednodušene riaditeľnosť): Hovoríme, že stavová rovnica (2.38a) (alebo páár  $(\mathbf{A}, \mathbf{B})$ ) je úplne riaditeľná, ak pre ľubovoľný počiatočný stav  $\mathbf{x}(0)$  a ľubovoľný cieľový stav  $\mathbf{x}(N)$  existuje taká postupnosť vstupov  $\mathbf{u}(k)$ ,  $k = 0, 1, 2, \dots, N$ , ktorá priviedie  $\mathbf{x}(0)$  do  $\mathbf{x}(N)$  v konečnom čase  $N$ . V opačnom prípade je stavová rovnica (2.38a) neriaditeľná. Na posúdenie riaditeľnosti systému (2.38) môžeme efektívne využiť podmienku, ktorá hovorí, že daný systém je *riaditeľný* vtedy a len vtedy, ak tzv. *matica riaditeľnosti*<sup>50</sup>

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \dots & \mathbf{A}^{n_x-1}\mathbf{B} \end{bmatrix}, \quad (2.39)$$

rozmeru  $n_x \times n_x \cdot n_u$ , má plnú riadkovú hodnosť (počet lineárne nezávislých riadkov), t. j.  $\text{rank}(\mathcal{C}) = n_x$ .

Úplná pozorovateľnosť (zjednodušene pozorovateľnosť): Hovoríme, že stavový model (2.38) (alebo páár  $(\mathbf{A}, \mathbf{C})$ ) je úplne pozorovateľný, ak ľubovoľný počiatočný stav  $\mathbf{x}(0)$  môže byť určený na základe znalosti výstupu  $\mathbf{y}(k)$  a postupnosti vstupov  $\mathbf{u}(k)$ ,  $k = 0, 1, 2, \dots, N$ , v konečnom čase  $N$ . V opačnom prípade je stavový model (2.38) nepozorovateľný. Na posúdenie pozorovateľnosti systému (2.38) môžeme podobne efektívne využiť podmienku, ktorá hovorí, že daný systém je pozorovateľný vtedy a len vtedy, ak tzv. *matica pozorovateľnosti*<sup>51</sup>

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n_x-1} \end{bmatrix}, \quad (2.40)$$

rozmeru  $n_y \cdot n_x \times n_x$ , má plnú stĺpcovú hodnosť (počet lineárne nezávislých stĺpcov), t. j.  $\text{rank}(\mathcal{O}) = n_x$ .

Poznamenajme tiež, že koncepty riaditeľnosti a pozorovateľnosti sú z matematického pohľadu *duálne*, pričom platí, že páár  $(\mathbf{A}, \mathbf{B})$  je riaditeľný vtedy a len vtedy, ak páár  $(\mathbf{A}^T, \mathbf{C}^T)$  je pozorovateľný. Uvedené podmienky možno taktiež analogicky uplatniť pre spojité ako aj časovo premenlivé lineárne systémy.

<sup>50</sup>angl. controllability matrix

<sup>51</sup>angl. observability matrix

Ilustrujme si analýzu uvedených vlastností diskrétneho lineárneho stavového modelu na príklade systému invertovaného kyvadla.

**Príklad 2.7.** Na základe diskrétneho lineárneho stavového modelu vyšetrite stabilitu, riaditeľnosť a pozorovateľnosť neriadeného systému invertovaného kyvadla na vozíku v konfigurácii s kyvadlom v jeho vztýčenej (invertovanej) nestabilnej rovnovážnej polohe.

Aby sme overili evidentnú nestabilitu invertovaného kyvadla v otvorennej slučke, je potrebné jeho nelineárny stavový model (2.9) linearizovať v jeho hornej, nestabilnej rovnovážnej polohe a potom zdiskretizovať. Postupujeme pritom analogicky ako v Pr. 2.3 a Pr. 2.5, pričom uvažujeme pracovný (rovnovážny) bod  $(\bar{\mathbf{x}}^T, \bar{u}) = (0 \ 0 \ 0 \ 0)$  a vzorkovaciu períodu  $T_s = 0.01$  s. Takto dostávame nasledujúci diskrétny LTI stavový model:

$$\Delta \mathbf{x}(k+1) = \underbrace{\begin{bmatrix} 1.0000 & 0.0100 & -0.0001 & -0.0000 \\ 0 & 1.0000 & -0.0196 & 0.0001 \\ 0 & 0 & 1.0029 & 0.0100 \\ 0 & 0 & 0.5874 & 0.9970 \end{bmatrix}}_{\mathbf{A}^\dagger} \mathbf{x}(k) + \underbrace{\begin{bmatrix} 0.0001 \\ 0.0250 \\ -0.0006 \\ -0.1247 \end{bmatrix}}_{\mathbf{B}^\dagger} \Delta u(k),$$

$$\Delta \mathbf{y}(k) = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{C}} \Delta \mathbf{x}(k) + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\mathbf{D}} \Delta u(k),$$

ktorý si pomocou príkazu `save sysd` uložíme do súboru *sysd.mat*.

Takto získaný model môžeme ďalej analyzovať z pohľadu vyšetrovaných vlastností. Začnime s overením jeho stability. Vytvorime si skript *vlastnostiDSM.m*, a načítajme si objekt a ďalej matice odvodeného diskrétneho stavového modelu:

Zdrojový kód 2.6. *vlastnostiDSM.m*

```
clear; clc; close all
load sysd % Nacitanie diskr. stav. modelu
[A,B,C,D]=ssdata(sysd); % Vycit. matic disk. stav. modelu
```

Na posúdenie jeho stability využijeme vlastné čísla  $\lambda_i$  matice dynamiky  $\mathbf{A}$ , ktoré sú riešením jej charakteristickej rovnice  $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$ . Tento problém možno v MATLABe jednoducho vyriešiť príkazom:

```
lambda=eig(A) % Vlastne cisla matice dynamiky A
```

ktorého výsledkom sú vlastné čísla  $\lambda_1 = 1$ ,  $\lambda_2 = 1$ ,  $\lambda_3 = 1.0766$  a  $\lambda_4 = 0.9233$ . V tomto prípade sú všetky  $\lambda_i$  reálne čísla a platí, že  $|\lambda_i| = \lambda_i$ . Vo všeobecnosti však  $\lambda_i$  často bývajú komplexné a vtedy na určenie ich veľkosti použijeme príkaz

```
abs(lambda) % Velkosť vlastných čísel
```

Kedže platí, že  $|\lambda_3| > 1$ , náš diskrétny systém získaný linearizáciou v pracovnom bode  $(\bar{x}, \bar{u}) = (0 \ 0 \ 0 \ 0)$  je nestabilný. Stabilitu systému môžeme overiť aj príkazom

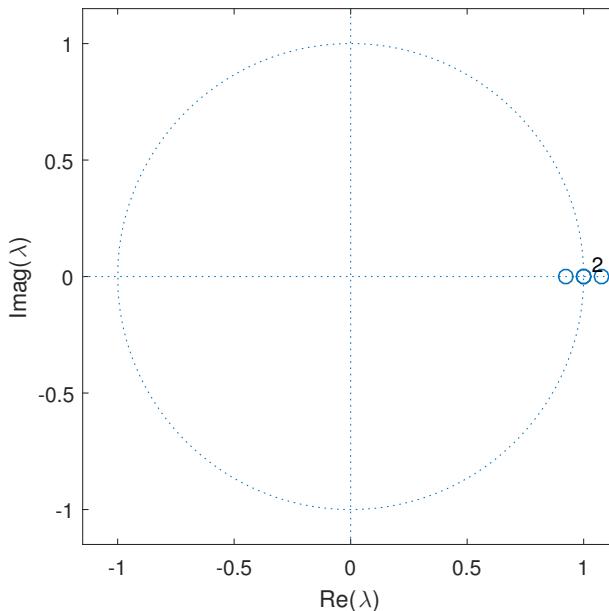
```
isstable(sysd) % Test stability systemu
```

ktorý vráti logickú 1 ak je daný systém stabilný a logickú 0 ak je systém na hranici stability alebo nestabilný (náš prípad).

Podmienku stability systému si môžeme znázorniť aj graficky – podľa umiestnenia získaných vlastných čísel  $\lambda_i$  v komplexnej rovine, napríklad pomocou príkazu `zplane()`:

```
figure('color', 'w') % Otvor. noveho grafickeho okna
zplane(lambda) % Znázor. vlast. čisel v z-rovine
xlabel('Re(\lambda)'); % Oznacenie osi x
ylabel('Imag(\lambda)'); % Oznacenie osi y
```

ktorých výsledok je znázornený na Obr. 2.16.



Obr. 2.16. Analýza stability pomocou znázornenia vlastných čísel  $\lambda_i$  v komplexnej  $z$ -rovine.

Ako je zrejmé, vlastné číslo  $\lambda_4$  leží vo vnútri znázornenej jednotkovej kružnice,  $\lambda_1$  a  $\lambda_2$  priamo na nej (symbol 2 v grafe označuje násobnosť vlastného čísla), zatiaľ čo  $\lambda_3$  leží mimo nej a jej vnútra – systém je teda v uvažovanej polohe nestabilný.

Ďalej si vyšetríme riaditeľnosť a pozorovateľnosť invertovaného kyvadla v jeho hornej, nestabilnej rovnovážnej polohe. Pokračujme v skripte *vlastnostiDSM.m*.

Ako už vieme, ak má byť nás systém úplne riaditeľný, jeho matica riaditeľnosti  $\mathcal{C}$  (2.39) musí mať hodnosť  $n_x$ . Maticu  $\mathcal{C}$  by sme mohli naprogramovať v cykle; my však na jej zostavenie a určenie hodnosti použijeme preddefinované príkazy:

`Co = ctrb(sysd)` % Matica riaditeľnosti  
`length(A)-rank(Bo)` % Počet neriaditeľnych stavov

a dostávame:

$$\mathcal{C} = \begin{bmatrix} 0.0001 & 0.0004 & 0.0006 & 0.0009 \\ 0.0250 & 0.0250 & 0.0250 & 0.0251 \\ -0.0006 & -0.0019 & -0.0031 & -0.0044 \\ -0.1247 & -0.1247 & -0.1255 & -0.1269 \end{bmatrix}, \quad \text{rank}(\mathcal{C}) = 4,$$

na základe čoho počet neriaditeľných stavov je  $n_x - \text{rank}(\mathcal{C}) = 4 - 4 = 0$ , a teda daný systém je úplne riaditeľný.

Odborným spôsobom numericky vyšetríme pozorovateľnosť systému, kde hlavnú úlohu hrá matica pozorovateľnosti  $\mathcal{O}$  (2.40):

`Ob = obsv(sysd)` % Matica pozorovateľnosti  
`length(A)-rank(Ob)` % Počet nepozorovateľných stavov

Dostávame:

$$\mathcal{O} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0.01 & -0.0001 & 0.00 \\ 0 & 0 & 1.0029 & 0.01 \\ 1 & 0.02 & -0.0004 & 0.00 \\ 0 & 0 & 1.0117 & 0.02 \\ 1 & 0.03 & -0.0009 & 0.00 \\ 0 & 0 & 1.0264 & 0.03 \end{bmatrix}, \quad \text{rank}(\mathcal{O}) = 4,$$

na základe čoho počet nepozorovateľných stavov je  $n_x - \text{rank}(\mathcal{O}) = 4 - 4 = 0$ , a teda daný systém je úplne pozorovateľný.

Poznamejme, že týmto spôsobom môžeme otestovať rovnaké vlastnosti spojitého linearizovaného modelu (2.23) (t. j. pred diskretizáciou), pričom sa dopracujeme k rovnakým záverom.

□

Na základe týchto zistení môže byť nás diskrétny linearizovaný model použitý pre návrh pozorovateľa stavu a stavového regulátora (napríklad LQ, MPC a pod.) pre stabilizáciu systému v jeho nestabilnej rovnovážnej polohe.

## 2.4 Odhad stavu a Kalmanov filter

V predchádzajúcich kapitolách sme si zopakovali ako možno na základe matematicko-fyzikálnej analýzy odvodiť vo všeobecnosti nelineárny, dynamický model vyšetrovaného systému v stavovom priestore a ukázali, ako sa dá pomocou linearizácie a diskretizácie získať jeho lineárna a časovo diskrétna approximácia, ktorá bude v okolí zvoleného pracovného bodu dostatočne dobre opisovať dynamiku daného systému. Takto odvodený, diskrétny lineárny stavový model je základným predpokladom pre návrh kvalitného lineárneho stavového riadenia.

Ako sme už naznačili v úvode tejto kapitoly, návrh spätnoväzbového stavového riadenia, ako napr. prediktívneho alebo LQ riadenia, vychádza z toho, že všetky prvky stavového vektora  $\mathbf{x}(k)$  sú prístupné meraniu – hovoríme, že sú merateľné. Toto sa však v prípade väčsiny reálnych systémov nedá prakticky zabezpečiť. Východiskom z tejto situácie je na základe meraných priebehov prvkov vektora výstupných veličín  $\mathbf{y}(k)$  pozorovať, resp. odhadnúť stav  $\mathbf{x}(k)$ . Ak je daný systém pozorovateľný (pozri Kap. 2.3), možno jeho stav na základe meraných výstupov úplne „zrekonštruovať“ pomocou tzv. pozorovateľa stavu<sup>52</sup>. V ďalšom si stručne uvedieme deterministický odhad stavu a odhad stavu stochastických systémov.

Teória *deterministického odhadu stavu* je dobre rozpracovaná. V prípade deterministického odhadu stavu uvažujeme systém bez významnejšieho šumu a bez významnejšej chyby merania. Uvažujme, že je opísaný nasledujúcim diskrétnym lineárnym časovo invariantným (LTI) stavovým modelom:

$$\mathbf{x}(k+1) = \mathbf{Ax}(k) + \mathbf{Bu}(k), \quad \mathbf{x}(0) = \mathbf{x}_0, \quad (2.42a)$$

$$\mathbf{y}(k) = \mathbf{Cx}(k) + \mathbf{Du}(k). \quad (2.42b)$$

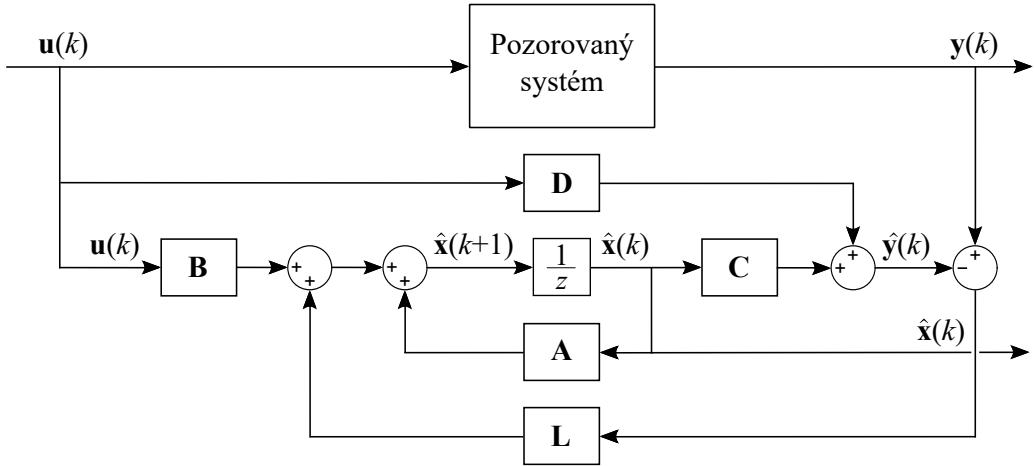
Z tohto modelu pozorovaného systému je zvyčajne odvodený aj model pozorovateľa, tak ako to v roku 1963 navrhol David Luenberger (nar. 1937). Stavová rovnica modelu pozorovateľa je v tomto prípade rozšírená o proporcionalnu spätnú väzbu

$$\mathbf{u}_L = \mathbf{L}(\mathbf{y}(k) - \hat{\mathbf{y}}(k)),$$

v ktorej je výstup pozorovateľa odčítaný od výstupu systému a ďalej prenásobený maticovým zosilnením  $\mathbf{L} \in \mathbb{R}^{n_x \times n_y}$ . Jeho úlohou je zabezpečiť čo možno najmenší rozdiel  $(\mathbf{y}(k) - \hat{\mathbf{y}}(k))$ <sup>53</sup>.

<sup>52</sup>V literatúre sa pojmom pozorovateľ, resp. pozorovač stavu (angl. state observer) často používa aj keď nejde o priame pozorovanie stavu, ale výstupu, keď by vhodnejším bolo použitie pojmu odhadovač stavu (angl. state estimator). My budeme používať v teórii riadenia zaužívaný pojem, pozorovateľ stavu, na označenie dynamického systému, ktorý poskytuje regulátoru odhad vnútorného stavu daného reálneho systému na základe jeho vstupov a výstupov.

<sup>53</sup>Všimnime si, že premenné pozorovateľa stavu sú tradične označované strieškou  $(\cdot)$  na odlišenie od premenných stavového modelu.



Obr. 2.17. Bloková schéma Luenbergerovho pozorovateľa stavu.

Výsledný model tzv. *Luenbergerovho pozorovateľa stavu* je potom daný rovnicami:

$$\hat{\mathbf{x}}(k+1) = \mathbf{A}\hat{\mathbf{x}}(k) + \mathbf{B}\mathbf{u}(k) + \mathbf{L}(\mathbf{y}(k) - \hat{\mathbf{y}}(k)), \quad \hat{\mathbf{x}}(0) = \hat{\mathbf{x}}_0, \quad (2.43a)$$

$$\hat{\mathbf{y}}(k) = \mathbf{C}\hat{\mathbf{x}}(k) + \mathbf{D}\mathbf{u}(k). \quad (2.43b)$$

Navrhnúť pozorovateľ s požadovanými vlastnosťami teda znamená vhodným spôsobom navrhnúť jeho zosilnenie  $\mathbf{L}$  tak, aby sa chyba odhadu, resp. pozorovania

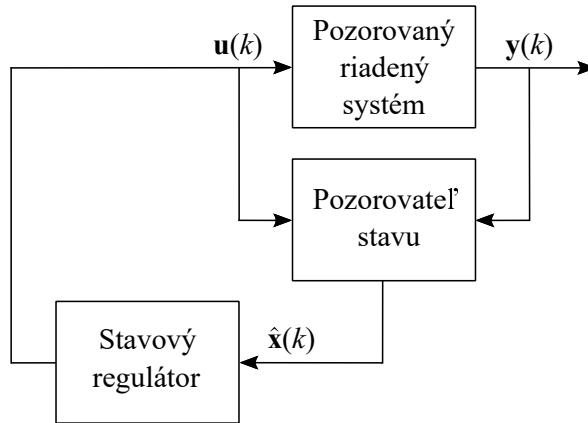
$$\mathbf{e}(k) = \mathbf{x}(k) - \hat{\mathbf{x}}(k) \quad (2.44)$$

pri počiatočnej chybe  $\mathbf{e}(0) = \mathbf{x}_0 - \hat{\mathbf{x}}_0$  blížila (konvergovala) v čase  $k \rightarrow \infty$  asymptoticky k nule. Pre dynamiku chyby odhadu s využitím (2.42)–(2.44) platí:

$$\begin{aligned} \mathbf{e}(k+1) &= \mathbf{x}(k+1) - \hat{\mathbf{x}}(k+1) \\ &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) - \mathbf{A}\hat{\mathbf{x}}(k) - \mathbf{B}\mathbf{u}(k) - \mathbf{L}(\mathbf{y}(k) - \hat{\mathbf{y}}(k)) \\ &= \mathbf{A}(\mathbf{x}(k) - \hat{\mathbf{x}}(k)) - \mathbf{L}\mathbf{C}(\mathbf{x}(k) - \hat{\mathbf{x}}(k)) \\ &= (\mathbf{A} - \mathbf{L}\mathbf{C})\mathbf{e}(k), \quad \hat{\mathbf{e}}(0) = \hat{\mathbf{e}}_0. \end{aligned} \quad (2.45)$$

Systém (2.45) bude vyslovovať formulovanéj úlohe návrhu pozorovateľa stavu, ak bude asymptoticky stabilný. Luenbergerov pozorovateľ diskrétneho systému (2.42) bude preto asymptoticky stabilný práve vtedy, keď vlastné čísla matice  $(\mathbf{A} - \mathbf{L}\mathbf{C})$  ležia vo vnútri jednotkovej kružnice, t. j.  $|\lambda(\mathbf{A} - \mathbf{L}\mathbf{C})| < 1$ .

Základná štruktúra diskrétneho Luenbergerovho pozorovateľa stavu je znázorená v blokovej schéme na Obr. 2.17.



Obr. 2.18. Stavové spätnoväzbové riadenie s deterministickým odhadom stavu.

Obr. 2.18 ilustruje, ako by sme odvodený pozorovateľ stavu využili pri návrhu spätnoväzbového stavového riadenia v prípade, že stav systému nie je plne meraťelný, t. j.  $\mathbf{y}(k) \neq \mathbf{x}(k)$ . Uvažujme, že výstupu systému pozorovateľa, deterministický odhad stavu  $\hat{\mathbf{x}}(k)$ , je vstupom optimálneho regulátora so stavovou spätnou väzbou

$$\mathbf{u}(k) = -\mathbf{K}\hat{\mathbf{x}}(k), \quad (2.46)$$

kde  $\mathbf{K}$  je maticové zosilnenie regulátora.

Poznamenanajme, že pri použití riadiaceho zákona (2.46) (s uvažovaním nulovej trvalej regulačnej odchýlky) je zaručená stabilita obvodu napriek tomu, že  $\mathbf{x}(k)$  a  $\hat{\mathbf{x}}(k)$  sa lišia. Dá sa dokázať, že ak pre úplne riaditeľný a úplne pozorovateľný LTI systém (2.42) navrhнемe stabilný pozorovateľ a stabilný spätnoväzbový regulátor, potom stavové spätnoväzbové riadenie kombinované s pozorovateľom stavu bude stabilné, o čom pojednáva tzv. veta o separácii – oddeliteľnosti problému návrhu regulátora od problému návrhu pozorovateľa. Táto hovorí, že vlastné čísla tohto systému spätnoväzbového riadenia s pozorovateľom pozostávajú z vlastných čísel matice  $\mathbf{A} - \mathbf{BK}$ , danej stavovou spätnou väzbou bez pozorovateľa, a z vlastných čísel matice  $\mathbf{A} - \mathbf{LC}$ . Príslušný systém dostaneme dosadením Rov. (2.46) a (2.44) do Rov. (2.42a), a jej rozšírením o dynamiku chyby odhadu (2.45):

$$\begin{bmatrix} \mathbf{x}(k+1) \\ \mathbf{e}(k+1) \end{bmatrix} = \begin{bmatrix} \mathbf{A} - \mathbf{BK} & \mathbf{BK} \\ \mathbf{0} & \mathbf{A} - \mathbf{LC} \end{bmatrix} \begin{bmatrix} \mathbf{x}(k) \\ \mathbf{e}(k) \end{bmatrix}, \quad \begin{bmatrix} \mathbf{x}(0) \\ \mathbf{e}(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{e}_0 \end{bmatrix},$$

$$\mathbf{y}(k) = [\mathbf{C} \quad \mathbf{0}] \begin{bmatrix} \mathbf{x}(k) \\ \mathbf{e}(k) \end{bmatrix}.$$

Dôkaz vety o separácii možno nájsť v [110, 24].

Venujme ďalej z praktických dôvodov väčšiu pozornosť odhadu stavu stochastických systémov a priblížme si koncept známeho *Kalmanovo filtra*. Kalmanov filter, navrhnutý R. Kálmánom, je v podstate špeciálnym typom Luenbergerovho pozorovateľa stavu *optimalizovaného* pre minimalizáciu vplyvu šumu merania a vo vstupe. Kalmanova filtrácia predstavuje algoritmus, ktorý využíva sériu meraní pozorovaných v čase a zahŕňajúcich stochastický šum a iné nepresnosti a produkuje odhady neznámych veličín, ktoré sú spravidla presnejšie, ako keby boli získané na základe jedného merania. Podobným spôsobom ako na uvedenú *fúzii údajov zo snímačov* sa Kalmanov filter<sup>54</sup> používa na výpočet štatisticky *optimálneho odhadu* stavu systému na základe zašumených meraní (výstupov systému).

Vysvetlime si ďalej stručne základný princíp odhadu stavu pomocou Kalmanovho filtra. Z praktického pohľadu ďalej uvažujme nominálny, diskrétny Kalmanov filter, čo je v podstate pôvodná formulácia, v ktorej bol v roku 1960 navrhnutý pre svoju číslicovú implementáciu. Tento predpokladá, že skutočný stav systému v čase  $k + 1$  sa z času  $k$  vyvíja podľa vzťahu<sup>55</sup>

$$\mathbf{x}_{k+1} = \mathbf{Ax}_k + \mathbf{Bu}_k + \mathbf{w}_k, \quad (2.47a)$$

$$\mathbf{y}_k = \mathbf{Cx}_k + \mathbf{v}_k, \quad (2.47b)$$

pričom druhá rovnica opisuje výstup – pozorovanie  $\mathbf{y}_k$  stavu systému  $\mathbf{x}_k$  v čase  $k$ . Je zrejmé, že ide o diskrétny lineárny časovo invariantný stavový model dynamického systému, kde stavová rovnica (rovnica dynamiky) (2.47a) je rozšírená o tzv. procesný šum<sup>56</sup>  $\mathbf{w}_k \in \mathbb{R}^{nx}$  a rovnica merania, resp. výstupu (2.47b) je doplnená o tzv. šum merania<sup>57</sup>  $\mathbf{v}_k \in \mathbb{R}^{ny}$ . Uvažujeme, že procesný šum a šum merania sú *nekorelované*<sup>58</sup> Gaussovské biele šumy s viacozmerným normálnym rozdelením

$$\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}), \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}),$$

t. j. s nulovou strednou hodnotou a so spoločným rozptylom – kovarianciou danou tzv. kovariančou maticou procesného šumu  $\mathbf{Q}$ , resp. kovariančou maticou šumu merania  $\mathbf{R}$ <sup>59</sup>. Šum merania môžeme odvodiť z presnosti jednotlivých meracích členov a zo znalostí charakteru procesu. Zložitejšie je opísať šum procesu často

<sup>54</sup>Kedže aj pri odhade stavu systému sa Kalmanov filter v podstate snaží „odfiltrovať“ šum od zašumených signálov, označuje sa všeobecne zaužívaným pojmom (stredne kvadratický) *filter*.

<sup>55</sup>Kedže pri optimálnom odhade stavu budeme pracovať s predikovanými hodnotami veličín, budeme ďalej pre ich značenie v diskrétnom čase používať dolný index. Túto formalizáciu budeme používať aj v kontexte prediktívneho riadenia. Uvažujeme tiež štandardný predpoklad, že  $\mathbf{D} = \mathbf{0}$ .

<sup>56</sup>angl. process noise

<sup>57</sup>angl. measurement noise

<sup>58</sup>Dve náhodné premenné,  $X$  a  $Y$ , sú nekorelované, ak ich kovariancia  $E[XY] - E[X]E[Y] = 0$ .

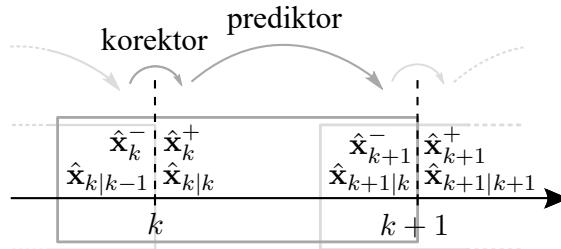
<sup>59</sup>Platí teda, že  $E[\mathbf{w}_k] = \mathbf{0}$  a  $E[\mathbf{v}_k] = \mathbf{0}$ , a  $E[\mathbf{w}_k \mathbf{w}_j^T] = \mathbf{Q} \delta_{kj}$  a  $E[\mathbf{v}_k \mathbf{v}_j^T] = \mathbf{R} \delta_{kj}$ , pričom  $E[\mathbf{w}_k \mathbf{v}_j^T] = \mathbf{0}$ .  $E$  predstavuje operátor strednej hodnoty náhodnej veličiny a funkcia  $\delta_{kj}$  je tzv. Kroneckerova delta, pre ktorú platí, že  $\delta_{kj} = 0$ , ak  $k \neq j$  a  $\delta_{kj} = 1$ , ak  $k = j$ .

predstavujúci efekt nemodelovanej dynamiky, o ktorom v mnohých prípadoch nemáme žiadne apriórne znalosti. Problém rozlíšenia šumu merania a procesného šumu nie je triviálny a v rámci teórie riadenia je predmetom robustného riadenia.

Stav samotného Kalmanovho filtra je daný nasledujúcimi dvoma premennými:

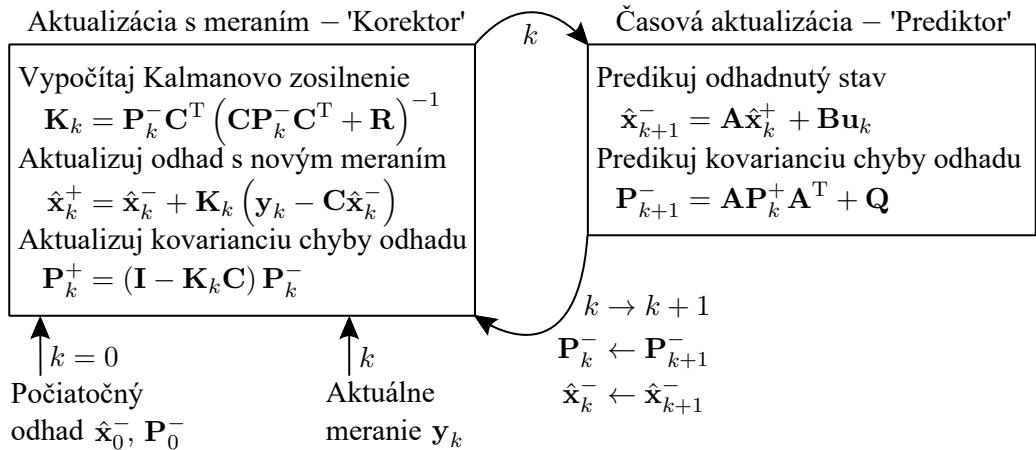
- $\hat{\mathbf{x}}_{k|k}$  – tzv. aposteriórny odhad stavu v čase  $k$  na základe pozorovaní do času  $k$  vrátane,  $\hat{\mathbf{x}}_{k|k} = E[\mathbf{x}_k]$ ;
- $\mathbf{P}_{k|k}$  – tzv. aposteriórna kovariančná matica chyby odhadu predstavujúca mieru neurčitosti odhadu stavu,  $\mathbf{P}_{k|k} = E[(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})^T]$ .

Algoritmus Kalmanovho filtra pracuje v dvoch koncepcne základných krokoch – korekcia a predikcia. Tzv. korektor upresňuje (koriguje) tzv. apriórny odhad stavu z predchádzajúceho kroku,  $\hat{\mathbf{x}}_{k|k-1}$ , jeho kombináciou s aktuálnym pozorovaním,  $\mathbf{y}_k$ , pozri (2.48b). Tento „vylepšený“ – aktualizovaný odhad,  $\hat{\mathbf{x}}_{k|k}$ , je označovaný ako aposteriórny odhad stavu. Potom tzv. prediktor na základe aposteriórneho odhadu stavu v čase  $k$  vypočíta už spomínaný apriórny – „predikovaný“ odhad stavu v nasledujúcom časovom kroku  $k+1$ ,  $\hat{\mathbf{x}}_{k+1|k}$  (pozri (2.49a)), ktorý však ešte nezahŕňa príspevok príslušného pozorovania  $\mathbf{y}_{k+1}$  a je preto opäť upresnený v rámci korektora v kroku  $k+1$ . Túto procedúru možno spolu s výpočtom apriórnej a aposteriórnej kovariančnej matice chyby odhadu a výpočtom Kalmanovho zosilnenia zhrnúť v súbore Rov. (2.48)–(2.49).



Obr. 2.19. Ilustrácia korekčného a predikčného kroku v rámci jednej iterácie Kalmanovho filtra s príslušným značením apriórneho a aposteriórneho odhadu stavu.

Uvedené značenie apriórneho a aposteriórneho odhadu stavu je znázornené na Obr. 2.19. Tmavosivý obdĺžnik zároveň označuje jednu iteráciu algoritmu Kalmanovho filtra v čase  $k$  s ilustráciou jeho korekčného a predikčného kroku. Všimnime si, že sme tu tiež využili často používanú zjednodušujúcu konvenciu v značení apriórnych a aposteriórnych hodnôt odhadnutého stavu a kovariančnej matice chyby odhadu:  $(\cdot)_k^- \equiv (\cdot)_{k|k-1}$ ,  $(\cdot)_k^+ \equiv (\cdot)_{k|k}$ ,  $(\cdot)_{k+1}^- \equiv (\cdot)_{k+1|k}$  atď. Korektor teda využíva pozorovanie  $\mathbf{y}_k$  na prechod z  $\hat{\mathbf{x}}_k^-$  na  $\hat{\mathbf{x}}_k^+$  a prediktor využíva predpokladaný model systému na predikciu stavu z  $\hat{\mathbf{x}}_k^+$  na  $\hat{\mathbf{x}}_{k+1}^-$ .



Obr. 2.20. Blokový diagram rekurzívneho algoritmu Kalmanovho filtra.

- Korektor

- optimálne Kalmanovo zosilnenie

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{C}^T \left( \mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R} \right)^{-1} \quad (2.48a)$$

- aktualizovaný – aposteriórny odhad stavu

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left( \mathbf{y}_k - \mathbf{C} \hat{\mathbf{x}}_{k|k-1} \right) \quad (2.48b)$$

- aktualizovaná – aposteriórna kovariančná matica chyby odhadu

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{C}) \mathbf{P}_{k|k-1} \quad (2.48c)$$

- Prediktor

- predikovaný – apriórny odhad stavu

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{A} \hat{\mathbf{x}}_{k|k} + \mathbf{B} \mathbf{u}_k \quad (2.49a)$$

- predikovaná – apriórna kovariančná matica chyby odhadu

$$\mathbf{P}_{k+1|k} = \mathbf{A} \mathbf{P}_{k|k} \mathbf{A}^T + \mathbf{Q} \quad (2.49b)$$

Uvedené kroky predikcie a korekcie sa v každom časovom kroku,  $k$ , opakujú – Kalmanov filter je teda tzv. *rekurzívny* algoritmom odhadu stavu. To znamená, že na výpočet apriórneho odhadu premenných  $\hat{\mathbf{x}}_{k+1|k}$  a  $\mathbf{P}_{k+1|k}$  v kroku  $k$  sú potrebné len ich hodnoty vypočítané v kroku  $k - 1$ , t.j.  $\hat{\mathbf{x}}_{k|k-1}$  a  $\mathbf{P}_{k|k-1}$ , a aktuálne meranie  $\mathbf{y}_k$ , pričom podstatným z pohľadu ďalšieho použitia je samozrejme aposteriórny odhad stavu,  $\hat{\mathbf{x}}_{k|k}$ . Na rozdiel od tzv. dávkových metód odhadu teda nie je potrebná znalosť celej histórie predchádzajúcich pozorovaní a/alebo odhadov. Rekurzívny algoritmus diskrétneho Kalmanovho filtra je ilustrovaný na Obr. 2.20.

Je zrejmé, že v čase  $k = 0$  musíme algoritmus Kalmanovho filtra inicializovať špecifikovaním počiatočného odhadu hodnôt stavového vektora a kovariančnej matice chyby odhadu, t. j.  $\hat{\mathbf{x}}_0$  a  $\mathbf{P}_0$ . Kovariančná matica  $\mathbf{P}_0$  je spravidla diagonálna, pričom jej diagonálne prvky prislúchajú očakávaným rozptylom príslušných stavov, t. j. vyjadrujú mieru istoty, ktorú máme v inicializácii počiatočných hodnôt stavov systému,  $\hat{\mathbf{x}}_0$ . Ak si teda myslíme, že odhadnutý počiatočný stav nie je blízky k jeho skutočnej hodnote, prvky kovariančnej matice by mali mať veľké hodnoty. Naopak, ak je inicializácia veľmi dobrá, t. j. máme vysokú istotu v presnosti počiatočného odhadu, volíme nižšie hodnoty. Volba  $\mathbf{P}_0 = \mathbf{0}$  by znamenala, že dané počiatočné stavy sú dokonale presné, čo sa v praxi takmer nikdy nestáva.



V literatúre sa pravdepodobne častejšie stretнем s formuláciou algoritmu Kalmanovho filtra s opačným poradím jeho krokov, t. j. prediktor–korektor. Ich chronologická interpretácia je v tomto tvare možno praktickejšia (daná iterácia algoritmu skončí výpočtom aposteriórneho odhadu), tu sme ju však neuprednostnili, keďže Kalmanov filter v danom prípade uvažuje stavovú rovnicu v posunutom tvare oproti (2.47a), t. j.  $\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$ . I keď odhad stavu je v oboch prípadoch ekvivalentný, simulačná implementácia nami uvažovanej formulácie algoritmu (korektor–prediktor) s dynamikou v tvare (2.47a) bude názornejšia.

Dôležitými ladiacimi parametrami Kalmanovho filtra sú už spomínané matice  $\mathbf{Q}$  a  $\mathbf{R}$ . Diagonálne prvky kovariančnej matice šumu merania  $\mathbf{R}$  predstavujú rozptyly jednotlivých meraných výstupov, ktoré sa dajú odvodiť z presnosti, resp. rozlíšenia príslušných snímačov. Kovariančná matica procesného šumu,  $\mathbf{Q}$ , sa naopak týka neistoty v stavovú rovnicu. Ak im dokonale dôverujeme, môžeme zvoliť  $\mathbf{Q} = \mathbf{0}$ ; takmer vždy je ale prítomná nejaká forma neurčitosti, či už ako dôsledok nepresnosti modelovania, chýb merania, diskretizácie, aproximácií derivácií atď. Niektoré formulácie ako zdroj procesného šumu uvažujú tiež merania vstupu.

Poznamenajme, že vzťah (2.48c) pre rekurzívny výpočet aposteriórnej kovariančnej matice chyby aposteriórneho odhadu,  $\mathbf{P}_{k|k}$ , je už zjednodušený. Ak by sme totiž v definícii  $\mathbf{P}_{k|k} = E \left[ (\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})^T \right]$  nahradili  $\hat{\mathbf{x}}_{k|k}$  podľa vzťahu (2.48b), a ďalej meranie  $\mathbf{y}_k$  nahradili podľa (2.47b), po úpravách by sme dostali:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{C}) \mathbf{P}_{k|k-1} (\mathbf{I} - \mathbf{K}_k \mathbf{C})^T + \mathbf{K}_k \mathbf{R} \mathbf{K}_k^T. \quad (2.50)$$

Tento vzťah, označovaný ako tzv. *Josephov tvar* rovnice aktualizácie kovariančnej matice, t. j. Riccatiho rovnice, platí pre akúkoľvek hodnotu zosilnenia  $\mathbf{K}_k$ . Ako si ďalej ukážeme, ak  $\mathbf{K}_k$  je optimálne Kalmanovo zosilnenie, potom možno vzťah

(2.50) zjednodušíť na tvar (2.48c). Ešte predtým si však ukážme ako získame vzťah na výpočet optimálneho Kalmanovho zosilnenia (2.48a).

Kalmanov filter je algoritmus optimálneho odhadu minimalizujúci strednú kvadratickú chybu, ktorá je typickou mierou kvality odhadu. Ako sa ďalej ukáže, kritérium minimálnej strednej kvadratickej chyby<sup>60</sup> viedie na odhad s kvadratickou stratovou, resp. účelovou funkciou. V našom prípade, chyba odhadu aposteriórneho stavu je definovaná ako  $\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}$  a snahou Kalmanovho filtra je minimalizácia očakávanej hodnoty kvadrátu veľkosti tohto vektora, t. j. strednej kvadratickej chyby  $E [\|\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}\|^2]$ . Toto zodpovedá minimalizácii stopy<sup>61</sup> kovariančnej matice chyby aposteriórneho odhadu,  $\mathbf{P}_{k|k}$ .

Roznásobením výrazov vo vzťahu (2.50) a jeho úpravou dostávame:

$$\begin{aligned}\mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{C} \mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1} \mathbf{C}^T \mathbf{K}_k^T + \mathbf{K}_k (\mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R}) \mathbf{K}_k^T \\ &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{C} \mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1} \mathbf{C}^T \mathbf{K}_k^T + \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T,\end{aligned}\quad (2.51)$$

kde  $\mathbf{S}_k = \mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R}$  predstavuje inováciu kovariancie chyby odhadu. Stopu matice  $\mathbf{P}_{k|k}$  minimalizujeme tak, že parciálnu deriváciu výrazu (2.51) podľa zosilnenia  $\mathbf{K}_k$  položíme rovnú nule. Využitím pravidiel maticového počtu dostávame:

$$\frac{\partial \text{tr} (\mathbf{P}_{k|k})}{\partial \mathbf{K}_k} = -2 (\mathbf{C} \mathbf{P}_{k|k-1})^T + 2 \mathbf{K}_k \mathbf{S}_k = 0. \quad (2.52)$$

Riešením Rov. (2.52) pre  $\mathbf{K}_k$  dostávame:

$$\mathbf{K}_k^* = \mathbf{P}_{k|k-1} \mathbf{C}^T \mathbf{S}_k^{-1} \quad (2.53)$$

$$= \mathbf{P}_{k|k-1} \mathbf{C}^T (\mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R})^{-1}. \quad (2.54)$$

Toto maticové zosilnenie  $\mathbf{K}_k^*$  je známe ako *optimálne* Kalmanovo zosilnenie, ktoré mimimalizuje stopu  $\mathbf{P}_{k|k}$ . Ak pre jednoduchosť ďalej zanedbáme značenie  $(\cdot)^*$  zdôrazňujúce optimalitu tohto zosilnenia, vidíme, že vzťah (2.54) je totožný s (2.48a).

Optimálne Kalmanovo zosilnenie  $\mathbf{K}_k$  „váhovaním“ tzv. inovácie  $\mathbf{y}_k - \mathbf{C} \hat{\mathbf{x}}_{k|k-1}$  vo vzťahu (2.48a) v podstate rozhoduje v akej miere budú meranie  $\mathbf{y}_k$  a apriórny odhad  $\hat{\mathbf{x}}_{k|k-1}$  prispievať k výpočtu aposteriórneho odhadu  $\hat{\mathbf{x}}_{k|k}$ . Ak je šum meraňia malý, meranie  $\mathbf{y}_k$  má väčšiu dôveru, a teda k výpočtu  $\hat{\mathbf{x}}_{k|k}$  prispieva viac ako odhad  $\hat{\mathbf{x}}_{k|k-1}$ . V opačnom prípade, ak je chyba apriórneho odhadu malá, má tento väčšiu dôveru, a teda rozhodujúcou mierou prispieva k výpočtu  $\hat{\mathbf{x}}_{k|k}$ . Túto skutočnosť možno dokázať aj matematicky určením hodnoty zosilnenia  $\mathbf{K}_k$  v dvoch

<sup>60</sup>angl. minimum mean square error (MMSE)

<sup>61</sup>V lineárnej algebre je stopa (angl. trace) štvorcovej matice definovaná ako súčet prvkov na jej hlavnej diagonále. Stopu matice  $\mathbf{A}$  matematicky označujeme ako  $\text{tr} (\mathbf{A})$ .

extrémnych prípadoch,  $\mathbf{R} \rightarrow \mathbf{0}$  a  $\mathbf{P}_{k|k-1} \rightarrow \mathbf{0}$  a jeho dosadením do vzťahu (2.48b):

$$\lim_{\mathbf{R} \rightarrow \mathbf{0}} \mathbf{K}_k = \lim_{\mathbf{R} \rightarrow \mathbf{0}} \frac{\mathbf{P}_{k|k-1} \mathbf{C}^T}{\mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R}} = \mathbf{C}^{-1} \quad \Rightarrow \quad \hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1},$$

$$\lim_{\mathbf{P}_{k|k-1} \rightarrow \mathbf{0}} \mathbf{K}_k = \lim_{\mathbf{P}_{k|k-1} \rightarrow \mathbf{0}} \frac{\mathbf{P}_{k|k-1} \mathbf{C}^T}{\mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R}} = \mathbf{0} \quad \Rightarrow \quad \hat{\mathbf{x}}_{k|k} = \mathbf{y}_k.$$

Ako sme už načrtli, rekurzívny vzťah pre výpočet kovariancie chyby aposteriórneho odhadu (2.51) možno zjednodušiť ak sa Kalmanovo zosilnenie rovná vyššie odvodenému optimálnemu zosilneniu (2.54), t. j.  $\mathbf{K}_k = \mathbf{K}_k^*$ . Ak obe strany Rov. (2.53) prenásobíme výrazom  $\mathbf{S}_k \mathbf{K}_k^T$ , dostávame:

$$\mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T = \mathbf{P}_{k|k-1} \mathbf{C}^T \mathbf{K}_k^T. \quad (2.55)$$

Je zrejmé, že na základe (2.55) sa posledné dva výrazy vo vzťahu (2.51) eliminujú, čoho následkom získame zjednodušený vzťah pre aktualizáciu kovariancie:

$$\begin{aligned} \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{C} \mathbf{P}_{k|k-1} \\ &= (\mathbf{I} - \mathbf{K}_k \mathbf{C}) \mathbf{P}_{k|k-1}, \end{aligned} \quad (2.56)$$

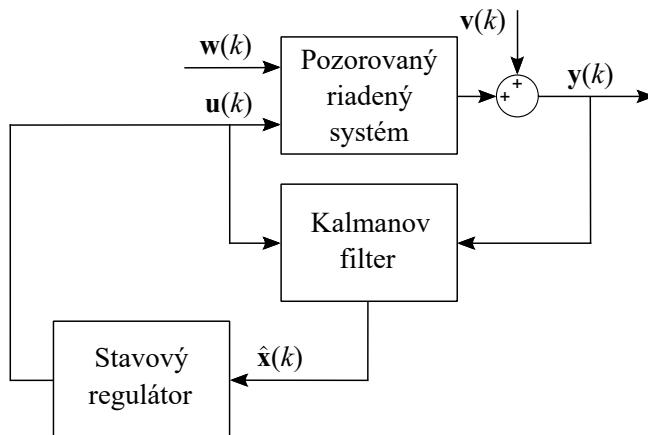
ktorý je totožný s (2.48c). V porovnaní so vzťahom (2.50) je tento výpočtovo menej náročný, a teda aj prakticky takmer vždy používaný – je však platný iba pre optimálne zosilnenie. Poznamenajme však, že v prípade nízkej aritmetickej presnosti zapríčinujúcej problémy s numerickou stabilitou, alebo uvažovania neoptimálneho Kalmanovho zosilnenia, by pre výpočet kovariančnej matice chyby aposteriórneho odhadu nemal byť použitý zjednodušený vzťah (2.56), ale Josephov tvar (2.50). V praktických implementáciách sa problémy s numerickou stabilitou (strata kladnej definitnosti alebo symetrie matice  $\mathbf{P}$ , výpočet inverzie  $\mathbf{S}_k$  atď.) Kalmanovho filtra naviac riešia pomocou rôznych numericky robustných a výpočtovo efektívnych faktorizačných/dekompozičných algoritmov a podobne; pozri napríklad [59].

V kontexte spomínamej minimalizácii strednej kvadratickej chyby odhadu poznámenajme, že Kalmanovu filtráciu nazývame tiež lineárnym kvadratickým odhadom<sup>62</sup> (LQE). LQE, resp. Kalmanov filter spolu s jeho duálnym problémom – lineárnym kvadratickým regulátorom (LQR) – rieši problém tzv. lineárneho kvadratického Gaussovského riadenia<sup>63</sup> (LQG), pričom už spomínaný princíp separácie garantuje, že LQE a LQR môžu byť navrhnuté a vypočítané nezávislé.

Obr. 2.21 znázorňuje použitie Kalmanovho filtra na optimálny odhad, resp. rekonštrukciu vnútorného stavu systému na základe meraných výstupov pre potreby stavového spätnoväzbového riadenia.

<sup>62</sup>angl. linear quadratic estimation (LQE)

<sup>63</sup>angl. linear quadratic-Gaussian (LQG) control



Obr. 2.21. Stavové spätnoväzbové riadenie s optimálnym odhadom stavu pomocou Kalmanovho filtra.

„Kalmanov filter je súbor matematických rovnic, ktorý predstavuje efektívny výpočtový (rekurzívny) nástroj na odhad stavu systému spôsobom, ktorý minimalizuje strednú hodnotu kvadrátu (rozptylu) chyby odhadu. Je veľmi užitočný vo viacerých ohľadoch: umožňuje odhad minulých, súčasných a do-konca budúcich stavov, a to aj v prípade keď presný charakter modelovaného systému nie je známy.“ [142]



Kalmanov filter sa používa na:

- získanie optimálneho odhadu stavových veličín, ktoré nie sú priamo merané, resp. merateľné;
- získanie najlepšieho odhadu sledovanej veličiny kombináciou – fúzioiu zašumiených meraní z viacerých senzorov.

Medzi jeho praktické výhody tiež patrí, že

- jeho návrh vyžaduje pomerne základné matematické znalosti;
- jeho rekurzívny algoritmus je obzvlášť vhodný pre online implementáciu v reálnom čase;
- ho možno použiť aj na priebežný odhad parametrov systému;
- tak ako moderné metódy optimálneho riadenia, využíva stavovo-priestorovú reprezentáciu dynamického systému.

Pre hlbšie štúdium problematiky optimálneho odhadu stavu a Kalmanovej filtrácie čitateľovi odporúčame špecifickú odbornú literatúru [134, 59].



Obr. 2.22. Rudolf E. Kálmán [44].

### Kálmán a jeho filter

Rudolf Emil Kálmán (1930 až 2016; Obr. 2.22) bol známy americký matematik narodený v Maďarsku. Z jeho prínosov je zdaleka najznámejší tzv. „Kalmanov filter“, ktorý navrhol v roku 1959 a neskôr v roku 1961 s R. S. Bucym jeho spojitéj verziu, tzv. „Kalman-Bucyho filter“. Kalmanov filter a jeho neskôrsie rozšírenia pre nelineárne problémy predstavujú najpoužívanejší „vedľajší“ produkt modernej teórie riadenia. Už krátko po svojom objavení sa významne osvedčil napríklad pri navigácii v rámci programu Apollo [58], čo vyžadovalo odhad trajektórií kozmickej lode vysielajúcej prvých ľudí na Mesiac. Jeho obľúbenosť v praktických aplikáciách vďačí najmä používaniu číslisových počítačov vo fáze návrhu ako aj implementácie. Z teoretického pohľadu Kalmanov filter zjednotil príbuzné koncepty odhadu stavu a riadenia a poukázal na ich dualitu. Už takmer 60 rokov od publikovania v článku „A new approach to linear filtering and prediction problems“ [72] si Kalmanov filter hľadá stále nové aplikácie v tak rôznorodých oblastiach ako sú napríklad predpovedanie počasia, investovanie do akcií, spracovanie signálov, ekonometrika, socioekonomika, satelitná navigácia, počítačové videnie, plánovanie a riadenie pohybu, monitorovanie stavu konštrukcií, seizmológia, riadenie motorov a mnohé iné.

Ukážme si ďalej na príklade invertovaného kyvadla odhad jeho stavov pomocou Kalmanovho filtra.

**Príklad 2.8.** Navrhnite a implementujte pozorovateľ stavu na báze Kalmanovho filtra pre optimálny odhad stavu systému invertovaného kyvadla schematicky znázorneného na Obr. 2.2. Jeho funkčnosť overte numerickou simuláciou, pričom uvažujte pohyb kyvadla výlučne v okolí jeho dolnej, stabilnej rovnovážnej polohy.

Pripomeňme si, že v našom príklade dynamického systému invertovaného kyvadla (2.9) uvažujeme meranie iba posunutia vozíka a uhlavej výchylky kyvadla, t. j.  $\mathbf{y} = [x_1 \ x_3]^T$ . Ich derivácie, t. j. rýchlosť teda nie sú priamo merané, čo je štandardným prípadom v praxi. Z pohľadu ďalšieho návrhu a implementácie stavového, napríklad prediktívneho riadenia, je teda potrebné priebežne (online) získavať celú informáciu o stave systému, a teda odhadovať plný stavový vektor  $\hat{\mathbf{x}}_k$ <sup>64</sup>.

Predpokladom pre návrh Kalmanovho filtra v odvodenom tvare (2.49)–(2.48) je diskrétny lineárny časovo-invariantný stavový model v tvare (2.47). My vzhľadom na zadanie budeme uvažovať konkrétny model systému linearizovaný v pravcovnom bode  $(\bar{\mathbf{x}}, \bar{u}) = (0 \ 0 \ \pi \ 0 \ 0)$ , ktorý sme si v tvare (2.35) odvodili v Kap. 2.3. Keďže uvažujeme stochastický systém (2.47), model (2.35) si rozšírime o procesný šum  $\mathbf{w}_k$  a šum merania  $\mathbf{v}_k$ , ktoré budú mať charakter nekorelovaných Gaussovských bielych šumov s normálnym rozdelením  $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$  a  $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ .

Odhad stavu simulovaného systému vykonáme v prostredí programu MATLAB. Ešte pred samotnou simuláciou si z praktických dôvodov vytvorime samostatnú funkciu `kalmanfilter()`, pomocou ktorej si kompaktne implementujeme rekurzívny algoritmus Kalmanovho filtra (pozri Obr. 2.20). Funkciu si vytvoríme v rovnomennom skripte `kalmanfilter.m`. Jej vstupom sú výstupný a vstupný signál,  $\mathbf{y}_k$  a  $\mathbf{u}_k$ , diskrétny LTI stavový model (2.35), počiatočný odhad stavu,  $\hat{\mathbf{x}}_0$ , a kovariančnej matice chyby odhadu,  $\mathbf{P}_0$ , a kovariančné matice šumov,  $\mathbf{Q}$  a  $\mathbf{R}$ , a výstupom odhad stavového vektora,  $\hat{\mathbf{x}}_k$ , v aktuálnom časovom kroku  $k$ .

Zdrojový kód 2.7. `kalmanfilter.m`

```
function x = kalmanfilter(y,u,x0,P0,sysd,Q,R)
[A,B,C,~]=ssdata(sysd); % Vycit. matic disk. stav. modelu
nx=length(A);
```

Pokračujeme inicializáciou apriórnych premenných  $\hat{\mathbf{x}}_{k|k-1}$  a  $\mathbf{P}_{k|k-1}$  v čase  $k = 0$ , t. j. zadaním počiatočného odhadu stavového vektora,  $\hat{\mathbf{x}}_0$ , a kovariančnej matice chyby odhadu,  $\mathbf{P}_0$ . Apriórny odhad stavu a kovariančnú maticu si pomocou príkazu `persistent` deklarujeme ako *pretrvávajúce*<sup>65</sup>, čo znamená, že budú lokál-

<sup>64</sup>Toto samozrejme znamená, že okrem nemeraných (nepozorovaných) stavov budú odhadnuté aj zašumené, priamo merané stavy, kde si ukážeme, že ich odhad má priaznivý filtračný efekt.

<sup>65</sup>angl. persistent

nymi premennými funkcie `kalmanfilter`, pričom ich hodnoty zostanú priebežne uchované v pamäti medzi jednotlivými volaniami tejto funkcie. Toto nám umožní označiť si  $\hat{x}_{k|k-1}$  a  $\hat{x}_{k+1|k}$ , resp.  $P_{k|k-1}$  a  $P_{k+1|k}$  spoločnou premennou `x_pred`, resp. `P_pred` („pred“ – predikovaný), t.j. bez zbytočného vytvárania a prepisovania špecificky značených premenných. Uvedená podmienka zabezpečí, že počatočné hodnoty `x_pred` a `P_pred` sa načítajú len pri prvom volaní tejto funkcie, kedy ešte nie sú uložené v jej pamäti. K ich samotnej volbe sa ešte vrátíme neskôr v rámci simulačného skriptu.

```
persistent x_pred P_pred
if isempty(x_pred)
    x_pred = x0;      % Pociatocny apriorný odhad stavu
    P_pred = P0;      % Poc. odhad apriornej kov. matice P
end
```

S pomocou apriórnej kovariančnej matice chyby odhadu pokračujeme v rámci ko-rektora výpočtom<sup>66</sup> optimálneho Kalmanovho zosilnenia  $\mathbf{K}_k$  podľa vzťahu (2.48a). Toto zosilnenie ďalej použijeme na výpočet aposteriórneho – aktualizovaného odhadu stavu podľa (2.48b) a tiež na aktualizáciu kovariancie podľa (2.48c).

```
K = (P_pred*C') / (C*C' + R); % Optim. Kalman. zosil.
x_odh = x_pred + K*(y - C*x_pred); % Apost. odhad stavu
P_odh = (eye(nx) - K*C)*P_pred; % Apost. kov. chyby odhadu
```

Nasleduje výpočet apriórneho – predikovaného odhadu stavu a kovariančnej matice podľa rovníc prediktora (2.49a) a (2.49b). Výsledok si označíme ako `x_pred` a `P_pred`.

```
x_pred = A*x_odh + B*u;      % Apriorný odhad stavu
P_pred = A*P_odh*A' + Q;      % Apriorna kov. chyby odhadu
```

Aposteriórne odhady `x_pred` a `P_pred` sú ako pretrvávajúce premenné opäť analogicky použité pri ďalšom volaní funkcie `kalmanfilter()`. Ešte predtým si získaný odhadnutý stav uložíme ako jej výstup a formálne ju ukončíme.

```
x = x_odh;                      % Odhadnutý stavovy vektor
end
```

Takto vytvorená funkcia<sup>67</sup> implementujúca rekurzívny algoritmus Kalmanovho filtra je pomerne univerzálna a možno ju efektívne volať v rámci simulácie. Vráťme sa teraz k simulácii systému invertovaného kyvadla s optimálnom odhadom stavu.

<sup>66</sup>V MATLABe je vo všeobecnosti z pohľadu rýchlosťi výpočtu a numerickej presnosti efektívnejšie namiesto príkazu `inv(A)*B`, resp. `B*inv(A)` použiť príkaz `A\B`, resp. `B/A`.

<sup>67</sup>MATLAB na implementáciu Kalmanovho filtra ponúka tiež vlastnú predvytvorenú funkciu `kalman()`, resp. `kalmd()`, ktoré sú súčasťou Control System Toolboxu.

Z praktických dôvodov<sup>68</sup> budeme ďalej uvažovať volnú odozvu systému z okolia jeho dolnej, stabilnej rovnovážnej polohy. Vytvorime si teraz simulačný skript *simKFkyvadlo.m*, na začiatku ktorého si vyčistíme pracovný priestor a tiež odstráname funkciu *kalmanfilter()*, resp. posledné hodnoty tej pretrvávajúcich premenných *x\_pred* a *P\_pred* z pamäte<sup>69</sup>.

Zdrojový kód 2.8. *simKFkyvadlo.m*

```
clear; clc; close all
clear kalmanfilter % Odstranenie funkcie z pamate
```

Pokračujeme načítaním diskrétneho LTI stavového modelu (2.35) z Pr. 2.6 a automatickým vyčítaním počtu jeho stavov, vstupov a výstupov.

```
load sysd % Nacitanie diskr. stav. modelu
[A,B,C,~]=ssdata(sysd); % Vycit. matic disk. stav. modelu
nx=length(A); % Pocet stavov
ny=size(C,1); % Pocet vystupov
```

Uvažujme počiatočný stav systému v jeho odchýlkovom tvaru  $\Delta \mathbf{x}_0 = [0 \ 0 \ -\frac{\pi}{8} \ 0]^T$  daný uhlovou výchylkou kyvadla  $-22.5^\circ$  ( $-\frac{\pi}{8}$  rad) od stabilnej rovnovážnej polohy, t. j. pracovného bodu  $\bar{\mathbf{x}}$ . Pripomeňme si, že tento v zmysle linearizácie nelineárneho modelu (2.9) získame pomocou transformácie  $\Delta \mathbf{x}_0 = \mathbf{x}_0 - \bar{\mathbf{x}}$  (pozri tiež Obr. 2.2).

```
x0=[0;0;7*pi/8;0]; % Pociatocny stav systemu
xlin=[0;0;pi;0]; % Pracovny (linearizacny) bod
dx0=x0-xlin; % Transf. stavu do odchylk. tvaru
```

Definujme si ďalej procesný šum  $\mathbf{w}_k$  a šum merania  $\mathbf{v}_k$ , ktoré budú mať charakter nekorelovaných Gaussovských bielych šumov s normálnym rozdelením  $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$  a  $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ . Príslušné kovariančné maticy  $\mathbf{Q} \in \mathbb{R}^{4 \times 4}$  a  $\mathbf{R} \in \mathbb{R}^{2 \times 2}$  uvažujme v nasledujúcim tvaru:

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \times 10^{-4} \end{bmatrix} = \text{diag}([0 \ 0 \ 0 \ 1e-4]), \quad (2.57)$$

$$\mathbf{R} = \text{diag}([5e-3 \ 5e-3]), \quad (2.58)$$

<sup>68</sup>Kedže neuvažujeme spätnoväzbový stabilizujúci regulátor, chceme aby sa systém nachádzal v blízkom okolí polohy stabilnej v otvorennej slučke, kde je jeho dynamika dobre opísaná linearizovaným modelom.

<sup>69</sup>Toto by sme alternatívne dosiahli príkazom *clear all* namiesto *clear* (menej efektívne).

t. j. na svojich diagonálach majú rozptyl<sup>70</sup> jednotlivých zložiek procesného šumu a šumu merania, pričom procesný šum pre jednoduchosť uvažujeme len v poslednej zo stavových rovníc (2.35a) opisujúcej výpočet uhlového zrýchlenia kyvadla<sup>71</sup>. V MATLABe si toto zapíšeme nasledovne:

```
w_var=[0 0 0 1e-4]'; % Rozptyl procesneho sumu
v_var=[5e-3 5e-3]'; % Rozptyl sumu merania
Q=diag(w_var); % Kovariancna matica proces. sumu
R=diag(v_var); % Kovariancna matica sumu merania
```

Pokračujeme definovaním simulačného času a predvytvorením štruktúr na ukladanie priebehov stavových a výstupných veličín. Analogicky si vytvoríme maticu  $\hat{dx}$  pre zápis odhadu stavového vektora.

```
T=10; % Konecny cas simulacie
cas=0:Ts:T-Ts; % Trvanie a krok simulacie
N=T/Ts; % Pocet krokov simulacie
dx=zeros(nx,N); % Prealok. matice pre ukladanie stavov
dx(:,1)=dx0; % Zapis pociatocneho stavu do matice dx
dy=zeros(ny,N); % Prealok. matice pre ukladanie vystupov
dhat=zeros(nx,N); % - / / - odhadnutych stavov
```

Ďalej si vygenerujeme náhodné priebehy procesného šumu  $\mathbf{w}_k$  a šumu merania  $\mathbf{v}_k$  s normálnym (Gaussovym) rozdelením s nulovými strednou hodnotou a rozptylmi danými maticami  $\mathbf{Q}$  (2.57) a  $\mathbf{R}$  (2.58). V MATLABe to vykonáme vytvorením matic (požadovaného rozmeru) náhodných<sup>72</sup> čísel s normálnym rozdelením pomocou príkazu `randn()`, ktoré po prvkoch prenásobíme ( $\cdot \cdot$ ) vektormi smerodajných odchýlok<sup>73</sup> príslušných šumov.

```
w=sqrt(w_var).*randn(nx,N); % Nahodny vektor proces. sumu
v=sqrt(v_var).*randn(ny,N); % Nahodny vektor sumu merania
```

Pred simuláciou odhadu stavu invertovaného kyvadla pomocou Kalmanovho filtra je potrebné si ešte zvoliť počiatočné hodnoty apriórneho odhadu stavového vektora a apriórnej kovariančnej matice chyby odhadu, v tomto prípade formálne značené  $\hat{\Delta}\hat{x}_{1|0}$  a  $\mathbf{P}_{1|0}$ , keďže v MATLABe začíname nenulovým krokom  $k = 1$ . V samotnom skripte si ich označíme intuitívne ako  $\hat{dx}_0$  a  $\hat{P}_0$  a zvolíme nasledovne:

<sup>70</sup>angl. variance

<sup>71</sup>Pre lepšiu ilustráciu viď plne nelineárnu dynamiku (2.9a).

<sup>72</sup>Ak chceme, aby dané náhodné sekvencie čísel boli pseudonáhodné, t. j. mohli sme ich opakovane využiť v rovnakom tvare v rôznych simuláciách, pred dané riadky používajúce príkaz `randn` napišeme príkazy `rng(1)` a `rng(2)`, ktorých zadáním pred budúcim použitím funkcie `randn()` túto inštruiujeme využiť príslušný uložený generátor náhodných čísel.

<sup>73</sup>Smerodajná odchýlka náhodnej premennej je druhou odmocninou jej rozptylu.

```
dx0=zeros (nx,1); % Pociatocny apriorny odhad stavu
P0=eye (nx); % Poc. odhad apriornej kov. matice P
```

V prípade uvedenej inicializácie pre ilustráciu predpokladáme, že systém sa nachádza v pokoji (stabilnej rovnovážnej polohe). Našej neistote v tento počiatočný stav prislúcha aj daná voľba diagonálnych prvkov kovariančnej matice

Ďalej môžeme pristúpiť k simulácii volnej odozvy stochastického diskrétneho stavového modelu (2.47), ktorú vykonáme jeho implementáciou vo `for` cykle s  $N$  krokmi. V každom kroku, resp. iterácii v rovnici dynamiky pripočítavame príslušný prvok vygenerovaného vektora procesného šumu,  $w(k)$ , a rovnako v rovnici výstupu (merania) príslušný prvok vektoru šumu merania  $v(k)$ . Kedže pozorujeme iba dva zo štyroch stavov systému, stavový vektor odhadneme volaním vytvorennej funkcie `kalmanfilter()`, ktorej vstupmi sú aktuálna hodnota zašumeneho výstupu, v tomto prípade nulový vstup, počiatočný apriórny odhad stavu a kovariancie chyby odhadu, model systému a kovariančné matice šumu merania a procesného šumu a jej výstupom je odhadnutý stav v danom kroku  $k$ .

```
for k=1:N
    dx (:,k+1)=A*dx (:,k)+w (:,k);
    dy (:,k)=C*dx (:,k)+v (:,k);
    dxhat (:,k)=kalmanfilter(dy (:,k),0,dx0,P0,sysd,Q,R);
end
```

Algoritmus diskrétneho Kalmanovho filtra (pozri Zdrojový kód 2.7) takto v každom kroku  $k$  na základe zašumeneho merania  $\Delta y_k$  vypočíta optimálny aposteriórny odhad stavu  $\hat{x}_k \equiv \hat{x}_{k|k}$ , ktorý sa zapíše do  $k$ -teho stĺpca matice  $dxhat$ . Kedže pracujeme s linearizovaným modelom v okolí stabilnej rovnovážnej polohy kyvadla, získané priebehy výstupov a odhadnutých stavov si teraz transformujme z odchýlkového tvaru späť do pôvodných súradníc<sup>74</sup>

```
x=dx+xlin; % Transf. stavov z dx na x
y=dy+xlin([1 3]); % Transf. vystupov z dy na y
xhat=dxhat+xlin; % Transf. odhadu z dxhat na xhat
```

Získané priebehy odhadnutých stavov  $\hat{x} = [\hat{x}_1 \hat{x}_2 \hat{x}_3 \hat{x}_4]^T$  spolu s meranými výstupmi  $y = [y_1 y_2]^T$  s, ako aj bez uvažovania šumu merania si vykreslíme pomocou už známych príkazov:

```
figure('color','w') % Otvorenie grafickeho okna
subplot(4,1,1) % Prvy z podgrafov
plot(cas,y(1,:)); % Vykresli meranie y1
```

<sup>74</sup>Priebehy stavov  $x$ , ktoré „nevidíme“, si ukladáme iba za účelom vykreslenia výstupov systému bez vplyvu meracieho šumu, pre ktoré platí  $y = Cx$ .

```

hold on; grid on           % Podrz graf a zobraz mriezku
plot(cas,x(1,1:end-1));  % Vykresli skutocny stav x1
plot(cas,xhat(1,:));     % Vykresli odhadnuty stav xhat1
ylabel('y_1, x_1 (m)');    % Oznacenie osi y
legend('stav-odhadnuty','stav-skutocny','meranie') % leg.
subplot(4,1,2)           % Druhy z podgrafov
plot(cas,x(2,1:end-1));  % Vykresli skutocny stav x2
hold on; grid on
plot(cas,xhat(2,:));     % Vykresli odhadnuty stav xhat2
ylabel('x_2 (ms^{-1})');
legend('stav-odhadnuty','stav-skutocny')
...
xlabel('t (s)');         % Oznacenie osi x

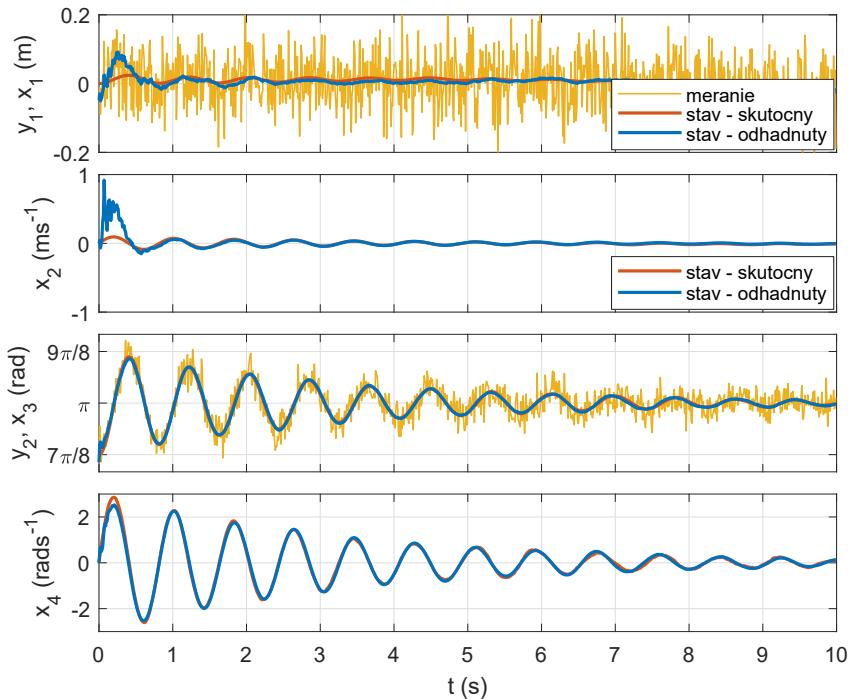
```

Poznamenajme, že vykreslenie tretieho a štvrtého podgrafa je analogické ako v prípade prvých dvoch stavov a ich odhadov, a preto je vo vyššie uvedenej časti skriptu pre prehľadnosť vynechané a nahradené symbolom „...“.

Grafický výstup skriptu *simKFkyvadlo.m* je zobrazený na Obr. 2.23, kde priebehy zašumených meraní  $y_1$  a  $y_2$  sú znázornené žltou farbou a na základe nich a príslušného modelu odhadnuté všetky štyri stavy modrou farbou. Všetky grafy sú zároveň pre overenie presnosti odhadu doplnené o skutočné priebehy stavov (červená farba) podľa (2.47a) (vrátane procesného šumu), ktoré v prípade reálneho systému samozrejme nie sú k dispozícii. Ako možno vidieť, odhady stavov veľmi dobre sledujú ich skutočné hodnoty, pričom šum merania je efektívne odfiltrovaný. Vidíme tiež, že hodnoty  $\hat{x}_k$  pomerne rýchlo skonvergujú z počiatočného odhadu  $\hat{x}_0$ , čo vidíme najmä na uhlovej výchylke, kde  $\hat{x}_3$  v rámci prvej vzorky skonverguje z hodnoty  $\hat{x}_{3,0} = \Delta\hat{x}_{3,0} + \bar{x}_3 = \pi$  na hodnotu blízku  $x_{3,0} = \Delta x_{3,0} + \bar{x}_3 = \frac{7}{8}\pi$ .

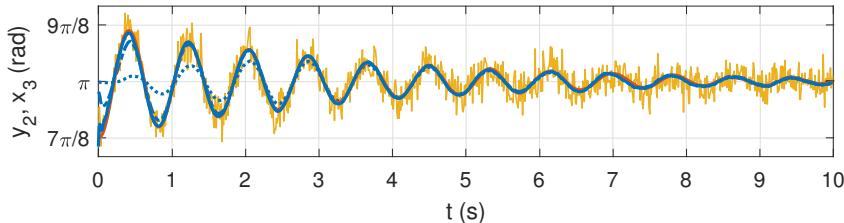
Pri tejto prfležitosti si tiež ilustrujme vplyv voľby počiatočného odhadu kovariančnej matice chyby odhadu,  $\mathbf{P}_0$ , na priebeh odhadu uhlovej výchylky kyvadla pri počiatočnom odhade stavu  $\hat{x}_0 = [0 \ 0 \ \pi \ 0]^T$ . Ako možno vidieť na Obr. 2.24, pri postupnom zmenšovaní hodnôt diagonálnych prvkov matice  $\mathbf{P}_0$  vyjadrujeme čoraz väčšiu istotu v počiatočný odhad stavu  $\hat{x}_0$ , ktorý sme si v prípade  $\hat{x}_3$  zámerne zvolili ako uhlovú výchylku 0 namiesto skutočných  $\frac{\pi}{8}$  rad od stabilnej rovnovažnej polohy  $x_{3,0} = \pi$ . Táto väčšia dôvera v nepresný počiatočný odhad stavu sa preto prejavuje jeho pomalšou konvergenciou k skutočnej hodnote. Poznamenajme, že všetky priebehy odhadu na Obr. 2.24 boli pre porovnanie získané pri uvažovaní reprodukovateľných, t.j. pseudonáhodných priebehov šumov  $\mathbf{w}$  a  $\mathbf{v}$ , identických ako v simulácii na Obr. 2.23.

Priebeh, resp. kvalitu odhadu by sme si mohli overiť aj vykreslením diagonálnych prvkov, resp. stopy aposteriórnej kovariančnej matice chyby odhadu stavu,



Obr. 2.23. Simulácia voľnej odozvy diskrétneho stavového modelu invertovaného kyvadla s odhadom jeho stavu pomocou Kalmanovho filtra.

$\mathbf{P}_k \equiv \mathbf{P}_{k|k}$ , ktorá je minimalizovaná v zmysle (2.52). V tom prípade je potrebné jej aktuálnu hodnotu v každom kroku  $k$  poslať na výstup funkcie `kalmanfilter` analogicky ako aposteriórny odhad stavu  $\hat{\mathbf{x}}_k \equiv \hat{\mathbf{x}}_{k|k}$  (pozri Zdrojový kód 2.7).

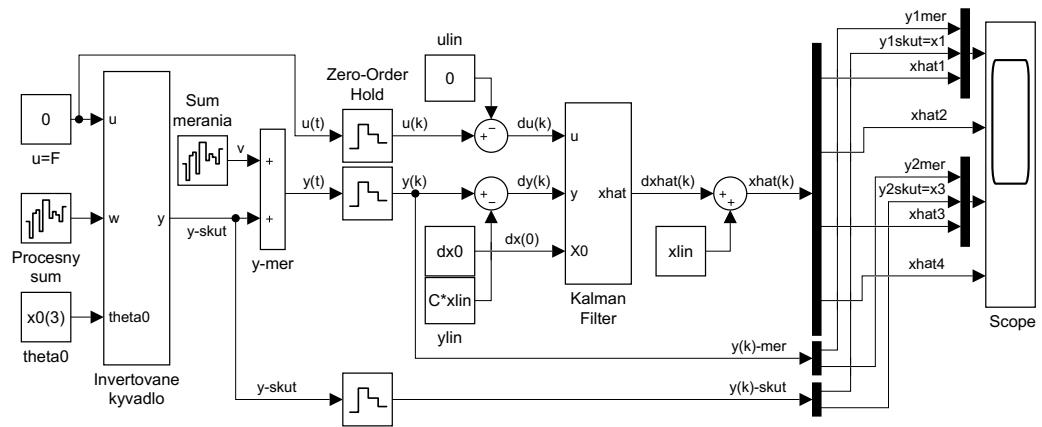


Obr. 2.24. Porovnanie odhadu uhlovej výchylky invertovaného kyvadla s uvažovaním počiatočného odhadu stavu  $\hat{\mathbf{x}}_0 = [0 \ 0 \ \pi \ 0]^T$  a kovariančnej matice chyby odhadu  $\mathbf{P}_0 = \mathbf{I}$  (plná modrá čiara),  $\mathbf{P}_0 = \mathbf{I} \cdot 1e-3$  (bodkočiarkovaná modrá čiara) a  $\mathbf{P}_0 = \mathbf{I} \cdot 1e-5$  (bodkovaná modrá čiara); červená čiara – skutočná uhlová výchylka, žltá čiara – meranie.

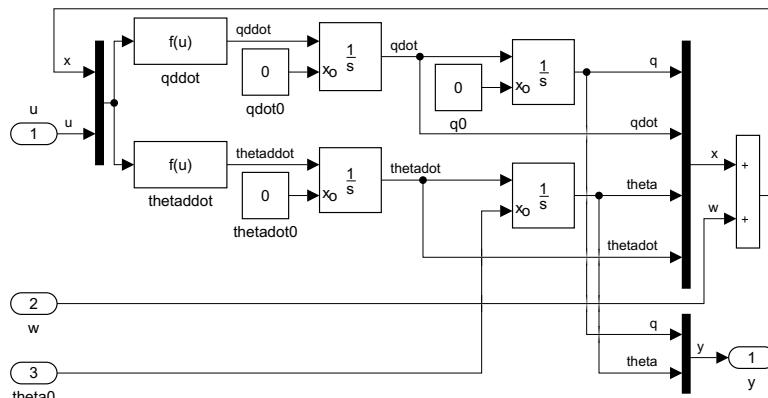
Tento príklad môžeme tiež jednoducho riešiť v prostredí MATLAB–Simulink. Všimnime si, že v prípade riešenia pomocou skriptu sme dynamiku systému simulovali diskrétnym LTI stavovým modelom (2.35) rozšíreným o šumové zložky v zmysle (2.47). Tento linearizovaný model je, ako už vieme, platný iba v blízkom okolí dolnej, stabilnej rovnovážnej polohy kyvadla. Aby sme sa neopakovali, implementujme v Simulinku plný nelineárny model invertovaného kyvadla (2.9). Aj keď ide o spojity model, ukážeme si, že ho môžeme jednoducho využiť v simulácii s Kalmanovym filtrom navrhnutým na základe diskrétneho linearizovaného stavového modelu (2.35).

Vytvorime si nový Simulink model a nastavme jeho simulačné parametre identicky ako v prípade simulácií spojitého modelu v Pr. 2.3, resp. Pr. 2.5. Začnime vytvorením podsystemu (*Subsystem*) implementujúceho spojity nelineárny stavový model (2.9)<sup>75</sup>. Postupujeme tak ako v prípade blokovej schémy na Obr. 2.3, ktorú v zmysle (2.47) rozšírimo o šum merania a iba pre názornosť aj o procesný šum. Tieto môžeme vygenerovať pomocou blokov *Band-Limited White Noise*, kde v poliach *Noise Power* zadáme rozptyly príslušných nekorelovaných, diskrétnych bielych šumov v tvaru  $v\_var \cdot Ts$ , resp.  $w\_var \cdot Ts$ , a v poliach *Seed* zadáme na vzájom rozdielne vektory pre generátor postupnosti náhodných čísel s normálnym rozdelením, napr. [12347] a [23341]. Aby sme zašumené meranie  $y_k$  mohli spracovať diskrétnym Kalmanovym filtrom, prevedieme daný signál cez blok *Zero-Order Hold*, ktorý jeho hodnotu podrží počas jednej vzorky v trvaní  $Ts$ . Kalmanov filter implementujeme jednoducho pomocou bloku *Kalman Filter* z Control System Toolboxu. Po jeho rozkliknutí nastavíme *Time Domain* na *Discrete-Time* a v záložke *Model Parameters*→*System Model* špecifikujeme diskrétny stavový model pomocou jeho matíc, v záložke →*Initial Estimates* nastavíme zadanie počiatočného odhadu cez vstupný port a v záložke →*Noise Characteristics* si vyberieme kovariančné matice procesného šumu a šumu merania,  $\mathbf{Q}$  a  $\mathbf{R}$ . Keďže uvažovaný Kalmanov filter je navrhnutý na základe diskrétneho LTI stavového modelu získaného linearizáciou nelineárnej dynamiky systému v stabilnej rovnovážnej polohe, musíme meranie a v našom prípade nulový vstupný signál na vstupe bloku transformovať do odchýlkového tvaru. Ako externý vstup bloku sme si tiež zvolili počiatočný odhad stavu, ktorý zadáme v odchýlkovom tvaru. Výstupom z bloku je aposteriórny odhad stavov systému,  $\hat{x}_k$ , ktorý pred vykreslením transformujeme do pôvodných súradníc. V záložke *Options* bloku Kalmanovho filtra môžeme podľa potreby pridať na výstup aj aposteriórnu kovariančnú maticu chyby odhadu. Do grafického znázornenia odhadnutých stavov môžeme podobne ako v skripte pridať aj príslušné merané výstupy systému s, ako aj bez uvažovania šumu merania.

<sup>75</sup>Keďže nelineárny stavový model v tvaru (2.9) vyžaduje znalosť fyzikálnych parametrov systému, tieto je potrebné pred spustením simulácie načítať pomocou príkazu `load param`.



Obr. 2.25. Bloková schéma ilustrujúca odhad stavu invertovaného kyvadla pomocou Kalmanovho filtra v prostredí Simulink.



Obr. 2.26. Podsystem Simulink schémy na Obr. 2.25 implementujúci nelineárny stavový model (2.9) invertovaného kyvadla.

Numerickou simuláciou schémy na Obr. 2.25 získame obdobné výsledky ako pomocou skriptu *simKFkyvadlo.m*. Formálnym rozdielom je len to, že neuvažujeme identicky vygenerované náhodné (pseudonáhodné) sekvencie šumov a tiež, že pre simuláciu systému uvažujeme nelineárny model. Podotknime však, že ak by sme na tento účel použili spojity, resp. diskrétny linearizovaný model, (2.23) resp. (2.35), tak ako v schéme na Obr. 2.9 resp. Obr. 2.15, jeho správanie sa v blízkom okolí pracovného bodu by bolo veľmi podobné, pozri Obr. 2.7, resp. Obr. 2.12.

□

## 2.5 Rozšírený Kalmanov filter

Kalmanov filter odvodený v predchádzajúcej podkapitole je definovaný, a teda aj použiteľný výlučne pre lineárne systémy. Ako už vieme, väčšina systémov v technickej praxi má však nelineárnu dynamiku, a preto ak to aplikácia vyžaduje, je nevyhnutné v tomto prípade uvažovať tzv. nelineárny odhad stavu<sup>76</sup>. Aj keď predmetom ďalších kapitol bude návrh lineárneho prediktívneho riadenia, stav systému je veľmi často nevyhnutné odhadovať aj mimo oblasti platnosti lineárneho, resp. linearizovaného modelu, na základe ktorého je daný regulátor navrhnutý, prípadne v celom pracovnom rozsahu systému.

Na základe silnej praktickej motivácie bolo teda krátko po návrhu Kalmanovho filtra navrhnuté aj jeho rozšírenie pre nelineárne systémy – rozšírený Kalmanov filter<sup>77</sup> (EKF), ktorý predstavuje štandard v teórii nelineárneho odhadu. Rozšírený Kalmanov filter v podstate *linearizuje* daný nelineárny model v pracovnom bode, ktorým je aktuálny odhad stavu. Odvodme si matematické pozadie diskrétneho EKF. Pre jeho návrh uvažujme diskrétny nelineárny stavový model v tvare<sup>78</sup>:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \quad (2.59a)$$

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k, \quad (2.59b)$$

kde  $\mathbf{w}_k$  a  $\mathbf{v}_k$  predstavujú procesný šum a šum merania, ktoré uvažujeme ako nekorelované viacozmerné Gaussovské šumy s kovarianciami  $\mathbf{Q}$  a  $\mathbf{R}$ . Uvedené šumy v stavovej rovnici (2.59a) a rovnici výstupu (2.59a) uvažujeme ako aditívne<sup>79</sup>.

Poznamenajme, že pre názornosť uvažujeme diskrétny nelineárny stavový model, ktorý sme si v Kap. 2.1 a 2.3 explicitne neodvodzovali. Podobným spôsobom je však možné použiť aj jeho spojité verziu, ako si načrtнемe neskôr.

Ako je zrejmé z (2.59) (bez uvažovania šumov), funkciu  $\mathbf{f}$  možno použiť na výpočet prediktovaného stavu na základe aktuálneho odhadu, a podobne funkciu  $\mathbf{h}$  možno použiť na výpočet predikovaného merania na základe predikovaného stavu. Funkcie  $\mathbf{f}$  a  $\mathbf{h}$  však nie je možné využiť na priamy výpočet kovariancie, a preto sa namiesto nich vypočítajú matice ich prvých derivácií – Jakobiány; pozri koncept linearizácie v Kap. 2.2. Táto linearizácia je v každom časovom kroku lokálne vyhodnotená v hodnote aktuálneho apriórneho, resp. aposteriórneho odhadu stavu, a následne použitá na korekciu a predikciu stavov algoritmu Kalmanovho filtra.

<sup>76</sup>angl. nonlinear state estimation

<sup>77</sup>angl. extended Kalman filter (EKF)

<sup>78</sup>Označenie nelineárnych funkcií,  $\mathbf{f}$  a  $\mathbf{h}$ , sme pre jednoduchosť ponechali rovnaké ako v prípade spojitého nelineárneho stavového modelu (2.8a). Ich rovnosť samozrejme z matematického pohľadu neplatí. Zároveň tak ako v prípade (2.47) prakticky neuvažujeme  $\mathbf{h}$  ako funkciu vstupu.

<sup>79</sup>Toto znamená, že stav  $\mathbf{x}_{k+1}$  je lineárne závislý od procesného šumu  $\mathbf{w}_k$ , a meranie  $\mathbf{y}_{k+1}$  je lineárne závislé od šumu merania  $\mathbf{v}_k$ . V zložitejšom prípade, ak by šumy  $\mathbf{w}$  a  $\mathbf{v}$  boli neaditívne, stavová rovnica a rovnica merania by mali tvar:  $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{w}_k, \mathbf{u}_k)$  a  $\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$ .

Zhrňme si teraz algoritmus diskrétneho rozšíreného Kalmanovho filtra, ktorý pozostáva z nasledujúcich krokov:

- Inicializácia – voľba počiatočných hodnôt apriórneho odhadu stavu,  $\hat{\mathbf{x}}_{0|k-1}$ , a apriórnej kovariančnej matice chyby odhadu,  $\mathbf{P}_{k|k-1}$ , v kroku  $k = 0$ :

$$\hat{\mathbf{x}}_{0|k-1} = E[\mathbf{x}_0], \quad (2.60a)$$

$$\mathbf{P}_{0|k-1} = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_{0|k-1})(\mathbf{x}_0 - \hat{\mathbf{x}}_{0|k-1})^T], \quad (2.60b)$$

ktorá je nasledovaná rekurzívnym výpočtom korektora a prediktora v krokoch  $k = 0, 1, 2, \dots$ :

- Korektor – výpočet aposteriórneho odhadu stavu,  $\hat{\mathbf{x}}_{k|k}$ , a aposteriórnej kovariančnej matice chyby odhadu,  $\mathbf{P}_{k|k}$ , s pomocou vypočítaného zosilnenia EKF,  $\mathbf{K}_k$ , a aktuálneho merania,  $\mathbf{y}_k$ , podľa:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{C}_k^T \left( \mathbf{C}_k \mathbf{P}_{k|k-1} \mathbf{C}_k^T + \mathbf{R} \right)^{-1}, \quad (2.61a)$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (\mathbf{y}_k - \mathbf{C}_k \hat{\mathbf{x}}_{k|k-1}), \quad (2.61b)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{C}_k) \mathbf{P}_{k|k-1}, \quad (2.61c)$$

kde matica výstupu  $\mathbf{C}_k$  je v každom kroku  $k$  získaná vyhodnotením Jacobiho linearizácie funkcie merania (2.59b) podľa  $\mathbf{x}$  v apriórnom odhade stavu

$$\hat{\mathbf{x}}_{k|k-1}, \text{ t.j. } \mathbf{C}_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}}.$$

- Prediktor – výpočet apriórneho odhadu stavu,  $\hat{\mathbf{x}}_{k+1|k}$ , doprednou simuláciou dynamiky, a apriórnej kovariančnej matice chyby odhadu,  $\mathbf{P}_{k+1|k}$ , pre nasledujúci krok podľa:

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{f}(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k), \quad (2.62a)$$

$$\mathbf{P}_{k+1|k} = \mathbf{A}_k \mathbf{P}_{k|k} \mathbf{A}_k^T + \mathbf{Q}, \quad (2.62b)$$

kde matica dynamiky  $\mathbf{A}_k$  je v každom kroku  $k$  získaná vyhodnotením Jacobiho linearizácie funkcie dynamiky (2.59a) podľa  $\mathbf{x}$  v aposteriórnom odhade

$$\text{stavu } \hat{\mathbf{x}}_{k|k-1} \text{ a vstupe } \mathbf{u}_k, \text{ t.j. } \mathbf{A}_k = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k}^{80}.$$

Uvedený korekčný a predikčný krok tvoria rekurzívny algoritmus rozšíreného Kalmanovho filtra.

<sup>80</sup>Ak by šumy  $\mathbf{v}$  a  $\mathbf{w}$  boli neaditívne, podobným spôsobom by sme vypočítali príslušné Jacobiány  $\mathbf{V}_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{v}} \right|_{\hat{\mathbf{x}}_{k|k-1}}$  a  $\mathbf{W}_k = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{w}} \right|_{\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k}$ , ktoré by do Rov. (2.61a) a (2.62b) vstupovali nasledovne:  $\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{C}_k^T (\mathbf{C}_k \mathbf{P}_{k|k-1} \mathbf{C}_k^T + \mathbf{V}_k \mathbf{R} \mathbf{V}_k^T)^{-1}$  a  $\mathbf{P}_{k+1|k} = \mathbf{A}_k \mathbf{P}_{k|k} \mathbf{A}_k^T + \mathbf{W}_k \mathbf{Q} \mathbf{W}_k^T$ .

Na rozdiel od svojho lineárneho náprotivku však rozšírený Kalmanov filter nie je optimálny odhadovač stavu. Samozrejme, ak by stavová rovnica a rovnica merania boli obe lineárne, EKF by bol totožný ako „obyčajný“ Kalmanov filter.  $\mathbf{K}_k$  (2.61a) preto nazývame tiež *takmer optimálne* zosilnenie.

Uvedený rekurzívny algoritmus predstavuje rozšírený Kalmanov filter prvého rádu. EKF vyšieho rádu môžu byť získané ponechaním viacerých členov rozvojov funkcií  $\mathbf{f}$  a  $\mathbf{h}$  do Taylorovho radu pri výpočte matíc  $\mathbf{A}_k$  a  $\mathbf{C}_k$ . Tieto však zvyknú byť výhodnejšie iba ak je šum merania malý.

Poznamenajme tiež, že Jakobiány  $\partial\mathbf{f}/\partial\mathbf{x}$  a  $\partial\mathbf{h}/\partial\mathbf{x}$  nie je nutné počítať online, t. j. v každej iterácii algoritmu, keďže sa dajú vypočítať vopred, t. j. analyticky. V tom prípade počas behu algoritmu postačuje na získanie matíc  $\mathbf{A}_k$  a  $\mathbf{C}_k$  tieto analytické výrazy vyhodnotiť v príslušnom  $\hat{\mathbf{x}}_{k|k-1}$ , resp.  $(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k)$ .

Stavový vektor  $\hat{\mathbf{x}}_k \equiv \hat{\mathbf{x}}_{k|k}$ , takmer optimálne odhadnutý EKF na základe meraných výstupov  $\mathbf{y}_k$ , môžeme využiť na rekonštrukciu vnútorného stavu daného nelineárneho systému pre potreby stavového spätnoväzbového riadenia analogicky ako na Obr. 2.17.



Zapamätajme si, že ak daný nelineárny systém možno dobre approximovať jeho linearizáciou, potom je rozšírený Kalmanov filter dobrou voľbou pre odhad jeho stavu. Treba mať však tiež na pamäti, že EKF má nasledujúce nevýhody, resp. obmedzenia:

- analytický ako aj numerický výpočet Jakobiánov môže byť náročný v dôsledku zložitých derivácií;
- EKF nemožno použiť na odhad stavu nelineárnych systémov, ktoré nie je možné lineárne approximovať (model systému je napríklad nespojity). Funkcie  $\mathbf{f}$  a  $\mathbf{h}$  musia preto byť spojite *diferencovateľné*.
- v prípade silne nelineárnych systémov Jacobiho linearizácia, t. j. linearizácia prvého rádu, neposkytuje ich dobrú approximáciu. Keďže danú approximačnú chybu EKF nezohľadňuje vo svojom korekčnom/predikčnom kroku, odhadnutá kovariančná matica zvykne „podhodnotiť“ skutočnú kovariančnú maticu chyby odhadu, čo sa v prípade silných nelineárít prejaví nielen neoptimálnym, ale nepresným odhadom. Preto, ak je počiatočný odhad stavu zlý, alebo je systém modelovaný nesprávne, filter môže veľmi rýchlo divergovať.

Uvedeným nedostatkom sa možno prakticky vyhnúť použitím iných metód nelineárneho odhadu. Tieto si stručne spomenieme neskôr.

**Príklad 2.9.** Navrhnite a implementujte odhad stavu systému invertovaného kyvadla na báze rozšíreného Kalmanovho filtra. Jeho funkčnosť overte numerickou simuláciou vynútenej odozvy systému v otvorenej slučke, v počiatočnej konfigurácii s kyvadlom nachádzajúcim sa v jeho hornej, nestabilnej rovnovážnej polohe.

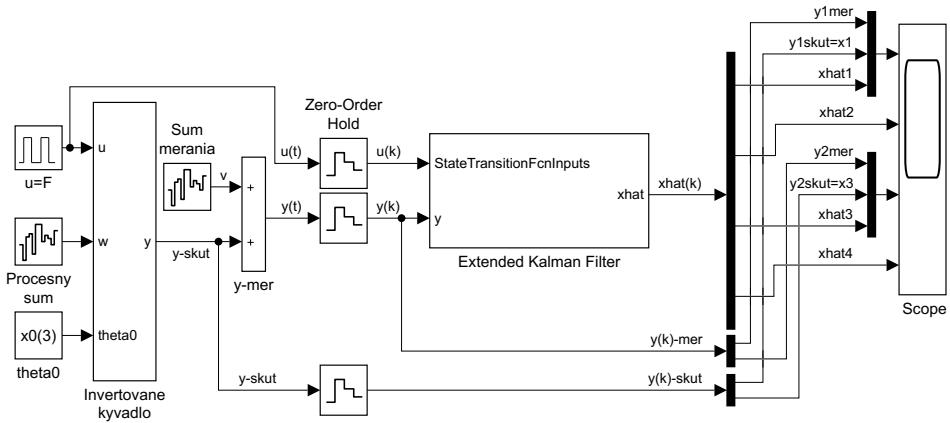
Uvažujeme teda, že dynamika a výstup systému na Obr. 2.2 sú opísané nelineárny stavovým modelom (2.9), pričom jeho počiatočný stav je  $\mathbf{x}_0 = [0 \ 0 \ 0]^T$ , čo zodpovedá nestabilnej rovnovážnej polohe. V tejto by systém teoreticky zotrval, pokiaľ naň nepôsobia žiadne vonkajšie sily. Ak by sme systém vybudili napríklad pôsobením silového impulzu na vozík, kyvadlo by sa vychýlilo z nestabilnej rovnovážnej polohy a tlmeným kmitaním postupne dostalo do jeho dolnej, stabilnej rovnovážnej polohy, pre ktorú platí  $\mathbf{x} = [0 \ 0 \ \pm\pi \ 0]^T$ . Kedže kyvadlo sa v tomto prípade pohybuje v celom svojom pracovnom rozsahu, uvažovanie nelineárneho modelu a teda aj nelineárneho odhadu stavu je nevyhnutné. Na tento účel využijeme rozšírený Kalmanov filter.

Príklad pre jednoduchosť nebudeme riešiť pomocou MATLAB skriptu<sup>81</sup>, ale praktickejšie v prostredí Simulink. Vytvorený model je na Obr. 2.27. Je zrejmé, že pri jeho vytváraní postupujeme z veľkej časti rovnako ako v prípade simulačnej schémy implementujúcej diskrétny Kalmanov filter z Pr. 2.8 na Obr. 2.25. Použijeme teda rovnaký podsystém s nelineárnym modelom (Obr. 2.26), a uvažujeme rovnaký procesný a merací šum. Rozdielom však je, že nemusíme jednotlivé signály transformovať z/do odchýlkového tvaru, pretože uvažovaný rozšírený Kalmanov filter je navrhnutý na základe diskrétneho nelineárneho stavového modelu. Kedže simulačný model je spojity, musíme vstupný a výstupný signal pred ich spracovaním EKF vzorkovať a tvarovať pomocou ZOH. Samotný EKF jednoducho implementujeme pomocou bloku *Extended Kalman Filter* z knižnice Control System Toolbox (dostupný od verzie MATLAB R2017a). Po rozkliknutí bloku si v záložke *System Model* → *State Transition* a → *Measurement* špecifikujeme nelineárnu funkciu dynamiky a v našom prípade lineárnu funkciu merania, ktoré si vytvoríme vo vlastných skriptoch ako je uvedené ďalej:

Zdrojový kód 2.9. funkciadynamiky.m

```
function x = funkciadynamiky (x, u)
Ts=0.01; % Vzork. perioda
m1=0.4; m2=0.08; l=0.2; g=9.81; b=0.1; % Parametre
```

<sup>81</sup>V tomto prípade a pri uvažovaní odvodeného diskrétneho EKF (2.60)-(2.62) by sme si potrebovali ešte zdiskretizovať spojity nelineárny stavový model (2.9), čo môžeme efektívne obísť riešením v Simulinku. Poznamenajme však, že implementácia v skripte by bola kombináciou algoritmu Kalmanovho filtra s prvkami Jacobiho linearizácie; vid Pr. 2.8 a Pr. 2.4. Alternatívne možno jednoducho využiť od verzie R2017a dostupnú funkciu `extendedKalmanFilter()`.



Obr. 2.27. Bloková schéma ilustrujúca odhad stavu invertovaného kyvadla pomocou rozšíreného Kalmanovho filtra v prostredí Simulink.

```

x=x+[ x (2);
(-m2*g*sin (x (3))*cos (x (3))+m2*l*((x (4))^2)*sin (x (3))
+b*m2*x (4)*cos (x (3))+u)/(m1+(sin (x (3))^2)*m2);
x (4);
((m1+m2)*(g*sin (x (3))-b*x (4))-(m2*l*((x (4))^2)*sin (x
(3))+u)*cos (x (3)))/(l*(m1+(sin (x (3))^2)*m2))] *Ts;

```

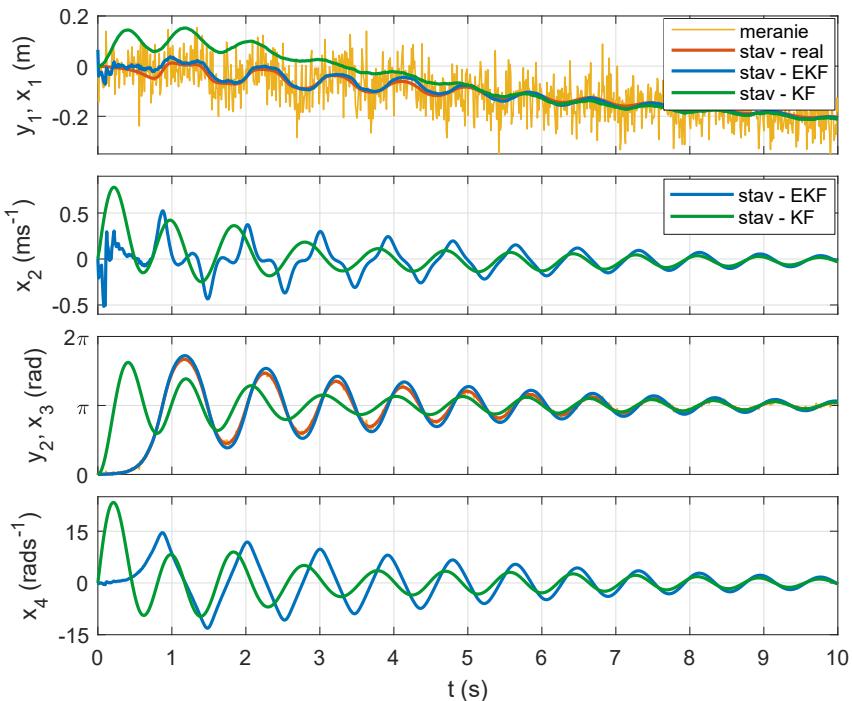
Zdrojový kód 2.10. funkciamerania.m

```

function y = funkciamerania(x)
y=[x (1);
x (3)];

```

Všimnime si, že keďže ide o diskrétny EKF, používame Eulerovu aproximáciu na diskretizáciu spojitej dynamiky v zmysle  $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}T_s$ , (2.34), zatiaľ čo rovinka merania je použiteľná ako diskrétna v rovnakom tvaru. Na urýchlenie je tiež možné po zakliknutí možnosti *jacobian* zadať analytický tvar príslušných Jakobiánov, čo pomôže znížiť výpočtový čas a numerické nepresnosti odhadu stavu. V spomínaných záložkách špecifikujeme tiež procesný a merací šum ako aditívne a zadáme ich príslušné kovariančné matice,  $\mathbf{Q}$  a  $\mathbf{R}$ . Dôležité je tiež v záložke *System Model* → *Initialization* zadať počiatočný odhad stavu,  $\hat{\mathbf{x}}_0 = \mathbf{0}$ , a kovariančnej matice odhadu stavu  $\mathbf{P}_0 = \mathbf{I}$ . Výstupom z bloku EKF je odhad stavu  $\hat{\mathbf{x}}_k$  v aktuálnom kroku simulácie,  $k$ , ktorého priebeh si ďalej uložíme, vykreslíme a prípadne porovnáme s priebehmi dostupných meraní pomocou bloku *Scope*.

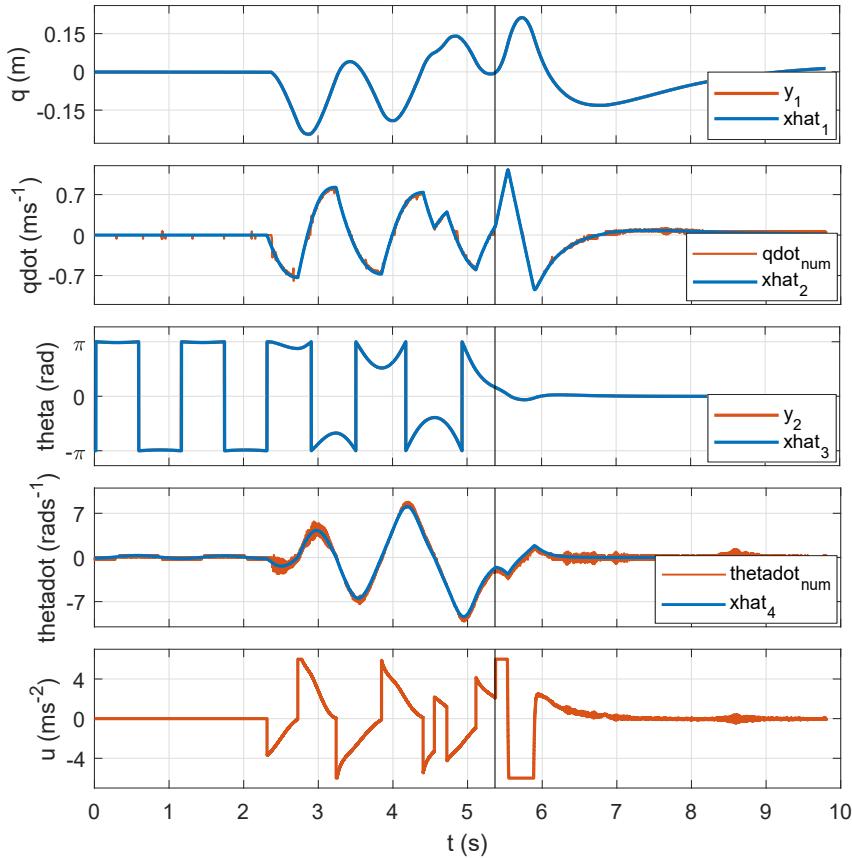


Obr. 2.28. Simulácia vynútenej odozvy nelineárneho stavového modelu invertovaného kyvadla s odhadom jeho stavu pomocou rozšíreného Kalmanovho filtra.

Získané priebehy odhadnutých stavov systému invertovaného kyvadla pomocou diskrétneho EKF sú znázornené na Obr. 2.28. Na ilustráciu sú tieto doplnené aj o odhad získaný pomocou Kalmanovho filtra navrhnutého na základe modelu linearizovaného v dolnej, stabilnej rovnovážnej polohe, ktorý je samozrejme mimo jej okolia veľmi nepresný. Ak by sme Kalmanov filter navrhli pre model linearizovaný v hornej, nestabilnej rovnovážnej polohe, odhad by veľmi rýchlo divergoval.

□

Pre zaujímavosť, Obr. 2.29 znázorňuje priebehy odhadu stavových veličín reálneho laboratórneho modelu invertovaného kyvadla znázorneného na Obr. 2.1. V uvedenom experimente môžeme vidieť riadené vyšvihnutie kyvadla z jeho dolnej, stabilnej rovnovážnej polohy,  $\theta = \pm\pi$ , do okolia hornej, nestabilnej rovnovážnej polohy,  $\theta = 0$ , kde platí  $\theta \in (-\frac{\pi}{6}, \frac{\pi}{6})$ , kde riadenie prevzal stabilizujúci prediktívny regulátor. Je zrejmé, že v tomto prípade, keď sa systém (minimálne počas vyšvihnutia) pohybuje v celom jeho pracovnom rozsahu, je použitie nelineárneho odhadu, v tomto prípade hybridného rozšíreného Kalmanovho filtra, nevyhnutné.



Obr. 2.29. Experimentálne výsledky odhadu stavu laboratórneho modelu invertovaného kyvadla pomocou rozšíreného Kalmanovho filtra.

Obrázok modrou farbou znázorňuje odhad jednotlivých stavov systému, pričom merania sú znázornené červenou farbou. Poznamenajme, že merané sú len výchylky  $x_1$  a  $x_3$ , pričom príslušné rýchlosťi  $x_2$  a  $x_4$  boli na ilustráciu vypočítané numerickou deriváciou<sup>82</sup> meraných výchyliek.

Ako je zrejmé, odhad stavových veličín je dostatočne presný, a tak dáva veľmi dobrý a zároveň nevyhnutný základ pre kvalitné stavové riadenie, pričom vypočítaný riadiaci vstup je znázornený na poslednom z podgrafov. Ako možno vidieť na priebehu uhlovej výchylky kyvadla, toto je postupným pridávaním energie do

<sup>82</sup>Týmto spôsobom možno vypočítať rýchlosť,  $v$ , na základe diskrétnych hodnôt príslušných výchyliek (meraní  $y$ ) pomocou numerickej integrácie prvého rádu, t. j. (späťnej) Eulerovej metódy, ako  $v(k) \approx (y(k) - y(k-1))/T_s$ . Takýto výpočet je však numericky veľmi citlivý a vyžaduje tiež ďalšiu filtráciu získaných signálov.

systému (pohybom vozíka) rozkmitané až sa dostane do okolia cieľovej nestabilnej polohy, v ktorej je potom stabilizované. Zvislé čierne čiary v jednotlivých grafoch ilustrujú moment prenutia regulátora vyšvihnutia na stabilizačný regulátor.



Ako je nám už známe, väčšina fyzikálnych systémov je opísaná spojitým modelom, zatial čo merania vstupujúce do algoritmu odhadu stavu sú získavané v diskrétnych časových okamihoch. Z tohto dôvodu je v praxi často používaný tzv. *spojito-diskrétny* alebo tiež *hybridný* rozšírený Kalmanov filter. Uvažujeme teda nasledujúci model:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) + \mathbf{w}(t), & \mathbf{w}(t) &\approx \mathcal{N}(\mathbf{0}, \mathbf{Q}(t)), \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k, & \mathbf{v}_k &\approx \mathcal{N}(\mathbf{0}, \mathbf{R}).\end{aligned}$$

Rekurzívny algoritmus preto vyzerá nasledovne:

- Korektor:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{C}_k^T \left( \mathbf{C}_k \mathbf{P}_{k|k-1} \mathbf{C}_k^T + \mathbf{R} \right)^{-1},$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left( \mathbf{y}_k - \mathbf{C}_k \hat{\mathbf{x}}_{k|k-1} \right),$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{C}_k) \mathbf{P}_{k|k-1},$$

$$\text{kde } \mathbf{C}_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}}.$$

- Prediktor:

$$\text{vyrieš } \begin{cases} \dot{\hat{\mathbf{x}}}(t) = \mathbf{f}(\hat{\mathbf{x}}(t), \mathbf{u}(t)), \\ \dot{\mathbf{P}}(t) = \mathbf{A}(t) \mathbf{P}(t) + \mathbf{P}(t) \mathbf{A}^T + \mathbf{Q}(t), \end{cases} \text{ pri } \begin{cases} \hat{\mathbf{x}}(t_k) = \hat{\mathbf{x}}_{k|k}, \\ \mathbf{P}(t_k) = \mathbf{P}_{k|k}. \end{cases}$$

$$\Rightarrow \begin{cases} \hat{\mathbf{x}}_{k+1|k} = \hat{\mathbf{x}}(t_{k+1}), \\ \mathbf{P}_{k+1|k} = \mathbf{P}(t_{k+1}). \end{cases}$$

$$\text{kde } \mathbf{A}(t) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}(t), \mathbf{u}(t)}.$$

pri inicializácii  $\hat{\mathbf{x}}_{0|-1} = E[\mathbf{x}_0]$  a  $\mathbf{P}_{0|-1} = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_{0|-1})(\mathbf{x}_0 - \hat{\mathbf{x}}_{0|-1})^T]$ .

Vidíme, že rovnice korektora sú identické ako pri diskrétnom EKF, zatial čo predikovaný stav a kovariancia sú získané riešením diferenciálnych rovnic prediktora s počiatočnou hodnotou rovnou aposteriornemu odhadu.

Analogicky možno tiež odvodiť lineárny, hybridný Kalmanov filter.

Poznamenajme, že na odhad stavu a parametrov nelineárnych systémov môžeme využiť aj rôzne iné a veľmi často efektívnejšie techniky ako napríklad tzv. nevoňavý Kalmanov filter, výberový filter, alebo odhad stavu s pohyblivým horizontom. Tieto sa používajú ak model systému nie je dobre známy alebo nepresný, respektíve dávajú zvyčajne lepšie výsledky, ak je model silne nelineárny, keď odhad získaný pomocou rozšíreného Kalmanovho filtra môže byť veľmi nepresný alebo dokonca divergovať.

*Nevoňavý*<sup>83</sup> Kalmanov filter<sup>84</sup> (UKF) [70, 71] využíva tzv. nevoňavú transformáciu, ktorá namiesto aproximácie nelineárneho modelu ako v prípade EKF aproximuje rozdelenie pravdepodobnosti pomocou tzv. sigma bodov. Týmto sa eliminuje potreba explicitného výpočtu Jakobiánov, ktorý môže byť v prípade zložitých funkcií  $\mathbf{f}$  a  $\mathbf{h}$  veľmi náročný až nemožný. Algoritmus UKF, i keď odhad podobne ako pri EKF nie je optimálny, nepotrebuje výpočet derivácií a je teda výpočtovo veľmi efektívny. Presnosť nevoňavej transformácie je až do 3. rádu Taylorovho rozvoja danej nelinearity, zatiaľ čo pri EKF ide o linearizáciu prvého rádu. Odhad pomocou UKF spravidla skonverguje k správnemu riešeniu rýchlejšie ako v prípade EKF[62]. V MATLABe môžeme UKF jednoducho implementovať pomocou funkcie `unscentedKalmanFilter()`, a v Simulinku pomocou bloku *Unscented Kalman Filter*.

Na veľmi podobnom princípe je založený *výberový* filter<sup>85</sup> [40], pričom od UKF sa zásadne odlišuje tým, že aproximuje nielen Gaussovo ale ľubovoľné rozdelenie. Na jeho reprezentáciu, ak nie je dopredu známa, je však potrebný väčší náhodný výber z daného typu rozdelenia pravdepodobnosti. V MATLABe ho možno implementovať funkciou `particleFilter()`.

Odhad stavu s *pohyblivým horizontom*<sup>86</sup> (MHE) [124, 125] je modernou a efektívnu optimalizačnou metódou odhadu stavu, ktorá na rozdiel od Kalmanovho filtra na nájdenie odhadu používa iteratívny prístup založený na tzv. nelineárnom programovaní (v prípade nelineárneho modelu systému). Pri MHE sa odhadnutý stav v každom kroku na základe modelu systému a histórie meraní určí ohraničenou minimalizáciou vhodnej zvolenej účelovej funkcie na horizonte konečnej dĺžky od aktuálneho času do minulosti. Vo svojej podstate tak MHE predstavuje duálny problém k problému prediktívneho riadenia (MPC).

---

<sup>83</sup>Ako povedal jeho hlavný tvorca Jeffrey Uhlmann, prílastok „nevoňavý“ vymyslel, keď jedného večera pracoval sám v laboratóriu a všimol si na stole niekoho dezodorant [43].

<sup>84</sup>angl. unscented Kalman filter (UKF)

<sup>85</sup>angl. particle filter

<sup>86</sup>angl. (nonlinear) moving horizon estimation ((N)MHE)

# Kapitola 3

## Predikcia

Čo znamená slovo predikcia? Je zložené z latinských slov „pre“, čo znamená pred, a „dicere“, čo znamená povedať. Predikcia je teda predpoved diania v budúcnosti na základe istých vedomostí alebo skúseností z minulosti. Otázka je, ako môžeme niečo predpovedať v riadení?

Určite nie pomocou veštenia (Obr. 3.1). Samozrejme, predvídať, respektíve predikovať niečo môžeme iba na základe matematického modelu. Zoberme si ako príklad predpoved počasia. Meteorológovia potrebujú na to nesmierne komplexný nelineárny model atmosferických javov a samozrejme aj známe počiatočné podmienky. Tento model je až taký nelineárny, že splňa definíciu chaotického systému<sup>1</sup>. Práve preto aj malé nepresnosti v určení počiatočných podmienok, ako je teplota a tlak, môžu spôsobiť obrovské rozdiely vo výsledkoch. Občas predpoved počasia vyzerá ako veštenie, ale ako vidíme, dôvody na nepresnosť sú exaktné.

Naštastie pri riadení dynamických systémov nemusíme využívať až tak komplexné matematické modely. Ako sme si už naznačili v predchádzajúcich kapitolách, naše bádanie obmedzíme výlučne na lineárne modely. Stále však máme na výber niekoľko spôsobov matematického opisu dynamiky, z ktorých si pre ďalšiu diskusiu vyberieme stavovú reprezentáciu. Analogicky k predpovedi počasia teda aj tu potrebujeme dve základné východiská:

- model a
- známe počiatočné podmienky,

ktorým v našom prípade sú

- diskrétna lineárna stavová reprezentácia a
- známy počiatočný stav.

---

<sup>1</sup>Teória chaosu má korene v modeloch predpovedí počasia.



Obr. 3.1. Veštenie nie je jediný spôsob ako predpovedať budúcnosť.

V tejto kapitole si vysvetlíme, ako môžeme v prediktívnom riadení odvodiť a využiť samotnú funkcionality predikcie.

### 3.1 O krok dopredu...

Uvažujme teraz, že na základe diskusie v Kap. 2 máme k dispozícii lineárny časovo invariantný diskrétny stavový model v tvare:

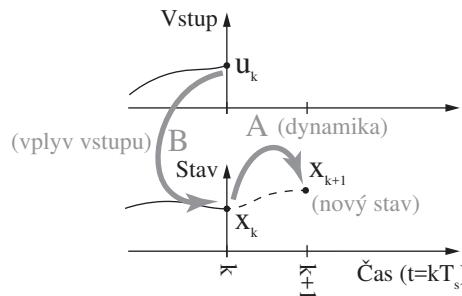
$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \quad (3.1a)$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k. \quad (3.1b)$$

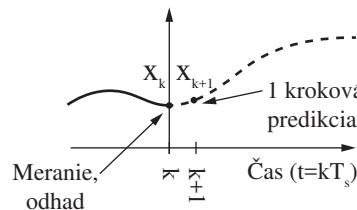
Označme si aktuálny diskrétny časový krok ako  $k$ . Čas  $k$  budeme jednoducho používať na vyjadrenie konceptu súčasnosti, t. j.  $k$  znamená *teraz*. V čase  $k$  máme potom k dispozícii stav  $\mathbf{x}_k$  na základe meraní zo snímačov alebo odhadnutý napríklad pomocou lineárneho Kalmanovho filtra (pozri Kap. 2.4). Tento stav musíme poznat na to, aby sme boli schopní predikovať ďalší vývoj dynamiky modelu.

V čase  $k + 1$  sme postúpili o krok ďalej a už máme k dispozícii stav  $\mathbf{x}_{k+1}$ . Tento stav už nepoznáme, nachádza sa o krok ďalej v budúcnosti vzhľadom na súčasný čas  $k$ . Ako ho teda môžeme predpovedať?

Odpovedou je práve náš stavový model (3.1), ktorý presne definuje ako bude vyzerat ďalší stav. V prípade, ak je súčasný stav známy a poznáme aj vstup  $\mathbf{u}_k$ ,



Obr. 3.2. Ak poznáme vstup a súčasný stav, na základe modelu vieme vypočítať nasledujúci stav.



Obr. 3.3. Diskrétny stavový model predstavuje jednokrokovú predikčnú rovnicu.

jednoducho dosadíme tieto známe veličiny do

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \quad (3.2)$$

na základe čoho dostaneme  $\mathbf{x}_{k+1}$  (Obr. 3.2).

Môžeme teda povedať, že diskrétny stavový model dynamiky systému predstavuje jednokrokovú predikciu stavu (Obr. 3.3). Toto je zrejmé v prípade diskrétneho modelu, ale aj v prípade spojitej stavovej reprezentácie dostaneme infinitezimálnu (nekonečne malú) zmenu stavu, teda určité smerovanie a tým predikciu stavu (Obr. 3.3). Pre analógiu v spojitom čase pozri Obr. 3.4.

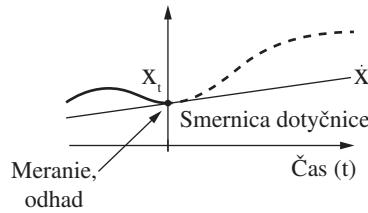
Diskrétna stavová rovnica predstavuje jednokrokovú predikciu stavu.



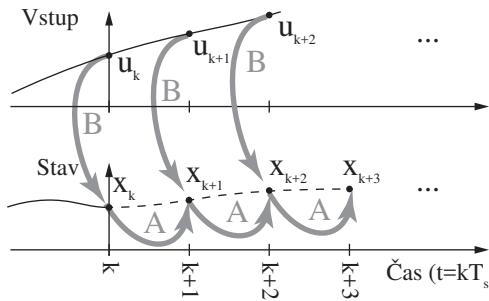
## 3.2 Predikcia do konca horizontu

Vypočítajme si teraz stav  $\mathbf{x}_{k+2}$  v čase  $k+2$ . Potrebujeme na to poznať stav  $\mathbf{x}_{k+1}$  z predchádzajúceho kroku a musíme poznať aj vstup  $\mathbf{u}_{k+1}$ , platí teda, že:

$$\mathbf{x}_{k+2} = \mathbf{A}\mathbf{x}_{k+1} + \mathbf{B}\mathbf{u}_{k+1}. \quad (3.3)$$



Obr. 3.4. Pri spojitej modeli predstavuje stavová rovnica istú analógiu predikcie, pretože derivácia predstavuje smernicu dotyčnice v danom čase  $t$ .



Obr. 3.5. Opakovaným (rekurzívnym) dosadzovaním stavov do predikčnej rovnice môžeme predikovať nasledujúce (budúce) stavy.

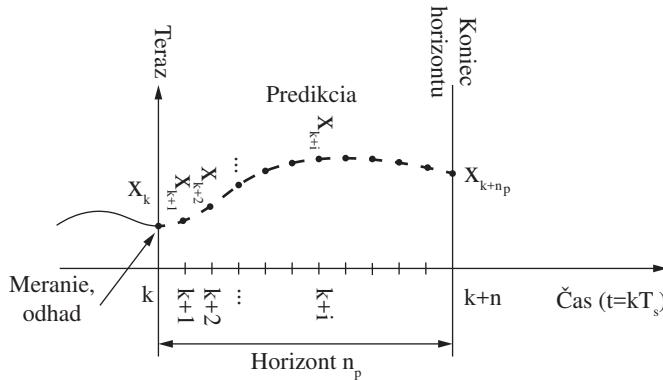
Stav  $\mathbf{x}_{k+1}$  ale môžeme vypočítať na základe Rov. (3.2) čo môžeme rovno dosadiť do Rov. (3.4) podla

$$\mathbf{x}_{k+2} = \mathbf{A}(\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k) + \mathbf{B}\mathbf{u}_{k+1}. \quad (3.4)$$

Vidíme, že ak poznáme počiatočnú hodnotu stavu  $\mathbf{x}_k$  na základe merania alebo odhadu, potom v prípade, že poznáme aj vstupy systému  $\mathbf{u}_k$  a  $\mathbf{u}_{k+1}$ , môžeme vypočítať ďalší stav  $\mathbf{x}_{k+2}$  (Obr. 3.5).

Prečo potom prestať pri stave  $\mathbf{x}_{k+2}$ ? Ak poznáme súčasný stav  $\mathbf{x}_k$  a vstupy do systému  $\mathbf{u}_k, \mathbf{u}_{k+1}, \mathbf{u}_{k+2}, \dots$  potom môžeme pomocou opakovaného – iným slovom *rekurzívneho* – dosadenia do stavovej rovnice predpovedať aj budúce stavy. Rekurzívny proces môžeme zopakovať lubovoľne veľakrát. Predpokladajme, že budúce stavy chceme poznáť len  $n_p$  krokov do budúcnosti. Číslo  $n_p$  potom predstavuje počet rekurzívnych dosadení do stavovej rovnice, a teda aj počet krokov predikcie do budúcnosti. Nazývame ho *horizont predikcie* alebo *predikčný horizont*. Horizont  $n_p$  je potom kladné celé číslo (skalár), t. j.  $n_p \in \mathbb{Z}, n_p \geq 1$ .

Do konca horizontu  $n_p$  môžeme vypočítať stavy podľa nasledujúcej sústavy

Obr. 3.6. Predikcia stavu  $n_p$  krokov do budúcnosti.

rovníc:

$$\begin{aligned}
 k & \quad \mathbf{x}_k = \mathbf{x}_k, \\
 k+1 & \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \\
 k+2 & \quad \mathbf{x}_{k+2} = \mathbf{A}\mathbf{x}_{k+1} + \mathbf{B}\mathbf{u}_{k+1} = \mathbf{A}^2\mathbf{x}_k + \mathbf{A}\mathbf{B}\mathbf{u}_k + \mathbf{B}\mathbf{u}_{k+1}, \\
 \vdots & \quad \vdots \\
 k+i & \quad \mathbf{x}_{k+i} = \mathbf{A}\mathbf{x}_{k+i-1} + \mathbf{B}\mathbf{u}_{k+i} \\
 \vdots & \quad \vdots \\
 k+n_p & \quad \mathbf{x}_{k+n_p} = \mathbf{A}\mathbf{x}_{k+n_p-1} + \mathbf{B}\mathbf{u}_{k+n_p-1} \\
 & \quad = \mathbf{A}^{n_p}\mathbf{x}_k + \mathbf{A}^{n_p-1}\mathbf{B}\mathbf{u}_k + \dots + \mathbf{A}\mathbf{B}\mathbf{u}_{k+n_p} + \mathbf{B}\mathbf{u}_{k+n_p-1},
 \end{aligned} \tag{3.5}$$

čo vlastne vyjadruje simuláciu stavu  $n_p$  krokov do budúcnosti (Obr. 3.6).

Ak poznáme súčasný stav a posledné vstupy do systému, na základe stavovej rovnice môžeme vypočítať ďalšie stavové vektory do ľubovoľne dlhého horizontu.



Na konci horizontu sme v čase  $k+n_p$  a tam dostaneme aj posledný predikovaný stav, ktorý si označíme ako  $\mathbf{x}_{k+n_p}$ , pozri Obr. 3.6. Všimnime si v súbore rovníc (3.5), že na predikciu  $\mathbf{x}_{k+n_p}$  potrebujeme celú postupnosť, alebo *sekvenciu*<sup>2</sup> vstupov, pričom posledný uvažovaný vstup  $\mathbf{u}_{k+n_p-1}$  je v čase  $(k+n_p-1)$ . Čas medzi začiatkom a koncom horizontu si označíme kladným celočíselným indexom  $i$ . Ľubovoľná vzorka vzdialenosť  $i$  krokov od začiatku je v čase  $k+i$ , a k nej prislúchajúci stav si označíme ako  $\mathbf{x}_{k+i}$ . Ďalším dôležitým faktorom je, že v čase  $k$  nemôžeme

<sup>2</sup>angl. sequence

hovorí o predikcii, keďže stav už poznáme. Tým pádom budú predikčné rovnice skrátené o prvý riadok, ktorý vyjadruje nadbytočnú informáciu, že  $\mathbf{x}_k = \mathbf{x}_k$ , a v čase  $k$  budú preto dané v tvare:

$$\begin{aligned}
 k+1 & \quad \mathbf{x}_{k+1} = \mathbf{Ax}_k + \mathbf{Bu}_k, \\
 k+2 & \quad \mathbf{x}_{k+2} = \mathbf{Ax}_{k+1} + \mathbf{Bu}_{k+1} = \mathbf{A}^2\mathbf{x}_k + \mathbf{ABu}_k + \mathbf{Bu}_{k+1}, \\
 \vdots & \quad \vdots \quad \vdots \\
 k+i & \quad \mathbf{x}_{k+i} = \mathbf{Ax}_{k+i-1} + \mathbf{Bu}_{k+i} = \dots, \\
 \vdots & \quad \vdots \quad \vdots \\
 k+n_p & \quad \mathbf{x}_{k+n_p} = \mathbf{Ax}_{k+n_p-1} + \mathbf{Bu}_{k+n_p-1} \\
 & \quad = \mathbf{A}^{n_p}\mathbf{x}_k + \mathbf{A}^{n_p-1}\mathbf{Bu}_k + \dots + \mathbf{ABu}_{k+n_p} + \mathbf{Bu}_{k+n_p-1}.
 \end{aligned} \tag{3.6}$$

Uvedená sústava rovníc plne vyjadruje našu snahu predikovať vývoj dynamiky modelu, avšak nie je úplne vhodná na formuláciu riadiaceho zákona. V ďalšej diskusii budeme súbor rovníc (3.6) postupne zapisovať v kompaktnom maticovom tvare. Najprv sa pozrime na to, ako by sme kompaktne vyjadrili predikčné rovnice v prípade modelu bez vstupov, teda tzv. autonómneho modelu.

### 3.3 Autonómny model

Autonómny systém buď nemá vstupy, ale môžeme aj predpokladať, že všetky akčné zásahy sú nulové, respektíve pevný stavový spätnoväzobný riadiaci zákon je už zahrnutý do dynamiky systému. Ak je daný systém autonómny, potom môžeme vyjadríť odozvu systému na počiatočné podmienky, t. j. podľa Kap. 2 pôjde o spomínanú voľnú odozvu. Jednoducho povedané, takáto sústava môže buť opísaná voľnú odozvu dynamického systému, alebo riadenú odozvu (vynútená odozva) v spätnoväzbovej slučke pomocou pevného stavového zosilnenia, ako môže poskytnúť napríklad LQ riadenie.

Preskúmajme bližšie Rov. (3.6), kde vidíme, že po odstránení vynútenej časti odozvy, teda za predpokladu, že  $\mathbf{B} = 0$ , dostaneme:

$$\begin{aligned}
 k+1 & \quad \mathbf{x}_{k+1} = \mathbf{Ax}_k, \\
 k+2 & \quad \mathbf{x}_{k+2} = \mathbf{Ax}_{k+1} = \mathbf{A}^2\mathbf{x}_k, \\
 k+3 & \quad \mathbf{x}_{k+3} = \mathbf{Ax}_{k+2} = \mathbf{A}^2\mathbf{x}_{k+1} = \mathbf{A}^3\mathbf{x}_k, \\
 \vdots & \quad \vdots \quad \vdots \\
 k+n_p & \quad \mathbf{x}_{k+n_p} = \mathbf{Ax}_{k+n_p-1} = \dots = \mathbf{A}^{n_p-1}\mathbf{x}_{k-1} = \mathbf{A}^{n_p}\mathbf{x}_k.
 \end{aligned} \tag{3.7}$$

Budúce stavy môžeme v ľubovoľnom čase  $k+i$  jednoducho vyjadríť pomocou vzťahu

$$\mathbf{x}_{k+i} = \mathbf{A}^i \mathbf{x}_k, \tag{3.8}$$

kde  $i = 1, 2, \dots, n_p$  a v prípade  $i = 0$  nevykonáme žiadnu predikciu, dostaneme iba nameraný alebo odhadnutý stav  $\mathbf{x}_k$ .

**Príklad 3.1.** Uvažujme autonómny systém, ktorý je vyjadrený pomocou matice dynamiky v nasledujúcom tvare:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -0.5 & 1 \end{bmatrix},$$

a v čase  $k$  máme k dispozícii odhadnutý stav  $\mathbf{x}_0 = [1 \ 0]^T$ . Opíšte predikciu pre vývoj stavu s uvažovaním horizontu  $n_p = 10$  krokov.

Najprv deklarujeme a inicializujeme maticu dynamiky, počiatočný stav a horizont predikcie:

Zdrojový kód 3.1. autonomnysystem.m

```
A=[0 1; -0.5 1]; % Matica dynamiky
x0=[1; 0]; % Pociatočny stav
np=10; % Horizont
```

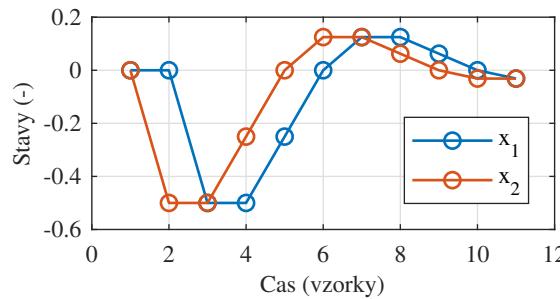
Potom inicializujeme prázdnú premennú  $\mathbf{X}$ . Táto inicializácia je potrebná na to, aby po prvom spustení cyklu nevyskytla chyba kvôli tomu, že program nepozná miesto uloženia premennej  $\mathbf{X}(:, 1)$ . Začíname `for` cyklus od 1 do dĺžky horizontu, a používame Rov. (3.8). V cykle môžeme vypočítať stavy pre  $(k+i)$ -tý krok. Bez súčasného stavu máme 10 stavov.

```
X=[]; % Prazdna premenna
for i=1:np % Cyklus
    X(:,i+1)=A^i*x0; % Model
end % Koniec cyklu
```

Výsledok si vykreslíme do obrázku pomocou:

```
figure(1) % Prazdny obrazok
plot(X') % X nakreslit
xlabel('Cas (vzorky)') % Popis X
ylabel('Stavy (-)') % Popis Y
legend('x_1','x_2') % Legenda
```

Výslednú simuláciu, t. j. predikované stavy autonómneho systému môžeme vidieť na Obr. 3.7.



Obr. 3.7. Predikovaná voľná odozva autonómneho systému.

□

Na zjednodušenie matematického opisu prediktívneho regulátora si teraz zavedme nové premenné. Označme najprv, podobne ako v Kap. 2, počet výstupov ako  $n_y$  a počet vstupov ako  $n_u$ . Matematicky  $n_y$  vyjadri dĺžku stĺpcového vektora výstupov  $\mathbf{y}_k$ , kym  $n_u$  dĺžku stĺpcového vektora vstupov  $\mathbf{u}_k$ . Z Kap. 2 si pamäťame, že vektor výstupov má toľko prvkov, kolko snímačov máme na systéme a vektor vstupov má toľko prvkov, kolko akčných členov používame.

Symbolom  $\vec{\cdot}$  budeme označovať skutočnosť, že ide o premennú, ktorá obsahuje predikované veličiny až do konca horizontu. Zavedme si teda značenie predikovaného stavu  $\vec{\mathbf{x}}_k$ , značenie sekvencie predikovaných výstupov  $\vec{\mathbf{y}}_k$  a značenie budúcich vstupov  $\vec{\mathbf{u}}_k$ . Dolný index  $\cdot_k$  bude stále označovať aktuálny čas, avšak v tomto prípade budeme hovoriť o aktuálnej predikcii ktorá platí v čase  $k$ ; inak povedané znamená, že posledná predikcia bola vykonaná v čase  $k$ . Napríklad  $\vec{\mathbf{x}}_k$  bude obsahovať predikované stavov od  $k+1$  až do  $k+n_p$ , čo je spolu  $n_p$  predikovaných stavov.



Premenné, ktorými budeme vyjadrovať budúcu sekvenciu stavov, výstupov a vstupov, majú nasledujúci tvar:

$$\vec{\mathbf{x}}_k = \begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{x}_{k+2} \\ \vdots \\ \mathbf{x}_{k+i} \\ \vdots \\ \mathbf{x}_{k+n_p-1} \\ \mathbf{x}_{k+n_p} \end{bmatrix}, \quad \vec{\mathbf{y}}_k = \begin{bmatrix} \mathbf{y}_{k+1} \\ \mathbf{y}_{k+2} \\ \vdots \\ \mathbf{y}_{k+i} \\ \vdots \\ \mathbf{y}_{k+n_p-1} \\ \mathbf{y}_{k+n_p} \end{bmatrix}, \quad \vec{\mathbf{u}}_k = \begin{bmatrix} \mathbf{u}_k \\ \mathbf{u}_{k+1} \\ \vdots \\ \mathbf{u}_{k+i} \\ \vdots \\ \mathbf{u}_{k+n_p-2} \\ \mathbf{u}_{k+n_p-1} \end{bmatrix} \quad (3.9)$$

Vidíme, že všetky nové premenné sú stĺpcové vektory, ktoré obsahujú budúce stavy, výstupy a vstupy zoradené pod sebou s rastúcim indexom; bude teda platiť, že  $\vec{\mathbf{x}}_k \in \mathbb{R}^{n_x \cdot n_p \times 1}$ ,  $\vec{\mathbf{y}}_k \in \mathbb{R}^{n_y \cdot n_p \times 1}$  a  $\vec{\mathbf{u}}_k \in \mathbb{R}^{n_u \cdot n_p \times 1}$ . Uvedený matematický zápis znamená, že dané vektory pochádzajú z množiny reálnych čísel  $\mathbb{R}$  príslušného rozmeru, napríklad majú  $n_x \cdot n_p$  riadkov.

Náš regulátor bude založený na predikcii budúcich stavov, a teda by mal prepočítať stavový model niekoľko krokov do budúcnosti, a tak získať všetky stavy až do konca horizontu. Túto úlohu musíme vykonať v každej vzorke počas riadenia v reálnom čase – vždy na základe aktuálne odhadnutého alebo nameraného stavu stavu  $\mathbf{x}_k = \hat{\mathbf{x}}_k$ . Keďže máme k dispozícii nový stav, celý proces predikcie musí byť zopakovaný. Namiesto rekurzívneho dosadzovania do stavových rovníc podľa Rov. (3.7) alebo opakovanej simulácie Rov. (3.8) proces predikcie môžeme upraviť tak, aby sme nemuseli vždy predikciu rekurzívne prepočítať.

Riešením je definícia *predikčnej rovnice*, ktorá bude obsahovať tzv. *predikčné maticy*. Všimnime si, že podľa Rov. (3.8) každý predikovaný stav závisí iba od matice dynamiky  $\mathbf{A}$  a od súčasného stavu  $\mathbf{x}_k$ . V prípade autonómneho systému teda môžeme vyjadriť budúce stavy ako

$$\vec{\mathbf{x}}_k = \begin{bmatrix} \mathbf{A}^1 \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^{n_p-2} \\ \mathbf{A}^{n_p-1} \\ \mathbf{A}^{n_p} \end{bmatrix} \mathbf{x}_k = \mathbf{M} \mathbf{x}_k. \quad (3.10)$$

Matica  $\mathbf{M}$  predstavuje predikčnú maticu dynamiky a je daná ako

$$\mathbf{M} = \begin{bmatrix} \mathbf{A}^1 & & \\ \mathbf{A}^2 & & \\ \vdots & & \\ \mathbf{A}^{n_p-2} & & \\ \mathbf{A}^{n_p-1} & & \\ \mathbf{A}^{n_p} & & \end{bmatrix}. \quad (3.11)$$

Toto znamená, že pre nominálny, respektíve časovo nepremenlivý autonómny systém stačí vypočítať  $\mathbf{M}$  iba raz, a ďalej vynásobením s aktuálnym stavom dostaneme celú sekvenciu predikovaných stavov.

**Príklad 3.2.** Vytvorte v MATLAB-e program na zostavenie predikčnej matice  $\mathbf{M}$  na základe ľubovoľnej matice dynamiky  $\mathbf{A}$  a dĺžky horizontu  $n_p$ . Na otestovanie jeho fukčnosti použite ľubovoľný diskrétny systém, ktorý môžete napríklad vygenerovať pomocou príkazu `drss()`.

Najprv deklarujeme premenné na zvolenú dĺžku horizontu a zvolíme tiež rád systému. Pomocou funkcie `drss()` vygenerujeme náhodný model `sys`, z ktorého môžeme vytiahnuť maticu  $\mathbf{A}$ :

Zdrojový kód 3.2. predikciaautonomna.m

```
nx=2; np=3;           % Rad sys. a horizont
sys=drss(2);          % Nahodny system
A=sys.A               % Matica dynamiky
```

Aby bol náš program použiteľný pre ľubovoľný systém, musíme určiť rád, teda rozmer matice  $\mathbf{A}$ . Matica je štvorcová, stačí preto zavolať príkaz `length()`. Po inicializácii práznej matice  $\mathbf{M}$  môžeme maticu dynamicky napĺňať príslušnými blokovými riadkami nasledujúcim spôsobom:

```
nx=length(A);          % Zistime rozmer A
M=[];                  % Prazdna premenna
for i=1:np              % Cyklus
    M = [M; A^i];        % Predikcna matica
end                     % Koniec cyklu
```

Napríklad pre náhodný systém

```
A =
-0.3428    -0.3140
 0.3140    -0.3428
```

a pre dĺžku horizontu  $n_p = 3$  krokov, dostávame:

```
M =
-0.3428    -0.3140
 0.3140    -0.3428
 0.0189    0.2152
-0.2152    0.0189
 0.0611    -0.0797
 0.0797    0.0611
```

čo môžeme rýchlo prekontrolovať porovnaním s Rov. (3.11). Prvý blokový riadok zodpovedá matici  $\mathbf{A}$ , druhý blok je  $\mathbf{A}^2$  a tretí  $\mathbf{A}^3$ . Toto môžeme overiť tým, že do príkazového riadku napišeme  $\mathbf{A}^2$  a  $\mathbf{A}^3$  a porovnávame so získaným výsledkom.

□

Dynamické rozširovanie premenných väčšina programovacích jazykov nepozná a rozmer matíc musí prekladač<sup>3</sup> vždy vopred poznáť. Rýchlosť výpočtu môžeme dokonca zvýšiť aj v MATLABe, ak premenné vopred deklarujeme do pracovnej pamäte. Maticu **M** môžeme vyplniť nulami, inicializovať pomocné premenné *j* a vložiť výsledok vždy na správne miesto, napríklad nasledovným spôsobom:

```
M=zeros(nx*np, nx);
j=0;
for i=1:nx:nx*np
    j=j+1;
    M(i:i+nx-1, :)=A^j;
end
```

Samozrejme, pri počítaní *i* vo **for** cykle, ako aj pri adresovaní matíc, môžeme využívať rôzne iné kreatívne programátorské triky, alebo napríklad nahradíť **for** cyklus za **while**. Ďalším podobným výpočtom môže byť napríklad aj cyklus

```
for i=1:np;
    M(i*nx-nx+1:i*nx, :)=A^i;
end
```

Ako vidíme, splnenie aj tejto veľmi jednoduchej programátorskej úlohy môže mať rôzne riešenia. Niektoré alternatívy môžu byť totožné, avšak, určité varianty môžu byť rýchlejšie, resp. môžu využívať menej pamäti.

**Príklad 3.3.** Vytvorte samostatnú a univerzálnu funkciu na vygenerovanie predikčnej matice **M** v MATLABe. Nepoužívajte dynamické rozširovanie veľkosti matice a otestujte svoju funkciu pomocou ľubovoľného diskrétneho systému.

Pri tvorbe funkcie potrebujeme definovať jednu výstupnú premennú, a to je matica **M**, a na tvorbu matice potrebujeme dve vstupné premenné, **A** a  $n_p$ . Miesto v pamäti môžeme alokovať napríklad vyplnením matice nulami a potom postupne adresujeme blokové časti **M**, kde dosadíme  $A^i$ , napríklad:

Zdrojový kód 3.3. predikciaM.m

```
function [M] = predikciaM(A, np) % Funkcia na M
nx=length(A); % Zistime rozmer A
M=zeros(np*nx, nx); % Preallocacia
for i=1:np % Cyklus
    M(i*nx-nx+1:i*nx, :)=A^i; % Predikcia
end % Koniec cyklu
```

<sup>3</sup>angl. compiler

Na testovanie funkcie môžeme použiť napríklad maticu

`A=[1 0.3; -0.1 0.5]`

a potom zavolať našu novú funkciu, kde prvý argument je matica **A** a druhý je dĺžka horizontu

`predikciaM(A, 3)`

z čoho dostaneme správny výsledok

```
ans =
1.0000    0.3000
-0.1000    0.5000
0.9700    0.4500
-0.1500    0.2200
0.9250    0.5160
-0.1720    0.0650
```

□

Aká presná je naša predikcia? To samozrejme závisí od toho, ako presne opíše model dynamiku modelovaného systému a ako presne vieme nameráť, respektívne rekonštruovať aktuálny stav. V prípade, že model je dobrou reprezentáciou reálnej dynamiky riadeného systému a stav je dobre odhadnutý, predikcia môže byť veľmi presná. Práve preto musíme zdôrazniť, že predikcia môže byť len taká dobrá, aký dobrý je samotný stavový model a odhad stavu.

**Príklad 3.4.** Uvažujme dlhý a tenký votknutý nosník, kde priamo meriame výchylku jeho voľného konca. Porovnajte experimentálne namerané údaje jeho voľnej odozvy  $y$  (mm) zo súboru *nosnik.mat* s predikciou získanou pomocou funkcie `predikciaM()`, ak diskrétny stavový model dynamiky bol identifikovaný ako

$$\mathbf{A} = \begin{bmatrix} 1.3072 & 0.0185 \\ -9.7355 & 0.6264 \end{bmatrix},$$

počiatočný stav je odhadnutý ako  $\hat{\mathbf{x}}_0 = [-0.1295 \ -59.0551]^T$  a vzorkovanie je  $T_s = 0.005$  s ( $f_s = 200$  Hz).

Najprv nahráme do pracovného prostredia MATLABu namerané dátá o výchylke konca nosníka zo súboru *nosnik.mat*, potom maticu dynamiky a odhadnutý stav na začiatku zo súboru *model.mat*. Deklarujeme vzorkovaciu períodu a vyberieme určitý úsek z dátového súboru na simuláciu:

Zdrojový kód 3.4. predikcianosnik.m

```
load nosnik % Nahraj data
load model % Nahraj A a pociatocny stav
Ts=0.005; % Vzorkovanie
y=y(1000:1500); % Vyber casti merania
```

Vybrali sme práve tento úsek, lebo tu poznáme počiatočný stav. Ak by sme vybrali iný úsek, museli by sme poznať správny počiatočný stav.

V ďalšom využívame funkciu `predikciaM()` na vygenerovanie predikčnej matice stavu s uvažovaním horizontu 500 krokov. Takýto dlhý horizont by bol nezvyčajný v prediktívnom riadení z dôvodu výpočtovej náročnosti, no pre tento konkrétny príklad je to prípustné. V ďalšom kroku využívame samotnú maticu  $\mathbf{M}$  podľa Rov. (3.8) na výpočet vektora  $\vec{x}_k$ :

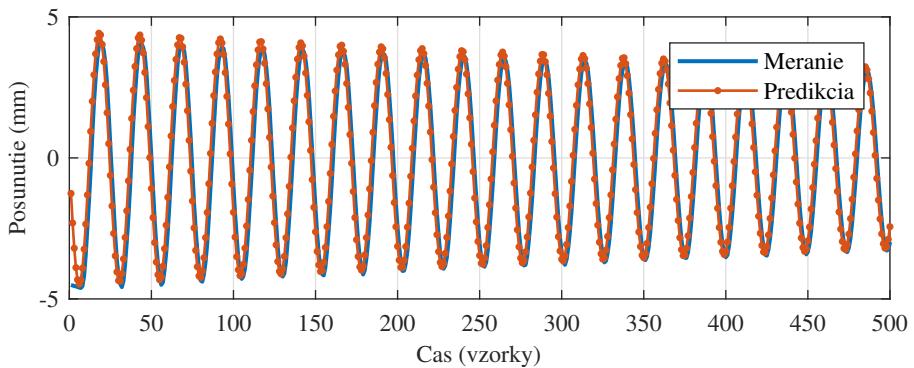
```
M=predikciaM(A, 500); % Matica M, horizont 500
xp=M*x0; % Predikovany stav
```

Na záver si vykreslíme namerané a predikované údaje. Kedže sme merali iba prvý stav, t. j. výchylku, musíme vybrať iba nepárne elementy z vektora  $\vec{x}_k$ , ktorý obsahuje jednotlivé predikované stavy pod sebou. Toto môžeme realizovať tak, že jeho prvky indexujeme ako `xp(1:2:end, :)`, čím získame nepárne riadky (Index `xp(2:2:end, :)` by naopak vybral párne riadky matice alebo vektora):

```
figure(1) % Novy obrazok
plot(y); % Nakresli meranie
hold on % Nechat graf
plot(xp(1:2:end, :), ' .-' ); % Neparne elementy
grid on % Mriezka
xlabel('Cas (vzorky)') % Os X
ylabel('Posunutie (mm)') % Os Y
legend('Meranie', 'Predikcia') % Legenda
```

Na Obr. 3.8 vidíme porovnanie meraných údajov a predikcie stavov. Vidíme, že napriek tomu, že používame iba lineárny časovo invariantný (LTI) stavový model druhého rádu, dynamiku nosníka môžeme predikovať až prekvapivo presne. Plný opis mechaniky dynamického správania sa nosníka, napríklad pomocou metódy konečných prvkov (MKP) by bol samozrejme ešte presnejší, ale pre účely riadenia často dobre fungujú aj takéto drastické zjednodušenia.

□



Obr. 3.8. Porovnanie predikcie výchylky voľného konca votknutého nosníka s jej experimentálnym meraním.

### 3.4 Predikcia vynútenej odozvy

V predchádzajúcej podkapitole sme si vytvorili matematický základ na predikciu stavov autonómneho systému, alebo voľnej odozvy systému v otvorenej slučke. Bez vplyvu akčných členov by sme nemohli hovoriť o riadení, preto musíme ďalej vyšetriť aj vplyv vynútenej časti odozvy na systém.

Pozrime sa ešte raz na rekurzívne dosadenie stavov v Rov. (3.6). Prvú časť, kde násobíme  $\mathbf{A}^i$  so stavom sme vyriešili pomocou predikčnej matice  $\mathbf{M}$ . Zostali iba výrazy, kde vystupujú matice násobené so vstupmi  $\mathbf{u}_i$ ,  $i = 1, 2, \dots, n_p - 1$ . Sekvenčiu vstupov sme už v predošej diskusii zjednotili do novej premennej  $\vec{\mathbf{u}}_k$ .



Predikčné rovnice v kompaktnom tvare môžeme napísat ako

$$\vec{\mathbf{x}}_k = \mathbf{M}\mathbf{x}_k + \mathbf{N}\vec{\mathbf{u}}_k, \quad (3.12)$$

kde jediná neznáma bude matica  $\mathbf{N}$ . Túto maticu budeme v ďalšej diskusii nazývať predikčnou maticou vynútenej odozvy.

Aby sme zistili aký tvar má matica  $\mathbf{N}$ , musíme sa pozrieť na Rov. (3.6) a zistíť akými výrazmi násobíme  $\vec{\mathbf{u}}_k$ . Vidíme, že pri prvom riadku máme iba stavovú rovnicu, teda násobíme  $\mathbf{B}$  a  $\mathbf{u}_k$ . Ostatné bloky  $\vec{\mathbf{u}}_k$  sú samozrejme stále prítomné a nemôžeme ich ignorovať, inak maticové násobenie nemôžeme vykonať. Násobíme preto nulou, ale keďže bloky  $\vec{\mathbf{u}}_k$  sú vo všeobecnosti neskalárne, násobíme maticou nul, ktorú si označíme ako  $\mathbf{0}$  (namiesto jednoduchej 0). Pre maticu nul teda platí  $\mathbf{0} = 0^{n_x \times n_u}$ , čo znamená, že bude obsahovať iba nuly a mať  $n_x$  riadkov a  $n_u$  stĺpcov.

V ďalšom riadku už  $\vec{u}_k$  násobíme s ďalším členom  $\mathbf{AB}$ , zostáva aj  $\mathbf{B}$ , a zvyšok násobíme zase nulami. V tretom riadku pribúda nový člen  $\mathbf{A}^2\mathbf{B}$ , zostáva  $\mathbf{AB}$ ,  $\mathbf{B}$  a zvyšok vyplníme nulami. Vidíme, že v každom  $i$ -tom riadku pridáme nový člen  $\mathbf{A}^{i-1}\mathbf{B}$ .

Predikčná matica vynútenej odozvy  $\mathbf{N}$  je daná ako

$$\mathbf{N} = \begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{A}^{n_p-3}\mathbf{B} & \mathbf{A}^{n_p-4}\mathbf{B} & \mathbf{A}^{n_p-5}\mathbf{B} & \cdots & \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}^{n_p-2}\mathbf{B} & \mathbf{A}^{n_p-3}\mathbf{B} & \mathbf{A}^{n_p-4}\mathbf{B} & \cdots & \mathbf{AB} & \mathbf{B} & \mathbf{0} \\ \mathbf{A}^{n_p-1}\mathbf{B} & \mathbf{A}^{n_p-2}\mathbf{B} & \mathbf{A}^{n_p-3}\mathbf{B} & \cdots & \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} \end{bmatrix} \quad (3.13)$$



kde  $\mathbf{N} \in \mathbb{R}^{n_x \cdot n_p \times n_u \cdot n_p}$ , teda bude obsahovať reálne čísla a pozostávať z  $n_x \cdot n_p$  riadkov a  $n_u \cdot n_p$  stĺpcov.

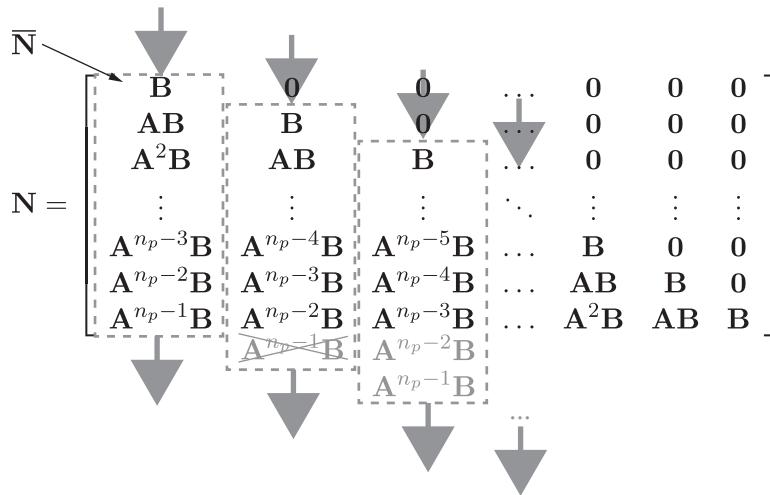
Ak máme nominálny časovo nepremenlivý model, maticu  $\mathbf{N}$  stačí prepočítať iba raz a platí pre každú možnú budúcu sekvenciu vstupov. Na praktický spôsob tvorby matice existuje toľko spôsobov kolko programátorov, samozrejme vždy musíme rátať aj s efektívnosťou programu. Jeden spôsob tvorby matice  $\mathbf{N}$  je graficky znázornený na Obr. 3.9. Najprv vytvoríme prvý stĺpec matice, nazvime ho  $\bar{\mathbf{N}}$ , kde  $i$ -tý blokový element je  $\mathbf{A}^{i-1}\mathbf{B}$  pre  $i = 1, 2, \dots, n_p$ , a teda

$$\bar{\mathbf{N}} = \left[ \mathbf{B} \quad \mathbf{AB} \quad \mathbf{A}^2\mathbf{B} \quad \dots \quad \mathbf{A}^{n_p-1}\mathbf{B} \quad \mathbf{A}^{n_p-1}\mathbf{B} \right]^T. \quad (3.14)$$

**Príklad 3.5.** Naprogramujte funkciu na vytvorenie predikčnej matice  $\mathbf{N}$  v MATLABe a otestujte jej funkčnosť pre horizont  $n_p = 3$ , rám systému  $n_x = 2$  a počet vstupov  $n_u = 2$ .

Výstupom funkcie bude predikčná matica vynútenej odozvy, pričom na vstupe potrebujeme poznať z modelu  $\mathbf{A}$  a  $\mathbf{B}$ , a žiadanú dĺžku horizontu  $n_p$ .

Aby bol program univerzálne použiteľný, okrem počtu stavov potrebujeme už vedieť aj počet vstupov. V MATLABe to môžeme vykonať jednoriadkovým príkazom `size()`. Stačí nám zistieť iba rozmer matice  $\mathbf{B}$ , keďže počet jej riadkov sa rovná počtu stavov systému a počet stĺpcov zodpovedá počtu vstupov do systému. Vyplňme maticu  $\mathbf{N}$  nulami a vytvorime aj prázdnú pomocnú maticu  $\bar{\mathbf{N}}$  ( $\mathbf{NN}$ ) na základe nasledujúcich príkazov:



Obr. 3.9. Vytvorenie predikčnej matice  $\bar{N}$  posunutím blokových stĺpcov smerom dole.

Zdrojový kód 3.5. predikciaN.m

```
function [N] = predikciaN(A, B, np) % Funkcia na N
[nx nu]=size(B); % Rozmer sustavy
N=zeros(nx*np, nu*np); % Vyplniť N nulami
NN=zeros(nx*np, nu); % Vyplniť NN nulami
```

Vyplňme najprv pomocnú maticu  $\bar{N}$  podľa prvého blokového stĺpca  $\bar{N}$

```
for i=1:np % Prvy stlpiec matice N
NN(i*nx-nx+1:i*nx, :)=[A^(i-1)*B];
end
```

podobne ako sme to robili pri matici  $M$ .

Teraz bloky  $N$  vyplňme rovnakými blokmi  $\bar{N}$  tak, že vlastne posúvame blokovou maticu  $\bar{N}$  smerom dole s krokom daným počtom stavov a súčasne v každej iterácii posunieme operáciu aj o jeden blok šírky počtu vstupov vpravo:

```
for i=1:np % Cela matica N posunom dole
N(i*nx-nx+1:end, i*nu-nu+1:i*nu)=NN(1:np*nx-(i-1)*nx
, :);
end
```

Na otestovanie funkčnosti novej funkcie môžeme použiť príkaz `drss()`, ktorý vytvorí náhodný stabilný diskrétny systém. Na rozdiel od predošlého príkladu,

teraz okrem matice dynamiky už potrebujeme aj maticu vstupov  $\mathbf{B}$ . Kedže príkaz `drss()` vždy tvorí SISO systém,  $\mathbf{B}$  môžeme vytvoriť voľne:

```
sys=drss(2); A=sys.A; B=[sys.B sys.B./2];
```

a potom s horizontom predikcie  $n_p = 3$  môžeme zavolať novú funkciu `predikciaN` ako

```
predikciaN(A,B,3)
```

Napríklad pri náhodne vygenerovanom systéme

```
A =
-0.2271 -0.3599
-0.3599 0.1386
```

a

```
B =
1.5326 0.7663
-0.7697 -0.3848
```

dostaneme maticu  $\mathbf{N}$  v tvare

```
N =
```

```
1.53 0.77 0 0 0 0
-0.77 -0.38 0 0 0 0
-0.07 -0.04 1.53 0.77 0 0
-0.66 -0.33 -0.77 -0.38 0 0
0.25 0.13 -0.07 -0.04 1.53 0.77
-0.07 -0.03 -0.66 -0.33 -0.77 -0.38
```

□

Ako vidíme, aj pri vytváraní predikčných matíc máme rôzne algoritmicke možnosti. Matica  $\mathbf{N}$  má štruktúru tzv. Toeplitzovej matice. Ak má matica elementy (bloky) v tvare

$$\mathbf{X} = \begin{bmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{bmatrix}, \quad (3.15)$$

potom ju nazývame Toeplitzova matica. Naša matica  $\mathbf{N}$  je potom nesymetrická, bloková dolná trojuholníková Toeplitzova matica. Príkaz `toeplitz()` v MAT-

LABe žiaľ nemôžeme používať<sup>4</sup>, pretože slúži iba na vytvorenie Toeplitzovej matice, ktorá nie je bloková (avšak vytvorená matica môže byť nesymetrická). Alternatívne však môžeme efektívne využiť napríklad príkaz `convmtx()` na vytvorenie tzv. konvolučnej matice.

Využime teraz naše nové funkcie `predikciaM()` a `predikciaN()` na to, aby sme predikovali vývoj stavov systému a porovnali získané výsledky s jednoduchou lineárной simuláciou.

**Príklad 3.6.** Aké budú predikované stavy systému z Pr. 3.1, ak použijeme tú istú maticu dynamiky  $\mathbf{A}$ , počiatočný stav  $\mathbf{x}_0 = \mathbf{x}_0$ , horizont  $n_p$ , a vplyv akčného člena je vyjadrený pomocou matice  $\mathbf{B} = [0 \ 1]^T$ . Uvažujme jednotkový skok na vstupe systému a porovnajme výsledok s lineárной simuláciou.

Najprv si zadefinujeme model, počiatočný stav, horizont predikcie a vstupný signál pomocou:

Zdrojový kód 3.6. kontrolapredikcie.m

```
A=[0 1; -0.5 1]; % Matica dynamiky
B=[0 1]'; % Matica vstupu
x0=[1; 0]; % Počiatocny stav
np=10; % Horizont
u=ones(np,1); % Jednotkovy skok
```

kde posledný riadok vytvorí vektor jednotiek, ktorý reprezentuje jednotkový skok na vstupe.

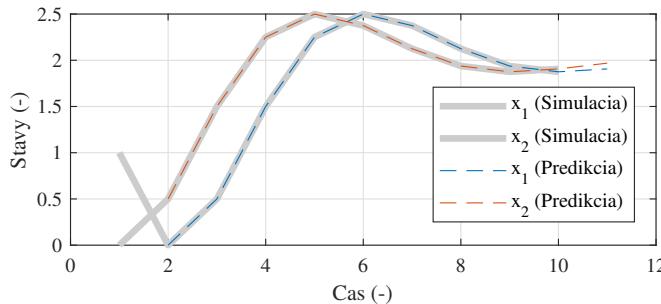
Najprv vypočítame predikčné matice pomocou nových funkcií a daného modelu. Budúce predikované stavy môžeme vypočítať podľa predikčnej rovnice (3.12):

```
M=predikciaM(A,np); % Matica M
N=predikciaN(A,B,np); % Matica N
xpr=M*x0+N*u; % Predikcna rovnica
```

Na kontrolu vytvoríme lineárnu simuláciu pomocou funkcie `lsim()`. Najprv si definujeme model v tvare objektu na uloženie stavového modelu `ss()`, kde prázdne hranaté zátvorky vyjadrujú absenciu matice  $\mathbf{C}$  a  $\mathbf{D}$ . Posledný vstupný parameter funkcie je vzorkovanie, ktoré môžeme zvoliť ako 1 s. Použijeme aj vektor vzoriek  $t$  a potom zavoláme funkciu na lineárnu simuláciu

```
sys=ss(A,B,[],[],1); % Stavovy objekt
t=1:1:np; % Casovy vektor
[y,t,Xsim]=lsim(sys,u',t,x0); % Linearna simulacia
```

<sup>4</sup>Výnimkou je stavový systém prvého rádu s  $n_x = 1$ .



Obr. 3.10. Porovnanie výsledkov predikcie s lineárной simuláciou.

Maticu  $\vec{x}_k$  musíme ešte prepísať do dvojriadkového formátu, kde každý riadok predstavuje jeden stav. Na správne porovnávanie výsledkov simulácie a predikcie pridávame aj neplatný prvý element pomocou NaN<sup>5</sup>. Môžeme na to používať zápis z Pr. 3.4:

```
Xpr=[ [NaN; xpr (1:2:end, :)] , [NaN; xpr (2:2:end, :)] ] ;
```

Potom výsledok môžeme porovnávať graficky pomocou príkazov 3.4:

```
figure(1) % Nový obrazok
plot(Xsim,'Color',[0.8 0.8 0.8], 'LineWidth', 3)
hold on % Neprekresliť graf
ax = gca; % Objekt označujuci os obrazku
ax.ColorOrderIndex = 1; % Farebny index od zaciatku
plot(Xpr,'--') % Nakresliť predikciu
xlabel('Cas (-)') % Os x
ylabel('Stavy (-)') % Os y, legenda
legend('x_1 (Simulacia)', 'x_2 (Simulacia)', 'x_1 (Predikcia)', 'x_2 (Predikcia)')
grid on % Mriezka
```

Na Obr. 3.10 vidíme, že naša predikcia je totožná s lineárной diskrétnou simuláciou. Rov. (3.12) pomocou predikčných matíc len zjednoduší proces lineárnej simulácie, kde predpokladáme, že poznáme počiatočný stav  $\mathbf{x}_k$  v čase  $k$  a sekvenčiu budúcich vstupov  $\vec{u}_k$ . Celý proces predikcie a lineárnej simulácie sme tak prepísali na jedinú algebraickú rovnicu.

Všimnime si malé rozdiely medzi simuláciou a predikciou. Pri predikcií nerátame s aktuálne dostupným stavom v čase  $k$ ,  $\mathbf{x}_k$ , preto predikciu začíname iba

<sup>5</sup>angl. Not a Number (NaN)

pre  $\mathbf{x}_{k+1}$ . Predikcia trvá potom presne 10 krokov, a preto je dlhšia ako samotná simulácia a končí v kroku  $\mathbf{x}_{n_p}$ . Poznamenajme, že na zjednodušenie vykreslenia výsledkov, sme neuvádzali veličiny na x-ovej osi, a preto MATLAB indexoval  $k$ -tú – z nášho pohľadu nultú – vzorku od jedna. V skutočnosti je graf posunutý o vzorku, a teda koniec predikcie by pripadol na index 10.

□

### 3.5 Predikčné matice

Výpočet predikčných matíc samozrejme môžeme vykonať aj naraz, môžeme sformulovať funkciu `predikcia()` ktorá vráti maticu  $\mathbf{M}$  a  $\mathbf{N}$  na výstupe a na vstupe potrebuje model  $(\mathbf{A}, \mathbf{B})$  a žiadanú dĺžku horizontu  $n_p$ :

Zdrojový kód 3.7. `predikcia.m`

```
function [M,N] = predikcia(A,B,np);
M=predikciaM(A,np); % Matica M
N=predikciaN(A,B,np); % Matica N
```

V ďalšom príklade využívame novú funkciu na vytvorenie predikčných matíc.

**Príklad 3.7.** Uvažujme zjednodušenú letovú dynamiku dopravného lietadla Boeing 777 v jednej osi, ktorá vyjadruje uhol stúpania<sup>6</sup> pozdĺž y-ovej osi  $\theta$  ako reakciu na zmenu uhla výškového kormidla  $\delta$ , na základe Obr. 3.11. Spojitý model môžeme vyjadriť v tvare [108]

$$\mathbf{A}_c = \begin{bmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{bmatrix},$$

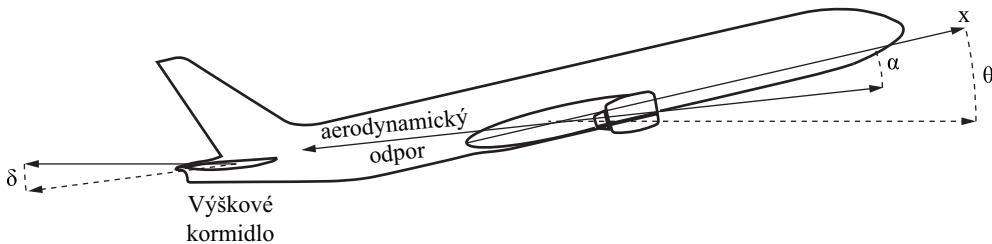
$$\mathbf{B}_c = \begin{bmatrix} 0.232 & 0.0203 & 0 \end{bmatrix},$$

kde stavový vektor obsahuje veličiny  $\mathbf{x} = [\alpha \ \dot{\theta} \ \theta]^T$  a  $\alpha$  je takzvaný uhol nábehu<sup>7</sup>. Ďalej predpokladajme vzorkovanie  $T_s = 0.5$  s, predikčný horizont  $n_p = 5$  krokov a nulové počiatočné podmienky.

Úlohou je zobraziť predikciu stúpania lietadla v každom časovom okamihu po skokovej zmeny na výškovom kormidle  $\delta = 0.1$  rad. Predpokladáme nepresnosť predikčného modelu  $\beta = 0.0005$ , ktorá je sčítaná s každým elementom spojitého modelu.

<sup>6</sup>angl. pitch

<sup>7</sup>angl. angle of attack



Obr. 3.11. Lietadlo Boeing 777-300ER.

Začnime deklaráciou vzorkovania, amplitúdy skokovej zmeny na vstupe, počia-  
točného stavu a horizontu predikcie. Ďalej, keďže vyšetrujeme odozvu na skokovú  
zmenu, budúca sekvencia vstupov počas horizontu  $\vec{u}_k$  je vektor s identickými  
elementami amplitúdy zmeny:

Zdrojový kód 3.8. lietadlo.m

```

Ts=0.5; % [s] Vzorkovanie
u=0.1; % [rad] Vstup
x=[0 0 0]'; % Pociatocny stav
np=5; % Horizont
upr=u*ones(np,1); % Buduca sekvencia vstupov

```

Ďalej deklarujme maticu dynamiky a vstupu spojitého modelu, ktorú budeme  
v tejto simulácii považovať za správnu. Tretia stavová premenná je klonenie lie-  
tadla, preto  $\mathbf{C} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ . Z matíc vytvoríme objekt stavového modelu pomocou  
príkazu `ss()`:

```

Ac = [-0.313 56.7 0; -0.0139 -0.426 0; 0 56.7 0];
Bc = [0.232; 0.0203; 0]; % Matica vstupu
C = [0 0 1]; % Matica vystupu
b777 = ss(Ac,Bc,C,[]); % "Skutocna" dynamika

```

Z pôvodného modelu urobíme teraz zmenenú dynamickú reprezentáciu s malou  
mierou nepresnosti. Na to vytvoríme nový objekt `model` s tým, že prirátame  
skalárne číslo  $\beta$  ku každému prvku, potom model zdiskretizujeme a extrahujeme  
matice  $\mathbf{A}$  a  $\mathbf{B}$ , čím dostaneme diskrétny model:

```

beta=0.0005; % Neistota modelu
model=ss(Ac+ones(3)*beta,Bc+ones(3,1)*beta,C,[]); % Diskretizacia
model=c2d(model,Ts); % Matice A a B disk. mod.
A=model.a; B=model.b;

```

Najprv si zobrazíme skokovú odozvu správneho systému, čo budeme uvažovať za „skutočnú“ odozvu. Potom môžeme vykresliť aj odozvu samotného modelu. Vidíme, že ide o nestabilný systém, kde model nereprezentuje skutočnú dynamiku úplne presne. Prvých pár krovov je odozva modelu blízka skutočnej odozve, no potom diverguje. Ak by sme teda urobili predikciu s nulovým počiatočným stavom a horizontom až dokonca simulácie, dostali by sme presne túto odozvu:

```

step(b777*u)                                % "Skutocna" odozva
hold on; grid on                          % Podrz graf, mriezka
step(model*u)                                % Odozva modelu
axis([0 10 0 0.4])                           % Nastavenie osi
xlabel('Cas')                                % Os x
ylabel('Uhol (rad)')                         % Os y

```

Následne si prepočítame predikčné matice a zvolíme dĺžku simulácie. V slučke vždy vypočítame predikciu stavu a zobrazíme ju na grafe pomocou symbolov krúžku, ďalej zobrazíme aj stav  $x_k$  pomocou symbolu kríza, ako to je znázornené na Obr. 3.12. Stav prepočítame na základe „skutočného“ modelu, tým predpokladáme, že napriek nesprávnemu predikčnému modelu máme dokonalé meranie a odhad stavu. Po každej iterácii simulácie zopakujeme proces a čakáme na stlačenie ľubovoľnej klávesy:

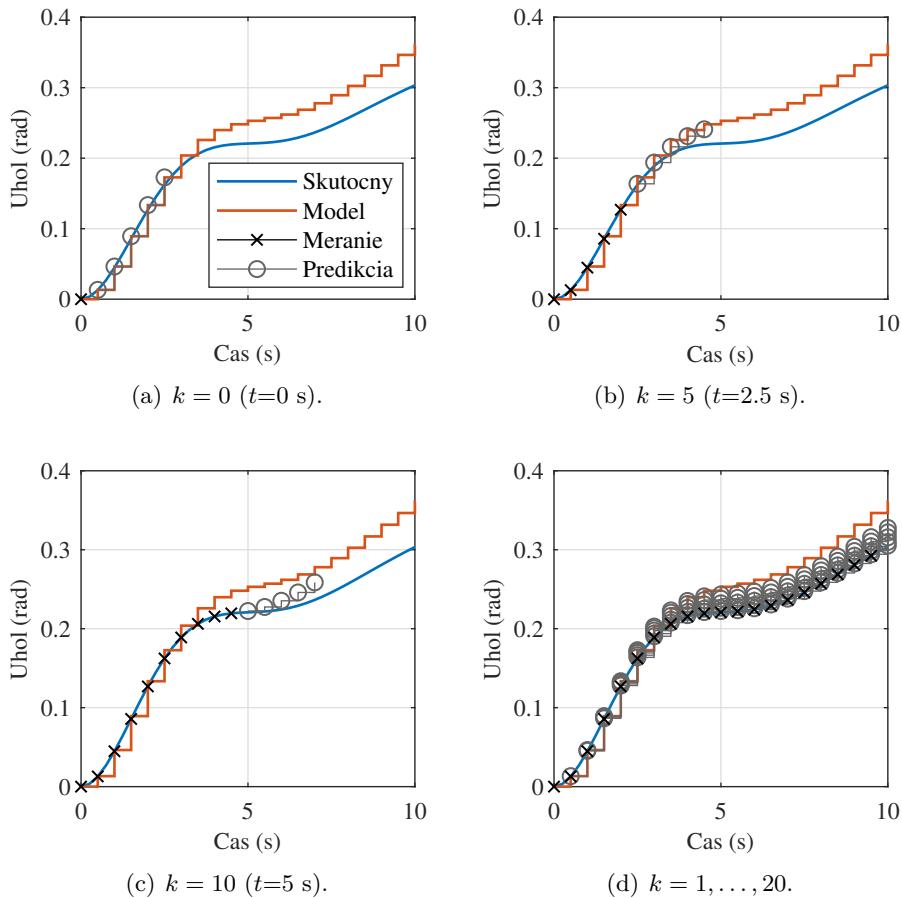
```

[M, N]=predikcia(A, B, np);           % Predikcne matice
dlzka=20;                                % Dlzka simulacie
for i=1:dlzka                         % For cyklus
    xpr=M*x+N*upr;                      % Predikcia
    stairs((i-1)*Ts, x(3), 'kx-')    % "Skutocny" stav
    stairs([i:i+np-1]*Ts, xpr(3:3:end,:)', 'o-', 'Color', (
        ones(3,1)*dlzka/(i+1))/dlzka)
    [Y, T, X]=lsim(b777, [u u], [(i-1) i]*Ts, x);
    x=X(end,:);                         % Dalsi "Skutocny" stav
    pause                                % Cakaj
end

```

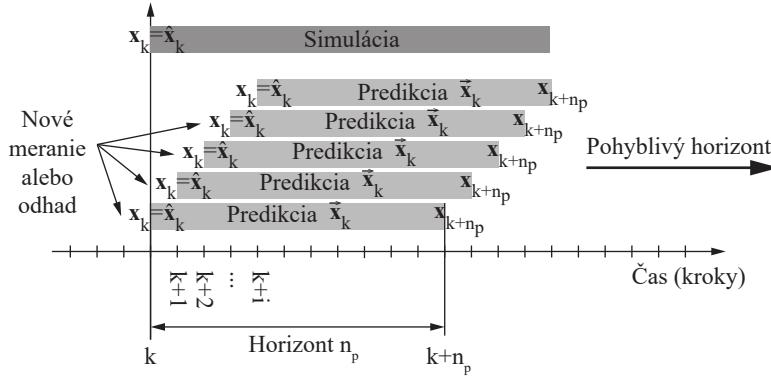
Obr. 3.12 znázorňuje proces predikcie na základe aktuálneho stavu. Vidíme, že kvôli nesprávnemu modelu simulácia značne diverguje od skutočnosti. Samozrejme, predikcia v čase  $k = 0$  na Obr. 3.12(a) presne sleduje túto trajektóriu, ale hlavne v prvých krovoch predpovedá skutočnú trajektóriu dostatočne korektnie.

Základná myšlienka predikcie je opakovane predikovať budúce stavy na základe novej informácie o stave. V každom časovom vzorke máme dostupný aktuálny stav, podľa ktorého posunieme horizont ďalej; predikcia na krátkom horizonte tak stále zostáva správna (Obr. 3.12(b) a Obr. 3.12(c)).



Obr. 3.12. Predikcia trajektórie odozvy Boeing 777 (o) a skutočné stavy (x).

□



Obr. 3.13. Idea pohyblivého horizontu predikcie.

### 3.6 Koncept pohyblivého horizontu

Koncept *pohyblivého horizontu* v prediktívnom riadení znamená, že predikčný horizont sa vplyvom nového meraného alebo odhadnutého stavu posúva každým vzorkovacím intervalom dopredu. Hoci proces predikcie je vlastne iba simulácia, práve vďaka dostupnosti nového stavu môžeme aktualizovať našu predikciu a posúvať tak predikčný horizont dopredu (Obr. 3.13).

Túto myšlienku ilustruje aj Pr. 3.7, kde simuláciu porovnávame s pohyblivým predikčným horizontom. Ako vidíme na Obr. 3.12, simulácia vplyvom nepresného modelu diverguje od skutočného priebehu. Na rozdiel od simulácie, predikčný horizont vždy vychádza z aktuálne nameraného, resp. odhadnutého stavu. Horizont sa posúva v čase dopredu tak, ako je to znázornené na Obr. 3.12(a)–3.12(d).

### 3.7 Predikcia daného stavu v budúcnosti

Všimnime si, že nepotrebujeme vypočítať celú postupnosť  $\vec{x}_k$  na to, aby sme predikovali iba jeden ľubovoľný stavový vektor  $\mathbf{x}_{k+i}$  ktorý je  $i$  krokov posunutý oproti súčasnosti. Kedže v ďalšej diskusii budeme potrebovať vyjadriť tieto individuálne stavy, preto si teraz zavedieme pojem  $i$ -tého blokového riadku predikčných matíc.

Ak blokové riadky matíc vyjadríme podľa

$$\mathbf{M}_i = \mathbf{A}^i, \quad (3.16)$$

$$\mathbf{N}_i = \left[ \mathbf{A}^{i-1}\mathbf{B} \ \mathbf{A}^{i-2}\mathbf{B} \ \mathbf{A}^{i-3}\mathbf{B} \ \dots \ \mathbf{A}^2\mathbf{B} \ \mathbf{AB} \ \mathbf{B} \right], \quad (3.17)$$

potom ľubovoľný stav v budúcnosti vyjadríme pomocou vzťahu

$$\mathbf{x}_{k+i} = \mathbf{M}_i \mathbf{x}_k + \mathbf{N}_i \vec{\mathbf{u}}_k = \mathbf{A}^i \mathbf{x}_k + \mathbf{N}_i \vec{\mathbf{u}}_k, \quad (3.18)$$

kde  $i = 1, 2, \dots, n_p$  a  $\vec{\mathbf{u}}_k$  predstavuje celú postupnosť vstupov aktívnych počas trvania horizontu.

Pre nasledujúcu diskusiu bude vhodné zaviesť koncept aj posledného blokového riadku predikčných matíc podľa

$$\mathbf{M}_{n_p} = \mathbf{A}^{n_p}, \quad (3.19)$$

$$\mathbf{N}_{n_p} = \begin{bmatrix} \mathbf{A}^{n_p-1}\mathbf{B} & \mathbf{A}^{n_p-2}\mathbf{B} & \mathbf{A}^{n_p-3}\mathbf{B} & \dots & \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} \end{bmatrix}, \quad (3.20)$$

kde teda  $\mathbf{M}_{n_p}$  a  $\mathbf{N}_{n_p}$  predstavujú posledný blokový riadok príslušných predikčných matíc, pomocou ktorých pre posledný stav  $\mathbf{x}_{n_p}$  na konci horizontu platí:

$$\mathbf{x}_{n_p} = \mathbf{M}_{n_p}\mathbf{x}_k + \mathbf{N}_{n_p}\vec{\mathbf{u}}_k = \mathbf{A}^{n_p}\mathbf{x}_k + \mathbf{N}_{n_p}\vec{\mathbf{u}}_k. \quad (3.21)$$

Ak teda chceme vybrať  $i$ -tý blok matice s počtom riadkov  $j$ , začiatok bloku sa nachádza v riadku  $(i-1)j+1$  a koniec v riadku  $ij$ . V MATLABe môžeme vybrať  $i$ -tý blok  $\bar{\mathbf{X}}$  ľubovoľnej matice  $\mathbf{X}$  s počtom riadkov  $j$  v bloku napríklad pomocou nasledujúceho príkazu:

`Xbar=X( (i-1)*j+1:i*j, :)`

**Príklad 3.8.** Uvažujme diskretizovaný model dvojitého integrátora so vzorkovacím intervalom  $T_s = 1$  s. Vyjadrite si ľubovoľný  $i$ -tý predikovaný stav, ak uvažujeme horizont predikcie  $n_p$ , počiatočný stav  $\mathbf{x}_k = \mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$  a jednotkový skok ako vstup systému.

Dobre známy spojity model dvojitého integrátora má tvar

$$\mathbf{A}_c = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{a} \quad \mathbf{B}_c = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

Na začiatku zdrojového kódu teda zadáme spojity model integrátora, počiatočný stav, vzorkovanie a horizont predikcie:

Zdrojový kód 3.9. blokoveriadky.m

```
Ac=[0 1; 0 0]; % Matica dynamiky
Bc=[0 1]'; % Matica vstupov
C=[1 0]; % Matica vystupov
x0=[1 1]'; Ts=1; np=10; % Parametre
```

Ďalej použijeme príkaz `c2dm()` na diskretizáciu modelu a vygenerujeme predikčné matice:

```
[A,B,C,D] = c2dm(Ac,Bc,C,0,Ts); % Diskretizacia
[M,N] = predikcia(A,B,np); % Predikcne matice
```

Sekvencia vstupov počas trvania predikčného horizontu je známa a môžeme ju vyjadriť pomocou jednotkového vektora. Podobne ako v predchádzajúcich príkladoch, aj teraz vypočítame celý predikovaný stav  $\vec{x}_k$  na základe Rov. (3.12), ktorý si potom graficky zobrazíme:

```
upr=ones(np,1); % Vstup (skok)
xpr=M*x0+N*upr % Predikovane stavov
Xpr=[xpr(1:2:end,:),xpr(2:2:end,:)]; % Vektor stavov
plot(Xpr,'.-'); % Vykresli predikciu
hold on; % Podrzat graf
```

Rov. (3.18) môžeme vyjadriť v MATLABe tak, že správne indexujeme blokové riadky predikčných matíc. Pre  $i$ -tý predikovaný stav  $\mathbf{x}_{k+i}$  (xi) teda platí:

```
i=5; % Index stavu
nx=length(A); % Počet stavov
xi=M((i-1)*nx+1:i*nx,:)*x0 +N((i-1)*nx+1:i*nx,:)*upr
```

Pre kontrolu výsledkov môžeme zobraziť  $\mathbf{x}_{k+i}$  pomocou:

```
plot(i,xi(1),'ko') % i-ty stav x1
plot(i,xi(2),'ko') % i-ty stav x2
grid on % Mriezka
xlabel('Cas (s)') % Os x
ylabel('Stavy (-)') % Os y
legend('x_1','x_2') % Legenda
```

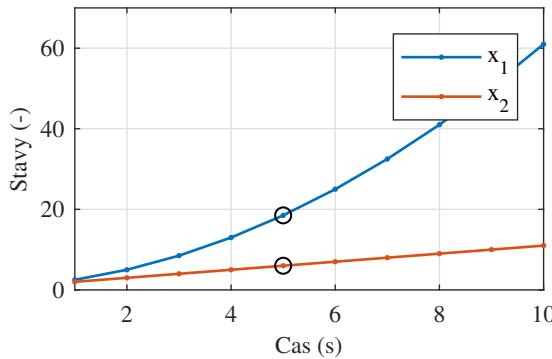
a na Obr. 3.14 vidíme, že stav  $\mathbf{x}_{k+i}$  podľa (3.18) je súčasťou predikovanej postupnosti stavov,  $\vec{x}_k$ .

□

## 3.8 Predikcia výstupov

V predchádzajúcej diskusii sme sa vždy sústredili na stavov systému. Ak by sme chceli vyjadriť sekvenču predikovaných výstupov,  $\vec{y}_k$ , alebo budúci výstup v ľubovoľnom čase,  $\mathbf{y}_{k+i}$ , máme veľmi jednoduchú úlohu: stačí totiž stavov uložené vektorovo pod sebou násobiť maticou  $\mathbf{L}$  podľa

$$\vec{y}_k = \mathbf{L} \vec{x}_k, \quad (3.22)$$



Obr. 3.14. Lubovoľný predikovaný stav (krúžky) dvojitého integrátora získaný pomocou príslušných blokových riadkov predikčných matíc.

kde  $\mathbf{L} \in \mathbb{R}^{n_y \cdot n_p \times n_x \cdot n_p}$  obsahuje matice výstupu na hlavnej blokovej diagonále:

$$\mathbf{L} = \begin{bmatrix} \mathbf{C} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{C} \end{bmatrix}, \quad (3.23)$$

kde  $\mathbf{0} = 0^{n_y \times n_x}$  predstavuje blok núl. Pre výstupy v ľubovoľnom čase potom platí

$$\mathbf{y}_{k+i} = \mathbf{C}\mathbf{x}_{k+i} = \mathbf{C}\mathbf{A}^i\mathbf{x}_k + \mathbf{C}\mathbf{N}_i\mathbf{u}_k, \quad (3.24)$$

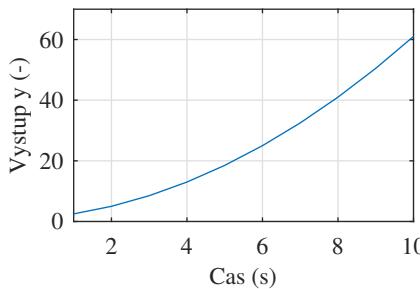
kde  $i = 1, 2, \dots, n_p$ .

Na tvorbu tejto transformačnej matice  $\mathbf{L}$  môžeme vytvoriť jednoduchú funkciu, ktorá len vyplní  $n_x \cdot n_p \times n_y$  veľkú maticu nulami a potom dosadí bloky  $\mathbf{C}^T$ :

Zdrojový kód 3.10. transformaciaL.m

```
function [L] = transformaciaL(C,np) % Funkcia na L
[ny nx]=size(C); % Rozmer sustavy
L=zeros(ny*np,np*np); % Prealokacia
for i=1:np % Cyklus
    L(i*ny-ny+1:i*ny,i*nx-nx+1:i*nx)=C;
end % Koniec cyklu
```

**Príklad 3.9.** Uvažujme dvojitý integrátor z predošlého Pr. 3.8. Pokračujme príklad so zobrazením vektora  $\vec{y}_k$ .



Obr. 3.15. Násobením transformačnou maticou  $\mathbf{L}$  získame predikované výstupy.

Zdrojový kód príkladu je na začiatku identický s príkladom Pr. 3.8. Potom si vytvoríme maticu  $\mathbf{L}$  pomocou novozavedeného príkazu `transformaciaL()` a graficky zobrazíme výsledky.

Zdrojový kód 3.11. predikciavystupu.m

```

L=transformaciaL(C,np) % Matica L
ypr=L*xpr; % Vektor ypr
figure(2) % Novy obrazok
plot(ypr) % i-ty stav x2
grid on % Mriezka
xlabel('Cas (s)') % Os x
ylabel('Vystup y (-)') % Os y

```

Z Obr. 3.15 je zrejmé, že pomocou matice  $\mathbf{L}$  môžeme prepočítať vektor výstupov  $\vec{y}_k$  z predikovaných stavov  $\vec{x}_k$ . V tomto prípade je matica výstupov  $\mathbf{C} = [1 \ 0]$ , a preto sekvenčia je výstupov identická s priebehom prvého predikovaného stavu.

□

Treba si uvedomiť, že na tvorbu riadiaceho zákona je užitočná skôr predikcia stavu  $\vec{x}_k$  ako predikcia výstupu  $\vec{y}_k$ . Samozrejme výstup priamo závisí od stavov a principiálne ide o vyjadrenie pozorovaných stavov. Inými slovami, výstupy sú fyzikálne merania, ktoré nemusia nutne poskytovať plný obraz o dianí vo vnútri systému. Práve preto sa v diskusii v ďalších kapitolách budeme sústredovať na stavu a predpokladať, že platí  $\mathbf{y}_k = \mathbf{C}\mathbf{x}_k$ . Odhad stavu z výstupu je v každom prípade úlohou pozorovateľa stavu, napríklad Kalmanovho filtra a náš riadiaci zákon – podobne ako pri LQ – budeme navrhovať na základe stavovej spätej väzby.

### Odkiaľ poznáme vstupy?

Načo však môžeme využiť odvodenú predikčnú rovnicu ak hľadáme práve optimálnu budúcu sekvenciu vstupov  $\vec{u}_k^*$ ? V tomto zápisе horný index  $\cdot^*$  bude znamenať, že ide o optimálny vstup v každom vzorkovacom čase, t.j. najlepší aký môžeme použiť pre daný príklad. Problémom však je, že predikčná rovnica (3.12) túto sekvenciu využíva ako známu veličinu práve na výpočet budúcich stavov  $\vec{x}_k$ . Načo je potom dobrá predikčná rovnica, ak nepoznáme vstupy? V ďalšej kapitole si zodpovieme práve otázku, ako možno vyjadriť kvalitu riadenia na základe vstupov a stavu, a tak definovať závislosť priebehu riadenia od vstupov.



## Kapitola 4

# Účelová funkcia

Účelová funkcia<sup>1</sup>, resp. kriteriálna funkcia alebo stratová funkcia<sup>2</sup>, je umelo vytvorený numerický indikátor abstraktnej vlastnosti nejakého komplexného konceptu. Táto funkcia zobrazuje, resp. prenáša viacero funkcií (trajektórií) premenných do reálneho čísla, ktoré by intuitívne vyjadrilo „cenu“ zmien na vstupe.

Napríklad sa jedným skalárnym číslom môže vyjadriť aké dobré riadenie máme, hoci máme viacero výstupov, stavov a vstupov. Účelové funkcie nepoužívame iba v riadení, ale nájdeme ich všade kde používame optimalizačné metódy. Účelová funkcia môže napríklad vyjadrovať hodnotu portfólia firemných akcií, hoci portfólio obsahuje komplexnú interakciu mnohých faktorov. S účelovými funkciami sa môžeme stretnúť napríklad aj v umelej inteligencii alebo v strojovom učení. V ďalšej diskusii sa budeme sústredovať iba na formuláciu účelovej funkcie pre prediktívne riadenie.

### 4.1 Aká je ideálna účelová funkcia?

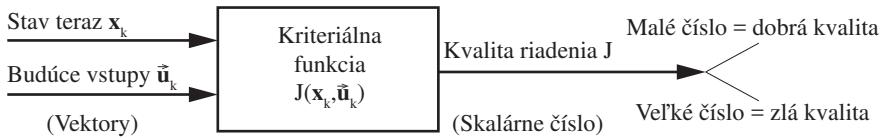
Výstup kriteriálnej funkcie je skalárne číslo, hoci na vstupe samotnej funkcie sa vyskytujú maticové a vektorové premenné. Práve v tomto spočíva sila tohto konceptu: zložité súvislosti vyjadrimo umelým číselným indikátorom. Skalárnu hodnotu na výstupe účelovej funkcie budeme označovať symbolom  $J$ .

Na riadenie je samozrejme vhodné zvoliť takú účelovú funkciu, ktorá vhodne vyjadruje komplexnú interakciu vstupov  $\vec{u}_k$  s dynamikou daného systému. Z predošej kapitoly už vieme, že predikovaná odozva dynamického systému závisí nielen od sekvencie budúcich vstupov, ale aj od aktuálneho stavu  $\mathbf{x}_k$ . Indikátor  $J$  potom určuje kvalitu riadenia podľa  $\mathbf{x}_k$  a  $\vec{u}_k$ . Matematicky to znamená, že máme

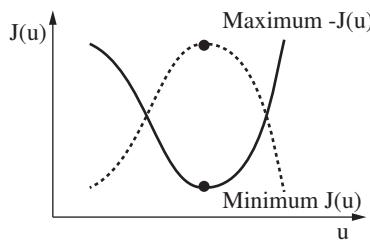
---

<sup>1</sup>angl. cost function

<sup>2</sup>angl. loss function



Obr. 4.1. Kriteriálna funkcia prediktívneho riadenia vyjadruje „cenu“, resp. kvalitu riadenia.



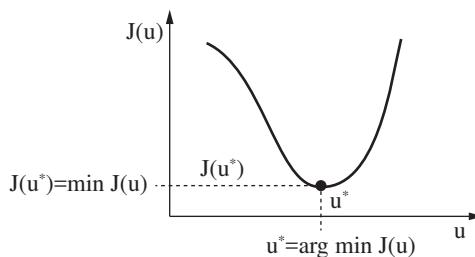
Obr. 4.2. Hľadanie maxima skalárnej funkcie môžeme chápať ako hľadanie minima tej istej funkcie so záporným znamienkom.

funkciu  $J(\mathbf{x}_k, \vec{u}_k)$  ktorá závisí od dvoch vektorov. Na Obr. 4.1 je znázorená kriteriálna funkcia, ktorá vyjadruje „cenu“, respektíve kvalitu budúcich vstupov  $\vec{u}_k$  pri danom stave  $\mathbf{x}_k$  pomocou skalára  $J$ .

#### 4.1.1 Optimalizačná úloha

Volba konkrétnej účelovej funkcie nikdy nie je pevne stanovená; dôvodom jej formulácie je, že chceme niečo optimalizovať. Optimalizačná úloha je vlastne iba hľadanie extrémov nejakej funkcie, teda jej minimálnej alebo maximálnej hodnoty. Dokonca pre naše účely stačí hovoriť iba o minimalizácii, lebo maximalizáciu môžeme pochopíť tak, že hľadáme minimum tej istej funkcie, ale s opačným znamienkom (Obr. 4.2).

Hoci už vieme, že v prediktívnom riadení chceme vyjadriť kvalitu riadenia a kriteriálna funkcia musí brať do úvahy aktuálny stav a budúce vstupy, v nasledujúcom si však ilustrujeme niektoré matematické pojmy všeobecne – bez ohľadu na ich ďalšie využitie v prediktívnom riadení. To znamená, že premenná  $u$  (resp.  $x$ ) bude označovať všeobecnú nezávislú vstupnú premennú funkcie  $J$ . Pre jednoduchšiu grafickú ilustráciu môžeme taktiež predpokladať, že nezávislá premenná  $u$  je skalár, a teda  $J(u)$  môžeme graficky znázorniť v dvojrozmernom priestore.



Obr. 4.3. Niekedy nás namiesto funkčnej hodnoty zaujíma hodnota argumentu funkcie v jej minime.

Aby sme kriteriálnu funkciu mohli používať na optimalizáciu, predpokladáme, že existuje taká hodnota premennej  $u$ , kde funkcia  $J$  nadobudne minimálnu hodnotu. Označme si optimálnu hodnotu argumentu pomocou hviezdičky ( $u^*$ ), matematicky potom hľadanie minima môžeme vyjadriť ako

$$J(u^*) = \min J(u), \quad (4.1)$$

kde výraz „ $\min$ “ vyjadruje optimalizačnú úlohu hľadania minima,  $J(u^*)$  vyjadruje akú hodnotu nadobudne optimalizačná úloha pri hodnote premennej  $u^*$ . Niekedy nehľadáme akú hodnotu má samotná funkcia  $J$  v optime, ale hľadáme priakej hodnote argumentu nadobúda kriteriálna funkcia minimum. Matematicky si toto označíme ako

$$u^* = \arg \min J(u), \quad (4.2)$$

kde „ $\arg$ “ znamená, že hľadáme argument, teda optimalizačnú premennú funkcie pri minime. Tento koncept je graficky znázornený na Obr. 4.3.

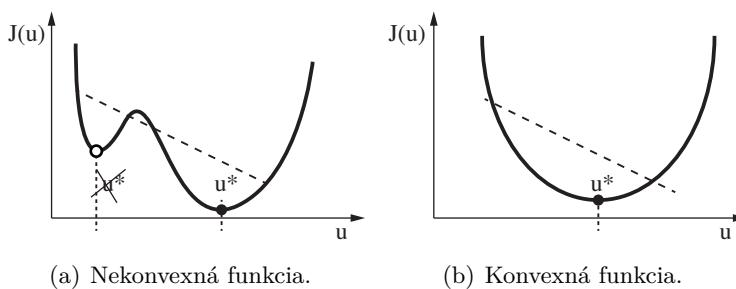
Niekedy treba optimalizačnú premennú bližšie špecifikovať. Napríklad, ak hodnota funkcie  $J(u, x)$  závisí na premennej  $u$  aj  $x$ , ale chceme ju minimalizovať iba podľa  $u$ , tak to vyjadrime ako

$$J(u^*) = \min_u J(u, x), \quad (4.3)$$

kde pod výrazom „ $\min$ “ píšeme optimalizačnú premennú, zatiaľ čo predpokladáme, že  $x$  je pevne dané.

### 4.1.2 Konvexnosť

Pre niektoré funkcie je nájdenie extrému pomerne ľahkou úlohou, no pre iné to môže byť problematické. Preto pri tvorení vhodnej účelovej funkcie musíme bráť



Obr. 4.4. Nekonvexná funkcia s lokálnym a globálnym extrémom a konvexná funkcia iba s globálnym extrémom.

do úvahy aj matematické vlastnosti danej optimalizačnej úlohy. Jasným cieľom môže byť výber takej účelovej funkcie, ktorá má iba globálne minimum a nemá lokálne minimá. Takoto funkciou je *konvexná funkcia*. Naopak, ak funkcia má aj lokálne minimá je nekonvexná (Obr. 4.4). Na Obr. 4.4 sme označili prerušovanú čiaru, ktorá spája dva body funkcie a pritom pretína samotnú funkciu. Ak taká čiara pre danú funkciu existuje, ide o nekonvexnú funkciu. Naopak, ak neexistuje žiadna taká čiara, funkcia je konvexná.

**Príklad 4.1.** Nakreslime si ľubovoľnú nekonvexnú funkciu s dvoma premennými  $\mathbf{u} = [u_1 \ u_2]^T$ , napríklad pomocou príkazu `peaks()`.

Zdrojový kód 4.1. `nekonvexna.m`

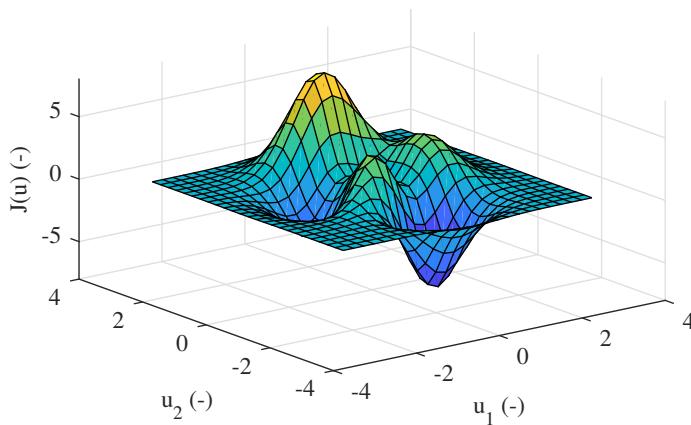
```
[X, Y, Z] = peaks(25); % Nekonvexna funkcia
surf(X, Y, Z); % Vykreslenie
xlabel('u_1 (-)') % Znacenie osi x
ylabel('u_2 (-)') % Znacenie osi y
zlabel('J(u) (-)') % Znacenie osi z
```

Príkaz `peaks()` vygeneruje známe logo softvéru MATLAB, ktoré predstavuje nekonvexnú funkciu (Obr. 4.5).

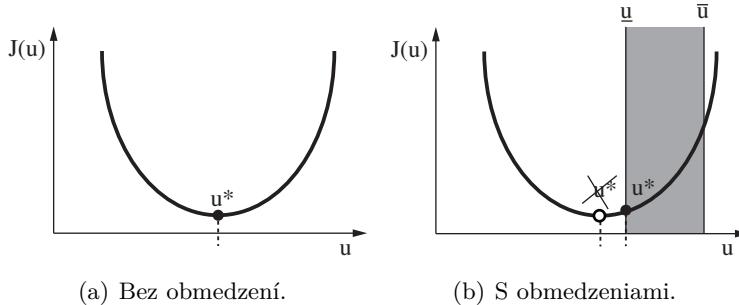
□

### 4.1.3 Obmedzenia

Nie je jedno, či môžeme využiť celý rozsah optimalizačnej premennej  $u$  alebo nie. Na Obr. 4.6 je grafická ilustrácia aj prípadu, že optimalizačnú premenú  $u$  môžeme



Obr. 4.5. Nekonvexná funkcia v 3D: logo MATLABu.

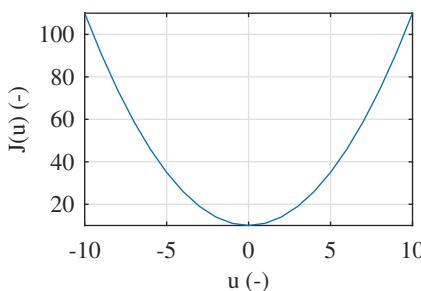


Obr. 4.6. Optimalizácia bez obmedzení a optimalizácia s obmedzeniami.

vybrať iba v rozmedzí dolného obmedzenia  $\underline{u}$  a horného obmedzenia  $\bar{u}$ . V takomto prípade platí iba riešenie zo sivej zóny Obr. 4.6. Na príklade je minimum funkcie mimo dovolenej zóny, t. j. mimo obmedzení; preto musíme vybrať riešenie, ktoré sa rovná obmedzeniu  $u^* = \underline{u}$ . Vidíme, že prítomnosť takýchto obmedzení značne skomplikuje optimalizačnú úlohu.

#### 4.1.4 Kvadratické účelové funkcie

Z pohľadu konvexnosti sú kvadratické matematické formulácie obvyklým kandidátom pre účelové funkcie, pretože majú jednoznačné globálne minimum. Kvadratické funkcie sú veľmi často zvolené pre optimalizačné úlohy, pretože ich mini-



Obr. 4.7. Kvadratická funkcia skalárnej nezávislej premennej.

malizácia je pomerne ľahko realizovateľná aj za prítomnosti obmedzení. Mnoho vedeckých a inžinierskych prístupov využíva kvadratické účelové funkcie, medzi nimi aj dobre známa metóda najmenších štvorcov, regresné modelovanie, či štatistické testy hypotéz. V oblasti riadenia je samozrejme najznámejším príkladom lineárne kvadratické riadenie (LQ), ktorá je aj pomenované podľa typu účelovej funkcie.

Ďalším dôvodom popularity kvadratických funkcií v optimalizácii je ich symetrickosť: kladná zmena na vstupe funkcie vytvorí na výstupe zmenu rovnakej amplitúdy ako záporná.

**Príklad 4.2.** Nakreslime si kvadratickú funkciu s jednou premennou  $J(u) = au^2 + bu + c$ , kde  $a = 1$ , kde  $b = 0$  a  $c = 10$ .

Najprv deklarujeme konštanty, potom vytvoríme číselný rozsah pre premennú  $u$  a vypočítame funkčné hodnoty v týchto bodoch. Výsledok zobrazíme graficky pomocou príkazu `plot()`, a označíme osi grafu:

Zdrojový kód 4.2. kvadratickafunkcia.m

```

a=1; b=0; c=10;      % Konstanty
u=-10:10;            % Premenna u
J=a*u.^2+b*u+c;    % Funkcia J
plot(u,J)            % Vykreslenie
grid on               % Mriezka
xlabel('u (-)')      % Znacenie osi x
ylabel('J(u) (-)')   % Znacenie osi y

```

Výsledkom je kvadratická funkcia  $J$  jednej premennej  $u$  (Obr. 4.7).

□

Premenné samozrejme nemusia byť iba skalárne; môžeme ich zhrnúť do vektorov, ktoré si označíme hrubým písmom  $\mathbf{u}$ . Z matematického hľadiska premenná  $\mathbf{u}$  môže mať ľubovoľný počet prvkov, t. j.  $\mathbf{u} = [u_1 \ u_2 \ \dots \ u_N]^T$  a stále môžeme formulovať skalárnu účelovú funkciu  $J(\mathbf{u})$ . Grafické zobrazenie skalárnej funkcie s viacerými premennými už je ale obmedzené: potrebujeme zobraziť jednu závislú funkčnú hodnotu  $J(\mathbf{u})$ , pričom zostávajú iba dve nezávislé premenné  $[u_1 \ u_2]^T$ , ktoré si môžeme ešte intuitívne graficky znázorniť.

**Príklad 4.3.** Nakreslime kvadratickú funkciu s dvoma premennými  $\mathbf{u} = [u_1 \ u_2]^T$ , danú vo forme

$$J(\mathbf{u}) = \mathbf{u}^T \mathbf{Q} \mathbf{u} + c,$$

kde

$$\mathbf{Q} = \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

a  $c = 10$ .

Funkciu môžeme prepísať do tvaru

$$\begin{aligned} J(\mathbf{u}) &= \mathbf{u}^T \mathbf{Q} \mathbf{u} + c \\ &= [u_1 \ u_2] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + 10 \\ &= u_1^2 + u_2^2 + 10 \end{aligned}$$

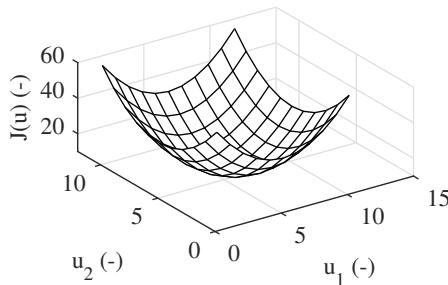
Na jej grafické znázornenie si vytvorime 2D siet premenných pomocou príkazu `meshgrid()` a vypočítajme príslušné funkčné hodnoty. Funkciu môžeme zobraziť pomocou príkazu `surf()`:

Zdrojový kód 4.3. kvadraticka3D.m

```
[u1 u2] = meshgrid(-10:10); % Premenna u
J = u1.^2+u2.^2+10; % Funkcia J
surf(J); % Vykreslenie
xlabel('u_1 (-)') % Znacenie osi x
ylabel('u_2 (-)') % Znacenie osi y
zlabel('J(u) (-)') % Znacenie osi z
```

Výsledkom je kvadratická funkcia závislá od dvoch premenných, zobrazená v 3D (Obr. 4.8).

□



Obr. 4.8. Skalárna kvadratická funkcia závislá od dvoch premenných.

#### 4.1.5 Minimalizácia kvadratickej funkcie

Ako už vieme, kvadratické funkcie sú konvexné. Nájsť extrém – v našom prípade minimum – kvadratickej funkcie v absencii obmedzení je jednoduchá úloha. Musíme nájsť bod, v ktorom funkcia ani nestúpa ani neklesá, inými slovami dotyčnica má nulový sklon. V takomto prípade má derivácia funkcie v danom bode nulovú hodnotu. Ak chceme teda nájsť extrém funkcie  $J(u)$ , musíme vypočítať

$$\frac{dJ}{du} = 0, \quad (4.4)$$

a z toho vyjadriť extrém  $u = u^*$ . Funkčnú hodnotu pri extréme nájdeme jeho spätným dosadením do funkcie, t.j.  $J(u^*)$ .

**Príklad 4.4.** Nájdite a zobrazte minimum skalárnej kvadratickej funkcie  $J(u) = 2u^2 - 2u + 3$ .

Derivujme funkciu a nájdime bod, v ktorom sa táto rovná nule:

$$\begin{aligned} \frac{dJ}{du} &= 0, \\ 4u^* - 2 &= 0, \\ u^* &= 0.5, \end{aligned}$$

a potom dostaneme aj príslušnú funkčnú hodnotu v nájdenom minime ako:

$$\begin{aligned} J(u^*) &= 2(u^*)^2 - 2(u^*) + 3, \\ &= 2(0.5)^2 - 2(0.5) + 3, \\ &= 2.5. \end{aligned}$$

V MATLABe môžeme pomocou rozšírenia Symbolic Toolbox tieto jednoduché výpočty prekontrolovať. Pomocou príkazu `syms` si deklarujeme symbolickú funkciu, ktorú potom aj špecifikujeme. Príkaz `diff()` vykoná deriváciu<sup>3</sup> danej funkcie a príslušnú rovnicu vyriešime<sup>4</sup> pomocou príkazu `solve()`:

Zdrojový kód 4.4. kvadraticafunkciamin.m

```
a=2; b=-2; c=3; % Konstanty
syms J(u) % Symbolicka funkcia
J(u)=a*u.^2+b*u+c; % Definica funkcie
Jd=diff(J(u)) % Derivacia
uast=solve(Jd,u) % Vyries pre u
juast=J(uast) % funkcnia hodnota
```

Ak spustíme makro, dostaneme nasledujúce výsledky:

```
Jd = 4*u - 2
jast = 1/2
juast = 5/2
```

ktoré sú totožné s výsledkami ručného výpočtu. Symbolické funkcie môžeme využívať aj pri grafickom zobrazení minima, keď dosadíme za premenné:

```
u=-2:0.1:3; % Premenna u
plot(u,J(u)) % Vykreslenie funkcie
hold on % Nechaj graf
plot(uast,J(uast), 'o') % Vykreslenie minima
grid on % Mriezka
xlabel('u (-)') % Znacenie osi x
ylabel('J(u) (-)') % Znacenie osi y
legend('J(u)', 'Minimum') % Legenda
```

z čoho dostaneme grafické zobrazenie funkcie a minima na Obr. 4.9.

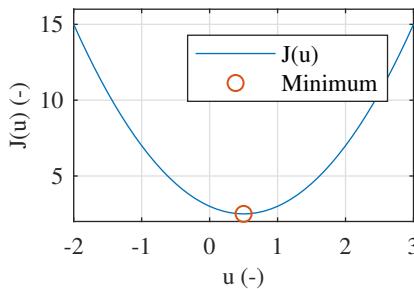
□

Podobné kroky musíme vykonať, ak kvadratická funkcia závisí od viacerých premenných  $\mathbf{u}$ . V tomto prípade parciálne derivujeme funkciu  $J(\mathbf{u})$

$$\nabla J(\mathbf{u}) = \left[ \frac{\partial J}{\partial u_1}, \frac{\partial J}{\partial u_2}, \dots, \frac{\partial J}{\partial u_N} \right]^T, \quad (4.5)$$

<sup>3</sup>angl. differentiation

<sup>4</sup>angl. solve



Obr. 4.9. Minimum skalárnej kvadratickej funkcie.

kde operátor  $\nabla$  (nabla) vyjadruje takzvaný gradient a matematicky znamená len to, že vykonáme parciálnu deriváciu podľa každého prvku vektorovej premennej  $\mathbf{u}$ , t. j. podľa  $u_1, u_2, \dots, u_N$ .

Ak chceme ďalej nájsť minimum funkcie, hľadáme opäť bod, v ktorom sa jej gradient rovná nule, teda riešime

$$\nabla J(\mathbf{u}) = \left( \frac{\partial J}{\partial u_1}, \frac{\partial J}{\partial u_2}, \dots, \frac{\partial J}{\partial u_N} \right) = 0, \quad (4.6)$$

výsledkom čoho je optimum  $\mathbf{u}^*$ , t. j. bod, v ktorom funkcia nadobúda minimálnu hodnotu.

**Príklad 4.5.** Nájdite minimum funkcie  $J(\mathbf{u}) = \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{G} \mathbf{u}$ , kde

$$\mathbf{H} = \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix} \quad \text{a} \quad \mathbf{G} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Ak dosadíme dané maticové výrazy a vynásobíme ich s príslušnými premennými, dostaneme

$$J(\mathbf{u}) = 2u_1^2 + u_1 + 2u_2^2.$$

Gradient tejto funkcie vypočítame ako parciálnu deriváciu funkcie podľa  $u_1$  a  $u_2$ , teda

$$\begin{aligned} \nabla J(\mathbf{u}) &= \left[ \frac{\partial J}{\partial u_1}, \frac{\partial J}{\partial u_2} \right]^T \\ &= \begin{bmatrix} 4u_1 + 1 \\ 4u_2 + 2 \end{bmatrix} \end{aligned}$$

čo pri  $\nabla J(\mathbf{u}) = \mathbf{0}$  vedie na  $u_1^* = -0.25$  a  $u_2^* = -0.5$  a príslušnú funkčnú hodnotu v optimu,  $J(\mathbf{u}^*) = 1.375$ .

Uvedené výpočty môžeme jednoducho overiť aj pomocou MATLABu. Po deklarovaní premenných  $H$  a  $G$  si vytvoríme symbolické premenné vektora  $u$ . Kvadratickú rovnicu môžeme zjednodušiť pomocou príkazu `simplify()` a gradient vypočítať pomocou funkcie `gradient()`. Výsledok, t. j. minimum funkcie dostaneme, ak vyriešime sústavu rovníc pri  $\nabla J(\mathbf{u}) = \mathbf{0}$ :

Zdrojový kód 4.5. kvadraticafunkcia3Dmin.m

```

H=[2,-1;                                % Konstanty
   1, 2];
G=[1 2];                                % Konstanty
u = sym('u', [2 1]);                    % Symbolicky vektor
J(u)=u.'*H*u+G*u;                      % Definica funkcie
J=simplify(J)                           % Zjednodusenie

```

a dostaneme identické výsledky ako pomocou ručného výpočtu:

```

J(u1, u2) = 2*u1^2 + u1 + 2*u2^2 + 2*u2
nablaJ(u1, u2) = 4*u1 + 1
                  4*u2 + 2
u1ast = -1/4
u2ast = -1/2

```

Pre zobrazenie funkcie môžeme využiť funkciu `surf()` a `plot3()`:

```

[u1,u2] = meshgrid(-2:0.25:1,-2:0.25:1);
surf(u1,u2,double(J(u1,u2)));      % Vykreslenie
hold on                            % Podrzanie grafu
plot3(u1ast,u2ast,J(u1ast,u2ast),'o','MarkerSize',10)
xlabel('u_1 (-)')                  % Znacenie osi x

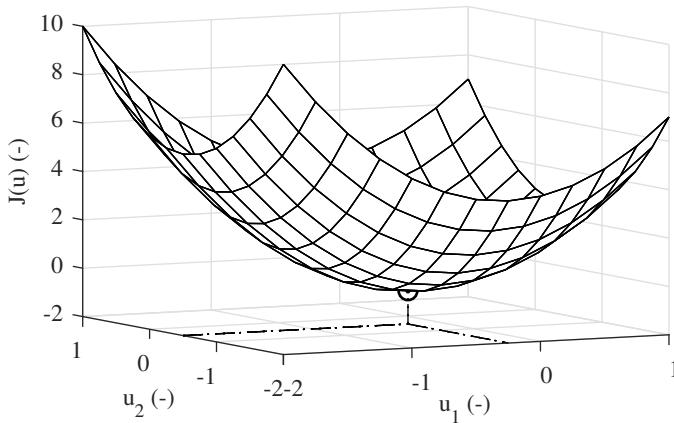
```

Dostaneme tak ilustráciu funkcie a jej minima, ako je znázornené na Obr. 4.10.

□

## 4.2 Účelová funkcia na konečnom horizonte

V nasledujúcom si na základe predošej diskusie vytvoríme účelovú funkciu v tvare vhodnom pre prediktívne riadenie. Účelová funkcia bude brať do úvahy vstupy a dynamickú odozvu riadeného systému, bude mať kvadratickú formu a jej závislá premenná bude skalárne číslo.



Obr. 4.10. Minimum kvadratickej funkcie v 3D.

#### 4.2.1 Hodnota úcelovej funkcie v budúcnosti

Označme pomocou  $j_k$  kvalitu riadenia vyjadrenú iba v jednom časovom okamihu  $k$ . V budúcnosti môžeme taktiež vyjadriť individuálne indikátory kvality, ak časový index posunieme dopredu  $i$  krokov, t. j.  $j_{k+i}$  pre  $i = 0, 1, 2, \dots$

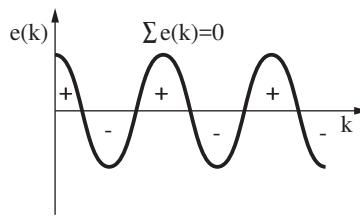
Sčítaním jednotlivých  $j_k$  v rámci trvania predikčného horizontu potom môžeme vyjadriť úcelovú funkciu  $J_k$  podľa

$$J_k = \sum_{i=0}^{n_p-1} j_{k+i}, \quad (4.7)$$

kde  $J_k$  už hovorí o predikovanej hodnote úcelovej funkcie v budúcnosti. Podobne ako sme rozlišovali stav  $\mathbf{x}_k$  v čase  $k$  a celú predikovanú sekvenciu stavov  $\vec{\mathbf{x}}_k$ , index  $k$  pri  $J_k$  bude označovať kvalitu riadenia v budúcnosti, až do konca horizontu. Všimnime si, že časové indexy začínajú v  $i = 0$  ( $k$ ) a končia v  $i = n_p - 1$  ( $k + n_p - 1$ ). Ak teda začíname počítať od nulového indexu,  $n_p$  krokov dlhý horizont dosiahneme v čase  $n_p - 1$ . Toto označenie je dôležité najmä ak uvažujeme vstupy riadenia, ktoré indexujeme od  $k$  do  $k + n_p - 1$  na konci horizontu.

#### 4.2.2 Odchýlka riadenia ako indikátor kvality

Za indikátor kvality riadenia môžeme jednoducho považovať odchýlku riadenia, resp. regulačnú odchýlku v časovom okamihu  $k$ , ktorá je daná rozdielom žiadanej hodnoty (referencie)  $r$  a aktuálne nameranej hodnoty  $y_k$ . Na začiatok uvažujme



Obr. 4.11. Odchýlku riadenia nemôžeme priamo používať ako indikátor kvality.

iba systém s jedným výstupom, potom dostaneme

$$j_k = e_k = r - y_k. \quad (4.8)$$

Na zjednodušenie ďalšieho odvodenia predpokladajme, že máme nulovú referenciu,  $r = 0$ , inými slovami to znamená, že systém chceme riadiť do rovnovážnej polohy. Potom odchýlka riadenia a zároveň aj indikátor kvality v čase  $k$  bude mať tvar

$$j_k = e_k = 0 - y_k = -y_k. \quad (4.9)$$

Indikátor kvality podľa (4.9) žiaľ nie je vhodný na návrh účelovej funkcie, lebo môže dávať aj kladné aj záporné čísla, a pri sčítaní viacerých časových okamihov

$$J_k = \sum_{k=0}^{n_p-1} e_k = \sum_{k=1}^{n_p-1} -y_k, \quad (4.10)$$

pre  $k = 0, 1, 2, \dots, n_p - 1$  nemá  $J_k$  žiadnu výpovednú hodnotu (Obr. 4.11).

Každá odchýlka  $e$  od referencie  $r$  je nežiaduca, a preto chceme vytvoriť takú účelovú funkciu, ktorá rovnako zohľadňuje kladné ako aj záporné zmeny oproti žiadanej hodnote. Bolo by dobré, keby každá zložka  $j_k$  účelovej funkcie  $J_k$  bola preto kladná. Aby sme pre každú odchýlku dostali kladné číslo, mohli by sme hypoteticky využiť aj absolútne hodnotu  $j_k = |e_k| = |-y_k| = |y_k|$ . Absolútne hodnota čísla je nelineárny matematický operátor, a aj napriek tomu, že by jej použitie viedlo na tzv. lineárne programovanie, z praktických dôvodov sa mu teraz vyhneme.

V prípade jedného vstupu by sme vedeli jednoducho používať kvadrát výstupov  $j_k = y_k^2$  a určite by sme dostali kladné číslo v každom časovom okamihu  $k$  aj pre viacero okamihov  $k = 0, 1, 2, \dots, n_p - 1$ . Kvalita riadenia v čase  $k$  pre systém s jedným výstupom potom bude daná vzťahom

$$j_k = e_k^2 = (-y_k)^2 = y_k^2, \quad (4.11)$$

ktorý je už vhodný ako základ návrhu účelovej funkcie v tvare

$$J_k = \sum_{i=1}^{n_p-1} j_k = \sum_{i=1}^{n_p-1} y_k^2. \quad (4.12)$$

Tento tvar je vhodný aj z pohľadu diskusie z Kap. 4.1.4, kde sme spomínali, že kvadratické funkcie sú výhodné pre optimalizačné úlohy.

Účelovú funkciu chceme formulovať všeobecne, aj pre systémy s viacerými výstupmi. V maticovej algebre kvadrát ľubovoľného vektora  $\mathbf{y}_k$  vyjadríme ako  $\mathbf{y}_k^T \mathbf{y}_k$ . Pre vektor výstupov potom dostaneme

$$j_k = \mathbf{y}_k^T \mathbf{y}_k, \quad (4.13)$$

čo vedie na účelovú funkciu v tvare

$$J_k = \sum_{i=1}^{n_p-1} \mathbf{y}_k^T \mathbf{y}_k. \quad (4.14)$$

Číslo  $j_k$  a  $J_k$  bude stále nezáporné a skalárne a čím má menšiu hodnotu, tým máme lepšie riadenie. Viacero výstupov taktiež nebude predstavovať problém, keďže násobením  $\mathbf{y}_k^T \mathbf{y}_k$  dostaneme opäť skalár.

### 4.2.3 Stavy v účelovej funkcií

Výstupy  $\mathbf{y}_k$  systému predstavujú veličiny, ktoré sú fyzicky dostupné pomocou meraní zo snímačov. Vieme, že jedným z dôvodov prečo v modernom riadení preferujeme stavovú reprezentáciu je, že stavy opisujú aj vnútorné dynamické väzby modelu. Nebolo by potom múdre používať na opis kvality riadenia iba výstupy, keď vieme, že  $\mathbf{y}_k = \mathbf{C} \mathbf{x}_k$ . Dosadením do Rov. (4.13) dostaneme výraz

$$j_k = \mathbf{y}_k^T \mathbf{y}_k = \mathbf{x}_k^T \mathbf{C}^T \mathbf{C} \mathbf{x}_k, \quad (4.15)$$

v ktorého „strede“ vidíme súčin  $\mathbf{C}^T \mathbf{C}$ . Nahradíme tento všeobecným výrazom  $\mathbf{Q}$  a dostaneme

$$j_k = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k, \quad (4.16)$$

kde  $\mathbf{Q}$  je takzvaná váhovacia<sup>5</sup>, resp. penalizačná matica<sup>6</sup> stavu.

---

<sup>5</sup>angl. weighting matrix

<sup>6</sup>angl. penalty matrix

Váhovaciu maticu  $\mathbf{Q}$  volíme pri návrhu riadenia ľubovoľne a pomocou nej môžeme doladiť správanie sa regulátora. Musíme ale pritom dodržať niekoľko podmienok. V prvom rade je zrejmé, že penalizačná matica stavov musí byť štvorcová a s rozmermi  $n_x \times n_x$ , aby sme násobením  $\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k$  dostali vždy skalárne číslo, teda

$$\mathbf{Q} \in \mathbb{R}^{n_x \times n_x}. \quad (4.17)$$

Ďalej, váhovacia matica vstupov musí byť symetrická, inak povedané po jej transpozícii dostaneme tú istú maticu:

$$\mathbf{Q} = \mathbf{Q}^T. \quad (4.18)$$

Matica  $\mathbf{Q}$  je kladne semidefinitná, čo je maticová analógia kladných skalárov a nuly. Dôvodom kladnej semidefinitnosti je to, že vždy chceme dostať kladné číslo  $j_k$ . Hypoteticky môžeme zvoliť aj samé nuly, čím by sme ale plne ignorovali vplyv stavov systému na kvalitu riadenia prostredníctvom kladnej hodnoty stratovej funkcie  $j_k$ . Matematicky túto podmienku môžeme vyjadriť ako

$$\mathbf{Q} \succeq \mathbf{0}. \quad (4.19)$$

Za penalizačnú maticu stavov  $\mathbf{Q}$  môžeme napríklad voliť

$$\mathbf{Q} = \mathbf{C}^T \mathbf{C}, \quad (4.20)$$

čo by znamenalo, že v účelovej funkcií penalizujeme namiesto stavov priamo výstupy, pretože  $\mathbf{y}_k = \mathbf{C} \mathbf{x}_k$ . Dosadením  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  do účelovej funkcie teda minimalizujeme stav alebo kombináciu stavov ktorú, meriame na výstupe.

Častou voľbou penalizačnej matice stavov je aj jednotková matica, t. j.  $\mathbf{Q} = \mathbf{I}$ , ktorá má taký význam, že rovnako penalizujeme všetky stavy. Ich minimalizácia a kvalita potom je rovnako dôležitá.

Pri dodržaní matematických predpokladov vlastností matice  $\mathbf{Q}$  môžeme robiť aj jemnejšie ladenie. Napríklad jednotkovú maticu môžeme vynásobiť skalárom  $q$ , t. j.  $\mathbf{Q} = q \mathbf{I}$ . Jednotlivé zložky na diagonále taktiež môžeme voľne meniť, pričom platí, že väčšie číslo znamená, že daný stav má v účelovej funkcií väčší význam – váhu. Uvažujme napríklad maticu

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}, \quad (4.21)$$

ktorej diagonálne prvky naznačujú, že prvý stav minimalizujeme – a tým pádom aj riadime – s  $10 \times$  väčšou váhou ako druhý stav. Inak povedané, správnosť a riadenie prvého prvku stavového vektora je desaťkrát dôležitejšie ako druhého.

#### 4.2.4 Vstupy v účelovej funkcií

Naša účelová funkcia v tvare (4.16) sa často rozširuje o člen uvažujúci akčné zásahy  $\mathbf{u}_k$  a ich vplyv na výslednú kvalitu riadenia.

Použime ďalej rovnakú logiku ako pri zavedení výstupov do účelovej funkcie, a zohľadnime aj vplyv vstupov na riadenie. Výraz  $\mathbf{u}_k^T \mathbf{u}_k$  (maticová analógia  $u_k^2$ ) potom vždy dáva kladný skalár, preto stratovú funkciu z Rov. (4.16) môžeme rozšíriť do tvaru

$$j_k = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{u}_k. \quad (4.22)$$

Na jemnejšie nastavovanie vplyvu vstupov zavedieme novú váhovaciu maticu a dostaneme

$$j_k = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k, \quad (4.23)$$

kde  $\mathbf{R}$  budeme ďalej nazývať váhovacou (penalizačnou) maticou vstupov.

Maticu  $\mathbf{R}$  podobne ako maticu  $\mathbf{Q}$  si volíme sami a vyjadrujeme tým relatívnu dôležitosť vplyvu jednotlivých vstupov. Tak ako to platí pri váhovaní stavov, aj tu musíme dodržať niekoľko logických podmienok. Penalizačná matica vstupov je vždy štvorcová matica rozmeru  $n_u \times n_u$

$$\mathbf{R} \in \mathbb{R}^{n_u \times n_u},$$

tak, aby sme násobením  $\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k$  dostali vždy skalárne číslo. Ďalej, rovnako ako  $\mathbf{Q}$  je matica  $\mathbf{R}$  symetrická, t. j.  $\mathbf{R} = \mathbf{R}^T$ . Váhovacia matica vstupov je kladne definitná, teda

$$\mathbf{R} \succ \mathbf{0},$$

kedže aj v tomto prípade platí, že chceme dostať kladné výsledky. Na rozdiel od  $\mathbf{Q}$  sa však štandardne neuvažuje nulové váhovanie; nejde teda o kladne *semidefinitnú* ale kladne definitnú maticu, kedže by sme tým eliminovali vplyv riadenia.

Podobné úvahy ako pri váhovaní stavov môžeme použiť aj na správne zvolenie váhovacej matice vstupov,  $\mathbf{R}$ . V prípade systému s jedným vstupom je to skalár, t. j.  $\mathbf{R} = R \in \mathbb{R}$ . Väčšie číslo pritom vyjadruje väčší príspevok vstupov do účelovej funkcie a zároveň konzervatívnejšie riadenie. Naopak, menšie číslo vyjadruje menší príspevok do účelovej funkcie, čo znamená, že si neželáme minimalizovať účinok vstupov a požadujeme tak agresívnejšie riadenie. Uvažujme napríklad maticu

$$\mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}.$$

Táto voľba by v prípade dvoch vstupov znamenala, že druhým vstupom chceme šetriť  $10 \times$  viac ako prvým a účinok prvého sa v riadení prejaví  $10 \times$  agresívnejšie.

### 4.2.5 Výsledná formulácia funkcie

Na základe Rov. (4.7) je teda účelová funkcia suma jednokrokových strát  $j_k$ ,

$$J_k = \sum_{i=0}^{n_p-1} j_{k+i} = \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right), \quad (4.24)$$

a vyjadruje kvalitu riadenia z pohľadu sledovania nulovej referencie a vplyvu vstupov od súčasnosti až do konca horizontu.

Všimnime si ešte raz indexovanie stavov a vstupov (Obr. 4.12). Prvý vstup, ktorý sa vyskytuje v účelovej funkcií,  $\mathbf{u}_k$ , je vlastne akčný zásah, ktorý nás zaujíma v súčasnosti (teda v čase  $k$ ). Prvým stavom v účelovej funkcií je  $\mathbf{x}_k$  a poznáme ho na základe merania alebo odhadu,  $\mathbf{x}_k = \hat{\mathbf{x}}_k$ , pričom naň nevplýva vstup  $\mathbf{u}_k$ . Hoci je tento stav základom predikcie, už ho nemôžeme korigovať pomocou vstupu  $\mathbf{u}_k$ . Keď dosadíme za jednotlivé predikované stavy v Kap. 4.3, uvidíme, že zatiaľ čo je tento stav nevyhnutný na vytvorenie predikcie, z pohľadu výpočtu vstupov je zanedbateľný.

Posledný vstup v účelovej funkcií (4.24) je  $\mathbf{u}_{k+n_p-1}$ , ktorý je aj posledným blokom vektora  $\vec{\mathbf{u}}_k$ . Vstupy  $\mathbf{u}_{k+i}$  pre  $i = 0, 1, \dots, n_p - 1$  dajú presne  $n_p$  krokov dlhý horizont. Posledný vstup potom vplyvom tvarovača potrvá od vzorky  $k+n_p-1$  až do  $k+n_p$  a teda pádom vplýva na posledný stav v horizonte, t. j. v čase  $k+n_p$ . V prípade stavov môžeme hovoriť o horizonte predikcie, ktorý potrvá  $n_p$  krokov od  $i = 1, \dots, n_p$  a pri vstupoch môžeme hovoriť o horizonte riadenia v trvaní od  $i = 0, \dots, n_p - 1$ . Dokonca je možné predstaviť si aj situáciu, keď tieto horizonty nemusia byť rovnako dlhé, ale v tejto učebnici budeme ďalej predpokladať, že horizont predikcie je totožný s horizontom riadenia a budeme teda používať iba značenie  $n_p$ .

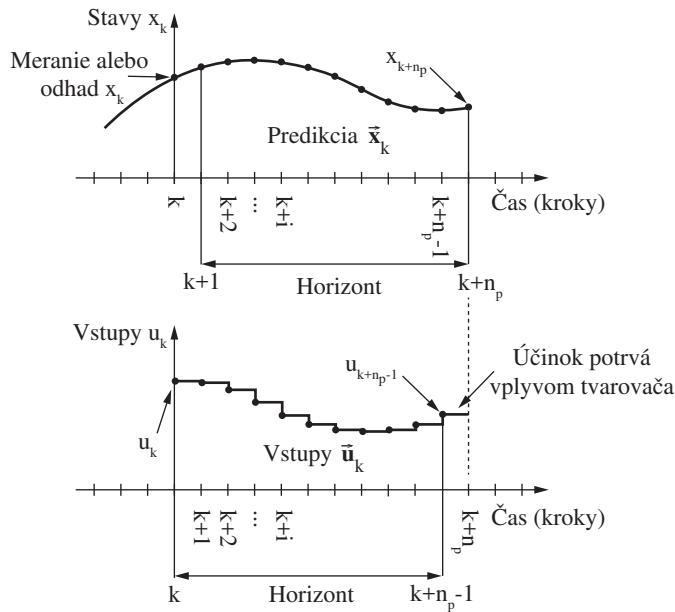
Účinok vstupov na stavy podľa Rov. (4.24) končí pri  $\mathbf{x}_{k+n_p-1}$ . Stav v poslednom kroku kritéria,  $\mathbf{x}_{k+n_p}$ , váhujeme samostatne, iba z dôvodu indikácie, či hodnota účelovej funkcie (4.24) smerom ku koncu horizontu predikcie klesá a systém sa stabilizuje. Označme stav na konci horizontu, tzv. koncový stav, ako  $\mathbf{x}_{k+n_p}$ , a váhujme tretí člen účelovej funkcie novou maticou  $\mathbf{P}$ .

Účelová funkcia je daná v tvare

$$J_k = \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}, \quad (4.25)$$



kde  $\mathbf{P}$  budeme nazývať váhovacou (penalizačnou) maticou koncového stavu  $\mathbf{x}_{k+n_p}$ . Je pritom jasné, že nemusíme uvažovať vstup  $\mathbf{u}_{n_p}$ , lebo na posledný stav vplýva iba predchádzajúci vstup  $\mathbf{u}_{n_p-1}$  (Obr. 4.12).



Obr. 4.12. Indexovanie vstupov a stavov v účelovej funkcií.

Maticu váhovania koncového stavu,  $\mathbf{P} \in \mathbb{R}^{n_x \times n_x}$ , si hypoteticky môžeme voľne určiť, ale neskôr bude táto jej voľba veľmi dôležitá. Má totiž podstatný vplyv na výber stabilizujúcej postupnosti riadenia, a teda stabilitu v uzavretej slučke. Dočasne môžeme napríklad predpokladať, že matica váhovania koncového stavu bude totožná s maticou váhovania stavov, t. j.  $\mathbf{P} = \mathbf{Q}$ . Potom by sme uvažovali účelovú funkciu v tvare

$$\begin{aligned}
 J_k &= \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{Q} \mathbf{x}_{k+n_p} \\
 &= \sum_{i=0}^{n_p} \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \sum_{i=0}^{n_p-1} \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i}.
 \end{aligned} \tag{4.26}$$

### 4.3 Kompaktný zápis účelovej funkcie

Rozšírime si teraz definíciu účelovej funkcie (4.25) o známy počiatočný stav

$$J_k = \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}, \quad (4.27)$$

vzhľadom na podmienku:

$$\mathbf{x}_k = \hat{\mathbf{x}}_k,$$

kde pridaná podmienka vyjadruje len to, že na začiatku horizontu máme k dispozícii údaj o stave na základe merania alebo odhadu  $\hat{\mathbf{x}}_k$ . Podmienka známeho stavu nie je jedinou podmienkou pri výpočte účelovej funkcie. Musíme špecifikovať aj spôsob akým chceme vypočítať ďalšie stavy  $\mathbf{x}_{k+i}$ . Rozšírime preto účelovú funkciu (4.27) o stavovú rovnicu a dostaneme:

$$J_k = \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}, \quad (4.28)$$

vzhľadom na podmienky:

$$\mathbf{x}_k = \hat{\mathbf{x}}_k,$$

$$\mathbf{x}_{k+i+1} = \mathbf{A} \mathbf{x}_{k+i} + \mathbf{B} \mathbf{u}_{k+i} \quad \text{pre } i = 0, 1, 2, \dots, n_p - 1.$$

Účelová funkcia v tvare Rov. (4.28) je sice z matematického hľadiska správna, ale nebolo by praktické ju využívať v počítačovom algoritme kvôli operáciu sumácie. Naštastie v predchádzajúcej kapitole sme si pomocou predikčných matíc vyjadrili budúce stavy  $\mathbf{x}_{k+i}$ .

Z predikčnej rovnice môžeme vybrať  $i$ -té blokové riadky, aby sme vypočítali budúci stav  $i$  krokov od aktuálnej vzorky v čase  $k$ . Dosadme za budúce stavy v účelovej funkcií na základe Rov. (3.18) a dostaneme

$$\begin{aligned} J_k = & \sum_{i=0}^{n_p-1} \left[ (\mathbf{M}_i \mathbf{x}_k + \mathbf{N}_i \vec{\mathbf{u}}_k)^T \mathbf{Q} (\mathbf{M}_i \mathbf{x}_k + \mathbf{N}_i \vec{\mathbf{u}}_k) + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right] + \quad (4.29) \\ & + (\mathbf{M}_{n_p} \mathbf{x}_k + \mathbf{N}_{n_p} \vec{\mathbf{u}}_k)^T \mathbf{P} (\mathbf{M}_{n_p} \mathbf{x}_k + \mathbf{N}_{n_p} \vec{\mathbf{u}}_k), \end{aligned}$$

kde  $\mathbf{M}_i$  a  $\mathbf{N}_i$  sú  $i$ -té blokové riadky predikčných matíc podľa Rov. (3.16)–(3.17) a  $\mathbf{M}_{n_p}$  a  $\mathbf{N}_{n_p}$  sú posledné blokové riadky na konci predikčného horizontu.

Všimnime si, že indexovanie blokov začíname od  $i = 0$ , čím vyjadrujeme aktuálny meraný, resp. odhadnutý stav  $\mathbf{x}_k = \hat{\mathbf{x}}_k$ . Následne pre  $i = 0$  nemusíme nič predikovať a máme

$$\mathbf{x}_k = \mathbf{M}_0 \mathbf{x}_k + \mathbf{N}_0 \vec{\mathbf{u}}_k = \mathbf{I} \mathbf{x}_k + \mathbf{0} \vec{\mathbf{u}}_k = \mathbf{x}_k, \quad (4.30)$$

a teda  $\mathbf{M}_0 = \mathbf{I}^{n_x \times n_x}$  a  $\mathbf{N}_0 = \mathbf{0}^{n_x \times n_u \cdot n_p}$ .

Vráťme sa k Rov. (4.29) a prenásobme kvadratické členy podľa pravidiel maticovej algebry. Môžeme na to využiť vzťah, ktorý platí pre kvadratické výrazy analogicky ako v skalárnom prípade  $(c+d)^2 = c^2 + 2cd + d^2$ , t. j. podľa

$$(\mathbf{C} + \mathbf{D})^2 = \mathbf{C}^T \mathbf{C} + 2\mathbf{C}^T \mathbf{D} + \mathbf{D}^T \mathbf{D}. \quad (4.31)$$

V našom prípade teda dostávame

$$\begin{aligned} J_k = & \sum_{i=0}^{n_p-1} \left[ \mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{M}_i \mathbf{x}_k + 2\mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k + \vec{\mathbf{u}}_k^T \mathbf{N}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k + \right. \\ & + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \Big] + \mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p} \mathbf{x}_k + 2\mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k + \\ & + \vec{\mathbf{u}}_k^T \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k, \end{aligned} \quad (4.32)$$

čo môžeme ďalej rozpísat ako

$$\begin{aligned} J_k = & \sum_{i=0}^{n_p-1} \mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{M}_i \mathbf{x}_k + \sum_{i=0}^{n_p-1} 2\mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k + \\ & + \sum_{i=0}^{n_p-1} \vec{\mathbf{u}}_k^T \mathbf{N}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k + \sum_{i=0}^{n_p-1} \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} + \\ & + \mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p} \mathbf{x}_k + 2\mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k + \\ & + \vec{\mathbf{u}}_k^T \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k. \end{aligned} \quad (4.33)$$

Ako vidíme, naša kriteriálna funkcia začína mať príliš zložitý matematický tvar, a tak sa ho poskúsme zjednodušiť. Zavedme si preto štvorcovú blokovo diagonálnu maticu  $\mathcal{R}$ , ktorá bude obsahovať penalizačnú maticu vstupov na hlavnej diagonále, teda

$$\mathcal{R} = \text{diag}(\mathbf{R}) = \begin{bmatrix} \mathbf{R} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{R} \end{bmatrix}, \quad (4.34)$$

čo je vlastne rozšírená matica váhovania vstupov platná pre celú postupnosť  $\vec{\mathbf{u}}_k$  namiesto  $\mathbf{u}_k$ .

Využitím novej rozšírenej matice váhovania vstupov môžeme zjednodušiť sumu budúcich vstupov a nahradíť individálne vstupy  $\mathbf{u}_{k+i}$  v časoch  $k + i$  vektorom

budúcich vstupov  $\vec{\mathbf{u}}_k$  ako

$$\sum_{i=0}^{n_p-1} \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} = \vec{\mathbf{u}}_k^T \begin{bmatrix} \mathbf{R} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{R} \end{bmatrix} \vec{\mathbf{u}}_k = \vec{\mathbf{u}}_k^T \mathcal{R} \vec{\mathbf{u}}_k. \quad (4.35)$$

Dosadme teraz zjednodušenú sumu do účelovej funkcie a všimnime si, že niektoré výrazy sú závislé iba od budúcich vstupov  $\vec{\mathbf{u}}_k$ , niektoré iba od nameraného alebo odhadnutého stavu  $\mathbf{x}_k$ , a niektoré od kombinácie týchto dvoch veličín:

$$\begin{aligned} J_k = & \sum_{i=0}^{n_p-1} \underbrace{\mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{M}_i \mathbf{x}_k}_{f(\mathbf{x}_k)} + \sum_{i=0}^{n_p-1} \underbrace{2\mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k}_{f(\mathbf{x}_k, \vec{\mathbf{u}}_k)} + \\ & + \sum_{i=0}^{n_p-1} \underbrace{\vec{\mathbf{u}}_k^T \mathbf{N}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k}_{f(\vec{\mathbf{u}}_k)} + \underbrace{\vec{\mathbf{u}}_k^T \mathcal{R} \vec{\mathbf{u}}_k}_{f(\vec{\mathbf{u}}_k)} + \\ & + \underbrace{\mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p} \mathbf{x}_k}_{f(\mathbf{x}_k)} + 2 \underbrace{\mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k}_{f(\mathbf{x}_k, \vec{\mathbf{u}}_k)} + \underbrace{\vec{\mathbf{u}}_k^T \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k}_{f(\vec{\mathbf{u}}_k)}. \end{aligned} \quad (4.36)$$

Zameňme ďalej poradie členov tak, aby sme mali výrazy kriteriálnej funkcie závislé najprv len od budúcich vstupov, potom od stavu a vstupov, a potom iba od stavu samotného, t.j. ako nasleduje:

$$\begin{aligned} J_k = & \sum_{i=0}^{n_p-1} \underbrace{\vec{\mathbf{u}}_k^T \mathbf{N}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k}_{f(\vec{\mathbf{u}}_k)} + \underbrace{\vec{\mathbf{u}}_k^T \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k}_{f(\vec{\mathbf{u}}_k)} + \underbrace{\vec{\mathbf{u}}_k^T \mathcal{R} \vec{\mathbf{u}}_k}_{f(\vec{\mathbf{u}}_k)} + \\ & + \sum_{i=0}^{n_p-1} \underbrace{2\mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{N}_i \vec{\mathbf{u}}_k}_{f(\mathbf{x}_k, \vec{\mathbf{u}}_k)} + 2 \underbrace{\mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \vec{\mathbf{u}}_k}_{f(\mathbf{x}_k, \vec{\mathbf{u}}_k)} + \\ & + \sum_{i=0}^{n_p-1} \underbrace{\mathbf{x}_k^T \mathbf{M}_i^T \mathbf{Q} \mathbf{M}_i \mathbf{x}_k}_{f(\mathbf{x}_k)} + \underbrace{\mathbf{x}_k^T \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p} \mathbf{x}_k}_{f(\mathbf{x}_k)}. \end{aligned} \quad (4.37)$$

Ďalej môžeme uvedené tri časti kriteriálnej rovnice napísať aj tak, že výrazy

prenásobíme z ľavej a pravej strany vektormi  $\vec{\mathbf{u}}_k$ , respektíve  $\mathbf{x}_k$ , a teda dostaneme

$$\begin{aligned}
 J_k = & \vec{\mathbf{u}}_k^T \underbrace{\left[ \sum_{i=0}^{n_p-1} \mathbf{N}_i^T \mathbf{Q} \mathbf{N}_i + \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} + \mathcal{R} \right]}_{\mathbf{H}} \vec{\mathbf{u}}_k + \\
 & + 2\mathbf{x}_k^T \underbrace{\left[ \sum_{i=0}^{n_p-1} \mathbf{M}_i^T \mathbf{Q} \mathbf{N}_i + \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} \right]}_{\mathbf{G}^T} \vec{\mathbf{u}}_k + \\
 & + \mathbf{x}_k^T \underbrace{\left[ \sum_{i=0}^{n_p-1} \mathbf{M}_i^T \mathbf{Q} \mathbf{M}_i + \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p} \right]}_{\mathbf{F}} \mathbf{x}_k,
 \end{aligned} \tag{4.38}$$

pričom v hranatých zátvorkách nám nezostali vstupy či stavy, ale iba matice, ktoré vieme vypočítať vopred, t. j. na základe dostupných predikčných matíc (modelu) a váhovania.

Nazvime prvú maticu v (4.38) Hessián  $\mathbf{H}$  (Hessova matica), kým druhú maticu nechajme teraz v transponovanej forme, teda  $\mathbf{G}^T$ , a potom výraz  $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$  je takzvaný gradient. Poslednú maticu, ktorá je závislá iba od stavu, si označíme  $\mathbf{F}$ . Význam Hessiánu a gradientu si bližšie objasníme v ďalších kapitolách, kde začneme účelovú funkciu aj minimalizovať.



Účelová funkcia v kompaktnom tvare na základe predikčných matíc je

$$J_k = \vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + 2\mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k + \mathbf{x}_k^T \mathbf{F} \mathbf{x}_k, \tag{4.39}$$

kde podľa predchádzajúceho odvodenia z Rov. (4.38) pre Hessián  $\mathbf{H}$  platí

$$\mathbf{H} = \sum_{i=0}^{n_p-1} \mathbf{N}_i^T \mathbf{Q} \mathbf{N}_i + \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p} + \mathcal{R}, \tag{4.40}$$

druhý člen  $\mathbf{G}^T$  má tvar

$$\mathbf{G}^T = \sum_{i=0}^{n_p-1} \mathbf{M}_i^T \mathbf{Q} \mathbf{N}_i + \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{N}_{n_p}, \tag{4.41}$$

respektíve taktiež platí, že

$$\mathbf{G} = \sum_{i=0}^{n_p-1} \mathbf{N}_i^T \mathbf{Q} \mathbf{M}_i + \mathbf{N}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p}, \quad (4.42)$$

a pre gradient teda platí  $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$ . Posledný člen účelovej funkcie,  $\mathbf{F}$ , má tvar

$$\mathbf{F} = \sum_{i=0}^{n_p-1} \mathbf{M}_i^T \mathbf{Q} \mathbf{M}_i + \mathbf{M}_{n_p}^T \mathbf{P} \mathbf{M}_{n_p}. \quad (4.43)$$

**Príklad 4.6.** Vypočítajte matice  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{F}$  pre diskrétny lineárny stavový model danú maticami

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 0 & 0.5 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0.05 \end{bmatrix} \quad \text{a} \quad \mathbf{C} = \begin{bmatrix} 1 & -1 \end{bmatrix},$$

pričom horizont predikcie je  $n_p = 5$  krokov a jednotlivé váhovacie matice sú zvolené ako  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$ ,  $\mathbf{R} = R = 0.1$  a  $\mathbf{P} = \mathbf{Q}$ .

V prvom kroku zadáme model, horizont a váhovacie matice. Predikčné matice môžeme vygenerovať pomocou funkcie `predikcia()`, ktorú sme si už vytvorili v predchádzajúcej kapitole:

Zdrojový kód 4.6. prvaucelovafunkcia.m

```

A=[1 2; 0 0.5]; B=[0; 0.05]; % Model
C=[-1 1]; % Model
np=5; % Horizont
Q=C'*C; % Vah. stavu
R=0.1; % Vah. vstupov
P=Q; % Konc. vah.
[M N]=predikcia(A,B,np); % Predikcne mat.

```

Ďalej zistíme počet stavov  $n_x$  a počet vstupov  $n_u$ , a vytvoríme maticu  $\mathcal{R}$  na základe Rov. (4.34) ako maticu s  $\mathbf{R}$  opakovane na hlavnej blokovej diagonále:

```

[nx nu]=size(B); % Pocet stavov a vstupov
RR=kron(eye(np),R); % Kaligraficke R

```

Príkaz `kron()` vypočíta tzv. Kroneckerov tenzorový súčet dvoch matíc.

Inicializujme si ďalej matice  $\mathbf{H}$ ,  $\mathbf{G}$ ,  $\mathbf{F}$  na základe nultého blokového riadku predikčných matíc  $\mathbf{M}_0 = \mathbf{I}$  a  $\mathbf{N}_0 = \mathbf{0}$ . Sústredíme sa teraz iba na prvý člen Rov. (4.40)–(4.43) kde vidíme operáciu sumácie a uvažujme rovnicu v čase  $k$  ( $i = 0$ ). Dosadením pre  $i = 0$  dostaneme:

```
H=zeros (np*nu,np*nu); % Prva nula N0'*Q*N0=0
G=zeros (np*nu,nx); % Prva nula N0'*Q*M0=0
F=Q; % Prva je M0*Q=Q;
```

Násobením  $\mathbf{N}_0 = \mathbf{0}$  pri  $i = 0$  pre  $\mathbf{H}$  a  $\mathbf{G}$  dostaneme matice núl, kým  $\mathbf{M}_0^T \mathbf{Q} \mathbf{M}_0 = \mathbf{I}^T \mathbf{Q} \mathbf{I} = \mathbf{Q}$ .

V ďalšej časti zdrojového kódu postupne sčítame zostávajúce blokové riadky predikčných matíc  $\mathbf{M}_i$  a  $\mathbf{N}_i$  pre  $i = 1, \dots, n_p - 1$ :

```
for i=1:np-1; % Prvy clen az do np-1
    H=H+N((i-1)*nx+1:i*nx,:)'*Q*N((i-1)*nx+1:i*nx,:);
    G=G+N((i-1)*nx+1:i*nx,:)'*Q*M((i-1)*nx+1:i*nx,:);
    F=F+M((i-1)*nx+1:i*nx,:)'*Q*M((i-1)*nx+1:i*nx,:);
end
```

Zostali nám už iba zvyšné výrazy Rov. (4.40)–(4.43), bez sumácie. Ku každej matici prirátame vplyv posledného stavu na základe posledného blokového riadku,  $\mathbf{M}_{n_p}$  a  $\mathbf{N}_{n_p}$  a v prípade  $\mathbf{H}$  ešte musíme prirátať aj  $\mathcal{R}$  (označené ako RR):

```
i=np; % Posledny riadok
H=H+N((i-1)*nx+1:i*nx,:)'*P*N((i-1)*nx+1:i*nx,:)+RR;
G=G+N((i-1)*nx+1:i*nx,:)'*P*M((i-1)*nx+1:i*nx,:);
F=F+M((i-1)*nx+1:i*nx,:)'*P*M((i-1)*nx+1:i*nx,:);
```

Výsledok môžeme vypísať do terminálu programu Matlab pomocou príkazov:

```
H, G, F % Vypis
```

a dostaneme matice:

```
H =
 0.1895    0.0609    0.0311    0.0054   -0.0092
 0.0609    0.1555    0.0298    0.0058   -0.0084
 0.0311    0.0298    0.1270    0.0066   -0.0069
 0.0054    0.0058    0.0066    0.1081   -0.0038
 -0.0092   -0.0084   -0.0069   -0.0038    0.1025

G =
 0.5156    1.9263
 0.3313    1.2713
```

```

0.1625      0.6363
0.0250      0.1039
-0.0500     -0.1922
F  =
6.0000      14.1563
14.1563     50.5752

```

□

Na zjednodušenie našej práce môžeme vytvoriť aj univerzálne využiteľnú funkciu na vypočítanie matíc **H**, **G** a **F**:

**Príklad 4.7.** Vytvorte univerzálne využiteľnú funkciu na výpočet parametrov kvadratickej úcelovej funkcie.

Výstupnými argumentami funkcie budú matice **H**, **G** a **F**, kým vstupnými sú model (**A**, **B**), horizont predikcie  $n_p$  a penalizačné matice (**Q**, **R** a **P**):

Zdrojový kód 4.7. ucelovafunkcia.m

```
function [H,G,F]=ucelovafunkcia(A,B,np,Q,R,P);
```

Nasledovne postupujeme na základe Pr. 4.6. Zistíme rozmery stavového vektora a vektora vstupu, a vytvoríme maticu  $\mathcal{R}$ . Ďalej postupujeme v zmysle Rov. (4.40)–(4.43):

```

[nx nu]=size(B);                                % Počet stavov a vstupov
[M N]=predikcia(A,B,np);                        % Predikcne mat.

RR=kron(eye(np),R);                            % Kaligraficke R

H=zeros(np*nu,np*nu);                          % Prva nula N0'*Q*N0=0
G=zeros(np*nu,nx);                            % Prva nula N0'*Q*M0=0
F=Q;                                         % Prva je M0*Q=Q;

for i=1:np-1;                                  % Prvy clen az do np-1
    H=H+N((i-1)*nx+1:i*nx,:)*Q*N((i-1)*nx+1:i*nx,:);
    G=G+N((i-1)*nx+1:i*nx,:)*Q*M((i-1)*nx+1:i*nx,:);
    F=F+M((i-1)*nx+1:i*nx,:)*Q*M((i-1)*nx+1:i*nx,:);
end
i=np;                                         % Posledny riadok
H=H+N((i-1)*nx+1:i*nx,:)*P*N((i-1)*nx+1:i*nx,:)+RR;
```

```

G=G+N( (i-1)*nx+1:i*nx, :)' * P * M( (i-1)*nx+1:i*nx, :) ;
F=F+M( (i-1)*nx+1:i*nx, :)' * P * M( (i-1)*nx+1:i*nx, :) ;

```

Na kontrolu novej funkcie `ucelovafunkcia()` môžeme zopakovať zadanie Pr. 4.6 a zavolať vytvorenú funkciu:

Zdrojový kód 4.8. kontrolauchelovej.m

```

A=[1 2; 0 0.5]; B=[0; 0.05]; % Model
C=[-1 1]; % Model
np=5; % Horizont
Q=C'*C; % Vahovanie stavu
R=0.1; % Vahovanie vstupov
P=Q; % Koncove vahovanie
[H, G, F]=ucelovafunkcia(A, B, np, Q, R, P) % Ucelova funkcia

```

Po spustení programu dostaneme rovnaké výsledky ako v predchádzajúcom príklade.

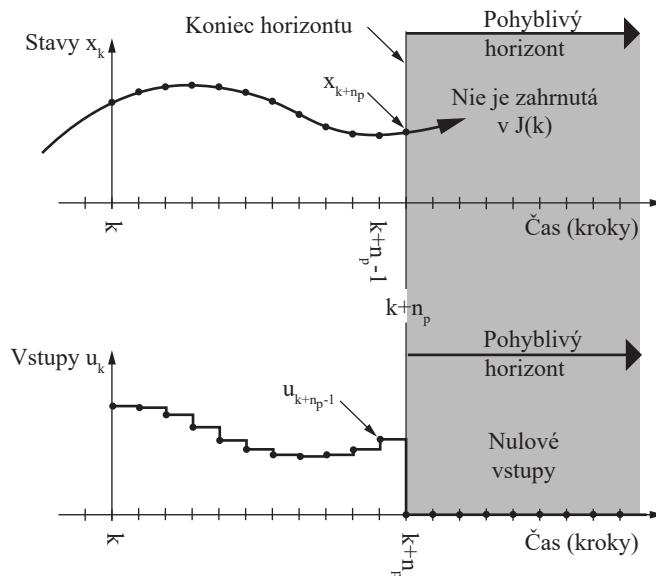
□

## 4.4 Zložená účelová funkcia

Rovnica (4.25), resp. (4.28) predstavuje účelovú funkciu definovanú na konečnom horizonte. Počítanie účelovej funkcie na konečnom horizonte je graficky znázornené na Obr. (4.13). Posledný vstup, ktorý je braný do úvahy pri výpočte hodnoty účelovej funkcie, je  $\mathbf{u}_{k+n_p-1}$  a ten vplyva na posledný stav  $\mathbf{x}_{k+n_p}$ . Ak v rámci horizontu priviedieme stav do nuly, ktorá je momentálne zvolená ako referencia ( $\mathbf{r} = 0$ ), nepotrebujeme už ďalšie akčné zásahy, môžeme teda predpokladať, že sú nulové. Nie je však garantované, že systém bude privedený do rovnovážneho stavu do konca horizontu. Potom účelová funkcia jednoducho predpokladá nulové vstupy a nepredikuje ďalšie stavy za horizontom.

Je dôležité rozlišovať to, čo predikuje účelová funkcia a ako by vyzeralo takéto riadenie v uzavretej slučke. Ak by sme rátali optimálne vstupy  $\tilde{\mathbf{u}}_k^*$  na základe tejto neúplnej účelovej funkcie, riadenie by neprestalo za horizontom nevyhnutne fungovať, pretože samotný horizont vždy posúvame dopredu. Znamenalo by to ale to, že naša účelová funkcia a teda aj celá riadiaca stratégia je za horizontom „slepá“.

Účelovú funkciu na konečnom horizonte môžeme použiť na výpočet stabilizujúcich vstupov, ak uvažujeme dostatočne dlhý horizont predikcie. V opačnom



Obr. 4.13. Ilustrácia kriteriálnej funkcie na konečnom horizonte<sup>7</sup>.

prípade, ak uvažujeme príliš krátky horizont, naša riadiaca slučka môže byť nestabilná. Ďalej v Kap. 5 budeme bližšie skúmať účinok príliš krátkeho horizontu na riadenie. Teraz skúsme rozmýšľať nad otázkou: aký je vlastne dostatočne dlhý horizont?

Najideálnejšie by bolo uvažovať vstupy a stavy na nekonečne dlhom horizonte  $n_p = \infty$  na základe účelovej funkcie

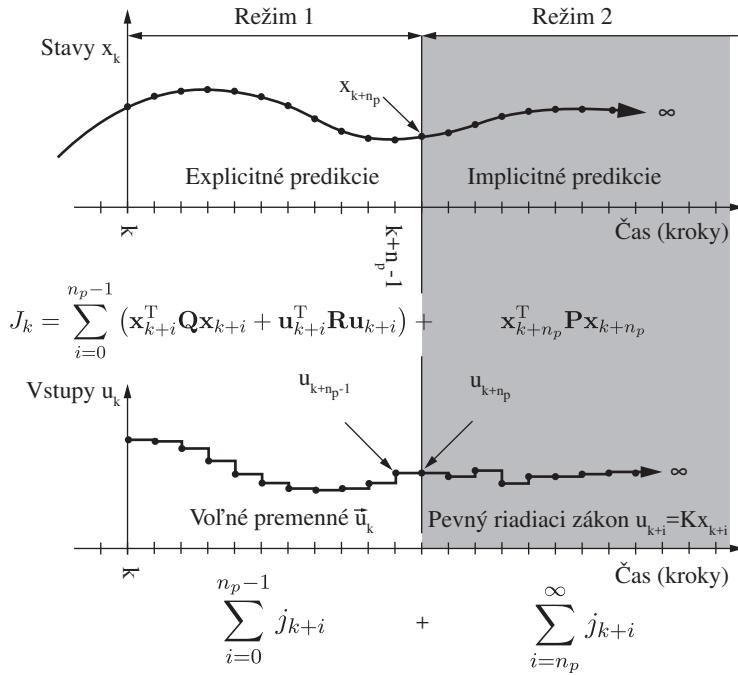
$$J_k = \sum_{i=0}^{\infty} j_{k+i} = \sum_{i=0}^{\infty} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right). \quad (4.44)$$

Na nekonečnom horizonte predikcia krok za krokom nie je možná, pretože by sme mali nekonečne veľa optimalizačných premenných a predikčné rovnice by sme taktiež rátali do nekonečna. Takúto účelovú funkciu teda nemôžeme prakticky realizovať.

Namiesto nekonečne veľa optimalizačných premenných však môžeme v prítomnosti obmedzení uvažovať riadenie s tzv. zloženým riadiacim zákonom<sup>8</sup> (Obr. 4.14). Jeho myšlienka je taká, že rozdelíme horizont na konečnú a nekonečnú časť

<sup>7</sup>Nulová hodnota vstupov za horizontom predikcie ilustruje skutočnosť, že tieto priamo nevystupujú v účelovej funkcií.

### <sup>8</sup>angl. dual mode control



Obr. 4.14. Ilustrácia kriteriálnej funkcie na nekonečnom horizonte.

podľa

$$J_k = \sum_{i=0}^{\infty} j_{k+i} = \sum_{i=0}^{n_p-1} j_{k+i} + \sum_{i=n_p}^{\infty} j_{k+i}, \quad (4.45)$$

kde prvý komponent účelovej funkcie vyjadri príspevok časti účelovej funkcie až do konca horizontu. Druhý komponent účelovej funkcie bude potom vyjadrovať kvalitu riadenia za horizontom.

Ako však možno vyjadriť príspevok účelovej funkcie za horizontom bez toho, aby sme použili nekonečne veľa optimalizačných premenných? Môžeme napríklad uvažovať tak, že za horizontom predikcie „prepne“ na riadenie s pevnou stavovou spätnou väzbou, kde môžeme riadiaci zákon vyjadriť v tvare

$$\mathbf{u}_{k+i} = \mathbf{K} \mathbf{x}_{k+i}, \quad (4.46)$$

a  $\mathbf{K}$  je stabilizujúce zosilnenie spätnej väzby, vstupy  $\mathbf{u}_{k+i}$  sú dané samotnými stavmi. Takéto zosilnenie môžeme vyjadriť na základe viacerých metód stavového

riadenia, ako sú napríklad  $H_2$  a  $H_\infty$ , alebo najčastejším riešením je vypočítať  $\mathbf{K}$  na základe LQ riadiaceho zákona, čo budeme ďalej predpokladať aj my.

Poznamenajme, že ak pevný riadiaci zákon uvažujeme bez záporného znamienka ako v Rov. (4.46), stále uvažujeme zápornú spätnú väzbu, t. j.

$$\mathbf{u}_{k+i} = -\mathbf{K}\mathbf{x}_{k+i}, \quad (4.47)$$

len predpokladáme, že znamienko je už zahrnuté do zosilnenia  $\mathbf{K}$ . V ďalšej diskusii budeme teda predpokladať takéto zosilnenie.

Máme teda dva režimy<sup>9</sup> riadenia. Prvý režim uvažuje voľné optimalizačné premenné a druhý predpokladá pevný riadiaci zákon, a teda platí, že

$$\mathbf{u}_{k+i} = \begin{cases} \text{voľné premenné} & i = 0, 1, \dots, n_p - 1 \\ \mathbf{K}\mathbf{x}_{k+i} & i = n_p, n_p + 1, \dots, \infty \end{cases} \quad \begin{array}{ll} \text{Režim 1,} & \\ \text{Režim 2.} & \end{array} \quad (4.48)$$

Treba si tiež dobre zapamätať jeden aspekt riadenia s obmedzeniami a zloženým riadiacim zákonom. Naša formulácia prediktívneho regulátora nikdy v skutočnosti „neprepne“ do druhého režimu riadenia. Prakticky teda nenastane situácia, keď algoritmus riadenia zrazu zmeníme z MPC na LQ. Tu môžeme – mylne – predpokladať, že počas procesu spätnoväzobného riadenia narazíme na koniec horizontu, keď už stačí využívať princíp LQ a stav násobit s pevnou určenou maticou. Ale pri riadení nikdy nenačíme na koniec horizontu, pretože horizont tlačíme pred sebou – práve to je koncept pohyblivého horizontu. Režim dva, teda pevná stavová spätná väzba, je potrebná iba na to, aby sme vyjadrili účinok riadenia za horizontom v samotnej účelovej funkcií a potom podľa toho rátali voľné optimalizačné premenné  $\vec{\mathbf{u}}_k$ .

Vráťme sa teraz k Rov. (4.45) a zamyslime sa nad tým, ako môžeme zahrnúť účinok pevnej spätej väzby do funkcie. Už vieme, že na to vôbec nepotrebujeme voľné premenné, keďže vstupy závisia od stavov podľa Rov. (4.46). Predstavme si teda také váhovanie  $\mathbf{P}$  posledného stavu  $\mathbf{x}_{k+n_p}$ , ktoré vyjadri kvalitu riadenia v režime dva, t. j. za horizontom predikcie, na základe LQ riadenia

$$J_{n_p} = \sum_{i=n_p}^{\infty} j_{k+i} = \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}, \quad (4.49)$$

a naša konečná účelová funkcia je teda stále daná Rov. (4.25). Z toho je zrejmé, že váhovanie posledného stavu  $\mathbf{P}$  môžeme sice ľubovoľne zvoliť, ale oplatilo by sa hlbšie zamyslieť nad tým, ako ho môžeme vypočítať tak, aby bola splnená Rov. (4.49).

<sup>9</sup>angl. mode

#### 4.4.1 Matica váhovania koncového stavu

V režime 2 bude pre nás systém platíť ustálený LQ riadiaci zákon. To znamená, že pre model daný maticami  $\mathbf{A}$  a  $\mathbf{B}$  a pre zvolené penalizácie stavov  $\mathbf{Q}$  a  $\mathbf{R}$ , môžeme jednoznačne vypočítať ustálenú hodnotu optimálneho LQ zosilnenie  $\mathbf{K}$ .



Predpokladajme, že ak maticu  $\mathbf{P}$  vypočítame ako riešenie rovnice

$$\mathbf{P} - (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P}(\mathbf{A} + \mathbf{B}\mathbf{K}) = \mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}, \quad (4.50)$$

potom platí, že

$$\sum_{i=n_p}^{\infty} (\mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i}) = \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}. \quad (4.51)$$

Inými slovami, ak model, váhy a LQ zosilnenie dosadíme do Rov. (4.50), posledný člen účelovej funkcie (4.25) vyjadri kvalitu riadenia za horizontom až do nekonečna. „Zvláštna“ rovnica, ktorá umožňuje výpočet koncového váhovania je tzv. diskrétna algebraická Riccatiho rovnica.

#### Jacopo Riccati a jeho rovnica

Riccatiho rovnica je typ matematického problému, ktorý sa často vyskytuje pri riešení problémov optimálneho riadenia a optimálneho odhadu. V kontexte našej diskusie nás zaujíma tzv. algebraická forma Riccatiho rovnice, ktorá je určená na riešenie dynamických problémov v ustálenom stave. Všeobecný tvar Riccatiho algebraickej rovnice<sup>a</sup> pre diskrétny čas (DARE<sup>b</sup>) je daný ako

$$\mathbf{P} - \mathbf{A}^T \mathbf{P} \mathbf{A} + \mathbf{A}^T \mathbf{P} \mathbf{B} (\mathbf{R} + \mathbf{B}^T \mathbf{P} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P} \mathbf{A} - \mathbf{Q} = 0. \quad (4.52)$$

Rov. 4.50 predstavuje variant DARE v tzv. Josephovom tvare.

Jacopo Francesco Riccati (1676–1754), Obr. 4.15), bol známy taliansky matematik, ktorý najprv získal doktorát z práva, no neskôr sa rozhodol študovať matematickú analýzu [117].

<sup>a</sup>Riccati pôvodne riešil nelineárnu diferenciálnu rovnicu prvého rádu s kvadratickým členom.

<sup>b</sup>angl. discrete algebraic Riccati equation (DARE).



Obr. 4.15. Jacopo Francesco Riccati.

Všimnime si, že na výpočet váhovania koncového stavu  $\mathbf{P}$  na základe Rov. (4.50) potrebujeme aj LQ zosilnenie  $\mathbf{K}$ . Prvým krokom pre jeho získanie teda musí byť výpočet LQ zosilnenia.

Z teórie riadenia diskrétnych lineárnych systémov vieme [85, 9, 51], že ustálené LQ-optimálne zosilnenie  $\mathbf{K}$  vypočítame zo vzťahu<sup>10</sup>

$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^T \mathbf{P} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P} \mathbf{A}, \quad (4.53)$$

kde poznáme všetky výrazy, ktoré sa zhodujú s našim zvyčajným značením. Význam matice  $\mathbf{P}$  je rovnaký ako v predchádzajúcej diskusii, keďže vyjadruje hodnotu kvadratickej účelovej funkcie na nekonečnom horizonte. Všimnime si ale nepríjemný paradox: na výpočet LQ zosilnenia podľa Rov. (4.53) by sme hned potrebovali  $\mathbf{P}$ , pričom túto môžeme vypočítať na základe diskrétnej algebraickej Riccatiho rovnice, ktorá zase využíva LQ zosilnenie. Máme tu klasickú kontradikciu: Čo vzniklo najprv, vajce alebo sliepka?<sup>11</sup>

Vyjasnením zdanlivého paradoxu je, že Riccatiho rovnicu musíme vypočítať naraz pre oba neznáme výrazy, a tak získať aj LQ optimálne zosilnenie  $\mathbf{K}$  ako aj maticu váhovania koncového stavu,  $\mathbf{P}$ . Môžeme to pochopíť aj tak, že vedľajším produkтом výpočtu LQ zosilnenia je matica  $\mathbf{P}$ . Z prostredia MATLABu vieme, že na základe príkazu `dlqr()` dostaneme LQ zosilnenie. Funkcia `dlqr()` pritom vlastne rieši Riccatiho rovnicu a v prípade jej volania v tvare

<sup>10</sup>Tu bez dôkazu.

<sup>11</sup>Z hľadiska evolúcie je to jednoduché: vajcia. Jaštery sa rozmnožovali pomocou vajíčok pred tým ako vtáky vôbec existovali [38].

$$[\mathbf{K}, \mathbf{P}] = \mathbf{dlqr}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$$

dostaneme aj maticu váhovania koncového stavu,  $\mathbf{P}$ .

Na riešenie Rov 4.52 existuje mnoho vyspelých numerických prístupov [56, 119, 120, 64] ale ich opis nie je predmetom tejto učebnice. Najjednoduchšia metóda je iteračná, kde označíme všetky  $\mathbf{P}$  indexom  $k$ . Metóda vyžaduje inicializáciu matice  $\mathbf{P}$  ako  $\mathbf{P}_0 \succ 0$ ,  $\mathbf{P}_0 = \mathbf{P}_0^T$ , napríklad  $\mathbf{P}_0 = \mathbf{I}$ . Jednu maticu  $\mathbf{P}_k$  necháme na ľavej strane rovnice, a ostatné premiestníme na jej pravú stranu, čím dostaneme tvar

$$\mathbf{P}_k = \mathbf{A}^T \mathbf{P}_k \mathbf{A} - \mathbf{A}^T \mathbf{P}_k \mathbf{B} \underbrace{(\mathbf{R} + \mathbf{B}^T \mathbf{P}_k \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P}_k \mathbf{A}}_{\mathbf{K}_k} + \mathbf{Q}, \quad (4.54)$$

v ktorom si všimnime, že sa vyskytuje aj LQ zosilnenie z Rov. (4.53), ktoré taktiež závisí od  $\mathbf{P}_k$ , teda

$$\mathbf{K}_k = (\mathbf{R} + \mathbf{B}^T \mathbf{P} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P}_k \mathbf{A}. \quad (4.55)$$

Riešením Rov. (4.54)–(4.55) pre  $k = 0, 1, 2, \dots, n$  iterácií dostaneme numerickú approximáciu  $\mathbf{P}$  aj  $\mathbf{K}$ . Pritom  $\mathbf{P}$  bude predstavovať ustálené riešenie algebraickej Riccatiho rovnice. Okrem toho, že tento prístup je výpočtovo neefektívny, vplyvom maticovej inverzie a iteratívneho procesu sa numerické chyby naakumulujú a výsledok môže byť nepresný alebo v istých prípadoch úplne divergovať.

#### 4.4.2 Ljapunova rovnica

Aby sme dokázali, že matica koncového váhovania  $\mathbf{P}$  vyjadruje komponent zloženej účelovej funkcie od konca horizontu až do nekonečna, budeme potrebovať aj vedieť čo je Ljapunova rovnica.

##### Aleksandr Ljapun a jeho rovnica

Ljapunova rovnica je štandardná maticová algebraická rovnica známa z teórie obyčajných diferenciálnych rovníc, ktorá sa používa na dôkaz stability rovnovážneho riešenia diferenciálnej rovnice. Neformálna definícia hovorí, že Ljapunova funkcia má kladné hodnoty všade okrem rovnovážneho stavu a klesá (alebo nerastie) pri každej trajektórii diferenciálnej rovnice. Ljapunova rovnica je pomenovaná po významnom ruskom matematikovi a fyzikovi Aleksandrovi Michailovičovi Ljapunovi (Obr. 4.16). V anglickej literatúre sa používa prepis jeho mena v tvare „Lyapunov“.



Obr. 4.16. Alexander Michajlovič Ljapunov (1857 až 1918).

Všeobecný tvar spojitej Ljapunovej rovnice je [21]

$$\mathbf{A}_c^T \mathbf{P} + \mathbf{P} \mathbf{A}_c + \mathbf{Q} = \mathbf{0}, \quad (4.56)$$

kde  $\mathbf{A}_c$ ,  $\mathbf{P}$  a  $\mathbf{Q}$  sú štvorcové matice, pričom  $\mathbf{P}$  a  $\mathbf{Q}$  sú symetrické ( $\mathbf{A}_c, \mathbf{P}, \mathbf{Q} \in \mathbb{R}^{n \times n}$ ,  $\mathbf{P} = \mathbf{P}^T$ ,  $\mathbf{Q} = \mathbf{Q}^T$ ). Interpretácia rovnice je taká, že ak je lineárny systém daný autonómnym modelom

$$\dot{\mathbf{x}}(t) = \mathbf{A}_c \mathbf{x}(t) \quad (4.57)$$

a definujme kvadratickú funkciu

$$V(\mathbf{x}(t)) = \mathbf{x}(t)^T \mathbf{P} \mathbf{x}(t), \quad (4.58)$$

kde  $V(\mathbf{x}(t))$  musí byť klesajúca teda mať zápornú deriváciu a potom

$$\dot{V}(\mathbf{x}(t)) = (\mathbf{A}_c \mathbf{x}(t))^T \mathbf{P} \mathbf{x}(t) + \mathbf{x}(t)^T \mathbf{P} (\mathbf{A}_c \mathbf{x}(t)) = -\mathbf{x}(t)^T \mathbf{Q} \mathbf{x}(t). \quad (4.59)$$

To znamená, že ak použijeme analógiu mechanických systémov, kde Rov. 4.57 opisuje dynamiku mechanického systému, tak  $\mathbf{x}(t)^T \mathbf{P} \mathbf{x}(t)$  predstavuje zovšeobecnenú energiu, a  $\mathbf{x}(t)^T \mathbf{Q} \mathbf{x}(t)$  predstavuje zovšeobecnenú disipáciu, resp. rozptýlenie energie, ako napríklad viskózne tlmenie [21].

Ďalšia interpretácia rovnice súvisí so stabilitou systému a hovorí, že pre každé kladne definitné  $\mathbf{Q}$  ( $\mathbf{Q} \succ 0$ ) nájdeme také kladne definitné  $\mathbf{P}$  ( $\mathbf{P} \succ 0$ ) podľa Rov. (4.56) vtedy a len vtedy, ak systém podľa Rov. (4.57) je globálne asymptoticky stabilný. Kvadratická funkcia  $V(\mathbf{x}(t)) = \mathbf{x}(t)^T \mathbf{P} \mathbf{x}(t)$  je potom využiteľná na overenie stability a nazývame ju Ljapunova funkcia. Ljapunova stabilita je podrobnejšie rozobraná v Kap. 8.

Ljapunova rovnica pre diskrétny systém

$$\mathbf{x}_{k+1} = \mathbf{A} \mathbf{x}_k \quad (4.60)$$

je všeobecne daná ako

$$\mathbf{A}^T \mathbf{P} \mathbf{A} - \mathbf{P} + \mathbf{Q} = \mathbf{0}. \quad (4.61)$$

Podobne ako pri spojitej verzii to môžeme pochopiť tak, že pre ľubovoľné  $\mathbf{Q} \succ 0$  môžeme nájsť  $\mathbf{P} \succ 0$  na základe Rov. (4.61), ak (4.60) je globálne asymptoticky stabilná. Tak ako predtým,  $V(\mathbf{x}_k) = \mathbf{x}_k^T \mathbf{P} \mathbf{x}_k$  je Ljapunova funkcia.

Za všeobecnú a ľubovoľnú maticu  $\mathbf{Q}$  z Rov. (4.61) si teraz dosadíme výraz  $\mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}$  podľa

$$\underbrace{\mathbf{A}^T}_{\mathbf{A} + \mathbf{B} \mathbf{K}} \underbrace{\mathbf{P}}_{\mathbf{A} + \mathbf{B} \mathbf{K}} \underbrace{\mathbf{A}}_{\mathbf{A} + \mathbf{B} \mathbf{K}} - \mathbf{P} + \underbrace{\mathbf{Q}}_{\mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}} = \mathbf{0}, \quad (4.62)$$

túto rovnicu prenásobíme  $-\mathbf{I}$  (zmeníme znamienko)

$$\mathbf{P} - (\mathbf{A} + \mathbf{B} \mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B} \mathbf{K}) - (\mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}) = \mathbf{0}, \quad (4.63)$$

a posledný výraz premiestnime s opačným znamienkom na pravú stranu a dosta-neme

$$\mathbf{P} - (\mathbf{A} + \mathbf{B} \mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B} \mathbf{K}) = (\mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K})$$

čo je vlastne totožný výraz s našim počiatočným predpokladom z Rov. (4.50)—dostali sme Josephov tvar diskrétnej algebraickej Riccatiho rovnice.

Všimnime si teda podobnosť Ljapunovej rovnice s algebraickou Riccatiho rovnicou. Prvý hlavný rozdiel oproti Ljapunovej rovnici je prítomnosť vstupov cez maticu vstupov  $\mathbf{B}$ . Intuitívne preto môžeme tvrdiť, že Riccatiho rovnica je variant Ljapunovej rovnice pri uvažovaní riadenia. Riccatiho rovnica je príbuzná Ljapunovej rovnici a pri istých podmienkach ich môžeme vzájomne transformovať. Kým Ljapunova rovnica vychádza z podmienok stability, pomocou Riccatiho rovnice dokážeme počítať zosilnenia pre optimálne riadenie a odhad.

#### 4.4.3 Váhovanie podľa diskrétnej algebraickej Riccatiho rovnice

Uvedený predpoklad podľa Rov. (4.50) musíme teraz dokázať. Jednoducho povedané, z diskrétnej algebraickej Riccatiho rovnice musíme dostať Rov. (4.51), ktorá hovorí o tom, že posledný člen zloženej účelovej funkcie,  $\mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}$ , je ekvivalentný príspevku účelovej funkcie pre režim dva zloženého riadiaceho zákona, vyjadrujúcemu kvalitu riadenia od  $n_p$  až do nekonečna.

Označme si teraz čas  $k + n_p$  za počiatočný (nultý) čas, kde máme stav  $\mathbf{x}_0 = \mathbf{x}_k$  (namiesto  $\mathbf{x}_{k+n_p}$ ), aby sme si v ďalšom odvodení zjednodušili matematické

označenie. Týmto sme presunuli index  $k$  zo začiatku horizontu až na koniec  $n_p$  krokov, t. j. namiesto Rov. 4.51 máme

$$\sum_{k=0}^{\infty} (\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k) = \mathbf{x}_0^T \mathbf{P} \mathbf{x}_0. \quad (4.64)$$

Po riešení DARE je dynamika systému v režime 2 v uzavretom tvare daná ako  $(\mathbf{A} + \mathbf{B}\mathbf{K})$ . Dosadme si ďalej do všeobecnej diskrétnej Ljapunovej funkcie (4.61). Výrazy v nej prenásobme zľava s  $\mathbf{x}_k^T$  a sprava s  $\mathbf{x}_k$ , aby sme dostali

$$\mathbf{x}_k^T \mathbf{P} \mathbf{x}_k - \mathbf{x}_k^T (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B}\mathbf{K}) \mathbf{x}_k = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{x}_k^T \mathbf{K}^T \mathbf{R} \mathbf{K} \mathbf{x}_k. \quad (4.65)$$

Nezabudnime, že teraz používame zjednodušené indexovanie a začiatok sme presunuli na koniec horizontu.

Definujme si ďalej Ljapunovu funkciu  $V(\mathbf{x}_k)$ , ktorá vyjadruje hodnotu kritéria  $\mathbf{x}_k^T \mathbf{P} \mathbf{x}_k$  v ľubovoľnej časovej vzorke  $k$  ako

$$V(\mathbf{x}_k) = \mathbf{x}_k^T \mathbf{P} \mathbf{x}_k, \quad (4.66)$$

a využime známe faktky, že vstupy v druhom móde počítame podľa LQ riadiaceho zákona, a ďalšie stavy vieme vypočítať podľa stavovej rovnice, a teda platí<sup>12</sup>, že

$$\mathbf{u}_k = \mathbf{K} \mathbf{x}_k, \quad (4.67)$$

$$\mathbf{x}_{k+1} = \mathbf{A} \mathbf{x}_k + \mathbf{B} \mathbf{u}_k, \quad (4.68)$$

z čoho je potom jasné, že na základe stavového modelu pre ďalší stav v uzavretej riadiacej slučke platí

$$\mathbf{x}_{k+1} = (\mathbf{A} + \mathbf{B}\mathbf{K}) \mathbf{x}_k. \quad (4.69)$$

Ak vieme vyjadriť LQ riadiaci zákon v uzavretej slučke a vypočítať ďalší stav, tak potom vieme vypočítať aj ďalšie hodnoty funkcie  $V(\mathbf{x}_k)$  tak, že dosadíme za stav  $\mathbf{x}_{k+1} = (\mathbf{A} + \mathbf{B}\mathbf{K}) \mathbf{x}_k$ , na základe čoho pre  $V(\mathbf{x}_{k+1})$  dostaneme

$$V(\mathbf{x}_{k+1}) = \mathbf{x}_{k+1}^T (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B}\mathbf{K}) \mathbf{x}_k. \quad (4.70)$$

Dosadme teraz  $V(\mathbf{x}_k)$  z Rov. (4.66) a  $V(\mathbf{x}_{k+1})$  z (4.70) do Ljapunovej rovnice (4.65) a dosadme LQ riadiaci zákon za  $\mathbf{u}_k = \mathbf{K} \mathbf{x}_k$  a jeho transpozíciu podľa

$$\underbrace{\mathbf{x}_k^T \mathbf{P} \mathbf{x}_k}_{V(\mathbf{x}_k)} - \underbrace{\mathbf{x}_k^T (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B}\mathbf{K}) \mathbf{x}_k}_{V(\mathbf{x}_{k+1})} = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \underbrace{\mathbf{x}_k^T \mathbf{K}^T \mathbf{R} \mathbf{K} \mathbf{x}_k}_{\mathbf{u}_k^T \mathbf{u}_k}, \quad (4.71)$$

<sup>12</sup>pozri poznámku a značenie zápornej spätej väzby na s. 205

z čoho vyplýva

$$V(\mathbf{x}_k) - V(\mathbf{x}_{k+1}) = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k. \quad (4.72)$$

Túto rovnicu vieme zosumovať pre časy  $k = 0, 1, 2, 3, \dots, \infty$ , a napísat ako

$$V(\mathbf{x}_0) - \lim_{k \rightarrow \infty} V(\mathbf{x}_k) = \sum_{k=0}^{\infty} (\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k), \quad (4.73)$$

pretože po sumácii pravej časti až do konečného času  $k = 1, 2, \dots, i$  (do  $i = \infty$ ) vlastne dostaneme

$$\begin{aligned} \{V(\mathbf{x}_0) - V(\mathbf{x}_1)\}_{k=0} + \{V(\mathbf{x}_1) - V(\mathbf{x}_2)\}_{k=1} + \{V(\mathbf{x}_2) - V(\mathbf{x}_3)\}_{k=2} + \dots \\ \dots + \{V(\mathbf{x}_{i-2}) - V(\mathbf{x}_{i-1})\}_{k=i-1} + \{V(\mathbf{x}_{i-1}) - V(\mathbf{x}_i)\}_{k=i} \end{aligned}$$

a všetko vypadne okrem prvého a posledného člena, t. j.  $V(\mathbf{x}_0)$  a pri  $k = \infty$   $\lim V(\mathbf{x}_k)$ . Už vidíme, že pravú stranu Ljapunovej rovnice sme dokázali pretransformovať do žiadaného tvaru, v ktorom predstavuje našu účelovú funkciu na nekonečnom horizonte.

Taktiež vieme, že na základe pôvodnej definície z Rov. (4.70), ktorá hovorí, že  $V(\mathbf{x}_{k+1}) = \mathbf{x}_k^T (\mathbf{A} + \mathbf{B} \mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B} \mathbf{K}) \mathbf{x}_k$ , potom tiež musí podľa rekurzie aj pre budúcu vzorku  $k$  platíť

$$V(\mathbf{x}_{k+i}) = \mathbf{x}_k^T (\mathbf{A} + \mathbf{B} \mathbf{K})^{iT} \mathbf{P} (\mathbf{A} + \mathbf{B} \mathbf{K})^i \mathbf{x}_k, \quad (4.74)$$

kde s každým  $i$  vypočítame ďalšiu hodnotu Ljapunovej funkcie v budúcnosti. Akú hodnotu bude však mať  $V(\mathbf{x}_k)$  ak  $k \rightarrow \infty$ ?

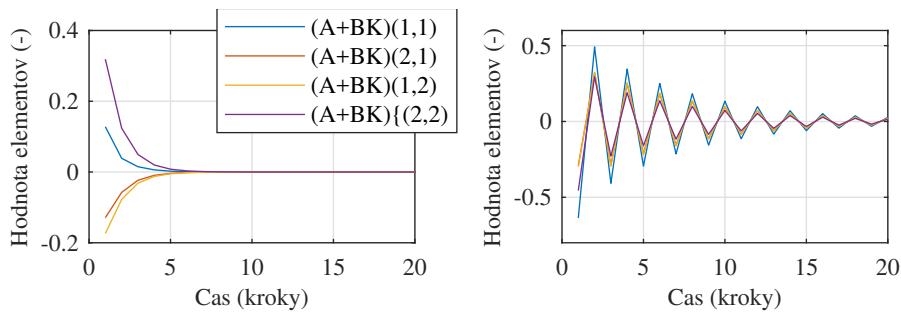
Vieme, že  $\mathbf{P}$  bude kladne definitná, teda nenulová matica. Stavy taktiež nemôžeme všeobecne považovať za nulové, ale v nekonečnom čase pri stabilnom systéme by mali konvergovať do nuly. Teraz skúsme numericky prešetriť aké hodnoty bude mať výraz  $(\mathbf{A} + \mathbf{B} \mathbf{K})$  pre časy  $k \rightarrow \infty$ , ak je systém je stabilný.

**Príklad 4.8.** Vypočítajte hodnotu výrazu  $(\mathbf{A} + \mathbf{B} \mathbf{K})^k$  pre  $k \rightarrow \infty$ . Na testovanie výsledkov použite náhodne vygenerovaný stabilný diskrétny stavový model.

Najprv vygenerujeme náhodnú lineárny systém pomocou príkazu `drss()`, potom si uložíme matice dynamiky  $\mathbf{A}$  a matice vstupu  $\mathbf{B}$  do premenných  $\mathbf{A}$  a  $\mathbf{B}$ , a zistíme rozmer stavového vektora:

Zdrojový kód 4.9. konvergenciadonuly.m

```
sys=drss(2); % Nahodna stab. sustava
A=sys.A; B=sys.B; % Matice A, B
[nx nu]=size(B); % Rozmer sustavy
```



Obr. 4.17. Hodnoty prvkov matice dynamiky systému riadeného stavovou spätnou väzbou s ustáleným zosilnením.

Zvolíme si jednotkovú penalizáciu stavov a vstupov a diskrétnu LQ zosilnenie vypočítame pomocou príkazu `dlqr()`:

```
Q=eye(nx); R=eye(nu); % Penalizacia
K=-dlqr(A,B,Q,R); % LQ zakon
```

Ďalej si pre zvolenú dĺžku simulácie vypočítame hodnoty prvkov autonómnej stavovej matice  $(\mathbf{A} + \mathbf{B}\mathbf{K})^k$ :

```
for k=1:20 % Zvysujuce k
    Psi=(A+B*K)^k; % Dynamika riad. sústavy
    PSI(k,:)=[Psi(1,1) Psi(2,1) Psi(1,2) Psi(2,2)];
end
```

a vykreslíme priebeh prvkov matíc do obrázku

```
plot(PSI) % Vykreslenie vysledku
grid on % Mriezka
xlabel('Cas (kroky)') % Os X
ylabel('Hodnota elementov (-)') % Os Y a legenda
legend('(A+BK) (1,1)', '(A+BK) (2,1)', '(A+BK) (1,2)', '(A+BK) (2,2)')
```

Pre náhodne generované stabilné systémy a stabilnú maticu  $(\mathbf{A} + \mathbf{B}\mathbf{K})$  dostaneme vždy také prvky matice  $(\mathbf{A} + \mathbf{B}\mathbf{K})^k$ , ktoré so zvyšujúcim  $k$  konvergujú k nule (Obr. 4.17). Vysvetlenie je jednoduché: ak je systém v uzavretej slučke stabilný, maticová spätná väzba musí stavu priviesť do nuly za nejaký čas.

□

Na základe predchádzajúceho príkladu a úvahy teda môžeme tvrdiť, že

$$(\mathbf{A} + \mathbf{B}\mathbf{K})^k \rightarrow \mathbf{0} \quad (4.75)$$

pre  $k \rightarrow \infty$  a potom platí aj

$$V(\mathbf{x}_{k+i}) = \mathbf{x}_k^T (\mathbf{A} + \mathbf{B}\mathbf{K})^{iT} \mathbf{P} (\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x}_k \rightarrow \mathbf{0}. \quad (4.76)$$

pre  $i \rightarrow \infty$ . Potom z Rov. (4.73) vypadne druhý výraz  $\lim V(\mathbf{x}_k)$  na ľavej strane, a zostane nám iba

$$V(\mathbf{x}_0) = \sum_{k=0}^{\infty} \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right). \quad (4.77)$$

Čomu sa rovná  $V(\mathbf{x}_0)$ ? Dosadíme Ljapunovu funkciu pre  $\mathbf{x}_0$  do predošej rovnice a zostane nám

$$\mathbf{x}_0^T \mathbf{P} \mathbf{x}_0 = \sum_{k=0}^{\infty} \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right), \quad (4.78)$$

čo je identické s Rov. (4.64).

Tým sme dokázali, že ak počítame maticu koncového váhovania  $\mathbf{P}$  ako ustálené riešenie diskrétnej algebraickej Riccatiho rovnice z Rov. (4.50), potom môžeme nahradíť hodnotu nekonečnej účelovej funkcie v móde 2 riadenia za konečný výraz  $\mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}$ .

#### 4.4.4 Hľadanie koncového váhovania pri známom LQ zosilnení

V prípade, že už poznáme LQ zosilnenie, stačí vyriešiť Ljapunovu rovnicu na nájdanie váhovania koncového stavu. Riešenie, tak ako v prípade Riccatiho rovnice, nie je úplne triviálne a existuje niekoľko alternatívnych metód [8, 7, 120]. Diskrétnu Ljapunovu rovnicu môžeme vyriešiť napríklad nájdením vlastných čísel pomocou tzv. QR algoritmu [80]. Najjednoduchšou numerickou metódou riešenia Ljapunovej rovnice je iteráčná metóda rekurzívneho dosadenia jednotlivých hodnôt na základe počiatočného odhadu. Prístup má niekoľko nevýhod, napríklad výpočtovo nie je efektívny a je aj nepresný, no jeho programová realizácia je jednoduchá.

Vychádzame z Rov (4.50) a výraz  $\mathbf{P}$  necháme na pravej strane rovnice, pričom  $(\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B}\mathbf{K})$  premiestnime na pravú stranu, t. j.

$$\mathbf{P} = (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P} (\mathbf{A} + \mathbf{B}\mathbf{K}) + \mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}. \quad (4.79)$$

Incializujme teraz  $\mathbf{P}_0$  tak, aby podmienky váhovania koncového stavu v účelovej funkcie boli dodržané: musíme vybrať symetrickú kladne definitnú maticu

$(\mathbf{P}_0 = \mathbf{P}_0^T, \mathbf{P}_0 \succ 0)$ . Najjednoduchšou voľbou, ktorá spĺňa tieto podmienky, je jednotková matica, teda môžeme voliť  $\mathbf{P}_0 = \mathbf{I}$  a potom to rekurzívne dosadiť do

$$\mathbf{P}_{i+1} = (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{P}_i (\mathbf{A} + \mathbf{B}\mathbf{K}) + \mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}, \quad (4.80)$$

pre  $i = 0, 1, \dots, n$ . Matica  $\mathbf{P}_i$  bude postupne konvergovať k riešeniu pre  $\mathbf{P}$ .

Na riešenie Ljapunovej rovnice MATLAB používa Bartels-Stewartov algoritmus [8], ktorý je založený na Schurovom rozklade matíc [68]. Všeobecnú diskrétnu Ljapunovu rovnicu podľa 4.61 môžeme riešiť pomocou príkazu

`X=dlyap (A, Q)`

kde vstupné premenné majú všeobecný význam a na prvý pohľad teda nesúvisia s našim konkrétnym problémom. Podľa manuálu funkcie `dlyap()`, algoritmus rieši diskrétnu Ljapunovu rovnicu

$$\bar{\mathbf{A}} \bar{\mathbf{X}} \bar{\mathbf{A}}^T - \bar{\mathbf{X}} + \bar{\mathbf{Q}} = 0. \quad (4.81)$$

Porovnávaním Rov. (4.81) a (4.50) zistíme, že ide o podobnú štruktúru a na riešenie môžeme preto použiť nasledujúci prepis:

$$\bar{\mathbf{X}} = \mathbf{P}, \quad (4.82)$$

$$\bar{\mathbf{A}} = (\mathbf{A} + \mathbf{B}\mathbf{K})^T, \quad (4.83)$$

$$\bar{\mathbf{Q}} = \mathbf{Q} + \mathbf{K}^T \mathbf{R} \mathbf{K}, \quad (4.84)$$

a teda zavolať funkciu so vstupnými argumentmi

`P = dlyap ( [A+B*K]', [Q+K'*R*K] )`

aby sme na výstupe dostali žiadanú maticu váhovania koncového stavu,  $\mathbf{P}$ .

**Príklad 4.9.** Vypočítajte váhovanie koncového stavu  $\mathbf{P}$  riešením Ljapunovej rovnice (4.50) pomocou iteračnej metódy. Použite model

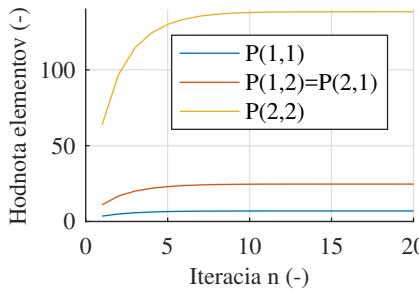
$$\mathbf{A} = \begin{bmatrix} 1.1 & 2.0 \\ 0.0 & 0.95 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0.0787 \end{bmatrix}, \quad (4.85)$$

a jednotkové penalizačné matice stavov a vstupu.

Na začiatku si definujeme model, váhy a rozmer problému:

Zdrojový kód 4.10. prikladljapunov.m

```
A=[1.1 2; 0 0.95]; B=[0; 0.0787]; % Model
[nx nu]=size(B); % Rozmery
Q=eye(nx); R=eye(nu); % Vahovanie
K=-dlqr (A,B,Q,R); % LQ zosilnenie
```



Obr. 4.18. Konvergencia prvkov matice  $\mathbf{P}$  pri iteratívnej metóde výpočtu Ljapunovej rovnice.

Ďalej si určíme počet iterácií a inicializujeme riešenie Ljapunovovej rovnice ako jednotkovú maticu:

```
n=20; % Iteracie
P=eye(2); % Inicializacia
```

Rov. 4.80 implementujeme v cykle a jej jedinečné prvky matice uložíme do postupnosti  $\mathbf{Px}$ :

```
for i=1:n % Iterujeme
P=(A+B*K)'*P*(A+B*K)+Q+K'*R*K % Lyapunova rovnica
Px(:,i)=[P(1,1);P(1,2);P(2,2)]; % Ulozenie do vektoru
end
```

Po spustení programu príkazový riadok MATLABu vypíše maticu  $\mathbf{P}$  a vidíme, že jej hodnoty pomaly ustália na:

```
P =
6.9301 24.6632
24.6632 138.3125
```

Tento proces si môžeme aj graficky znázorniť vykreslením závislosti hodnoty prvkov na iterácií:

```
plot(Px') % Vykreslit elementy
grid on % Mriezka
legend('P(1,1)', 'P(1,2)=P(2,1)', 'P(2,2)') % Os x
xlabel('Iteracia i (-)') % Os x
ylabel('Hodnota elementov (-)') % Os y
```

Na Obr. 4.18 vidíme, ako postupne hodnoty prvkov matice konvergujú k ustáleným hodnotám a správnemu riešeniu. V tomto prípade – aspoň vizuálne – stačilo

desať iterácií na dosiahnutie použiteľného výsledku. Žiaľ iteračné numerické riešenie môže byť veľmi nepresné pri určitých modeloch a penalizáciách.

□

**Príklad 4.10.** Vytvorte z iteračnej metódy riešenia Ljapunovej rovnice univerzálnu funkciu na výpočet váhovania koncového stavu.  $\mathbf{P}$ .

Výstupom funkcie bude matica  $\mathbf{P}$ , pričom vstupnými argumentami budú matice modelu ( $\mathbf{A}, \mathbf{B}$ ), váhy ( $\mathbf{Q}, \mathbf{R}$ ) a počet iterácií:

Zdrojový kód 4.11. ljapiter.m

```
function P = ljapiter(A, B, Q, R, n);
```

Ďalej zopakujeme relevantnú časť predchádzajúceho príkladu, a teda určíme rozmer systému,  $\mathbf{LQ}$  zosilnenie a inicializujeme odhad výsledku ako jednotkovú maticu:

```
[nx nu]=size(B); % Rozmery
K=-dlqr(A, B, Q, R); % LQ zosilnenie
P=eye(2); % Inicializacia
```

Záverom iterujeme formuláciu Ljapunovej rovnice pre nás prípad:

```
for i=1:n % Iterujeme
P=(A+B*K)'*P*(A+B*K)+Q+K'*R*K; % Lyapunova rovnica
end
```

Výsledok môžeme overiť pomocou stavového modelu z predchádzajúceho príkladu

Zdrojový kód 4.12. ljaptest.m

```
A=[1.1 2; 0 0.95]; B=[0; 0.0787]; % Model
Q=eye(2); R=1; % Vahovanie
P=ljapiter(A, B, Q, R, 20) % Iteracne riesenie
```

kde po spustení testovacieho programu dostaneme identické riešenie ako v predchádzajúcom príklade.

□

Našu funkciu na iteračné riešenie diskrétnej Ljapunovej rovnice môžeme zlepšiť tak, že namiesto vopred daného pevného počtu iterácií sa algoritmus sám rozhodne – napríklad na základe zmien vo výsledku – keď dokončí výpočet. Funkciu môžeme tiež doplniť o kontrolu správneho rozmeru matíc, o dodržaní matematických podmienok váhovania a o mnoho iných vylepšení.

**Príklad 4.11.** Iteračnou metódou vyriešte Riccatiho rovnicu pre problém LQ riadenia systému z Pr. 4.9.

Riešenie príkladu začneme definíciou modelu, rozmerov, váhovania, inicializáciou  $\mathbf{P}_0$  a deklaráciou počtu potrebných iterácií:

Zdrojový kód 4.13. prikladriccati.m

```
A=[1.1 2; 0 0.95]; B=[0; 0.0787]; % Model
[nx nu]=size(B); % Rozmery
Q=eye(nx); R=eye(nu); % Vahovanie
P=eye(2); % Inicializacia
n=50; % Iteracie
```

Ďalej  $n$ -krát iterujeme Riccatiho rovnicu podľa iteratívnej metódy z Rov. (4.54)–(4.55):

```
for i=1:n % Iterujeme
    K=inv(R+B'*P*B)*B'*P*A; % LQ zosilnenie
    P=A'*P*A-A'*P*B*inv(R+B'*P*B)*B'*P*A+Q; % Riccatiho rov.
end
```

Výsledok vypíšeme do príkazového riadku a na kontrolu vypočítame LQ zosilnenie aj na základe `dlqr()`:

```
P, K % Vypis
Kx=dlqr(A, B, Q, R) % Kontrola
```

V príkazovom riadku vidíme výsledok

```
P =
    6.9301    24.6635
    24.6635   138.3141
K =
    1.1500    7.6605
Kx =
    1.1500    7.6605
```

z čoho je jasné, že výsledky sú z pohľadu LQ zosilnenia identické na 5 číslic. Rozdiely uvidíme iba ak odčítame výsledok iteráčnej metódy od zabudovanej funkcie pomocou príkazu  $K-Kx$ . Rozdiely sú na úrovni  $1E-10$ , čo je akceptovateľné. Na rozdiel od LQ zosilnenia, pri výpočte matice  $\mathbf{P}$  ihneď uvidíme aj malý rozdiel oproti predchádzajúcim príkladom.

□

**Príklad 4.12.** Vytvorte vlastnú funkciu na výpočet LQ-optimálneho zosilnenia na základe iteratívneho riešenia Riccatiho rovnice. Výsledky porovnajte s algoritmom `dlqr()`.

Výstupnými argumentmi funkcie budú **K** a **P**. Na vstupe funkcie musíme poznáť model (**A**, **B**), váhovanie (**Q**, **R**) a žiadaný počet iterácií:

Zdrojový kód 4.14. `iterdlqr.m`

```
function [K P]=iterdlqr(A,B,Q,R,n)
```

Pri obsahu funkcie vychádzame z predchádzajúceho príkladu. Po inicializácii **P** postupne iterujeme Riccatiho rovnicu:

```
P=eye(length(A)); % Inicializacia
for i=1:n % Iterujeme
    K=inv(R+B'*P*B)*B'*P*A; % LQ zosilnenie
    P=A'*P*A-A'*P*B*inv(R+B'*P*B)*B'*P*A+Q; % Riccatiho rov.
end
```

Výsledky môžeme preveriť na základe modelu a váh z predchádzajúceho príkladu:,

Zdrojový kód 4.15. `testiterdlqr.m`

```
A=[1.1 2; 0 0.95]; B=[0; 0.0787]; % Model
Q=eye(2); R=1; % Vahovanie
[K,P]=iterdlqr(A,B,Q,R,50) % Nase riesenie
[Kx,Px]=dlqr(A,B,Q,R) % Kontrola
max(max(abs(P-Px))) % Max rozdiel P
max(max(abs(K-Kx))) % Max rozdiel K
```

Po spustení testovacieho programu môžeme porovnať výsledné matice **K** a **P** a zistíme, že na prvý pohľad vyzerajú podobne.

Po vypočítaní maximálneho rozdielu vidíme, že je to na akceptovateľnej úrovni  $1E-10$ . Ak do príkazového riadku napišeme `eps`, MATLAB vráti rozdelenie plávajúcich čísel, ktoré vyjadruje presnosť reprezentácie čísel s desatinnou bodkou na počítači. Pri moderných architektúrach je to na úrovni  $2E-16$ , a teda môžeme konštatovať, že rozdiel je stále na úrovni 6 desatinných miest.

Aj keď ide o iteratívnu metódu, uvedená realizácia je stále veľmi priamočiara. Mohli by sme napríklad kontrolovať rozmery a vlastnosti vstupných parametrov, preveriť stabilitu, či automaticky ukončiť iterácie pri určitej konvergenčnej podmienke.

□

**Príklad 4.13.** Vypočítajte maticu váhovania koncového stavu,  $\mathbf{P}$ , pre diskrétnu lineárny stavový systém z Pr. 4.9 pomocou príkazu `dlyap()` ak predpokladáme, že poznáte LQ zosilnenie  $\mathbf{K}$ .

Začneme definíciou systému, penalizačných matíc a výpočtom LQ zosilnenia:

Zdrojový kód 4.16. lјapunov.m

```
A=[1.1 2; 0 0.95]; B=[0; 0.0787]; % Model
[nx nu]=size(B); % Rozmery
Q=eye(nx); R=eye(nu); % Vahovanie
K=-dlqr(A,B,Q,R); % LQ zosilnenie
```

Zavolaním funkcie `dlyap()`

```
P = dlyap([A+B*K]', [Q+K'*R*K]) % Koncové vahovanie
```

dostaneme koncovú penalizačnú maticu  $\mathbf{P}$  v tvare:

```
P =
 6.9301 24.6635
 24.6635 138.3141
```

□

## 4.5 Formulácia zloženej účelovej funkcie v kompaktnom tvarе

Vypočítanú penalizačnú maticu koncového stavu môžeme dosadiť do účelovej funkcie. Týmto spôsobom dostaneme účelovú funkciu definovanú na nekonečnom horizonte a získame tak riadenie so zloženým riadiacim zákonom. Stále platí, že účelovú funkciu formulujeme podľa Rov. (4.28), avšak definíciu musíme rozšíriť o ďalšiu podmienku: vstupy za horizontom sú vypočítané na základe pevnej maticovej spätnej väzby. Matematicky môžeme účelovú funkciu definovať ako

$$J_k = \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}, \quad (4.86)$$

vzhľadom na podmienky:

$$\begin{aligned} \mathbf{x}_k &= \hat{\mathbf{x}}_k, \\ \mathbf{x}_{k+i+1} &= \mathbf{A} \mathbf{x}_{k+i} + \mathbf{B} \mathbf{u}_{k+i} \quad \text{pre } i = 0, 1, 2, \dots, n_p - 1, \\ \mathbf{u}_{k+i} &= \mathbf{K} \mathbf{x}_{k+i}, \quad \text{pre } i = n_p, n_p + 1, \dots, \infty \end{aligned}$$

kde posledná podmienka definuje pevné spätnoväzbové riadenie za horizontom.

V prípade kompaktného zápisu z Rov. (4.39) nemusíme vykonať žiadnu zmenu, a ak máme  $\mathbf{P}$  už správne vypočítanú, dostaneme matice  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{F}$  pre zloženú účelovú funkciu.

**Príklad 4.14.** Vypočítajte matice  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{F}$  pre stavový systém a penalizačné matice z Pr. 4.6. Uvažujte riadenie na nekonečnom horizonte, t. j. zloženú účelovú funkciu.

Na začiatku postupujeme podľa Pr. 4.6, definujeme si model, horizont predikcie a váhovacie matice:

Zdrojový kód 4.17. zlozenafunkcia.m

```
A=[1 2; 0 0.5]; B=[0; 0.05]; % Model
C=[-1 1]; % Model
np=5; Q=C'*C; R=0.1; % Horizont a vahy
```

Na rozdiel od predchádzajúcich príkladov si teraz nezvolíme sami maticu koncového váhovania, napríklad ako  $\mathbf{P} = \mathbf{Q}$ , ale použijeme zložený riadiaci zákon. Najprv vypočítame  $\mathbf{LQ}$  zosilnenie a potom vyriešime Ljapunovu rovnicu:

```
K=-dlqr(A,B,Q,R); % LQ zosilnenie
P=dlyap([A+B*K]',[Q+K'*R*K]); % Koncove vahovanie
```

Matice  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{F}$  môžeme vypočítať pomocou funkcie `ucelovafunkcia()`:

```
[H,G,F]=ucelovafunkcia(A,B,np,Q,R,P) % Ucelova funkcia
```

Dostávame nasledujúci výsledok:

$\mathbf{H} =$

0.3213	0.1900	0.1548	0.1183	0.0821
0.1900	0.2820	0.1510	0.1165	0.0814
0.1548	0.1510	0.2434	0.1131	0.0799
0.1183	0.1165	0.1131	0.2061	0.0771
0.0821	0.0814	0.0799	0.0771	0.1714

$\mathbf{G} =$

1.1882	4.5894
0.9899	3.8796
0.7933	3.1350
0.6002	2.3834
0.4140	1.6488

$\mathbf{F} =$

9.4322	27.7459
27.7459	104.3859

a v porovnávaní s Pr. 4.6 vidíme, že v dôsledku zloženého riadiaceho zákona sme dostali odlišné matice, ktoré v sebe zahŕňajú aj vplyv druhého módu riadenia.  $\square$

### Ako získame vstupy?

V predchádzajúcej kapitole sme sa dozvedeli spôsob výpočtu budúcich stavov na základe predikčnej rovnice. Taktiež sme si definovali matematický spôsob výjadrenia kvality riadenia. Nezabudnime však na naše hlavné poslanie: vypočítať optimálne akčné zásahy. Ako môžeme zistiť tieto neznáme vstupy?

Problém výpočtu optimálnych akčných zásahov môžeme v podstate formulovať takto:



Ak poznáme aktuálny stav  $\mathbf{x}_k$ , aká je sekvencia budúcich vstupov  $\vec{\mathbf{u}}_k$ , ktorá má za následok minimálnu hodnotu účelovej funkcie  $J_k$ ?

V Kap. 5 odpovieme na túto zaujímavú otázku najprv pri absencii procesných obmedzení a potom si objasníme aj význam obmedzení pre prediktívne riadenie.

## Kapitola 5

# Prediktívne riadenie bez obmedzení a formulácia procesných obmedzení

V predchádzajúcej kapitole sme si vytvorili účelovú funkciu  $J_k$ , ktorá vyjadruje kvalitu riadenia na definovanom horizonte. Ak si vo všeobecnosti označíme účelovú funkciu ako

$$J_k = f(\mathbf{x}_k, \vec{\mathbf{u}}_k), \quad (5.1)$$

potom úlohou prediktívneho riadenia je hľadanie vektora budúcich vstupov  $\vec{\mathbf{u}}_k$  ktorý predstavuje najlepšie možné riadenie pre aktuálny stav  $\mathbf{x}_k$ . Inými slovami, väčšinou nás nezaujíma priamo hodnota účelovej funkcie, ale hľadáme pri akom argumente  $\vec{\mathbf{u}}_k$  dosiahne minimum, teda matematicky

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} J_k(\mathbf{x}_k, \vec{\mathbf{u}}_k), \quad (5.2)$$

kde hviezdičkou označíme optimálnu hodnotu sekvencie vstupov a pod výrazom „min“ vyznačíme optimalizačnú premennú. Keďže stav na základe merania alebo odhadu je pevne daný, už ju zmeniť nemôžeme. Na druhej strane, v každej vzorke máme nový stav, preto z časového pohľadu je to premenlivou súčasťou účelovej funkcie. Jednoducho povedané, vektor  $\vec{\mathbf{u}}_k^*$  bude obsahovať najlepšie možné vstupy do akčných členov do konca horizontu pre aktuálny stav  $\mathbf{x}_k$  – je to *optimálne riešenie* nášho problému z Rov. (5.2) pre vzorku  $k$ .

Prvou úlohou tejto kapitoly preto bude nájdenie  $\vec{\mathbf{u}}_k^*$  bez prítomnosti procesných obmedzení, potom si bližšie prešetríme spôsob formulácie obmedzení pre optimalizačnú úlohu.

## 5.1 Prediktívne riadenie bez obmedzení

Najprv si bližšie špecifikujme, aký problém musíme vyriešiť na získanie optimálnych vstupov. Z predchádzajúcej kapitoly už vieme, že zloženú účelovú funkciu pre lineárny, časovo invariantný stavový systém môžeme definovať podľa Rov. (4.86). Potom musíme na základe Rov. 5.2 nájsť riešenie,  $\vec{\mathbf{u}}_k^*$ , nasledujúceho optimalizačného problému ako

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} \left[ \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \right], \quad (5.3)$$

vzhľadom na podmienky:

$$\begin{aligned} \mathbf{x}_k &= \hat{\mathbf{x}}_k, \\ \mathbf{x}_{k+i+1} &= \mathbf{A} \mathbf{x}_{k+i} + \mathbf{B} \mathbf{u}_{k+i} \quad \text{pre } i = 0, 1, 2, \dots, n_p - 1, \\ \mathbf{u}_{k+i} &= \mathbf{K} \mathbf{x}_{k+i}, \quad \text{pre } i = n_p, n_p + 1, \dots, \infty. \end{aligned}$$

kde  $\mathbf{K}$  je ustálené zosilnenie LQ riadiaceho zákona.

Na základe diskusie z Kap. 4, ak máme model  $\mathbf{A}$ ,  $\mathbf{B}$ , váhy  $\mathbf{Q}$ ,  $\mathbf{R}$  a ak koncové váhovanie  $\mathbf{P}$  rátame podľa Ljapunovej rovnice (4.50), účelovú funkciu v kompaktnom tvare môžeme vyjadriť na základe Rov. (4.39). Optimalizačná úloha je potom nájsť nasledujúce riešenie optimalizačného problému

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} (\vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + 2 \mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k + \mathbf{x}_k^T \mathbf{F} \mathbf{x}_k) \quad (5.4)$$

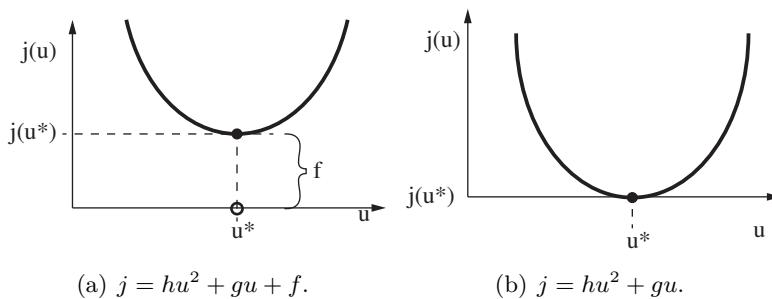
pričom  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{F}$  dostaneme z Rov (4.40)–(4.43).

### 5.1.1 Vplyv posledného výrazu

Všimnime si pri našej optimalizačnej úlohe niekoľko zaujímavých vecí. Po prvej, samotný stav  $\mathbf{x}_k$  v danej vzorke  $k$  je vždy pevne daný a nie je teda vôbec súčasťou optimalizačnej úlohy. Kedže stav zmeráme alebo odhadneme pre vzorku  $k$ , je tento konštantný, a preto výrazy, ktoré násobíme iba stavom, môžeme považovať pre aktuálnu vzorku konštantné, t. j.

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} (\vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + 2 \underbrace{\mathbf{x}_k^T \mathbf{G}^T}_{\text{konšt.}} \vec{\mathbf{u}}_k + \underbrace{\mathbf{x}_k^T \mathbf{F} \mathbf{x}_k}_{\text{konšt.}}),$$

kde výraz  $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$  je pre vzorku  $k$  konštantný, pretože maticu  $\mathbf{G}^T$  násobíme stavom. Tento výraz má pre našu optimalizačnú úlohu špeciálny význam, pretože  $\mathbf{g}$  budeme nazývať *gradient*.



Obr. 5.1. Hodnota posledného člena kvadratickej funkcie vplýva na hodnotu funkcie, ale nie na hodnotu argumentu v minime.

Ďalej je jasné, že posledný člen neobsahuje žiadne optimalizačné premenné  $\vec{u}_k$ . Z toho vyplýva pre minimalizáciu účelovej funkcie ďalšia zaujímavá vlastnosť. Už vieme, že naša účelová funkcia má kvadratický tvar; je maticovou analógiou skalárnej rovnice  $j = hu^2 + gu + f$ . Táto skalárna rovnica opisuje  $j$  v závislosti  $u$  ako parabolu. Môžeme si predstaviť nie príliš užitočné prediktívne riadenie s horizontom  $n_p = 1$ , ktorého problém by sa zredukoval na hore uvedenú skalárnu kvadratickú funkciu. Ak by sme pridali ešte jeden vstup do horizontu, t. j.  $n_p = 2$ , už by sme dostali plochu možných vstupov  $\vec{u}_k$ , plus v ďalšom, treťom rozmere by sme vykreslili skalárnu hodnotu účelovej funkcie. Geometrický útvar opísaný takoto účelovou funkciou je trojrozmerná parabola; paraboloid. Akonáhle pridávame ďalšie predikčné kroky do horizontu, opustíme 3D priestor a grafické znázornenie bude obtiažne. To neznamená, že úloha nemá matematický a praktický zmysel. Predikčný horizont dĺžky  $n_p$  vytvorí  $n_p + 1$  rozmerný geometrický útvar, kde hľadáme  $\vec{u}_k$ ; navyše stále máme kriteriálnu funkciu v skalárnom tvaru. Vo všeobecnosti by sme teda museli vykresliť mnohorozmernú parabolu, tzv. hyperparaboloid v mnohorozmernom priestore (hyperpriestore). Aj keď nie sme schopní nakresliť a graficky ilustrovať takúto mnohorozmerný optimalizačný problém, základné matematické a geometrické princípy tu stále platia.

Vráťme sa k účelovej funkciu a graficky si ju zobrazme pre prípad s jediným predikovaným vstupom (Obr. 5.1). Cieľom prediktívneho riadenia je nájsť optimálny vstup tak, že hľadáme minimálnu hodnotu kriteriálnej funkcie. Absolútna hodnota kriteriálnej funkcie nás preto nezaujíma, ale iba to *kde* (v ktorom bode) nadobúda svoje minimum. Parabolu, teda funkčnú hodnotu pre danú vzorku môžeme posúvať hore – dole, pričom príslušná hodnota argumentu sa nezmení.

**Príklad 5.1.** Nakreslite účelovú funkciu  $j = u^2 + 0.5u + f$ , kde  $f$  je náhodné číslo.

Deklarujme si konštanty a člen  $f$  si vygenerujeme náhodne pri každom spustení. Vytvoríme si vektor nezávislej premennej  $u$  a vypočítame hodnoty pre celý vektor na základe funkcie  $j(u)$ :

Zdrojový kód 5.1. kvadratickaF.m

```

h=1; g=0.5; f=rand(1); % Konstanty, vyraz f
u=-1:0.01:0.5; % Premenna u
j=h*u.^2+g*u+f; % Funkcia j

```

Ďalej si funkciu zobrazíme, označíme osi a podržíme graf pre ďalšie spustenie:

```

plot(u, j) % Vykreslenie
 xlabel('u (-)') % Znacenie osi x
 ylabel('J(u) (-)') % Znacenie osi y
 grid on % Mriezka
 hold on % Podrzat graf

```

Funkcia dosiahne minimálnu hodnotu vždy pri  $u^* = -0.25$  a v tomto bode môžeme vypočítať aj funkčné hodnoty  $j(u^*)$  pre rôzne  $f$ . Nakreslime si v bode minima hviezdíčku a označme príslušné  $u^*$  a  $j(u^*)$  na osi  $x$  a  $y$ :

```

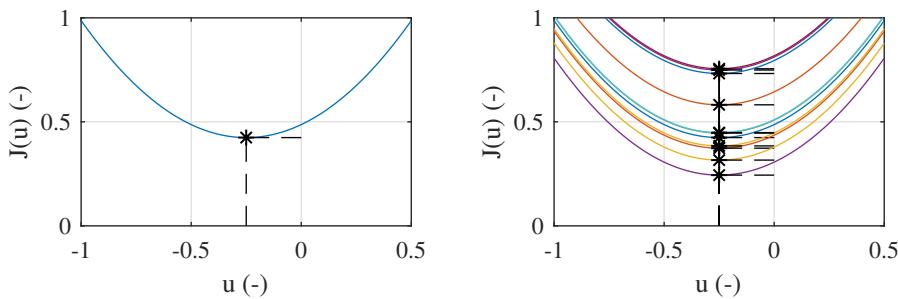
uast=-0.25 % Minimum je uast
ju=h*uast^2+g*uast+f % Hodnota j(uast)
plot(uast,ju,'k*') % Tu je minimum
line([uast,uast],[0,ju],'Color','k','LineStyle','--')
line([0,uast],[ju,ju],'Color','k','LineStyle','--')

```

Ak opakovane spustíme program, vidíme, že tvar paraboly sa nemení, iba posúva v smere osi  $y$  vplyvom náhodného člena  $f$  (Obr. 5.2). Ani komponent minima na osi  $x$ , teda hodnota argumentu minimalizácie, sa nemení, iba jej funkčná hodnota. Toto samozrejme matematický platí aj pre vektory  $\mathbf{u}$  ľubovoľnej veľkosti.

□

Posledný člen účelovej funkcie neobsahuje závislosť na optimalizačnej premennej  $\vec{u}_k$ , a preto iba mení funkčnú hodnotu  $J_k(\vec{u}_k)$ , pričom nemá vplyv na optimum  $\vec{u}_k^*$ .



Obr. 5.2. Argument minimalizácie kvadratickej funkcie, t. j.  $u^*$ , ostáva nezávislý na jej poslednom člene.

Základný optimalizačný problém prediktívneho riadenia sa zredukuje na

$$\vec{u}_k^* = \arg \min_{\vec{u}_k} \left( \vec{u}_k^T \mathbf{H} \vec{u}_k + 2\mathbf{x}_k^T \mathbf{G}^T \vec{u}_k \right). \quad (5.5)$$



Fakt, že pre samotný optimalizačný problém môžeme vynechať daný posledný člen bez toho, aby sme zmenili výsledok ešte však neznamená, že to isté môžeme urobiť ak rátame – napríklad pre diagnostické účely – hodnotu účelovej funkcie. Ako vieme, pre účelovú funkciu platí

$$\begin{aligned} J_k &= \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \\ &= \vec{u}_k^T \mathbf{H} \vec{u}_k + 2\mathbf{x}_k^T \mathbf{G}^T \vec{u}_k + \mathbf{x}_k^T \mathbf{F} \mathbf{x}_k, \end{aligned} \quad (5.6)$$

a odstránením posledného člena by sme túto rovnosť dvoch ekvivalentných tvarov porušili. Taktiež je užitočné si uvedomiť, že pri pôvodnej forme účelovej funkcie je každý člen garantované kladný vplyvom kvadrátu (a pretože  $\mathbf{Q} \succeq 0$ ,  $\mathbf{R} \succeq 0$ ,  $\mathbf{P} \succeq 0$ ), preto aj výsledná hodnota účelovej funkcie je stále kladná. Pri kompaktnom zápisu účelovej funkcie je jej výsledná hodnota, člen  $2\mathbf{x}_k^T \mathbf{G}^T \vec{u}_k$  však už nemusí vždy nadobúdať iba kladné hodnoty, t. j.

$$\sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \neq \vec{u}_k^T \mathbf{H} \vec{u}_k + 2\mathbf{x}_k^T \mathbf{G}^T \vec{u}_k.$$

Nezabudnime na skalárnu analógiu, kde pri  $j = hu^2 + gu + f$  (a  $h > 0$ ) prvý člen musí byť kladný, druhý už nie.

Tabuľka 5.1. Vybrané pravidlá derivácie maticových výrazov. Všetky premenné sú všeobecné,  $\mathbf{A}$  označuje maticu,  $\mathbf{x}$  a  $\mathbf{b}$  označujú vektory,  $a$  skalár a skalárny čitateľ; derivujeme podľa vektora  $\mathbf{x}$ .

Výraz	Derivácia	Podmienky
$\frac{\partial a}{\partial \mathbf{x}}$	$\mathbf{0}$	$a$ nie je funkciou $\mathbf{x}$
$\frac{\partial ay}{\partial \mathbf{x}}$	$a \frac{\partial y}{\partial \mathbf{x}}$	$a$ nie je funkciou $\mathbf{x}$ a $y = f(\mathbf{x})$
$\frac{\partial \mathbf{b}^T \mathbf{x}}{\partial \mathbf{x}}$	$\mathbf{b}$	$\mathbf{b}$ nie je funkciou $\mathbf{x}$
$\frac{\partial \mathbf{x}^T \mathbf{A} \mathbf{x}}{\partial \mathbf{x}}$	$(\mathbf{A} + \mathbf{A}^T) \mathbf{x}$	$\mathbf{A}$ nie je funkciou $\mathbf{x}$
$\frac{\partial \mathbf{x}^T \mathbf{A} \mathbf{x}}{\partial \mathbf{x}}$	$2\mathbf{A}\mathbf{x}$	$\mathbf{A}$ nie je funkciou $\mathbf{x}$ , $\mathbf{A}$ je symetrická ( $\mathbf{A} = \mathbf{A}^T$ )

### 5.1.2 Riešenie problému MPC bez obmedzení

Našou úlohou je riešiť problém definovaný v Rov. (5.5). Z predchádzajúcej kapitoly vieme, že musíme nájsť bod, v ktorom sa gradient funkcie rovná nule, teda

$$\nabla J(\vec{\mathbf{u}}_k) = \mathbf{0}. \quad (5.7)$$

Deriváciou Rov. (5.6) podľa  $\vec{\mathbf{u}}_k$  dostaneme gradient v tvare

$$\nabla J(\vec{\mathbf{u}}_k) = \frac{\partial(\vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k)}{\partial \vec{\mathbf{u}}_k} + \frac{\partial(2\mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k)}{\partial \vec{\mathbf{u}}_k} + \cancel{\frac{\partial(\mathbf{x}_k^T \mathbf{F} \mathbf{x}_k)}{\partial \vec{\mathbf{u}}_k}}, \quad (5.8)$$

kde platia vybrané pravidlá derivácie maticových výrazov z Tab. 5.1.

Na prvý výraz pri  $\mathbf{H} = \mathbf{H}^T$  môžeme používať identitu rovnako platnú pre skalárnu funkciu

$$\frac{\partial \vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k}{\partial \vec{\mathbf{u}}_k} = (\mathbf{H} + \mathbf{H}^T) \vec{\mathbf{u}}_k = 2\mathbf{H} \vec{\mathbf{u}}_k,$$

čo samozrejme platí, pretože Hessián je symetrický a výsledkom  $\vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k$  je skalár. Môžeme to pochopiť aj ako istú skalárnu analógiu  $d(Hu^2)/du = 2Hu$ . Pravidlá derivácie maticových (vektorových) výrazov čiastočne vysvetľujú aj potrebu zvoliť symetrické  $\mathbf{Q}$  a  $\mathbf{R}$ . Keby sme dostali nesymetrický Hessián, potom by neplatila derivácia prvého členu Rov. (5.6) tak, ako sme si to tu uviedli.

Pre druhý člen môžeme celú parciálnu deriváciu násobiť konštantným číslom, teda platí

$$\frac{\partial(2\mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k)}{\partial \vec{\mathbf{u}}_k} = 2 \frac{\partial(\mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k)}{\partial \vec{\mathbf{u}}_k}, \quad (5.9)$$

potom podľa pravidiel násobenia transponovaným vektorom a pretože  $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$  je vektor konštant, ktorý nezávisí od  $\vec{\mathbf{u}}_k$ , a pretože  $(\mathbf{g}^T)^T = (\mathbf{x}_k^T \mathbf{G}^T)^T = \mathbf{G} \mathbf{x}_k$ , dostaneme

$$2 \frac{\partial(\mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k)}{\partial \vec{\mathbf{u}}_k} = 2 \mathbf{G} \mathbf{x}_k, \quad (5.10)$$

čo môžeme pochopiť tiež ako skalárnu analógiu  $d(2xGu)/du = 2xG$ . Keby sme napriek diskusii z Kap. 5.1.1 rátali s posledným členom  $\mathbf{x}_k^T \mathbf{F} \mathbf{x}_k$ , pomocou pravidiel z Tab. 5.1 by sme dostali

$$\frac{\partial(\mathbf{x}_k^T \mathbf{F} \mathbf{x}_k)}{\partial \vec{\mathbf{u}}_k} = \mathbf{0}, \quad (5.11)$$

lebo celý výraz je konštantný skalár ktorý, nezávisí na  $\vec{\mathbf{u}}_k$ , a derivácia konštanty je nula. Okrem vyššie uvedenej geometrickej interpretácie toto aj matematicky potvrzuje, že pri minimalizácii účelovej funkcie nepotrebuje uvažovať jej posledný, konštanntný člen.

Gradient účelovej funkcie (5.6) má potom tvar

$$\nabla J(\vec{\mathbf{u}}_k) = 2\mathbf{H} \vec{\mathbf{u}}_k + 2\mathbf{G} \mathbf{x}_k. \quad (5.12)$$

Kedže funkcia v bode kde gradient sa rovná nule, nerastie ani neklesá a uvažujeme konvexnú funkciu kde jediný takýto bod označuje extrém (minimum funkcie)

$$2\mathbf{H} \vec{\mathbf{u}}_k + 2\mathbf{G} \mathbf{x}_k = \mathbf{0}, \quad (5.13)$$

z čoho po presunutí druhého člena na pravú stranu a predelením 2 dostaneme

$$\mathbf{H} \vec{\mathbf{u}}_k = -\mathbf{G} \mathbf{x}_k. \quad (5.14)$$

Ak chceme zo získaného výrazu vypočítať  $\vec{\mathbf{u}}_k$ , musíme ešte niečo urobiť s Hessiánom na ľavej strane. Deliť samozrejme nemôžeme, namiesto toho rovnicu zlava násobíme inverziou Hessiánu

$$\mathbf{H}^{-1} \mathbf{H} \vec{\mathbf{u}}_k = -\mathbf{H}^{-1} \mathbf{G} \mathbf{x}_k, \quad (5.15)$$

a pretože  $\mathbf{H}^{-1} \mathbf{H} = \mathbf{I}$ , dostávame

MPC riadiaci zákon v absencii procesných obmedzení má uzavretý tvar

$$\vec{\mathbf{u}}_k = -\mathbf{H}^{-1} \mathbf{G} \mathbf{x}_k. \quad (5.16)$$



**Príklad 5.2.** Vypočítajte sekvenčiu optimálnych riadiacich vstupov pre systém opísaný pomocou dvojitého integrátora s maticou vstupu  $\mathbf{B}_c = [0 \ 1]$ , ak jeho počiatočný stav je  $\mathbf{x}_0 = [1 \ 1]$ . Uvažujte zložený riadiaci zákon,  $\mathbf{Q} = \mathbf{I}$ ,  $\mathbf{R} = 1$ , diskretizáciu s  $T_s = 1$  s a horizont predikcie  $n_p = 5$  krokov.

Začnime definovaním dynamiky, váhovacích matíc a iných parametrov a diskretizujme systém pomocou:

Zdrojový kód 5.2. integrator.m

```

Ac=[0 1; 0 0]; Bc=[0 1]'; % Dynamika
Q=eye(2); R=1; % Váhy
x0=[1 1]'; Ts=1; np=5; % Parametre
[A,B,C,D] = c2dm(Ac,Bc,[1 1],0,Ts); % Diskretizacia

```

Maticu koncového váhovania vypočítame ako riešenie Riccatiho rovnice, a pri vytvorení účelovej funkcie použijeme model, váhy a horizont:

```

[K,P]=iterdlqr(A,B,Q,R,100); % Koncové vah.
[H,G]=ucelovafunkcia(A,B,np,Q,R,P); % Ucelova f.

```

Nakoniec vypočítame Rov. (5.16)

```
upr=-inv(H)*G*x0 % Predikcia u
```

na základe čoho dostávame sekvenčiu budúcich optimálnych vstupov  $\vec{u}_k$  v tvare

```

upr =
-1.4629
-0.0750
0.2196
0.1798
0.0942

```

čo predstavuje sekvenčiu optimálnych vstupov  $\vec{u}_k^* = [u_k^* \ u_{k+1}^* \ \dots \ u_{k+4}^*]^T$ .

□

### 5.1.3 Riadenie v spätnej väzbe

Teraz už vieme ako môžeme vypočítať sekvenčiu optimálnych vstupov v MPC bez obmedzení, ale ako môžeme tento poznatok použiť v spätnoväzbovom riadení? Vo vektore  $\vec{u}_k^*$  je predsa oveľa viac prvkov ako v danom vzorkovacom intervale môžeme použiť. Riešenie je veľmi jednoduché: stačí využiť iba prvý člen z vypočítanej

optimálnej sekvencie, kým ostatné ignorujeme. Ak postupnosť optimálnych vstupov má tvar

$$\vec{\mathbf{u}}_k^* = \begin{bmatrix} \mathbf{u}_k^* \\ \mathbf{u}_{k+1}^* \\ \vdots \\ \mathbf{u}_{k+n_p-2}^* \\ \mathbf{u}_{k+n_p-1}^* \end{bmatrix} \quad (5.17)$$

na akčné členy vo vzorke  $k$  posielame iba  $\mathbf{u}_k^*$  a ostatné, t. j.  $\mathbf{u}_{k+1}^*, \dots, \mathbf{u}_{k+n_p-1}^*$ , nepoužijeme. Môžeme na to použiť aj jednoduchý vzťah

$$\mathbf{u}_k^* = \mathcal{I} \vec{\mathbf{u}}_k^*, \quad (5.18)$$

kde  $\mathcal{I}$  je bloková matica  $\mathcal{I} = \begin{bmatrix} \mathbf{I} & 0 & 0 & \dots & 0 \end{bmatrix}$  rozmeru  $n_u \times n_p \cdot n_u$ . V programovej realizácii prediktívneho riadenia nemusíme výsledok prenásobiť  $\mathcal{I}$ , stačí používať správne indexovanie členov výsledku.

**Príklad 5.3.** V pokračovaní Pr. 5.2 vyberte prvý člen  $\vec{\mathbf{u}}_k$ .

Po spustení zdrojového kódu z Pr. 5.2 dostaneme  $\vec{\mathbf{u}}_k$  a môžeme použiť zvyčajné indexovanie matíc v MATLABe na to, aby sme zo sekvencie vstupov  $\vec{\mathbf{u}}_k^*$  vybrali prvý prvok  $\mathbf{u}_k^*$ :

Zdrojový kód 5.3. integratorprvycelen.m (Čiastočný výpis.)

```
u=upr(1) % Prvy clen
```

čím dostaneme prvý člen vektora upr. Ak by sme mali vektor vstupov rozmeru  $n_u \times 1$  (nu), mohli by sme použiť príkaz

```
u=uast(1:nu) % Prvy clen
```

na výber vektora vstupov z  $\vec{\mathbf{u}}_k$ .

□

V ďalšej vzorke zopakujeme tento proces a  $\vec{\mathbf{u}}_k$  prepočítame podľa nového dostupného stavu. Algoritmus prediktívneho riadenia bez obmedzení potom môžeme rozdeliť na dve časti. Pre nominálny model máme časť tvorby účelovej funkcie, čo môžeme vypočítať vopred. Tento výpočet predstavuje *offline* časť MPC algoritmu (Alg. 5.1.), ktorá sa vykoná iba raz pred samotným riadením v uzavretej slučke.

Potom nasleduje takzvaná *online* časť algoritmu (Alg. 5.2.), v ktorej na základe nového stavu vypočítame sekvenciu optimálnych vstupov, z ktorej ako akčný zásah ďalej použijeme iba prvý prvok.

**Algoritmus 5.1** MPC bez obmedzení: offline

- 
- |  |                                |
|--|--------------------------------|
| 1: $\mathbf{P} \leftarrow \text{Rov. (4.50)}$                          | ▷ výpočet koncového zosilnenia |
| 2: $\mathbf{H}, \mathbf{G} \leftarrow \text{Rov. (4.40), Rov. (4.42)}$ | ▷ účelová funkcia              |
- 

**Algoritmus 5.2** MPC riadenie bez obmedzení: online

- 
- |   |                               |
|---|-------------------------------|
| 1: <b>pokial 1 urob</b>   | ▷ v každej vzorke, nekonečne  |
| 2: $\mathbf{x}_k \leftarrow \hat{\mathbf{x}}_k$                             | ▷ odhad, meranie              |
| 3: $\vec{\mathbf{u}}_k \leftarrow -\mathbf{H}^{-1} \mathbf{G} \mathbf{x}_k$ | ▷ MPC bez obmedzení           |
| 4: $\mathbf{u}_k \leftarrow \mathcal{I} \vec{\mathbf{u}}_k$                 | ▷ prvý element na akčné členy |
| 5: <b>koniec pokial</b>   |                               |
- 

**Príklad 5.4.** Pre systém, váhovanie a parametre z Pr. 5.2 vytvorte simuláciu na základe MPC riadenia bez obmedzení.

V prvom kroku deklarujeme model, váhovacie matice a parametre. Postup je rovnaký ako v Pr. 5.2, preto si dalej ukážeme iba čiastočný výpis zdrojového kódu. V offline časti vypočítame  $\mathbf{P}$ ,  $\mathbf{H}$  a  $\mathbf{G}$ :

Zdrojový kód 5.4. integratorspatnavazba.m (Čiastočný výpis.)

```
[K, P] = iterdlqr(A, B, Q, R, 100); % Koncové vah.
[H, G] = ucelovafunkcia(A, B, np, Q, R, P); % Ucelová f.
```

Nasleduje online časť. Pri skutočnom riadení by sme procedúru zopakovali v nekonečnom cykle. Kedže uvažujeme simuláciu, môžeme použiť aj cyklus konečnej dĺžky. Kedže stavy presne poznáme, vynecháme algoritmus odhadu z postupnosti. Potom nasleduje výpočet  $\vec{\mathbf{u}}_k$  a výber jeho prvého člena. Odozvu systému (na miesto skutočného riadeného systému) vypočítame pomocou stavového modelu:

```
for i=1:10 % Trvanie sim.
    upr = -inv(H) * G * x; % Predikcia u
    u = upr(1); % Prvy clen
    x = A * x + B * u; % Aplikacia u
    vykreslitXU(x, u) % Priebeh x, u
end
```

Pri spätnoväzbovej simulácii nás zaujíma časová odozva vnútorných stavov systému aj príslušné akčné zásahy. Vykresliť takýto priebeh bude častou úlohou, preto bude najlepšie ak si na tento účel vytvoríme univerzálnu funkciu.

Nazvime si našu novú funkciu napríklad **vykreslitXU**, kde XU zpredstavuje stavy a vstupy. Funkcia potom bude v každej vzorke simulácie na vstupe zbierať stav  $\mathbf{x}_k$  a vstup  $\mathbf{u}_k$ :

Zdrojový kód 5.5. vykreslitXU.m

```
function vykreslitXU(x,u) % Funkcia na vykreslenie x,u
```

Na jej začiatku použijeme príkaz `persistent` ktorým, ako už vieme, vytvoríme pretrvávajúce premenné, ktorých hodnota sa uchová aj pri opakovanom spúštaní funkcie. Zistíme, či premenná `X` už existuje pomocou funkcie `isempty` a ak nie, inicializujeme matice ako prázdne. Ďalej môžeme dynamicky rozširovať `X` a `U` aktuálnymi stavmi a vstupmi:

```
persistent x u % Trvale premenne
if isempty(X)
    X=[]; U=[];
end % Ak este neexist.
X=[X x]; % Inicializuj
U=[U u]; % Koniec podm.
% Skladovat x v X
% Skladovat u v U
```

Aby sme predošli zbytočným varovaniam o nadmernom počte opisov v legende grafov, vypneme tento typ varovania pomocou:

```
wid='MATLAB:legend:IgnoringExtraEntries';
warning('off',wid) % Vypnut varovan.
```

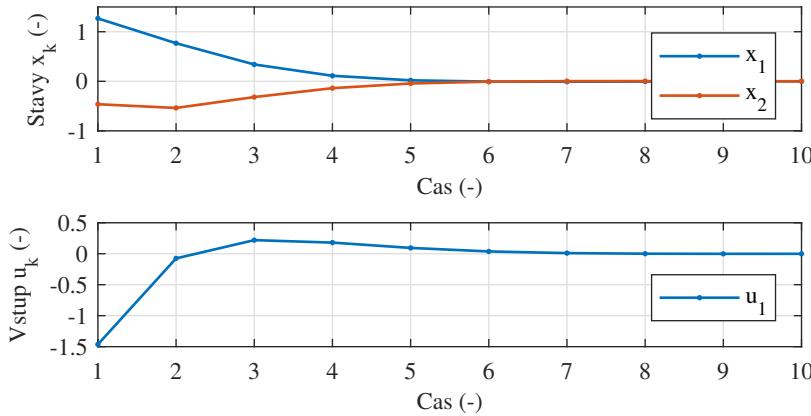
Najprv si vykreslíme stavy, označíme osi a legendu a zapneme mriežku:

```
subplot(2,1,1) % Cast obrazku
plot(X,'LineWidth',1,'Marker','.') % Stavy
 xlabel('Cas (-)') % Os x
 ylabel('Stavy x_k (-)') % Os y
 legend('x_1','x_2','x_3','x_4','x_5') % Legenda
 grid on % Mriezka
```

Potom to isté urobíme aj pre riadiace vstupy:

```
subplot(2,1,2) % Cast obrazku
plot(U,'LineWidth',1,'Marker','.') % Vstupy
 xlabel('Cas (-)') % Os x
 ylabel('Vstupy u_k (-)') % Os y
 legend('u_1','u_2','u_3','u_4','u_5') % Legenda
 grid on % Mriezka
end
```

Poznamenajme, že `vykreslitXU` pri každom spustení prekresluje nový obrázok. Ak chceme, môžeme si nechať krokoval preces pomocou príkazu `pause`, respektíve môžeme aj spomalit simuláciu pomocou vstupného argumentu do príkazu `pause`.



Obr. 5.3. Stavy a vstupy dvojitého integrátora pri MPC riadení bez obmedzení.

Ked spustíme zdrojový kód *integratorspatnavazba.m*, dostaneme odozvu ako na Obr. 5.3. Vidíme, že z počiatočných stavov systém konvergoval k žiadanej nule vplyvom riadiacich vstupov, ktoré boli vypočítané pomocou algoritmu MPC – zatiaľ bez obmedzení.

□

### Zobrazenie priebehu účelovej funkcie

Hoci na výpočet akčných zásahov nepotrebujeme priamo využiť hodnotu účelovej funkcie, pri vývoji a ladení algoritmu je často užitočné sledovať jej priebeh. Ak uvažujeme prediktívne riadenie bez obmedzení, môžeme očakávať monotónne klesajúcu hodnotu účelovej funkcie. Toto samozrejme platí, iba ak nesimulujeme vplyv porúch a systém konverguje z počiatočného stavu do rovnovážneho.

**Príklad 5.5.** Vykreslite priebeh účelovej funkcie pri regulácii systému z Pr. 5.2. Vytvorte univerzálne využiteľnú funkciu na tento účel.

Novú funkciu môžeme nazvať napríklad `vykreslitJ()`. Vstupné parametre budú aktuálny stav  $\mathbf{x}_k$ , vstupy  $\vec{\mathbf{u}}_k$ , Hessián  $\mathbf{H}$ ,  $\mathbf{G}$  a tentoraz budeme v zmysle Rov. (5.6) potrebovať aj maticu  $\mathbf{F}$ :

Zdrojový kód 5.6. `vykreslitJ.m`

```
function vykreslitJ(x, upr, H, G, F) % Vykreslenie J
```

Ďalej postupujeme podobne ako v Pr. 5.4. Jednotlivé hodnoty účelovej funkcie  $J_k(j)$  budeme ukladať do premennej  $J_k$ , ktorú inicializujeme ako pretrvávajúcu premennú. Ak je premenná  $J_k$  prázdna, znamená to, že funkcia je spustená prvýkrát a musíme ju inicializovať ako prázdnú entitu. Ďalej vypočítame hodnotu účelovej funkcie a uložíme ju do dynamicky sa rozširujúcej matice:

```

persistent Jk                                % Trvale premenne
if isempty(Jk)                         % Ak este neexist.
    Jk=[ ] ;                                % Inicializuj
end                                         % Koniec podm.
    j=upr' *H*upr+2*x' *G' *upr+x' *F*x;    % Ucelova funkcia
    Jk=[Jk j] ;                            % Skladovanie Jk

```

Potom si vykreslíme výsledok, označíme osi, zapneme mriežku, prípadne graf môžeme vylepšiť o ďalšie prvky:

```

plot(Jk,'LineWidth',1,'Marker','.')    % Stavy
xlabel('Cas (-)')                           % Os x
ylabel('Ucelova f. J_k (-)')                 % Os y
legend('J_k')                                % Legenda
grid on                                     % Mriezka
end                                         % Koniec funkcie

```

Novú funkciu môžeme rýchlo otestovať tak, že v zdrojovom kóde Pr. 5.4 budeme uvažovať aj s tretím výstupom funkcie `ucelovafunkcia()`, aby sme dostali aj **F**:

Zdrojový kód 5.7. `testvykreslitJ.m` (Čiastočný výpis.)

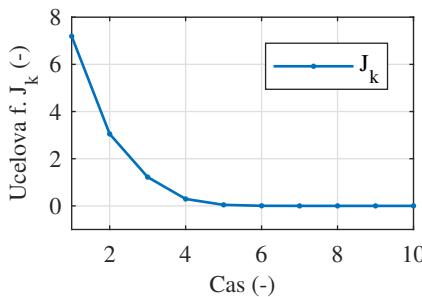
```
[H,G,F]=ucelovafunkcia(A,B,np,Q,R,P) ;    % Ucelova f.
```

Vymeníme relevantný riadok na kreslenie výsledkov:

```
vykreslitJ(x,upr,H,G,F)                % Priebeh j
```

Po spustení programu získame priebeh účelovej funkcie znázornený na Obr. 5.4. Vidíme, že po štarte simulácie hodnota účelovej funkcie monotónne klesá. Je to očakávateľné, pretože máme dokonalú znalosť o dynamike, riadime bez porúch alebo obmedzení. Neskoršie uvidíme, že z priebehu účelovej funkcie môžeme dospiť k záverom aj ohľadom stability MPC riadenia.

□



Obr. 5.4. Grafická ilustrácia hodnoty účelovej funkcie pri MPC riadení dvojitého integrátora.

### Predikované vs. skutočné stavy

Predikcia priebehu stavov môže byť dokonalá len vtedy, ak máme dokonalý model bez vplyvu vonkajších porúch a úplnú znalosť o stave. Samozrejme túto ideálnu situáciu môžeme dosiahnuť iba v simulácii.

Vykreslenie predikovaných stavov v každom vzorkovacom intervale môže mať okrem didaktického aspektu význam aj pri vývoji a ladení samotného algoritmu.

**Príklad 5.6.** Vytvorte funkciu na porovnávanie predikovaných stavov so skutočnými stavmi. Na overenie funkčnosti použite model dvojitého integrátora z Pr. 5.2. Najprv predpokladajte dokonalú znalosť modelu, potom zvýšte hodnoty prvkov matice  $\mathbf{A}$  o približne 20 % na simuláciu „skutočného“ systému tak, aby sme napodobnili nedokonalú znalosť riadeného systému.

Novú funkciu môžeme nazvať napríklad `vykreslitXpred()`. Funkcia potrebuje niekoľko vstupných argumentov, a to aktuálny stav  $\mathbf{x}_k$ , budúce vstupy  $\vec{\mathbf{u}}_k$ , model  $(\mathbf{A}, \mathbf{B})$  a horizont predikcie  $n_p$ :

Zdrojový kód 5.8. `vykreslitXpred.m`

```
function vykreslitXpred(x, upr, A, B, np) % Predikcia
```

Pokračujeme deklaráciou pretrvávajúcich premenných, ktorými budú matice na ukladanie stavov v jednotlivých časových vzorkách, predikčné matice, časový index  $k$  a rád systému  $n_x$ . Pri prvom spustení funkcie inicializujeme maticu  $\mathbf{X}$  ako prázdnú maticu a čas začíname indexovať od jedna<sup>1</sup>. Ďalej si zistíme rád systému a vytvoríme predikčnú maticu. Pri každom volaní funkcie si uložíme stav a vypočítame aktuálnu sekvenciu predikovaných stavov  $\vec{\mathbf{x}}_k$ :

<sup>1</sup>Je to programové zjednodušenie motivované tým, že MATLAB začína indexovať od 1.

```

persistent X M N k nx
if isempty(X)
  X=[]; k=1;
  nx=length(x);
  [M,N] = predikcia(A,B,np);
end
X=[X x];
xpr=M*x+N*upr;

```

% Trvale premenne  
 % Ak este neexist.  
 % Inicializuj X, k  
 % Pocet stavov  
 % Predikcne matice  
 % Koniec podmienky  
 % Skladovat x v X  
 % Predikcna rovn.

Nasleduje samotné vykreslenie výsledkov. Najprv získame identifikátor osí (`ax=gca`) a pri každom spustení reštartujeme poradie farieb (`ax.ColorOrderIndex = 1`). Pre každý stav vykreslíme jeho „skutočnú“ hodnotu, t.j. hodnotu, ktorá je priamo meraná alebo odhadnutá. Proces zopakujeme pre predikciu a zvýšime časový index o jeden krok:

```

ax = gca; ax.ColorOrderIndex = 1;           % Restart farieb
for j=1:nx                                % "Skutočne" stavky
  plot(X(j,:),'LineWidth',1,'Marker','o','LineStyle','-')
end                                         % Koniec for
ax = gca; ax.ColorOrderIndex = 1;           % Restart farieb
for j=1:nx                                % Predik stavky
  plot(k+1:k+np,xpr(j:nx:end,:),'LineWidth',0.3,'Marker'
    , 'x','LineStyle','--')
end                                         % Koniec for
k=k+1;                                       % Dalsi krok

```

Funkciu ukončíme opisom grafu:

```

hold on                                     % Neprekreslit
grid on                                     % Mriezka
xlabel('Cas (-)')                         % Os x
ylabel('Stavy x_k (-)')                     % Os y
end

```

Funkciu otestujeme príkladom dvojitého integrátora, kde test začíname identicky ako pri Pr. 5.2 a plný výpis zdrojového kódu preto nebudeme opakovať.

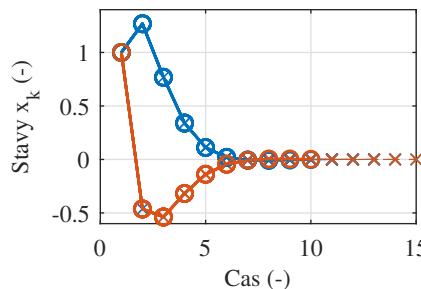
Vytvorime si diagonálnu maticu `delta`, ktorou budeme násobiť maticu dynamiky systému. Ak jej prvky prenásobíme iným číslom ako 1, „pokazíme“ tak model a tým simulujeme nepresnosť modelu oproti skutočnému systému:

Zdrojový kód 5.9. `testvykreslitXpred.m` (Čiastočný výpis)

```

delta=1.2 *eye(2);                         % Nepresnosť mod

```



Obr. 5.5. Ak je model systému dokonale známy, predikované stavy ( $\times$ ) sú identické so skutočnými stavmi ( $\circ$ ) v spätej väzbe.

Pokračujeme online časťou algoritmu, kde využijeme funkciu `vykreslitXpred()`, počkáme na stlačenie klávesy a potom vypočítame odozvu systému:

```

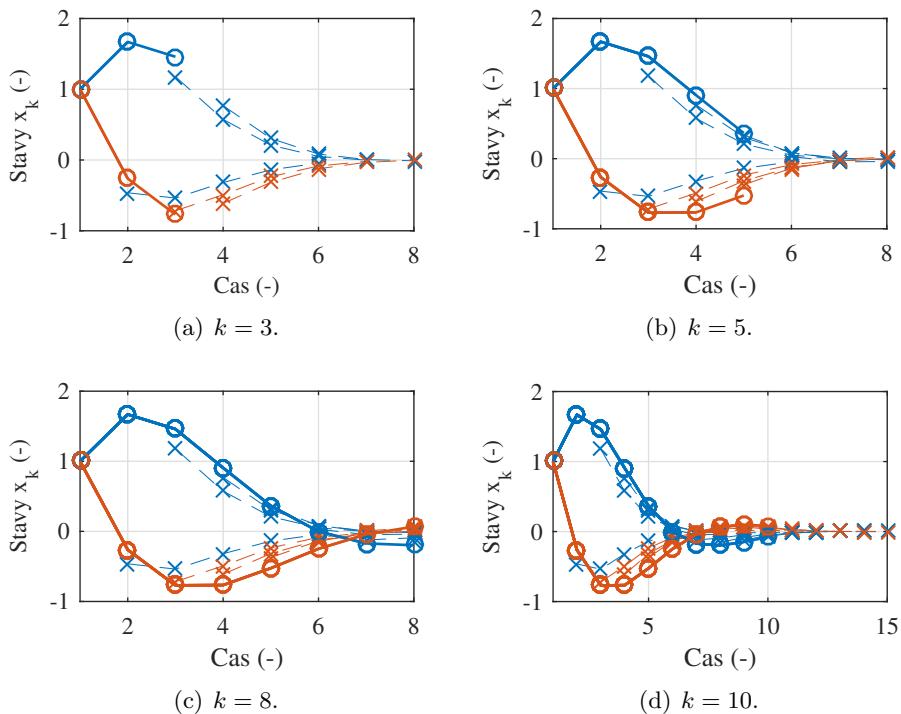
for i=1:10
    upr=-inv(H)*G*x;
    u=upr(1);
    vykreslitXpred(x,upr,A,B,np);
    pause
    x=A*delta*x+B*u;
end

```

% Trvanie sim.  
 % Predikcia u  
 % Prvy clen  
 % Priebeh Xpred  
 % Pauza  
 % Model

Po spustení testovacieho programu musíme stlačiť ľubovoľnú klávesnicu na krokovanie v simulácii. Keď predpokladáme dokonalé vedomosti o dynamike a neprítomnosť porúch, získame predikované stavy, ktoré sú totožné so stavmi v spätej väzbe (Obr. 5.5).

Akonáhle nepoznáme dynamiku systému dokonale, predikcie nie sú totožné s priebehom stavov v uzavretej slučke. Môžeme zmeniť  $\delta \neq \mathbf{I}$ , napríklad zvýšiť elementy dynamiky o 20 %. Po spustení simulácie môžeme pomaly krokovat a uvidíme, že predikcie nie sú perfektné, no stále dobre ukazujú očakávaný trend zmien (Obr. 5.6).



Obr. 5.6. Stavy v spätej väzbe ( $\circ$ ) nie sú totožné s predikovanými stavmi ( $\times$ ) ak dynamiku systému nepoznáme dokonale.

□

### Predikované vs. skutočné vstupy

Podobne ako v prípade stavov, aj pri vstupov sa môže vypočítaná sekvencia  $\vec{u}_k$  značne lísiť od priebehu vstupov v uzavretej slučke riadenia.

**Príklad 5.7.** Vytvorte univerzálnu funkciu na porovnávanie predikovaných a skutočných vstupov. Otestujte funkciu na príklade dvojitého integrátora z Pr. 5.2.

Nová funkcia, napríklad `vykreslitUpred()`, bude na vstupe potrebovať aktuálny vstup  $\mathbf{u}_k$  a sekvenciu vypočítaných (predikovaných) vstupov  $\vec{u}_k$ :

Zdrojový kód 5.10. `vykreslitUpred.m`

```
function vykreslitUpred(u, upr) % Porovnanie u/upr
```

Postupujeme podobne ako v Pr. 5.6. Na začiatku si vytvoríme pretrvávajúce premenné a pri prvom spustení inicializujeme štruktúru na uloženie  $\mathbf{u}_k$  a časový

index. Na základe rozmeru  $\mathbf{u}_k$  zistíme počet vstupov a na základe  $\vec{\mathbf{u}}_k$  predikčný horizont. Pri každom volaní si uložíme nové  $\mathbf{u}_k$ :

```

persistent U nu np k
if isempty(U)
    U=[]; k=1;
    nu=length(u);
    np=length(upr);
end
U=[U u];

```

% Trvale premenne  
 % Ak este neexist.  
 % Inicializuj U, k  
 % Pocet vstupov  
 % Predikcny hor.  
 % Koniec podmienky  
 % Skladovat x v X

Pred kreslením reštartujeme poradie farieb a nakreslíme do grafu skutočné vstupy v spätej väzbe pre každý prvok vektora vstupov. Tento postup zopakujeme aj pre predikovanú sekvenciu vstupov a inkrementujeme časový index  $k$ :

```

ax = gca; ax.ColorOrderIndex = 1;      % Restart farieb
for j=1:nu                            % "Skutočne" stavy
  plot(U(j,:),'LineWidth',1,'Marker','o','LineStyle','--')
end                                    % Koniec for
ax = gca; ax.ColorOrderIndex = 1;      % Restart farieb
for j=1:nu                            % Predikov. stavy
  plot(k:k+np-1,upr','LineWidth',0.3,'Marker','x','
        'LineStyle','--')
end                                    % Koniec for
k=k+1;

```

% Dalsi krok

Na záver graf podržíme, zapneme mriežku a označíme osi:

```

hold on                                % Neprekreslit
grid on                                % Mriezka
xlabel('Cas (-)')                      % Os x
ylabel('Vstup u_k (-)')                  % Os y
end

```

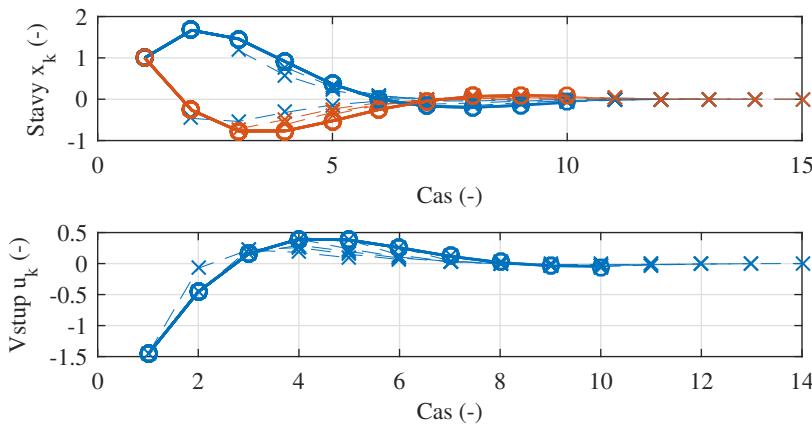
Modifikáciou programu *testvykreslitXpred.m* môžeme otestovať správnosť našej novej funkcie. Na zobrazenie výsledkov vykonáme nasledujúcu zmenu:

Zdrojový kód 5.11. *testvykreslitUpred.m*

```

subplot(2,1,1)                          % Horna cast
vykreslitXpred(x,upr,A,B,np);          % Priebeh Xpred
subplot(2,1,2)                          % Dolna cast
vykreslitUpred(u,upr);                  % Priebeh Upred

```



Obr. 5.7. Zobrazenie priebehov skutočných stavov a vstupov vo spätej väzbe (○) a priebehov predikovaných stavov a vstupov (×).

Po spustení testovacieho programu dostaneme priebeh na Obr. 5.7, kde vidíme, že vplyvom nedokonalej znalosti dynamiky sa predikovaná postupnosť vstupov nezhoduje s ich skutočnými priebehmi.  $\square$

## 5.2 Vlastnosti MPC bez obmedzení

V nasledujúcich sekciách budeme sa zaoberať niektorými zaujímavými vlastnosťami stratégie MPC v absencii procesných obmedzení. Prešetríme aký vzťah má MPC k LQ riadeniu a objasníme vplyv dĺžky horizontu na odozvu v spätej väzbe.

### 5.2.1 Vzťah LQ a MPC bez obmedzení

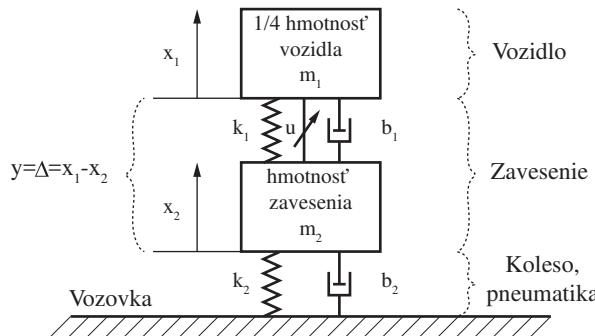
Označme si výraz  $\mathbf{H}^{-1}\mathbf{G}$  z Rov. 5.16 ako

$$\mathcal{K} = -\mathbf{H}^{-1}\mathbf{G}, \quad (5.19)$$

potom sekvenciu budúcich optimálnych vstupov môžeme vypočítať na základe

$$\vec{\mathbf{u}}_k = \mathcal{K}\mathbf{x}_k, \quad (5.20)$$

čo veľmi pripomína na LQ riadiaci zákon ( $\mathbf{u}_k = \mathbf{K}\mathbf{x}_k$ , ak  $\mathbf{K}$  zahŕňa zápornú spätnú väzbu). Keďže stratégiju prediktívneho riadenia bez obmedzení sme dostali v uzavretom tvare, vytvorili sme riadiaci zákon s pevným maticovým zosilnením – presne ako v prípade LQ alebo iných stratégijí.



Obr. 5.8. Štvrtinový model aktívneho (odpruženia) podvozku autobusu.

Podobnosť MPC a LQ sa tu však nekončí. Uvažujme teraz riadenie so zloženým riadiacim zákonom, kde pre tvorbu účelovej funkcie v režime dva predpokladáme LQ stratégiu. Potom počas trvania horizontu v režime jedna používame takú istú účelovú funkciu ako je definované pri LQ riadení a po konci horizontu taktiež používame LQ. Nie je potom MPC bez obmedzení to isté ako LQ riadenie? Odpoved' je jednoznačné áno:



Stratégia prediktívneho riadenia so zloženým riadiacim zákonom na základe LQ v režime dva a bez obmedzení je rovnocenné s LQ riadením.

**Príklad 5.8.** Porovnajte LQ a MPC riadenie bez obmedzení v spätnoväzbovej simulácii aktívneho tlmenia podvozku autobusu. Dynamiku zavesenia podvozku modelujeme štvrtinovým modelom podľa Obr. 5.8, kde stavový vektor je  $\mathbf{x}_k = [x_1 \ \dot{x}_1 \ \Delta_1 \ \dot{\Delta}_1]$  a diskrétny stavový model je uložený v súbore *bus.mat* (pozri *modelbus.m* a [108]). Stavy vähujme pomocou matíc  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  a  $\mathbf{R} = 10$ , horizont je  $n_p = 5$  krokov dlhý a počiatočný stav je  $\mathbf{x}_0 = [0.05 \ 0.05 \ 0.05 \ 0.05]^T$ .

Na zjednodušenie príkladu predpokladajme, že žiadana hodnota pre výstup  $y$  je  $r = y = \Delta = 0$  a systém chceme priviesť do rovnovážneho stavu. Takýto predpoklad je aj prakticky možný, ak stav odčítame od referenčného stavu; inak povedané, posunieme začiatok stavového systému do žiadanej hodnoty.

Začíname načítaním diskretizovaného modelu zo súbora *bus.mat*. Máme iba jeden akčný člen, používame druhý stĺpec stavového modelu. Zistíme počet vstupov a deklarujeme váhy a dĺžku horizontu. Nasleduje deklarácia počiatočného stavu, čo v tomto prípade budeme zvlášť ukladať pre LQ ( $\mathbf{x}_{1q}$ ) a MPC ( $\mathbf{x}_{mpc}$ )

riadenie. Inicializujeme pomocnú premennú  $U_d$ , do ktorého uložíme rozdiel medzi jednotlivými riadiacimi stratégiami:

Zdrojový kód 5.12. lqvsmpc.m

```
load bus.mat % Nahraj model
A=bus.a; B=bus.b(:,2); C=bus.c; % Model autobusu
[nx nu]=size(B); % Rozmery
Q=C'*C; R=10; np=5; % Vahy a horizont
x1q=[0.05 0.05 0.05 0.05]'; xmpc=x1q; % Pociatocny stav
Ud=[]; % Rozdiel v u
```

Pokračujeme s výpočtom LQ zosilnenia a koncovej váhovacej matice a vypočítame matice účelovej funkcie:

```
[K,P]=iterdlqr(A,B,Q,R,100); % Koncove vah.
[H,G]=ucelovafunkcia(A,B,np,Q,R,P); % Ucelova f.
```

V online časti najprv simulujeme LQ riadenie,

```
for i=1:25 % Trvanie sim.
    ulq=-K*x1q; % LQ zakon
    x1q=A*x1q+B*ulq; % Aplikacia u
```

potom MPC riadenie bez obmedzení:

```
upr=inv(H)*G*xmpc; % Predikcia u
umpc=upr(1:nu,:); % Prvy clen pred.
xmpc=A*xmpc+B*umpc; % Aplikacia u
```

V simulácii pokračujeme vykreslením stavov a vstupov pre obidve typy riadenia:

```
figure(1) % Prvy obrazok
vykreslitXU(x1q,ulq)
figure(2) % Druhy obrazok
vykreslitXU(xmpc,umpc)
```

Akčné zásahy môžeme porovnávať jednoduchým odčítaním hodnôt, čo aj realizuje koniec zdrojového kódu:

```
figure(3) % Treti obrazok
Ud=[Ud ulq-umpc];
plot(Ud,'k') % Skladuj rozdiel.
 xlabel('Cas (-)') % Vykresli rozd.
 ylabel('Rozdiel v u_k (m)') % Os y
 grid on; hold on; % Mriezka, podrz.
end
```

Po spustení príkladu získame priebehy, ktoré sú znázornené na Obr. 5.9. Po vizuálnej inšpekции priebehu riadenia podvozku pomocou LQ (Obr. 5.9(a)) a pomocou MPC (Obr. 5.9(b)) nevidíme žiadny rozdiel. Priebeh stavov aj vstupov je identický a rozdiel polohy vozidla a podvozku ( $y = \Delta$ ) a rýchlosť ( $\dot{\Delta}$ ) sa ustálil na žiadanej hodnote. Rozdiely dvoch riadiacich metód môžeme porovnávať hlavne podľa rozdielu akčných zásahov. V tomto prípade sú rozdiely na Obr. 5.9(c) iba numerické a vzhľadom na konečnú presnosť numerickej reprezentácie  $\epsilon$  (eps) zanedbateľné.  $\square$

Pozrime sa teraz bližšie na Rov. 5.20. Ak platí, že pre systém  $\mathbf{A}$  a  $\mathbf{B}$  je matica  $\mathbf{K}$  ustálené LQ-optimálne zosilnenie vrátane zápornej spätej väzby, dynamiku systému v spätej väzbe  $\mathbf{u}_k = \mathbf{K}\mathbf{x}_k$  opíšeme pomocou vnútornej spätej väzby

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\underbrace{\mathbf{K}\mathbf{x}_k}_{\mathbf{u}_k} = (\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x}_k, \quad (5.21)$$

a dostaneme autonómny systém. V prediktívnom riadení bez obmedzení potom v každom predikovanom kroku  $k + i$  kde  $i = 0, 1, \dots, n_p - 1$  vlastne uvažujeme výpočet vstupov na základe LQ riadenia

$$\begin{aligned} \mathbf{u}_k &= \mathbf{K}\mathbf{x}_k, \\ \mathbf{u}_{k+1} &= \mathbf{K}\mathbf{x}_{k+1}, \\ \mathbf{u}_{k+2} &= \mathbf{K}\mathbf{x}_{k+2}, \\ &\vdots = \vdots \\ \mathbf{u}_{k+n_p-1} &= \mathbf{K}\mathbf{x}_{k+n_p-1}. \end{aligned} \quad (5.22)$$

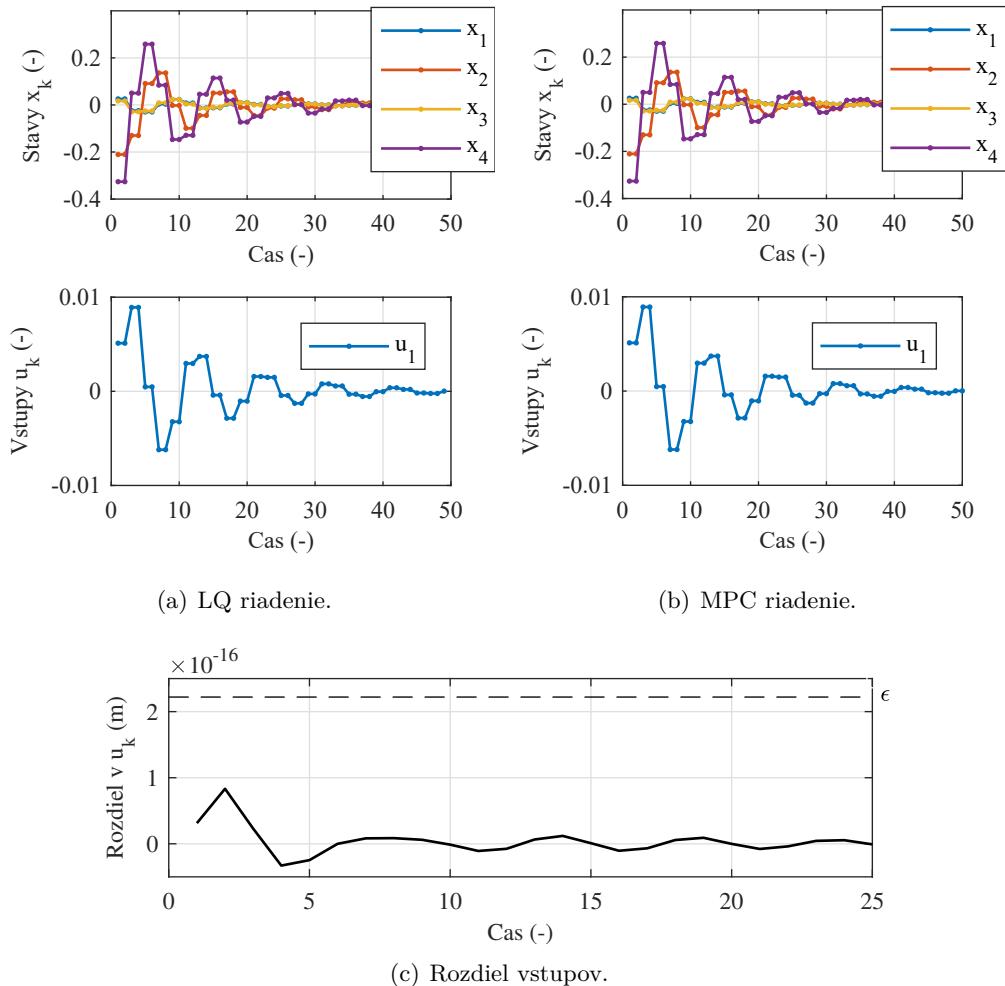
Dynamiku v spätej väzbe môžeme na základe Rov. (5.21) predikovať pre ľubo-voľný stav v budúcnosti ako

$$\mathbf{x}_{k+i+1} = (\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x}_k, \quad (5.23)$$

a keďže

$$\begin{aligned} \mathbf{u}_k &= \mathbf{K}\mathbf{x}_k, \\ \mathbf{u}_{k+1} &= \mathbf{K}\mathbf{x}_{k+1} = \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x}_k, \\ \mathbf{u}_{k+2} &= \mathbf{K}\mathbf{x}_{k+2} = \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^2 \mathbf{x}_k, \\ &\vdots = \vdots \\ \mathbf{u}_{k+n_p-1} &= \mathbf{K}\mathbf{x}_{k+n_p-1} = \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_p-1} \mathbf{x}_k. \end{aligned} \quad (5.24)$$

potom pre optimálnu sekvenciu vstupu platí, že



Obr. 5.9. Spätnoväzbová odozva systému a riadiace vstupy sú v prípade LQ riadenia a MPC bez obmedzení identické.

$$\vec{u}_k = \underbrace{\begin{bmatrix} \mathbf{K} \\ \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K}) \\ \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^2 \\ \vdots \\ \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_p-1} \end{bmatrix}}_{\mathcal{K}} \mathbf{x}_k, \quad (5.25)$$

kde  $\mathbf{K} = \mathcal{I}\mathcal{K}$ ,  $\mathcal{I} = [\mathbf{I} \ \mathbf{0} \dots \mathbf{0}]$ , t. j. prvý riadok predstavuje samotné LQ zosilnenie.

**Príklad 5.9.** Porovnajte riešenie problému prediktívneho riadenia bez obmedzení s LQ riadením tak, že porovnáte bloky výrazu  $\mathcal{K} = -\mathbf{H}^{-1}\mathbf{G}$  a  $\mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i$  pre  $i = 0, 1, \dots, n_p - 1$ . Na porovnanie použite model a parametre z Pr. 5.8.

Začiatok príkladu je identický Pr. 5.8: načítame model, deklarujeme váhy, vy-počítame riešenie Riccatiho rovnice pre LQ problém a vytvoríme účelovú funkciu:

Zdrojový kód 5.13. porovnavaniematic.m

```
load bus.mat % Nahraj model
A=bus.a; B=bus.b(:,2); C=bus.c; % Model autobusu
[nx nu]=size(B); % Rozmery
Q=C'*C; R=10; np=5; % Vahy a horizont
[K,P]=iterdlqr(A,B,Q,R,100); % Koncove vah.
[H,G]=ucelovafunkcia(A,B,np,Q,R,P); % Ucelova f.
```

Najprv vypočítame  $\mathcal{K}$ , potom zahrnieme záporné znamienko späťnej väzby do zosilnenia a inicializujeme prázdnú premennú:

```
Kcal= -inv(H)*G % K kaligraficke
K=-K; % Spatna vazba
Ki=[]; % Prazdna matica
```

V cykle vyplníme novú premennú  $\mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i$  pre  $i = 0, 1, \dots, n_p - 1$ :

```
for i=1:np % Horizont
    Ki=[Ki; K*(A+B*K)^(i-1)] % LQ ekvivalencia
end
```

Po spustení programu vidíme, že pre  $\mathcal{K}$  (Kcal) dostaneme výsledok

```
Kcal =
0.1349 -0.0273 -0.0059 0.0005
0.1614 0.0177 -0.0005 -0.0002
-0.0219 0.0279 0.0040 -0.0005
-0.1265 0.0003 0.0022 -0.0001
-0.0432 -0.0200 -0.0018 0.0003
```

a ďalej môžeme sledovať ako pre  $\mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{i-1}$  postupne dostávame identické (blokové) riadky. Príkazom `max(max(abs(Kcal-Ki)))` môžeme overiť rozdiel dvoch výrazov, čo je zanedbateľné a spôsobené konečnou presnosťou numerických výpočtov v MATLABe.

□

### Ak MPC bez obmedzení je LQ, prečo ho vôbec potrebujeme?

V predchádzajúcej diskusii sme si názorne demonštrovali to, že prediktívne riadenie so zloženou účelovou funkciou podľa LQ spätej väzby v druhom režime je totožné s LQ riadením. Môžeme si pomysliť, ak sú identické, prečo ich vôbec potrebujeme rozlišovať?

Táto identita platí len ak vyberieme LQ riadiaci zákon pre druhý režim. Ak by sme si vybrali inú pevnú maticovú spätnú väzbu (napríklad pri uvažovaní  $H_2$  alebo  $H_\infty$  normy), už by sme dostali iné akčné zásahy. Ďalej, nemusíme vždy na tvorbu zloženej účelovej funkcie používať ako koncové váhovanie  $\mathbf{P}$  riešenie Ljapunovej rovnice, ale môžeme považovať  $\mathbf{P} = \mathbf{Q}$  a dostaneme prediktívne riadenie na konečnom horizonte. Ako  $\mathbf{P}$  môžeme zvoliť aj úplne inú váhu, napríklad pomocou koncového váhovania môžeme ovplyvniť aj stabilitu riadenia.

Je pravda, že z pohľadu prediktívneho riadenia práve zložená účelová funkcia je najlepší výber. Doteraz sme ale neuvažovali s obmedzeniami. Ako to neskôršie objasníme, v prítomnosti procesných obmedzení už nemôžeme hovoriť o rovnocennosti týchto dvoch metód a budeme tak môcť očakávať, že MPC bude poskytovať lepšiu kvalitu riadenia.

#### 5.2.2 Vplyv dĺžky predikčného horizontu

Vráťme sa teraz k jednoduchej formulácii prediktívneho riadenia s konečným horizontom. Inými slovami, budeme predpokladať, že matica váhovania koncového stavu je  $\mathbf{P} = \mathbf{Q}$ , a preto stratégia „vidí“ iba do konca horizontu. Ďalej si preto ukážeme, ako vplýva dĺžka horizontu  $n_p$  na spätnoväzbové riadenie.

Ak si zvolíme dostatočne dlhý horizont predikcie, mali by sme dostať veľmi podobné výsledky v spätej väzbe ako pri riadení so zloženým riadiacim zákonom. To je samozrejné, pretože pri  $n_p = \infty$  by sme jednoducho dostali LQ riadenie s ustáleným zosilnením. Pri postupnom skracovaní horizontu  $n_p \neq \infty$  však nastane zaujímavá situácia: aj keby sme mali dokonalú znalosť o dynamike systému, predikcie by boli stále odlišné od odozvy systému v spätej väzbe. Po ďalšom skrátení horizontu dokonca môžeme dostať nestabilné riadenie. To prakticky znamená, že MPC riadenie s konečným horizontom môže fungovať, ale riadenie ani v absencii procesných obmedzení nie je „optimálne“, dokonca ani stabilita nie je garantovaná<sup>2</sup>. Striktne povedané, riešenie je stále optimálne – totiž splňa kriteériálnu funkciu, ktorá teraz definuje kvalitu na konečnom horizonte – ale predĺžením horizontu by sme dostali lepšie výsledky. Inými slovami:

<sup>2</sup>Samozrejme stále môžeme vyšetriť stabilitu.



Prediktívne riadenie s konečným horizontom nemusí vždy zaručovať najlepšiu kvalitu ani stabilitu riadenia – ani v prípade, že neuvažujeme prítomnosť procesných obmedzení.

V nasledujúcom príklade môžeme vyskúšať vplyv dĺžky horizontu na vlastnosti prediktívneho riadenia:

**Príklad 5.10.** Pre diskrétny model dynamického systému daný stavovými matícami

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1.1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \quad \text{a} \quad \mathbf{C} = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (5.26)$$

uvažujeme MPC riadenie konečným horizontom, kde matica váhovania koncového stavu je  $\mathbf{P} = \mathbf{Q}$ , váhovanie stavu je  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$ , váhovanie vstupu je  $\mathbf{R} = 0.01$ , a počiatočný stav  $\mathbf{x}_0 = [0.5 \ -0.5]^T$ . Vypočítajte LQ zosilnenie  $\mathbf{K}$  a zosilnenie MPC riadiaceho zákona bez obmedzení ( $\mathbf{K}_{n_p} = \mathcal{IK}_{n_p} = \mathcal{IH}_{n_p}^{-1} \mathbf{G}_{n_p}$ ) pre horizonty  $n_p = 20, 10, 5, 4$  a 2 krokov a vyšetrite stabilitu uzavretej slučky riadenia. Graficky porovnajte stavy a vstupy v spätej väzbe s predikciami.

Začíname deklaráciou matíc dynamiky, váhovania, počiatočného stavu a do premennej np na začiatok inicializujeme 20 krokov dlhý horizont, čo neskôr budeme postupne znižovať:

Zdrojový kód 5.14. dlzkahorizontu.m

```
A=[1 1; 0 1.1]; B=[0 0.01]'; C=[-1 1]; % Dynamika
Q=C'*C; R=0.01; x=[0.5 -0.5]'; % Vahy, x0
np=20; % Horizont
```

V offline časti algoritmu MPC teraz nepočítame algebraickej riešenie Riccatiho rovnice, pretože uvažujeme riadenie s konečným horizontom, kde  $\mathbf{P} = \mathbf{Q}$ . Účelovú funkciu počítame pomocou vlastného príkazu ucelovafunkcia():

```
P=Q; % Koncove vah.
[H, G]=ucelovafunkcia (A, B, np, Q, R, P); % Ucelova f.
```

Najprv vypočítame LQ zosilnenie na základe dlqr(), potom  $\mathbf{K}_{n_p}$  pre daný horizont, z čoho pomocou výberu prvého bloku dostaneme zosilnenie metódy prediktívneho riadenia pre horizont  $\mathbf{K}_{n_p}$ . Pre toto zosilnenie vypočítame maticu dynamiky systému v uzavretej slučke a vypočítaním vlastných čísel získame informácie o stabilité:

Tabuľka 5.2. Zmena zosilnenia riadiaceho zákona a vlastné čísla matice dynamiky v závislosti od predikčného horizontu pre MPC riadenie s konečným horizontom.

$n_p$	LQ	20	10	5	4	2
$\mathbf{K}_{n_p}$ eig( $\mathbf{A} - \mathbf{B}\mathbf{K}$ )	[7.55 54.93]	[7.54 54.86]	[7.39 53.82]	[4.1418.52]	[1.65 7.52]	[-1.1 0.01]

```

K=dlqr(A,B,Q,R)                                % Riesenie LQ
Kcal=inv(H)*G;                                % K kaligraficke
K=Kcal(1,:);                                 % Prvy blok
Psi=A-B*K; eig(Psi)                           % Stabilita

```

Nasleduje spätnoväzbová simulácia online častou MPC algoritmu. Trvanie simulácie prípadne môžeme zmeniť na základe dĺžky `for` cyklu. Stavy a vstupy porovnávame s predikciou pomocou funkcií `vykreslitXpred` a `vykreslitUpred` (Pr. 5.6, Pr.5.7):

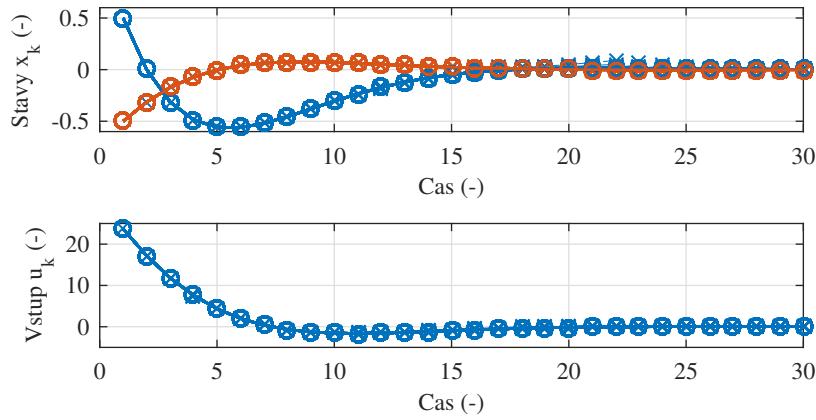
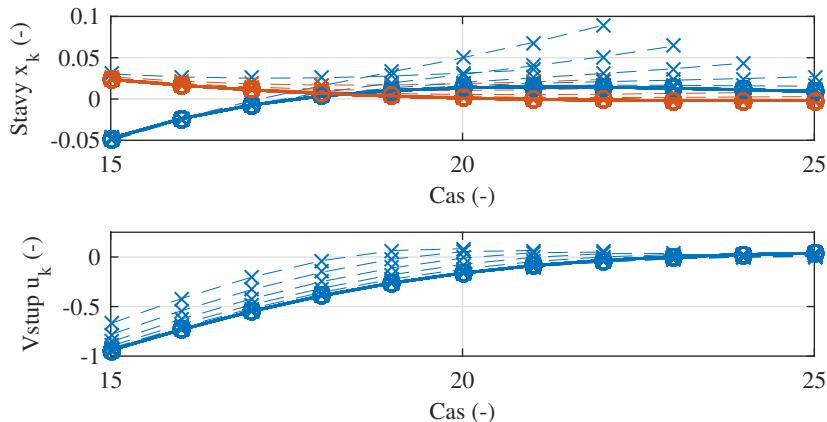
```

for i=1:30                                     % Trvanie sim.
    upr=-inv(H)*G*x;                         % Predikcia u
    u=upr(1);                                 % Prvy clen
    subplot(2,1,1)                            % Horna cast
    vykreslitXpred(x,upr,A,B,np);            % Priebeh Xpred
    subplot(2,1,2)                            % Dolna cast
    vykreslitUpred(u,upr);                   % Priebeh Upred
    x=A*x+B*u;                                % Model
end

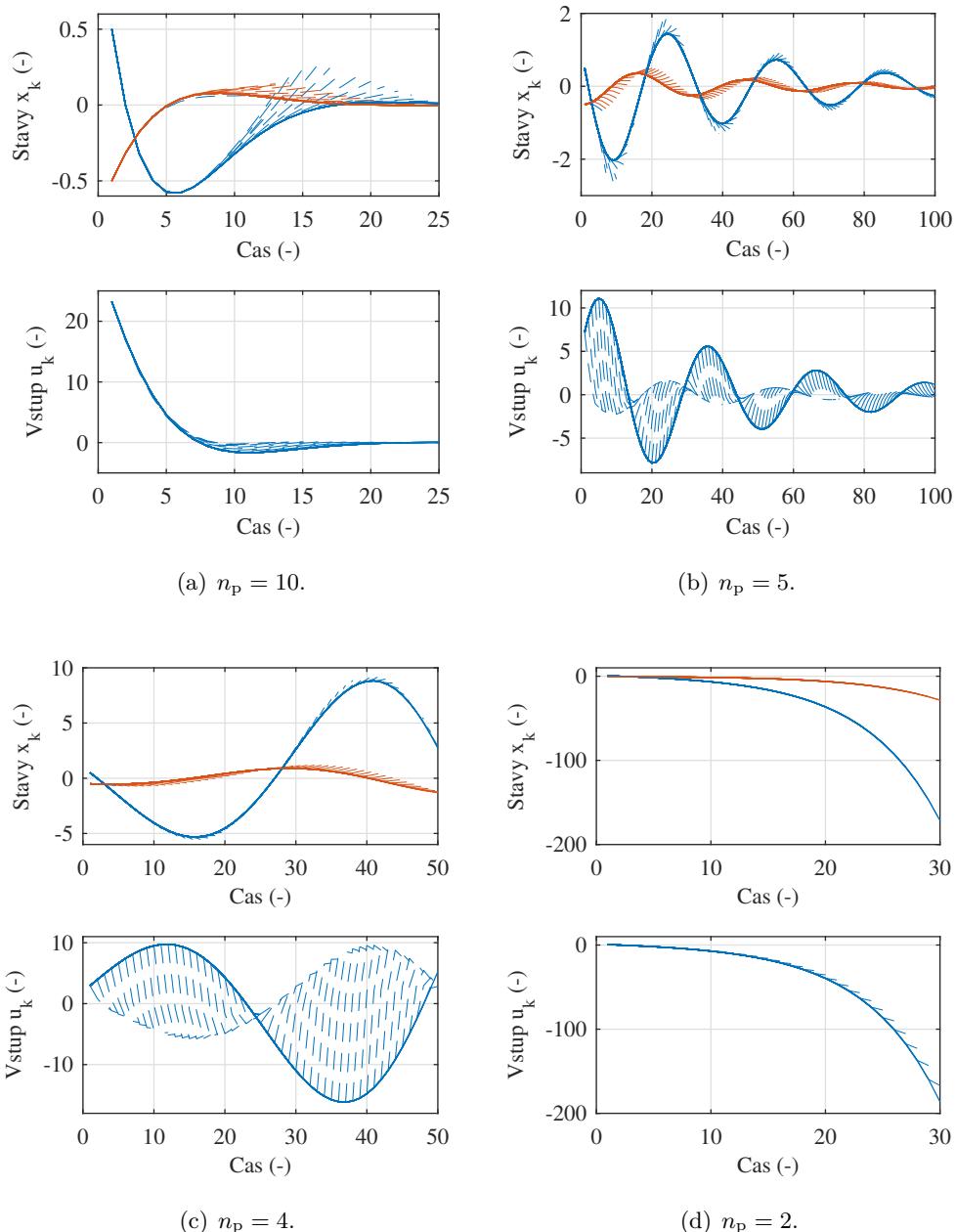
```

Ak je príklad dokončený, môžeme ďalej spustiť simuláciu pre horizont  $n_p = 20$  a dostaneme odozvu na Obr. 5.11 a zosilnenie  $\mathbf{K}_{20}$  z Tab. 5.2. Porovnaním  $\mathbf{K}_{20}$  s LQ zosilnením nevidíme veľké rozdiely a to platí aj pre vlastné čísla uzavretého obvodu. Obr. 5.11 svedčí o stabilnom riadiacom priebehu a predikcie na prvý pohľad súhlasia s uzavretou slučkou. Pri bližšom preverení priebehu na Obr. 5.10(b) vidíme, že predikcia nie je úplne presná, môžeme detegovať malé zmeny. V tomto prípade, aj keď máme dokonalú znalosť o dynamike systému a keď máme k dispozícii dokonalé meranie stavu a napriek tomu, že nemáme vonkajšie poruchy; vplyvom konečného predikčného horizontu sa predikcie nezhodujú s priebehom v uzavretej slučke. Ináč povedané, riadenie by mohlo byť lepšie.

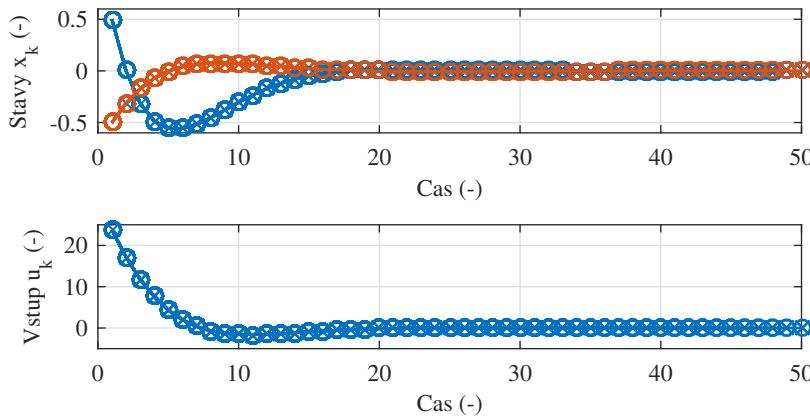
Zmenšujme dĺžku horizontu ďalej. Pri  $n_p = 10$  stále máme stabilný nekmitavý priebeh, ale rozdiel na Obr. 5.11(a) medzi uzavretou slučkou (hrubá plná čiara) a predikciami (tenšia prerusená čiara) je väčší a máme horšiu kvalitu riadenia. Znížením horizontu na  $n_p = 5$  už dramaticky zmeníme dynamiku uzavretého

(a)  $n = 20$ .(b)  $n = 20$  (detail).

Obr. 5.10. Spätnoväzbová odozva systému na vstupy (○) vs. predikcia stavov a vstup (×) pri prediktívnom riadení bez obmedzení a na konečnom horizonte  $n_p = 20$  krokov.



Obr. 5.11. Spätnoväzbová odozva systému na vstupy (—) vs. predikcia stavov a vstupu (---) prediktívneho riadenia bez obmedzení a na konečnom horizonte  $n_p = 10, 5, 4$  a 2. Ak horizont  $n_p \leq 4$ , spätnoväzbové riadenie je nestabilné.



Obr. 5.12. Spätnoväzbová odozva (○) je identická s predikovanou odozvou (×) v prípade riadenia s nekonečným horizontom pomocou zloženého riadiaceho zákona.

obvodu. Na Obr. 5.11(b) vidíme, že sice riadenie je stále stabilné, je kmitavé a má značne zníženú kvalitu. Toto potvrdí aj Tab. 5.2, kde sa prvky matice zosilnenia  $\mathbf{K}_5$  významne líšia od LQ zosilnenia a reálna časť vlastných čísel matice dynamiky uzavretého obvodu približuje k jednotke. Ak znížime predikčný horizont na  $n_p = 4$  stratíme stabilitu, o čom svedčí kmitavý nestabilný priebeh na Obr. 5.11(c) a vlastné čísla matice dynamiky, ktoré sa už nachádzajú mimo jednotkovej kružnice. Ak zvolíme  $n_p = 2$ , nestabilita priebehu je jednoznačná aj z Obr. 5.11(d) aj z vlastných čísel matice dynamiky uzavretého riadiaceho obvodu,  $\Psi = \mathbf{A} - \mathbf{B}\mathbf{K}$ .

Ak uvažujeme riadenie na nekonečnom horizonte pomocou zloženého riadiaceho zákona, teda  $\mathbf{P}$  vypočítame ako riešenie Riccatiho rovnice (napríklad  $[\mathbf{K}, \mathbf{P}] = \text{dlqr}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$ ), získame odozvu ako na Obr. 5.12. Vidíme, že pri dokonale známej dynamike systému sú predikcie totožné s odozvou v uzavretej slučke len ak uvažujeme predikcie na nekonečnom horizonte pomocou zloženej účelovej funkcie.

□

Nielen predikované vstupy a stavy sa líšia od „skutočných“ vstupov a stavov vplyvom konečného horizontu, ale aj hodnota účelovej funkcie. Ak riadime odozvu systému z počiatočného stavu do rovnovážneho a neuvažujeme poruchy, priebeh hodnoty účelovej funkcie v spätej väzbe má monotónne klesať do nuly. Ak to tak nie je, hodnota účelovej funkcie môže počas simulácie aj rásť. Jedna z takých situácií – napriek dokonalému modelu – je aj MPC riadenie s konečným horizontom. Vykreslenie účelovej funkcie má preto aj určitú diagnostickú hodnotu.

**Príklad 5.11.** Vykreslite priebeh hodnôt účelovej funkcie pre MPC riadenie systému z Pr. 5.10. Porovnajte priebeh účelovej funkcie v spätnoväzbovej simulácii pre horizonty  $n_p = 20, 10, 6, 5, 4$  a  $2$ .

Na realizáciu príkladu môžeme vychádzať zo zdrojového kódu *dlzkahorizontu.m* z Pr. 5.10. Väčšina zdrojového kódu zostáva identická, preto tu ďalej uvádzame iba čiastočný výpis. Rozšírimo výstupné premenné funkcie *ucelovafunkcia()* aj o maticu **F**:

Zdrojový kód 5.15. *dlzkahorizontuucelova.m* (Čiastočný výpis.)

```
[H, G, F] = ucelovafunkcia(A, B, np, Q, R, P); % Ucelova f.
```

V online časti stačí zavolať správnu funkciu na vykreslenie hodnôt účelovej funkcie:

```
vykreslitJ(x, upr, H, G, F) % Kreslenie J
```

Po spustení dostaneme priebeh hodnôt účelovej funkcie počas spätnoväzbovej simulácie ako na Obr. 5.13. Z predošlého príkladu (Pr. 5.10) vieme, že pri horizonte  $n_p = 20$  a  $n_p = 10$  spätnoväzbová odozva bola prakticky nerozlísiteľná od LQ odozvy. Zosilnenia  $\mathbf{K}_{20}$  a  $\mathbf{K}_{10}$  boli podobné LQ zosilneniu. Priebehy Obr. 5.13(a)–5.13(b) monotónne klesajú, voľným okom nevidíme problém.

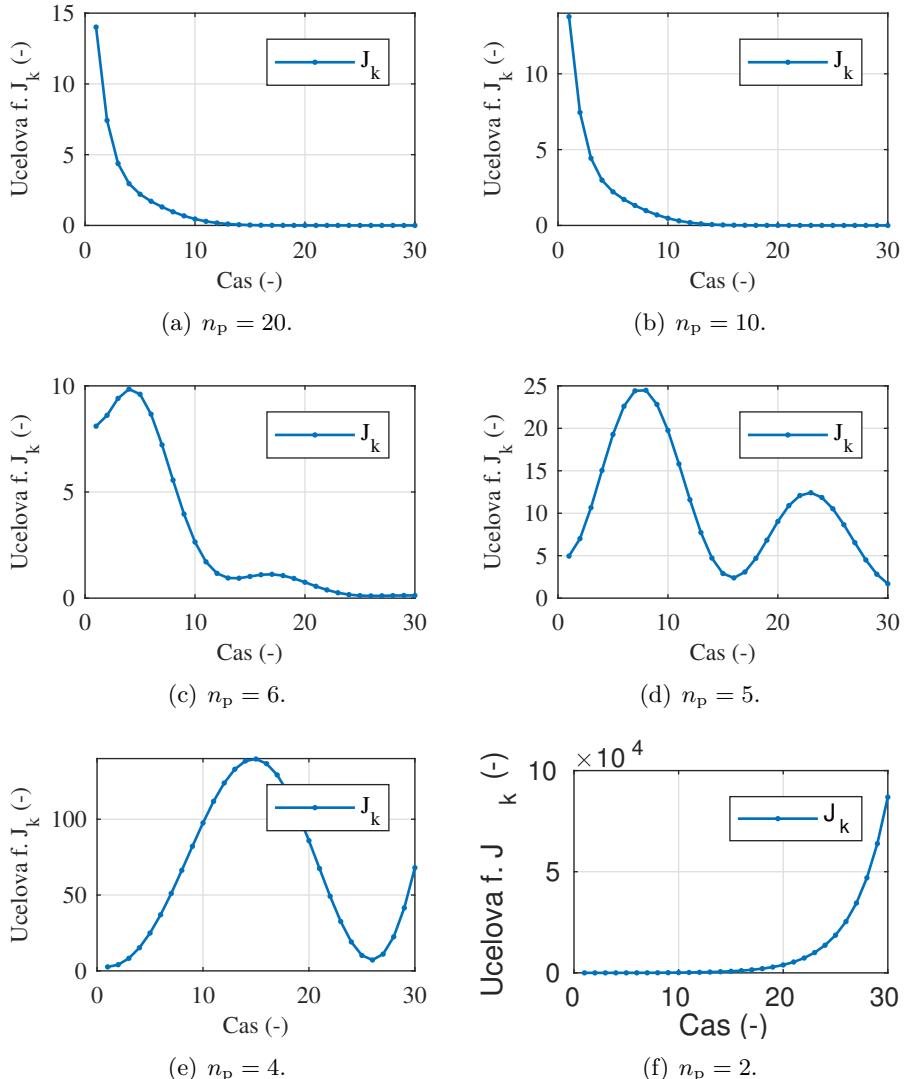
Príklad 5.10 ukázal, že od horizontu  $n_p = 4$  kroky a nižšie nastal jednoznačný problém: riadiaca slučka bola nestabilná. Podľa Obr. 5.13(c)–5.13(d) prvé náznaky problémov sú detegovateľné už skôr. Účelová funkcia v uzavretom obvode najprv začína stúpať, až potom klesá, a toto sa zopakuje aj neskôr pri vzorkách  $k = 10$ – $20$ . Hodnota účelovej funkcie v uzavretej slučke nie je zhodná s predikciou kvôli krátkemu horizontu. Na základe Pr. 5.10 vieme, že pri horizonte  $n_p = 4$  riadenie je nestabilné, čo vidíme aj na Obr. 5.13(c)–5.13(d), kde vidíme najprv kmitavý nestabilný a potom nekmitavý nestabilný priebeh hodnôt.

□

Účelovú funkciu môžeme využiť aj na ladenie algoritmu a porovnávanie výsledkov. Plocha pod priebehom účelovej funkcie v spätnoväzbovom obvode je dobrým indikátorom „kvality“ riadenia. Keďže uvažujeme diskrétny systém, hodnoty účelovej funkcie stačí sčítať. Čím je táto suma nižšia, tým lepšie riadenie máme.

**Príklad 5.12.** Vytvorte funkciu na zrátanie hodnôt účelovej funkcie. Funkciu potom otestujte systémom z Pr. 5.10.

Najprv si vytvoríme novú funkciu, nazveme ju *kvalitaJ*, keďže priamo súvisí s kvalitou regulácie. Na vstupe bude potrebovať aktuálny stav  $\mathbf{x}_k$  a vypočítané vstupy  $\vec{\mathbf{u}}_k$  a matice **H**, **G** a **F**. Na výstupe vráti numerický indikátor kvality riadenia, ktorý je sumou hodnôt účelovej funkcie:



Obr. 5.13. Hodnota účelovej funkcie  $J_k$  prediktívneho riadenia bez obmedzení na konečnom horizonte  $n_p = 20, 10, 6, 5, 4$  a 2 krokov. Hodnota účelovej funkcie by mala monotónne klesať, prvé náznaky problémov sú jasne detegovateľné už pri  $n_p = 6$ , kým uzavretý obvod stráca stabilitu pri  $n_p \leq 4$ .

Zdrojový kód 5.16. kvalitaJ.m

```
function q = kvalitaJ(x, upr, H, G, F) % Ratanie kvality
```

Deklarujeme prázdnú premennú  $j$  ako pretrvávajúcu premennú, ktorú pri prvom spustení inicializujeme nulou:

```
persistent j % Trvale premenne
if isempty(j)
    j=0; % Ak este neexist.
end % Inicializuj
% Koniec podm.
```

Indikátor kvality vypočítame ako sumu hodnôt účelovej funkcie. Výsledok  $q$  vždy vrátime na výstup:

```
j=j+upr'*H*upr+2*x'*G'*upr+x'*F*x; % Ucelova funkcia
q=j; % Vratit vysledok
end % Koniec funkcie
```

Novú funkciu môžeme otestovať napríklad so systémom a s parametrami z Pr. 5.10. Zmeníme iba niekoľko riadkov, t. j. v každej časovej iterácii zavoláme novú funkciu:

Zdrojový kód 5.17. testkvalitaJ.m (Čiastočný výpis.)

```
q=kvalitaJ(x, upr, H, G, F); % Ratanie indik.
```

a na konci iba pozrieme na hodnotu tak, že napíšeme premennú  $q$  do príkazového riadku.

Pre horizont  $n_p = 20$  krokov dostaneme  $\sum J_k = 36.76$  (-) a pre horizont  $n_p = 10$  dostaneme  $\sum J_k = 36.78$ . Napriek tomu, že pri nakreslení priebehu sme nevedeli najst rozdiel, polovičný horizont vplýval na kvalitu riadenia negatívne. Pri  $n_p = 6$  už máme  $\sum J_k = 86$ , pri  $n_p = 5$  je  $\sum J_k = 322$ , pri  $n_p = 4$  je  $\sum J_k = 1937$  a pri  $n_p = 2$  už suma hodnôt účelovej funkcie je pätfierné číslo.

□

## 5.3 Formulácia obmedzení

V predchádzajúcej diskusii sme poukázali na zaujímavý fakt, že prediktívne riadenie na nekonečnom horizonte bez procesných obmedzení je rovnocenné LQ riadeniu. Klúčový výraz je tu *bez obmedzení*. Každý reálny systém však má svoje

procesné obmedzenia, ktoré musíme dodržať. Otázkou je, či pri návrhu riadenia alebo pri overení stability riadiaceho obvodu berieme tieto obmedzenia do úvahy.

Keď zostrojíme riadiacu slučku pomocou PID alebo iných tradičných riadiacich prístupov, návrh predpokladá, že systém a akčné členy nemajú žiadne obmedzenia. Tieto algoritmy „nevedia“, že vstupy do akčných členov sa musia pohybovať medzi procesnými obmedzeniami, resp. že stavy systému sa nemôžu voľne hýbať cez stavový priestor bez ohraničení. Samozrejme, pri realizácii algoritmov musíme brať ohľad aj na technické možnosti akčných členov.

### 5.3.1 Saturácia vstupov

Bežným riešením pri praktickej implementácii tradičných algoritmov je orezávanie, nazývané tiež *saturácia* vstupov (pozri Alg. 5.3). Používame na to jednoduchú logiku: ak vstup vypočítaný riadiacim algoritmom je väčší ako maximálne dovolený vstup, tak použijeme maximálny dovolený vstup. Podobne, ak vstup vypočítaný riadením je menší ako minimálny dovolený vstup, použijeme minimálny dovolený vstup. Na ochranu akčných členov môže tento jednoduchý prístup dokonale fungovať. V čom je potom problém?

---

#### Algoritmus 5.3 Orezávanie vstupov — saturácia

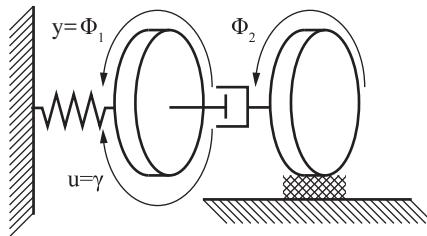
---

1: $u \leftarrow \text{PID}, LQ$ , atď.	▷ výpočet akčného zásahu
2: <b>ak</b> $u \geq u_{\max}$ <b>potom</b>	▷ ak viac ako maximum
3: $u = u_{\max}$	▷ je maximum
4: <b>inak ak</b> $u \leq u_{\min}$ <b>potom</b>	▷ ak menej ako minimum
5: $u = u_{\min}$	▷ je minimum
6: <b>inak</b>	▷ ak je v poriadku
7: $u = u$	▷ bez zmeny
8: <b>koniec ak</b>	

---

Problém je práve v tom, že navrhujeme riadiaci prístup bez obmedzení; potom v reáli robíme niečo úplne iné. To môže vplývať nielen na kvalitu regulačného pochodu, ale aj na stabilitu obvodu, najmä pri systémoch s viacerými riadenými akčnými členmi. Regulačnú slučku môžeme naladiť tak, aby sme sa vstupmi nepriblížili k ohraničeniam. Na jednej strane to predpokladá veľmi konzervatívne a neagresívne riadenie a na druhej strane stále nie je garantované, že bez saturačných limitov neporušíme obmedzenia.

**Príklad 5.13.** Pohyb elektromechanického systému môžeme ekvivalentne opísať dynamikou dvoch zotrvačníkov (Obr. 5.14). Vstup do systému je krútiaci moment a výstup je uhlové pootočenie. Zotrvačníky modelujme pomocou stavového systému tretieho rádu zo súboru *zotrvacnik.mat* (pozri *modelzotrivacnik.mat* [29]),



Obr. 5.14. Schematický náčrt elektromechanického systému dvoch zotrvačníkov.

kde stavový vektor opisuje uhlové výchylky a rýchlosťi rotačných telies podľa  $\mathbf{x} = [\Phi_1 \ \dot{\Phi}_1 \ \Phi_2]^T$ . Vykonaljte numerickú simuláciu LQ riadenia aktívneho brzdenia systému z počiatočnej polohy a rýchlosťi  $\mathbf{x}_0 = [0.2 \ 0.1 \ 0.1]^T$  pri váhovaní účelovej funkcie maticou  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  a  $\mathbf{R} = 0.01$ . Porovnajte výsledky aj so simuláciou v prípade obmedzenia krútiaceho momentu na akčnom člene na  $\pm 0.5$  pomocou saturácie vstupov.

Začneme načítaním modelu do pamäti, potom zadáme počiatočný stav, maximálne a minimálne obmedzenie akčného člena, váhy a na záver vypočítame LQ zosilnenie:

Zdrojový kód 5.18. zotrvacnikorezavanie.m

```

load zotrvacnik                                % Model a maticy
A=zotrvacnik.a; B=zotrvacnik.b; C=zotrvacnik.c;
X0=[0.2 0.1 0.1]';                                % Počiatocny stav
umax=0.5; umin=-umax;                            % Obmedzenia
Q=C'*C; R=0.01;                                  % Vahovanie nom.
K=-dlqr(A,B,Q,R);                            % LQ nom.

```

Priebeh stavov, vstupu a výstupu si počas simulácie uložíme do pomocných premenných. V uzavretej slučke vypočítame akčný zásah podľa ustáleného LQ zosilnenia a používame ho priamo v modeli napríklad pomocou:

```

Xlq=X0;                                         % Pomocna prem.
for i=1:100                                     % Dlzka sim.
    Ulq(i)=K*Xlq(:,i);                          % LQ riadenie u
    Xlq(:,i+1)=A*Xlq(:,i)+B*Ulq(i);           % Stavy x
    Ylq(i)=C*Xlq(:,i);                          % Vystupy y
end

```

Pokračujeme ďalšou simuláciou, kde vstupy vypočítané LQ zákonom orežeme na základe maximálnych a minimálnych dovolených hodnôt. Takto saturované vstupy potom používame na výpočet odozvy:

```

Xsat=X0;                                % Pomocna prem.
for i=1:100
    Usat(i)=K*Xsat(:,i);                % Dlzka sim.
    if Usat(i)>=umax                   % LQ riadenie u
        Usat(i)=umax;                  % u viac ako max
    end
    if Usat(i)<=umin                   % u menej ako min
        Usat(i)=umin;                  % u je min
    end
    Xsat(:,i+1)=A*Xsat(:,i)+B*Usat(i); % Stavy x
    Ysat(i)=C*Xsat(:,i);                % Vystupy y
end

```

Výsledky vykreslíme pre LQ riadenie bez obmedzení ako aj saturované LQ:

```

subplot(2,1,1)                            % Vystupy
plot(Ylq); hold on; plot(Ysat); grid on
xlabel('Cas (-)'); ylabel('Vystup y (rad)')
legend('y_{LQ}', 'y_{LQ-Sat}')

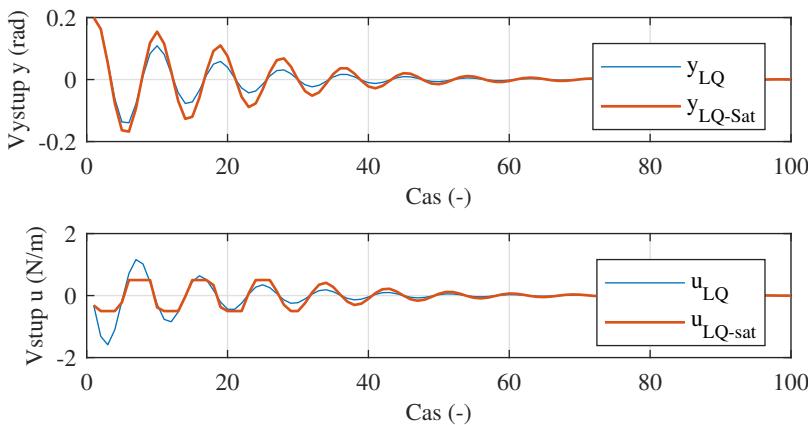
subplot(2,1,2);                           % Vstupy
plot(Ulq); hold on; plot(Usat); grid on
xlabel('Cas (-)'); ylabel('Vstup u (N/m)');
legend('u_{LQ}', 'u_{LQ-sat}')

```

Dostaneme odozvu znázornenú na Obr. 5.15. V prípade LQ riadenia bez obmedzení akčné zásahy presahujú  $\pm 1$  N/m a vplyvom aktívneho brzdenia systém zastavíme efektívne a rýchlo. Ak kvôli technologickým obmedzeniam vstupov musíme saturovať akčné zásahy, dostaneme priebeh horšej kvality. Vypočítali sme pomerne agresívne LQ optimálne riadenie, no v konečnom dôsledku ho nemôžeme používať na obmedzenie akčného člena.

□

**Príklad 5.14.** Pokračujte v Pr. 5.13 nájdením takého ladenia LQ regulátora, pri ktorom akčné zásahy ostanú v dovolenom rozsahu. Porovnajte výsledky s priebehom riadenia bez obmedzení.



Obr. 5.15. Porovnanie LQ riadenia bez obmedzení a so saturovaným (orezávaným) LQ riadením.

Teraz simulujme situáciu, keď pri návrhu riadiaceho zákona už budeme uvažovať vplyv obmedzení a zvolíme takú stratégiu, že sa snažíme nalaďiť LQ zákon tak, aby sme neporušili obmedzenia.

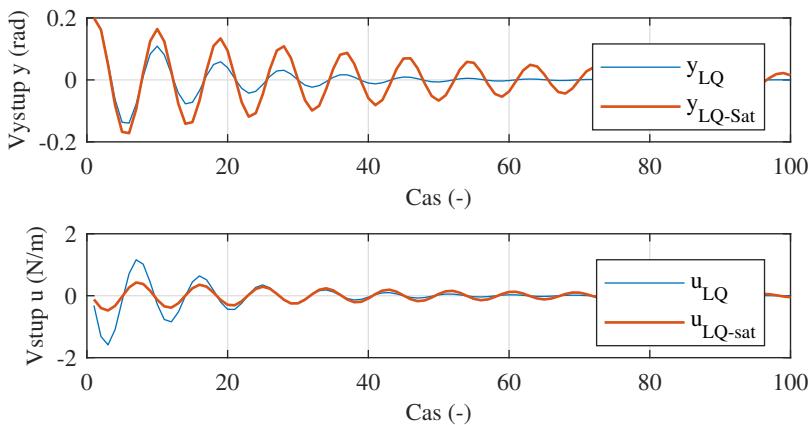
Prevažná väčšina zdrojového kódu zostáva rovnaká ako v Pr. 5.13, ďalej si ukážeme iba rozdiely. Vypočítajte alternatívne LQ zosilnenie pre saturované riadenie a použite ho v simulácii. Zvoľte si napríklad o rád menej „agresívne“ váhovanie  $\mathbf{R} = 0.1$ :

Zdrojový kód 5.19. zotravacnikladenie.m

```
R2=0.1; % Váhovanie nom.
K2=-dlqr(A,B,Q,R2); % LQ nom.
Xsat=X0; % Pomocna prem.
for i=1:100 % Dĺžka sim.
    Usat(i)=K2*Xsat(:,i); % LQ riadenie u
    if Usat(i)>=umax % u viac ako max
```

Spustením simulácie dostaneme odozvu na Obr. 5.16. Riadenie bez obmedzení je rovnaké ako v predchádzajúcim príklade. Tentokrát sme si v záujme dodržania procesných obmedzení zvolili konzervatívnejšie váhovanie vstupov. Ako vidíme, akčné zásahy zostávajú v dovolených medziach. Takéto konzervatívne ladenie na druhej strane dramaticky vplýva na kvalitu riadenia.

Konzervatívne akčné zásahy naviac stále negarantujú dodržanie obmedzení na



Obr. 5.16. Porovnanie LQ riadenia bez obmedzení s konzervatívne naladeným saturovaným LQ riadením

vstupy: na systém môže vplyváť taká vonkajšia porucha, pri ktorej už LQ zákon vypočíta vyššie vstupy ako sú dovolené alebo naladené. Práve preto musíme aj pri konzervatívnom ladení využívať saturáciu vstupov.

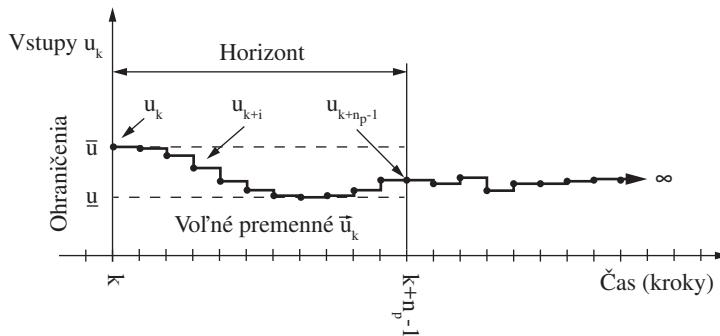
□

Sila metódy prediktívneho riadenia je práve v tom, že procesné obmedzenia uvažuje prirodzene – už pri návrhu samotného algoritmu. Prediktívne riadenie je jediným z veľmi efektívnych prístupov, ktorý uvažuje obmedzenia na úrovni algoritmu a ošetruje obmedzenia stavov systému [101].

Teraz sa pozrieme bližšie na to, ako vôbec môžeme obmedzenia matematicky definovať tak, aby sme s nimi mohli počítať v optimalizačnej úlohe na počítači. Začneme najčastejším typom procesných obmedzení: obmedzeniami na vstupy.

### 5.3.2 Obmedzenia vstupov

Obmedzenia na absolútne hodnotu amplitúdy vstupov sú pravdepodobne najčasťejšie využívané procesné obmedzenia. Každý akčný člen má svoje limity, ktoré sú definované na základe technických, ekonomických alebo bezpečnostných dôvodov. Napríklad pre elektrický motor môžeme zadať minimálne a maximálne otáčky, pre servomotor maximálny a minimálny uhol otočenia, alebo pre vykurovacie teliesko maximálne napätie atď.



Obr. 5.17. Ilustrácia ohraničení absolútnej hodnoty vstupov.

Minimálny dovolený akčný zásah si ďalej ošetríme pomocou dolného ohraničenia<sup>3</sup>, čo si v nasledujúcich kapitolách označíme symbolom  $\underline{u}$ . Podčiarknutie znamená, že ide o minimálnu dovolenú amplitúdu akčných zásahov. Všimnime si, že  $\underline{u}$  je všeobecne vektor ohraničení, pretože môžeme mať viacero akčných členov a definujeme limity pre každú z nich. Podobne zadávame aj maximálne dovolené akčné zásahy pomocou horného ohraničenia<sup>4</sup>  $\bar{u}$ , kde čiara nad premennou označuje najväčšie dovolené amplitúdy vstupov.

Ohraničenia amplitúdy vstupov potom definujú nerovnosť

$$\bar{u} \leq u_{k+i} \leq \underline{u}, \quad (5.27)$$

pre  $i = 0, 1, 2, \dots, n_p - 1$ . Všimnime si dôležitú podmienku, ktorá značí, že nestačí dodržať obmedzenia pre aktuálny vstup  $u_k$ , ale obmedzenia musia platiť počas celého horizontu (Obr. 5.17).

Ohraničenia vo forme dvoch nerovností nie sú praktické pre algoritmizáciu, preto si ich rozdelíme na dve nezávislé nerovnosti. Pravú stranu s horným obmedzením môžeme uviesť bez zmien, kým dolné ohraničenie transformujeme na ekvivalentnú formu a dostaneme

$$u_{k+i} \leq \bar{u} \quad (5.28a)$$

$$-u_{k+i} \leq -\underline{u} \quad (5.28b)$$

pre  $i = 0, 1, \dots, n_p - 1$ . Toto znamená, že pre každý vstup počas trvania horizontu musíme dodržať podmienku (5.28a) pre horné a podmienku (5.28b) pre

<sup>3</sup>angl. lower bound

<sup>4</sup>angl. upper bound

dolné ohraničenie. Uvažujme teraz nerovnosť iba pre horné obmedzenie vstupov a rozpišme si ju na horizonte v nasledujúcom maticovom tvare

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u}_k \\ \mathbf{u}_{k+1} \\ \vdots \\ \mathbf{u}_{k+n_p-2} \\ \mathbf{u}_{k+n_p-1} \end{bmatrix} \leq \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix} \bar{\mathbf{u}}. \quad (5.29)$$

kde  $\mathbf{I}$  je jednotková matica rozmeru  $n_u \times n_u$ ,  $\mathbf{0}$  je matica nul rozmeru  $n_u \times n_u$ . Vstupy máme vyjadrené v rámci vektora predikovaných vstupov, a teda musí platiť

$$\mathbf{I} \vec{\mathbf{u}}_k \leq \mathbf{1} \bar{\mathbf{u}}, \quad (5.30)$$

kde  $\mathbf{I}$  je jednotková matica rozmeru  $n_u \cdot n_p \times n_u \cdot n_p$  a  $\mathbf{1}$  označuje blokový stĺpcový vektor (v konečnom dôsledku matica) jednotkových matíc rozmeru  $n_u \cdot n_p \times n_u$ . Tento proces môžeme analogicky zopakovať aj pre prípad dolných ohraničení, kde dostaneme

$$-\mathbf{I} \vec{\mathbf{u}}_k \leq -\mathbf{1} \bar{\mathbf{u}}. \quad (5.31)$$

Maticové nerovnosti (5.30) a (5.31) môžeme ďalej kompaktne zapísť v tvare nasledujúcej maticovej nerovnosti:

$$\begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \vec{\mathbf{u}}_k \leq \begin{bmatrix} \mathbf{1} \bar{\mathbf{u}} \\ -\mathbf{1} \bar{\mathbf{u}} \end{bmatrix}. \quad (5.32)$$

Tento tvar nerovností bude využiteľný aj pre iné typy označení. Všimnime si, že násobíme maticu optimalizačnou premennou  $\vec{\mathbf{u}}_k$ , tým dostaneme vektor a každý prvok výsledného vektora musí byť menší alebo rovnať sa príslušnému prvku stĺpcového vektora na pravej strane. Označme vo všeobecnosti maticu, ktorou násobíme optimalizačnú premennú  $\mathbf{A}_c$  a označme vektor na pravej strane  $\mathbf{b}_0$ <sup>5</sup>.

Potom pre obmedzenia vo všeobecnosti platí

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_0, \quad (5.33)$$

kde v prípade obmedzení na absolútnu hodnotu amplitúdy vstupov máme

$$\mathbf{A}_c = \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \quad \text{a} \quad \mathbf{b}_0 = \begin{bmatrix} \mathbf{1} \bar{\mathbf{u}} \\ -\mathbf{1} \bar{\mathbf{u}} \end{bmatrix}, \quad (5.34)$$

kde  $\mathbf{A}_c \in \mathbb{R}^{2 \cdot n_u \cdot n_p \times n_u \cdot n_p}$  a  $\mathbf{b}_0 \in \mathbb{R}^{2 \cdot n_u \cdot n_p \times 1}$ .

<sup>5</sup>Dolný index vychádza z anglického výrazu na obmedzenia: constraints.

**Príklad 5.15.** Maximálne vstupné napätie malého jednosmerného elektrického motora je 6 V. Vplyvom statických elektromechanických javov motor nie je možné roztočiť, ak vstupné napätie je menšie ako 2 V. Vytvorte matice  $\mathbf{A}_c$  a  $\mathbf{b}_0$  na opis obmedzení vstupov, ak predikčný horizont má dĺžku  $n_p = 3$  kroky.

Dolné obmedzenie je  $\bar{u} = 3$  V, horné obmedzenie je  $\underline{u} = 6$  V a máme iba jeden vstup, t. j.  $n_u = 1$ ). Najprv si deklarujeme všetko, čo poznáme – horizont predikcie a obmedzenia na vstup:

Zdrojový kód 5.20. motorobmedzenievstupu.m

```
np=3; % [-] Horizont
ul=2; % [V] Horne obmedzenie
uh=6; % [V] Dolne obmedzenie
```

Maticu  $\mathbf{A}_c$  môžeme v MATLABe vytvoriť jednoducho pomocou príkazu `eye()`. Matica  $\mathbf{1}$  je bloková matica jednotkových matíc, ale keďže máme iba jeden vstup, dostaneme vektor rozmeru  $n_p \cdot n_u \times 1$ , ktorý si vytvoríme pomocou `ones()`. Na základe vzťahov (5.34) potom dostaneme:

```
Ac=[eye(np); -eye(np)] % Matica Ac
One=ones(np,1); % Matica (vektor) 1
b0=[One*uh;-One*ul] % Matica b0
```

Výsledkom je matica  $\mathbf{A}_c$

```
Ac =
1 0 0
0 1 0
0 0 1
-1 0 0
0 -1 0
0 0 -1
```

a vektor  $\mathbf{b}_0$

```
b0 = 6
6
6
-2
-2
-2
```

ktoré vyjadrujú obmedzenia na amplitúdu vstupov platné počas trvania predikčného horizontu podľa vzťahu (5.27). □

**Príklad 5.16.** Hladina tekutiny v nádrži je udržiavaná v rozmedzí 0.2 m a 0.4 m. Teplota tekutiny musí byť najmenej  $60^\circ\text{C}$ , avšak nie viac ako  $90^\circ\text{C}$ . Čerpadlo a vykurovacie telo je riadené v lokálnej späťnej väzbe. Vytvorte matice  $\mathbf{A}_c$  a  $\mathbf{b}_0$  na opis obmedzení vstupov, ak predikčný horizont je  $n_p = 4$  kroky.

Na rozdiel od Pr. 5.15 tu máme viac akčných členov: čerpadlo  $u_1$  a vykurovacie telo  $u_2$ . Vektor dolného obmedzenia potom je  $\underline{\mathbf{u}} = [0.2 \ 60]^\text{T}$  a vektor horného obmedzenia vstupom bude  $\bar{\mathbf{u}} = [0.4 \ 90]^\text{T}$ .

Začneme definovaním horizontu predikcie, obmedzení na vstupy a zistíme si počet vstupov:

Zdrojový kód 5.21. nadrzobmedzenievstupu.m

```
np=4;                                     % [-] Horizont
ul=[0.2 60]';                            % [m, C] Horne obm.
uh=[0.4 90]';                            % [m, C] Dolne obm.
nu=length(uh);                          % Pocet vstupov
```

V prípade  $\mathbf{A}_c$  postupujeme rovnako ako v Pr. 5.15:

```
Ac=[eye(np*nu); -eye(np*nu)]           % Matica Ac
```

Teraz máme viac vstupov, preto musíme blokový vektor  $\mathbf{1}$  vytvoriť zvlášť a potom z toho na základe vzťahu (5.34) dostaneme aj vektor  $\mathbf{b}_0$ :

```
One=[];                                    % Prazdna matica 1
for i=1:np
    One=[One; eye(nu)];
end
b0=[One*uh;-One*ul];                     % Cez horizont
                                         % Vyplnenie matic 1
                                         % Koniec cyklu
                                         % Matica B0
```

Po spustení programu dostaneme výrazy, ktoré kompaktne opisujú nerovnosť (5.27).

□

**Príklad 5.17.** Vytvorte univerzálnu funkciu na výpočet  $\mathbf{A}_c$  a  $\mathbf{b}_0$  na definovanie obmedzenia amplitúdy vstupov. Funkčnosť navrhnutej funkcie otestujte pomocou zadania z Pr. 5.16.

Funkciu si nazvime **obmedzeniavstupov**. Na výstupe musí vrátiť  $\mathbf{A}_c$  a  $\mathbf{b}_0$ , kým na vstupe potrebujeme zadať obmedzenia a dĺžku horizontu:

Zdrojový kód 5.22. obmedzeniavstupov.m

```
function [Ac b0] = obmedzeniavstupov(ul,uh,np) % Funkcia
```

Ďalej môžeme využiť zdrojový kód z Pr. 5.16 v prakticky identickej podobe:

```

nu=length(uh); % Počet vstupov
Ac=[eye(np*nu); -eye(np*nu)]; % Matica Ac
One=[];
for i=1:np
    One=[One; eye(nu)]; % Vyplnenie maticy 1
end % Koniec cyklu
b0=[One*uh;-One*ul]; % Matica b0

```

Správnu činnosť novej funkcie môžeme otestovať pomocou zadania z Pr. 5.16:

Zdrojový kód 5.23. testobmedzeniavstupov.m

```

np=4; % Horizont
ul=[0.2 60]'; uh=[0.4 90]'; % Obmedzenia
[Ac Bc]=obmedzeniavstupov(ul,uh,np) % Test funkcie

```

Dostaneme identické výsledky ako v predchádzajúcom príklade.

□

### 5.3.3 Obmedzenia stavov

Jeden z dôvodov, prečo moderné riadiace metódy využívajú stavovú reprezentáciu pri návrhu je to, že pri modelovaní dynamických javov môžeme počítať aj s vnútornými veličinami, ktoré nie sú priamo merateľné. To znamená, že môžeme špecifikovať a jemne ladiť vnútorné procesy riadeného systému.

Predstavme si teraz situáciu, že máme aktívne riadenie podvozku vozidla. Jedinou merateľnou veličinou je zrýchlenie v jednom bode, ale v stavovom modeli máme posunutie na úplne inom mieste. Ako by sme teda definovali limitné dovolené hodnoty tejto veličiny? Jedine obmedzeniami na stavu modelu dynamiky riadeného systému.

Ak začneme rozmýšľať nad tým, ako by sme dokázali dodržať tieto hranice pomocou LQ riadenia, neprišli by sme na žiadne jednoduché riešenie. Nemôžeme na to totiž použiť orezávanie (saturáciu) tak ako v prípade vstupov. Veľmi efektívnym prístupom umožňujúcim zohľadniť stavové obmedzenia je práve prediktívne riadenie [101]. Pozrime sa najprv na spôsob, ako vôbec môžeme formulovať stavové obmedzenia.

Pre maximálnu absolútну hodnotu amplitúdy stavov budeme používať horné obmedzenie stavového vektora  $\bar{x}$ , a pre minimálnu povolenú hodnotu  $\underline{x}$ . Tým

pádom musia byť stavy na horizonte predikcie väčšie ako dolné a zároveň menšie ako horné ohraničenia, t. j.

$$\mathbf{x} \leq \mathbf{x}_{k+i} \leq \bar{\mathbf{x}}, \quad (5.35)$$

pre  $i=1,2,\dots,n_p$ . Nechceme tiež, aby ohraničenia na stavy boli dodržané iba pre ďalší stav, ale na celom horizonte predikcie.

Podobne ako pri obmedzení amplitúdy vstupov aj tu by bolo lepšie vyjadriť kombináciu dvoch nerovností jednou kompaktnou nerovnosťou. Platí teda, že

$$\mathbf{x}_{k+i} \leq \bar{\mathbf{x}}, \quad (5.36)$$

$$-\mathbf{x}_{k+i} \leq -\underline{\mathbf{x}}, \quad (5.37)$$

pre  $i = 1, \dots, n_p$ . Podobne ako v predchádzajúcom prípade môžeme výraz ďalej zjednodušiť rozpisáním postupnosti predikovaných stavov na horizonte v tvare

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{x}_{k+2} \\ \vdots \\ \mathbf{x}_{k+n_p-1} \\ \mathbf{x}_{k+n_p} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix} \mathbf{x}_k. \quad (5.38)$$

kde  $\mathbf{I}$  je jednotková matica rozmeru  $n_x \times n_x$  a  $\mathbf{0} = 0^{n_x \times n_x}$ . Obmedzenia stavov môžeme pre horné ohraničenie amplitúdy zapísat kompaktne ako

$$\mathbf{I} \vec{\mathbf{x}}_k \leq \mathbf{1} \bar{\mathbf{x}} \quad (5.39)$$

kde  $\mathbf{I}$  je jednotková matica rozmeru  $n_x \cdot n_p \times n_x \cdot n_p$  a  $\mathbf{1}$  označuje blokový stĺpcový vektor jednotkových matíc rozmeru  $n_x \cdot n_p \times n_x$ . Analogicky pre prípad dolného obmedzenia stavov dostaneme

$$-\mathbf{I} \vec{\mathbf{x}}_k \leq -\mathbf{1} \underline{\mathbf{x}}. \quad (5.40)$$

Tu na rozdiel od obmedzení na vstupy musíme vyriešiť ešte jednu dôležitú záležitosť. Hľadáme budúce vstupy, ktoré závisia na trajektórii stavov, hoci tu chceme ohraničiť práve túto trajektóriu. Navyše v našom optimalizačnom probléme (5.5) vôbec nevystupujú stavy  $\mathbf{x}_{k+i}$ , iba aktuálne meranie alebo odhad  $\mathbf{x}_k$ , čo už ovplyvňovať tak či tak nevieme. Riešením je reformulácia stavových obmedzení tak, aby trajektória budúcich stavov bola vyjadrená pomocou optimalizačnej premennej  $\vec{\mathbf{u}}_k$ . Používame na to predikčnú rovnicu (3.12), ktorú dosadíme do nerovností (5.39) a (5.40) a dostaneme

$$\mathbf{I}(\mathbf{M}\mathbf{x}_k + \mathbf{N}\vec{\mathbf{u}}_k) \leq \mathbf{1} \bar{\mathbf{x}}, \quad (5.41)$$

$$-\mathbf{I}(\mathbf{M}\mathbf{x}_k + \mathbf{N}\vec{\mathbf{u}}_k) \leq -\mathbf{1} \underline{\mathbf{x}}, \quad (5.42)$$

čo po prenásobení členov predikčnej rovnice zapíšeme ako

$$\mathbf{M}\mathbf{x}_k + \mathbf{N}\vec{\mathbf{u}}_k \leq \mathbf{1}\bar{\mathbf{x}}, \quad (5.43)$$

$$-\mathbf{M}\mathbf{x}_k - \mathbf{N}\vec{\mathbf{u}}_k \leq -\mathbf{1}\underline{\mathbf{x}}. \quad (5.44)$$

Pripomeňme si, že aktuálny stav  $\mathbf{x}_k$  je v každej vzorke daný a nie je to funkciou optimalizačnej premennej  $\vec{\mathbf{u}}_k$ . Premiestnime teraz výraz  $\mathbf{M}\mathbf{x}_k$  na pravú stranu nerovností a dostaneme nasledujúcu sústavu nerovností

$$\mathbf{N}\vec{\mathbf{u}}_k \leq \mathbf{1}\bar{\mathbf{x}} - \mathbf{M}\mathbf{x}_k, \quad (5.45)$$

$$-\mathbf{N}\vec{\mathbf{u}}_k \leq -\mathbf{1}\underline{\mathbf{x}} + \mathbf{M}\mathbf{x}_k, \quad (5.46)$$

kde na pravej strane opäť vidíme sekvenču budúcich vstupov. V maticovom tvare môžeme vyjadriť obmedzenia stavu ako

$$\begin{bmatrix} \mathbf{N} \\ -\mathbf{N} \end{bmatrix} \vec{\mathbf{u}}_k \leq \begin{bmatrix} \mathbf{1}\bar{\mathbf{x}} - \mathbf{M}\mathbf{x}_k \\ -\mathbf{1}\underline{\mathbf{x}} + \mathbf{M}\mathbf{x}_k \end{bmatrix}. \quad (5.47)$$

Získaný výsledok pre čas  $k$  sa opäť podobá všeobecnej formulácii nerovnosti (5.33) ( $\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_0$ ), ale obsahuje aj ďalší člen, ktorý je závislý na aktuálnom stave  $\mathbf{x}_k$ :

$$\begin{bmatrix} \mathbf{N} \\ -\mathbf{N} \end{bmatrix} \vec{\mathbf{u}}_k \leq \begin{bmatrix} \mathbf{1}\bar{\mathbf{x}} \\ -\mathbf{1}\underline{\mathbf{x}} \end{bmatrix} + \begin{bmatrix} -\mathbf{M} \\ \mathbf{M} \end{bmatrix} \mathbf{x}_k. \quad (5.48)$$

Obmedzenia na stav potom môžeme vyjadriť vo všeobecnom tvare

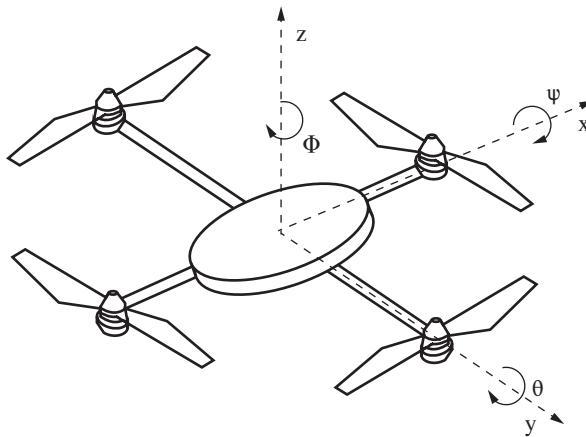
$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k, \quad (5.49)$$

kde

$$\mathbf{A}_c = \begin{bmatrix} \mathbf{N} \\ -\mathbf{N} \end{bmatrix}, \quad \mathbf{b}_0 = \begin{bmatrix} \mathbf{1}\bar{\mathbf{x}} \\ -\mathbf{1}\underline{\mathbf{x}} \end{bmatrix} \quad \text{a} \quad \mathbf{B}_0 = \begin{bmatrix} -\mathbf{M} \\ \mathbf{M} \end{bmatrix}. \quad (5.50)$$

**Príklad 5.18.** Autonómny lietajúci robot – dron – sa pohybuje v uzavretej hale a má k dispozícii presný odhad svojej polohy v jednotlivých osiach  $x$ ,  $y$  a  $z$  v miestnosti na základe vnútorného lokalizačného podsystému (Obr. 5.18). Na základe magnetometra poznáme aj uhlovú orientáciu definovanú na základe tzv. Eulero-vých uhlov  $\psi$ ,  $\theta$  a  $\phi$ . Pomocou inerciálnych meraní a algoritmu odhadu vieme odhadnúť aj rýchlosťi  $\dot{x}$ ,  $\dot{y}$  a  $\dot{z}$  a uhlové rýchlosťi  $\dot{\psi}$ ,  $\dot{\theta}$  a  $\dot{\phi}$  pomocou gyroskopu. Dynamický stav dronu môžeme potom opísť v ľubovoľnom čase nasledujúcim stavovým vektorom:

$$\mathbf{x} = [x \ y \ z \ \psi \ \theta \ \phi \ \dot{x} \ \dot{y} \ \dot{z} \ \dot{\psi} \ \dot{\theta} \ \dot{\phi}]^T.$$



Obr. 5.18. Schematický náčrt konfigurácie kvadrokoptéry v priestore.

Nulový bod uvažujeme v dolnom rohu miestnosti a hala má rozmery  $20 \times 10 \times 5$  m. Dovolená rýchlosť v každom smere je  $\pm 15$  m/s, dovolené pootočenie  $\pm 1.5$  rad, a dovolená uhlová rýchlosť  $\pm 1$  rad/s. Horizont predikcie je  $n_p = 3$  kroky. Vytvorte matice potrebné na definíciu obmedzení stavov. Linearizovaný model dronu je uložený v súbore *dron.mat* (pozri *modeldronu.m* a [130]).

Na začiatok si vyjadríme vektory dolných a horných obmedzení. Na základe zadania teda platí, že

$$\begin{aligned}\bar{\mathbf{x}} &= [20 \ 10 \ 5 \ 1.5 \ 1.5 \ 1.5 \ 15 \ 15 \ 15 \ 1 \ 1 \ 1]^T, \\ \mathbf{x} &= [0 \ 0 \ 0 \ -1.5 \ -1.5 \ -1.5 \ -15 \ -15 \ -15 \ -1 \ -1 \ -1]^T.\end{aligned}$$

Začíname s načítaním modelu, deklaráciou dĺžky predikčného horizontu a obmedzení:

Zdrojový kód 5.24. stavovedron.m

```
load dron; A=dron.a; B=dron.b; % Model
[nx nu]=size(B); % Rozmery
np=3; % Horizont
xh=[20 10 5 1.5 1.5 1.5 15 15 15 1 1 1]'; % Obmedzenia
xl=[0 0 0 -1.5 -1.5 -1.5 -15 -15 -15 -1 -1 -1]';
```

Tentoraz potrebujeme aj predikčné matice  $\mathbf{M}$  a  $\mathbf{N}$ , na základe ktorých môžeme priamo vytvoriť matice  $\mathbf{A}_c$  a  $\mathbf{B}_0$ :

```
[M, N]=predikcia(A, B, np); % Predikcne m.
Ac=[ N; -N] % Matica Ac
B0=[ -M; M] % Matica B0
```

Maticu  $b_0$  vytvoríme podobne ako v prípade obmedzení vstupov, ale tentokrát efektívne využijeme príkaz **repmat**:

```
b0=[ repmat(eye(nx), np, 1)*xh; % Matica b0
-repmat(eye(nx), np, 1)*xl]; % ...
```

Po spustení zdrojového kódu dostaneme matice potrebné na udefinovanie stavových obmedzení.  $\square$

**Príklad 5.19.** Vytvorte univerzálnu funkciu na výpočet matíc nerovností, ktoré definujú stavové obmedzenia na horizonte. Otestujte novú funkciu na Pr. 5.18.

Funkciu si pomenujme **obmedzeniastavov()**. Výstupné argumenty funkcie budú matice  $A_c$ ,  $b_0$  a  $B_0$  kým na jej vstupe budeme potrebovať horné a dolné obmedzenia stavov, horizont predikcie a model systému:

Zdrojový kód 5.25. obmedzeniastavov.m

```
function[Ac b0 B0] = obmedzeniastavov(xl, xh, np, A, B)
```

Ďalej môžeme využívať zdrojový kód z predchádzajúceho príkladu. Vyčítame si počet stavov a vytvoríme predikčné matice. Z predikčných matíc môžeme priamo vytvoriť bloky  $A_c$  a  $B_0$ , potom pomocou  $\mathbf{1}$  aj  $b_0$ :

```
[nx nu]=size(B); % Rozmery
[M, N]=predikcia(A, B, np); % Predikcne m.
Ac=[ N; -N]; % Matica Ac
B0=[ -M; M]; % Matica B0
b0=[ repmat(eye(nx), np, 1)*xh; % Matica b0
-repmat(eye(nx), np, 1)*xl]; % ...
```

Výslednú funkciu testujeme pomocou zadania z Pr. 5.18 takto:

Zdrojový kód 5.26. testobmedzeniastavov.m

```
load dron; A=dron.a; B=dron.b; np=3 % Model
xh=[20 10 5 1.5 1.5 1.5 15 15 15 1 1 1]'; % Obmedzenia
xl=[0 0 0 -1.5 -1.5 -1.5 -15 -15 -15 -1 -1 -1]';
[Ac b0 B0] = obmedzeniastavov(xl, xh, np, A, B) % Funkcia
```

$\square$

Ako zistujeme, matice sú identické ako v Pr. 5.18 a množina nerovností presne definuje ohraničenia stavov.

### 5.3.4 Kombinácia rôznych typov obmedzení

Často uvažujeme viaceré typy procesných obmedzení súčasne. Spôsob kombinácie jednotlivých typov procesných obmedzení je jednoduchý, stačí blokovo kombinovať jednotlivé matice do blokového vektora a uložiť tieto zložky „nad seba“.



Procesné obmedzenia budeme vyjadrovať vo všeobecnom tvare

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_c, \quad (5.51)$$

kde  $\mathbf{A}_c$  je matica  $\mathbf{A}_c \in \mathbb{R}^{n_o \times n_u \cdot n_p}$ ,  $\mathbf{b}_c$  je stípcový vektor  $\mathbf{b}_c \in \mathbb{R}^{n_o \times 1}$  a  $n_o$  je počet jednotlivých nešpecifikovaných obmedzení. Môžeme vymyslieť aj nové typy procesných obmedzení; ak ich vieme vyjadriť v tvare nerovnosti (5.51), môžu byť súčasťou formulácie MPC.

V prípade, že uvažujeme iba obmedzenia vstupov, máme jednoduchú úlohu:  $\mathbf{b}_c = \mathbf{b}_0$  podľa Rov. (5.34). V prípade stavových obmedzení celá ľavá strana bude pre danú vzorku  $k$  vždy konštantný vektor, t. j.

$$\mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k, \quad (5.52)$$

kde  $\mathbf{b}_0$  a  $\mathbf{B}_0$  sú dané podľa Rov. (5.50) a  $\mathbf{b}_c$  musíme prepočítať pre každú vzorku.

Pri kombinácii viacerých typov obmedzení postačí výsledný súbor nerovností napísť vo všeobecnom tvare nerovnosti (5.51). Uvažujme ako príklad bežné kombináciu obmedzení amplitúdy stavov a vstupov, t. j. máme

$$\begin{aligned} \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \vec{\mathbf{u}}_k &\leq \begin{bmatrix} \mathbf{1}\bar{\mathbf{u}} \\ -\mathbf{1}\mathbf{u} \end{bmatrix} \\ &\downarrow \\ \begin{bmatrix} \mathbf{N} \\ -\mathbf{N} \end{bmatrix} \vec{\mathbf{u}}_k &\leq \begin{bmatrix} \mathbf{1}\bar{\mathbf{x}} \\ -\mathbf{1}\mathbf{x} \end{bmatrix} + \begin{bmatrix} -\mathbf{M} \\ \mathbf{M} \end{bmatrix} \mathbf{x}_k, \end{aligned}$$

z čoho dostaneme

$$\underbrace{\begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \\ \mathbf{N} \\ -\mathbf{N} \end{bmatrix}}_{\mathbf{A}_c} \vec{\mathbf{u}}_k \leq \underbrace{\begin{bmatrix} \mathbf{1}\bar{\mathbf{u}} \\ -\mathbf{1}\mathbf{u} \\ \mathbf{1}\bar{\mathbf{x}} \\ -\mathbf{1}\mathbf{x} \end{bmatrix}}_{\mathbf{b}_c} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -\mathbf{M} \\ \mathbf{M} \end{bmatrix}}_{\mathbf{b}_c} \mathbf{x}_k, \quad (5.53)$$

kde nevyužité bloky môžeme vyplniť maticami nul  $\mathbf{0}$  vhodného rozmeru a kombináciu obmedzení máme vo všeobecnom tvare nerovnosti (5.51).

**Príklad 5.20.** Vstup do jednosmerného motora je napäťový signál o veľkosti 3 až 6 V. Diskrétny stavový model uložený v súbore *motor.mat* (pozri *modelmotor.mat* [108]) vyjadruje uhlové pootočenie v rad, uhlovú rýchlosť v rad/s a prúd v rotore pomocou stavového vektora. Motor je využitý na polohovanie servomechanizmu, a môže sa pohybovať v uhlovom rozmedzí  $\pm 90^\circ$  a rýchlosťou  $\pm 10$  rad/s. Maximálny prúdový odber je  $\pm 1$  A. Vytvorte matice na definovanie týchto procesných obmedzení, ak predikčný horizont je  $n_p = 3$  krokov a aktuálne odhadnutý stav je  $\hat{x}_k = [0.1 \ 0.5 \ 0.5]^T$ .

Definujme si najprv procesné obmedzenia. Podľa zadania máme obmedzenia vstupu  $\underline{u} = 3$  a  $\bar{u} = 6$ . Uhlos  $90^\circ$  predstavuje 1.57 rad, a teda obmedzenia na stavy sú  $\underline{x} = [-1.57 \ -10 \ -1]^T$  a  $\bar{x} = [1.57 \ 10 \ 1]^T$ .

Najprv načítame model zo súboru, deklarujeme dĺžku horizontu a zistíme počet stavov. Pre tento prípad zadáme iba jeden odhadnutý stav, no v späťnej väzbe by sme vždy rátali s novým stavom. Pokračujeme deklaráciou obmedzení:

Zdrojový kód 5.27. *motorobmedzeniakombinovane.m*

```
load motor.mat; A=motor.a; B=motor.b; % Model
np=3; % Horizont
[nx,nu]=size(B); % Rozmery
xk=[0.1 0.5 0.5]'; % Aktualny stav
ul=3; uh=6; % Obmedz. vstupu
xl=[-1.57 -10 -1]'; xh=-xl; % Obmedz. stavov
```

Na výpočet matíc, ktoré definujú množinu nerovností pre obmedzenia, používame naše nové funkcie:

```
[Acu b0u] = obmedzeniavstupov(ul,uh,np) % Obm. u
[Acx b0x B0x] = obmedzeniastavov(xl,xh,np,A,B) % Obm. x
```

Aj keď stav je premenlivý, matice  $A_c$ ,  $b_0$  a  $B_0$  sú pre nominálny systém konštantné. Výsledné výrazy môžeme vytvoriť tak, že uložíme bloky obmedzení na vstupy a stavy „nad seba“ pre  $A_c$  a  $b_0$ , kým v prípade  $B_0$  nevyužité časti vyplníme nulami:

```
Ac=[Acu;Acx] % Kombin. Ac
b0=[b0u; b0x] % Kombin. b0
B0=[zeros(2*np*nu,nx); B0x] % Kombin. B0
```

V skutočnosti  $b_c$  vždy musíme prepočítať na základe aktuálneho stavu:

```
bc=b0+B0*xk % Vzdy podla xk
```

V príkazovom riadku vidíme nerovnosti v maticovom tvare aj výslednú kombináciu obmedzení. Pre nominálny model máme vždy  $\mathbf{A}_c$  pevne dané, kým  $\mathbf{b}_c$  prepočítame na základe aktuálneho stavu.

□

**Príklad 5.21.** Vytvorte univerzálnu funkciu na tvorbu matíc  $\mathbf{A}_c$ ,  $\mathbf{b}_0$  a  $\mathbf{B}_0$  pre procesné obmedzenia vstupov a stavov. Novú funkciu otestujte pomocou zadania z Pr. 5.20 tak, že uvažujte a) len obmedzenia na vstup, b) len obmedzenia na stavy, a c) kombináciu obmedzení.

Novú funkciu môžeme nazvať `obmedzenia()`. Na výstupe budeme očakávať matice  $\mathbf{A}_c$ ,  $\mathbf{b}_0$  a  $\mathbf{B}_0$ , kým na vstupe môžeme mať rôzny počet vstupov. Iba pre obmedzenia vstupov máme 3 vstupné parametre, iba pre obmedzenia stavov 5 parametrov a maximum je 7 v prípade, že uvažujeme kombináciu obmedzení na amplitúdy vstupov a stavov. Označme si parametre na vstupe  $p1$  až  $p7$ , potom definícia funkcie bude

Zdrojový kód 5.28. `obmedzenia.m`

```
function [Ac b0 B0]=obmedzenia(p1,p2,p3,p4,p5,p6,p7)
```

Výber správnej funkcionality realizujeme tak, že spočítame vstupné premenné cez funkciu `nargin` a potom vyberáme medzi jednotlivými prípadmi pomocou príkazov `switch-case`. V prípade troch premenných máme obmedzenia na vstup, pričom parametre dosadíme na správne miesta, a teda použijeme rovnakú syntax ako v prípade funkcie `obmedzeniavstupov()`:

```
switch nargin
case 3 % Iba obmedzenia na vstup
    [Ac b0] = obmedzeniavstupov(p1,p2,p3); % Obm. u
    B0=[];

```

Ak máme 3 parametre, predpokladáme obmedzenia na stavy a správnou kombináciu poradia vstupných premenných využijeme syntax `obmedzeniastavov()`:

```
case 5 % Iba stavove obmedzenia
    [nx,nu]=size(p5); % Rozmery
    [Ac b0 B0] = obmedzeniastavov(p1,p2,p3,p4,p5);
```

V prípade, že máme 7 parametrov, použijeme kombináciu obmedzení na amplitúdy vstupov a stavov:

```
case 7 % Kombinacia obmedzeni
```

```

[nx,nu]=size(p7);                                % Rozmery
[Acu b0u] = obmedzeniavstupov(p1,p2,p5);
[Acx b0x B0x] = obmedzeniastavov(p3,p4,p5,p6,p7);
Ac=[Acu;Acx]                                     % Kombin. Ac
b0=[b0u; b0x]                                     % Kombin. b0
B0=[zeros(2*p5*nu,nx); B0x]                      % Kombin. B0

```

Všetky ostatné prípady počtu vstupných obmedzení sú nesprávne, čo teda môžeme označiť ako chybu:

```

otherwise
    disp('Chyba na vstupe.');
end

```

Novú funkciu na tvorbu obmedzení môžeme otestovať pomocou zadania z predchádzajúceho príkladu tak, že vyskúšame rôzne kombinácie typu obmedzení. Začiatok zdrojového kódu je identický Pr. 5.20, preto uvádzame iba čiastočný výpis zdrojového kódu:

Zdrojový kód 5.29. testobmedzenia.m (Čiastočný výpis)

```

[Ac b0] = obmedzenia(ul,uh,np)                % Iba u
[Ac b0 B0] = obmedzenia(xl,xh,np,A,B)          % Iba x
[Ac b0 B0] = obmedzenia(ul,uh,xl,xh,np,A,B)    % Komb.

```

Po spustení testovacieho programu uvidíme, že pri každom variante sme dostali správne definičné matice procesných obmedzení.

□

### 5.3.5 Iné typy obmedzení

Okrem obmedzení na amplitúdu vstupov a stavov existuje ešte niekoľko ďalších typov procesných obmedzení. Obmedzenia na výstupy môžeme definovať ako

$$\underline{\mathbf{y}} \leq \mathbf{y}_{k+i} \leq \bar{\mathbf{y}}, \quad (5.54)$$

pre  $i=1,2,\dots,n_p$ , kde  $\underline{\mathbf{y}}$  je minimálne prípustný výstup a  $\bar{\mathbf{y}}$  je maximálny prípustný výstup. Kedže  $\mathbf{y}_{k+i} = \mathbf{Cx}_{k+i}$  potom platí aj

$$\underline{\mathbf{y}} \leq \mathbf{Cx}_{k+i} \leq \bar{\mathbf{y}}, \quad (5.55)$$

a na základe predikčnej rovnice aj

$$\mathbf{CM}_i \mathbf{x}_k + \mathbf{CN}_i \vec{\mathbf{u}}_k \leq \bar{\mathbf{y}} \quad (5.56)$$

$$-\mathbf{CM}_i \mathbf{x}_k - \mathbf{CN}_i \vec{\mathbf{u}}_k \leq -\underline{\mathbf{y}}. \quad (5.57)$$

pre  $i=1,2,\dots,n_p$ .

Niekedy chceme obmedziť namiesto absolútnej hodnoty vstupov zmenu vstupov vzľadom na predchádzajúcu vzorku

$$\Delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}, \quad (5.58)$$

kde potom príslušné obmedzenie

$$\underline{\Delta \mathbf{u}} \leq \Delta \mathbf{u}_{k+i} \leq \overline{\Delta \mathbf{u}} \quad (5.59)$$

platí pre  $i = 0, 1, \dots, n_p - 1$  a  $\underline{\Delta \mathbf{u}}$  je minimálna a  $\overline{\Delta \mathbf{u}}$  maximálna dovolená zmena vstupov. Niektoré formulácie prediktívneho riadenia predpokladajú  $\vec{\Delta \mathbf{u}}_k$  za optimalizačnú premennú a v tom prípade používame rovnaký postup ako v prípade procesných obmedzení v absolútном tvare. Môžeme definovať aj zmenu stavov  $\Delta \mathbf{x}_k$ , potom by sme mohli obmedzenia napísať v tvare  $\underline{\Delta \mathbf{x}} \leq \Delta \mathbf{x}_{k+i} \leq \overline{\Delta \mathbf{x}}$   $i=1,2,\dots,n_p$ , kde  $\underline{\Delta \mathbf{x}}$  je minimálna povolená zmena stavov a  $\overline{\Delta \mathbf{x}}$  je maximálna.

Iné obmedzenia ako obmedzenia vstupov, resp. stavov v tejto učebnici už nebudeme uvažovať. Dôležité je zapamätať si, že ak vieme obmedzenia vyjadriť v univerzálnom kompaktnom tvare podľa nerovnosti (5.51), potom môžu byť súčasťou formulácie prediktívneho riadenia.

## 5.4 Problém prediktívneho riadenia s obmedzeniami

Problém prediktívneho riadenia bez obmedzení (5.3) sme si definovali ako minimalizáciu kvadratickej účelovej funkcie, ku ktorej sme si doplnili podmienky aj na znalosť stavu (meranie, odhad), na zdroj predikcií (model) a na riadenie v druhom režime (LQ riadiaci zákon) zloženého riadiaceho zákona.

Pri prediktívnom riadení s obmedzeniami musíme rozšíriť túto definíciu o procesné obmedzenia. Ak všeobecne predpokladáme prítomnosť vstupných aj stavových obmedzení, problém prediktívneho riadenia môžeme definovať matematicky ako hľadanie optimálnej postupnosti akčných zásahov podľa

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} \left[ \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \right], \quad (5.60)$$

vzhľadom na podmienky:

$$\begin{aligned} \underline{\mathbf{u}} \leq \mathbf{u}_{k+i} \leq \bar{\mathbf{u}} & \quad \text{pre } i = 0, 1, \dots, n_p - 1, \\ \underline{\mathbf{x}} \leq \mathbf{x}_{k+i} \leq \bar{\mathbf{x}} & \quad \text{pre } i = 1, 2, \dots, n_p, \\ \mathbf{x}_k = \hat{\mathbf{x}}_k, & \\ \mathbf{x}_{k+i+1} = \mathbf{A} \mathbf{x}_{k+i} + \mathbf{B} \mathbf{u}_{k+i} & \quad \text{pre } i = 0, 1, 2, \dots, n_p - 1, \\ \mathbf{u}_{k+i} = \mathbf{K} \mathbf{x}_{k+i}, & \quad \text{pre } i = n_p, n_p + 1, \dots, \infty. \end{aligned}$$



Túto štandardnú formuláciu MPC s obmedzeniami môžeme často vidieť v knihách; rozdiely môžeme nájsť – okrem odlišného značenia – najmä pri definícii obmedzení. Napríklad, ak rátame iba s obmedzeniami na vstup, stačí vynechať druhú podmienku  $\underline{\mathbf{x}} \leq \bar{\mathbf{x}}_k \leq \underline{\mathbf{x}}$ .

Už vieme, že podobná „školská“ definícia MPC nie je príliš praktická pre numerickú implementáciu a ďalšiu prácu. Práve preto sme si odvodili kompaktnú formu účelovej funkcie v tvare (5.6), ktorá už v sebe zahŕňa predikčné rovnice na základe stavového modelu a druhý režim riadenia.

Procesné obmedzenia môžeme tiež vyjadriť v kompaktnom zápisu podľa (5.51), kde stačí vedieť, že ak optimalizačnú premennú  $\vec{\mathbf{u}}_k$  násobíme nejakou maticou  $\mathbf{A}_c$ , výsledok musí byť menší ako vektor  $\mathbf{b}_c$ . Praktický problém prediktívneho riadenia s obmedzeniami je potom kombináciou Rov. (5.5) a nerovností (5.51), t. j.

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} \left( \vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + 2 \mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k \right), \quad (5.61)$$

vzhľadom na podmienku:

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_c.$$



V tejto kapitole sme hľadali takú optimálnu sekvenčiu vstupov  $\vec{\mathbf{u}}_k^*$ , aby sme našli bod, v ktorom sa gradient účelovej funkcie (5.6) rovná nule. Uvažujme teraz ďalšiu podmienku: splnené musia byť tiež obmedzenia v tvare nerovnosti. Ako teda môžeme vyriešiť úlohu prediktívneho riadenia s obmedzeniami?

### 5.4.1 Riešenie problému MPC s obmedzeniami

Z matematického hľadiska uvažujeme kvadratickú účelovú funkciu s obmedzeniami v tvare nerovnosti.



Minimalizáciu kvadratickej účelovej funkcie s obmedzeniami – teda v našom prípade MPC s obmedzeniami – nevieme vyriešiť v uzavretom tvaru<sup>a</sup>.

<sup>a</sup>Pojmom uzavretý tvar máme na mysli riešenie bez potreby nástrojov numerickej optimalizácie. Nezamieňajme si ho preto s explicitným riešením problému MPC, ktoré možno získať pomocou techniky parametrického programovania; pozri Kap. 1.6.3.

Je to možno prekvapujúce, ale znamená to, že pre matematický problém (5.61) univerzálne riešenie v tvare „vzťahu“ neexistuje, t. j. nemôžeme ho manuálne odvodíť. To ale neznamená, že hľadanie  $\vec{u}_k$  je beznádejné, pretože môžeme použiť numerické riešenie problému (5.61) pomocou techniky tzv. *kvadratického programovania*.

Skúsme teraz oddeliť naše myslenie od sveta automatického riadenia a predpokladajme, že  $\mathbf{u}$  je všeobecná premenná. Kvadratické programovanie potom rieši nasledujúci matematický problém nájdenia  $\mathbf{u}^*$  podľa

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} \left( \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} \right),$$

vzhľadom na podmienku:

$$\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c,$$

ktorému prislúcha minimum všeobecnej kvadratickej funkcie pri dodržaní obmedzení v tvare nerovnosti. Všeobecná úloha kvadratického programovania známa z matematiky už na prvý pohľad vyzerá podobne ako naša úloha MPC v riadení. V nasledujúcej kapitole sa pozrieme bližšie na kvadratické programovanie, a teda aj na riešenie problému prediktívneho riadenia s obmedzeniami.

## Kapitola 6

# Kvadratické programovanie

V predchádzajúcej kapitole sme si kompaktne opísali niekoľko častých typov procesných obmedzení a formulovali matematickú definíciu problému prediktívneho riadenia v prítomnosti obmedzení v tvare

$$\vec{u}_k^* = \arg \min_{\vec{u}_k} \left( \vec{u}_k^T \mathbf{H} \vec{u}_k + 2 \mathbf{x}_k^T \mathbf{G}^T \vec{u}_k \right),$$

vzhľadom na podmienku:

$$\mathbf{A}_c \vec{u}_k \leq \mathbf{b}_c.$$

V tejto kapitole sa bližšie pozrieme na uvedený matematický problém a ukážeme si niekoľko spôsobov jeho riešenia. Pokiaľ nás zaujíma online riešenie MPC problému (pozri Kap. 1.6.3), hľadanie minima kvadratickej účelovej funkcie v prítomnosti obmedzení nemá uzavretý tvar. To znamená, že nemôžeme odvodiť univerzálne riešenie v tvare funkčného vzťahu<sup>1</sup>. Namiesto toho musíme siahnuť po nástrojoch numerickej matematiky a postupne, krok po kroku – inými slovami *iteratívne* – sa približovať k riešeniu. Riešenie existuje vždy, ale jeho zložitosť silne závisí od charakteru problému [115]. Je zrejmé, že tento proces nie je až taký jednoduchý. Preto v tejto kapitole zanecháme za sebou svet automatického riadenia a oprášime si vedomosti z matematiky. Hlavný cieľ kapitoly je teda nájsť spôsob vyhľadania minima kvadratickej funkcie v prítomnosti obmedzení.

V rámci tejto kapitoly budeme používať všeobecné premenné, pričom ponecháme značenie pripomínajúce naše pôvodné poslanie: automatické riadenie. Namiesto vektora budúcich optimálnych vstupov  $\vec{u}_k$  budeme ďalej uvažovať všeobecný vektor  $\mathbf{u}$  a namiesto vektora optimálnych vstupov  $\vec{u}_k^*$  teda hľadáme optimálnu hodnotu argumentu účelovej funkcie,  $\mathbf{u}^*$ . Nebudeme minimalizovať kvad-

---

<sup>1</sup>Toto samozrejme neplatí v prípade explicitného MPC, kde pomocou techniky parametrického programovania môžeme získať explicitné riešenie optimalizačného problému MPC; vid Kap. 1.6.3.

ratickú kriteriálnu funkciu  $J(\vec{\mathbf{u}}_k, \mathbf{x}_k)$ , namiesto toho uvažujme všeobecnú kvadratickú funkciu  $f(\mathbf{u}_k)$ . Kým Hessián  $\mathbf{H}$  a vektor  $\mathbf{g}$  si ponechajú svoje matematické vlastnosti, ako napríklad symetrickosť a kladnú definitnosť  $\mathbf{H}$ , budú mať všeobecný charakter, čo samozrejme platí aj pre obmedzenia  $\mathbf{A}_c$  a  $\mathbf{b}_c$ . Na záver si zavedieme ďalší symbol,  $\mathbf{u}^\circ$ , ktorý bude vyjadrovať riešenie kvadratickej funkcie bez obmedzení.

## 6.1 Problém kvadratického programovania

Kvadratická funkcia s množinou lineárnych obmedzení, resp. ohraničení tvorí problém *kvadratického programovania* (QP)<sup>2</sup>. Obmedzenia môžu byť definované v tvare rovnosti, aj v tvare nerovností. Všeobecný problém kvadratického programovania má vo vede, technike a ekonomike také množstvo aplikácií, že mnohí ho považujú za samostatnú disciplínu aplikovanej matematiky [69].

Problém kvadratického programovania teda zahŕňa kvadratickú funkciu a množinu obmedzení v tvare nerovností. Všeobecne môžeme zahrnúť do formulácie aj obmedzenia v tvare lineárnej rovnosti

$$\mathbf{A}_e \mathbf{u} = \mathbf{b}_e, \quad (6.1)$$

kde, podobne ako v prípade nerovností<sup>3</sup>, riadky matice  $\mathbf{A}_e$  a vektora  $\mathbf{b}_e$  vyjadrujú  $n_e$  rovnosti, ktoré všetky musia platiť v riešení. V prediktívnom riadení existujú formulácie, pri ktorých si môžeme predstaviť aj potrebu obmedzení v tvare rovností, no väčšinou sú nepraktické lebo obmedzujú priestor možných riešení príliš príse.



Problém kvadratického programovania definujeme ako riešenie problému

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} \left( \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} \right), \quad (6.2)$$

vzhľadom na podmienku:

$$\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c,$$

$$\mathbf{A}_e \mathbf{u} = \mathbf{b}_e,$$

kde  $\mathbf{u} \in \mathbb{R}^{n \times 1}$ , teda je  $n$ -prvkový vektor reálnych čísel. Ďalej, pre Hessián platí, že  $\mathbf{H} \in \mathbb{R}^{n \times n}$  a  $\mathbf{H} = \mathbf{H}^T$ , t.j. ide o štvorcovú symetrickú maticu reálnych čísel.

<sup>2</sup>angl. quadratic programming (QP)

<sup>3</sup> $\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c$  budeme nazývať obmedzenia v tvare nerovnosti, napriek tomu, že sú nestriktné, t.j. pripúšťajú aj rovnost, a tým ich pojmovu odlišíme od obmedzení v tvare rovnosti  $\mathbf{A}_e \mathbf{u} = \mathbf{b}_e$ .

Pre konvexné kvadratické programovanie je potrebná aj podmienka kladnej definitnosti, t.j.  $\mathbf{H} \succ 0$ , v prípade záporne definitej matice by išlo o nekonvexný problém s lokálnymi riešeniami [115], a pri  $\mathbf{H} = 0$  by sme nemohli hovoriť ani o kvadratickom programovaní. Ďalej platí  $\mathbf{g} \in \mathbb{R}^{n \times 1}$ , uvažujeme  $n_o$  obmedzení v tvare nerovnosti, pričom  $\mathbf{A}_c \in \mathbb{R}^{n_o \times n}$  a  $\mathbf{b}_c \in \mathbb{R}^{n_o \times 1}$ , čo znamená, že pre každý riadok maticovej nerovnosti máme jedno obmedzenie pre  $\mathbf{u}$ . Podobne, máme  $n_e$  obmedzení v tvare rovností, a teda  $\mathbf{A}_e \in \mathbb{R}^{n_e \times n}$  a  $\mathbf{b}_e \in \mathbb{R}^{n_e \times 1}$ . Odkiaľ sme však zobrať faktor  $1/2$  na začiatku účelovej funkcie? Je to predpoklad, ktorý značne zjednoduší neskoršie matematické operácie, pretože po prvej a druhej derivácii funkcie z nej isté členy jednoducho vypadnú [25].

**Príklad 6.1.** Vyjadrite nasledujúcu kvadratickú funkciu

$$f(\mathbf{u}) = f(u_1, u_2) = u_1^2 + 9u_2^2 - 10u_1 - 54u_2,$$

vzhľadom na obmedzenia:

$$u_1 \geq 0,$$

$$u_2 \geq 0,$$

$$u_2 \leq 4 - u_1,$$

v tvare všeobecného problému kvadratického programovania podľa Rov. (6.2).

Funkciu  $f(\mathbf{u})$  môžeme po jednoduchej algebraickej úprave napísť v tvare

$$f(\mathbf{u}) = \frac{1}{2} \underbrace{[u_1 \quad u_2]}_{\mathbf{u}^T} \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & 18 \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{\mathbf{u}} + \underbrace{\begin{bmatrix} -10 & -54 \end{bmatrix}}_{\mathbf{g}^T} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{\mathbf{u}},$$

ktorý je totožný so štandardnou formuláciou z Rov. (6.2), kde

$$\mathbf{H} = \begin{bmatrix} 2 & 0 \\ 0 & 18 \end{bmatrix},$$

$$\mathbf{g} = \begin{bmatrix} -10 \\ -54 \end{bmatrix}.$$

Pre obmedzenia v tvare nerovností platí

$$\begin{aligned} -u_1 &\leq 0, \\ -u_2 &\leq 0, \\ u_1 + u_2 &\leq 4 \end{aligned}$$

a môžeme ich napísť v kompaktnom maticovom tvare ako

$$\underbrace{\begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix}}_{\mathbf{A}_c} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{\mathbf{u}} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix}}_{\mathbf{b}_c},$$

čím dostaneme aj obmedzenia vo všeobecnom tvare z Rov. (6.2), kde

$$\mathbf{A}_c = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix},$$

$$\mathbf{b}_c = \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix}.$$

□

Na riešenie problému kvadratického programovania existujú rôzne numerické metódy. Patria medzi ne metódy založené na myšlienke

- aktívnych množín<sup>4</sup> [48, 115],
- vnútorného bodu<sup>5</sup> [113],
- združeného gradientu<sup>6</sup>,
- priemetu gradientu<sup>7</sup>,
- varianty tzv. simplexového algoritmu<sup>8</sup> atď. . .

Všetky spomenuté metódy majú numerický charakter a k riešeniu sa dopracujú iteračne, t. j. postupne. Rôzne metódy pritom nie sú úplne rovnocenné: niektoré sú vhodné na problémy s veľkým počtom optimalizačných premenných, ale s malým počtom obmedzení, resp. naopak; niektoré si dobre poradia s maticami, ktoré obsahujú veľa nulových prvkov atď. Kým každá dobre navrhnutá metóda sa dopracuje k riešeniu, je vždy otázne ako efektívne, resp. ako rýchlo vieme nájsť riešenie daného problému.

<sup>4</sup>angl. active set

<sup>5</sup>angl. interior point

<sup>6</sup>angl. conjugate gradient

<sup>7</sup>angl. gradient projection

<sup>8</sup>angl. simplex algorithm

Pre konkrétny prípad prediktívneho riadenia môžeme tvrdiť, že otázka všeobecne „najlepšieho“ riešiťa je stále otvorená [101, 143]. Problém prediktívneho riadenia má veľa špecifických vlastností, ktoré sa dajú využiť na zvýšenie efektivity hľadania optima. Medzi nimi je aj to, že napríklad formulácia MPC obsahuje veľa nulových prvkov, čo vytvorí takzvané *riedke maticy*<sup>9</sup>. Pri riedkych maticiach si môžeme ušetriť veľa výpočtových operácií. Ďalej, pri prediktívnom riadení môžeme využiť aj to, že s výnimkou veľkých porúch alebo zmien žiadanej hodnoty riešenia sa nasledujúce akčné zásahy od seba veľmi neodlišujú. Na základe toho môžeme predchádzajúcim riešením problému QP „naštartovať“ a urýchliť tak riešenie v ďalšej vzorke [101].

Ako to už bolo spomenuté v úvode kapitoly, kvadratické programovanie môžeme považovať za dôležitú súčasť numerickej matematiky. Našim cieľom preto nebude predstaviť si každý možný spôsob riešenia, ale budeme sa sústrediť na prvé dva: na metódu aktívnej množiny a metódu vnútorného bodu. Predtým ako začneme analyzovať metódy riešenia, pozrime sa na geometrickú interpretáciu nášho problému.

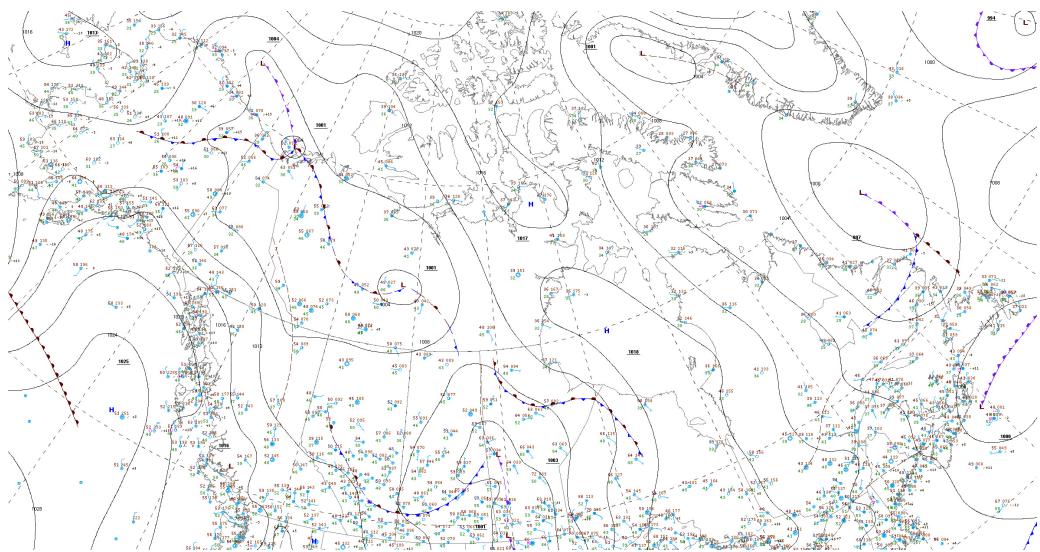
## 6.2 Trochu geometrie: vrstevnice, gradient a ohraničenia v priestore

Pri analýze možností riešenia problému kvadratického programovania sa budeme často odkazovať aj na geometrickú interpretáciu samotného matematického problému. Pre ilustráciu niektorých nových pojmov si zvolíme jednoduchú funkciu s dvoma nezávislými premennými. Tako zobrazíme priestor optimalizačnej premennej v 2D, kde funkčná hodnota predstavuje ďalší, tretí rozmer. Samozrejme, toto ponímanie platí aj pre viac premenných, no geometrický výklad by bol obtiažný. V nasledujúcich podkapitolách si uvedieme prehľad zobrazenia vrtsevníc, gradientu a lineárnych obmedzení pre skalárnu kvadratickú funkciu s dvoma premennými.

### 6.2.1 Vrstevnice funkcie

Pre zobrazenie hodnôt kvadratickej funkcie s dvoma premennými potrebujeme nakresliť paraboloid v 3D, a tým občas získame nešikovnú grafickú reprezentáciu. Pre zobrazenie hodnôt funkcie s dvoma premennými preto často používame iba *vrstevnice* funkcie. Vrstevnice predstavujú priemet rovnakých funkčných hodnôt do priestoru nezávislých premenných. Vrstevnica predstavuje body, v ktorých daná kombinácia nezávislých premenných na výstupe funkcie dáva tú istú hodnotu. S vrstevnicami sa často stretávame pri kartografických mapách, kde rovnakú

<sup>9</sup>angl. sparse matrix



Obr. 6.1. Rovnaké úrovne atmosferického tlaku vyjadrené pomocou vrstevníc [140].

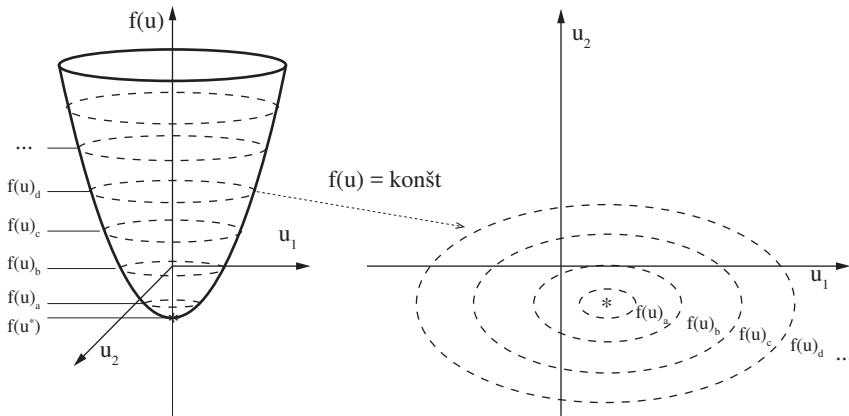
nadmorskú výšku na dvojrozmernej mape znázorňujeme pomocou vrstevníc. S vrstevnicami sa môžeme stretnúť aj pri predpovedi počasia, kde priestorové rozloženie atmosferického tlaku meteorológovia zobrazujú pomocou vrstevníc rovnakého tlaku (Obr. 6.1). Vrstevnice<sup>10</sup> funkcie sa nazývajú tiež izočiary alebo ekvipotenciálne krivky, t. j. krivky s rovnakým potenciálom.

Na Obr. 6.2 vidíme ilustráciu konceptu izočiar. Vľavo je parabola, ktorá predstavuje kvadratickú funkciu  $f(u_1, u_2)$  v trojrozmernom priestore. Rovnakú hodnotu funkcie  $f(\mathbf{u})_a$  môžeme dosiahnuť rôznymi kombináciami vstupných premenných  $u_1, u_2$ . Pri danej skalárnej hodnote funkcie dostaneme 2D priemet trojrozmernej paraboly v tvare elipsy. takéto priemety môžeme nájsť pre ľubovoľné funkčné hodnoty; môžeme si zvolať napríklad rovnomerné rozloženie funkčných hodnôt. Priemety danej funkcie do dvojrozmerného priestoru optimalizačných premenných predstavujú jej vrstevnice.

Najjednoduchším spôsobom zobrazenia vrstevníc v MATLABe je definícia symbolickej funkcie a potom využitie príkazu

**fcontour (f)**

<sup>10</sup>angl. contour lines



Obr. 6.2. Kvadratická funkcia dvoch nezávislých premenných a jej vrstevnice.

ktorý vykreslí vrstevnice funkcie. Ak chceme špecifikovať rozsah vstupných premenných, zavoláme tento príkaz v tvare:

**fcontour(f, [minrozsa maxrozsa])**

V nasledujúcom príklade si porovnáme kvadratickú funkciu s dvoma premennými a jej vrstevnice.

**Príklad 6.2.** Zobrazte kvadratickú funkciu z Pr. 6.1 v 3D a porovnajte jej plochu v priestore s vrstevnicami funkcie. Príklad riešte ručne a v MATLABe overte neohraničené minimum tejto funkcie a zobrazte výsledok graficky.

Začneme hľadaním bodu, v ktorom gradient funkcie nadobúda nulovú hodnotu, t. j.  $\nabla f(\mathbf{u}) = 0$ , a vypočítame parciálne derivácie funkcie podľa každej jej nezávislej premennej:

$$\begin{aligned}\frac{\partial f(u_1, u_2)}{\partial u_1} &= 2u_1 - 10 = 0, \\ \frac{\partial f(u_1, u_2)}{\partial u_2} &= 18u_2 - 54 = 0\end{aligned}$$

z čoho po označení minima funkcie bez obmedzení ako  $\mathbf{u}^\circ$  získame  $u_1^\circ = 5$  a  $u_2^\circ = 3$ .

V zdrojovom kóde začneme deklaráciou matice  $\mathbf{H}$  a  $\mathbf{g}$ , a vytvoríme symbolický vektor  $\mathbf{u}$ . Kvadratickú funkciu môžeme napísať priamo ako:

Zdrojový kód 6.1. zobrazkvad.m

$\mathbf{H} = [2 \quad 0; \quad 0 \quad 18];$

*% Matica H*

```

g=[-10 -54]';
% Vektor g
u = sym('u', [2 1]);
% Nezávislá premenna
f(u) = 1/2*u'*H*u+g'*u;
% Kvadrat. funkcia

```

Ďalej vypočítame gradient funkcie a vyriešime rovnicu  $\nabla f(\mathbf{u}) = 0$ :

```

nablaf=gradient(f,u);
% Gradient
uast=solve(nablaf==0);
% Vyries pre u

```

Na jednoduché vykreslenie funkčných hodnôt môžeme používať príkaz `fsurf()`. Druhý vstupný argument príkazu je limit zobrazenia grafického výstupu:

```

hranice=[-20 20];
% Zobrazenie
fsurf(f,hranice)
% Kvadraticka 3D

```

Zapneme si mriežku, podržíme graf, označíme osi, a pomocou príkazu `colorbar` zapneme farebnú legendu funkčných hodnôt:

```

grid on; hold on;
xlabel('u_1'); ylabel('u_2')
zlabel('f(u)');
colorbar
% Mriezka, podrzanie
% Oznacenie osi
% Oznacenie osi
% Znacenie

```

Na záver si zobrazíme neobmedzené minimum funkcie a jej vrstevnice:

```

plot(uast.u1,uast.u2,'wo')
% Opt. bez obmedz.
fcontour(f,hranice);
% Vrstevnice

```

Po spustení zdrojového kódu dostaneme trojrozmernú ilustráciu predmetnej kvadratickej funkcie, jej vrstevníc a neobmedzeného minima (Obr. 6.3). Plochu si tiež môžeme ľuboľne otáčať v priestore a tým preskúmať súvislosť funkčných hodnôt a vrstevnice funkcie.

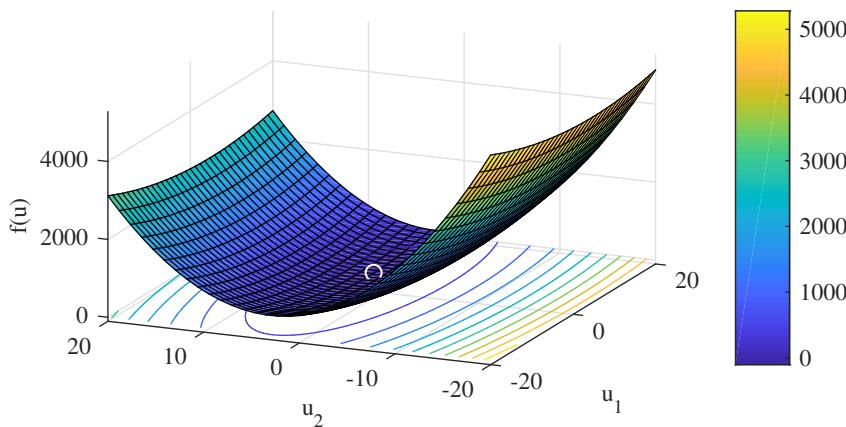
□

Všimnime si tiež niekoľko zaujímavostí na Obr. 6.3. V prípade, že si funkčné hodnoty nakreslíme v rovnomerne rozložených úrovniach, môžeme na základe vrstevníc pozorovať niektoré vlastnosti funkcie. Čím je stúpanie funkcie strmšie, tým sú jej vrstevnice hustejšie. Naopak, čím sú vrstevnice redšie, tým je stúpanie pôvodnej funkcie menšie. Blízko minima máme samozrejme najmenšie stúpanie, a preto je tam naša kvadratická funkcia „plochá“. Dokonca pri minime máme nulové stúpanie, čo presne vyjadruje podmienka  $\nabla f(\mathbf{u}) = 0$ . Neobmedzené minimum je potom v strede sústredných elipsoidov. Hustotu rozdelenia úrovní funkčných hodnôt si v MATLABe môžeme nastaviť pomocou argumentov funkcie `fcontour()`:

```

fcontour(f,[minrozsah maxrozsah],'LevelStep',level)

```



Obr. 6.3. Plocha kvadratickej funkcie v 3D a jej priemet do 2D – vrstevnice. Symbol „○“ označuje minimum funkcie bez uvažovania obmedzení.

Úlohou vrstevníc je zjednodušenie zobrazenia funkčných hodnôt v priestore nezávislých premenných. Práve preto už v ďalšej diskusii nebudeme zobrazať 3D plochy, ale iba vrstevnice funkcií. Vytvorme si preto teraz univerzálny príkaz na zobrazenie vrstevnice kvadratickej funkcie s dvoma premennými.

**Príklad 6.3.** Vytvorte univerzálnu funkciu na zobrazenie vrstevníc kvadratickej funkcie s dvomi premennými na základe matíc  $H$  a  $g$ . Znázornite aj minimum funkcie bez obmedzení. Na testovanie príkazu použite postup z Pr. 6.1.

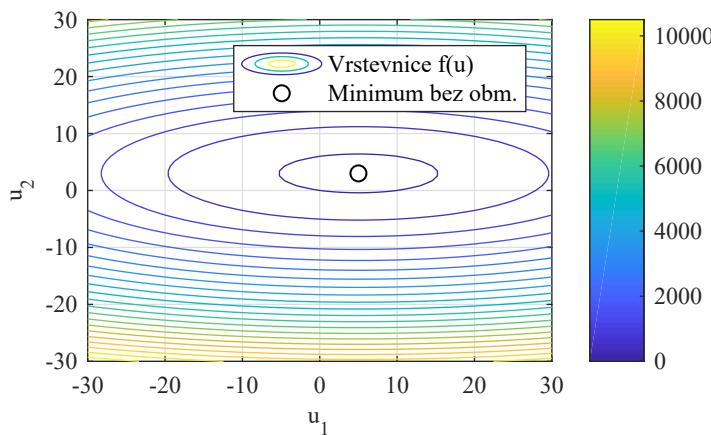
Nová funkcia bude slúžiť výlučne na grafické znázornenie kvadratickej funkcie, preto používame vstupné argumenty  $H$  a  $g$  a neuvažujeme výstupné argumenty. Posledný vstupný argument bude špecifikovať hustotu vykreslenia vrstevníc:

Zdrojový kód 6.2. vykreslitvrstevnice.m

```
function vykreslitvrstevnice(H, g, hranice, level)
```

Ďalej postupujeme v zmysle Pr. 6.2 a začíname deklaráciou symbolickej premennej, definíciou funkcie, výpočtom gradientu a hľadaním neobmedzeného minima:

```
u = sym('u', [2 1]); % Nezávisla prem.
f(u) = 1/2*u'*H*u+g'*u; % Kvadrat. funk.
nabla f=gradient(f,u); % Gradient f(u)
uast=solve(nabla f==0); % Vyries pre u
```



Obr. 6.4. Vrstevnice kvadratickej funkcie dvoch premenných. Symbol „o“ označuje minimum funkcie bez obmedzení.

Pri zobrazení vrstevníc si teraz špecifikujeme aj ich hustotu, a to pomocou páru vstupných argumentov `LevelStep` a `level`. Táto možnosť určí lineárne rozdenenie krokovania funkčných hodnôt. Na záver si označíme osi, zobrazíme legendu a neobmedzené minimum:

```

fcontour(f,hranice,'LevelStep',level); % Vrstevnice f(u)
colorbar % Legenda kont.
grid on; % Mriezka
hold on; % Podrzanie grafu
 xlabel('u_1'); ylabel('u_2') % Oznacenie osi
 plot(uast.u1,uast.u2,'ko') % Bez obmedz.

```

Na testovanie využívame Pr. 6.1, kde po deklarácii matíc **H** a **g** zavoláme novo vytvorený príkaz:

Zdrojový kód 6.3. `testvykreslitvrstevnice.m`

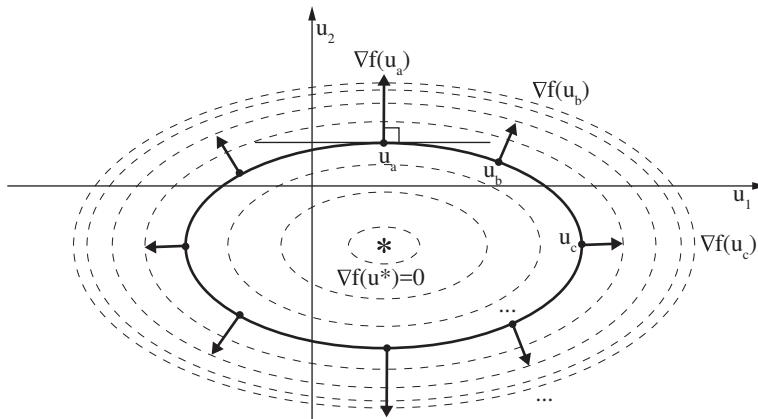
```

vykreslitvrstevnice(H,g,[-30 30],500) % Vrstevnice
legend('Vrstevnice f(u)', 'Minimum bez obm.') % Legenda

```

Po pustení testovacieho programu dostaneme výsledok podľa Obr. 6.4. Všimnime si, že vrstevnice sú pri priblížení sa k minimu hustejšie, pričom v strede sústredných elipsoidov nájdeme aj neobmedzené minimum.

□



Obr. 6.5. Vzťah vrstevnice a gradientu funkcie. V konkrétnom bode  $\mathbf{u}_a$  dostaneme vektor gradientu  $\nabla f(\mathbf{u}_a)$ , ktorý znázorňuje smer najväčšieho rastu funkcie a svojou dĺžkou vyjadruje strmosť funkcie v danom bode. Vektor  $\nabla f(\mathbf{u}_a)$  je kolmý na dotyčnicu vrstevnice v bode  $\mathbf{u}_a$ .

### 6.2.2 Gradienty a vrstevnice

V Kap. 4 sme si predstavili pojem gradientu funkcie a spôsob, akým ho možno využiť na nájdenie minima funkcie bez obmedzení. Vieme, že veľký gradient vyjadruje veľkú zmenu funkčných hodnôt, zatiaľ čo nulový gradient predstavuje extrém konvexnej, neobmedzenej funkcie. Gradient teda súvisí aj s vrstevnicami funkcie.

Z predchádzajúcich príkladov je zrejmé, že čím je rast/pokles funkcie vyšší, tým hustejšie vrstevnice dostaneme pri rovnakom rozostupe funkčných hodnôt. Smer gradientu funkcie v danom bode je vždy kolmý na vrstevnicu a ukazuje smer rastu funkcie (Obr. 6.5). Pretože gradient v danom bode má aj svoj smer, hovoríme o vektore, ktorý potom počítame pre konkrétny bod. Gradient funkcie  $\nabla f(\mathbf{u})$  vlastne vytvorí ďalšiu funkciu (samotný gradient), do ktorej po dosadení súradníc za konkrétny bod získame vektor ukazujúci najstrmejšie stúpanie. Hodnotu gradientu môžeme vypočítať pre ľubovoľný vstupný argument funkcie. Veľkosť – z geometrického pohľadu dĺžka vektora gradientu v danom bode vyjadruje veľkosť zmeny v hodnote funkcie, inými slovami strmosť funkcie v danom bode.

**Príklad 6.4.** Vytvorte príkaz na vykreslenie vrstevnice a súvisiacich gradientov pri danej hodnote kvadratickej funkcie. Výsledný zdrojový kód overte pomocou Pr. 6.1 pri  $f(\mathbf{u}) = 100$ . Ručne vypočítajte gradient funkcie v bode  $\mathbf{u} = [3 \ 2]^T$ .

Začneme definíciou vstupno-výstupných argumentov. Na vstupe budeme potrebovať matice kvadratickej funkcie,  $\mathbf{H}$  a  $\mathbf{g}$  a parametre na opis hraníc kreslenia

vrstevnice. Posledný argument  $f(\mathbf{u})$  (`fu`) vyjadruje funkčnú hodnotu, pri ktorej budeme kresliť čiaru a gradient:

Zdrojový kód 6.4. `vykreslitgradient.m`

```
function vykreslitgradient (H, g, hranice, fu)
```

Pokračujeme definovaním symbolického vektora a kvadratickej funkcie podobne ako v predchádzajúcich príkladoch. Teraz definujeme novú funkciu v bode, kde sa  $f(\mathbf{u})$  rovná nejakej hodnote `fu`. Nová funkcia bude mať tvar elipsy, ktorú môžeme nakresliť pomocou príkazu `fimplicit()` pre zobrazenie implicitných funkcií:

```
u = sym('u', [2 1]); % Premenna u
f(u) = 1/2*u'*H*u+g'*u; % Kvadrat. f.
F=f==fu; % Elipsa
fc=fimplicit(F, hranice, 'k', 'LineWidth', 1); % Vrstevnica
hold on % Podrz graf
```

V ďalšom kroku si nájdeme súradnice elipsy, napríklad pomocou výberu informácie zo samotného grafu. Hustotu zobrazenia gradientov znížime o rát pomocou interpolácie hodnôt príkazom `downsample()` na zmenu vzorkovania. Ak chceme zmeniť hustotu zobrazenia, môžeme zmeniť druhý argument príkazu:

```
h = findobj(gca, 'Type', 'ImplicitFunctionLine');
U1=get(h, 'Xdata'); % Uloz x
U2=get(h, 'Ydata'); % Uloz y
U1=downsample(U1, 10); % Menej vz.
U2=downsample(U2, 10); % Menej vz.
```

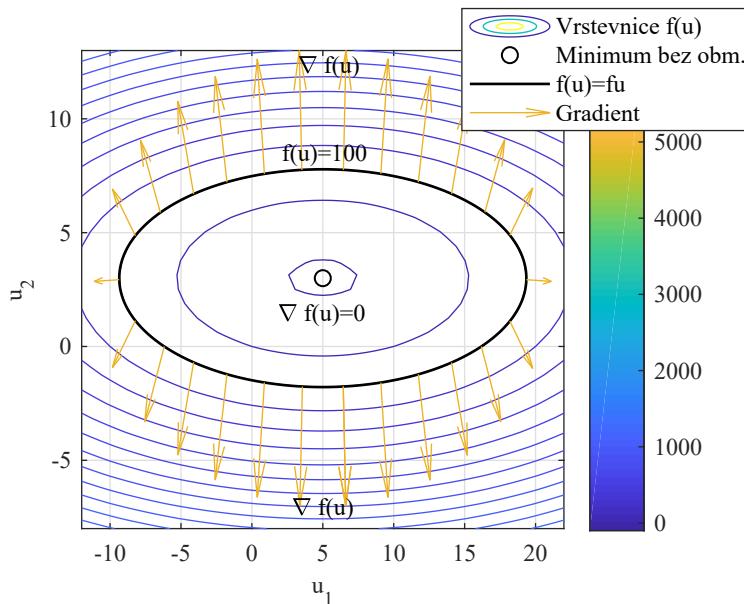
Vypočítame gradient a dosadíme súradnice nezávislej premennej  $\mathbf{u}$  v tvare elipsy. Vektorové pole môžeme zobraziť v MATLABe pomocou príkazu `quiver()`:

```
nf1=gradient(f, u(1)); % Grad. u1
nf2=gradient(f, u(2)); % Grad. u2
NF1 = subs(nf1, [u(1), u(2)], {U1, U2}); % Dosadenie
NF2 = subs(nf2, [u(1), u(2)], {U1, U2}); % Dosadenie
quiver(U1, U2, NF1, NF2); % Zobraz g.
```

Pokračujme ďalej otestovaním vytvoreného príkazu. Zadáme matice z Pr. 6.1, limity vykreslenia a zvolíme konštantnú hodnotu funkcie  $f(\mathbf{u}) = 100$ :

Zdrojový kód 6.5. `testvykreslitgradient.m`

```
H=[2 0; 0 18]; g=[-10 -54]'; % Matica H
hranice=[-22 22]; fu=100; % Hranica a f(u)
```



Obr. 6.6. Vrstevnica a gradienty funkcie v danej funkčnej hodnote.

Vrstevnice a gradienty  $\nabla f(\mathbf{u}) = 100$  si zobrazíme pomocou novovytvorenej funkcie:

```
vykreslit_vrstevnice(H,g,hranice,100)      % Vrstevnice
vykreslit_gradient(H,g,hranice,fu)          % Gradient
legend('Vrstevnice f(u)', 'Minimum bez obm.', 'f(u)=fu', 'Gradient')
axis([-12,22,-8,13])                         % Nastavenie osi
```

Po spustení programu dostaneme grafickú reprezentáciu gradientov funkcie v hodnote  $f(\mathbf{u}) = 100$  (Obr. 6.6). Ako môžeme vidieť, gradienty sú kolmé na vrstevnicu v daných bodoch, pričom ich dĺžka vyjadruje mieru stúpania funkcie  $f(\mathbf{u})$ .

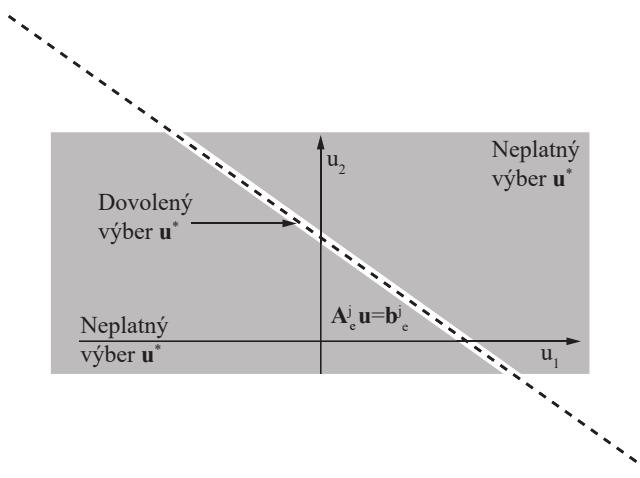
Nasleduje ručný výpočet gradientu. Po parciálnej derivácii kvadratickej funkcie z príkladu Pr. 6.1 dostaneme

$$\nabla f(\mathbf{u}) = \begin{bmatrix} 2u_1 - 10 \\ 18u_2 - 54 \end{bmatrix}$$

a po dosadení za  $\mathbf{u} = [3 \ 2]^T$

$$\nabla f(u_1 = 3, u_2 = 2) = \begin{bmatrix} 2 \\ -18 \end{bmatrix}.$$

□



Obr. 6.7. Obmedzenie v tvare rovnosti v priestore dvoch optimalizačných premenných. Sivá plocha predstavuje ich nedovolené hodnoty,  $\mathbf{u}^*$  musí ležať na danej priamke.

### 6.2.3 Obmedzenia

Na základe problému kvadratického programovania (6.2) už vieme, že v množine obmedzení  $\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c$  každý riadok vyjadruje nerovnosť, ktorú musíme rešpektovať pri minimalizácii kvadratickej funkcie. Označme si jednotlivé riadky tejto množiny indexom  $i$  a získame  $n_o$  nerovností v tvare

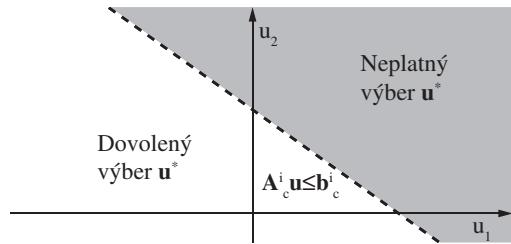
$$\mathbf{A}_c^i \mathbf{u} \leq \mathbf{b}_c^i, \quad \text{pre } i = 1, 2, \dots, n_o. \quad (6.3)$$

Podobne, v prípade obmedzení v tvare rovnosti si označme riadky pomocou indexu  $j$ , a potom dostaneme  $n_e$  rovníc v tvare

$$\mathbf{A}_e^j \mathbf{u} = \mathbf{b}_e^j, \quad \text{pre } i = 1, 2, \dots, n_e. \quad (6.4)$$

Pre optimalizačnú premennú dĺžky  $n$  dostaneme pre každú rovnosť  $j$  rovinu, v ktorej musí daná optimalizačná premenná ležať. Keďže vo všeobecnosti pracujeme v  $n$ -rozmernom priestore, môžeme hovoriť o tzv. *nadrovine*<sup>11</sup>. V prípade troch optimalizačných premenných dostaneme plochu v 3D, kým v prípade dvoch dostaneme priamku (Obr. 6.7). Vidíme, že podmienka v tvare rovnosti je prísna, lebo výsledok optimalizácie musí ležať na priesecníku rovín (resp. priamok).

<sup>11</sup> angl. hyperplane



Obr. 6.8. Obmedzenie v tvare nerovnosti v priestore dvoch optimalizačných premenných. Sivá plocha predstavuje nedovolené voľby  $\mathbf{u}^*$ .

V prípade nerovností každý  $i$ -tý riadok rozdelí priestor platných riešení  $\mathbf{u}^*$  na polpriestory definované príslušnými nadrovinami. Jedna ich strana pritom predstavuje platný výber (polpriestor) optimalizačných premenných, druhá predstavuje neplatný výber. Pre prípad troch optimalizačných premenných tieto polpriestory môžeme nakresliť do 3D (samotnú funkciu už nie) a v prípade dvoch optimalizačných premenných dostaneme polroviny v zmysle Obr. 6.8.

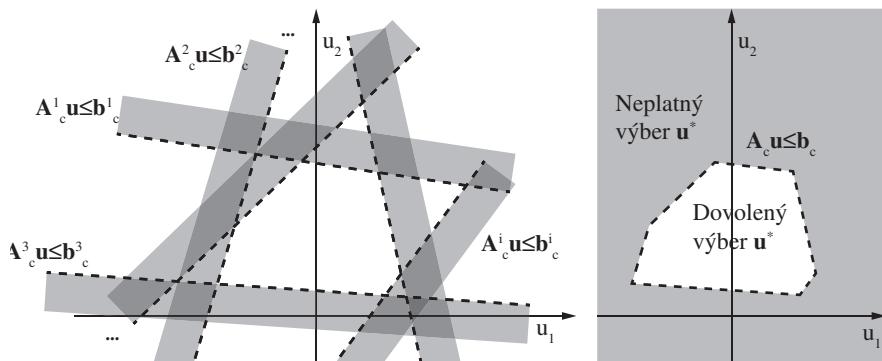
Keďže máme spolu  $i$  obmedzení v tvare nerovnosti, každé z nich vytvorí polpriestory, resp. nadroviny dovolených hodnôt  $\mathbf{u}^*$ . Prienik týchto obmedzení vytvára množinu možných riešení QP. Konečný výsledok optimalizácie môže teda vychádzať iba z tejto množiny, pričom všetky riešenia mimo nej sú neprípustné. Situácia je znázornená pre prípad dvoch optimalizačných premenných na Obr. 6.9. Každá nadrovia vytvorí prípustné a neprípustné riešenia, na základe čoho dostaneme ohraničený mnohosten, takzvaný *polyeder*<sup>12</sup>. Vnútro polyédra a jeho hranice sú teda prípustné, mimo neho sa riešenie nachádzať nemôže.

Určité konfigurácie obmedzení však nemusia byť navzájom kompatibilné. Takoéto obmedzenia jednoducho nevytvárajú priestor, kde by sme vôbec mohli nájsť platné riešenie. Hovoríme, že obmedzenia sú *nezlučiteľné*<sup>13</sup>. Môžeme si predstaviť situáciu, kedy obmedzenia v tvare nerovnosti vytvoria mnohosten možných riešení, no stačí jedno obmedzenie mimo tejto množiny a obmedzenia budú nezlučiteľné (Obr. 6.10(a)). Ďalšia častá situácia je znázornená na Obr. 6.10(b), kde obmedzenia v tvare nerovností sú navzájom nekompatibilné, lebo nevytvárajú žiadnu platnú množinu pre riešenie problému.

**Príklad 6.5.** Naprogramujte univerzálny príkaz na zobrazenie jednotlivých lineárnych obmedzení optimalizačnej úlohy, ak obmedzenia sú dané v tvare  $\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c$  (respektíve  $\mathbf{A}_e \mathbf{u} = \mathbf{b}_e$ ). Program otestujte pomocou zadania z Pr. 6.1.

<sup>12</sup> angl. polyhedron

<sup>13</sup> angl. infeasible



Obr. 6.9. Jednotlivé obmedzenia v tvare nerovnosti vytvárajú množinu zlučiteľných riešení QP. Sivá plocha predstavuje nedovolené voľby  $\mathbf{u}^*$ .

Funkcia bude slúžiť len na ilustračné účely, preto nemá výstupy. Ako vstupné argumenty potrebujeme maticu  $\mathbf{A}_c$ , vektor  $\mathbf{b}_c$  a hranice zobrazenia obmedzení:

Zdrojový kód 6.6. vykreslitobmedzenia.m

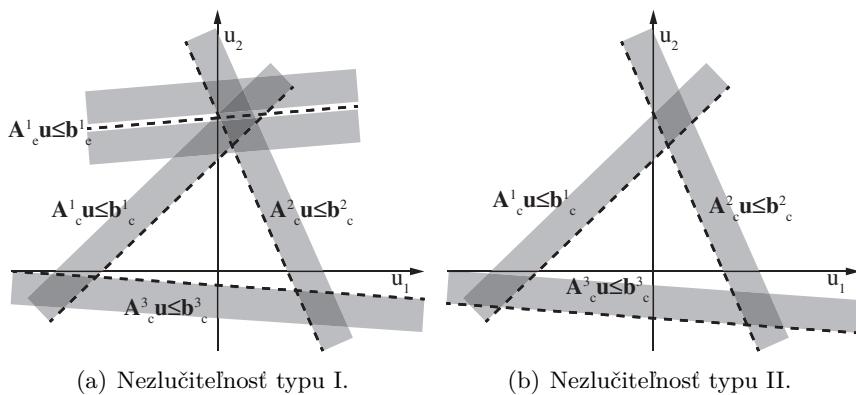
```
function vykreslitobmedzenia(Ac,bc,hranice)
```

Pokračujeme deklaráciou symbolického vektora a rovno zapneme podržanie grafu a mriežky. Musíme prejsť každú nerovnosť z množiny  $\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c$ , a preto pomocou `for` cyklu iterujeme cez dĺžku vektora  $\mathbf{b}_c$ . Ak dané obmedzenie tvorí zvislú (vertikálnu) priamku, dostávame špeciálny prípad, pre ktorý môžeme priamo znázorniť obmedzenie takto:

```
u = sym('u', [2 1]); % Nezávisla premenna
hold on; grid on; % Podržanie grafu
for i=1:length(bc) % Pre vsetky obmedz.
    if Ac(i,2)==0 % Je to zvisla ciara?
        l=line([bc(i)/Ac(i,1) bc(i)/Ac(i,1)],hranice);
        l.Color='k'; l.LineStyle='--'; l.LineWidth=1;
    else
        u2=(bc(i)-Ac(i,1)*u(1))/Ac(i,2); % inak nakresli ciaru
        fplot(u2,'k--',hranice,'LineWidth',1);
    end
end
```

Pre všetky ostatné príklady vyjadrieme závislú premennú  $u_2$ , a zobrazíme funkciu pomocou príkazu `fplot()`:

```
else
    u2=(bc(i)-Ac(i,1)*u(1))/Ac(i,2); % inak nakresli ciaru
    fplot(u2,'k--',hranice,'LineWidth',1);
end
```



Obr. 6.10. Určité kombinácie obmedzení vytvárajú obmedzenia, ktoré sú nezlučiteľné s riešením QP.

Nasleduje tvorba testovacieho programu. Najprv si deklarujeme kvadratickú funkciu a obmedzenia. Vrstevnice účelovej funkcie si vykreslíme pomocou rutiny `vykreslitvrstevnice()`, kým obmedzenia vykreslíme pomocou novovytvorennej rutiny `vykreslitobmedzenia()`:

Zdrojový kód 6.7. `testvykreslitobmedzenia.m`

```

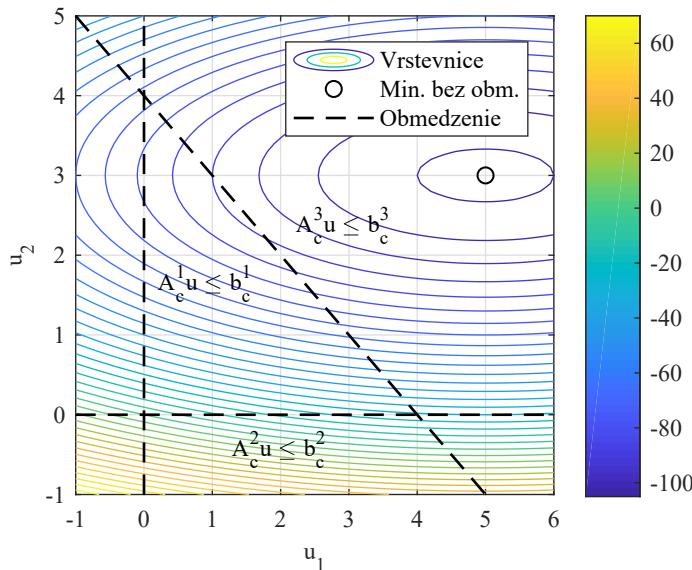
H=[2 0; 0 18]; g=[-10 -54]'; % H, g
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]; % Ac, bc
vykreslitvrstevnice(H,g,[-1 6],5) % Vrstev. funkcie
vykreslitobmedzenia(Ac,bc,[-1 6]) % Obmedzenia
axis([-1 6 -1 5]) % Zobrazenie
legend('Vrstevnice','Min. bez obm.','Obmedzenie')

```

Po spustení testovacieho programu dostaneme výstup podľa Obr. 6.11. Výsledkom predchádzajúcich príkladov dostaneme ilustráciu vrstevníc funkcie, zatiaľ čo nový príkaz postupne vykreslí priamky, ktoré predstavujú hranice obmedzení v tvare nerovností. Začína s  $i = 1$  ( $u_1 \geq 0$ ), čo je vodorovná priamka v nule, pokračuje s  $i = 2$  ( $u_2 \geq 0$ ), čo je zvislá priamka v nule, a na záver diagonálou pre  $i = 3$  ( $u_2 \leq -u_1 + 4$ ). V tomto príklade dostaneme množinu zlučiteľných hodnôt optimalizačných premenných v tvare pravouhlého trojuholníka.

□

Samozrejme náš optimalizačný problém môže obsahovať aj desiatky až tisíce nerovností, ktoré by aj v prípade dvoch optimalizačných premenných mohli tvoriť



Obr. 6.11. Lineárne obmedzenia znázornené v priemete kvadratickej funkcie.

zložité polyedrické útvary. V prípade viacerých optimalizačných premenných stále platia geometrické princípy, no priame grafické zobrazenie už nie je možné.

Ak samotná kvadratická funkcia má gradient, majú aj obmedzenia gradienty? Samozrejme! Označme teraz obmedzenia v tvare nerovnosti funkciou  $\mathbf{g}(\mathbf{u})$ , kde

$$\mathbf{g}(\mathbf{u}) = \mathbf{A}_c \mathbf{u} - \mathbf{b}_c, \quad (6.5)$$

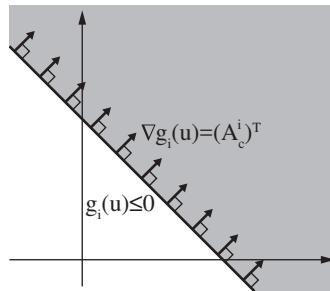
potom samotné obmedzenie budeme mať v tvare

$$\mathbf{g}(\mathbf{u}) \leq 0. \quad (6.6)$$

Ak si označíme  $i$ -té obmedzenie dané  $i$ -tým riadkom  $\mathbf{A}_c$  a  $\mathbf{b}_c$  ako  $g_i(\mathbf{u})$ . potom gradient pre dané obmedzenie je

$$\nabla g_i(\mathbf{u}) = \frac{\partial(\mathbf{A}_c^i \mathbf{u} - \mathbf{b}_c^i)}{\partial \mathbf{u}} = (\mathbf{A}_c^i)^T. \quad (6.7)$$

Inými slovami, gradient bude mať konštantnú veľkosť a jeho smer bude kolmý na nadrovinu predstavujúcu dané obmedzenie, ako to je znázornené na Obr. 6.12. Gradient lineárneho obmedzenia bude teda kolmý na nadrovinu predstavujúcu dané obmedzenie.



Obr. 6.12. Gradient lineárneho obmedzenia v tvare nerovnosti.

**Príklad 6.6.** Vytvorte funkciu na grafické znázornenie gradientov lineárnych obmedzení v 2D. Program otestujte pomocou zadania z Pr. 6.1 pre posledné, tretie obmedzenie.

Funkciu vytvoríme za účelom vykreslenia gradientu konkrétneho obmedzenia, t. j. priamky. Kedže ide o funkciu na grafické zobrazenie, nebude mať výstupné argumenty. Na vstupe však potrebujeme poznáť matice  $\mathbf{A}_c$  a  $\mathbf{b}_c$ , ktoré definujú danú nerovnosť,  $i$  označujúce index nerovnosti a hranice kreslenia:

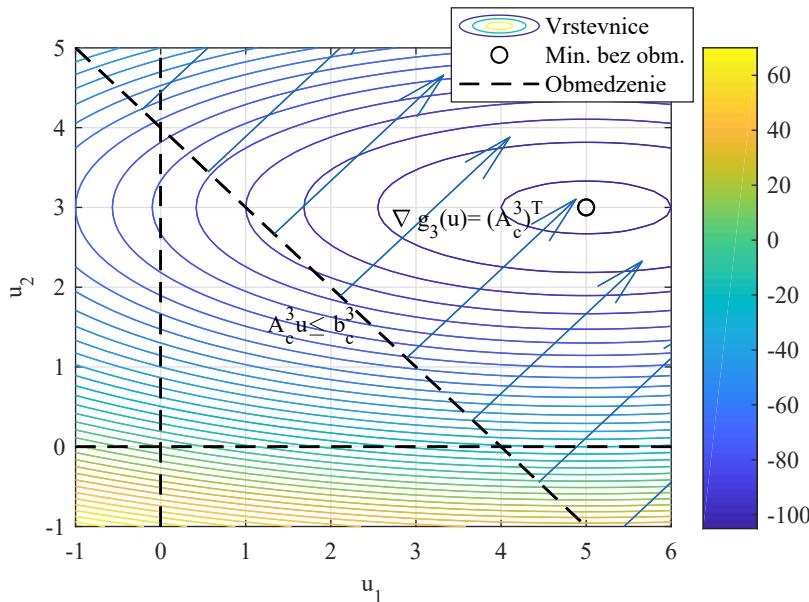
Zdrojový kód 6.8. vykreslitgradobm.m

```
function vykreslitgradobm(Ac,bc,i,hranice)
```

Pokračujeme definíciou rozšírenia vykreslovania, gradient zväčšíme desaťkrát. Vytvoríme vektor bodov nezávislej premennej  $u_1$ . Musíme zistiť, či ide o obmedzenie v tvare zvislej priamky. Vypočítame závislú premennú  $u_2$ , a v daných bodoch vždy zobrazíme gradient, ktorý je jednoducho daný riadkom matice  $\mathbf{A}_c$ . Vektory vykreslíme pomocou príkazu `quiver()`:

```
N=10; % Rozšírenie
U1=linspace(hranice(1),hranice(2),N); % Body
if Ac(i,2)==0 % Zvisla priamka?
    U2=U1; U1=zeros(1,N); % Zvisla priamka
    quiver(U1, U2, ones(1,N)*Ac(i,1), ones(1,N)*Ac(i,2));
else % Nie je zvisla
    U2=bc(i)-Ac(i,1).*U1./Ac(i,2); % Priamka
    quiver(U1, U2, ones(1,N)*Ac(i,1), ones(1,N)*Ac(i,2));
end
```

Novú funkciu môžeme otestovať pomocou zdrojového kódu z Pr. 6.5, kde musíme pridať iba relevantný riadok na zavolanie novej funkcie:

Obr. 6.13. Gradient obmedzenia s indexom  $i = 3$ .

Zdrojový kód 6.9. `testvykreslitgradientobmedzeni.m`

```
vykreslitgradobm (Ac, bc, 3, [-1 6]) % Grad. obmedz.
```

Po spustení dostaneme grafické zobrazenie gradientu posledného obmedzenia (Obr. 6.13). Vidíme, že gradienty v jednotlivých bodoch sú vždy kolmé na obmedzenia. Poznamenajme, že znázornenie obrázku v MATLABe má vplyv na zdanlivú kolmost gradientov. □

### 6.3 Kvadratické programovanie v MATLABe

Predtým ako si bližšie opíšeme matematické metódy riešenia problému kvadratického programovania, pozrime sa ešte na praktický nástroj, ktorý je určený na tento účel v MATLABe. Súčasťou rozšírenia *Optimization Toolbox* sú vyspelé algoritmy na optimalizáciu, inými slovami na hľadanie minima či maxima rôznych matematických optimalizačných problémov. Softvér na riešenie optimalizačných úloh je neformálne známy aj ako „riešič“<sup>14</sup>.

<sup>14</sup>angl. solver

Optimization Toolbox obsahuje aj riešič problému kvadratického programovania, ktorý môžeme zavolať príkazom `quadprog()`. Zdanivo jednoduchý príkaz v sebe zahŕňa rovno niekoľko spôsobov riešenia kvadratického programovania. Ak do príkazového riadku napíšeme `help quadprog()`, dozvieme sa, že príkaz rieši problém v tvare

```
min 0.5*x'*H*x + f'*x      subject to: A*x<=b
```

čo je totožné s našou formuláciou z Rov. (6.2) v absencii obmedzení v tvare rovností. Kvadratický problém s obmedzeniami potom môžeme vyriešiť volaním

```
u = quadprog(H, g, Ac, bc)
```

a dostaneme hľadaný argument v optime  $u^*$ . V prípade, že uvažujeme všeobecný problém QP aj s obmedzeniami v tvare rovnosti, môžeme použiť nasledujúcu formuláciu vstupných argumentov:

```
u = quadprog(H, g, Ac, bc, Ae, be)
```

Pridaním ďalšieho výstupného argumentu dostaneme aj funkčnú hodnotu v mi-

nime:

```
[u, f] = quadprog(H, g, Ac, bc)
```

V prípade, že existuje riešenie pre zadaný problém, `quadprog()` vráti stav riešenia s tvrdením

```
Minimum found that satisfies the constraints
```

čo znamená, že algoritmus našiel riešenie zlúčiteľné s obmedzeniami.

**Príklad 6.7.** Vyriešte úlohu kvadratického programovania z Pr. 6.1. Znázornite vrstevnice kvadratickej funkcie, riešenie bez uvažovania obmedzení, obmedzenia a riešenie pri uvažovaní obmedzení.

Na zobrazenie vrstevníc, obmedzení a riešenia bez uvažovania obmedzení vy-

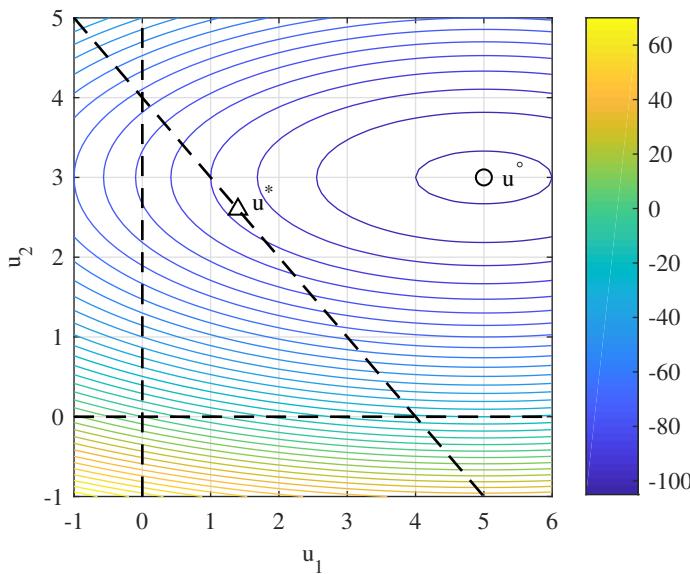
chádzame zo zdrojového kódu z Pr. 6.5. V zdrojovom kóde pokračujeme riešením

problému QP pomocou riešiča `quadprog()`, a výsledok vykreslíme do grafu:

Zdrojový kód 6.10. prveqp.m

```
[u, f] = quadprog(H, g, Ac, bc) % Riesenie QP
plot(u(1), u(2), 'k^') % Zobrazenie u*
```

Po spustení programu `quadprog()` tento oznamí, že úspešne našiel obmedzené minimum funkcie. Kedže sme nepotlačili výpis do príkazového riadku, dostaneme nasledujúci výstup:



Obr. 6.14. Riešenie minimalizácie QP a zobrazenie jeho argumentu v optim.

```

u =
 1.4000
 2.6000
f =
 -91.6000

```

čo znamená, že minimum pri uvažovaní obmedzení je v bode  $\mathbf{u}^* = [1.4 \ 2.6]^T$ , a minimálna hodnota funkcie v tomto bode je  $f(\mathbf{u}^*) = -91.6$ . Na Obr. 6.14 vidíme získanú geometrickú ilustráciu príkladu. Minimum kvadratickej funkcie bez uvažovania obmedzení,  $\mathbf{u}^\circ$ , sa nachádza mimo zlúčiteľnej množiny. Minimum pri uvažovaní obmedzení sa nachádza presne na hrane trojuholníka definovaného obmedzeniami.

□

### 6.3.1 Diagnostika riešiča

Rutina riešiča `quadprog()` tiež informuje používateľa po jej spustení pomocou výpisu stavu<sup>15</sup> o tom, či sa dopracovala k riešeniu a keď nie, z akého dôvodu. Ak pridáme ďalší výstupný argument do volania riešiča v tvare

`[u, f, e] =quadprog(H, G, Ac, b0)`

potom premenná `e` obsahuje túto diagnostickú informáciu. Presný opis statusov závisí aj od typu zvoleného algoritmu, ale niektoré diagnostické informácie sú spoločné. Ak algoritmus vráti status označený číslom 1, znamená to, že sa riadne dopracoval k riešeniu.

Príslušná numerická metóda je iteráčná, t. j. trvá určitý počet krokov. Vplyvom nepresnosti reprezentácie čísel môžu vzniknúť situácie, keď riešenie problému QP vyžaduje príliš veľa krokov, ktoré presahujú maximálny dovolený počet iterácií. Riešenie rozsiahlych problémov QP taktiež môže vyžadovať veľký počet krokov, respektíve miešanie veľkých a malých čísel môže vytvoriť numericky zle podmienenú úlohu. Rutina `quadprog()` je vždy spustená s predvoleným počtom iterácií, ktorý môžeme znížiť alebo zvýšiť. Ak `quadprog()` presiahne dovolený počet iterácií a stále nenájde riešenie, vráti nulu.

Stačí pridať jednu nerovnosť, ktorá nie je zlúčiteľná so zvyškom obmedzení, a dostávame nezlučiteľný problém. V tomto prípade riešič `quadprog()` vráti číslo -2. Existuje aj situácia, kedy kombinácia funkcie a obmedzení vedie na riešenie pri nekonečnej funkčnej hodnote, inak povedané existujú také `u`, pre ktoré  $\min f(\mathbf{u}) = -\infty$ . V prípade, že riešič vráti status -3, indikuje to, že daný problém je neohraničený.

Ďalší zaujímavý prípad nastáva, ak neplatí kladná definitnosť Hessiánu. Dostaneme tak nekonvexnú formuláciu, ktorú algoritmus deteguje statusom -6. Najdôležitejšie statusy rutiny `quadprog()` sú zhrnuté v Tab. 6.1.

Ak zadáme nesymetrický Hessián, algoritmus pokračuje v riešení a dostaneme zdanlivo správne výsledky. V tomto prípade ale `quadprog()` varuje užívateľa na nesymetrickosť Hessiánu a potom zmení pôvodný nesymetrický výraz na

$$\mathbf{H} = \frac{\mathbf{H} + \mathbf{H}^T}{2}, \quad (6.8)$$

čím sa pôvodný Hessián  $\mathbf{H}$  transformuje do symetrického tvaru.

**Príklad 6.8.** Vyskúšajte rôzne diagnostické výstupy programu `quadprog()` pre zadanie z Pr. 6.1 tým, že a) používate nezmenený príklad, b) znížite dovolený počet krokov na dva, c) pridáte nezlučiteľné obmedzenie  $u_1 \geq 5$ , d) Hessián zadáte so záporným znamienkom, e) vytvoríte nesymetrický Hessián zmenou  $\mathbf{H}_{\{1,2\}} = 1$ .

<sup>15</sup>angl. exit flag

Tabuľka 6.1. Diagnostické statusy rutiny quadprog().

Status	Výraz
1	Nájdené zlúčiteľné riešenie.
0	Prekročený počet krokov.
-2	Nenájdené zlúčiteľné riešenie.
-3	Nekonečný problém.
-6	Nekonvexný problém.

Začneme definovaním problému a pridajme ďalší výstupný argument pre výpis statusu. Pôvodný problém vráti riešenie a status  $e=1$ , čo znamená, že algoritmus našiel pôvodné riešenie:

Zdrojový kód 6.11. statusy.m

```
H=[2 0; 0 18]; g=[-10 -54]'; % H, g
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]'; % Ac, bc
[u,f,e]=quadprog(H,g,Ac,bc) % Riesenie QP
```

Namiesto toho, aby sme formulovali numericky zle podmienený problém, nastavme malý maximálny počet dovolených iterácií:

```
nastavenia = optimoptions('quadprog','MaxIter',2);
[u,f,e]=quadprog(H,g,Ac,bc,[],[],[],[],[],nastavenia)
```

Po spustení programu riešič vráti nasledujúce chybové hlásenie

Solver stopped prematurely

čo znamená, že niektoré z podmienok ukončenia boli aktivované ešte pred nájdením správneho riešenia. Ak sa pozrieme na diagnostický výstup, dostaneme  $e=0$ , čo vysvetluje túto situáciu: riešič dosiahol maximálny dovolený počet iterácií.

Ďalej vyskúšajme pridať nezlučiteľné obmedzenia. Ak pridáme podmienku  $u_1 \geq 5$  ( $-u_1 \leq -5$ ), neexistuje žiadne správne riešenie. Vytvorime alternatívne obmedzenia  $Acx$  a  $bcx$  a potom zavolajme rutinu QP:

```
Acx=[Ac; -1 0]; bcx=[bc; -5];
[u,f,e]=quadprog(H,g,Acx,bcx) % Nezlučitelne
% Status -2
```

Dostaneme chybové hlásenie v tvare

The problem is infeasible

čo znamená, že daný problém je nezlučiteľný. Na výstupe nájdeme prázdné entity pre argument a funkčnú hodnotu ( $u = []$ ,  $f = []$ ) a výstupný status bude  $e = -2$ .

Ak je Hessián záporne definitný, potom funkcia už nie je v tvare komplexnej paraboly, ale v tvare tzv. „sedla“. Sedlo má nekonvexný tvar s možnými lokálnymi riešeniami. Vyskúšajme to jednoduchou zmenou Hessiánu na  $H = -H$ :

```
Hx=-H; % Zap. definitne
[u, f, e]=quadprog(Hx, g, Ac, bc) % Varovanie
```

Po pustení príkazov dostaneme varovanie

The problem is non-convex

ktoré nás upozorňuje na nekonvexnosť funkcie a vráti status -6.

Zmeňme teraz Hessián tak, aby bol nesymetrický a začnime výpočet:

```
Hx=[2 1; 0 18]; % Nesymetricke H
[u, f, e]=quadprog(Hx, g, Ac, bc) % Varovanie
Hx=(Hx+Hx')/2 % Nove H
```

Rutina úspešne zbehne, ale červenou farbou vypíše do príkazového riadku varovanie

Warning: Your Hessian is not symmetric.

Resetting  $H = (H+H')/2$ .

čo znamená, že Hessián neboli symetrický, a tak ho rutina hned na začiatku prepočítala podľa  $H = (H + H^T)/2$ , čím sa nás pôvodný Hessián zmenil a dostali sme iba podobný výraz. Všimnime si, že namiesto

$$H = \begin{bmatrix} 2 & 1 \\ 0 & 18 \end{bmatrix}$$

sme dostali

$$H = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 18 \end{bmatrix}.$$

□

**Príklad 6.9.** Vygenerujte náhodnú kombináciu obmedzení pre Pr. 6.1, kým výpočet nebude predčasne zastavený z dôvodu prevýšenia maximálneho počtu možných iterácií.

Začneme definíciou  $H$  a  $g$  Pr. 6.1. Inicializujeme si výstupný status číslom 1, čo znamená, že riešič riadne našiel minimum QP. Vykonáme cyklus while až kým výpočet predčasne neskončí a pred každým výpočtom vygenerujeme dve náhodné obmedzenia:

Zdrojový kód 6.12. itermax.m

```

H=[2 0; 0 18]; g=[-10 -54]'; % H, g
exitflag=1 % Riadne o.k.
while exitflag~=0 % Kym nezast.
    Ac=rand(2); % Nahodne Ac
    bc=rand(2,1); % Nahodne bc
    [u,f,exitflag]=quadprog(H,g,Ac,bc); % Vypocet QP
end % Koniec cyklu

```

Po spustení programu dostaneme niekoľko desiatok až stoviek správnych riešení, ale po konečnom čase sa podmienený cyklus zastaví s varovaním o prekročenie maximálneho počtu krokov. Ak do príkazového riadku napíšeme `Ac` a `bc`, uvidíme aké obmedzenia vytvorili pre riešič túto náročnú numerickú situáciu.

□

Po pridaní ďalšieho výstupného argumentu v tvare

```
[u,f,e,vystup]=quadprog(H,G,Ac,b0)
```

dostaneme na výstupe riešiča ešte podrobnejšiu diagnostickú informáciu v rámci štruktúry `vystup`. Jej message tejto štruktúry obsahuje výstupný odkaz riešiča, `algorithm` obsahuje názov algoritmu.

Problém kvadratického programovania riešime numerickou metódou, pričom každá metóda potrebuje niekoľko krokov na to, aby sa dopracovala k riešeniu. Počet krokov potrebných na nájdenie minima funkcie môže byť dôležitou diagnostickou veličinou. Pole `iterations` obsahuje počet iterácií.

**Príklad 6.10.** Prešetríte diagnostický výstup pre Pr. 6.1. Potom zmeňte obmedzenia tak, aby ste navýšili počet iterácií potrebný na výpočet riešenia.

Na začiatku si definujeme problém a zavoláme riešič s dodatočnou výstupnou premennou:

Zdrojový kód 6.13. pocetiteracii.m

```

H=[2 0; 0 18]; g=[-10 -54]'; % H, g
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]'; % Ac, bc
[u,f,e,v]=quadprog(H,g,Ac,bc); % Vypocet QP
v % Iteracie

```

Po spustení prvej časti príkladu získame diagnostický výstup uložený v štruktúre `v`. Pole `v.iterations` obsahuje počet iterácií, čo v tomto prípade je iba 4.

Zmeňme obmedzenia a zopakujme proces:

```

Ac = [ 0.0094 0.1798; % Ine Ac
        0.1868 0.5021]
bc = [ 0.1922 0.7321]', % Ine bc
[u, f, e, v]=quadprog(H, g, Ac, bc); % Vypocet QP
v % Iteracie

```

Po opakovanom spustení vidíme, že pole `v.iterations` už má hodnotu 34. Riešiť teda potreboval rádovo viac krokov na nájdenie minima identickej kvadratickej funkcie, ale s inými obmedzeniami.

□

## 6.4 Metóda aktívnych množín

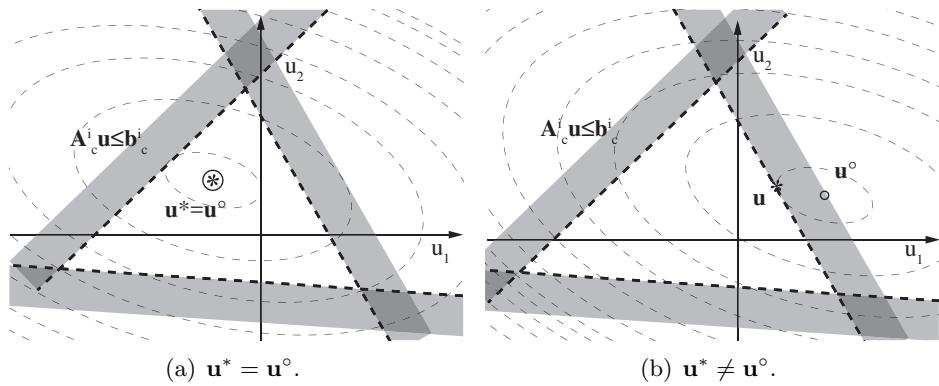
Rozlišujeme dva rôzne základné typy riešenia QP podľa polohy optima bez uvažovania obmedzení vzhľadom na obmedzenia. Ak je riešenie QP totožné s neobmedzenou minimálnou hodnotou kvadratickej funkcie, teda  $\mathbf{u}^* = \mathbf{u}^\circ$ , optimum sa nachádza niekde v množine zlúčiteľných riešení. Táto situácia je znázornená na Obr. 6.15(a), kde vidíme, že identické riešenie by sme dostali aj keby sme vôbec neuvažovali obmedzenia. Stred sústredných elíps, teda gradientov, je súčasťou množiny možných prípustných riešení.

Druhá možnosť je znázornená na Obr. 6.15(b), kde sa neobmedzené minimum kvadratickej funkcie nachádza mimo zlúčiteľnej množiny, a tým pádom nemôže byť platným riešením QP. Z uvedeného logicky vyplýva, že optimálne riešenie QP sa musí nachádzať na niektorom z obmedzení, respektíve na priesecníku viacerých obmedzení.

Všimnime si tiež ďalšiu zaujímavú vlastnosť obmedzení ilustrovanú na Obr. 6.16(a). Obmedzenia, na ktorých sa nenachádza obmedzené riešenie QP  $\mathbf{u}^*$ , nie sú vôbec súčasťou riešenia. Takéto obmedzenia budeme v ďalšej diskusii nazývať ako *neaktívne obmedzenia*. Na Obr. 6.15(a) neboli aktívne žiadne obmedzenia, a preto minimum bez uvažovania obmedzení je riešením celého QP. Na druhej strane, ak minimum bez uvažovania obmedzení nemôže byť riešením celého QP, zostáva jedno alebo viac obmedzení, na ktorých sa bude nachádzať skutočné riešenie. Tieto obmedzenia budeme ďalej nazývať *aktívne obmedzenia*. Známa metóda riešenia problému QP je založená práve na báze *aktívnych množín*<sup>16</sup> obmedzení a môžeme ju stručne nazvať QP aktívnych množín<sup>17</sup> (AS).

<sup>16</sup>angl. active-set (AS)

<sup>17</sup>angl. active-set QP



Obr. 6.15. Poloha „neobmedzeného“ riešenia problému QP vzhľadom na obmedzenia.

Na Obr. 6.15(b) sme už odstránili neaktívne obmedzenia. Ak vieme, že riešenie sa nachádza priamo na aktívnych obmedzeniach, nehovoríme už o nerovnosti, ale o rovnosti. To znamená, že aktívne obmedzenia potom tvoria obmedzenie kvadratickej funkcie v tvare rovnosti.



Neaktívne obmedzenia môžeme z optimalizačného problému odstrániť bez toho, aby to ovplyvnilo jeho konečné riešenie [26].

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} \left( \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} \right), \quad (6.9)$$

vzhľadom na podmienku:

$$\mathbf{A}_c^i \mathbf{u} = \mathbf{b}_c^i, \quad i \in \mathbf{a}^*,$$

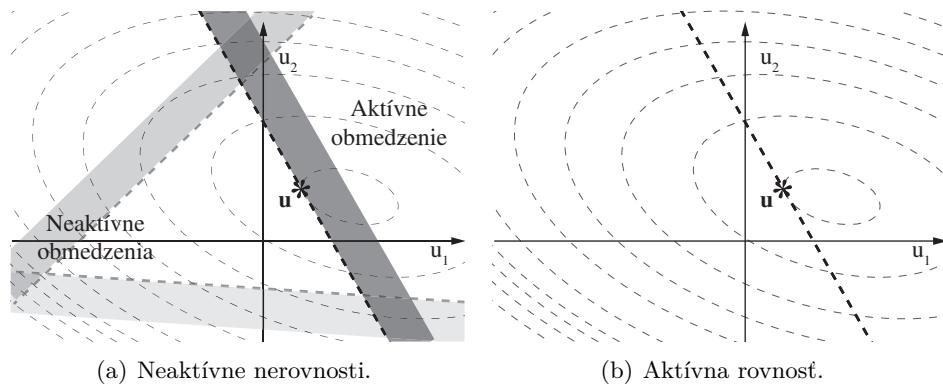
alebo obmedzenia v tvare  $g(\mathbf{u}) = \mathbf{A}_c \mathbf{u} - \mathbf{b}_c$  a samotnú kvadratickú funkciu  $f(\mathbf{u}) = 0.5 \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u}$  môžeme vyjadriť aj všeobecne v zmysle nasledujúceho problému:

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} f(\mathbf{u}), \quad (6.10)$$

vzhľadom na podmienku:

$$g_i(\mathbf{u}) = 0, \quad i \in \mathbf{a}^*,$$

kde  $\mathbf{a}^* = \{i : \mathbf{A}_c^i \mathbf{u}^* = \mathbf{b}_c^i\}$  je množina aktívnych obmedzení pri riešení  $\mathbf{u}^*$  problému kvadratického programovania [26]. Vektor  $\mathbf{a}^*$  obsahuje indexy riadkov, ktoré sú relevantné na výpočet obmedzeného minima problému QP. Ak poznáme  $\mathbf{a}^*$ , obmedzenia máme v tvare rovnosti a nie nerovnosti ako v pôvodnej formulácii.



Obr. 6.16. Ilustrácia aktívnych a neaktívnych obmedzení v tvare nerovnosti a rovnosti.

**Príklad 6.11.** Uvažujme problém kvadratického programovania z Pr. 6.1. Vypočítajte a zobrazte riešenie problému QP v prípade a) pôvodného problému, b) iba pre poslednú nerovnosť, a c) iba pre poslednú nerovnosť, ale obmedzenie zadajme v tvare rovnosti.

Začneme výpočtom a vykreslením pôvodného problému QP. V Tomto prípade stačí, ak zobrazíme posledné riadky matíc  $\mathbf{A}_c$  a  $\mathbf{b}_c$ :

Zdrojový kód 6.14. odstranenieobmedzeni.m

```

H=[2 0; 0 18]; g=[-10 -54]'; % H, g
Ac=[-1 0; 0 -1; 1 1]; % Matica Ac
bc=[0 0 4]; % Vektor bc
vykreslitvrstevnice(H,g,[-1 6],5) % Vrstevn. funkcie
vykreslitobmedzenia(Ac(3,:),bc(3),[-1 6]) % Obmedzenia
axis([-1 6 -1 5]) % Zobrazenie
[u,f]=quadprog(H,g,Ac,bc) % Riesenie QP
plot(u(1),u(2),'k^') % Zobrazenie u*

```

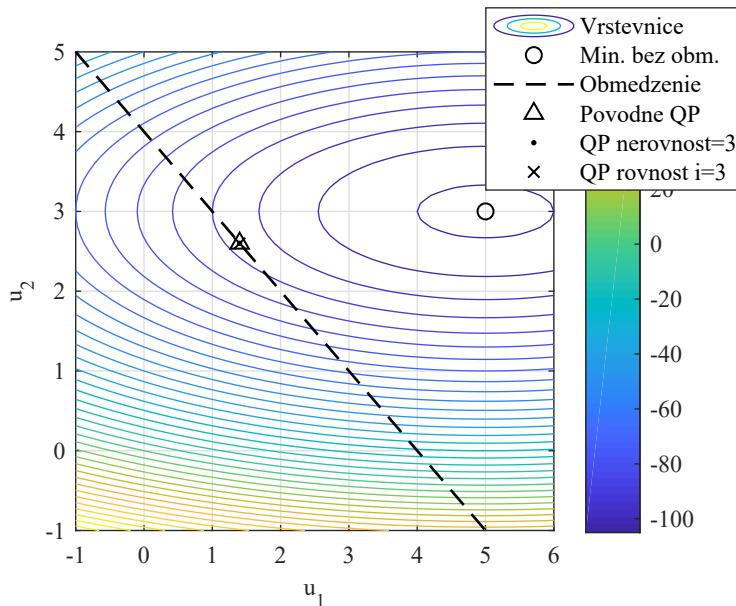
Nové obmedzenie dostaneme tak, že uvažujeme iba tretiu z pôvodných nerovností. Problém vyriešime pomocou rutiny quadprog() rovnako ako predtým:

```

Ac=Ac(3,:); bc=bc(3); % Ac, bc (iba i=3)
[u,f]=quadprog(H,g,Ac,bc) % Riesenie QP
plot(u(1),u(2),'k.') % Zobrazenie u*

```

zmeňme ďalej obmedzenie v tvare nerovnosti na obmedzenie v tvare rovnosti. Tretí a štvrtý vstupný argument quadprog() sú obmedzenia v tvare nerovnosti, ktoré



Obr. 6.17. Riešenie ostáva rovnaké aj po odstránení neaktívnych obmedzení.

tentoraz neuvažujeme. Obmedzenia v tvare rovnosti môžeme tak zadať ako piaty a šiesty argument:

```

Ae=Ac; be=bc; % Rovnosti
[u, f]=quadprog(H, g, [], [], Ae, be) % Riešenie QP
plot(u(1), u(2), 'kx') % Zobrazenie u*
legend('Vrstevnice', 'Min. bez obm.', 'Obmedzenie', %
'Povodne QP', 'QP nerovnost=3', 'QP rovnost i=3')

```

Po spustení príkladu príkazový riadok MATLABu vypíše riešenie pre všetky tri prípady. Vidíme, že všetky tri riešenia sú správne a identické ako je to graficky znázornené na Obr. 6.17. Odstránením neaktívnych obmedzení a uvažovaním aktívnych obmedzení v tvare rovnosti stále dostaneme správne riešenie QP.  $\square$

Samozrejme, vopred nemôžeme vedieť správnu konfiguráciu neaktívnych a aktívnych obmedzení  $\mathbf{a}^*$ , ktoré vedú na správne riešenie  $\mathbf{u}^*$ . Môžeme ale použiť stratégiu, pri ktorej daná aktívna množina obmedzení vždy viedie na zlepšené riešenie problému v každom kroku, pričom zistíme aj správnosť riešenia a súčasne upravujeme aj formuláciu aktívnej množiny.

Vobre navrhnutom algoritme QP nevyberáme a netestujeme aktívne množiny náhodne, namiesto toho sa používajú rôzne vyspelé stratégie na to, aby sme nepotrebovali vyskúšať všetky kombinácie možných konfigurácií. Aktívne množiny potom môžeme vybrať napríklad tak, aby sme v každej iterácii znížili hodnotu  $f(\mathbf{u})$ , a tým sa posunuli bližšie k lepšiemu, až napokon k optimálnemu riešeniu.

Aby sme vyriešili problém kvadratického programovania, predpokladajme, že poznáme počiatočný odhad riešenia  $\mathbf{u}_0$ , ktorý je zlučiteľný s obmedzeniami a tomu prislúchajúcu aktívnu množinu  $\mathbf{a}_0$ . Indexom  $p$  budeme označovať iterácie riešiča. Prvotný hrubý náčrt algoritmu kvadratického programovania, ktorý využíva koncept aktívnych množín obmedzení môžeme zhrnúť na základe Alg. 6.1 [22, 48, 101, 26].

---

#### Algoritmus 6.1 QP: Metóda aktívnych množín (hrubý náčrt)

---

- 1:  $\mathbf{u}_0, \mathbf{a}_0, p = 0$  ▷ Zlučiteľný odhad riešenia a prislúchajúca množina.
  - 2: **pokial**  $\text{riešenie} \neq 1$  **urob**
  - 3:    $\mathbf{u}_p \leftarrow \text{Rov. (6.11) pri } \mathbf{a}_p$  ▷ Riešenie rovnosti pri danom  $\mathbf{a}_p$
  - 4:   **ak**  $\mathbf{u}_p = \mathbf{u}^*$  **potom** ▷ Testujeme správnosť riešenia
  - 5:     riešenie  $\leftarrow 1$  ▷ Ukončíme algoritmus, máme riešenie
  - 6:   **inak** ▷ Ak nemáme riešenie
  - 7:      $p \leftarrow p + 1$  ▷ Pre ďalšiu iteráciu
  - 8:     Nájdeme obmedzenie  $i$ , ktoré porušilo zlučiteľnosť.
  - 9:      $\mathbf{a}_{p+1} \leftarrow \text{pridáme } i \text{ do aktívnej množiny}$  ▷ Nová aktívna množina
  - 10:   **koniec ak**
  - 11: **koniec pokial**
- 

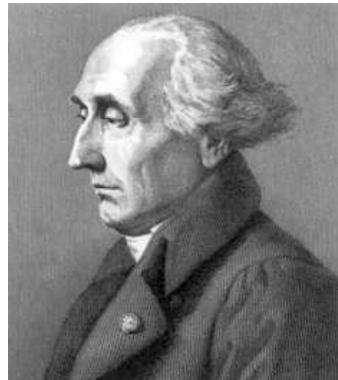
Algoritmus 6.1 otvára niekoľko nových otázok. Predpokladajme na chvíľu, že poznáme množinu aktívnych obmedzení  $\mathbf{a}^*$  alebo poznáme čiastkové riešenie  $\mathbf{u}_p$  a príslušnú množinu  $\mathbf{a}_p$ . Prvý problém vychádza z testovania správnosti riešenia. Ako môžeme zistiť, či riešenie problému QP (6.11), t. j. minimalizácie kvadratickej funkcie s obmedzeniami v tvare rovnosti pri danej konfigurácii aktívnej množiny je zároveň optimum problému QP, alebo nie je?

#### 6.4.1 Test optimality: Lagrangeove multiplikátory

Prvou úlohou pri rozvinutí hrubého algoritmu QP z Alg. 6.1 je navrhnutý test, ktorý hovorí o optimalite riešenia, teda presne určí, či dané riešenie je už konečné riešenie. Optimalitu čiastočného riešenia môžeme určiť na základe Lagrangeových multiplikátorov<sup>18</sup>.

---

<sup>18</sup>angl. Lagrange multipliers



Obr. 6.18. Joseph-Louis Lagrange (1736 až 1813) [42].

### Lagrange a jeho multiplikátory

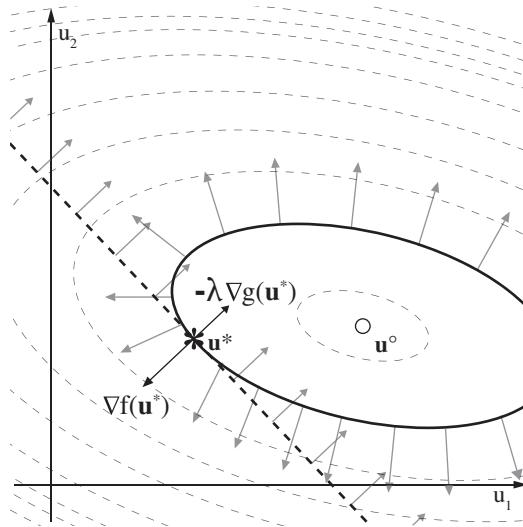
Joseph-Louis Lagrange (Obr. 6.18) bol známy taliansky matematik, fyzik a astronóm. Je známy pre svoje príspevky v matematickej analýze aj za prínos v mechanike. Lagrange bol jeden zo zakladateľov variačného počtu spolu s Leonhardom Eulerom, ktorého osobne poznal. Metóda Lagrangeových multiplikátorov je matematická stratégia nájdenia extrému funkcie v prítomnosti obmedzení v tvare rovnosti.

Z predchádzajúcich kapitol už vieme, že na to, aby sme otestovali optimalitu konvexného optimalizačného problému bez obmedzení,  $\min_{\mathbf{u}} f(\mathbf{u})$ , musíme jednoľucho vyskúšať, či je pre dané riešenie  $\mathbf{u}_p$  (nutná) podmienka  $\nabla_{\mathbf{u}} f(\mathbf{u}_p) = 0$  platná alebo nie. Ak dosadíme nejaké riešenie do gradientu funkciu a dostaneme nulu, máme optimálne riešenie. V prítomnosti obmedzení však už táto úloha nebude tak jednoduchá.

Vo všeobecnosti môžeme predpokladať, že pre problém s celkovo  $n_o$  obmedzeniami v tvare nerovnosti dostaneme riešenie pri aktívnej množine s  $m$  aktívnymi obmedzeniami v tvare rovnosti na základe

$$\begin{aligned} \mathbf{u}^* &= \arg \min_{\mathbf{u}} f(\mathbf{u}), \\ &\text{vzhľadom na podmienku:} \\ &g_i(\mathbf{u}) = 0, \quad i = 1, \dots, m, \end{aligned} \tag{6.11}$$

čo samozrejme taktiež platí pre danú aktívnu množinu  $\mathbf{a}_p$ , ktorá prislúcha problému (6.11), kde  $m$  aktívnych obmedzení má tvar  $g_i(\mathbf{u}) = \mathbf{A}_c^i \mathbf{u} - \mathbf{b}_c^i$ .



Obr. 6.19. Gradient funkcie a obmedzení sú v optime rovnobežné, pričom majú opačný smer a inú veľkosť.

Kedže teraz uvažujeme už aj obmedzenia v tvare rovnosti, nestačí iba vykonať test rovnosti gradientu nule. Namiesto toho musíme nejak do podmienky rovnosti gradientu nule zahrnúť aj vplyv obmedzení.

Vo všeobecnosti kvadratická rovnica predstavuje parabolu v mnohorozmernom priestore a každé obmedzenie v tvare rovnosti predstavuje jednu plochu. Pre jednoduchší výklad si predstavme parabolu v 3D priestore a plochu, ktorá vytvára obmedzenie v 3D. Pri danej konfigurácii aktívnej množiny hľadáme bod, v ktorom sa parabola dotýka daného obmedzenia. Ak si tú istú situáciu predstavíme v 2D priestore, hľadáme vrstevnicu, ktorá sa práve dotýka obmedzenia (Obr. 6.19). Tento bod dotyku bude predstavovať naše riešenie. Zo sveta funkcií so skalárnymi premennými vieme, že dotyčnice vyjadrujeme pomocou derivácie funkcie, čo pre viacparametrové funkcie robíme pomocou gradientov.

Ak by sme mali iba jedno aktívne obmedzenie ( $m = 1$ ), potom by sme podmienku optimality na základe tejto úvahy mohli napísat v tvare

$$\nabla_{\mathbf{u}} \mathbf{f}(\mathbf{u}^*) = -\lambda \nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}^*). \quad (6.12)$$

Uvedená podmienka hovorí, že smer gradientu minimalizovanej funkcie v optime  $\mathbf{u}^*$  musí byť presne opačný ako smer gradientu obmedzenia. Záporné znamienko na pravej strane vyjadruje opačnú orientáciu vektorov gradientu v bode  $\mathbf{u}^*$ , zatiaľ čo rovnosť výrazov na oboch stranách vyjadruje ich rovnobežnosť. Skalár  $\lambda$

je tzv. *Lagrangeov multiplikátor*<sup>19</sup>, a potrebujeme ho vo formulácii podmienky, pretože zatiaľ čo gradienty sú rovnobežné a opačne orientované, ich dĺžky už nemusia byť totožné. Podmienku optimality pre jedno obmedzenie v tvare rovnosti z Rov. (6.13) môžeme veľmi jednoducho zmeniť pre prípad viacerých obmedzení. Koncept je graficky znázornený aj na Obr. 6.19.

Je zrejmé, že optimalizovaná funkcia a obmedzenia sa musia v optime dotýkať. Lagrangeove multiplikátory vyjadrujú, že napriek tomu môže byť ich rast v danom bode odlišný. Pri formulácii podľa Rov. (6.13) má preto záporné znamienko Lagrangeovho multiplikátora tiež geometrický význam. Táto skutočnosť vyjadruje jednoducho to, že nie je jedno, z ktorej strany sa funkcia dotýka obmedzenia v tvare nerovnosti. Nezápornosť multiplikátora, t. j. podmienka  $\lambda \geq 0$ , tvorí dôležitú súčasť stratégie riešenia optimalizačných úloh.

Pre prípad obmedzenia v tvare rovnosti,  $\mathbf{h}(\mathbf{u})$ , môžeme analogicky tvrdiť, že v optime musí platiť

$$\nabla_{\mathbf{u}} \mathbf{f}(\mathbf{u}^*) = -\nu \nabla_{\mathbf{u}} \mathbf{h}(\mathbf{u}^*), \quad (6.13)$$

kde  $\nu$ , podobne ako  $\lambda$ , je Lagrangeov multiplikátor, ktorý ovplyvňuje rozdiel stúpania funkcie. Na rozdiel od obmedzení v tvare rovnosti tu nepotrebujeme dodatočnú podmienku kladnosti multiplikátora  $\nu$ , pretože ide o rovnosť, kde „strana“ dotyku nie je podstatná. Nezabudnime, že tu nehovoríme o aktívnej množine: všetky obmedzenia v tvare rovnosti musia platiť.

**Príklad 6.12.** Graficky znázornite gradient vrstevnice v riešení QP  $\mathbf{f}(\mathbf{u}^*)$  z Pr. 6.1 a gradient posledného obmedzenia s indexom  $i = 3$ . Ručne vypočítajte hodnotu gradientov v optime, t. j.  $\nabla \mathbf{f}(\mathbf{u}^*)$ ,  $\nabla \mathbf{g}(\mathbf{u}^*)$ , potom nájdite  $\lambda$  podľa Rov. (6.13). Výsledky overte výpočtom v MATLABe a vykreslite gradienty v optime do grafu.

Začneme definovaním príkladu, hraníc zobrazenia grafu, riešením QP pomocou quadprog a nakreslením riešenia:

Zdrojový kód 6.15. smergradientov.m

```

H=[2 0; 0 18]; g=[-10 -54]'; % H, g
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]; % Ac, bc
hranice=[-6 6]; % Hranice
[uast,f]=quadprog(H,g,Ac,bc) % Riesenie QP
plot(uast(1),uast(2),'k^') % Zobraz u*
hold on % Podrz graf

```

<sup>19</sup>angl. Lagrange multiplier

V príklade pokračujeme znázornením situácie pomocou nami už vytvorených funkcií. Vykreslíme si vrstevnice funkcie, posledné obmedzenie, gradienty vrstevnice v  $f(\mathbf{u}^*)$  a gradienty posledného obmedzenia:

```

vykreslitvrstevnice(H,g,hranice,100)           % Vrstevn. f
vykreslitobmedzenia(Ac(3,:),bc(3),hranice)    % Obmedzenie
vykreslitgradient(H,g,hranice,f)              % Gradient f
vykreslitgradobm(Ac,bc,3,hranice)              % Gradient g
axis([hranice [-5 6]])                         % Zobrazenie

```

Gradient kvadratickej funkcie a obmedzenia v optime dostaneme ako

$$\nabla f(\mathbf{u}^*) = \mathbf{H}\mathbf{u}^* + \mathbf{g} = \begin{bmatrix} 2 & 0 \\ 0 & 18 \end{bmatrix} \begin{bmatrix} 1.4 \\ 2.6 \end{bmatrix} + \begin{bmatrix} -10 \\ -54 \end{bmatrix} = \begin{bmatrix} -7.2 \\ -7.2 \end{bmatrix},$$

$$\nabla g_{i=3}(\mathbf{u}^*) = (\mathbf{A}_c^{i=3})^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

potom v zmysle Rov. (6.13) platí

$$\begin{bmatrix} -7.2 \\ -7.2 \end{bmatrix} = -\lambda \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

na základe čoho  $\lambda = 7.2$ . Vidíme, že vektor gradientov v optime majú rovnaký smer, opačnú orientáciu a inú dĺžku (veľkosť). Lagrangeov multiplikátor je kladný, čím vyjadrujeme aj platnú stranu obmedzenia v tvare nerovnosti.

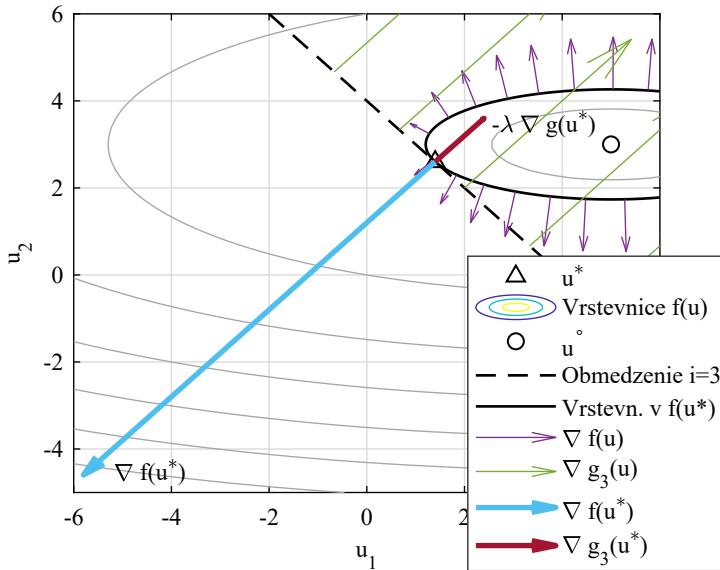
Výsledky teraz prekontrolujeme výpočtom a vizualizáciou v MATLABe. Vypočítame gradienty a nakreslíme ich pomocou príkazu `quiver()`. Posledný, nulový vstupný argument príkazu znamená, že vypneme automatické škálovanie dĺžky vektorov:

```

nf = H*uast+g;                                % nf f(uast)
quiver(uast(1),uast(2), nf(1),nf(2),0);      % Zobraz nf.
ng = Ac(3,:);                                % ng g3(uast)
quiver(uast(1),uast(2), ng(1),ng(2),0);      % Zobraz ng.
legend('u^{\ast}', 'Vrstevnice f(u)', 'u^{\circ}', 
      'Obmedzenie i=3', 'Vrstevnica v f(u^{\ast})', '\nabla f(u)', 
      '\nabla g_3(u)', '\nabla f(u^{\ast})', '\nabla g_3(u^{\ast})')

```

Po spustení príkladu dostaneme výsledok podľa Obr. 6.20. Dĺžky gradientov vykreslovaných pomocou našich funkcií sú automaticky zmenené, preto nie sú totožné s dĺžkami gradientov v optime.



Obr. 6.20. Gradienty v bode riešenia sú rovnobežné, majú opačný smer a rôzne dĺžky (veľkosti).

□

Ak chceme overiť, či čiastkové riešenie  $\mathbf{u}_p = \mathbf{u}^*$  je globálnym riešením problému (6.11) pre aktuálnu aktívnu množinu  $\mathbf{a}_p$  v iterácii  $p$ , potom musíme nájsť také nezáporné Lagrangeove multiplikátory  $\lambda_i^*, i = 1, 2, \dots, m$ , ktoré splňajú podmienky [27]

$$\nabla_{\mathbf{u}} \mathbf{f}(\mathbf{u}^*) + \sum_{i=1}^m \lambda_i^* \nabla_{\mathbf{u}} g_i(\mathbf{u}^*) = 0, \quad i = 1, 2, \dots, m; \quad (6.14)$$

$$g_i(\mathbf{u}^*) = 0, \quad i = 1, 2, \dots, m.$$

To znamená, že nestačí nájsť bod dotyku funkcie a obmedzení, ale musia platiť aj pôvodné obmedzenia. Inými slovami, ak nájdeme množinu Lagrangeových operátorov  $\lambda_i^* \geq 0$  pre dané čiastkové riešenie, čiastkové riešenie je zároveň riešením celého pôvodného problému kvadratického programovania.

#### 6.4.2 Zovšeobecnenie Lagrangeových multiplikátorov

Podmienku optimality z Rov. (6.14) môžeme rozšíriť aj pre  $n_e$  dodatočných obmedzení v tvare rovnosti  $\mathbf{A}_e \mathbf{u} = \mathbf{b}_e$  podľa pôvodnej všeobecnej formulácie problému

QP z Rov. (6.2), ktoré sú definované ako

$$h_j(\mathbf{u}) = \mathbf{A}_e^j \mathbf{u} - \mathbf{b}_e^j, \quad j = 1, \dots, n_e, \quad (6.15)$$

respektíve

$$\mathbf{h}(\mathbf{u}) = \mathbf{A}_e \mathbf{u} - \mathbf{b}_e \quad (6.16)$$

pre všetky obmedzenia v tvare rovnosti.

Všimnime si, že v prípade obmedzení iba v tvare rovnosti nemôžeme hovoriť o aktívnej množine, pretože všetky obmedzenia v tvare rovnosti musia byť splnené v riešení. Zatial čo pre obmedzenia v tvare nerovnosti uvažujeme iba aktívnu podmnožinu ( $m$  z celkových  $n_o$ ), pre rovnosti musíme uvažovať celú množinu.

Nevyhnutné a postačujúce podmienky, aby riešenie  $\mathbf{u} = \mathbf{u}^*$  bolo globálnym optimom sú dané takzvanými *Karush-Kuhn-Tuckerovymi (KKT) podmienkami optimality*<sup>20</sup> podľa [48]:

$$\nabla_{\mathbf{u}} \mathbf{f}(\mathbf{u}^*) + \sum_{i=1}^m \lambda_i \nabla_{\mathbf{u}} g_i(\mathbf{u}^*) + \sum_{j=1}^{n_e} \nu_j \nabla_{\mathbf{u}} h_j(\mathbf{u}^*) = 0, \quad (6.17a)$$

$$g_i(\mathbf{u}^*) \leq 0, \quad i = 1, \dots, m; \quad (6.17b)$$

$$h_j(\mathbf{u}^*) = 0, \quad j = 1, \dots, n_e; \quad (6.17c)$$

$$\nu_j \geq 0, \quad i = 1, \dots, m; \quad (6.17d)$$

$$\nu_j h_j(\mathbf{u}^*) = 0, \quad i = 1, \dots, m; \quad (6.17e)$$

kde  $\nu_i$  sú dodatočné Lagrangeove multiplikátory pre obmedzenia v tvare rovnosti. Ak sú uvedené KKT podmienky splnené, potom  $\mathbf{u}^*$  je optimálnym riešením problému kvadratického programovania, ktorý zahŕňa obmedzenia v tvare rovnosti aj nerovnosti. Rov. (6.17a) predstavuje tzv. podmienku stacionarity<sup>21</sup>, Rov. (6.17b)–Rov. (6.17c) vyjadrujú podmienku primárnej zlučiteľnosti<sup>22</sup>, Rov. (6.17d) podmienku duálnej zlučiteľnosti<sup>23</sup>, a Rov. (6.17e) podmienku komplementarity doplnkových premenných<sup>24</sup>. Podrobnejšiu diskusiu problematiky KKT podmienok a teórie duality možno nájsť napríklad v knihe [22].

Sústava rovníc (6.17), ktoré definujú KKT podmienky, nie je riešený priamo, ale numericky. Existujú pritom rôzne numerické spôsoby ako zefektívniť riešenie KKT podmienok [115].

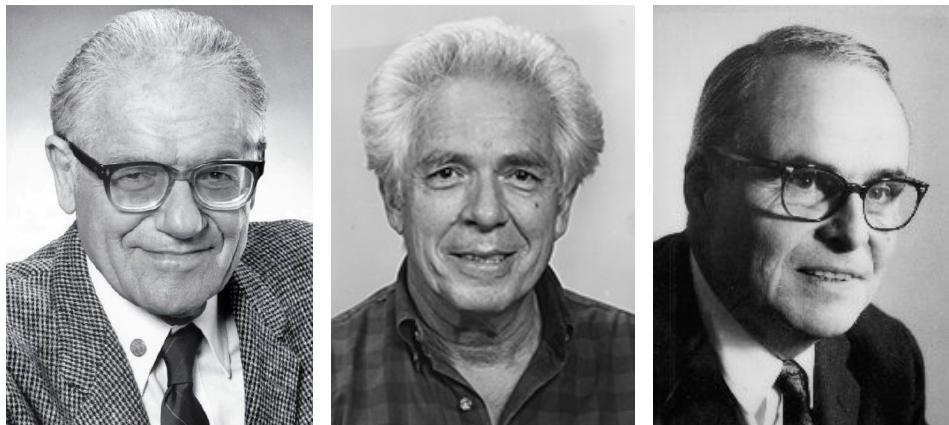
<sup>20</sup>angl. Karush-Kuhn-Tucker (KKT) optimality conditions

<sup>21</sup>angl. stationarity

<sup>22</sup>angl. primal feasibility

<sup>23</sup>angl. dual feasibility

<sup>24</sup>angl. complementary slackness



(a) W. Karush (1917–1997) (b) H.W. Kuhn (1925–2004) (c) A.W. Tucker (1905–1995)

Obr. 6.21. Karush, Kuhn a Tucker [36, 41, 90].

### Karush, Kuhn, Tucker, a ich známa podmienka optimality

Pôvodne podmienky na určenie optimality nelineárneho programovania boli pomenované ako Kuhn-Tuckerove podmienky po Haroldovi Kuhnovi (Obr. 6.21(b)) a Albertovi Tuckerovi (Obr. 6.21(c)), ktorí ich sformulovali v roku 1951. Neskôr ale výskumníci prišli na to, že William Karush (Obr. 6.21(a)) už tieto podmienky predtým našiel a opísal vo svojej diplomovej práci z roku 1939 [36]!

Kým Lagrangeove multiplikátory dovoľujú iba obmedzenia v tvare rovnosti, KKT sústava zovšeobecňuje podmienku optimality aj pre problémy s obmedzeniami v tvare nerovnosti. Podmienky KKT nie sú určené iba pre prípad kvadratického programovania ale pomocou nich môžeme vyriešiť mnoho iných optimalizačných problémov.

Všetci menovaní pôsobili ako profesori matematiky: Kuhn a Tucker na Princetone a Karush na Kalifórnskej štátnej univerzite v Northridge. Albert Tucker je známy nie len vďaka KKT podmienkam, ale aj pre svoje príspevky v teórii hier. Známy matematický problém s názvom „Väzňova dilema“ bol tak pomenovaný Tuckerom v 50. rokoch.

Zaujímavým faktom je, že Kuhn a Tucker dobre poznali známeho matematika Johna Nasha, ktorý získal Nobelovu cenu v 1994 za svoje prínosy v ekonomike. Kým vedúcim dizertačnej práce Nasha bol Tucker, inšpiráciou známeho filmu „Čistá duša“ bola publikácia od Kuhna [82].

### 6.4.3 KKT podmienky pre QP

Už vieme, že podmienky optimality môžeme vo všeobecnosti opísať pomocou tzv. KKT podmienok (6.17). Teraz si dosadme za všeobecnú funkciu našu kvadratickú funkciu a za všeobecné obmedzenia naše konkrétnie lineárne obmedzenia.

Po dosadení za kvadratickú funkciu  $\mathbf{f}(\mathbf{u})$  a obmedzenia  $\mathbf{g}(\mathbf{u})$  a  $\mathbf{h}(\mathbf{u})$  môžeme výrazy derivovať, aby sme našli gradienty v tvare

$$\nabla_{\mathbf{u}} \mathbf{f}(\mathbf{u}) = \frac{\partial \left( \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} \right)}{\partial \mathbf{u}} = \mathbf{H} \mathbf{u} + \mathbf{g}, \quad (6.18)$$

$$\nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}) = \frac{\partial (\mathbf{A}_c \mathbf{u} - \mathbf{b}_c)}{\partial \mathbf{u}} = \mathbf{A}_c^T, \quad (6.19)$$

$$\nabla_{\mathbf{u}} \mathbf{h}(\mathbf{u}) = \frac{\partial (\mathbf{A}_e \mathbf{u} - \mathbf{b}_e)}{\partial \mathbf{u}} = \mathbf{A}_e^T. \quad (6.20)$$

Označme si teraz zostavu obmedzení pri aktívnej množine  $\mathbf{a}_p$  v kroku  $p$  ako  $g_a(\mathbf{u})$ , pričom aktívnu podmnožinu pôvodných nerovností máme danú ako  $\mathbf{A}_c^a \leq \mathbf{b}_c^a$ . Dosadením derivácií do (6.17) dostaneme KKT podmienky pre kvadratické programovanie v tvare [101]:

$$\mathbf{H} \mathbf{u} + \mathbf{A}_c^{aT} \boldsymbol{\lambda} + \mathbf{A}_e^T \boldsymbol{\nu} = -\mathbf{g} \quad (6.21)$$

$$\mathbf{A}_c^a \mathbf{u} = \mathbf{b}_c^a \quad (6.22)$$

$$\mathbf{A}_e \mathbf{u} = \mathbf{b}_e \quad (6.23)$$

kde multiplikátory  $\boldsymbol{\lambda}$  prislúchajúce neaktívnym obmedzeniam sú nulové, čo môžeme zabezpečiť dodatočnou podmienkou. Inými slovami, musíme rátať len s rovnicami, ktoré súvisia s aktívnymi obmedzeniami a keď sú Lagrangeove multiplikátory nezáporné, potom čiastkové riešenie  $\mathbf{u}_p$  je globálnym riešením problému kvadratického programovania  $\mathbf{u}^* = \mathbf{u}_p$ .

KKT podmienku optimality pre všeobecný problém kvadratického programovania môžeme formulovať v maticovom tvare ako

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}_c^{aT} & \mathbf{A}_e^T \\ \mathbf{A}_c^a & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_e & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -\mathbf{g} \\ \mathbf{b}_c^a \\ \mathbf{b}_e \end{bmatrix}. \quad (6.24)$$

zatial čo pre obmedzenia v tvare nerovnosti dostaneme

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}_c^{aT} \\ \mathbf{A}_c^a & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{g} \\ \mathbf{b}_c^a \end{bmatrix}. \quad (6.25)$$

Ak je Hessián kladne definitný a  $\mathbf{A}_c^a$  a  $\mathbf{A}_e$  majú plnú hodnosť, ľavá strana výrazov, ktorú môžeme nazvať *KKT maticou*, ktorá je nesingulárna a neznáme výrazy majú jednoznačné riešenie [115].

**Príklad 6.13.** Vyriešte podmienku optimality na základe Rov. (6.25) pre zadanie z Pr. 6.1 pre aktívnu množinu *a*) všetkých pôvodných obmedzení, t.j.  $\mathbf{a} = \{1, 2, 3\}$  *b*) dvoch posledných obmedzení, t.j.  $\mathbf{a} = \{2, 3\}$ , a *c*) posledného obmedzenia, t.j.  $\mathbf{a} = \{3\}$ .

Začneme definíciou QP z Pr. 6.1:

Zdrojový kód 6.16. lagrange.m

```
H=[2 0; 0 18]; g=[-10 -54]'; % H, g
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]'; % Ac, bc
```

V prvom prípade uvažujme všetky obmedzenia. Ak symbolicky prepíšeme Rov. (6.25) ako  $\mathbf{Wz} = \mathbf{V}$ , tak jej riešenie môžeme vypočítať napríklad inverziou maticy  $\mathbf{W}$  a násobením zľava, aby sme dostali  $\mathbf{z} = \mathbf{W}^{-1}\mathbf{V}$ :

```
AC=Ac; BC=bc; % Vsetky obmedz.
[no n]=size(AC); % Pocet obm.
W=[H AC' ; % Lava strana
  AC zeros(no)]
V=[-g; % Prava strana
   BC]
z=inv(W)*V % Riesenie
```

Lavá strana výrazu, v našom prípade  $\mathbf{W}$ , predstavuje KKT maticu. Inverzia nie je bežným spôsobom riešenia sústavy rovníc, v tomto prípade ju používame iba na ilustračné účely. Po spustení programu dostaneme nasledujúci výsledok:

```
Warning: Matrix is close to singular or badly
scaled. Results may be inaccurate. RCOND =
4.625929e-19.
> In lagrange (line 12)
z =
 1.0e+17 *
 0
 0
-1.4412
-1.4412
-1.4412
```

Po prvej, vidíme, že inverzia bola numericky zle podmienená. Vieme, že KKT matica nebude singulárna, ale pre výpočet s konečnou presnosťou táto konfigurácia stále predstavuje výzvu. Pre algoritmy QP sú podobné detaily veľmi dôležité a v

prítomnosti desiatok alebo stoviek obmedzení je operácia inverzie problematická, a preto je výpočet KKT podmienok v praxi riešený inými numerickými prístupmi. Napriek tomu sme sa dopracovali k riešeniu, kde Lagrangeove multiplikátory sú záporné, a teda získaný výsledok nemôže byť správny.

Vyskúšajme teraz aktívnu množinu  $\mathbf{a} = \{2, 3\}$ , kde uvažujeme iba posledné dve obmedzenia. Samozrejme, z predchádzajúcich príkladov už vieme, že ani to nebude správne; napriek tomu je však zaujímavé vypočítať výsledok, ktorý bude:

$$\begin{aligned} \mathbf{z} = \\ 4 \\ 0 \\ -52 \\ 2 \end{aligned}$$

V tomto prípade sme už nenašli na numerické problémy, stále sme však nenašli riešenie, pretože prvý Lagrangeov multiplikátor je záporný. Ponúkané nesprávne riešenie  $\mathbf{u} = [4 \ 0]^T$  by znamenalo, že optimum sa nachádza v „hornom rohu“ obmedzení. Prvý Lagrangeov multiplikátor prislúcha obmedzeniu  $i = 2$ . Mohli by sme teda zvoliť takú stratégiu, že odstránime z aktívnej množiny tie obmedzenia, ktorým prislúcha záporný Lagrangeov multiplikátor.

Na záver si ešte otestujme aktívnu množinu  $\mathbf{a} = \{3\}$ , kde by sme na základe vyššie uvedenej logiky odstránili druhé obmedzenie. Po vyriešení problému dostaneme výsledok

$$\begin{aligned} \mathbf{z} = \\ 1.4000 \\ 2.6000 \\ 7.2000 \end{aligned}$$

Uvažujeme pritom iba jedno obmedzenie, a teda iba jeden multiplikátor, ktorý je kladný a navyše totožný s našim výpočtom z Pr. 6.12. Prvé dva prvky získaného vektoru  $\mathbf{z}$  predstavujú správne riešenie.

□

#### 6.4.4 Iteratívne zlepšenie riešenia

Rozumnú stratégiu praktickej implementácie algoritmu kvadratického programovania pomocou metódy aktívnych množín môžeme založiť na modifikácii kvadratickej účelovej funkcie (6.2). Namiesto uvažovania s absolútou hodnotou čiastočného riešenia  $\mathbf{u}_p$  v iterácii  $p$  budeme používať prírastkové<sup>25</sup> zlepšenia  $\mathbf{u}_p + \Delta\mathbf{u}$ .

<sup>25</sup>angl. incremental

Ak toto dosadíme do kvadratickej účelovej funkcie  $f(\mathbf{u})$ , dostaneme jej novú, prírastkovú formu v tvare

$$f(\mathbf{u}_p + \Delta \mathbf{u}) = \frac{1}{2}(\mathbf{u}_p + \Delta \mathbf{u})^T \mathbf{H}(\mathbf{u}_p + \Delta \mathbf{u}) + \mathbf{g}^T(\mathbf{u}_p + \Delta \mathbf{u}), \quad (6.26)$$

čo môžeme zjednodušiť na

$$f(\mathbf{u}_p + \Delta \mathbf{u}) = f(\mathbf{u}_p) + \frac{1}{2} \Delta \mathbf{u}^T \mathbf{H} \Delta \mathbf{u} + (\mathbf{g}^T + \mathbf{u}_p^T \mathbf{H}) \Delta \mathbf{u}. \quad (6.27)$$

Pôvodný problém QP s obmedzeniami v tvare rovnosti potom môžeme napísť v prírastkovom tvare pre danú aktívnu množinu ako

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} \left( \frac{1}{2} \Delta \mathbf{u}^T \mathbf{H} \Delta \mathbf{u} + \mathbf{g}_p^T \Delta \mathbf{u} \right), \quad (6.28)$$

vzhľadom na podmienku:

$$\mathbf{A}_c^i \Delta \mathbf{u} = 0, \quad i \in \mathbf{a}^*,$$

kde

$$\mathbf{g}_p^T = (\mathbf{g}^T + \mathbf{u}_p^T \mathbf{H}), \quad (6.29)$$

respektíve v prípade prítomnosti obmedzení vo forme rovnosti

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} \left( \frac{1}{2} \Delta \mathbf{u}^T \mathbf{H} \Delta \mathbf{u} + \mathbf{g}_p^T \Delta \mathbf{u} \right), \quad (6.30)$$

vzhľadom na podmienku:

$$\mathbf{A}_c^i \Delta \mathbf{u} = 0, \quad i \in \mathbf{a}^*,$$

$$\mathbf{A}_e \Delta \mathbf{u} = 0, \quad (6.31)$$

čím sme dostali QP, ale tentokrát už iba s obmedzeniami v tvare rovnosti.

Poznamenajme, že nerovnosti v tejto novej formulácii sú taktiež zmenené odstránením výrazu  $\mathbf{b}_c^i$ . Toto môžeme vysvetliť dosadením prírastkového riešenia  $\mathbf{u}_p + \Delta \mathbf{u}$  do nerovnosti, aby sme dostali

$$\mathbf{A}_c^i \mathbf{u}_p + \mathbf{A}_c^i \Delta \mathbf{u} = \mathbf{b}_c^i, \quad (6.32)$$

z ktorého musíme zaistiť len aby  $g_i(\Delta \mathbf{u}) = \mathbf{A}_c^i \Delta \mathbf{u} = 0$ , teda aby sme dodržali platnosť pôvodného obmedzenia. Ak platí  $\mathbf{A}_c^i \Delta \mathbf{u} = 0$ , platí aj rovnosť  $\mathbf{A}_c^i \mathbf{u}_p + 0 = \mathbf{b}_c^i$ .

Použime teraz nové obmedzenie  $\mathbf{g}(\Delta \mathbf{u})$  spolu s novou kvadratickou funkciou  $\mathbf{f}(\Delta \mathbf{u})$ , dosadíme ich do všeobecných podmienok optimality podľa Rov. (6.14) a

označme si nové Lagrangeove multiplikátory pomocou  $\Delta\lambda$ , ktoré už nemusia byť nevyhnutne kladné. Dostávame nasledujúcu podmienku optimality

$$\begin{aligned}\nabla f(\Delta u) + \Delta\lambda \nabla g_a(\Delta u) &= \mathbf{0}, \\ g_a(\Delta u) &= \mathbf{0},\end{aligned}\tag{6.33}$$

kde index  $a$  vyjadruje fakt, že uvažujeme len s aktívnymi obmedzeniami z množiny  $\mathbf{a}_p$  v iterácii  $p$ .

Parciálna derivácia kvadratickej funkcie a obmedzení podľa premennej  $\Delta u$  sú

$$\nabla_{\Delta u} f(\Delta u) = \frac{\partial \left( \frac{1}{2} \Delta u^T \mathbf{H} \Delta u + \mathbf{g}_p^T \Delta u \right)}{\partial \Delta u} = \mathbf{H} \Delta u + \mathbf{g}_p,\tag{6.34}$$

$$\nabla_{\Delta u} g_a(\Delta u) = \frac{\partial (\mathbf{A}_c^a \Delta u - \mathbf{b}_c)}{\partial \Delta u} = (\mathbf{A}_c^a)^T,\tag{6.35}$$

a dosadením do Lagrangeových podmienok dostaneme [101]

$$\mathbf{H} \Delta u + \mathbf{A}_c^{aT} \Delta \lambda = -\mathbf{g}_p,\tag{6.36}$$

$$\mathbf{A}_c^a \Delta u = \mathbf{0},\tag{6.37}$$

čo môžeme napísať v maticovej forme ako

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}_c^{aT} \\ \mathbf{A}_c^a & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\mathbf{g}_p \\ \mathbf{0} \end{bmatrix}.\tag{6.38}$$

Pre optimizačný problém, ktorý obsahuje aj obmedzenia v tvare rovnosti, môžeme použitím všeobecných Karush-Kuhn-Tuckerových podmienok (6.17) a zavedením multiplikátora  $\Delta\nu$ , ktorý nie je nutne kladný, napísať [22, 48, 101]

$$\mathbf{H} \Delta u + \mathbf{A}_c^{aT} \Delta \lambda + \mathbf{A}_e^T \Delta \nu = -\mathbf{g}_p,\tag{6.39}$$

$$\mathbf{A}_c^a \Delta u = \mathbf{0},\tag{6.40}$$

$$\mathbf{A}_e \Delta u = \mathbf{0},\tag{6.41}$$

čo môžeme opäť napísať aj v maticovej forme ako

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}_c^{aT} & \mathbf{A}_e^T \\ \mathbf{A}_c^a & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_e & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta \lambda \\ \Delta \nu \end{bmatrix} = \begin{bmatrix} -\mathbf{g}_p \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}.\tag{6.42}$$

### 6.4.5 QP algoritmus na báze aktívnych množín

Prvotnú stratégiu kvadratického programovania z predchádzajúcich strán môžeme ďalej upresniť takto [101]: Aby sme vyriešili problém kvadratického programovania, predpokladajme, že poznáme počiatočný odhad riešenia, ktorý je zlučiteľný s obmedzeniami  $\mathbf{u}_0$  a aktívnu množinu  $\mathbf{a}_0$ . Existujú rôzne vylepšené stratégie na nájdenie počiatočnej konfigurácie aktívnych množín, pričom samotný výpočet počiatočnej hodnoty riešenia nemusí byť vždy jednoduchý.

Pri danej hodnote  $\mathbf{u}_p$  vyriešme modifikovaný problém s uvažovaním rovnosti z Rov. (6.42), ktorý prislúcha aktuálnej aktívnej množine  $\mathbf{a}_p$ . Dostaneme vylepšenie riešenia v tvare  $\mathbf{u}_{p+1} = \mathbf{u}_p + \Delta \mathbf{u}$ . Zlučiteľnosť riešenia  $\mathbf{u}_{p+1}$  s pôvodnými obmedzeniami preveríme platnosťou  $\mathbf{A}_c \mathbf{u}_{p+1} \leq \mathbf{b}_c$ .

Ak je riešenie zlučiteľné s obmedzeniami, môžeme ho akceptovať ako zlepšenie, ktoré bude základom ďalšej iterácie a naviac môžeme uvažovať danú aktívnu množinu aj v ďalšej iterácii. Ak riešenie nie je zlučiteľné s obmedzeniami, vykonáme jednoduchý algoritmus hľadania po priamke v smere  $\Delta \mathbf{u}$  a nájdeme bod  $\mathbf{u}_{p+1} = \mathbf{u}_p + \alpha_l \Delta \mathbf{u}$  ( $0 < \alpha_l < 1$ ), v ktorom je obmedzenie porušené. Táto procedúra predstavuje jednoduché *hľadanie po čiare*<sup>26</sup>, kde zistujeme priesecník nového riešenia, t. j. úseku  $\{\mathbf{u}_p, \mathbf{u}_p + \Delta \mathbf{u}_p\}$ , a jednotlivých obmedzení. Toto môžeme vykonať na základe

$$\alpha = \min \left( \frac{\mathbf{b}_c^i - \mathbf{A}_c^i \mathbf{u}_p}{\mathbf{A}_c^i \Delta \mathbf{u}_p} \right), \quad (6.43)$$

kde v prípade viacerých riešení môžeme vybrať najmenšie číslo, lebo to prislúcha najbližšiemu porušenému obmedzeniu. Samozrejme, existujú aj iné stratégie pri hľadaní bodu porušenia obmedzení. Nové obmedzenie môžeme pridať k aktívnej množine, čím získame novú aktívnu množinu  $\mathbf{a}_{p+1}$ .

Ďalej výpočtom pôvodnej podmienky KKT na základe Rov. (6.14) overíme optimalitu riešenia  $\mathbf{u}_{p+1}$ . Ak  $\mathbf{u}_{p+1}$  nie je globálne optimum problému QP, zopakujeme danú procedúru; v opačnom prípade ide o globálne optimum problému QP a  $\mathbf{u}^* = \mathbf{u}_{p+1}$  je konečné riešenie, t. j. nemusíme ďalej iterovať algoritmus. Ešte efektívnejšie, a výpočtovo úspornejšie, je ukončiť algoritmus ak už nedosiahneme ďalšiu zmumu  $\Delta \mathbf{u}$  [115]. Algoritmus riešenia problému QP na báze metódy aktívnych množín je zhrnutý v Alg. 6.2.

**Príklad 6.14.** Nájdite obmedzené minimum kvadratickej funkcie s obmedzeniami z Pr. 6.1 pomocou QP metódy aktívnych množín z Alg. 6.2.

Začneme zadáním účelovej funkcie a obmedzení a pridáme hranice zobrazenia grafov. Vykreslíme vrstevnice funkcie a obmedzenia. Najskôr zavoláme aj funkciu `quadprog()` na overenie neskorších výsledkov a riešenie nakreslíme do grafu:

<sup>26</sup>angl. line search

**Algoritmus 6.2** QP: Metóda aktívnych množín

---

```

1: u0, a0, p = 0, riešenie ← 0           ▷ Zlučiteľný odhad riešenia a prislúchajúca
   množina
2: pokial riešenie ≠ 1 urob                   ▷ Hlavný cyklus
3:    $\Delta \mathbf{u}_p \leftarrow \text{Rov. (6.42)}$  pre  $i \in \mathbf{a}_p$            ▷ Riešenie prírastkovej KKT
4:    $\mathbf{u}_{p+1} \leftarrow \mathbf{u}_p + \Delta \mathbf{u}_p$                    ▷ Nový odhad riešenia
5:   ak  $\mathbf{A}_c \mathbf{u}_{p+1} \geq \mathbf{b}_c$  potom           ▷ Boli obmedzenia porušené?
6:      $\mathbf{a}_{p+1} \leftarrow \mathbf{a}_p + \text{nové porušenie}$            ▷ Nová aktívna množina
7:      $\alpha \leftarrow (\mathbf{b}_c^i - \mathbf{A}_c^i \mathbf{u}_p) / (\mathbf{A}_c^i \Delta \mathbf{u}_p)$  pre  $i \in \mathbf{a}_p$            ▷ Priesečník
8:      $\mathbf{u}_{p+1} \leftarrow \mathbf{u}_p + \alpha \Delta \mathbf{u}_p$            ▷ Korekcia na porušenom obmedzení
9:   inak                                         ▷ Obmedzenia neporušené
10:     $\begin{bmatrix} \mathbf{u}_p & \boldsymbol{\lambda} & \boldsymbol{\nu} \end{bmatrix}^T \leftarrow \text{Rov. (6.24)}$  pre  $i \in \mathbf{a}_p$            ▷ Riešenie KKT
11:    ak  $\lambda_i \geq 0, \nu_i \geq 0, i \in \mathbf{a}_p$  potom           ▷ Kladné multiplikátory
12:      riešenie ← 1                                     ▷ Máme riešenie
13:       $\mathbf{u}^* = \mathbf{u}_p$                                ▷ Riešenie
14:    koniec ak
15:  koniec ak
16: koniec pokial

```

---

Zdrojový kód 6.17. aktivnamnozina.m

```

H=[2 0; 0 18]; g=[-10 -54]'; % Povodny probl.
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]'; % Obmedzenia
hr=[-1, 6]; % Hranice graf.
vykreslitvrstevnice(H,g,hr,5) % Vrstev. funkcie
vykreslitobmedzenia(Ac,bc,hr) % Obmedzenia
legend('Kont.', 'u^{\circ}', 'g_i(u)', 'AutoUpdate', 'off')
axis([hr -1 5]) % Zobrazenie
uQP=quadprog(H,g,Ac,bc) % Kontrola
plot(uQP(1),uQP(2), 'k^') % Pociatocne rie.

```

Program pokračujeme počiatočnou inicializáciou hodnôt niektorých premenných. Premenná  $\text{riesenie}$  bude mať nulovú hodnotu, pretože na začiatku ešte nepoznáme riešenie QP. Ďalej zistíme počet optimalizačných premenných a obmedzení na základe rozmerov matice  $\mathbf{A}_c$ . Nasleduje voľba počiatočnej aktívnej množiny  $\mathbf{a}_0$  a počiatočného odhadu riešenia  $\mathbf{u}_0$ . Zvolíme si prázdnú množinu a odhad riešenia, ktorý je zlučiteľný s obmedzeniami a počiatočný odhad vykreslíme do ilustrácie riešenia:

```

riesenie=0;                                % Nemame riesenie
[no n]=size(Ac);                         % Pocet obm.
a=false(no,1);                            % Poc. aktivna mn
up=[1 1]';                                % Poc. riesenie
plot(up(1),up(2),'k+')                  % Pociatocne rie.

```

Pokračujeme cyklom na iterovanie algoritmu QP. Výpočet bude bežať až kým nedosiahneme riešenie. Najprv vyberieme riadky z  $A_c$  ktoré prislúchajú aktívnej množine  $a_p$  a dostaneme  $A_c^a$ . Vypočítame gradient v prírastkovom tvare  $g_p^T$  podľa Rov. (6.29) a zistíme počet práve aktívnych obmedzení. Potom vyriešime maticovú rovnosť (6.36) (všeobecne Rov. (6.38)) tak, že získame prírastok riešenia  $\Delta u_p$ :

```

while (riesenie~=1)                      % Kym nemame rie.
    AC=Ac(a==1,:);                      % Obmedzenia v ap
    gp=(g'+up'*H)';                     % Inkr. grad. gp
    [NO n]=size(AC);                    % Pocet obmedzeni
    W=[H AC'; AC zeros(NO)];          % Lava strana
    V=[-gp; zeros(NO,1)];              % Prava strana
    z=inv(W)*V;                        % du, lambda
    du=z(1:n);                         % Zmena v delta u
    upp=up+du;                         % Novy smer v up

```

Dostali sme nový prírastok riešenia, a tým nové riešenie  $u_{p+1} = u_p + \Delta u_p$ . Teraz musíme overiť, či je toto riešenie zlučiteľné z obmedzeniami. Ak je porušené niektoré z obmedzení pri  $u_{p+1}$ , vykonáme podmienku a nakreslíme nesprávne riešenie do grafu. Na základe Booleovskej logiky môžeme zistiť, ktoré bolo nové porušenie obmedzenia (exkluzívne alebo bitxor()), a zostaviť podľa toho novú aktívnu množinu  $a_{p+1}$  (alebo bitor()).

```

if (any(~(Ac*(up+du)<=bc))) ;      % Porusene obm.
    plot(upp(1),upp(2),'k+')          % Vykreslit up
    da=Ac*(up+du)>=bc;              % Porusene teraz
    dc=bitxor(a,da);                % z toho nove
    a=bitor(a,da);                  % Nova akt. mn.

```

Pokračujeme hľadaním riešenia až do porušenia obmedzenia. Jednoducho vypočítame priesčníky priamky a z toho dostaneme škálovanie  $\alpha$ , na základe ktorého zmeníme  $u_{p+1} = u_p + \alpha \Delta u_p$  a nakreslíme modifikované riešenie do grafu:

```

alpha=(bc(dc,:)-Ac(dc,:)*up)./(Ac(dc,:)*du); % Porusene obm.
up=up+min(alpha)*du;                            % Nove up.
plot(up(1),up(2),'k+')                        % Vykreslit up

```

V prípade, že posledné riešenie neporušilo obmedzenia, môžeme prekontrolovať, či sme dosiahli globálne minimum. Môžeme na to využiť Rov. (6.25) (všeobecne Rov. (6.24)) a dostaneme vektor riešenia aj Lagrangeových multiplikátorov  $\lambda$ :

```

else                                     % Zlucitelne ries
    up=up+du;                           % Neporusene obm.
    plot(up(1),up(2), 'k+');          % Vykreslit up
    AC=Ac(a==1,:); BC=bc(a==1);       % Obmedzenia v ap
    [NO n]=size(AC);                 % Pocet obm.
    WW=[H AC'; AC zeros(NO)];       % Lava strana
    VV=[-g; BC];                      % Prava strana
    zz=inv(WW)*VV;                  % u, lambda

```

Ak sú multiplikátory kladné, získali sme riešenie, a tým ukončíme aj cyklus:

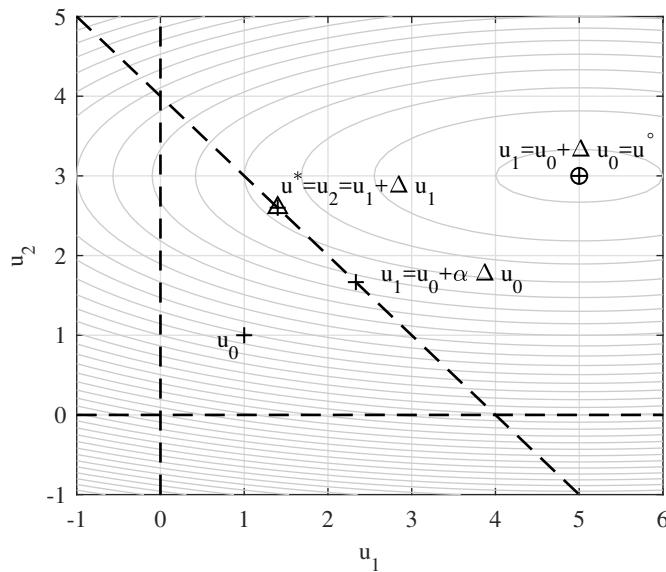
```

if min(z(n+1:end))>=0           % ak lambda>=0
    riesenie=1;                         % Mame riesenie
    u=up                                % Toto je ries.
    return                            % Vratenie z cyk.
end                                % Koniec if
end                                % Koniec if
end                                % Koniec while

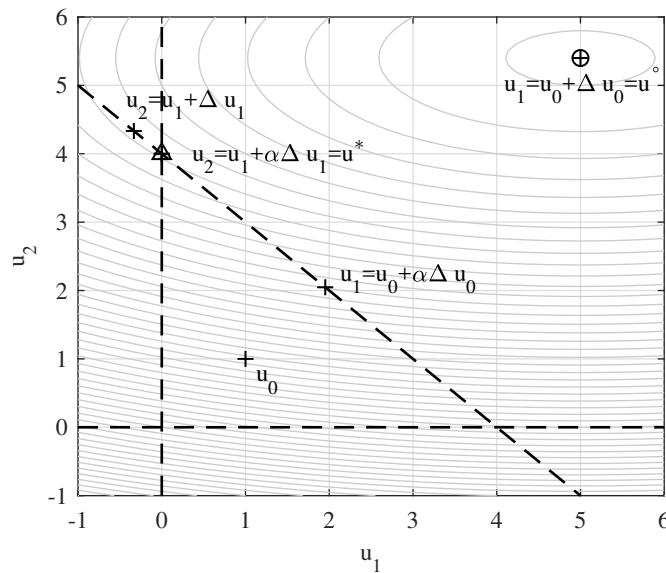
```

Po spustení programu na riešenie problému z Pr. 6.1 dostaneme priebeh riešenia ako na Obr. 6.22. Začíname z počiatočného odhadu  $\mathbf{u}_0$ , ktorý sa nachádza v zlučiteľnej oblasti. Keďže  $\mathbf{a}_0$  je prázdna množina, prírastok riešenia  $\Delta\mathbf{u}_0$  nás priviedie do neobmedzeného minima. Toto algoritmus vyhodnotí ako porušenie zlučiteľnosti obmedzení, preto začíname hľadať priesčník priamky  $\{\mathbf{u}_0, \mathbf{u}_0 + \Delta\mathbf{u}_0\}$ . Výpočet určí škálovaciu konštantu  $\alpha$ , ktorá vyznačí bod  $\mathbf{u}_1$ . Cyklus sa vykoná opäť a dostaneme prírastok riešenia  $\Delta\mathbf{u}_1$ , z čoho získame  $\mathbf{u}_2 = \mathbf{u}_1 + \Delta\mathbf{u}_1$ . Nové riešenie už je zlučiteľné s obmedzeniami a po testovaní optimality môžeme zastaviť algoritmus, pretože sme našli globálne riešenie.

Algoritmus môžeme otestovať zmenou príkladu, kde modifikujeme  $\mathbf{H}_{\{2,2\}} = 10$ . Obmedzenia sú nezmenené, ale neobmedzené minimum už bude „nad“ horným okrajom množiny obmedzení. Algoritmus teraz nájde správne riešenie, ktoré sa nachádza práve v tomto bode. Uvedený progres môžeme sledovať na Obr. 6.23. Začíname rovnako ako v prvom prípade: z počiatočného odhadu sa dostaneme do neobmedzeného minima, zistíme porušenie obmedzenia  $i = 3$ , na základe čoho vypočítame korekciu  $u_1$ . Nový prírastok  $\Delta\mathbf{u}_1$  nás ale opäť priviedie mimo prípustnej oblasti, a preto musíme znova zistiť, ktoré obmedzenie porušilo zlučiteľnosť riešenia. Teraz hľadáme priesčník obmedzení s priamkou  $\{\mathbf{u}_1, \mathbf{u}_1 + \Delta\mathbf{u}_1\}$ . Zistíme, že to bolo obmedzenie  $i = 1$ , čo nás po korekcií priviedie do riešenia  $\mathbf{u}_2$ .



Obr. 6.22. Iterácie načrtnutého QP algoritmu na báze metódy aktívnych množín.



Obr. 6.23. Metóda aktívnych množín po zmene Hessiánu.

□

Samozrejme, môžeme si predstaviť veľa situácií, keď by náš počiatočný náčrt algoritmu QP metódy aktívnej množiny nefungoval správne. Realizácia riešiča uvedená v Pr. 6.14 slúži najmä na ilustračné účely. Praktické algoritmy aktívnej množiny sú zložitejšie a poradia si aj s obtiažnejšími príkladmi. V našej realizácii nebolo ošetrených mnoho problémov, techniky zvládnutia ktorých sú bežne implementované v pokročilých algoritmoch – niektoré z nich platné aj pre iné metódy ako AS – napríklad:

- dočasné povolenie obmedzení,
- tolerancia nepresnosťí,
- riešenie tzv. degenerovaných problémov<sup>27</sup>,
- riešenie situácie keď je gradient aktívnej množiny lineárne závislý na riešení,
- ošetrenie zle podmienených KKT matíc,
- detekcia nekonvexnosti problému,
- detekcia nesymetrického Hessiánu,
- lepšie numerické riešenie KKT podmienok,
- využitie riedkej štruktúry matíc na urýchlenie výpočtov,
- a iné.

Kým pochopenie vnútornej logiky riešiča QP je potrebné aj na automatizačné účely a z pohľadu MPC – aspoň na ocenenie zložitosti problému – našim cieľom nie je vývoj univerzálnych riešičov. Ako to už bolo spomenuté aj v úvode, optimalizáciu, vrátane kvadratického programovania, môžeme považovať za samostatnú a vyspelú oblasť numerickej matematiky, preto medzi našimi ambíciami v tejto učebnici nie je vyvinúť dokonalú realizáciu príslušného algoritmu. Pre viac informácií pozri [115, 48, 22].

## 6.5 Metóda vnútorného bodu

Metóda vnútorného bodu má tú obrovskú výhodu pred ostatnými algoritmami QP, že zložitosť riešenia (časová, pamäťová) so zvyšujúcim sa počtom parametrov narastá v najhoršom prípade polynomiálne. Pri väčsine algoritmov, vrátane metódy aktívnej množiny, zložitosť riešenia rastie geometricky, teda v najhoršom prípade aj exponenciálne. Z tohto dôvodu môže byť QP algoritmus na báze metódy vnútorných bodov v prípade veľkých optimalizačných problémov s mnohými parametrami rádovo rýchlejší.

<sup>27</sup>angl. degenerate

Ako si ukážeme neskôr, hrubá a naivná realizácia algoritmu vnútorného bodu je pomerne jednoduchá, avšak jej výsledkom je nespoľahlivý riešič. Praktické algoritmy vnútorného bodu obsahujú komplexné triky a pokročilé techniky súčasnej aplikovanej matematiky. Kým pred pár desiatkami rokov neboli algoritmy vnútorného bodu príliš vyspelé, dnes už máme k dispozícii spoľahlivé a výpočtovo efektívne riešiče pre štandardné optimalizačné problémy.

O neustálom vývoji optimalizačných metód svedčia aj zmeny v implementácii rutiny `quadprog()`. Program je v podstate súborom riešičov, kde jedným zo spôsobov riešenia QP bola aj metóda aktívnych množín. Metóda však bola v roku 2016 odstránená a ako prvotný spôsob riešenia problému QP si rutina `quadprog()` zvolí práve metódu vnútorných bodov [139].

Základnou myšlienkou tzv. metódy *vnútorného bodu*<sup>28</sup> (IP) je transformácia pôvodného optimalizačného problému s obmedzeniami na podobný optimalizačný problém, ale bez obmedzení. Toto môžeme dosiahnuť napríklad tým, že si neobmedzenú kvadratickú funkciu rozšírimo o tzv. *bariérovú funkciu*<sup>29</sup>), ktorá v sebe zohľadňuje prítomnosť obmedzení.

Úlohou bariérovej funkcie je rozšíriť účelovú funkciu bez obmedzení o skalárnu funkciu, ktorej hodnota je malá, ak je riešenie v aktuálnom kroku práve zlučiteľné s obmedzeniami. Naopak, hodnota bariérovej funkcie bude rásť, ak je riešenie v danej iterácii blízko k obmedzeniam. Hodnota bariérovej funkcie by mala byť na obmedzeniach nekonečná.

$$\tilde{f}(\mathbf{u}) = f(\mathbf{u}) + \psi(\mathbf{u}), \quad (6.44)$$

kde  $f(\mathbf{u})$  je pôvodná účelová funkcia,  $\psi(\mathbf{u})$  je bariérová funkcia, ktorá je sformulovaná na základe obmedzení, a  $\tilde{f}(\mathbf{u})$  je nová funkcia bez obmedzení.

Kvadratickú funkciu rozšírenú o bariérovú funkciu môžeme matematicky zápisť v tvare [101, 27]

$$\tilde{f}(\mathbf{u}) = \mu \left[ \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} + \psi(\mathbf{u}) \right], \quad (6.45)$$

kde  $\mu$  je skalárna premenná, ktorú si určuje samotný algoritmus počas svojich iiterácií. Všimnime si, že pôvodná kvadratická funkcia zostáva nezmenená. Účinok obmedzení je tak zahrnutý iba vo funkciu  $\psi(\mathbf{u})$ .

Ak máme obmedzenia v tvare  $\mathbf{A}_c^i \mathbf{u} \leq \mathbf{b}_c^i$ , potom typickou bariérovou funkciou môže byť napríklad tzv. logaritmická bariéra v tvare

$$\psi(\mathbf{u}) = \sum_{i=1}^{n_o} \left[ -\log(\mathbf{b}_c^i - \mathbf{A}_c^i \mathbf{u}) \right], \quad (6.46)$$

<sup>28</sup>angl. interior point

<sup>29</sup>angl. barrier function

ale existujú aj alternatívne formulácie.

Kombináciou pôvodnej funkcie a bariérovej funkcie dostaneme nový, neohraňčený optimalizačný problém v tvare

$$\mathbf{u}_{p+1} = \arg \min_{\mathbf{u}} \left\{ \mu_p \left[ \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} \right] + \sum_{i=1}^{n_o} \left[ -\log(\mathbf{b}_c^i - \mathbf{A}_c^i \mathbf{u}) \right] \right\} \quad (6.47)$$

pre dané  $\mu_p$ . Minimalizáciu bez obmedzení pritom vždy naštartujeme zlučiteľným odhadom z predchádzajúceho kroku  $\mathbf{u}_p$ . Ak minimalizáciu začíname v zlučiteľnej množine, bariérová funkcia zabezpečí zlučiteľnosť ďalších riešení, pretože v blízkosti obmedzení nadobúda nekonečnú hodnotu. Samotný názov metódy je odvodený z tohto konceptu: riešenie vždy zostáva vnútri danej množiny obmedzení.

Hrubú metódu riešenia QP na základe algoritmu vnútorného bodu môžeme zhrnúť nasledovne [101]: Začneme počiatočným odhadom riešenia  $\mathbf{u}_0$ , ktorý je zlučiteľný. Častou stratégiou je začať hľadanie v strede množiny zlučiteľných obmedzení. V takom prípade hovoríme, že  $\mathbf{u}_0$  je v tzv. *analytickom strede*<sup>30</sup> obmedzení. Musíme tiež určiť aj nejakú počiatočnú hodnotu skaláru  $\mu_0$ .

Vyriešme optimalizačný problém (6.47). Na riešenie tohto neobmedzeného optimalizačného problému máme na výber niekoľko rôznych metód. Riešenie inicializujeme pre  $\mu_p$  a pomocou počiatočného odhadu z predchádzajúceho kroku, t. j.  $\mathbf{u}_p$ , dostaneme nový, vylepšený odhad.

Zmeňme teraz hodnotu  $\mu_{p+1}$  podľa určitej zvolenej stratégie. V najjednoduchšom prípade môžeme navyšovať túto hodnotu logaritmicky. Riešenie Rov. (6.47) preto zopakujeme s novými  $\mu_{p+1}$  a  $\mathbf{u}_{p+1}$  až kým proces nezastavíme vybranou stratégiou ukončenia. Z počiatočného odhadu budeme pomaly konvergovať cez čiastočné riešenia až do optima. Trajektória čiastočných riešení sa nazýva *vnútorná trajektória*<sup>31</sup>. V najjednoduchšom prípade vykonáme konečný počet iterácií, pričom posledný výsledok považujeme za správny. Hrubý náčrt algoritmu metódy vnútorného bodu je zhrnutý aj v Alg. 6.3.

**Príklad 6.15.** Vypočítajte riešenie QP problému z Pr. 6.1 pomocou metódy vnútorného bodu na základe Alg. 6.3. Za podmienku ukončenia výpočtu zvoľte pevný počet krokov a za stratégiu určovania, t. j. navyšovania parametra  $\mu$  logaritmické rozdelenie 20 čísel v rozmedzí  $10^{-4}$  až  $10^2$ . Ako počiatočný odhad riešenia si zvoľte  $\mathbf{u}_0 = [4/3 \ 4/3]^T$ .

Najprv si vytvoríme samotnú účelovú funkciu. Pre tento príklad môžeme bariérovú funkciu napísat v konkrétnom tvare; samozrejme vo všeobecnosti máme  $n_o$  takýchto obmedzení.

<sup>30</sup>angl. analytic center

<sup>31</sup>angl. central path

**Algoritmus 6.3** QP: Metóda vnútorného bodu (Hrubý náčrt.)

---

1: <b>u</b> <sub>0</sub> , $\mu_0$ , $p \leftarrow 0$ , riešenie $\leftarrow 0$	▷ Zlučiteľný odhad riešenia
2: <b>pokial</b> riešenie $\neq 1$ <b>urob</b>	▷ Hlavný cyklus
3: <b>u</b> <sub><math>p+1</math> <math>\leftarrow</math> Rov. (6.47), pre <b>u</b><sub><math>p</math>, <math>\mu_p</math></sub></sub>	▷ Minimalizácia funkcie
4: $\mu_{p+1}$	▷ Stratégia navýšenia skalárnej hodnoty
5: <b>ak</b> ukončenie <b>potom</b>	▷ Podmienka na ukončenie
6: <b>u</b> <sup>*</sup> $\leftarrow$ <b>u</b> <sub><math>p+1</math></sub>	▷ Konečné riešenie
7:     riešenie $\leftarrow 1$	▷ Zastavenie výpočtu
8: <b>koniec ak</b>	
9: <b>koniec pokial</b>	

---

Zdrojový kód 6.18. objfun.m

```
function f=objfun (x, mu, H, g, Ac, bc)
f=(mu* ( (1/2)*x'*H*x+g'*x) +(-log(bc(1)-Ac(1,:)*x)-log(bc(2)-Ac(2,:)*x)-log(bc(3)-Ac(3,:)*x)) ;
```

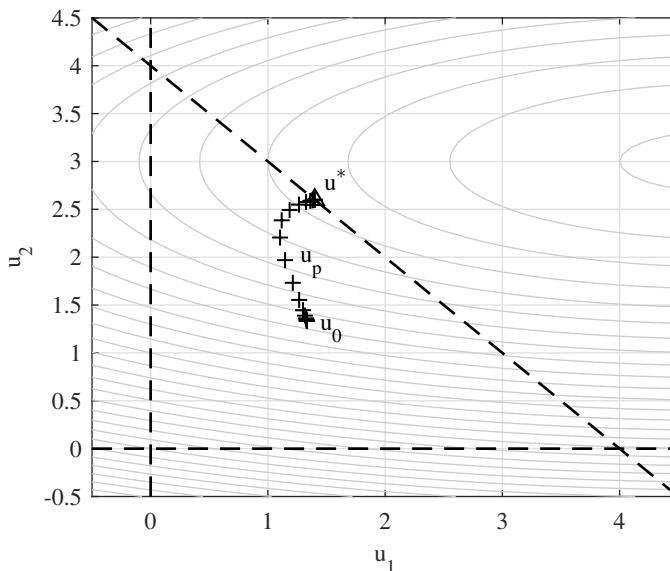
Nasleduje definovanie problému a grafické zobrazenie vrstevníc funkcie a obmedzení. Hneď na začiatku môžeme tiež vypočítať správny výsledok pomocou riešiča quadprog():

Zdrojový kód 6.19. vnutornybod.m

```
H=[2 0; 0 18]; g=[-10 -54]'; % Povodny prob.
Ac=[-1 0; 0 -1; 1 1]; bc=[0 0 4]'; % Obmedzenia
hr=[-0.5, 4.5]; % Hranice graf.
vykreslitvrstevnice(H,g,hr,5) % Vrstev. funk.
vykreslitobmedzenia(Ac,bc,hr) % Obmedzenia
legend('Vrst.', 'u^{\circ}', 'g_i(u)', 'AutoUpdate', 'off')
axis([hr hr]) % Zobrazenie
uQP=quadprog(H,g,Ac,bc) % Kontrola
plot(uQP(1),uQP(2), 'k^') % Nakreslit uQP
```

Pokračujeme inicializáciou počiatočného odhadu, ktorý si vykreslíme do grafického zobrazenia. Na definovanie parametra  $\mu$  použijeme jednoduchú techniku logaritmického rozdelenia jeho rozsahu:

```
up=[4/3 4/3]'; % Pociatočne up
plot(up(1),up(2), 'k+') % Nakreslit up
mu=logspace(-4,2,20); % Logaritmickie
```



Obr. 6.24. Iterácie QP algoritmu na báze metódy vnútorného bodu.

Jednoduchou podmienkou ukončenia algoritmu bude dosiahnutie konečného počtu iterácií. Najprv si dosadíme konkrétné hodnoty matík do účelovej funkcie, potom nájdeme minimum napr. pomocou funkcie `fminsearch()` a čiastočné výsledky zakreslíme do grafu:

```

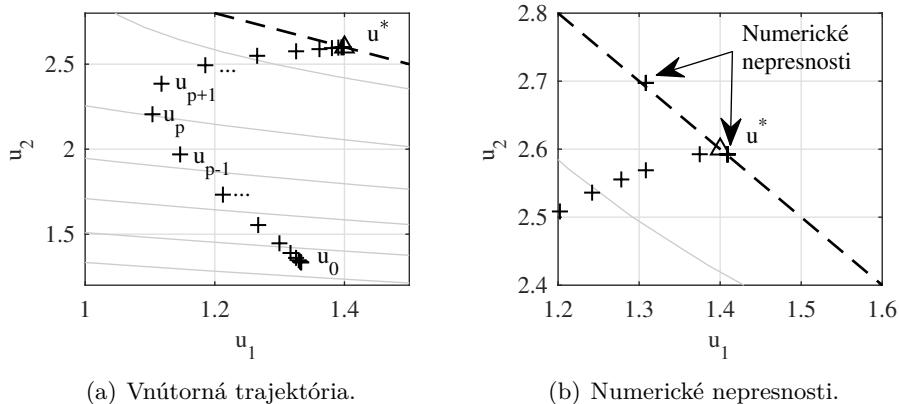
for i=1:length(mu)
    f = @(up) objfun(up,mu(i),H,g,Ac,bc); % Obj. funk.
    [up fup]=fminsearch(f,up); % Minimaliz.
    plot(up(1),up(2),'k+'); % Nakresliť up
end

```

Po spustení programu dostaneme grafické znázornenie krokov algoritmu riešiča QP na základe metódy vnútorného bodu (Obr. 6.24). Vidíme, že jednotlivé čiastkové riešenia sa vždy nachádzajú v zlučiteľnej oblasti.

Postupné riešenie problému vytvára spomínaný už vnútorný trajektóriu, ktorá sa začína v analytickom strede a končí v riešení (Obr. 6.25(a)). Na prvý pohľad sa zdá, že sme dosiahli správne riešenie. Po priblížení grafu však vidíme, že riešenie vplyvom numerických nepresností nie je dokonalé. Zmenou spôsobu rozdelenia  $\mu$  môžeme dostať aj značne nepresné výsledky, ako to je znázornené na Obr. 6.25(b).

Nepresnosť je tu už tak značná, že dostaneme riešenia, ktoré sú nezlučiteľné



Obr. 6.25. Vnútorná trajekória riešenia a možné numerické nepresnosti.

s obmedzeniami. Jeden z dôvodov prečo sú komerčné riešiče oveľa zložitejšie ako naša verzia IP je, že si musia vedieť poradiť aj s podobnými prípadmi numerickej nepresnosti [101].

□

Podobne ako pri metóde aktívnych množín, ani tu nie je našou ambíciou predstaviť si metódu vnútorného bodu úplne podrobne. Programovanie riešiča je zložitá úloha, a práve preto existuje niekoľko komerčných produktov na optimalizáciu kvadratickej funkcie s lineárnymi obmedzeniami.

Naša jednoduchá interpretácia algoritmu QP slúži hlavne na ilustračné účely a preto sa neodporúča vývoj vlastného QP riešiča na základe týchto úvah. Kým prvé IP QP riešiče sledovali túto logiku, dnešné implementácie napríklad nepotrebujú na začiatku zlučiteľný odhad, do konca ani neprehľadávajú postupne vnútorný zlučiteľný priestor, ale dopracujú sa k zlučiteľnému riešeniu pred koncom iterovania. Moderné implementácie IP sú kombinované aj s riešením KKT podmienok podobne ako pri AS metóde. Viac informácií možno nájsť v [113, 121, 126, 115, 22].

### Ako ďalej?

Ako už vieme, problém kvadratického programovania je vo všeobecnom tvare daný ako (6.2). Takiež sme už zistili, že riešenie QP nie je triviálne, ale v tejto kapitole sme sa naučili, že je zvládnuteľné pomocou numerických riešičov ako je napríklad `quadprog()`. Poznáme aj problém MPC, ktorý sa značne podobá na QP, ale pri

prvom člene kvadratickej funkcie chýba skalár  $1/2$ , a pri druhom člene účelovej funkcie MPC máme naopak číslo 2.

Ako si ukážeme v nasledujúcej kapitole, problém MPC s obmedzeniami nepredstavuje nič iné ako problém kvadratického programovania, ktorý môžeme vyriešiť rovnako ako akýkoľvek iný všeobecný problém QP. V Kap. 7 ho dokonca budeme riešiť v každej vzorke – čím skompletizujeme algoritmus prediktívneho riadenia s obmedzeniami...



## Kapitola 7

# Prediktívne riadenie s obmedzeniami

V tejto kapitole si nepredstavíme žiadne zásadne nové teoretické pojmy, ale zjednotíme si poznatky z predchádzajúcich kapitol a názorne demonštrujeme prediktívne riadenie s obmedzeniami ako ucelený algoritmus spätnoväzobného riadenia. Pripomeňme si tiež, že sa pritom sústredíme na lineárne časovo nepremenlivé systémy a kvadratické kritérium riadenia. Model dynamického systému využijeme na výpočet predikčných matíc, pomocou ktorých formulujeme účelovú funkciu. Obmedzenia zlúčime do obmedzenia v tvare nerovnosti a zužitkujeme aj techniku kvadratického programovania na výpočet optimálneho akčného zásahu. Inými slovami, spojíme poznatky z Kap. 5 ohľadom formulácie úlohy MPC s obmedzeniami a z Kap. 6 na matematické riešenie výsledného optimalizačného problému.

Prvým cieľom tejto kapitoly teda bude vyšetriť vzťah medzi všeobecnou úlohou kvadratického programovania a optimalizačnou úlohou, ktorú predstavuje problém MPC s obmedzeniami. Potom môžeme vypočítať riešenie úlohy pre jednu časovú vzorku a potom proces zopakovať v každom kroku spätnoväzobného riadenia.

Pri tvorbe účelovej funkcie sme predpokladali, že systém chceme priviesť do daného rovnovážneho stavu, ktorý budeme uvažovať ako nulový požadovaný stav. Ďalej si tiež ukážeme, akým spôsobom môžeme pomocou MPC sledovať predpísanú referenčnú trajektóriu žiadaneho výstupu. V niektorých praktických aplikáciach je výhodné použiť prírastkovú formu akčných zásahov, preto si predstavíme aj transformáciu predikčného modelu na tzv. prírastkový tvar.

Na záver kapitoly si demonštrujeme aj spôsob simulácie prediktívneho riadenia v prostredí MATLAB–Simulink. Kým jedným z poslanií Simulinku je umožniť preklad algoritmov na iné výpočtové platformy, dozvieme sa, že v prípade zabudovanej rutiny kvadratického programovania takýto preklad nie je možný.

## 7.1 Prediktívne riadenie a kvadratické programovanie

Na začiatok si porovnajme problém (5.61), ktorý predstavuje optimalizačnú úlohu v prediktívnom riadení s obmedzeniami, teda hľadanie

$$\vec{\mathbf{u}}_k^* = \arg \min_{\vec{\mathbf{u}}_k} \left( \vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + 2 \mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k \right),$$

vzhľadom na podmienku:

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_c,$$

so všeobecnou formuláciou problému kvadratického programovania (6.2) s obmedzeniami v tvare nerovnosti

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} \left( \frac{1}{2} \mathbf{u}^T \mathbf{H} \mathbf{u} + \mathbf{g}^T \mathbf{u} \right),$$

vzhľadom na podmienku:

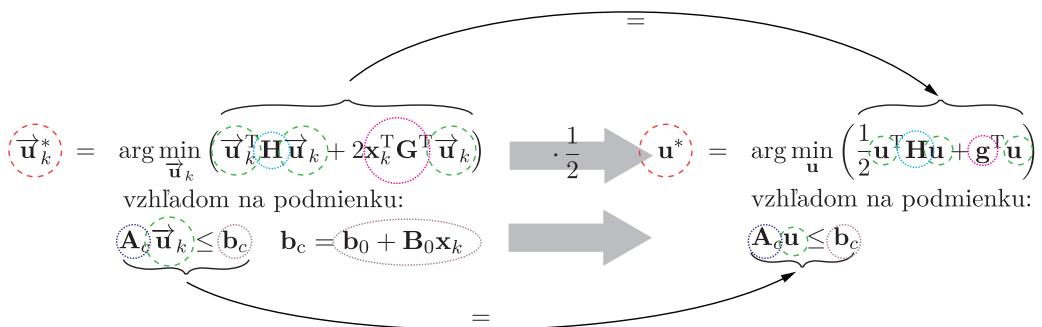
$$\mathbf{A}_c \mathbf{u} \leq \mathbf{b}_c,$$

kde premenné  $\mathbf{u}$ ,  $\mathbf{u}^*$ ,  $\mathbf{H}$ ,  $\mathbf{g}$ ,  $\mathbf{A}_c$ ,  $\mathbf{b}_c$  majú všeobecný matematický význam. Nezabudnime, že v prípade úlohy prediktívneho riadenia uvažujeme všeobecný tvar obmedzení  $\mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k$  podľa Rov. (5.52), ktorý sa po násobení s aktuálnym stavom  $\mathbf{x}_k$  v danej vzorke  $k$  zredukuje na stĺpcový vektor  $\mathbf{b}_c$ .

Podobnosť optimalizačných úloh je zrejmá hneď na prvý pohľad. Predpokladajme teraz, že za všeobecnú optimalizačnú premennú  $\mathbf{u}$  si dosadíme postupnosť budúci vstupov  $\vec{\mathbf{u}}_k$  vo vzorke  $k$ , a teda hľadáme  $\vec{\mathbf{u}}_k^*$  ako riešenie daného QP. V prípade, že si obmedzenia zhrnieme do množiny obmedzení v tvare nerovnosti, obmedzenia pre MPC budú formulované kompatibilne so všeobecným problémom QP, za predpokladu prepočtu vektora  $\mathbf{b}_c$  v danej vzorke.

Pozrime sa teraz na samotnú minimalizovanú funkciu. Vplyv stavov a vstupov na horizonte vyjadrujeme pomocou účelovej funkcie, z ktorej po dosadení predikčného modelu získame Hessián  $\mathbf{H}$ , ktorý dalej priamo pužijeme vo všeobecnej formulácii problému QP. Podobne získame aj gradient  $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$ , resp.  $\mathbf{g} = \mathbf{G} \mathbf{x}_k$ , ktorý je taktiež súčasťou pôvodnej formulácie QP.

Zostáva už iba jeden rozdiel a konkrétnie ten, že prvý člen všeobecnej formulácie QP je delený skalárom dva, kým druhý člen pri MPC je násobený skalárom dva. Ak by sme predelili celú účelovú funkciu MPC číslom dva, dostali by sme identickú formuláciu akú má uvedený QP. Toto môžeme bez problémov urobiť, pričom nijak nezmeníme výsledok minimalizácie,  $\vec{\mathbf{u}}_k^*$ , ale iba predelíme hodnotu účelovej funkcie, čo je z pohľadu riešenia problému nepodstatné. Logika je tu podobná ako v diskusii v Kap. 5.1.1, kde sme argumentovali tým, že vyniechanie posledného, konštantného člena účelovej funkcie nemá vplyv na riešenie daného



Obr. 7.1. Formulácia optimalizačného problému MPC je ekvivalentná so všeobecnenou formuláciou QP.

problému. Ekvivalentnosť optimalizačnej úlohy MPC a problému QP je ilustrovaná aj na Obr. 7.1.

Online optimalizačný problém prediktívneho riadenia môžeme podľa potreby prepísať na nasledujúci tvar

$$\vec{u}_k^* = \arg \min_{\vec{u}_k} \left( \frac{1}{2} \vec{u}_k^T \mathbf{H} \vec{u}_k + \underbrace{\mathbf{x}_k^T \mathbf{G}^T}_{\mathbf{g}^T} \vec{u}_k \right), \quad (7.1)$$

vzhľadom na podmienku:

$$\mathbf{A}_c \vec{u}_k \leq \mathbf{b}_c,$$

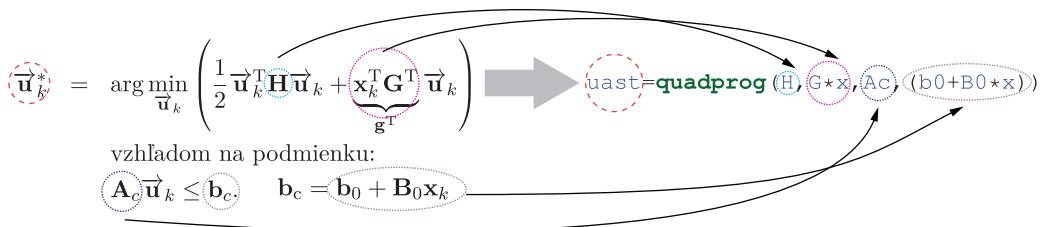
ktorý je ekvivalentný so všeobecnou formuláciou problému kvadratického programovania.



## 7.2 Výpočet optimálneho vektora vstupov

Predpokladajme teraz, že optimálne riešenie MPC problému (7.1) v danej vzorke  $k$  získame pomocou riešiča `quadprog()`. Pre daný LTI model máme k dispozícii Hessián  $\mathbf{H}$ , kym gradient závisí vždy od aktuálneho stavu riadeného systému, teda  $\mathbf{g}^T = \mathbf{x}_k^T \mathbf{G}^T$ . Obmedzenia máme kompaktne vyjadrené pomocou matice  $\mathbf{A}_c$  a vektora  $\mathbf{b}_c$ , kde  $\mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k$ .

Na základe predchádzajúcej diskusie môžeme všeobecný problém kvadratického programovania a optimalizačnú úlohu prediktívneho riadenia považovať za



Obr. 7.2. Riešenie optimalizačnej úlohy MPC pomocou riešiča „quadprog“.

rovnocenné. Potom, ak napríklad riešič `quadprog()` uvažuje nasledujúcu syntax

`u=quadprog(H, g, Ac, bc)`

dosadením za  $H \leftarrow \mathbf{H}$ ,  $g \leftarrow \mathbf{g}$ , kde  $\mathbf{g} = \mathbf{G}\mathbf{x}_k$ , získame účelovú funkciu a náhradou  $Ac \leftarrow \mathbf{A}_c$  a  $bc \leftarrow \mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0\mathbf{x}_k$  dostaneme obmedzenia. Optimálnu postupnosť vstupov v danom časovom kroku vypočítame ako

`uast=quadprog(H, G*x, Ac, (b0+B0*x))`

kde za  $x$  v kažkom kroku  $k$  dosadíme aktuálny meraný, resp. odhadnutý stav  $\mathbf{x}_k$ , resp.  $\hat{\mathbf{x}}_k$ ; pozri Obr. 7.2).

**Príklad 7.1.** Vypočítajte postupnosť optimálnych vstupov  $\vec{u}_k$  pre aktívne riadenie podvozku autobusu z Pr. 5.8, ak predpokladáme aktuálne odhadnutý stav  $\hat{\mathbf{x}}_k = [0.1 \ 0 \ 0 \ 0]^T$  a symetrické obmedzenie na akčný zásah  $-0.1 \leq u_k \leq 0.1$ .

Začneme s definíciou príkladu analogicky ako v Pr. 5.8. Definujeme si počet stavov a vstupov, deklarujeme váhovanie, predikčný horizont a obmedzenia:

Zdrojový kód 7.1. MPCjedenkrok.m

```

load bus.mat % Nacitaj model
A=bus.a; B=bus.b(:,2); C=bus.c; % Model autobusu
nx=length(A); % Pocet stavov
Q=C'*C; R=10; np=20; % Vahy a horizont
uh=0.01; ul=-uh; % Obmedzenia vstupu

```

Potom využijeme funkciu na výpočet LQ zosilnenia a váhovania koncového stavu, vypočítame zložky účelovej funkcie a zostavíme matice obmedzení vstupu:

```

[K, P]=iterdlqr(A, B, Q, R, 100); % Koncove vahovanie
[H, G]=ucelovafunkcia(A, B, np, Q, R, P); % Ucelova funkcia
[Ac, bc]=obmedzenia(ul, uh, np); % Obmedzenia na u

```

Ďalej deklarujeme odhadnutý stav, zavoláme riešič a vypočítame aj LQ-optimálny akčný zásah:

```
x=[0.1 0 0 0]'; % Poc. stav
uopt=quadprog(H,G*x,Ac,bc); % MPC
ulq=-K*x; % LQ
```

Po spustení programu riešič začína varovným hlásením o nesymetricite Hessiánu. Toto varovanie môžeme ignorovať, keďže náš Hessián je nesymetrický iba z numerického, nie logického hľadiska. Môžeme to prekontrolovať tak, že do príkazového riadku napíšeme

```
max(max(abs(H-H'))))
```

čo nájde najväčší prvok z absolútnej hodnoty rozdielu matice a jej transpozície. Tento je v ideálnom prípade nulový, v našom prípade kvôli numerickej nepresnosti nájdeme chybu  $< 1E-15$ , čo je zanedbateľné. Napriek tomu riešič pre istotu prepočíta Hessián ako  $\mathbf{H} = (\mathbf{H} + \mathbf{H}^T)/2$ .

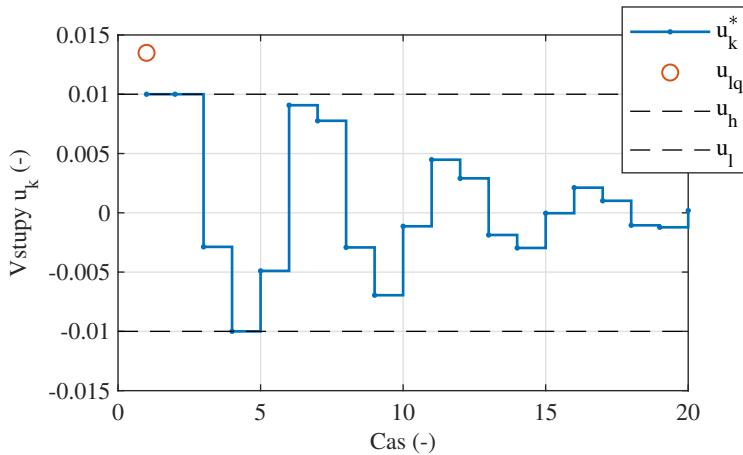
Kedže sme nevypli verbálne zhodnotenie výsledkov, riešič oznamí úspešné nájdenie obmedzeného minima. Pre optimálnu postupnosť vstupov po spustení programu dostaneme pre daný stav výsledok

```
uopt =
0.0100
0.0100
-0.0029
-0.0100
-0.0049
0.0091
[...]
```

pričom LQ-optimálny akčný zásah sa vypočíta ako  $ulq=0.0135$ . Poznamenajme, že pri LQ riadiacom zákone neuvažujeme procesné obmedzenia, zatiaľ čo výsledkom riešenia MPC problému sú akčné zásahy, ktoré rešpektujú dané maximálne a minimálne dovolené ohraničenia vstupu.

Vypočítanú postupnosť optimálnych vstupov si znázornime aj graficky. Začneme vykreslením postupnosti MPC a LQ optimálneho akčného zásahu. Obmedzenia si znázorníme pomocou zvislých čiar a označíme si osi a jednotlivé časti grafu:

```
stairs(uopt,'.-')
hold on
stairs(ulq,'o') % MPC vstupy
% Podrž graf
% LQ vstup
```



Obr. 7.3. Optimálna postupnosť budúcich vstupov na riadenie podvozku autobusu v čase  $k$ .

```

line([0,np],[uh uh],'Color','k','LineStyle','--') % uh
line([0,np],[ul ul],'Color','k','LineStyle','--') % ul
xlabel('Cas (-)') % Os X
ylabel('Vstupy u_k (-)') % Os Y
legend('u_k^{*}', 'u_{lq}', 'u_h', 'u_l') % Legenda
grid on % Mriezka

```

Výsledný graf je znázornený na Obr. 7.3. Treba si uvedomiť, že obrázok neznázorňuje výsledky spätnoväzobného riadenia, vidíme iba výsledok riešenia optimálizačného problému MPC v aktuálnom časovom kroku  $k$  pre stav  $\mathbf{x}_k$ . Postupnosť budúcich vstupov získaná pomocou MPC predstavuje optimálne, t. j. najlepšie možné riešenie pri absencii vonkajších porúch a predpoklade dokonalého modelu – presne tak ako to je definované v procese predikcie.

Tvrdenie, že by sme dostali rovnaký výsledok, ak by sme orezali vstup získaný pomocou LQ riadenia, nie je vo všeobecnosti pravdivé, pretože MPC rieši problém s už zahrnutými obmedzeniami, a to na celom horizonte predikcie. MPC tak „vidí“, že akčné zásahy je potrebné rátať tak, aby zároveň neboli porušené dané obmedzenia. □

Ak dostaneme varovania o nesymetricite Hessiánu ako v predchádzajúcom príklade, môžeme maticu  $\mathbf{H}$  vopred prepočítať na symetrickú, aby sme si odstránili numerické chyby.

Riešič budeme ďalej volať v každom časovom kroku  $k$ , a preto je výpis stavu riešiča zbytočný. Keď riešič quadprog() voláme pre danú simuláciu niekoľko stov až tisíckrát, aj výpis stavu riešenia môže značne spomaliť našu simuláciu. Práve preto je užitočné tento slovný výpis vypnúť pomocou funkcie optimoptions(), ktorá uloží rôzne nastavenia riešiča do zvolenej premennej. Na vypnutie výpisu statusu riešiča môžeme zavolať túto funkciu ako

```
o=optimoptions('quadprog','Display','none');
```

a funkciu quadprog() potom zavoláme aj so vstupným argumentom o.

**Príklad 7.2.** Zopakujte predchádzajúci príklad, ale pred samotným zavolaním riešiča quadprog() transformujte Hessián na symetrickú formu a vypnite slovné hodnotenie riešenia.

Postupujeme identicky ako pri Pr. 7.1. Pred zavolaním riešiča Hessián prepočítame na symetrickú formu a nastavíme vypnutie výpisu statusu riešiča pomocou funkcie optimoptions(). Riešič musíme teraz pustiť tak, že nevyužité vstupné argumenty nahradíme prázdnym výrazom, [], a ako posledný argument zadáme premennú, ktorá obsahuje nastavenia riešiča.

Zdrojový kód 7.2. vypnutVerbose.m

```
H=(H+H')/2; % Symetr. H
o=optimoptions('quadprog','Display','none'); % Vyp. vypis
uopt=quadprog(H,G*x,Ac,bc,[],[],[],[],[],o) % MPC - u(k)
```

Po spustení programu už nevidíme varovanie a riešič nezobrazí slovné hodnotenie úspešnosti riešenia.

□

## 7.3 MPC v uzavretej slučke

V predchádzajúcej diskusii sme si ukázali spôsob riešenia problému MPC v danom, konkrétnom časovom kroku. Samozrejme, optimálna sekvencia akčných zásahov,  $\vec{u}_k^*$ , by bola použiteľná iba v prípade riadenia v absencii vonkajších porúch, čo je nereálne. Potrebujeme preto uvažovať stratégiju spätnoväzobného riadenia.

Ak v čase  $k$  máme k dispozícii  $\vec{u}_k^*$ , riešenie je veľmi jednoduché:



Prediktívne riadenie v uzavretej slučke využije iba prvý člen  $\mathbf{u}_k^* \in \mathbb{R}^{n_u \times 1}$  vypočítanej optimálnej postupnosti  $\vec{\mathbf{u}}_k^* \in \mathbb{R}^{n_u \cdot n_p \times 1}$ , a teda akčný zásah v čase  $k$  je daný ako

$$\mathbf{u}_k^* = \mathcal{I} \vec{\mathbf{u}}_k^*, \quad (7.2)$$

kde  $\mathcal{I} = [\mathbf{I} \ \mathbf{0} \dots \mathbf{0}]^{n_u \times n_u \cdot n_p}$ ,  $\mathbf{I} \in \mathbb{R}^{n_u \times 1}$ .

V programovej realizácii samozrejme nemusíme vektor  $\vec{\mathbf{u}}_k^*$  nutne násobiť s  $\mathcal{I}$ , stačí využívať dané nástroje programovacieho prostredia. Napríklad, ak výsledný vektor predikovaných akčných zásahov označíme premennou  $\mathbf{U}$  a premenná  $\mathbf{nu}$  označuje počet vstupov, potom v MATLABe postačuje odkázať sa na aktuálny akčný zásah ako  $\mathbf{U}(1:\mathbf{nu})$ .

V zmysle konceptu pohyblivého horizontu uvedený proces v ďalšom časovom kroku znova opakujeme pre nové meranie alebo odhad stavu.

Algoritmus prediktívneho riadenia s obmedzeniami podobne ako pri absencii obmedzení delíme na dve rozdielne časti: *offline*<sup>1</sup> a *online*<sup>2</sup> algoritmus. V prípade offline algoritmu výpočet koncového váhovania a účelovej funkcie musíme rozšíriť o vytvorenie matíc  $\mathbf{A}_c$  a  $\mathbf{b}_c$  formulujúcich obmedzenia v tvare nerovnosti. Procedúra offline výpočtu uvedených zložiek MPC s obmedzeniami je zhrnutá v Alg. 7.1.

### Algoritmus 7.1 MPC s obmedzeniami: offline

- |   |                             |
|---|-----------------------------|
| 1: $\mathbf{P} \leftarrow$ Rov. (4.50)                              | ▷ váhovanie koncového stavu |
| 2: $\mathbf{H}, \mathbf{G} \leftarrow$ Rov. (4.40), Rov. (4.42)     | ▷ účelová funkcia           |
| 3: $\mathbf{A}_c, \mathbf{b}_x \leftarrow$ Rov. (5.34); Rov. (5.50) | ▷ obmedzenia                |

Pri spätnoväzbovom riadenie uvažujeme online časť, kde výpočet akčných zásahov na základe pevného maticového zosilnenia nahradíme riešením problému kvadratického programovania, čo predstavuje online časť MPC algoritmu, ktorá je zhrnutá v Alg. 7.2.

**Príklad 7.3.** Vykonajte simuláciu spätnoväzbového riadenia výšky podvozku autobusu z Pr. 7.1, ak po naštartovaní vozidla vodič prejde z vyššieho parkoviska na o 5 cm nižšiu vozovku. Uvažujme váhovanie vstupu  $\mathbf{R} = 0.01$  a predikčný horizont  $n_p = 10$  krokov.

<sup>1</sup>angl. nepriamy

<sup>2</sup>angl. priamy

**Algoritmus 7.2** MPC riadenie s obmedzeniami: online

- 
- |  |                              |
|--|------------------------------|
| 1: <b>pokiaľ 1 urob</b>                                    | ▷ v každej vzorke, nekonečne |
| 2: $\mathbf{x}_k \leftarrow \hat{\mathbf{x}}_k$            | ▷ odhad, meranie             |
| 3: $\vec{\mathbf{u}}_k \leftarrow$ problém (7.1)           | ▷ riešenie QP                |
| 4: $\mathbf{u}_k \leftarrow \mathcal{I}\vec{\mathbf{u}}_k$ | ▷ prvý prvok na akčné členy  |
| 5: <b>koniec pokiaľ</b>                                    |                              |
- 

Príklad začíname podobne ako Pr. 7.1, formulujeme si model systému a MPC problém. Zadanie príkladu možno interpretovať tak, že zvyšujeme komfort cestujúcich prostredníctvom dynamického nastavovania výšky podvozku. Prechod z výšky 5 cm na vozovku potom môžeme interpretovať ako riadenie z počiatočného stavu  $\mathbf{x}_0 = [0.05 \ 0 \ 0 \ 0]^T$  do rovnovážneho stavu.

Pokračujeme definíciou dĺžky simulácie a počiatočného stavu. Simulačná slučka zahŕňa výpočet akčného zásahu, a preto v každom kroku zavoláme riešič quadprog () na riešenie QP. Dynamickú odozvu vozidla simulujeme pomocou stavovej rovnice. V MATLABe toto implementujeme v nasledujúcom skripte:

Zdrojový kód 7.3. MPCuzavretaslucka.m

```

run=20;                                % Dlzka sim.
X=[0.05 0 0 0]';                         % Poc, stav
for i=1:run                                % Slucka sim.
    U (:,i)=quadprog(H,G*X (:,i),Ac,bc,[],[],[],[],[],o);
    X (:,i+1)=A*X (:,i)+B*U (1,i);          % MPC sim.
end

```

Výsledky môžeme jednoducho zobraziť pomocou vlastnej, predvytvorenej funkcie vykreslitXU(), pomocou ktorej znázorníme aj obmedzenia na vstup:

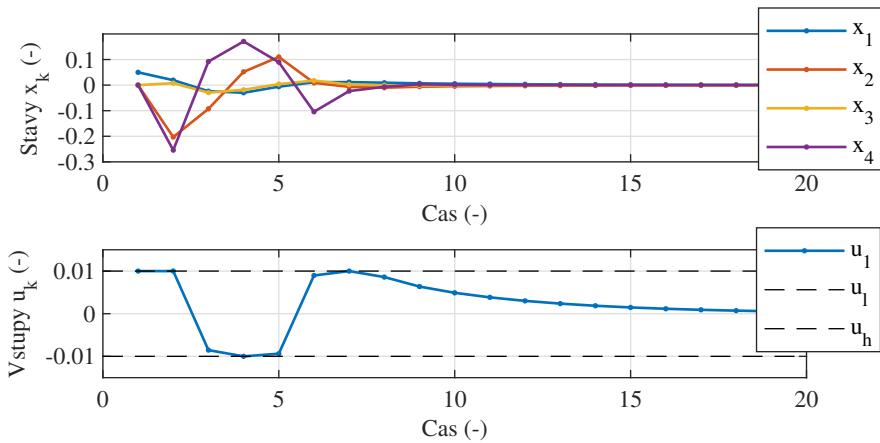
```

vykreslitXU(X,U(1,:));                  % Vykreslenie x,u
line([0,run], [uh uh], 'Color','k', 'LineStyle','--') % uh
line([0,run], [ul ul], 'Color','k', 'LineStyle','--') % ul

```

Po spustení programu dostaneme výsledok simulácie ako na Obr. 7.4. Vidíme, že akčný člen počas spätnovázobného riadenia respektoval procesné obmedzenia. Stratégia MPC už teda vopred zarátala do aktuálnych vstupov aj budúci vývoj dynamiky vozidla ako aj prítomnosť obmedzení na danom horizonte predikcie. Tretí stav vyjadruje najdôležitejšiu veličinu: výškový rozdiel vozidla a zavesenia kolesa, ktorý sme úspešne a rýchlo priviedli do referenčného stavu.

Vykonajme ďalej rýchlu simuláciu na overenie účinnosti riadenia. Definujme si novú pomocnú stavovú premennú, do ktorej si budeme ukladať výsledky simulácie



Obr. 7.4. Stavy a vstup pri spätnoväzbovom riadení podvozku autobusu pomocou algoritmu MPC s obmedzeniami.

bez riadenia. V slučke budeme simulať lineárny stavový model, čím vypočítame odozvu na prechod na vozovku bez aktívneho riadenia podvozku. Výsledok vykreslíme do jedného grafu, označíme osi a vytvoríme legendu podľa nasledujúceho skriptu:

```

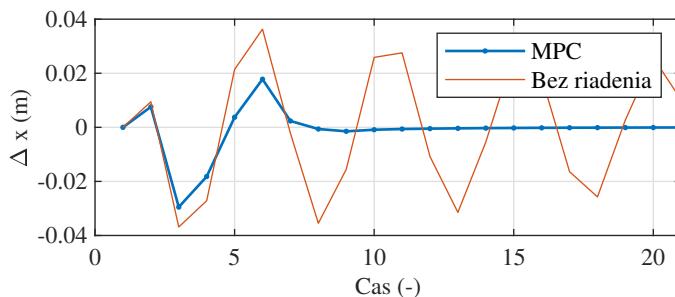
Xo=X;
for i=1:run
    Xo (:, i+1)=A*Xo (:, i)+B*0;
end
figure(2)
plot(C*X); hold on; plot(C*Xo);
xlabel('Cas (-)'); ylabel ('\Delta x (m)')
grid on
legend('MPC', 'Bez riadenia')

```

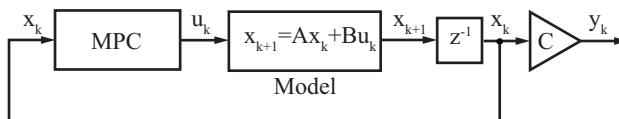
% Pociat. stav  
 % Slucka simulacie  
 % Bez riadenia  
 % Koniec slucky  
 % Nove graf. okno  
 % Vystup  
 % Oznac. osi  
 % Mriezka  
 % Legenda

Výsledok je znázornený na Obr. 7.5. Vidíme, že vplyvom riadiaceho systému sme zvýšili komfort jazdy.

□



Obr. 7.5. Porovnanie rozdielu výchylky neodpruženej hmoty (nadstavby) voči výchylke odpruženej hmoty (kolesu) vozidla pri prechode cez prekážku s aktívnym riadením podvozku a bez neho.



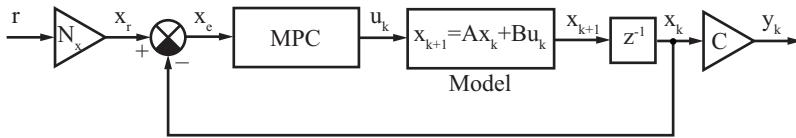
Obr. 7.6. Schematický náčrt spätnoväzbového riadenia do nulovej referencie s úplným meraním stavu.

## 7.4 Sledovanie referenčnej trajektórie

Doposiaľ sme pri odvodení stratégie prediktívneho riadenia za účelom zjednodušenia výkladu predpokladali, že systém musíme dovesti do rovnovážneho stavu, inými slovami, že máme nulovú referenčnú úroveň. V tomto prípade je vstupom do algoritmu MPC nameraný alebo odhadnutý stav, ako to je znázornené na Obr. 7.6, kde v blokovej schéme celkovo absentuje reprezentácia žiadanej veličiny.

Existuje mnoho praktických aplikácií, kde dynamický systém musíme udržiavať na nulovej žiadanej hodnote, ale sledovanie predpísanej žiadanej trajektórie má v automatickom riadení taktiež obrovský význam. V tejto podkapitole sa preto bližšie pozrieme na prípady riadenia so sledovaním trajektórie židaných hodnôt.

Je dôležité poznamenať, že odvodenie riadiaceho zákona pre prípad nenulovej referencie neplatí výlučne pre prediktívne riadenie. Nasledujúce techniky zabezpečenia sledovania predpísanej trajektórie sú všeobecne platné aj pre iné metódy stavového spätnoväzbového riadenia, pričom boli všeobecne používané dávno pred vyuvinutím prediktívneho riadenia.



Obr. 7.7. Schematický náčrt spätnoväzobného riadenia podľa referenčnej trajektórie pomocou matice stavového príkazu.

#### 7.4.1 Referenčné vstupy

Našim cieľom bude vyjadriť žiadanú veličinu  $r$  vo forme ekvivalentného žiadaného referenčného stavu  $\mathbf{x}_r$ . Na výpočet referenčného stavu môžeme zaviesť novú transformačnú maticu, ktorú budeme nazývať maticou stavového príkazu<sup>3</sup>  $\mathbf{N}_x$  podľa

$$\mathbf{N}_x \mathbf{r} = \mathbf{x}_r. \quad (7.3)$$

Maticu  $\mathbf{N}_x$  môžeme formulovať na základe matice výstupu  $\mathbf{C}$ . Napríklad, ak prvý element stavového vektora modelu druhého rádu je priamo meraný snímačmi a chceme, aby tento stav sledoval referenčnú trajektóriu, maticu stavového príkazu môžeme vytvoriť ako  $\mathbf{N}_x = \mathbf{C}^T [1 \ 0]^T$ .

Trikom pre realizáciu spätnoväzbového stavového riadenia so sledovaním predpisanej trajektórie je predpoklad, že namiesto dynamického stavu systému chceme priviesť rozdiel medzi skutočným a referenčným stavom do nuly. Tento rozdiel teda môžeme chápať ako istú stavovú regulačnú odchýlku<sup>4</sup>, a vyjadriť ako

$$\mathbf{x}_e = \mathbf{x}_k - \mathbf{x}_r = \mathbf{x}_k - \mathbf{N}_x \mathbf{r}. \quad (7.4)$$

Tento koncept je graficky znázornený na Obr. 7.7, kde žiadaná hodnota je najprv pomocou matice stavového príkazu transformovaná na žiadaný stav a potom sa vypočíta rozdiel medzi skutočným a žiadaným stavom, ktorý ďalej vstupuje do algoritmu MPC ako aktuálna stavová odchýlka riadenia. Stratégia MPC sa potom teda snaží minimalizovať rozdiel medzi žiadaným a skutočným stavom.

Ak online časť algoritmu riešime pomocou rutiny `quadprog()` a predpokladáme, že aktuálna žiadaná hodnota je daná premennou  $r$  a matica stavového príkazu je  $\mathbf{Nx}$ , formuláciu optimalizačného problému prispôsobíme takto:

```
uast=quadprog (H, G* (x-Nx*r) ,Ac ,bc)
```

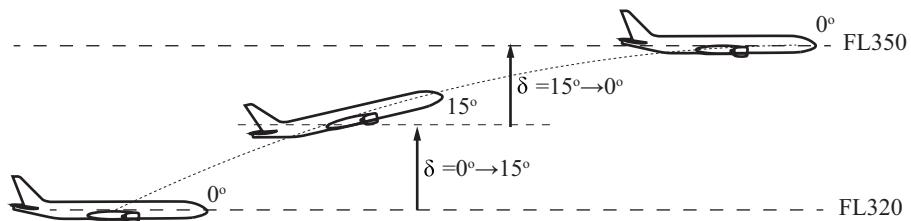
Toto samozrejme platí nielen pre MPC, ale pre akúkoľvek inú stratégiu stavového riadenia.

<sup>3</sup>angl. state command matrix

<sup>4</sup>Index  $\cdot_e$  je odovodený z anglického výrazu pre chybu: error.



Obr. 7.8. Dopravné lietadlo Boeing 777 [67].



Obr. 7.9. Zmena letovej výšky lietadla Boeing 777-300ER.

**Príklad 7.4.** Uvažujme dynamiku dopravného lietadla Boeing 777-ER (Obr. 7.8) z Pr. 3.7. Navrhnite riadenie servomechanizmu výškového kormidla pomocou algoritmu prediktívneho riadenia, ak z bezpečnostných dôvodov sa výškové kormidlo môže pohybovať iba v rozmedzí  $-0.1 \leq u_k \leq 0.1$  rad, a zároveň zmena stúpania z nuly na  $15^\circ$  a späť sa musí dosiahnuť za približne 1 minútu letu. Navrhnutý systém otestujte simulačne tak, že uvažujte nulové stúpanie (náklon) na začiatku s tým, že lietadlo letí na letovej hladine FL300 (30000 stôp), a potom pilot dostane z riadiacej veže príkaz na zvýšenie letovej hladiny na FL350, preto zmení výškové kormidlo na  $\sim 15^\circ$  (0.25 rad) a po dosiahnutí novej letovej hladiny lietadlo vráti do vodorovnej polohy (Obr. 7.9). Na začiatku manévra je stúpanie a zmena stúpania nulová spolu s uhlom nábehu.

Na začiatku postupujeme prakticky identicky ako v prípade Pr. 3.7. Definujeme si spojitu linearizovanú reprezentáciu letovej dynamiky v smere stúpania lietadla, zvolíme vzorkovanie a vytvoríme samostatný objekt na uloženie spojitej dynamiky. Model zdiskretizujeme a vyčítame matice diskrétnej reprezentácie:

Zdrojový kód 7.4. stavovyprikaz.m

```

Ac = [-0.313 56.7 0; -0.0139 -0.426 0; 0 56.7 0];
Bc = [0.232; 0.0203; 0]; % Matica vstupu

```

```

C = [0 0 1];                                % Matica vystupu
Ts=0.5;                                     % Vzorkovanie [s]
b777 = ss (Ac,Bc,C,[]);                    % Spojity model
model=c2d(b777,Ts);                        % Diskretizacia
A=model.a; B=model.b;                      % Matice A a B

```

Na základe predchádzajúcej diskusie si vytvoríme aj maticu stavového príkazu  $\mathbf{N}_x$ . Budeme sledovať jednoduchú referenčnú trajektóriu stúpania lietadla, a teda  $\mathbf{N}_x = \mathbf{C}^T$ . Žiadanú hodnotu stúpania zadáme napríklad do vektora  $\mathbf{rr}$ , z ktorého neskôr môžeme na základe simulačného času vybrať správne prvky. Počiatočný stav lietadla zvolíme na základe zadania ako nulový:

```

Nx=C';                                     % Stavovy prikaz
rr=[0.25 0];                                % Ziad. hodnoty y
X=[0 0 0]';                                  % Pociatocny stav

```

Pokračujeme deklarovaním parametrov simulácie a riadenia. Zvolíme si 150 krokov dlhú simuláciu, čo predstavuje viac ako 1 minútu letu. Vyčítame si počet stavov a vstupov diskrétneho stavového modelu, zvolíme príslušné váhovacie matice a horizont predikcie. Deklarujeme si tiež obmedzenia na vstupy:

```

run=150;                                     % Dlzka simulacie
[nx nu]=size(B);                            % Pocet x a u
Q=eye(nx); R=1; np=25;                      % Vahy a horizont
uh=0.1; ul=-uh;                            % Min a max vstup

```

Nasleduje offline časť algoritmu MPC. Vypočítame koncové váhovanie, matice účelovej funkcie a matice obmedzení v tvare nerovnosti:

```

[K, P]=iterdlqr(A,B,Q,R,100);              % Konc. vahovanie
[H, G]=ucelovafunkcia(A,B,np,Q,R,P);       % Ucelova funkcia
[Ac bc]=obmedzenia(ul,uh,np);              % Obmedzenia u

```

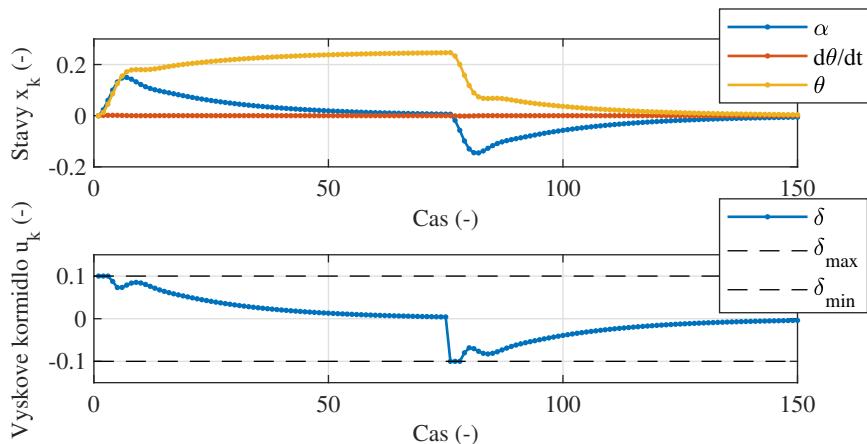
Ďalej si vypneme niektoré varovania. Drobné numerické nedostatky Hessiánu môžeme odstrániť jeho symetrizáciou, vypneme slovné hodnotenie výsledkov optimalizácie a nakoniec môžeme deaktivovať aj varovanie algoritmu `lsim()`, ktorý budeme využívať na simuláciu odozvy lietadla:

```

H=(H+H')/2;                                  % Sym. H
o=optimoptions('quadprog','Display','none');  % Vypnut
warning('off','Control:analysis:LsimStartTime') % Lsim

```

Nasleduje online časť algoritmu MPC a samotná simulácia. Proces budeme opakovať až do konca simulácie. Najprv si určíme referenčnú trajektóriu a pomocou podmienky `if` ju v polovici trvania simulácie zmeníme na nulové stúpanie.



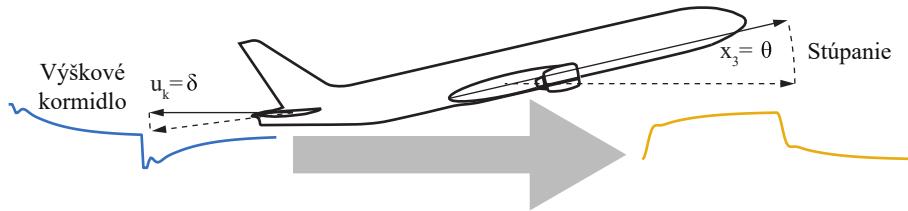
Obr. 7.10. Prediktívne riadenie letovej výšky lietadla Boeing 777-300ER.

Simulovať môžeme samozrejme ľubovoľnú referenčnú trajektóriu, skoková zmena žiadaného stúpania by však nebola príliš vhodná pre reálne lietadlo. Nasleduje výpočet optimálnej postupnosti akčných zásahov na základe kvadratickej optimálizácie. Všimnime si drobnú zmenu na vstupnej strane funkcie, kde teraz zahŕňame aj vplyv referenčnej trajektórie.

Poznamenajme, že aj keď vykonávame simuláciu, na odozvu dynamiky systému nemusíme nutne využívať diskrétny model, ktorý je identický predikčným modelom. Ak napríklad predpokladáme lineárne riadenie ale máme k dispozícii komplexnejší nelineárny model, môžeme ho využiť na simuláciu dynamiky systému. V tomto príklade si tento koncept ilustrujeme tak, že namiesto diskrétneho predikčného modelu použijeme pôvodný spojity model tak, že vypočítame jeho odozvu pomocou funkcie `lsim()` a výsledky si uložíme do premennej `X`:

Náš program ukončíme grafickým zobrazením výsledkov a obmedzení:

```
vykreslitXU(X,U(1,:)); % Vykreslenie x,u
line([0,run],[uh uh],'Color','k','LineStyle','--') % uh
line([0,run],[ul ul],'Color','k','LineStyle','--') % ul
```



Obr. 7.11. Ilustrácia vstupov a výstupov pri zmene letovej výšky.

Po spustení simulácie dostaneme grafické znázornenie stavov a vstupu ako vidíme na Obr. 7.10. Horný graf znázorňuje v poradí uhol nábehu, zmenu stúpania a samotné stúpanie, kym spodný graf znázorňuje dynamické zmeny v nastavení uhla výškového kormidla. Všimnime si, že na začiatku a v strede simulácie boli obmedzenia aktívne. Na Obr. 7.11 je znázornená grafická ilustrácia tohto manévra.

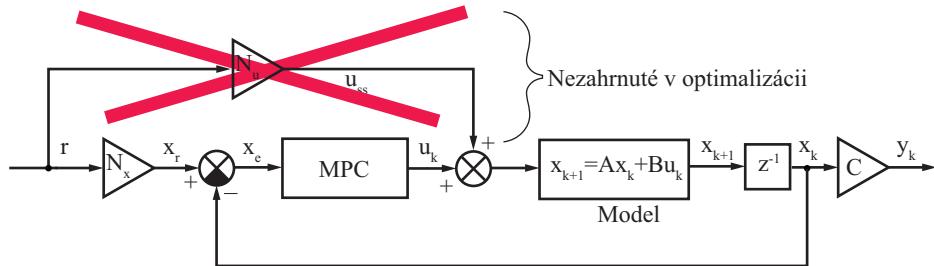
□

#### 7.4.2 Integračná zložka

Pri probléme sledovania referenčnej trajektórie žiaľ úplne všetko nevyriešime iba posunutím referenčnej úrovne stavu. Predstavme si teraz systém, kde musíme sledovať predpísanú referenčnú trajektóriu. Predpokladajme na chvíľu, že nás dynamický systém dosiahol predpísaný referenčný stav. Čo sa pritom stane so vstupmi? Budú v ustálenom stave nulové. Prípad riadenia výškového kormidla z Pr. 7.4 dobre ilustruje tento koncept. Pri žiadanej zmene stúpania riadiaca stratégia zmení aj nastavenie výškového kormidla. Ak stúpanie dosiaholo požadovanú úroveň, postupne môžeme kormidlo vrátiť do pôvodnej nulovej polohy. V ustálenom stave už ďalej nepotrebujeme žiadny akčný zásah.

Niektoré systémy však potrebujú účinok akčného zásahu na to, aby mohli udržať žiadanú trajektóriu v ustálenom stave. Jednoducho povedané, nemôžeme predpokladať, že hned po privedení systému do ustáleného stavu môžeme vypnúť akčné členy. Vo všeobecnosti potrebujeme účinok akčných členov na udržanie systému v ustálenom stave. Predstavme si napríklad chemický reaktor, kde cieľom je sledovať predpísaný profil teploty a riadiť príkon vykurovacích telies. Po zmene žiadanej hodnoty teploty nemôžeme vypnúť vykurovacie telesá, kedže musíme zabezpečiť konštantnú teplotu aj v novom ustálenom stave.

Kompenzácia účinku ustálených vstupov na systém pomocou doprednej väzby je častým riešením problému sledovania predpísanej trajektórie [51]. Táto myšlienka je graficky znázornená na Obr. 7.12, kde výraz  $N_u$  vyjadruje účinok riadiaceho člena v ustálenom stave. V našom prípade žiaľ takéto riešenie nie je vždy



Obr. 7.12. Schematický náčrt doprednej väzby na zabezpečenie sledovania žiadannej referenčnej trajektórie pomocou tzv. matice stavového príkazu a s riadiacim členom v ustálenom stave.

vhodné – mali by sme zložku vektora vstupov, ktorá obide MPC riadiaci zákon, vstup by teda nebol výlučne riešením optimalizačnej úlohy.

Ďalším dôvodom prečo takéto riešenie nie je dostatočne vhodné ani v prípade LQ riadenia je nepresnosť modelov. Ak sledovanie referenčnej trajektórie odvodíme zo stavov a vstupov v ustálenom stave na základe samotného modelu, skutočný stav alebo výstup nemusí nutne sledovať žiadanú hodnotu. Lepším riešením preto je zavedenie tzv. integračnej zložky do realizácie algoritmu riadenia.

Vypočítajme si sumu – inými slovami diskrétnu integráciu – odchýlky riadenia,  $e_k$ , medzi skutočným výstupom a referenciou podľa

$$\mathbf{x}_{k+1}^I = \mathbf{x}_k^I + \mathbf{e}_k, \quad (7.5)$$

kde  $\mathbf{x}_k^I$  bude nový pomocný stav systému, ktorý vyjadruje sumu odchýlky riadenia. Rov. (7.5) teda predstavuje model dynamiky vývoja chyby riadenia, čo je jednoduchá diskrétna sumácia. Rovnicu (7.5) môžeme ekvivalentne vyjadriť aj integráciou späť v čase a dostaneme

$$\mathbf{x}_k^I = \mathbf{x}_{k-1}^I + \mathbf{e}_{k-1}. \quad (7.6)$$

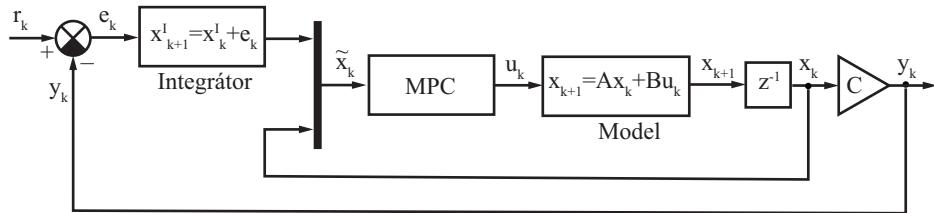
Nezabudnime, že v  $Z$ -oblasti operátor  $z^{-1}$  znamená jednokrokové spätné posunutie signálu, t. j. určitú pamäť. Ak predpokladáme sledovanie iba jedného výstupu, kde  $e_k = r_k - y_k$ , tak platí

$$\tilde{x}(z) = z^{-1}\tilde{x}(z) + z^{-1}e(z), \quad (7.7)$$

a po vyjadrení diskrétnej prenosovej funkcie dostaneme

$$F(z) = \frac{z^{-1}}{1 - z^{-1}} = \frac{1}{z - 1}, \quad (7.8)$$

čo je vlastne  $Z$ -transformácia doprednej Eulerovej integrácie.



Obr. 7.13. Integrácia odchýlky riadenia na sledovanie referenčnej trajektórie.

Teraz si vyjadrite aktuálnu odchýlku riadenia ako rozdiel žiadanej hodnoty a skutočného sledovaného výstupu a dosadme ju do Rov. (7.5), aby sme dostali

$$\mathbf{x}_{k+1}^I = \mathbf{x}_k^I + (\mathbf{r}_k - \mathbf{y}_k), \quad (7.9)$$

čo môžeme ďalej po dosadení za výstupy zapísť ako

$$\mathbf{x}_{k+1}^I = \mathbf{x}_k^I + (\mathbf{r}_k - \mathbf{C}\mathbf{x}_k). \quad (7.10)$$

Kombinujme teraz integrátor z Rov. 7.10 so samotným dynamickým modelom systému, aby sme dostali rozšírenú<sup>5</sup> stavovú reprezentáciu. Dynamiku riadeného systému často potrebujeme rozšíriť o nové funkcie. Rozšírením pôvodnej dynamiky môžeme zahrnúť do modelu účinok filtrov alebo napríklad vytvoriť prírastkovú formu riadiacich algoritmov. V tomto prípade bude rozšírený model predikcie obsahovať iba integrátor a dostávame tak

$$\begin{bmatrix} \mathbf{x}_{k+1}^I \\ \mathbf{x}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\mathbf{C} \\ \mathbf{0} & \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k^I \\ \mathbf{x}_k \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{B} \end{bmatrix} \mathbf{u}_k + \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \mathbf{r}_k. \quad (7.11)$$

Ak označíme nový, rozšírený stav ako  $\tilde{\mathbf{x}}_k$ , potom model rozšírený o integrátor bude mať tvar

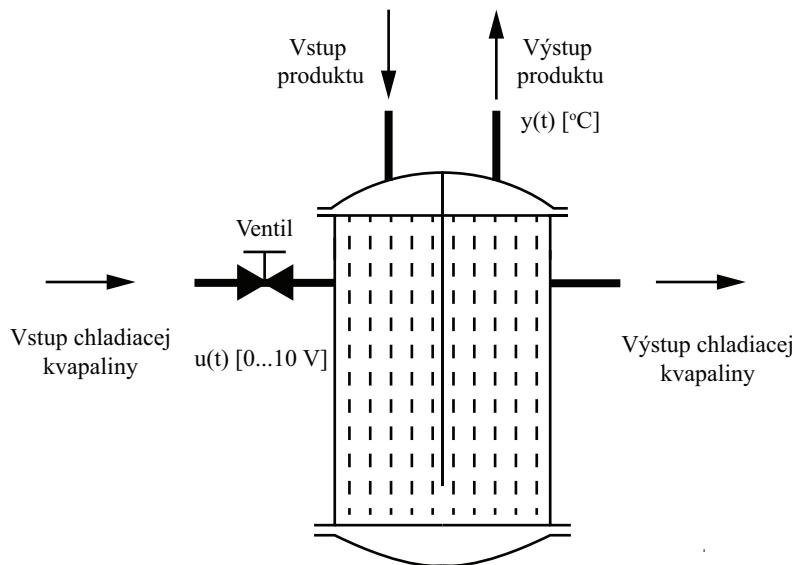
$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k + \tilde{\mathbf{B}}\mathbf{u}_k + \mathbf{r}_k, \quad (7.12)$$

kde  $\tilde{\mathbf{A}}$  a  $\tilde{\mathbf{B}}$  sú nové matice dynamiky a vstupu rozšírenej dynamiky.

Príslušná bloková schéma je znázornená na Obr. 7.13. Integrátor jednoducho sčítava chybu riadenia. Tento stav spolu s dynamickým stavom tvoria rozšírený stav. Prediktívny regulátor bude preto vychádziať z rozšíreného modelu.

**Príklad 7.5.** Uvažujme výmenník tepla znázornený na Obr. 7.14, ktorý pomočou prietoku chladiacej kvapaliny konštantnej teploty na vstupnej strane chladí produkt na výstupnej strane. Vieme, že prietok chladiacej kvapaliny na vstupnej strane je riadený pomocou ventilu, ktorý prijíma napäťový signál 0 až 10 V.

<sup>5</sup>angl. augmented, extended



Obr. 7.14. Ilustrácia zjednodušenej schémy výmenníka tepla.

Na jednotku zmeny zavretia ventilu stúpne teplota produktu o  $10^{\circ}\text{C}$ . Po skokovej zmene stavu ventilu dosiahne kvapalina na výstupe  $63\%$  ustálenej hodnoty po  $60\text{ s}$ . Vytvorte MPC riadenie teploty produktu na základe zmien polohy ventilu chladiacej kvapaliny. Sledujte skokové zmeny teplotného profilu z počiatočnej teploty  $25^{\circ}\text{C}$  na  $80^{\circ}\text{C}$  a v polovici simulácie na  $50^{\circ}\text{C}$ .

Výmenník môžeme na základe dostupných údajov modelovať prenosom prvého rádu<sup>6</sup> a môžeme predpokladať zosilnenie  $k = 10^{\circ}\text{C/V}$  a časovú konštantu  $\tau = 60\text{ s}$ . Pre prenos prvého rádu v tvare

$$G(s) = \frac{k}{\tau s + 1} \quad (7.13)$$

platí diferenciálna rovnica

$$\tau \dot{y}(t) + y(t) = k u(t), \quad (7.14)$$

kde  $u(t)$  je napätie do ventilu a  $y(t)$  je teplota na výstupe. Diferenciálnu rovnicu voľbou  $x(t) = y(t)$  prepíšme na stavovú rovnicu a dostaneme

$$\dot{x}(t) = -\frac{1}{\tau} + \frac{k}{\tau} u(t), \quad (7.15)$$

<sup>6</sup>V skutočnom výmenníku by sme potrebovali modelovať minimálne aj oneskorenie, čo tu pre zjednodušenie príkladu neuvažujeme.

z čoho vyplýva, že  $\mathbf{A}_c = -\frac{1}{\tau}$ ,  $\mathbf{B}_c = -\frac{k}{\tau}$  a  $\mathbf{C}_c = 1$ .

Spojity model si zdiskretizujeme s uvažovaním vzorkovacej periódy  $T_s = 5$  s a simuláciu necháme bežať 150 krokov. Profil žiadanych teplôt si uložíme do vektora, z ktorého budeme počas behu simulácie vyberať správne elementy. Za váhovanie stavov a vstupu si zvolíme jednotkovú maticu a horizont predikcie s dĺžkou  $n_p = 15$  krokov. Vo voľbe váhovania stavov musíme už rátať s rozšíreným modelom: máme dynamiku prvého rádu a jeden sledovaný výstup, na základe čoho dostávame rozšírený systém druhého rádu. Zadanie jasne definuje aj obmedzenia na vstupe:

Zdrojový kód 7.5. integrator.m

```
Ts=5; % Vzorkovanie [s]
run=150; % Dĺžka simulacie
rr=[80 50]; % Ziadana teplota
Q=eye(2); R=1; np=15; % Vahy a horizont
uh=10; ul=0; % Min a max vstup
```

Na základe vyššie uvedeného odvodenia poznáme aj model výmenníka. Spojity model vytvoríme na základe parametrov prenosu a potom ho zdiskretizujeme pomocou príkazu `c2dm()`:

```
k=10; tau=60; % Parametre modelu
Ac=-1/tau; Bc=k/tau; Cc=1; Dc=0; % Spojity model
[A,B,C,D]=c2dm(Ac,Bc,Cc,Dc,Ts); % Diskretny model
vymennik=ss(Ac,Bc,Cc,Dc); % Objekt modelu
```

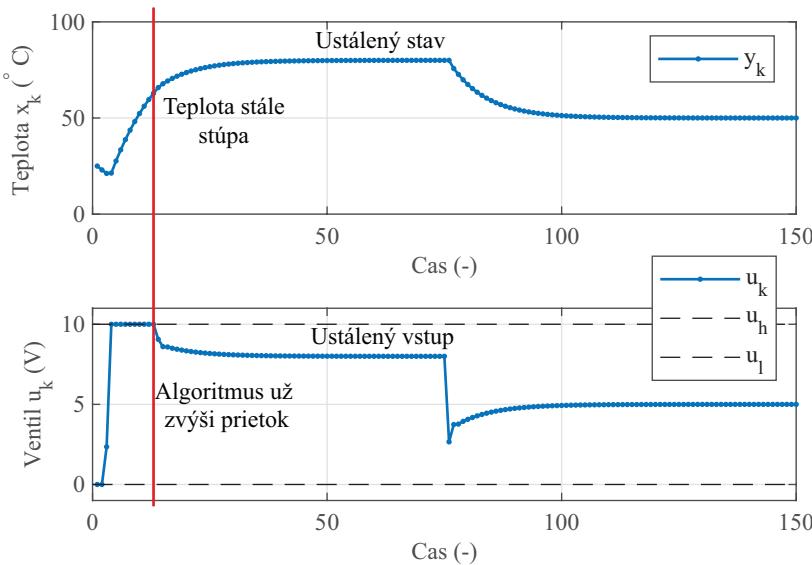
Model rozšírený o integrátor dostaneme podľa Rov. (7.12). Teplota výmenníka na začiatku simulácie je  $25^\circ\text{C}$ , čo môžeme jednoducho zahrnúť do počiatočného stavu systému:

```
At=[1 -C; 0 A]; % Rozsirene A
Bt=[0; B]; % Dynamika + intg.
X=25'; xI=0; % Rozsirene B
% Pociatoc. stav.
```

Nasleduje offline výpočet koncového váhovania, matíc účelovej funkcie a matíc obmedzení vstupu v tvare nerovnosti:

```
[K,P]=iterdlqr(At,Bt,Q,R,100); % Konc. vahovanie
[H,G]=ucelovafunkcia(At,Bt,np,Q,R,P); % Ucelova funkcia
[Ac bc]=obmedzenia(ul,uh,np); % Obmedzenia u
```

Pokračujeme symetrizáciou Hessiánu a vypnutím varovaní:



Obr. 7.15. Sledovanie predpísaného priebehu teploty produktu vo výmenníku tepla pomocou rozšírenia stavovej reprezentácie o integračnú zložku.

```

H=(H+H')/2; % Sym. H
o=optimoptions('quadprog','Display','none'); % Vypnut
warning('off','Control:analysis:LsimStartTime') % Lsim

```

Čo sa týka online časti simulácie poznamenajme, že integrácia odchýlky riadenia je jednoduchý proces sumácie rozdielu žiadanej hodnoty a sledovaného výstupu. Do riešiča kvadratickej optimalizácie musíme špecifikovať rozšírený stav, ktorý v tomto prípade pozostáva z integrálu odchýlky a samotnej diskrétnej dynamiky:

```

for i=1:run % Simul. slucka
  if i<=run/2; r=rr(1); else r=rr(2); end % Referencia
  xI=xI+(r-X(i)); % Integrator
  U(:,i)=quadprog(H,G*[xI; X(:,i)],Ac,bc,[],[],[],[],o
    );
  [y,t,x]=lsim(vymennik,[U(1,i) U(1,i)],[(i-1) i]*Ts,X(:,i));
  X(:,i+1)=x(end,:); % Aktualny stav
end

```

Na záver si graficky znázorníme výsledky simulácie. Dynamický stav systému sa rovná teplote produktu a na vstupe vidíme akčné zásahy poslané na ventil:

```
vykreslitXU(X,U(1,:)); % Vykreslenie x,u
line([0,run],[uh uh],'Color','k','LineStyle','--') % uh
line([0,run],[ul ul],'Color','k','LineStyle','--') % ul
```

Po spustení programu dostaneme podobnú odozvu ako na Obr. 7.15. Horný graf znázorňuje teplotu produktu z výmenníka, kým dolný graf predstavuje akčné zásahy posielané na ventil. Vidíme, že na udržiavanie danej teploty potrebujeme podržať akčné zásahy aj v ustálenom stave, čo po rozšírení o integrátor MPC algoritmus správne vykoná. Ako možno vidieť, na začiatku simulácie regulátor úplne uzavŕší ventil, aby prudko zvýšil teplotu. Všimnime si, že stratégia prediktívneho riadenia už vopred „vidí“ potrebu začať pomaly znižovať teplotu na ustálenú úroveň. Z maximálne uzavretého ventilu prediktívny regulátor uberať ešte když teplota stále stúpa! Taktiež vidíme, že obmedzenia na vstup sú pritom dodržané.

□

**Príklad 7.6.** Vytvorte univerzálnu funkciu na rozšírenie stavovej reprezentácie o sledovanie žiadanej trajektórie výstupov na základe integrátora. Funkciu otestujte na Pr. 7.5.

Na strane vstupných argumentov bude nová funkcia potrebovať matice **A**, **B** a **C** dynamickej odozvy systému. Na výstupnej strane funkcia vráti rozšírené matice **Ā**, **Ā** a **Ā**. Funkciu si teda definujeme pomocou nasledujúceho skriptu:

Zdrojový kód 7.6. rozsirenieintegrator.m label

```
function [At,Bt,Ct] = rozsirenieintegrator(A,B,C)
```

Pokračujeme zistením počtu stavov, vstupov a výstupov:

```
[ny nx]=size(C); % Pocet vystupov
[nx nu]=size(B); % Pocet stavov, vstupov
```

Na záver vytvoríme rozšírené matice na základe Rov. 7.12:

```
At=[eye(ny,nx) -C; % Rozsirene A
    zeros(nx) A]; % Dynamika + integrator
Bt=[zeros(nx,nu); B]; % Rozsirene B
Ct=zeros(ny,nx) C]; % Rozsirene C
```

Novú funkciu môžeme otestovať jednoduchou modifikáciou skriptu z Pr. 7.5 (pozri *testrozsirenieintegrator.m*). □

## 7.5 Prírastky vstupov a ich obmedzenie

V prípade niektorých praktických aplikácií je výhodnejšie vyjadriť akčné zásahy vo forme zmeny vstupu oproti jeho predchádzajúcej hodnote, ako používať amplitúdu akčných zásahov. Predstavme si napríklad polohovanie osí CNC stroja. Servomotor, ktorý obsluhuje danú os, musíme riadením neustále držať v žiadanej polohe. Po dosiahnutí požadovanej polohy ale už pri absencii porúch nemusíme do motora posielat žiadne nové akčné zásahy. Inak povedané, zmena veľkosti akčného zásahu oproti jeho predchádzajúcej hodnote je nulová. Pri výskytu poruchy potom vždy posielame do servomotora iba potrebnú úroveň korekcie.

Ďalšou motiváciou na výpočet zmien akčných zásahov namiesto ich amplitúdy sú samotné procesné obmedzenia. Pri potrebe obmedzení rýchlosťi zmeny akčných zásahov úlohu môžeme jednoducho preformovať tak, že zvyčajné obmedzenia vstupov budú vlastne obmedzovať zmenu vstupov.

V takomto prípade hovoríme o tzv. prírastkovom alebo inkrementálnom tvaru algoritmu riadenia. Rovnako ako pre PID riadenie aj pre stavové riadenie môžeme formulovať algoritmus v prírastkovom tvaru. Tento tvar pritom nie je špecifický pre prípad prediktívneho riadenia, ale podobný postup platí aj v prípade iných metód stavového riadenia.

Prírastok, teda zmenu akčného zásahu, pritom môžeme vyjadriť ako

$$\Delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}, \quad (7.16)$$

z čoho si aktuálny akčný zásah v čase  $k$  vyjadríme ako

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \Delta \mathbf{u}_k. \quad (7.17)$$

Poznamenajme, že Rov. (7.17) môžeme posunúť v diskrétnom čase o jeden krok dopredu, pričom dostaneme akčný zásah v čase  $k+1$  definovaný ako

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \Delta \mathbf{u}_{k+1}. \quad (7.18)$$

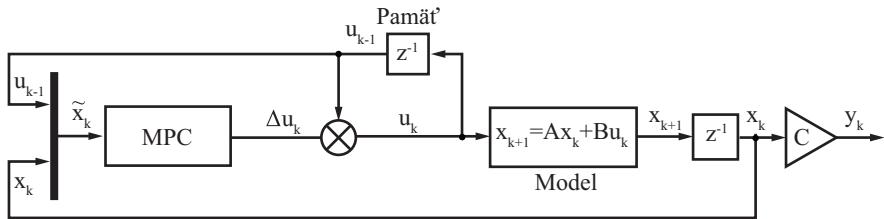
Ak dynamiku riadeného systému rozšírime o Rov. (7.18), platí

$$\begin{bmatrix} \mathbf{u}_{k+1} \\ \mathbf{x}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B} & \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{u}_k \\ \mathbf{x}_k \end{bmatrix} + \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \Delta \mathbf{u}_{k+1}. \quad (7.19)$$

Podobne môžeme rozšíriť stavový model aj o Rov. (7.17), a dostaneme

$$\begin{bmatrix} \mathbf{u}_k \\ \mathbf{x}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B} & \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{k-1} \\ \mathbf{x}_k \end{bmatrix} + \begin{bmatrix} \mathbf{I} \\ \mathbf{B} \end{bmatrix} \Delta \mathbf{u}_k. \quad (7.20)$$

Musíme si uvedomiť, že pôvodnú stavovú reprezentáciu sme takto rozšírili o novú stavovú dynamiku. Na nový stav, ktorý vyjadruje sčítanie prírastkov, pritom musíme v algoritme spätnoväzbového riadenia vytvoriť istú formu „odhadu“; ktorá



Obr. 7.16. Schematický náčrt MPC s rozšírením predikčnej dynamiky na prírastkový tvar.

sa v tomto prípade zredukuje na jednoduchú sumáciu. Pri praktickej programovej realizácii pritom musíme dbať hlavne na to, *kde* vykonáme sčítanie prírastku a prechádzajúcej hodnoty vstupu – po alebo pred výpočtom  $\Delta \mathbf{u}_k$  (Obr. 7.16).

Podobne ako v prípade integračnej zložky, ak si označíme nový, rozšírený stav pomocou stavovej premennej  $\tilde{\mathbf{x}}_k$ , potom model transformovaný na prírastkový tvar bude

$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k + \tilde{\mathbf{B}}\mathbf{u}_k, \quad (7.21)$$

kde  $\tilde{\mathbf{A}}$  a  $\tilde{\mathbf{B}}$  predstavujú nové matice dynamiky a vstupu rozšírenej dynamiky.

Pri rozšírení stavovej reprezentácie musíme zmeniť tiež štruktúru váhovacích matíc. Váhovanie rozšíreného stavu bude pozostávať z pôvodného váhovania vstupov  $\mathbf{R}$  a stavov  $\mathbf{Q}$ , t. j. bude dané maticou

$$\tilde{\mathbf{Q}} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{bmatrix}, \quad (7.22)$$

ktorej rozmery sú  $(n_x + n_u) \times (n_x + n_u)$ . Význam novej váhovacej matice vstupov,  $\tilde{\mathbf{R}}$ , sa zmení tak, že teraz bude penalizovať prírastky vstupov v účelovej funkcií.

**Príklad 7.7.** Zopakujte príklad na aktívne spätnoväzbové riadenie výšky podvozku autobusu z Pr. 7.3, ale tentoraz uvažujte rozšírenie dynamiky na prírastkový tvar vstupu. Namiesto obmedzenia absolútnej hodnoty výšky podvozku autobusu teraz v záujme komfortu cestujúcich bude našou snahou obmedziť rýchlosť, resp. zmenu akčného zásahu na  $\dot{u}(t) = 0.1 \text{ m/s}$ .

Kedže realizácia riadiaceho systému bude diskrétna, musíme interpretovať rýchlosť zmeny výšky na dovolenú zmenu akčného zásahu v danom vzorkovacom intervale. Vzorkovanie modelu z Pr. 7.3 je  $T_s = 0.1 \text{ s}$ , za jednu vzorku môžeme výšku zmeniť maximálne o  $\Delta u_k = 0.01 \text{ m}$  na to, aby sme počas jednej sekundy nemenili výšku viac ako  $1 \text{ m/s}$ . Obmedzenie prírastku akčného zásahu je potom dané ako  $-0.01 \leq \Delta u_k \leq 0.01 \text{ (m/s na vzorku)}$ .

Na začiatku si načítame diskrétnu verziu štvrtinového modelu. Pre váhovanie stavov, váhovanie vstupov a dĺžku horizontu môžeme zvoliť rovnaké hodnoty ako v pôvodnom príklade:

Zdrojový kód 7.7. inkrementalny.m label

```
load bus.mat % Nacitaj model
A=bus.a; B=bus.b(:,2); C=bus.c; % Model autobusu
Q=C'*C; R=0.01; np=10; % Vahy a horizont
```

Pokračujeme zistením počtu stavov, vstupov a výstupov stavového modelu a deklarujeme minimálnu a maximálnu dovolenú zmene akčného zásahu:

```
[nx,nu]=size(B); % Rozmery modelu
ny=size(C,1); % ...
duh=0.01; dul=-duh; % Obmedzenia du
```

Nasleduje samotná transformácia stavovej reprezentácie. Pôvodné matice **A**, **B** a **C** rozšírime, aby sme dostali namiesto absolútnej hodnoty vstupov ich prírastok:

```
At=[eye(nu) zeros(nu,nx); % Rozsirene A
     B      A ]; % Dynamika+inkr.
Bt=[eye(nu); zeros(nx,nu)]; % Rozsirene B
Ct=[zeros(nu,ny) C]; % Rozsirene C
```

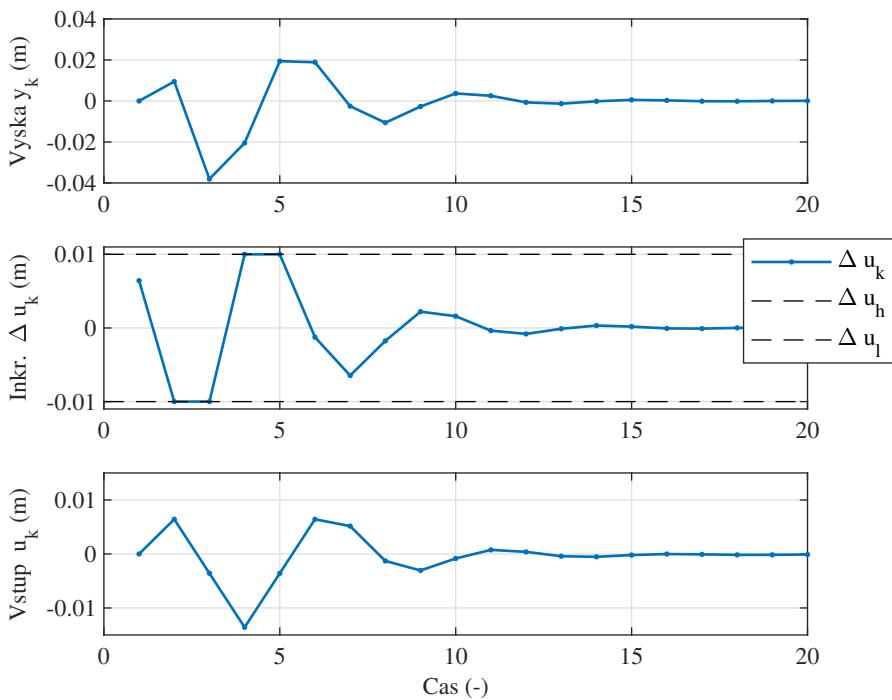
Nezabudnime na prispôsobenie váhovacích matíc stavu a vstupu. Váhovanie stavu rozšírenej reprezentácie  $\tilde{Q}$  bude pozostávať z pôvodného váhovania absolútnej hodnoty vstupov a z pôvodného váhovania stavov. Váhovanie vstupov  $\tilde{R}$  bude penalizovať zmeny akčného zásahu:

```
Qt=[R zeros(nu,nx); % Rozsirene Q
     zeros(nx,nu) Q]; % ...
Rt=1; % Rozsirene R
```

Offline časť výpočtu zostáva prakticky nezmenená, ale musíme uvažovať rozšírenú dynamiku ako aj rozšírené váhovanie stavov a vstupov na výpočet matice váhovania koncového stavu a matíc účelovej funkcie. Obmedzenia zostavujeme pre prírastok vstupov:

```
[K,P]=iterdlqr(At,Bt,Qt,Rt,100); % Vahovanie xnp
[H,G]=ucelovafunkcia(At,Bt,np,Qt,Rt,P); % Ucelova funkcia
[Ac,bc]=obmedzenia(dul,duh,np); % Obmedzenia na u
```

Ako sme si už zvykli, vypneme slovné hodnotenie riešenia QP a pre istotu prevedieme Hessián na symetrickú formu:



Obr. 7.17. MPC riadenie výšky podvozku autobusu pomocou výpočtu vstupov v prírastkovom tvare s ohľadom na ich obmedzenia.

```
H=(H+H')/2; % Symetr. H
o=optimoptions('quadprog','Display','none'); % Vyp. vypis
```

Pre simuláciu MPC riadenia v uzavretej slučke si najprv zvolíme dĺžku simulácie a počiatočný stav. Nezabudnime, že v predikčnom modeli máme ďalší stav, ktorý vyjadruje sumáciu zmien akčných zásahov. Amplitúdu akčných zásahov musíme preto inicializovať tiež. Riešiť QP zavoláme už s využitím rozšíreného stavu a prírastok vstupov stále počítame jednoduchou sumáciou:

```
run=20; % Dlzka simulacie
X=[0.05 0 0 0]'; % Pociatoocny stav
U=0; % Pociatoc. vstup
for i=1:run % Simul. slucka
    dU(:,i)=quadprog(H,G*[U(i); X(:,i)],Ac,bc
    ,[],[],[],[],[],o);
```

```

U(i+1)=U(i)+dU(1,i); % Inkrementuj u
X(:,i+1)=A*X(:,i)+B*U(1,i); % Simul. dynamiky
end

```

Výsledky simulácie v zmysle výstupu, prírastkov a amplitúdy vstupu si zobražíme do grafu:

```

subplot(3,1,1); plot(C*X); grid on % Vystup, mriezka
ylabel('Vyska y_k (m)'); % Os y, znacenie
subplot(3,1,2); plot(dU(1,:)); grid on % Prirastok vstupu
ylabel('Inkr. \Delta u_k (m)'); % Os y, znacenie
line([0,run],[duh duh],'Color','k','LineStyle','--')
line([0,run],[-duh -duh],'Color','k','LineStyle','--')
subplot(3,1,3); plot(U); grid on % Vstup u
ylabel('Vstup u_k (m)'); % Os y, znacenie
xlabel('Cas (-)'); % Os x, znacenie

```

Po spustení simulácie dostaneme výsledky znázornené na Obr. 7.17. Stredný graf zobrazuje prírastky akčného zásahu, ktoré sú obmedzené v zmysle zadania. V tomto príklade sme teda namiesto amplitúdy akčných zásahov (spodný graf) obmedzili ich prírastok – zmenu výšky podvozku.

□

**Príklad 7.8.** Vytvorte univerzálnu funkciu na rozšírenie stavového modelu o prírastok vstupov. Novú funkciu overte a otestujte na Pr. 7.7.

Podobne ako v prípade funkcie na rozšírenie stavovej reprezentácie o integrátor, aj tu potrebujeme ako jej vstupné argumenty matice stavového modelu, **A**, **B** a **C**, na výpočet rozšíreného stavového modelu:

Zdrojový kód 7.8. rozsirenieinkrementy.m

```
function [At,Bt,Ct] = rozsirenieinkrementy(A,B,C)
```

Po zistení rozmerov matíc môžeme vytvoriť rozšírenie na základe Rov. 7.20:

```

[nx nu]=size(B); % Rozmery modelu
ny=size(C,1); % ...
At=[eye(nu) zeros(nu,nx); % Rozsirene A
    B A];
Bt=[eye(nu); zeros(nx,nu)]; % Dynamika+inkrement
Ct=zeros(nu,ny) C]; % Rozsirene B
% Rozsirene C

```

Výslednú funkciu otestujeme modifikáciou Pr. 7.7, kde na rozšírenie použijeme našu novú funkciu `rozsirenieinkrementy()` (pozri *testrozsirenieinkrementy.m*).

□

V prípade prírastkovej formy výpočtu akčného zásahu budú naše pôvodné obmedzenia na vstupy namiesto amplitúdy  $u_k$  ohraničovať iba zmenu vstupov  $\Delta u_k$ . Ak chceme obmedziť aj amplitúdu vstupu, máme jednoduchú úlohu – stačí totiž sformulovať podmienku v tvare stavového obmedzenia. Nezabudnime, že takto vstúpi do úlohy riadenia komplexná interakcia stavov a vstupov, kde obmedzenia na prírastky značne vplývajú aj na ďalší vývoj amplitúdy vstupov.

**Príklad 7.9.** Zopakujte príklad aktívneho riadenia podvozku autobusu z Pr. 7.7. Okrem obmedzenia na rýchlosť zmeny v záujme komfortu cestujúcich uvažujte aj fyzikálne limity zdvihu akčného člena  $u_{\max} = \pm 10$  mm.

Zdvih akčného člena môžeme v úlohe riadenia interpretovať ako obmedzenie na amplitúdu vstupu, ktoré bude mať tvar  $-0.01 \leq u_k \leq 0.01$  (m), pričom je ho pri formulácii MPC problému potrebné uvažovať na celom horizonte predikcie.

Prevažná časť programovej realizácie príkladu ostáva zachovaná, preto si ďalej uvedieme iba relevantné zmeny v zostavení obmedzení. Začíname deklaráciou obmedzení prírastku vstupu, potom deklarujeme aj maximálnu dovolenú amplitúdu vstupu. Obmedzenie amplitúdy zadáme ako stavové obmedzenie. Pokiaľ pritom nechceme obmedziť samotné vnútorné stavy systému, môžeme na príslušné miesta zadať nereálne veľké čísla:

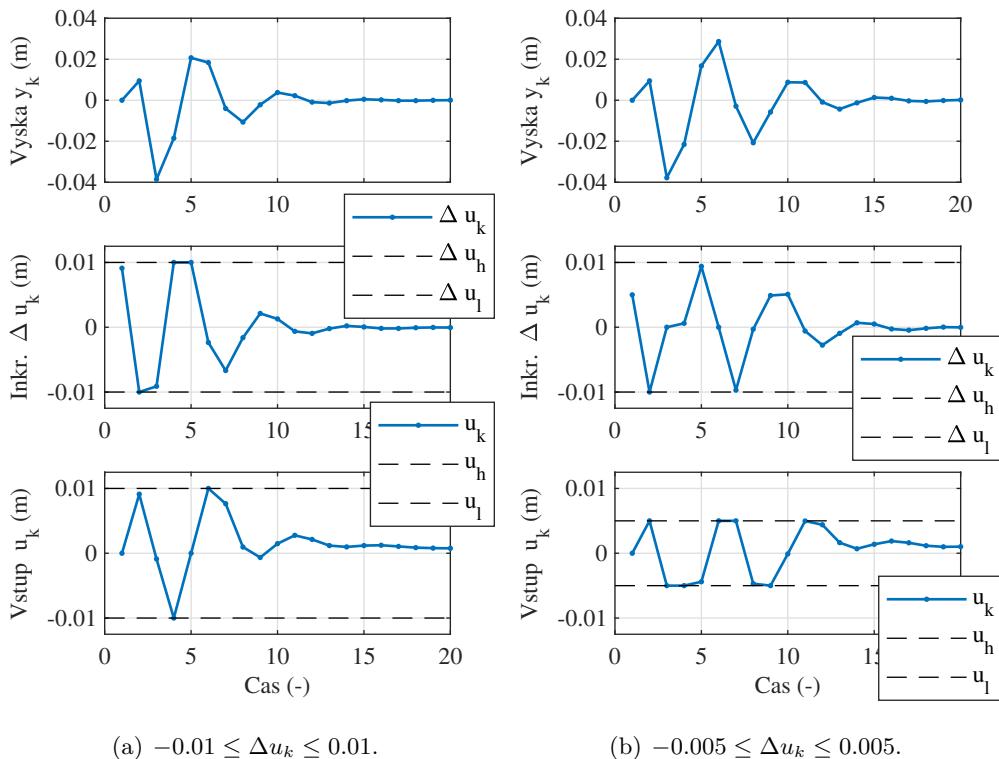
Zdrojový kód 7.9. kombinovaneobmedzenia.m label

```
duh=0.01; dul=-duh; % Prírastky
uh= 0.01; % Amplituda
xh=[uh 1E8 1E8 1E8]'; % Stavy
x1=-xh; % Symetr. x
[Ac b0 B0]=obmedzenia(dul,duh,x1,xh,np,At,Bt); % Obmedz.
```

Volanie rutiny kvadratického programovania vyžaduje zmenu na strane zadávania obmedzení, kde musíme pri každej vzorke prerátať  $\mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0 \tilde{\mathbf{x}}_k$ :

```
dU(:,i)=quadprog(H,G*[U(i); X(:,i)],Ac,(b0+B0*[U(i);
X(:,i)]),[],[],[],[],[],o);
```

Po spustení simulácie a doplnení vykreslenia výsledkov o obmedzenie na amplitúdu vstupov dostaneme priebeh znázornený na Obr. 7.18. Vidíme, že obmedzenia na prírastky sú často aktívne, kým obmedzenie na amplitúdu vstupu je taktiež aktívne.



Obr. 7.18. Simulácia prediktívneho riadenia aktívneho podvozku autobusu s obmedzeniami na amplitúdu ako aj prírastky vstupu.

Uvedené obmedzenia si samozrejme nemôžeme predstaviť ako jednoduchú saturáciu daných veličín, keďže algoritmus musí s nimi rátať pri riešení optimalizačného problému a prispôsobiť im priebeh riadenia. Ak by sme znížili dovolený zdvih akčných členov na  $\pm 5$  mm, obmedzenia na stav by boli aktívne vo väčšej miere ako obmedzenia na rýchlosť akčného zásahu.

□

## 7.6 MPC v prostredí Simulink

Účelovú funkciu – a v niektorých prípadoch aj obmedzenia – stačí vypočítať iba raz, za predpokladu, že neuvažujeme zmenu predikčného modelu alebo obmedzení

počas riadenia. V takomto prípade sa implementácia úlohy prediktívneho riadenia v prostredí MATLAB–Simulink zredukuje iba na opakované riešenie problému QP a prípadnú odozvu modelovanej dynamiky. Musíme však nájsť spôsob ako riešiť problém kvadratického programovania v prostredí Simulink.

Hotový programový blok na riešenie problému QP by sme žiaľ v knižnici Simulinku nenašli, pretože súčasná verzia<sup>7</sup> softvéru túto možnosť neponúka a je pravdepodobné, že podobnú funkcia neponúkne ani v budúcych verziach.

Najjednoduchšia alternatíva je volanie funkcie `quadprog()` priamo zo simulácie v Simulinku, žiaľ toto nie je úplne ideálne. Cieľom Simulinku totiž nie je iba poskytovať alternatívne grafické rozhranie na modelovanie a riešenie rôznych inžinierskych úloh, ale aj schopnosť generovať počítačový zdrojový kód<sup>8</sup>, ktorý je prekladateľný<sup>9</sup> priamo do spustiteľného súboru na danej výpočtovej platorme – do tzv. *exekutív*<sup>10</sup>. Interpretovaný jazyk MATLABu je výborný na vývoj algoritmov, ale z hľadiska výpočtového výkonu je neefektívny. Simulink umožní z blokovej schémy vygenerovať program v jazyku C/C++, ktorý je potom preložený do exekutív pre danú výpočtovú platformu. V najjednoduchšom prípade takto môžeme zrýchliť simulácie na PC, ale môžeme aj vytvoriť hotové algoritmy riadenia a používať ich na vnorených mikroprocesoroch. Simulink umožňuje využívať väčšinu vlastných blokov ako aj MATLAB funkcií na ich preklad do exekutív. Ak chceme preklaadať MATLAB funkcie do C/C++, Simulink zavolá rozšírenia *Matlab Coder* alebo *Simulink Coder*. Ak využívame napríklad funkciu `eig()` na výpočet vlastných čísel, Simulink najprv vytvorí z danej funkcie alebo bloku C/C++ zdrojový kód, ktorý ďalej prekladá na cieľovú platformu. Prečo je to relevantné? Lebo takto zrýchli simuláciu na PC alebo umožní, aby náš program bežal aj na vnorených<sup>11</sup> platformách, ako je napríklad Arduino. Žiaľ, v čase písania tejto učebnice práve `quadprog()` tvoril výnimku.



V Simulinku nie je dostupný zdrojový kód rutiny `quadprog()`, a preto nie je možný jej preklad do exekutív.

Vlastný funkčný blok v Simulinku nájdeme v knižnici pod *Simulink*→*User-Defined Functions*→*MATLAB Function*. Ak otvoríme funkčný blok (Obr. 7.19), môžeme rovno začať písť vlastnú funkciu, ktorá bude ďalej využiteľná v Simulinku (Obr. 7.20). Vstupné a výstupné parametre sa objavia aj graficky na vytvo-

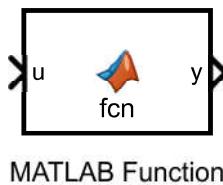
<sup>7</sup>R2018a

<sup>8</sup>angl. source code

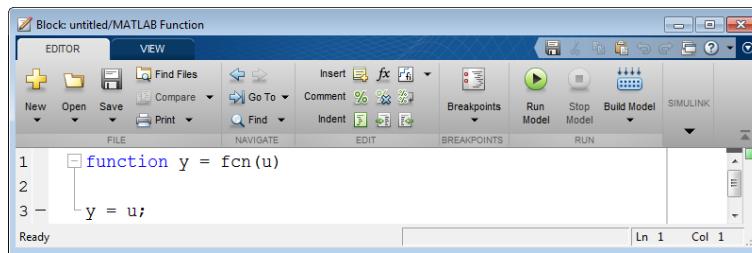
<sup>9</sup>angl. compile

<sup>10</sup>angl. executable (file)

<sup>11</sup>angl. embedded



Obr. 7.19. Vlastný funkčný blok v Simulinku na základe interpretovaného jazyka MATLABu.



Obr. 7.20. Písanie vlastnej funkcie pre Simulink pomocou jazyku MATLAB.

renom bloku. Ak používame MATLAB funkcie, ktoré sú dostupné pre preklad do exekutív, Simulink sa bude snažiť vytvoriť komplikovanú verziu. To znamená, že v simulácii nebude používať interpretovaný kód, t. j. nebude to „bežať“ tak ako v príkazovom riadku MATLABu.

Poznamenajme, že ak by sme priamo zavolali `quadprog()` v Simulinku, dostali by sme nasledujúce chybové hlásenie:

```
Function 'quadprog' not supported for code generation.
```

Naštastie existuje aj taká možnosť, že vopred ohlásime, že danú funkciu Simulink nemá ani skúsiť prekladať a stačí, ak ju využije ako interpretovanú MATLAB funkciu. Ohlásíť vonkajšiu funkciu môžeme cez direktívú

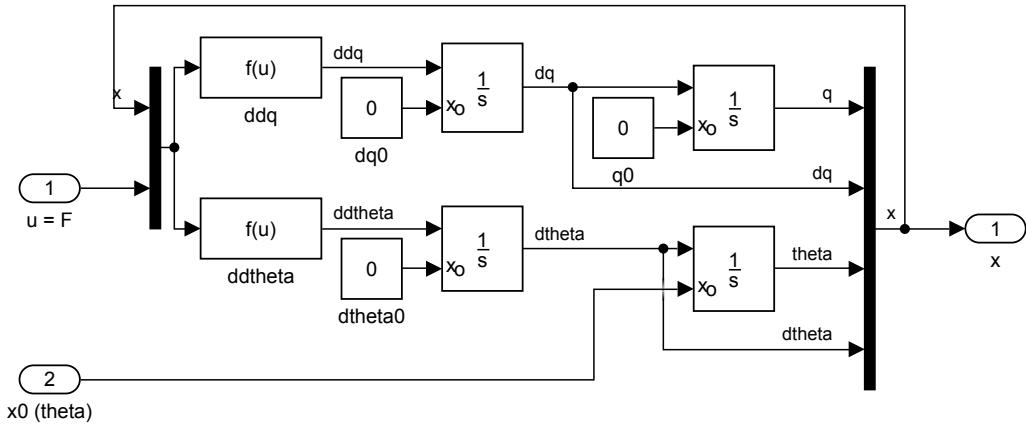
```
coder.extrinsic('nazovfunkcie');
```

čo znamená, že vo funkcionálite generovania kódu (`coder`) nastavíme našu funkciu ako vonkajšiu<sup>12</sup> (`extrinsic`). Ak teda použijeme

```
coder.extrinsic('quadprog');
```

potom môžeme v Simulinku zavolať aj riešič `quadprog()` s tým obmedzením, že kód nebude prekladaný do exekutív.

<sup>12</sup>angl. extrinsic



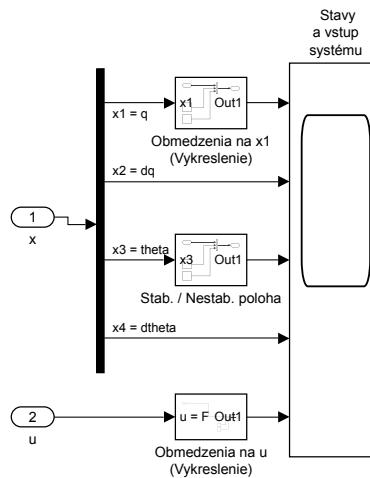
Obr. 7.21. Podštém nelineárneho modelu invertovaného kyvadla.

Ak už vieme ako v Simulinku implementovať riešenie QP, je stále otázne aké časti formulácie budeme dynamicky meniť. Ako to bolo spomenuté na začiatku, budeme uvažovať nepremenlivý model predickie a pevne dané obmedzenia. Ak sú potrebné aj stavové obmedzenia, vstup do problému QP musíme prepočítať dynamicky na základe signálu, ktorý obsahuje stav alebo jeho odhad. V prípade, ak uvažujeme premenlivý model predikcie, teda adaptívne prediktívne riadenie, aj účelovú funkciu by sme mali priebežne prepočítavať počas riadenia. V tejto podkapitole budeme uvažovať najjednoduchší prípad, kde môžeme MPC jednoznačne rozdeliť na offline a online časť.

Nasledujúci príklad demonštruje realizáciu PID a LQ riadenia na zopakovanie zručnosti spätnoväzbového riadenia v Simulinku. Potom si predstavíme MPC riadenie v Simulinku s použitím riešiča `quadprog()`:

**Príklad 7.10.** Vytvorte schému spätnoväzbového PID a LQ riadenia invertovaného kyvadla v prostredí MATLAB–Simulinku. Na začiatku simulácie predpokladajte vychýlenie kyvadla z jeho hornej, nestabilnej rovnovážnej polohy o  $30^\circ$ , pričom vozík sa nachádza v strede dráhy. Cieľom riadenia je stabilizácia kyvadla v nestabilnej rovnovážnej polohe. Dráha vozíka je dlhá 1 m a absolútна hodnota sily pôsobiacej na vozík musí byť menšia ako  $\pm 5$  N. Porovnajte odozvu kyvadla v a) otvorennej slučke, b) pri automaticky naladenom PID riadení a c) pri LQ riadení. Uvažujte spôsob implementácie procesných obmedzení a ich vplyv na riadenie.

Ako riadený systém môžeme použiť nelineárny model systému z Pr. 2.3. Predpokladajme dokonalú znalosť dynamických stavov, t.j. tentoraz vynecháme zo simulačnej realizácie algoritmus na odhad stavu. Začneme s implementáciou nelineárneho modelu kyvadla, ktorú môžeme prebrať priamo z Pr. 2.3. Vyznačme



Obr. 7.22. Podsystém na grafické znázornenie výstupov simulácie.

myšou strednú časť simulačnej schémy (pozri Obr. 2.3), ktorá obsahuje samotnú dynamiku kyvadla a stlačením *CTRL+G* vytvorime podsystém<sup>13</sup>, aby sme dostali prehľadnejšiu simulačnú schému. Výsledkom bude podsystém, ktorý je znázornený na Obr. 7.21. Podsystém uvažuje vstupné brány<sup>14</sup> pre akčný zásah a pre tretí počiatočný stav, ktorý simuluje uhlovú výchylku kyvadla. Podsystém tiež obsahuje jednu výstupnú bránu<sup>15</sup> na dynamický stav.

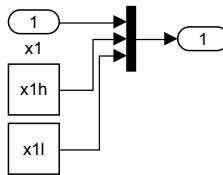
Vytvorime si teraz ďalší podsystém z Pr. 2.3 ktorý bude zodpovedný za vykreslenie výsledkov. Označme si na ľavej časti pôvodnej simulácie demultiplexer a virtuálny osciloskop a vytvorime z nich podsystém podľa Obr. 7.22. Podsystém bude obsahovať dve vstupné brány: jednu na stavy a jednu na vstup.

Pre ľahšie orientovanie sa v grafických výsledkov môžeme dokresliť horizontálne čiary, ktoré budú symbolizovať procesné obmedzenia a referenčné výchylky kyvadla. Jeden z týchto podsystémov je znázornený na Obr. 7.23. Daný podstavček obsahuje jeden vstup, ktorý symbolizuje napríklad prvý dynamický stav – polohu vozíka. Jedným z fyzikálnych obmedzení kyvadla je dĺžka dráhy vozíka. Ak predpokladáme 1 m dlhú dráhu, tak vozík sa z nulovej polohy môže pohybovať na dráhe  $\pm 0.5$  m. Cielom tohto podstavčku je vykresliť tieto ohraničenia ako vodorovné čiary. Z knižnice vyberieme *Simulink*→*Signal Routing*→*Mux* a pomocou dialógového okna si vytvoríme trojbránový multiplexer. Do ďalších brán priveďme bloky konštant, ktoré budú symbolizovať horné a dolné obmedzenia danej

<sup>13</sup> angl. subsystem

<sup>14</sup> angl. input port

<sup>15</sup> angl. output port



Obr. 7.23. Podsystém na dokreslenie dvoch obmedzení do virtuálneho osciloskopu.

veličiny. Napríklad  $x1h$  bude symbolizovať horné obmedzenie prvého stavu a  $x1l$  dolné obmedzenie prvého stavu. Takéto podsystémy môžeme vytvoriť napríklad na obmedzenia vstupu alebo na vyznačenie stabilnej ( $\pi$  rad) a nestabilnej (0 rad) rovnovážnej polohy kyvadla.

Ďalej si vytvorime schému spätnoväzbovej simulácie riadenia invertovaného kyvadla podľa Obr. 7.24. Môžeme začať spojením výstupu podsystému kyvadla a grafického bloku. Môžeme pokračovať vložením dvoch manuálnych prepínačov z knižnice *Simulink*→*Signal Routing*→*Manual Switch*. Prvý manuálny prepínač bude slúžiť na prepínanie medzi neriadeným a riadeným kyvadlom, preto na prvý vstup prepínača pripojíme blok konštanty *Simulink*→*Sources*→*Constant*, ktorú prepíšeme na nulovú hodnotu. Výstup prepínača prepojíme na vstup nelineárneho modelu kyvadla. Riadenie budeme realizovať pomocou číslicového počítača, preto medzi prepínač a model môžeme vložiť blok tvarovača nultého rádu, ktorú nájdeme v *Simulink*→*Discrete*→*Zero-Order Hold*. Blok tiež potrebuje informáciu o vzorkovaní, kde zadáme premennú  $Ts$ , ktorá predstavuje vzorkovaciu períodu.

Pokračujeme privedením signálu na druhý vstup prepínača. V zadaní príkladu máme obmedzenie na veľkosť akčného zásahu  $-5 \leq u_k \leq 5$  N. Kedže uvažujeme iba PID a LQ riadenie, jediný spôsob na dodržanie akčných zásahov je nasýtenie (saturácia) vstupov. Preto nájdeme blok *Simulink*→*Discontinuities*→*Saturation* a nastavíme premenné  $uh$  a  $ul$  ako definované horné a dolné limity nasýtenia signálu. Ďalej vložíme do schémy druhý manuálny prepínač, ktorý bude slúžiť na výber PID alebo LQ riadenia. Na realizáciu PID algoritmu môžeme vybrať hotový blok diskrétneho algoritmu z *Simulink*→*Discrete*→*Discrete PID Controller*. Ako spätnú väzbu predpokladáme priame meranie stavov, preto môžeme z výstupu podsystému kyvadla priviesť signál do PID bloku. Kedže uvažujeme nulovú žiadanosť hodnotu, budeme používať iba uhol kyvadla ako vstup do bloku PID. Použijeme preto štvorvýstupový demultiplexer z *Simulink*→*Signal Routing*→*Demux* na rozklad zbernicového<sup>16</sup> signálu stavu na štyri komponenty, z ktorých využijeme len tretiu zložku – uhlovú výchylku kyvadla. Týmto je slučka na PID riadenie pripravená.

<sup>16</sup> angl. bus

Na implementáciu LQ riadenia potrebujeme poznať iba LQ zosilnenie, ktoré si označíme premennou  $K_{1q}$ . Vložme do schémy blok zosilnenia z *Simulink*→*Math Operations*→*Gain* a dvojitým kliknutím na blok otvorime dialógové okno. Tu musíme nastaviť spôsob násobenia na *Multiplication*→*Matrix( $K^*u$ )(u vector)*, t. j. maticové násobenie, kde zosilnením násobíme stavový vektor. Ako zosilnenie uvedieme LQ zosilnenie so záporným známenkom. Na vstup bloku zosilnenia priveďieme stav a výstup pripojíme na manuálny prepínač. Výslednú blokovú schému možno nájsť v súbore *kyvadloLQ.slx*.

Blokové schémy môžu pri rôznych operáciach automaticky spúštať časť MATLAB kódu, MATLAB skript alebo aj funkciu. Na deklaráciu parametrov samotného kyvadla môžeme tieto údaje napríklad priamo zabudovať do bloku. Otvorme *File*→*Model Properties* a kliknime na záložku *Callbacks*, ktorá ponúka práve túto funkciu. Pri otvorení funkcie budeme potrebovať parametre, preto klikneme na *PreLoadFcn* a zadáme parametre

```
m1=0.4; % Hmotnosť vozika [kg]
m2=0.08; % Hmotnosť kyvadla [kg]
l=0.2; % Vzdialenosť taziska od osi rotacie [m]
g=9.81; % Tiazevo zrychlenie [m/s2]
b=0.1; % Koeficient trenia [m/s]
```

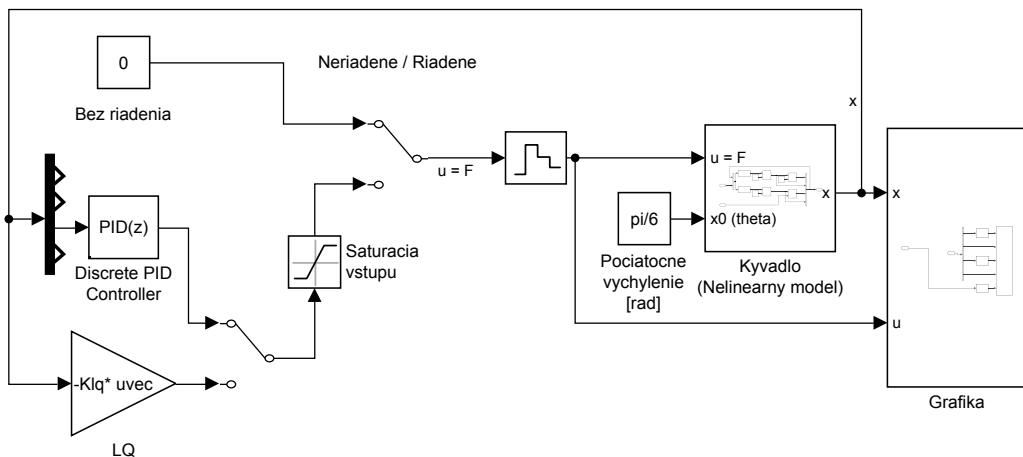
zo zadania Pr. 2.3.

Naša schéma obsahuje parametre ako vzorkovanie  $T_s$ , obmedzenie vstupu  $u_h$  a  $u_l$ , LQ zosilnenie  $K_{1q}$ , a na grafické zobrazenie napríklad obmedzenie dráhy vozíka  $x_{1h}$  a  $x_{2h}$ . Tieto parametre si zadáme vo vonkajšom skripte, ktorý môžeme nazvať napríklad *kyvadloOfflineLQ.m*. Potom, ak chceme pred každým spustením simulačnej schémy prepočítať parametre, môžeme do časti *File*→*Model Properties*→*Model Properties*→*Callbacks*→*StartFcn* zadat aj samotný názov súboru *kyvadloOfflineLQ*.

Vytvorime si teraz spomínaný skript. Pri návrhu LQ riadenia vychádzajme z lineárneho modelu, ktorý sme v Pr. 2.3 získali linearizáciou nelineárneho systému v jeho hornej, nestabilnej rovnovážnej polohe. Na začiatku si načítame tento model zo súboru *kyvadlo.mat*. Nasleduje uloženie matíc a vyčítanie ich rozmerov. Vzorkovanie musíme dodržať na základe nášho diskretizovaného modelu, a preto si tento údaj načítame z uloženého súboru. Nasleduje definícia maximálneho akčného zásahu a orientačného obmedzenia na prvý stav predstavujúci výchylku vozíka:

Zdrojový kód 7.10. *kyvadloOfflineLQ.m*

```
load kyvadlo.mat % Linearizovany model
A=kyvadlo.a; B=kyvadlo.b; C=kyvadlo.c;
```



Obr. 7.24. Výsledná bloková schéma spätnoväzbového PID a LQ riadenia kyvadla.

```
[nx nu]=size(B); % Rozmery stavoveho modelu
Ts=kyvadlo.Ts; % Vzorkovacia perioda [s]
uh=5; % Max. sila na vozik [N]
ul=-uh; % Min. sila na vozik [N]
x1h=0.5; % Max. draha vozika [m]
x1l=-x1h; % Min. draha vozika [m]
```

Pokračujme výpočtom LQ zosilnenia. Na začiatok si zvolme váhovanie stavov  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  a váhovanie vstupu  $\mathbf{R} = R = 1$ . Na výpočet zosilnenia môžeme využiť zabudovanú funkciu `d1qr()`:

```

Qlq=C'*C;                                % Vahovanie stavov
Rlq=1;                                    % Vahovanie vstupu
Klq=dlqr(A,B,Qlq,Rlq);                 % LQ zosilnenie

```

Nasleduje ladenie PID riadenia. Manuálne prepínače nastavme na PID riadenú odozvu. Dvojitým kliknutím na blok PID riadenia otvorime dialógové okno. Na ladenie môžeme zvoliť nástroj automatického ladenia. Kliknutím na *Tune...* sa otvorí nástroj *PID Tuner*. Odozva je na prvý pohľad prijateľná, a preto môžeme nastavené hodnoty prevziať do bloku kliknutím na *Update Block*. Dostaneme hodnoty  $P = 67.9$ ,  $I = 210.8$ ,  $D = 4.7$  a za filter  $N = 193.2$ .

Časť schémy pre simuláciu voľnej odozvy a PID riadenia je hotová. Prepínače na začiatok nastavme na režim simulácie bez riadenia. Po spustení dostaneme odozvu na Obr. 7.25(a). Vstup do systému v piatom grafe je samozrejme počas celého trvania simulácie nulový, keďže nepoužívame žiadne akčné zásahy. Uhlová

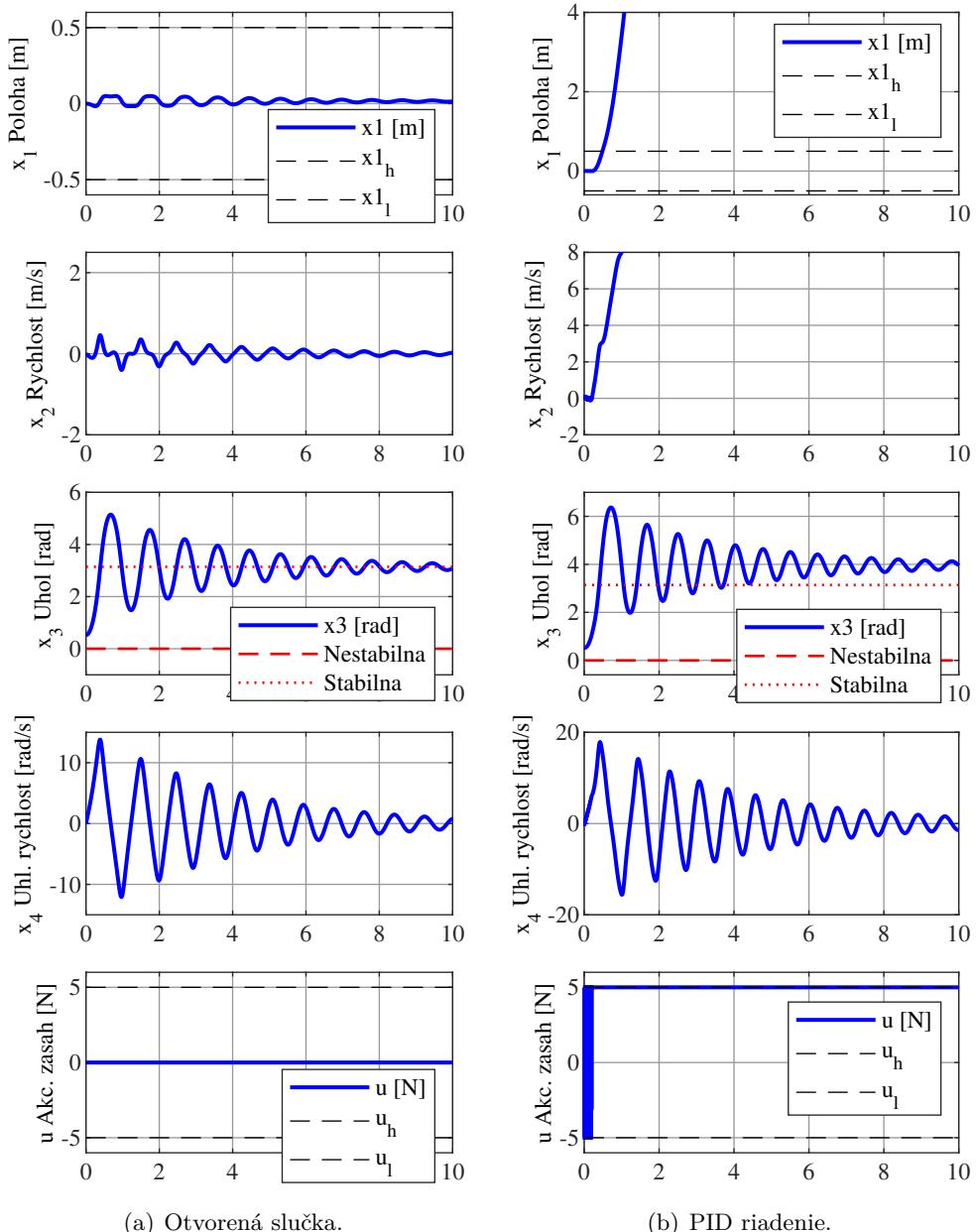
výchylka kyvadla v treťom grafe začína  $30^\circ$  ( $\pi/6$  rad) od hornej, nestabilnej rovnovážnej polohy. Bez riadenia kyvadlo samozrejme padne a kmitá okolo dolnej, stabilnej rovnovážnej polohy ( $\pi$ ), kde sa po niekoľkých sekundách aj ustáli. Simulácia tiež veľmi správne zachytí aj fyzikálny efekt malého pasívneho pohybu vozíka z dôvodu kmitania kyvadla.

Po prepnutí na PID riadenie a spustení simulácie získame výsledky znázornené na Obr. 7.25(b). Na prvý pohľad vidíme, že napriek tomu, že automatické naladenie parametrov regulátora vykazovalo stabilnú odozvu, výchylka vozíka exponenciálne rastie a v konečnom dôsledku získame nestabilnú odozvu. V skutočnosti by vozík samozrejme rýchlo narazil na koniec dráhy, a preto je zvyšná časť simulácie nerealistická. Napriek tomu má uhlová výchylka kyvadla stále istú výpovednú hodnotu: jej priebeh sa stabilizuje okolo rovnovážnej polohy. Aj keď je daná situácia nerealistická, simulácia tiež veľmi správne zachytí aj efekt vozíka, ktorý je stále v pohybe a môžeme očakávať aj zmenu rýchlosťi: kyvadlo je trochu vychýlené.

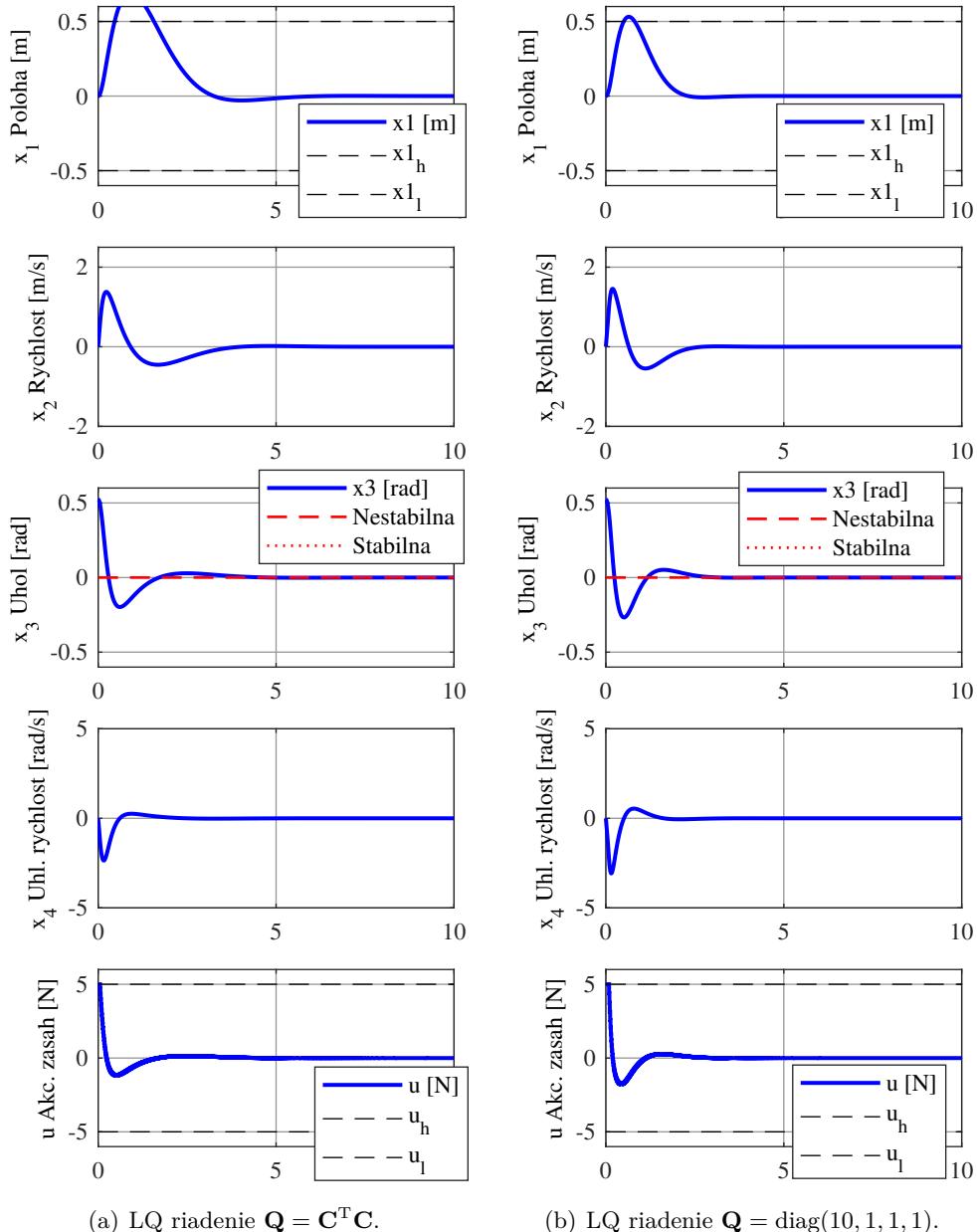
Príčinu nestability odozvy vidíme v grafe, ktorý predstavuje akčné zásahy. Na začiatku vidíme agresívne oscilujúce vstupy, ktorých priebeh sa končí na úrovni 5 N. Aj keď vypočítame ideálne naladené parametre PID regulátora pre linearizovaný model, realita môže byť značne odlišná. Hlavným kameňom úrazu je vlastné nasýtenie signálu, ktoré nie je uvažované pri návrhu PID riadenia a ani pri overení stability pomocou dobre známych metód. Samozrejme, nie je vylúčené, že iným ladením dostaneme stabilné aj použiteľné priebehy v spätej väzbe aj napriek saturácii vstupov. Ďalším zaujímavým faktom je aj samotná slučka spätej väzby. Všimnime si, že PID regulátor berie do úvahy iba uhlovú výchylku kyvadla. Stavový vektor tiež obsahuje ďalšie dôležité informácie: polohu a rýchlosť vozíka, uhlovú rýchlosť kyvadla, ale PID riadenie s nimi vôbec neuvažuje. Pristúpme teda k ďalšiemu spôsobu riadenia: LQ.

Po zmene prepínačov na režim LQ riadenia a spustení simulácie dostaneme odozvu znázornenú na Obr. 7.26(a). Na treťom grafe vidíme uhlovú výchylku kyvadla, z ktorej je zrejmé, že z odchýlky od hornej, nestabilnej polohy riadiaci systém úspešne priviedol kyvadlo do tejto žiadanej polohy. Akčné zásahy dosiahnu na začiatku simulácie maximálne dovolené hodnoty, ktoré sú dodržané vďaka nasýteniu signálu. Pozrime sa ďalej na výchylku vozíka. Ako vidíme, vozík za príbližne pol sekundy narazí na koniec dráhy – v tejto situácii väčšina laboratórnych modelov z bezpečnostných dôvodov vypne akčný člen (motor) a kyvadlo tak spadne dole. V tomto prípade máme teda logické obmedzenie stavu, bez rešpektovania ktorého nemôžeme systém reálne riadiť. Ako sa takáto situácia riesí v praxi?

Môžeme napríklad upraviť váhovaciu maticu stavu pri výpočte LQ zosilnenia. Konzervatívnejším váhovaním môžeme dosiahnuť funkčnú riadiacu slučku, ale za cenu zníženia celkovej kvality. Napríklad zmenou matice váhovania stavov na  $\mathbf{Q} =$



Obr. 7.25. Volná odozva invertovaného kyvadla a neúspešné PID riadenie s automaticky naladenými konštantami.



Obr. 7.26. LQ riadenie invertovaného kyvadla.

$\text{diag}([10 \ 1 \ 1 \ 1])$ . Výsledok simulácie je znázornený na Obr. 7.26(b). Maximálna výchylka vozíka sa už približuje k dovolenej hranici, ale stále mierne prevyšuje limit – vozík by stále narazil na koniec dráhy. Postupným ladením by sme určite dostali na hodnotu zosilnenia, pri ktorom vozík nenašazí na koniec dráhy.

Takáto zmena na konzervatívnejšie riadenie však zároveň prináša aj zhorsenie kvality riadenia. Čím konzervatívnejšie ladíme riadenie, tým dlhšie bude stabilizačia kyvadla trvať. Predpokladajme teraz, že pri určitej voľbe váhovacích matíc **Q** a **R** sa vozík bude pohybovať v povolených medziach. Je však takéto ladenie LQ regulátora ideálne pre náš systém a zároveň zaručí, že vozík sa bude vždy pohybovať na dovolenej dráhe? Žiaľ nie. Dané ladenie bude platíť iba pre počiatočnú polohu vozíka v strede dráhy a počiatočnú odchýlku kyvadla  $30^\circ$  od jeho nestabilnej polohy a menšie a aj to v absencii vonkajších porúch. Ak chceme uvažovať väčšie vychýlky, respektíve uvažujeme aj vplyv porúch, musíme váhovanie nastaviť ešte konzervatívnejšie. Ako vidíme, bežný konzervatívny spôsob riadenia v skutočnosti rigorózne nevyrieší dodržanie obmedzení stavov.

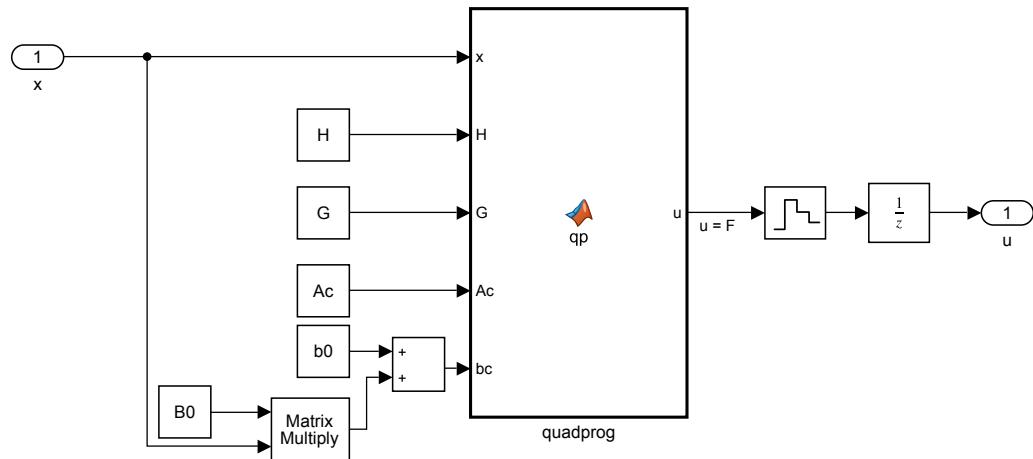
□

**Príklad 7.11.** Vytvorte simuláciu prediktívneho riadenia invertovaného kyvadla na základe Pr. 7.10. Uvažujte stavové obmedzenie na polohu vozíka  $-0.5 \leq x_1 \leq 0.5$  m a  $-5 \leq u \leq 5$  N na akčný zásah.

Vychádzajme z blokovej schémy spätnoväzbového riadenia, ktorá bola výsledkom Pr. 7.10. Vytvorme si prázdný podsystém, ktorý prijíma na vstupnej bráne aktuálny stav systému a na výstupnej bráne vráti akčné zásahy. Začneme vytvorením bloku na riešenie problému QP. Nájdime si v knižnici blok *Simulink* → *User-Defined Functions* → *MATLAB Function* a premiestnime ho do nového podsystému. Dvojitým kliknutím otvorme vlastný funkčný blok, v ktorom môžeme definovať úlohu riešenia QP.

Na implementáciu riešenia problému QP pomocou `quadprog()` v Simulinku môžeme použiť rôzne techniky. My si vytvoríme vlastnú funkciu, ktorá do funkčného volania riešiča `quadprog()` privedie aktuálny stav, účelovú funkciu a obmedzenia. Vstupnými argumentmi teda sú **x<sub>k</sub>**, **H**, **G**, **A<sub>c</sub>** a **B<sub>c</sub>**. Simulink potrebuje tiež poznáť rozmer výstupnej veličiny, a preto si ju na začiatku inicializujeme ako číslo – a tak bude program výstup považovať za skalár. Funkciu `quadprog()` dalej vylúčime z prekladu pomocou direktív `coder.extrinsic()`. Nasleduje samotné volanie riešiča `quadprog()`, ktorý vráti postupnosť  $\vec{u}_k^*$  do premennej **U**. Jej prvý prvok bude využitý ako akčný zásah:

```
function u = qp(x, H, G, Ac, bc) % Riesenie QP
u=0; % Inicializacia
```



Obr. 7.27. Podsystem na online riešenie problému MPC.

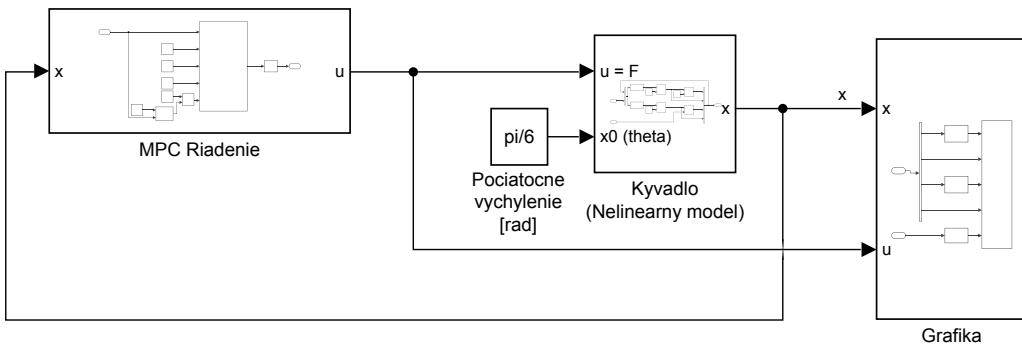
```

coder.extrinsic('quadprog'); % Neexistuje C verzia
U=quadprog(H,G*x,Ac,bc); % Volanie
u = U(1); % Výstup

```

Po zatvorení definičného skriptu funkcie dostaneme vlastný blok, ktorý bude očakávať stav  $\mathbf{x}_k$  a definičné matice  $\mathbf{H}$ ,  $\mathbf{G}$ ,  $\mathbf{A}_c$  a  $\mathbf{b}_c$  na vstupu. Výstup bloku môžeme pomocou tvarovača nultého rádu podržať a výsledný signál poslať na výstupnú bránu. Stav  $\mathbf{x}_k$  bude blok prijímať na základe vstupného signálu. Veličiny  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{A}_c$  závisia len na offline časti výpočtu MPC, a preto ich v prípade no-minálneho riadiaceho zákona môžeme načítať z pracovného priestoru MATLABu. Označme tieto matice premennými  $\mathbf{H}$ ,  $\mathbf{G}$  a  $\mathbf{A}_c$  a implementujme ich pomocou blokov konštant (*Simulink*→*Sources*→*Constant*). Posledný vstupný údaj bude potrebovať online prepočet na základe aktuálneho stavu  $\mathbf{x}_k$ , a preto musíme realizovať rovnicu  $\mathbf{b}_c = \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k$ . Matice  $\mathbf{b}_0$  a  $\mathbf{B}_0$  získame z offline výpočtu a skladujeme ich v premenných  $\mathbf{b}_0$  a  $\mathbf{B}_0$ . Maticové násobenie vykonáme pomocou bloku *DSP System Toolbox*→*Math Functions*→*Matrices and Linear Algebra*→*Matrix Operations*→*Matrix Multiply*, kde prvý vstup je signál z bloku konštanty  $\mathbf{B}_0$  a druhým je aktuálny stav. Výsledok sčítame so signálom z bloku konštanty  $\mathbf{b}_0$  pomocou bloku sčítania *Simulink*→*Math Operations*→*Add*. Výsledný tvar podsystemu je znázornený na Obr. 7.27.

Nezabudnime ani na číslicovú realizáciu skutočného algoritmu prediktívneho riadenia. Signál z algoritmu bude určite diskrétny, ale pretože v samotnej simulácii „miešame“ spojity model s diskrétnym riadením pre istotu môžeme vložiť do schémy blok tvarovača cez *Simulink*→*Discrete*→*Zero-Order Hold*.



Obr. 7.28. Bloková schéma spätnoväzbového riadenia invertovaného kyvadla pomocou prediktívneho riadenia s obmedzeniami.

Veľmi dôležitá pre realistickú simuláciu je tiež logika posunutia (oneskorenia) akčného zásahu. V procedurálnom programovacom jazyku máme v podstate dve možnosti: akčný zásah môžeme aplikovať ihneď po tom ako je dostupný z riešiča, čo by žiaľ vyžadovalo komplexnú zmienu návrhu algoritmu a samotnej teórie riadenia. Druhé a prakticky najčastejšie využívané riešenie je ponechať aplikáciu akčného zásahu na ďalšiu vzorku. Takto vložíme do obvodu blok oneskorenia, no na druhej strane platia naše predpoklady ohľadom vzorkovania. Pri dostatočne rýchлом vzorkovaní nebude oneskorenie vstupu hrať významnú rolu. Pre simuláciu v Simulinku má blok oneskorenia aj význam prevencie vytvorenia algebraických slučiek. Ak by sme daný blok nepoužili, mohla by nastať taká situácia, že akčný zásah by sme priamo potrebovali na výpočet akčného zásahu, napríklad ak by sme vložili do slučky Kalmanov filter. Blok *Simulink*→*Discrete*→*Unit Delay* do schémy teda vložíme ešte pred výstupom z podsystému.

Celková konfigurácia blokovej schémy je znázornená na Obr. 7.29. Všimnime si, že v prípade nulovej referencie online časť MPC predstavuje zdanivo jednoduchý vstupno-výstupný blok. Na zjednodušenie úlohy sme neuvažovali ani odhad stavu.

Parametre použité v blokovej schéme musia byť známe ešte pred spustením simulácie. Preto v *File*→*Model Properties*→*Model Properties*→*Callbacks*→*StartFcn* definujeme názov súboru na offline výpočet úlohy MPC, napríklad v našom prípade to môže byť *kyvadloOfflineMPC*, ktorý pred štartom simulácie spustí súbor *kyvadloOfflineMPC.m*.

Vytvorme si teraz offline časť algoritmu MPC. Ako zvyčajne, nahráme linearizovaný predikčný model a uložíme matice stavovej reprezentácie. Zistíme počet stavov a vstupov systému a načítame si aj vzorkovanie linearizovaného diskrétneho stavového modelu:

Zdrojový kód 7.11. kyvadloOfflineMPC.m

```
load kyvadlo.mat % Linearizovany model
A=kyvadlo.a; B=kyvadlo.b; C=kyvadlo.c;
[nx nu]=size(B); % Rozmery stavoveho modelu
Ts=kyvadlo.Ts; % Vzorkovacia perioda [s]
```

Definujme si tiež dĺžku horizontu predikcie. Vzorkovanie je v porovnaní s rýchlosťou dominantnej dynamiky systému malé, a preto si zvolíme horizont  $n_p = 50$ . Na začiatok predpokladajme váhovanie stavov a vstupu  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  a  $\mathbf{R} = R = 1$ :

```
np=50; % Predikcny horizont
Q=C'*C; % Vahovanie stavov
%Q=eye(nx); Q(1)=10; % Ine vahovanie
R=1; % Vahovanie vstupu
```

V offline časti algoritmu pokračujeme výpočtom penalizácie koncového stavu, matíci účelovej funkcie a symetrizáciou Hessiánu:

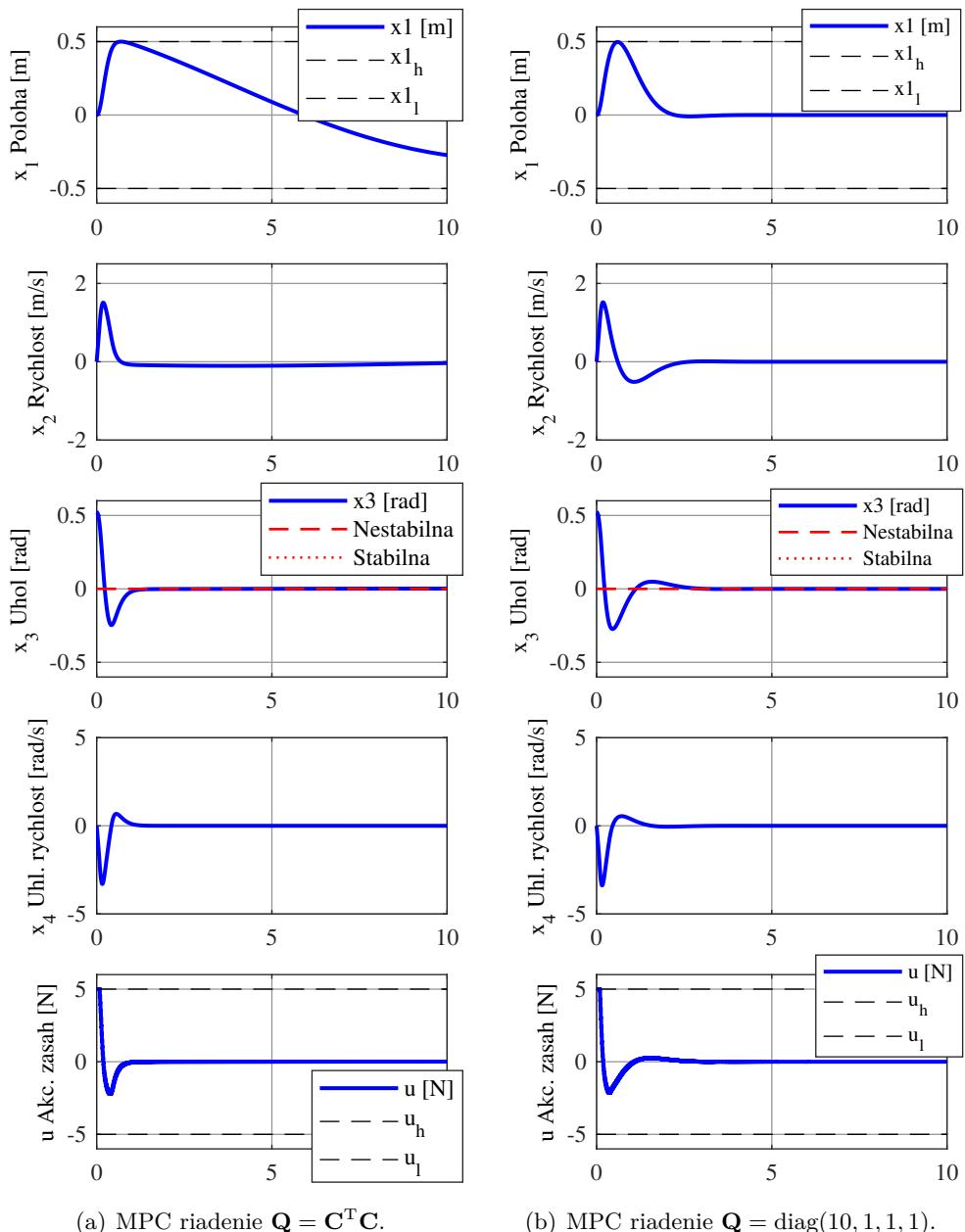
```
[K, P]=iterdlqr(A, B, Q, R, 100); % Vah. koncov. stavu
[H, G]=ucelovafunkcia(A, B, np, Q, R, P); % Ucelova funkcia
H=(H+H')/2; % Symetrizacia H
```

Pokračujeme deklaráciou a výpočtom obmedzení. Vytvoríme si premenné na uloženie matíci obmedzení vstupu. Definujeme maximálnu a minimálnu dráhu vozíka. Ak nechceme obmedzenia na ostatné stavy brať do úvahy pri výpočte akčného zásahu, môžeme zadat aj nereálne veľké hodnoty. Nakoniec zavoláme našu vlastnú funkciu na vytvorenie obmedzení:

```
ul=-uh; % Min. sila na vozik [N]
x1h=0.50; % Max. draha vozika [m]
x1l=-x1h; % Min. draha vozika [m]
xh=[x1h 1E8 1E8 1E8]'; % Obmedzenia stavov
x1=-xh; % Symetricke
[Ac b0 B0]=obmedzenia (ul, uh, x1, xh, np, A, B); % Obmedzenia
```

Tým sme ukončili výpočet parametrov potrebných na simuláciu v Simulinku.

Po spustení simulácie v Simulinku ihneď môžeme sledovať výsledky simulácie, ktorých získanie je významne pomalšie ako napríklad výsledky simulácie LQ riadenia. Dôvodom je to, že riešenie QP s dlhým horizontom predikcie je výpočtovo náročnejšia optimalizačná procedúra, na ktorú potrebuje náš počítač využiť značnú časť výkonu. Je preto pravdepodobné, že 10–sekundová simulácia bude v skutočnosti prebiehať oveľa dlhšie ako desať sekúnd. Čo to znamená? Počítač by



Obr. 7.29. Prediktívne riadenie invertovaného kyvadla s obmedzením polohy vozíka a akčného zásahu.

nevedel realizovať túto simuláciu v reálnom čase – teda simulácia je pomalšia ako realita. Príčna, okrem zložitosti úlohy, je zjavná: volanie funkcie `quadprog()` zo Simulinku bez prekladu do exekutívne je neefektívne. Simuláciu by sme mohli značne zrýchliť, ak by sme zvolili iný typ riešiča, ktorý má dostupný zdrojový kód pre preklad do exekutívne.

Po spustení simulácie získame priebehy znázornené na Obr. 7.29(a). Začnime s analýzou uhlovej výchylky kvyadla, ktorá je z počiatočnej odchýlky veľmi rýchlo privedená do nestabilnej hornej polohy. Vstupy rešpektujú obmedzenia aj bez saturácie, keďže algoritmus ich zohľadňuje priamo pri riešení optimalizačnej úlohy. Najzaujímavejší priebeh vidíme na hornom grafe, ktorý ukazuje priebeh výchylky vozíka. Hned na začiatku vidíme, že stavové obmedzenie je dokonale rešpektované – vozík teda nemôže naraziť na koniec dráhy. Aj keď samotné kvyadlo je už stabilizované, vozík je stále v pohybe, dokonca z jedného konca dráhy prejde na druhý. Takýto pohyb je prirodzený a očakávaný; ak by sme si ho neželali, môžeme zmeniť váhovanie.

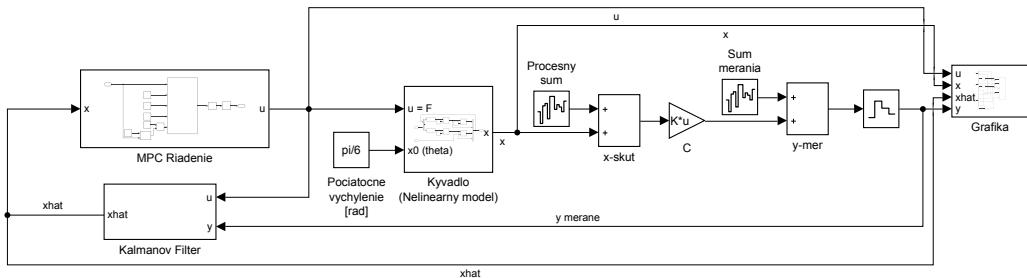
Po zmene matice váhovania stavov na  $\mathbf{Q} = \text{diag}([10 \ 1 \ 1 \ 1])$  a spustení simulácie získame odozvu na Obr. 7.29(b). Kvyadlo je stabilizované trochu pomalšie, ale na druhej strane sa vozík vráti do nulovej polohy rýchlejšie.

□

**Príklad 7.12.** Vytvorte blokovú schému prediktívneho riadenia invertovaného kvyadla na základe Pr. 7.11 s predpokladom, že k dispozícii namiesto plnej informácie o stave máme iba zašumené meranie výchylky vozíka a uhlovej výchylky kvyadla. Rozptyl procesného šumu bol odhadnutý na úrovni  $\sigma_w = [0 \ 0 \ 0 \ 1E-5]^T$  bez vzájomnej korelácie jednotlivých stavov. Rozptyl šumu merania výchylky vozíka je  $\sigma_{v_1} = 1E-3$  a uhlovej výchylky kvyadla je  $\sigma_{v_2} = 1E-3$  bez vzájomnej korelácie signálov zo snímačov. Predpokladajte maximálny akčný zásah na úrovni  $\pm 6$  N.

Vychádzame z hotového Pr. 7.11 a ďalej uvedieme iba zmeny potrebné na zahrnutie vplyvu zašumeného merania a procesu odhadu do simulácie. Náš simulačný model invertovaného kvyadla je nelineárny, pričom na riadenie použijeme predikčný model získaný jeho linearizáciou v pracovnom bode. Predpokladajme, že systém sa pri riadení bude pohybovať v okolí pracovného bodu, a teda stavy môžeme odhadnúť aj pomocou lineárneho Kalmanovho filtra, t. j. nepotrebujeme rozšírený Kalmanov filter.

Samotný podsystém na riešenie MPC necháme nezmenený. Do hlavnej schémy najprv vložíme blok Kalmanovho filtra, ktorý nájdeme v knižnici Simulinku pod *Control System Toolbox*→*State Estimation*→*Kalman Filter*. Dvojitým kliknutím otvoríme dialógové okno na nastavenie parametrov Kalmanovho filtra. Ďalej si v záložke pod *Block Parameters*: *Kalman Filter*→*Model Parameters*→*Model Source*



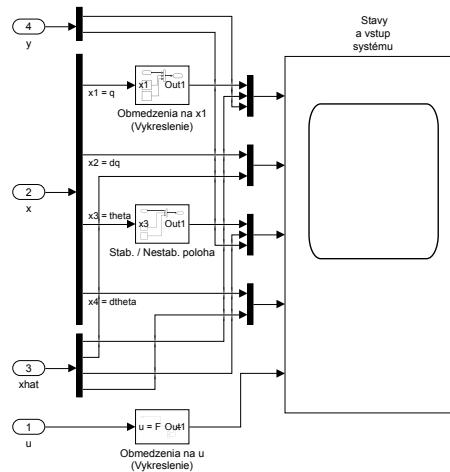
Obr. 7.30. Bloková schéma simulácie MPC riadenia invertovaného kyvadla so simuláciou zašumenej merania.

→ Individual  $A, B, C, D$  matrices zvolíme stavový model, t. j. zadáme premenné  $A$ ,  $B$  a  $C$ , v ktorých sú uložené definičné matice linearizovaného diskrétneho stavového modelu. V časti *Noise Characteristics* môžeme deklarovať aj kovariančné matice procesného šumu a merania. Označme si kovariančnú maticu procesného šumu premennou  $Q_f$  a kovariančnú maticu merania premennou  $R_f$ . Ostatné parametre môžeme ponechať ako predvolené.

Výstup z bloku bude odhadnutý stav  $\hat{x}_k$ , ktorý si priviedieme na vstup bloku MPC riadenia. Blok Kalmanovho filtra potrebuje poznáť akčné zásahy prichádzajúce do dynamického systému. Pripojme teraz signál z výstupu bloku prediktívneho riadenia na vstupníbránu  $u$  Kalmanovho filtra. V predchádzajúcom príklade sme vložili do bloku MPC riadenia jednokrokové oneskorenie, čo má v tejto simulácii dôležitý význam. Ak by sme ho vynechali, vytvorili by sme algebraickú slučku, pretože MPC potrebuje stav na výpočet vstupu a Kalmanov filter zároveň potrebuje vstup na výpočet stavu.

Kalmanov filter má druhú vstupnú bránu  $y$ , na ktorej v tomto prípade očakáva zašumený signál merania polohy vozíka a uhlovej výchylky kyvadla. Na emuláciu týchto javov využijeme „skutočný“ stav z nelineárneho modelu. Zoberieme signál stavu a priviedieme ho na jednu zo vstupných brán bloku sčítania (*Simulink*→*Math Operations*→*Add*). Ďalej si v knižnici pod *Simulink*→*Band-Limited White Noise*→*Add* nájdeme blok na generovanie signálu diskrétneho bieleho šumu. Dvojitým kliknutím zadáme parameter „výkon“ signálu ako  $w\_var \cdot Ts$ , ktorý predstavuje rozptyl procesného šumu na jednu vzorku. Vzorkovanie si zvolíme tradične ako  $Ts$ , pričom generátor náhodných čísel môžeme naštartovať s jadrom 23341. Signál, ktorý vychádza z bloku sumácie predstavuje stav systému sčítaný s procesným šumom.

Stav zaľažený procesným šumom si ďalej násobením maticou  $C$  transformujme na reálne meranie, čo realizujeme pridaním bloku zosilnenia na signál.



Obr. 7.31. Odporúčaný podsystém na vizualizáciu výsledkov simulácie.

Teraz nasleduje pridanie šumu merania, čo vyžaduje identické bloky ako v prípade procesného šumu. Preto myšou vyznačíme bloky *Add* a *Band-Limited White Noise* a vytvoríme novú kópiu. Do jednej zo vstupných brán bloku sčítania si prividieme hypoteticky dokonalé meranie. Na druhú vstupnú bránu bloku sčítania prividieme šum merania. Blok otvoríme a zadáme  $v\_var \star Ts$  ako rozptyl šumu merania a  $Ts$  ako vzorkovanie. Ako jadro generátora náhodného signálu uvažujme 12347. V podstate nie je dôležité akú sekvenciu použijeme na naštartovanie generátora náhodných čísel, ale charakter procesného šumu a šumu merania by nemal mať vzájomný súvis, a preto zadáme rôzne jadrá generátora. Výstup z bloku sumácie teraz predstavuje signál zo snímačov, ktorý je zaľažený šumom merania. Tento signál podržíme počas jednej vzorky, čím simulujeme proces diskrétneho merania. Signál zo snímačov si ďalej prividieme na vstupnú bránu y Kalmanovho filtra. Finálna schéma zo súboru *kyvadloMPCaKF.slx* je znázornená na Obr. 7.30.

Zostáva nám ešte vykreslenie výsledkov, kde odporúčaný podsystém je znázorený na Obr. 7.31. V prípade každého známeho stavu môžeme pridať príslušný odhad. Preto pridáme ďalšiu vstupnú bránu (*Simulink*→*Ports & Subsystems*→*In1*) pre odhady. Signál si rozložíme na štyri zložky pomocou demultiplexera *Simulink*→*Signal Routing*→*Demux*. Každý z odhadnutých stavov môžeme teraz zlúčiť pomocou multiplexera *Simulink*→*Signal Routing*→*Mux* na vstupnú bránu osciloskopu.

K dispozícii máme aj zašumený signál zo snímačov. Zašumený signál merania výchylky vozíka prividieme na prvý graf prislúchajúci stavu  $x_1$  a signál z inkrementálneho snímača na tretí graf prischlúchajúci stavu  $x_3$  virtuálneho osciloskopu.

Signál vstupuje do subsystému prostredníctvom vstupnej brány, je rozložený na dve zložky pomocou demultiplexera a následne privezený pomocou multiplexera na zbernicu osciloskopu podobne ako v prípade odhadnutého stavu.

Nakoniec nám ostáva iba definícia zdroja premenných. Pre tento účel môžeme zmeniť súbor, ktorý má byť spustený na začiatku simulácie. Nájdime *File→Model Properties→Model Properties→Callbacks* pod záložkou *StartFcn* zadajme príkaz *kyvadloOfflineMPCaKF*. Takto naša bloková schéma pred štartom spustí súbor *kyvadloOfflineMPCaK.m*.

Vytvorime teraz súbor *kyvadloOfflineMPCaK.m*, ktorý bude založený na predošлом príklade, preto tu uvedieme iba niekoľko zmien. Zadanie príkladu obsahuje podmienku obmedzenia hodnoty akčného zásahu na 6 N, a preto horný limit vstupu zmeníme na  $uh=6$  N<sup>17</sup>. Iné zmeny programu nevyžaduje, pokračujeme iba definíciou rozptylu šumu a kovariančných matíc pre Kalmanov filer:

Zdrojový kód 7.12. *kyvadloOfflineMPCaKF.m*

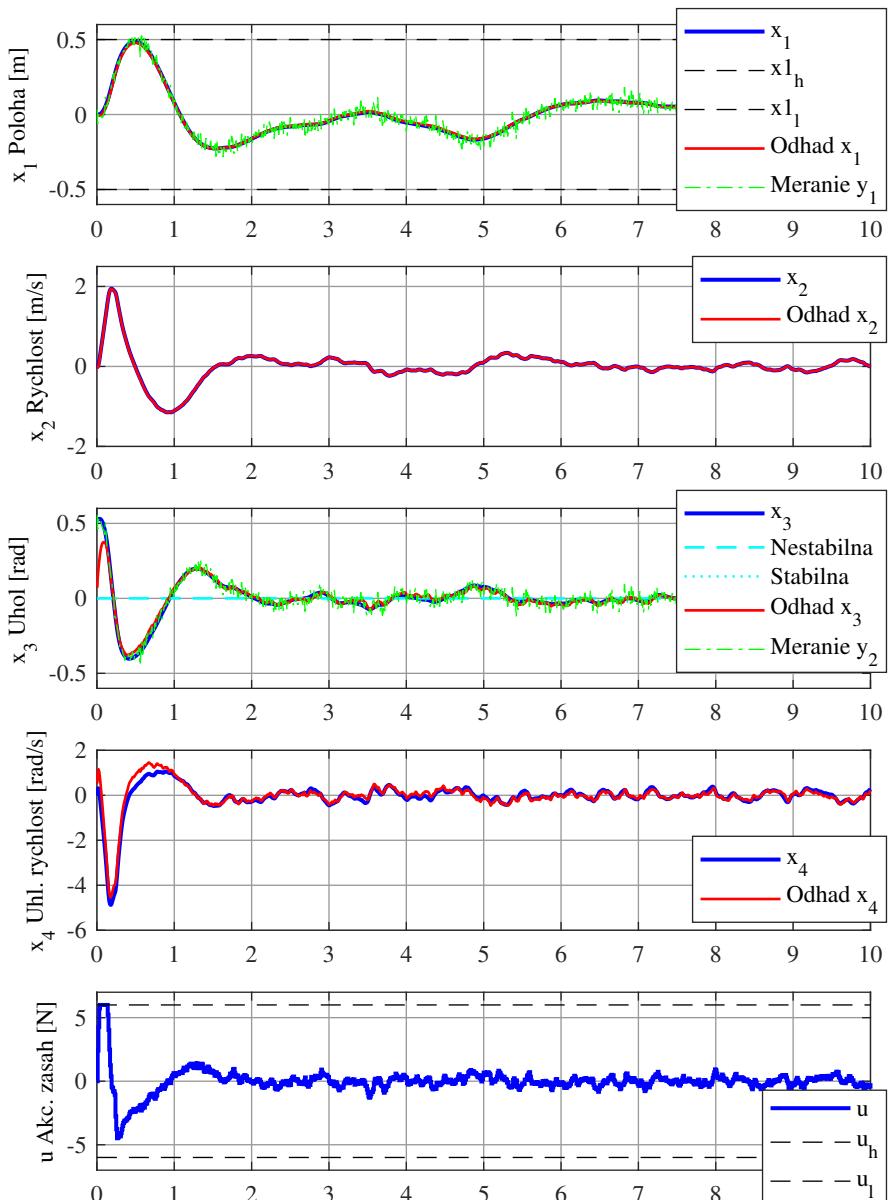
```
w_var=[0 0 0 1e-5]'; % Rozptyl procesneho sumu
v_var=[1e-3 1e-3]'; % Rozptyl sumu merania
Qf=diag(w_var); % Kovariancna matica proces. sumu
Rf=diag(v_var); % Kovariancna matica sumu merania
```

Po spustení simulácie získame odozvu na Obr. 7.32. Podobnosť bude závisieť najmä od správneho vytvorenia podsystému grafického zobrazenia a od nastavenia farieb signálov. Prvý graf znázorňuje výchylku vozíka. Na základe merania zo snímača je poloha meraná veľmi nespolahlivo. Odhad Kalmanovho filtra napriek tomu takmer dokonale kopíruje skutočný vnútorný stav dynamického systému.

V prípade merania uhlovej výchylky kyvadla máme tiež k dispozícii silne zašumený signál zo snímača. Napriek tomu Kalmanov filter dokázal odhadnúť skutočný priebeh uhlovej výchylky kyvadla. Na začiatku, približne v čase 0 až 0.2 s odhad a skutočný stav divergujú. To je očakávané, pretože Kalmanov filter sme inicializovali na začiatku simulácie bez bližšej špecifikácie kovariančnej matice odhadu stavov. Táto časovo premenlivá matica je však v priebehu niekoľkých krokov ustálená. V praxi môžeme tiež proces odhadu naštartovať s vopred vypočítanou maticou kovariancie odhadu stavov.

Prednostou Kalmanovej filtrácie je aj to, že napriek tomu, že poznáme iba dva zo štyroch dynamických stavov systému, nemerané stavy algoritmus dokáže verne zrekonštruovať. Toto je prípad rýchlosťi vozíka a uhlovej rýchlosťi kyvadla (druhý a štvrtý graf). Zo zašumených signálov sme tak zrekonštruovali aj vlastnú dynamiku systému.

<sup>17</sup>Táto zmena súvisí so zachovaním zlučiteľnosti obmedzení, pozri nasledujúcu kapitolu.



Obr. 7.32. Prediktívne riadenie invertovaného kyvadla s obmedzeniami výchylky vozíka a akčného zásahu pri neúplnej informácii o stave v prítomnosti šumu.

Z pohľadu spätnoväzbového riadenia algoritmus MPC efektívne vráti kyvadlo do hornej, nestabilnej rovnovážnej polohy. Tak ako v prípade skutočného laboratórneho modelu invertovaného kyvadla, je pravdepodobné, že regulátor musí stále aplikovať určité malé akčné zásahy na to, aby udržiaval systém v danej polohe. V tomto prípade algoritmus riadenia reaguje na rôzne šumy v systéme, čo je súmožejme ľahko riešiteľné pomocou jemnejšieho ladenia regulátora alebo použitím filtrov. Tak ako v predchádzajúcim príklade, akčné zásahy rešpektujú maximálnu dovolenú hodnotu, čo vidíme aj na začiatku simulácie. Ešte dôležitejšie pre prípad invertovaného kyvadla je, že vozík počas celej simulácie zostáva na predpísanej dráhe: prediktívne riadenie umožnilo implementáciu stavových obmedzení.

□

V tejto kapitole sme už získali potrebné zručnosti, pomocou ktorých vieme vytvoriť vlastný prediktívny regulátor systému s procesnými obmedzeniami. Prístup a algoritmus, ktorý sme uvažovali v tejto učebnici zodpovedá moderným trendom a je široko akceptovaný v akademickej ako aj priemyselnej oblasti. Vieme teda už o prediktívnom riadení všetko?

Žiaľ nie. Doteraz sme sa vôbec nedotkli otázky stability. Analýza stability je dôležitou súčasťou návrhu spätnoväzbových riadiacich obvodov. Klasická výuka teórie riadenia kladie veľký dôraz na vyšetrenie stability lineárnych spätnoväzbových riadiacich systémov. Už vieme, že nasýtenie signálu prináša do regulačného obvodu nelinearitu, a teda už nemôžeme zaručiť jeho stabilitu a neplatia ani dobre známe poučky o umiestnení pólov prenosu alebo vlastných čísel matice dynamiky systému. Prediktívne riadenie s obmedzeniami neobsahuje prvok nasýtenia, ale nelinearita je stále prítomná. Dokonca tu nejde ani o uzavretú formu riadiaceho zákona, a teda „póly prenosu“ nevieme ani vypočítať.

Prediktívne riadenie s obmedzeniami nezaručuje stabilitu riadenia. Môžeme si zvoliť príliš krátky horizont alebo príliš prísne obmedzenia a ľahko sa dostaneme situácie, že pri riadení môžeme stratíť stabilitu spätnoväzbovej slučky riadenia. Dokonca v Kap. 6 sme sa stretli tiež s výrazom zlučiteľnosti obmedzení. Problém MPC nie je ani zaručene zlučiteľný, to znamená, že môže ľahko nastať taká situácia, kedy je daný problém QP neriešiteľný. V ďalšej kapitole sa preto pozrieme na stabilitu a zlučiteľnosť úlohy MPC s obmedzeniami.

## Kapitola 8

# Stabilita a riešiteľnosť MPC

Stabilita nepredstavuje iba nejakú abstraktnú teoretickú vlastnosť spätnoväzbových riadiacich systémov. Nestabilný regulátor v spätej väzbe môže totiž aplikovať na systém nevhodný vstup až do bodu, keď nastane zlyhanie akčných členov alebo dokonca dôjde k poškodeniu samotného riadeného systému. Takéto nechcené správanie nemusí predstavovať iba materálne škody a poškodenie vybavenia, ale aj nebezpečenstvo potenciálneho zranenia, ba dokonca ohrozenia života.

Jedným z klasických didaktických systémov v oblasti automatického riadenia je invertované kyvadlo, ktoré sme často využívali ako v príklad v Kap. 2. Tento systém, zobrazený na Obr. 2.1, pozostáva z motorom poháňaného lineárneho vedenia s vozíkom, na ktorom je rotačne uchytené volne sa pohybujúce kyvadlo. Ako už vieme, snahou riadenia je tu vyšvihnuť kyvadlo do jeho hornej rovnovážnej polohy a potom ho v nej aj udržať prostredníctvom pohybu vozíka. Tento systém je iba čiastočne ovládaný, nelineárny a je tiež zrejmé, že horná rovnovážna poloha kyvadla je v otvorennej slučke, t. j. bez riadenia, nestabilná. Stabilita spätnoväzbového riadenia je preto v tomto aplikačnom príklade zásadná a prípadná strata stability sa prejaví úplne zretelne – kyvadlo spadne a dostane sa do svojej dolnej, stabilnej rovnovážnej polohy. Napriek tomu, že problémy so stabilitou riadenia nie sú v prípade rôznych riadených systémov podobne hmatateľné a zjavné, je dôležité vedieť zaručiť (garantovať), že systém v uzavretej slučke s prediktívnym regulátorom, je a tiež zostane vždy stabilný.

Pravdepodobne najjednoduchšou definíciou stability je tzv. stabilita v zmysle „ohraničený vstup  $\rightarrow$  ohraničený výstup“<sup>1</sup>, ktorá hovorí, že odozva systému na ohraničený vstupný signál je takisto ohraničená. V opačnom prípade, ak má výbudenie systému ohraničeným vstupom za následok neohraničenú odozvu na výstupu, znamená to, že daný systém je nestabilný. Stabilitu tradičných riadiacich

---

<sup>1</sup>angl. bounded input – bounded output (BIBO) stability

systémov pritom môžeme garantovať veľmi jednoducho. V prípade diskrétneho regulátora v uzavretom tvari sa všetky póly prenosovej funkcie (resp. vlastné čísla matice dynamiky v stavopriestorovej reprezentácii) spätnoväzbového regulačného obvodu musia nachádzať vo vnútri jednotkovej kružnice (viď Kap. 2.3). Ako si však ďalej ukážeme, v prípade MPC s obmedzeniami to už nebude také jednoduché.

Ak v probléme MPC s účelovou funkciou definovanou na nekonečnom horizonte neuvažujeme žiadne obmedzenia, predikované a skutočné trajektórie stavov a vstupov budú identické<sup>2</sup> a hodnota kritéria konečná; a teda bude zaručená stabilita daného procesu. Takýto problém je výpočtovo zvládnuteľný, keďže si vieme jednoducho vypočítať napríklad ustálené stavové LQ zosilnenie na základe účelovej funkcie na nekonečnom horizonte. Poznamenajme, že predikované a skutočné odozvy by boli teoreticky totožné aj v prípade ohraničeného MPC problému za predpokladu, že použijeme nekonečne dlhý predikčný horizont a príslušnú nekonečnú účelovú funkciu. Tento predpoklad by síce viedol na proces so zaručenou stabilitou, avšak nekonečne dlhý horizont by logicky vyžadoval nekonečný počet optimalizačných premenných v probléme MPC, čo samozrejme nie je prakticky možné. Namiesto uvažovania nekonečného horizontu predikcie môžeme však použiť konečný horizont s konečným počtom optimalizačných premenných v kombinácii s váhovaním koncového stavu nahradzajúcim časť účelovej funkcie od konca horizontu do nekonečna – inými slovami riadenie so zloženým riadiacim zákonom predstavené v Kap. 4. Nanešťastie, takáto úvaha na druhej strane komplikuje otázkou stability, pretože dané ohraničenia uvažujeme len na konečnom horizonte, zatiaľ čo zlučiteľnosť ohraničení a v konečnom dôsledku aj stabilita a rekurzívna riešiteľnosť<sup>3</sup> MPC problému nie je zaručená. V prípade určitých rušení potom môže regulátor vypočítať takú postupnosť akčných zásahov, ktorá má za následok nestabilnú odozvu s potenciálne neželanými následkami.

Táto kapitola sa preto bude venovať problematike stability prediktívneho riadenia, pričom jej hlavným cieľom je oboznámiť čitateľa so základmi potrebnými na formuláciu moderného MPC riadiaceho zákona, ktorý okrem optimality umožní zaručiť stabilitu ako aj riešiteľnosť daného MPC problému. Pre čitateľa hľadajúceho podrobnejšie informácie o stabilite a alternatívnych spôsobov jej zaručenia, odporúčame práce [107, 74, 18, 19].

## 8.1 Vývoj MPC so zaručenou stabilitou

Prvé formulácie prediktívneho riadenia neboli schopné zaručiť stabilitu spätnoväzbového riadiaceho systému. To však neodradilo profesionálov z priemyslu od

<sup>2</sup>Samozrejme za predpokladu, že neuvažujeme chyby modelovania alebo vonkajšie rušenia.

<sup>3</sup>angl. recursive feasibility

používania MPC v praktických aplikáciách, kde uvažovanie v otvorenej slučke stabilných systémov a dlhých predikčných horizontov v neprítomnosti obmedzení umožnili vyhnúť sa väčšine problémov so stabilitou; podmienky zaručenia stability v striktne teoretickom ponímaní však neboli formulované.

Stabilita MPC bola v počiatkoch vyšetrovaná pre prípad prediktívnych regulátorov s kvadratickou účelovou funkciou bez obmedzení [53]. Pre daný regulátor sa v podstate iba zhodnotil vplyv dĺžky horizontu a voľby ladiacich parametrov, na základe čoho sa určilo, či je stabilizujúci alebo nie. Neskôr sa tento priamočiary prístup ukázal ako nepostačujúci [15], keďže viaceré príklady poukázali na potrebu určitej apriornej metódy zaručenia stability [135]. V súčnosti si preto ďalej predstavme základné nástroje garancie stability, medzi ktoré patria:

- koncové obmedzenia (obmedzenia koncového stavu) v tvare rovnosti;
- váhovanie (penalizácia) koncového stavu;
- cieľové (koncové) množiny;
- kombinácia cieľovej množiny a váhovania koncového stavu;
- podmienka kontraktivity (zmršťovania) stavu a iné.

### 8.1.1 Koncové obmedzenia v tvare rovnosti

Riešenie otázky zaručenia stability lineárnych systémov s obmedzeniami bolo neskôr navrhnuté prostredníctvom takzvaných koncových obmedzení. Tieto koncové obmedzenia predstavovali konkrétnu požiadavku na regulátor a sice takú, že stavy systému sa musia v určitom počte krokov za horizontom  $n_p$  rovnať nule. Na konci horizontu a za ním teda uvažujeme, že stavy sú nulové, pričom vstupy uvažujeme taktiež ako nulové [89]. Matematicky môžeme túto podmienku zapísat ako:

$$\mathbf{x}_{k+i} = \mathbf{0} \quad \text{pre } i \geq n_p \quad \text{a} \quad \mathbf{u}_{k+i} = \mathbf{0} \quad \text{pre } i \geq n_p. \quad (8.1)$$

Na definovanie koncových obmedzení môžeme tiež použiť alternatívnu interpretáciu: vyžadujeme, aby bol koncový stav  $\mathbf{x}_{k+n_p} = \mathbf{0}$  a všetky nasledujúce stavy súčasťou tzv. cieľovej (koncovej) množiny<sup>4</sup>, a teda  $\mathbf{x}_{k+n_p} \in \Omega$ . V tomto prípade je cieľová množina v podstate zhodná s počiatkom, t. j.  $\Omega = \{\mathbf{0}\}$ .

S koncovým obmedzením v tvare rovnosti a kvadratickou účelovou funkciou je možné analyticky zaručiť stabilitu lineárnych systémov bez obmedzení, ak dokážeme, že daná účelová funkcia nie je rastúca. Publikácia [75] sa stala v podstate základom pre ďalšie stabilizujúce MPC prístupy. Autori v nej navrhli ohraničený MPC regulátor využívajúci uvedenú myšlienku pre nelineárne časovo premenlivé

<sup>4</sup>angl. terminal, target set

systémy, poukazujúc na to, že účelová funkcia MPC s konečným horizontom sa so zväčšujúcim horizontom blíži účelovej funkcie na nekonečnom horizonte. Niekoľko ďalších prác navrhlo určité varianty ako aj nové algoritmy, ktoré v podstate využívajú uvedenú základnú myšlienku koncových obmedzení v tvare rovnosti, ako napr. práce [34, 112, 81]. Účelová funkcia pri týchto „ranných“ metódach bola uvažovaná ako konečná, t. j. definovaná iba na konečnom horizonte predikcie, pričom jej zvyšková časť za horizontom predikcie ešte nebola uvažovaná.

### 8.1.2 Váhovanie koncového stavu

Jedným z prvých pokusov zaručenia stability regulátora je tiež použitie matice váhovania koncového stavu,  $\mathbf{P}$ . Ako už vieme z Kap. 4, váhovanie koncového stavu, ktoré sa pridáva k účelovej funkcií, je matematicky dané ako  $J_T = \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}$ . Taktiež už vieme, že najlogickejšou voľbou matice  $\mathbf{P}$  je zvoliť ju ako riešenie Ljapunovej rovnice (4.50) pre zvolený stabilizujúci riadiaci zákon, čo následne zabezpečí stabilitu prostredníctvom nerastúcej účelovej funkcie  $J$ . Takáto voľba matice  $\mathbf{P}$  v podstate zaručí, že pridanie výrazu  $\mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}$  k účelovej funkcií definovanej na konečnom horizonte,  $J_\infty = J_{n_p}$ , zahrnie hodnotu  $J$  pre  $n_p \rightarrow \infty$ , čím súhrne získame účelovú funkciu na nekonečnom horizonte, t. j. platí  $J_\infty = J_{n_p} + J_T$ . Matematicky toto môžeme vyjadriť ako:

$$\mathbf{x}_{k+i} \rightarrow \mathbf{0} \quad \text{pre } i \rightarrow \infty \quad \text{a} \quad \mathbf{u}_{k+i} \rightarrow \mathbf{0} \quad \text{pre } i \geq n_p, \quad (8.2)$$

čo znamená, že stavy a vstupy systému sa v čase idúcim do nekonečna musia blížiť k nule. Samotné koncové váhovanie vyžaduje, aby stav pritom zostal v cieľovej množine, ktorá je nekonečne veľká a teda rovná celému stavovému priestoru, t. j.  $\Omega = \mathbb{R}^{n_x \times n_x}$ . Ak je riadený systém nestabilný, do konca horizontu predikcie musia byť jeho nestabilné módy rovné nule, pričom daný prístup je použitá pre zostávajúce stabilné módy systému.

### 8.1.3 Cieľové množiny

Ako intuitívne tušíme, obmedzenie na koncový stav v tvare rovnosti je pre väčšinu aplikácií príliš prísne, pretože výrazne obmedzuje množinu možných počiaťočných podmienok, z ktorých je možné priviesť systém v konečnom počte krokov do rovnovážneho stavu. Obmedzenie v tvare rovnosti má tiež spravidla za následok regulačný pochod s príliš agresívnymi akčnými zásahmi. Táto požiadavka bola preto neskôr nahradená použitím spomínaných koncových, resp. cieľových množín, ktoré predstavujú dodatočné obmedzenie v tvare nerovnosti. Cieľová množina  $\Omega$  je v tomto prípade zvolená ako konečná podmnožina stavového priestoru, t. j.  $\Omega \subseteq \mathbb{R}^{n_x \times n_x}$ . Snahou regulátora teda bolo priviesť stav systému do tejto množiny

v konečnom počte krokov. Hodnota účelovej funkcie za týmto konečným horizontom bola však v uvedenom prístupe stále uvažovaná ako nula, t.j.  $J_T = 0$ .

Namiesto požiadavky, aby stavy a vstupy vo vnútri cieľovej množiny  $\Omega$  boli nulové, ako v prípade obmedzení v tvaru rovnosti, prevzal úlohu riadenia lokálnej stabilizujúci riadiaci zákon. Priebeh vstupov vo vnútri cieľovej množiny,  $\mathbf{u}_{k+i}$ ,  $i \geq n_p$  tak už neuvažujeme ako nulový; namiesto toho stav do počiatku (rovnovážneho stavu) pre čas idúci do nekonečna priviedie spätnoväzbový stavový riadiaci zákon s ustáleným zosilnením  $\mathbf{K}$ . Matematicky to môžeme zapísť takto<sup>5</sup>

$$\mathbf{x}_{k+i} \rightarrow \mathbf{0} \quad \text{pre } i \rightarrow \infty \quad \text{a} \quad \mathbf{u}_{k+i} = \mathbf{Kx}_{k+i} \quad \text{pre } i \geq n_p. \quad (8.3)$$

Tento koncept sme si predstavili v Kap. 4 ako MPC so zloženou účelovou funkciou, kde v režime 1 vstupy uvažujeme ako voľné optimalizačné premenné a režim 2 predstavuje pevný spätnoväzbový riadiaci zákon v uzavretom tvaru. Ako už tiež vieme, ako zosilnenie lokálneho stabilizačného riadenia si najčastejšie volíme zosilnenie lineárneho kvadratického regulátora,  $\mathbf{K}$ .

#### 8.1.4 Kombinácia cieľovej množiny a váhovania koncového stavu

Stabilizujúca formulácia MPC používaná v súčasnosti je kombináciou konceptov opísaných v predchádzajúcich dvoch podkapitolách: váhovania koncového stavu a cieľových množín. V tomto prípade je koncová časť účelovej funkcie nenulová, a celková predikovaná hodnota účelovej funkcie sa preto rovná hodnote účelovej funkcie na nekonečnom horizonte, t.j.  $J_\infty = J_{n_p} + J_T$ . Stav systému je navyše nútene privedený do cieľovej množiny, kde riadenie prevezme stabilizujúci spätnoväzbový riadiaci zákon podľa (8.3). Tento prístup si podrobnejšie rozvinieme vo zvyšnej časti tejto kapitoly, pričom si pripomeňme, že prístup riadenia so zloženým horizontom bol už vysvetlený v Kap. 4. Aby sme si zhrnuli základy moderných formulácií MPC s obmedzeniami a zaručenou stabilitou, môžeme povedať, že využívajú nasledujúce koncepty:

- Stabilizujúci riadiaci zákon  $\mathbf{u}_{k+i} = \mathbf{Kx}_{k+i}$  pre  $i \geq n_p$ , vedúci na riadenie so zloženým horizontom; ktorý uvažuje použitie nenulových vstupov za predikčným horizontom  $n_p$ .
- Cieľová množina  $\Omega$  predstavujúca obmedzenie koncového stavu  $\mathbf{x}_{k+n_p}$ , ktorá má vlastnosť pozitívnej invariantnosti [74], t.j. jednoducho povedané: ak raz do nej stav vojde, už ju nemôže opustiť.
- Koncové váhovanie  $J_T = \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}$ , ktoré umožňuje zohľadniť celkovú účelovú funkciu  $J_\infty = J_{n_p} + J_T$ , zodpovedajúcu nekonečne dlhému predikčnému horizontu.

<sup>5</sup>Pri  $\mathbf{u}_{k+i} = \mathbf{Kx}_{k+i}$ ,  $i \geq n_p$  uvažujme, že záporná spätná väzba je už zahrnutá v zosilnení  $\mathbf{K}$ .

V neprítomnosti obmedzení môže voľba koncového váhovania viesť na účelovú funkciu na nekonečnom horizonte, čo predstavuje ideálnu situáciu – stabilita je zaručená a nie je potrebná priebežná (online) optimalizácia. Táto situácia je v podstate základom prediktívneho riadenia bez obmedzení opísaného v Kap. 5. Zavedením vstupných, stavových a procesných obmedzení na systém sa však predikovaná hodnota účelovej funkcie,  $J_\infty$ , odlišuje od jej skutočnej hodnoty získanej pri realizácii riadenia v uzavretej slučke. Spôsobom ako ošetrí túto skutočnosť je v tomto prípade uvažovanie cielovej množiny  $\Omega$  navrhnutej okolo rovnovážneho stavu, v ktorej úlohu riadenia preberá lokálny stabilizujúci spätnoväzbový riadiaci zákon, a teda suma hodnoty účelovej funkcie na konečnom horizonte a v koncovom stave sa rovná hodnote skutočnej účelovej funkcie.

Použitie účelovej funkcie ako Ljapunovej funkcie na zaručenie stability systémov bez obmedzení bolo navrhnuté už pomerne skoro v práci [30]. Tento prístup sa ukázal ako rozumný a sú na ňom založené súčasné stabilizujúce MPC formulácie. Neskôr bolo zaručenie stability na základe Ljapunovej funkcie vypracované pre spojité systémy [106], pričom prvá aplikácia tohto konceptu na diskrétny systém s obmedzeniami bola prezentovaná v publikácii [75] a neskôr v [12]. Uvedené práce predstavujú základné spracovanie otázky stability MPC, ktoré je založené na monotónne klesajúcej účelovej funkcie.

Okrem uvedených existujú aj rôzne iné varianty a stratégie pre zaručenie stability prediktívneho riadenia. Ich obšírnejší prehľad možno nájsť v prácach [107, 74].

## 8.2 Stabilita MPC s nekonečným horizontom

Potenciálne problémy so stabilitou MPC s obmedzeniami možno demonštrovať, ak si vykreslíme priebeh účelovej funkcie  $J_k$  v čase pre systém opísaný dokonalým modelom a bez uvažovania neznámych rušení. Optimálna účelová funkcia by v takom prípade mala stále klesať, avšak vďaka nelineárnej povahy MPC riadiaceho zákona v uzavretej slučke môže účelová funkcia aj rásť, a to aj v prípade, že celková odozva ostáva stabilná. Tento rast naznačuje, že trajektória vstupov v uzavretej slučke nesleduje optimálnu predikovanú trajektóriu, kedže predikovaná hodnota účelovej funkcie neklesá nepretržite.

Na nasledujúcom príklade si ilustrujeme, že voľba počiatočnej podmienky môže pri prediktívnom riadení systémov s uvažovaním ohraničení viesť k nestabilnej odozve, aj v prípade, že uvažujeme účelovú funkciu na nekonečnom horizonte za predpokladu váhovania koncového stavu.

**Príklad 8.1.** Uvažujme prediktívne riadenie nestabilného, neminimálno-fázového systému opísaného nasledujúcim diskrétnym stavovým modelom druhého rádu:

$$\mathbf{A} = \begin{bmatrix} 1.1 & 2 \\ 0 & 0.95 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0.0787 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} -1 & 1 \end{bmatrix}, \quad (8.4)$$

pričom vstup systému podlieha ohraničeniam  $-1 \leq u_k \leq 1$ . V rámci MPC problému uvažujme váhovacie matice  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  a  $\mathbf{R} = 0.01$ , predikčný horizont  $n_p$ , a váhovanie koncového stavu,  $\mathbf{P}$ , si zvolme ako riešenie Ljapunovej rovnice (4.50), kde  $\mathbf{K}$  je ustálené LQ zosilnenie.

Daný ohraničený MPC problém si formulujme a vyriešme v MATLABe. Začneme deklarovaním modelu, obmedzení vstupu, váhovania a horizontu predikcie.

Zdrojový kód 8.1. prikladstabilita.m

```
clear; close all; clc
A=[1.1 2;0 0.95]; B=[0;0.0787]; C=[-1 1]; % Stavovy model
uh=1; ul=-uh; % Obmedzenia
Q=C'*C; R=0.01; np=2; % Vahy a horiz.
[K,P]=iterdlqr(A,B,Q,R,100); % Vah. konc. stavu
```

Ďalej si pomocou predvytvorených funkcií nájdeme matice účelovej fukcie zloženého MPC riadiaceho zákona a matice obmedzení:

```
[H,G,F]=ucelovafunkcia(A,B,np,Q,R,P); % Ucelova funkcia
[Ac,bc]=obmedzenia(ul,uh,np); % Formulacia obmedzeni
```

Vypočítaný Hessián si tradične prevedieme na jeho symetrickú formu a vypneme informatívne výpisu riešiča.

```
H=(H+H')/2; % Symetr. H
o=optimoptions('quadprog','Display','none'); % Vyp. vypis
```

Prediktívne riadenie systému simulujeme v uzavretej slučke z počiatočného stavu  $\mathbf{x}_0 = [0.5 \ -0.5]^T$ , pričom na riešenie príslušného problému kvadratického programovania v tvare (7.1) použijeme rutinu quadprog. V rámci simulačného cyklu si tiež podľa (4.39) vypočítame a uložíme hodnotu optimálnej predikovanej účelovej funkcie v tvare (4.39), t. j.

$$J_k = \vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + 2\mathbf{x}_k^T \mathbf{G}^T \vec{\mathbf{u}}_k + \mathbf{x}_k^T \mathbf{F} \mathbf{x}_k.$$

```
run=100; % Trvanie simulacie
x=[0.5 -0.5]'; % Pociatocny stav
for i=1:run % Simulacna slucka
    u(:,i)=quadprog(H,G*x(:,i),Ac,bc,[],[],[],[],[],o);
    x(:,i+1)=A*x(:,i)+B*u(1,i); % Simulacia modelu
```

```

J(i)=u(:,i)'*H*u(:,i)+2*x(:,i)'*G'*u(:,i)+x(:,i)'*F*
      x(:,i); % Hod. ucelovej f.
end

```

Na záver si vykreslíme vypočítanú trajektóriu optimálnych vstupov a prislúchajúci výstup zo systému. Znázorníme si tiež priebeh hodnôt optimálnej účelovej funkcie.

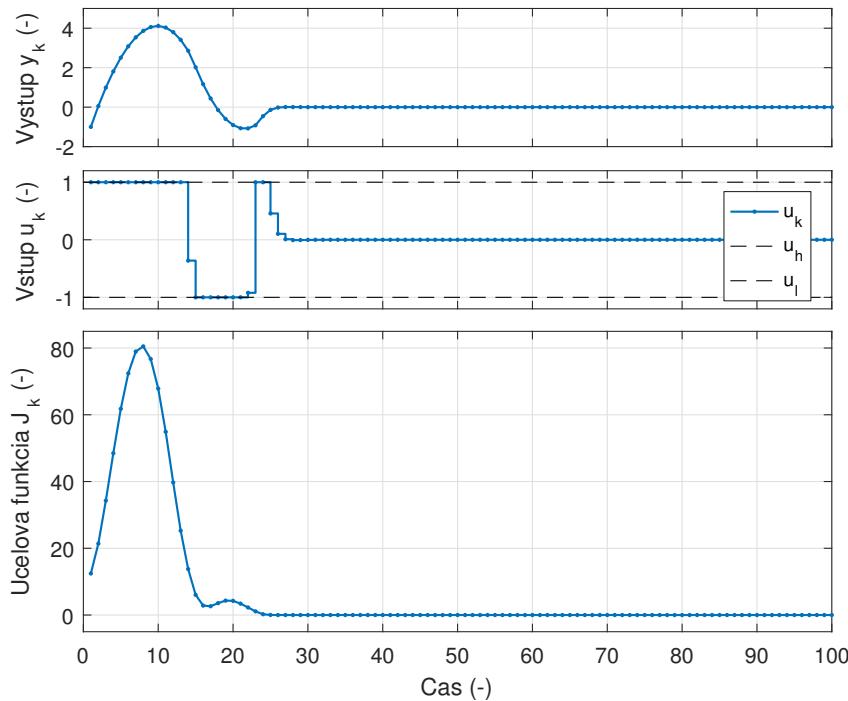
```

figure('color','w') % Obrazok
subplot(3,1,1) % Graf 1
plot(C*x,'LineWidth',1,'Marker','.') % Vystup
ylabel('Vystup y_k (-)') % Oznac. osi y
grid on % Mriezka
subplot(3,1,2) % Graf 2
stairs(u(1,:),'LineWidth',1,'Marker','.') % Vstup
hold on; grid on % Podrz+mriezka
line([0,run],[uh uh],'Color','k','LineStyle','--') % uh
line([0,run],[ul ul],'Color','k','LineStyle','--') % ul
xlabel('Cas (-)') % Oznac. osi x
ylabel('Vstup u_k (-)') % Oznac. osi y
legend('u_k','u_h','u_l') % Legenda
subplot(3,1,3) % Graf 3
plot(J,'LineWidth',1,'Marker','.') % Hod. ucel. f.
grid on % Mriezka
xlabel('Cas (-)') % Oznac. osi x
ylabel('Ucelova funkcia J_k (-)') % Oznac. osi y

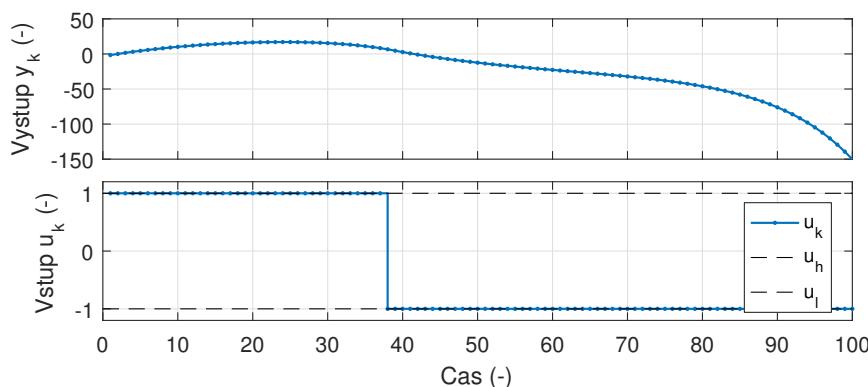
```

Získané priebehy sú znázornené na Obr. 8.1. Ako je zrejmé z odozvy, riadený systém sa pomerne rýchlo dostane z počiatočného do požadovaného stavu, pričom regulátor rešpektuje dané obmedzenia akčného zásahu. Ak však simuláciu zopakujeme pre počiatočný stav  $\mathbf{x}_0 = [0.8 \ -0.8]^T$ , získame nestabilnú odozvu systému znázornenú na Obr. 8.2. Aj keď je daný MPC problém v každom kroku riešiteľný, táto nestabilita môže mať veľmi vážne následky.

Ako už vieme, stabilitu systému v prípade online riešenia ohraničeného MPC problému už nemôžeme overiť analýzou jeho pôlov v uzavretej slučke. Z priebehu optimálnych hodnôt účelovej funkcie na Obr. 8.1 je zrejmé, že aj v prípade stabilnej odozvy daný priebeh spočiatku rastie a regulátor má potenciálne destabilizujúci charakter. Hoci sú predikované trajektórie stavov nevyhnutne stabilné (kedže účelová funkcia na nekonečnom horizonte nadobúda konečnú hodnotu), uvedený počiatočný nárasť  $J_k$  naznačuje, že trajektória vstupov systému v uzavretej slučke nesleduje optimálnu predikovanú trajektóriu vstupov, keďže v opačnom prípade by hodnota účelovej funkcia nerástla.



Obr. 8.1. Simulácia riadeného systému z počiatočného stavu  $\mathbf{x}_0 = [0.5 \ - 0.5]^T$  — priebeh výstupu, vstupu a účelovej funkcie.



Obr. 8.2. Simulácia riadeného systému z počiatočného stavu  $\mathbf{x}_0 = [0.8 \ - 0.8]^T$  — priebeh výstupu a vstupu.

□

Formálne možno stabilitu prediktívneho riadenia s obmedzeniami vyšetriť uvažovaním účelovej funkcie  $J_k$  ako Ljapunovej funkcie. Cieľom Ljapunovej analýzy stability nie je vyšetriť stabilitu zvlášť pre jednotlivé regulátory; naopak, snahou je formulovať podmienky, pri ktorých bude MPC riadiaci zákon stabilný vo všeobecnosti. Na základe týchto podmienok možno navrhnúť modifikovaný MPC regulátor, ktorý zaručuje stabilitu. Ako si ďalej ukážeme, stabilita systému v uzavretej slučke je úzko spojená so zlučiteľnosťou obmedzení. Všeobecnú analýzu stability MPC s obmedzeniami prvýkrát uviedol Mayne a kol. v publikácii [107].

Podľa Ljapunovej vety o stabiliti [98], aplikovanej na diskrétnu systému v práci [14], môžeme definovať  $\mathbf{x}_0$  ako rovnovážny bod systému  $\mathbf{x}_{k+1} = f(\mathbf{x}_k)$  vtedy a len vtedy, ak  $f(\mathbf{x}_0) = \mathbf{x}_0$ . Ako zvyčajne nás bude zaujímať rovnovážny bod  $\mathbf{x} = \mathbf{0}$ , pre ktorý  $f(\mathbf{0}) = \mathbf{0}$ . Podobne možno definovať takzvaný stabilný rovnovážny bod: stav  $\mathbf{x} = \mathbf{0}$  je stabilným rovnovážnym bodom systému, ak pre všetky  $k > 0$  stav zostane v ľubovoľne malom okolí stavového priestoru obsahujúcom  $\mathbf{x} = \mathbf{0}$  vždy, keď počiatočná podmienka  $\mathbf{x}(0)$  leží dostatočne blízko k  $\mathbf{x} = \mathbf{0}$ . Matematicky to možno vyjadriť tak, že pre všetky  $R > 0$  existuje také  $r > 0$ , že pre všetky  $k > 0$  [98, 26] platí:

$$\|\mathbf{x}_0\| < r \Rightarrow \|\mathbf{x}_0\| < R, \forall k > 0. \quad (8.5)$$

Podľa Ljapunovej druhej metódy pre vyšetrenie stability, keď je norma  $\|\mathbf{x}\|$  dostatočne malá,  $\mathbf{x} = \mathbf{0}$  bude stabilným rovnovážnym bodom ak existuje spojito diferencovateľná skalárna funkcia  $V(\mathbf{x})$ , ktorá je [98, 26]:

1. kladne definitná,
2. a platí, že  $V(f(\mathbf{x})) - V(\mathbf{x}) \leq 0$ ,

pričom  $\mathbf{x}_{k+1} = f(\mathbf{x})$ .

Podobným spôsobom je možné definovať asymptotickú konvergenciu. Podľa tohto bude daná funkcia  $\ell(x)$  konvergovať k nule, ak  $V(\mathbf{x})$  je spojito diferencovateľná skalárna funkcia, ktorá je

1. kladne definitná,
2. a platí, že  $V(f(\mathbf{x})) - V(\mathbf{x}) \leq -\ell(\mathbf{x}) \leq 0$ .

Po úprave druhej podmienky dostávame nasledujúci výraz:

$$\ell(\mathbf{x}) \leq V(\mathbf{x}_k) - V(\mathbf{x}_{k+1}), \quad (8.6)$$

ktorý po sumácii oboch strán pre  $k = 0, 1, 2, \dots, \infty$  nadobudne tvar

$$\sum_{k=0}^{\infty} \ell(\mathbf{x}) \leq V(\mathbf{x}_0) - \lim_{k \rightarrow \infty} V(\mathbf{x}_k). \quad (8.7)$$

Pravá strana tejto rovnice je v dôsledku  $V(\mathbf{x}_k) \geq 0$  konečná, čo má dôležitý vplyv na konvergenciu  $\ell(\mathbf{x})$ . V zmysle Ljapunovej druhej metódy platí  $V(\mathbf{x}_k) \geq V(\mathbf{x}_{k+1})$ , t. j. funkčná hodnota v nasledujúcom kroku je menšia ako v súčasnom kroku, čo znamená, že  $V(\mathbf{x}_k)$  sa pre  $k \rightarrow \infty$  blíži ku konečnej limitnej hodnote. Na základe uvedeného možno teda povedať, že ak je pravá strana rovnice (8.7) konečná, tak hodnota  $\ell(\mathbf{x})$  sa musí pre  $k \rightarrow \infty$  blížiť k nule. Poznamenajme, že ak je súčtom nekonečného radu konečná hodnota, potom daný rad musí konvergovať k nule.

Uvedeným spôsobom môžeme vykonať aj analýzu stability prediktívneho riadenia. Ak pre jednoduchosť uvažujeme, že MPC problém je vždy riešiteľný, t. j. pre všetky  $k$  existujú predikované trajektórie vstupov a stavov spĺňajúce ohraničenia, potom optimálna predikovaná účelová funkcia (4.86) je funkciou súčasného stavu  $\mathbf{x}_k$  v zmysle Ljapunovej funkcie:

$$J(\mathbf{x}_k) = V(\mathbf{x}_k). \quad (8.8)$$

Aby sme splnili spomínanú vlastnosť asymptotickej konvergencie, účelová funkcia musí byť kladne definitná. Účelová funkcia  $J_k$  je kladne definitná funkcia  $\mathbf{x}_k$ , ak platí niektorá z nasledujúcich podmienok:

1. matica  $\mathbf{Q}$  je kladne definitná.
2. pár  $(\mathbf{A}, \mathbf{Q}^{1/2})$  je pozorovateľný,

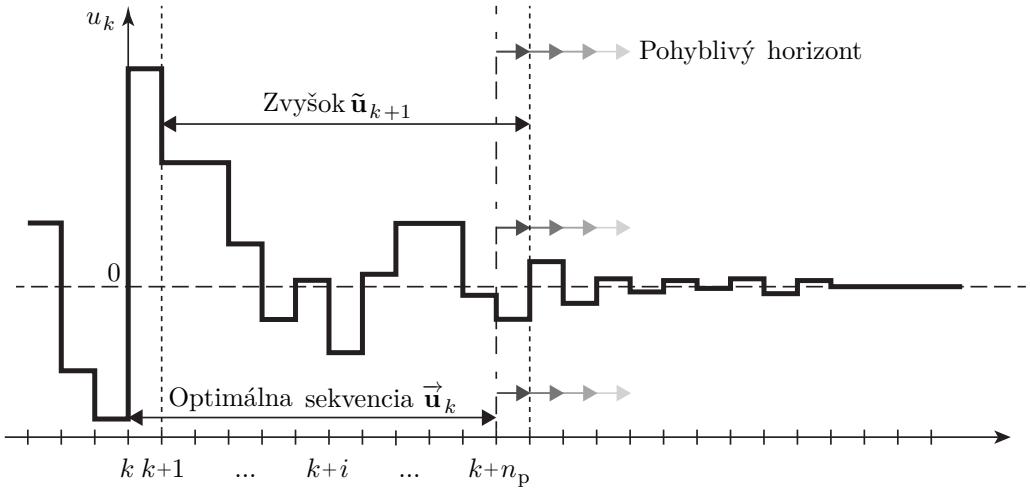
kde  $\mathbf{Q} = \mathbf{Q}^{1/2^T} \mathbf{Q}^{1/2}$ . Prvá podmienka zaručuje, že prvý člen, a teda aj celá suma v (4.86) je kladne definitná.

Ak si váhovanie koncového stavu,  $\mathbf{P}$ , zvolíme tak, že  $J_k$  predstavuje účelovú funkciu s nekonečným horizontom a optimálna predikovaná sekvencia vstupov  $\mathbf{u}_k$  vypočítaná v kroku  $k$  je zlučiteľná pre daný optimalizačný problém v kroku  $k+1$ , potom je optimálna predikovaná účelová funkcia nerastúca a splňa, že:

$$J_{k+1} - J_k \leq - \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right). \quad (8.9)$$

Ak je optimálna predikovaná sekvencia vstupov  $\mathbf{u}_k$  zlučiteľná v kroku  $k+1$ , potom (8.9) platí, pretože optimálna hodnota účelovej funkcie v aktuálnom kroku  $k$  musí byť aspoň taká malá (alebo menšia) ako hodnota tzv. zvyšku <sup>6</sup> optimálnej sekvencie vstupov predikovanej v predchádzajúcom kroku. Podmienka nerastúcej hodnoty účelovej funkcie (8.9) sa nazýva tiež metódou *priamej stability* a je použitá na dôkaz stability prediktívneho riadenia s obmedzeniami v rôznych prácach [75, 106, 109, 131]

<sup>6</sup>angl. tail



Obr. 8.3. Optimálna predikcia  $\vec{u}_k$  v čase  $k$  a jej zvyšok  $\tilde{u}_{k+1}$ .

Na ilustráciu uvedeného tvrdenia uvažujme optimálnu predikovanú sekvenčiu vstupov  $\mathbf{u}_k$  v kroku  $k$  a túto uvažujme ako základ pre predikciu v nasledujúcom časovom kroku  $k+1$ :

$$\tilde{\mathbf{u}}_{k+1} = \left[ \mathbf{u}_{k+1}^T \quad \mathbf{u}_{k+2}^T \quad \dots \quad \mathbf{u}_{k+n_p-1}^T \quad (\mathbf{Kx}_{k+n_p})^T \right]^T, \quad (8.10)$$

kde  $\tilde{\mathbf{u}}_{k+1}$  označujeme ako zvyšok optimálnej predikovanej sekvencie  $\vec{u}_k$ , pozri Obr. 8.3. Ako vidíme, jeho posledný prvok je daný pevným spätnoväzbovým riadiacim zákonom platným v režime 2, pre ktorý platí  $\mathbf{u}_{k+n_p} = \mathbf{Kx}_{k+n_p}$ .

Rov. (4.86) predstavuje účelovú funkciu s nekonečným horizontom. Účelovú funkciu  $\tilde{J}_k$  prislúchajúcemu zvyšku  $\tilde{\mathbf{u}}_{k+1}$  možno potom vyjadriť ako účelovú funkciu  $\tilde{J}_k$  v kroku  $k$  minus časť účelovej funkcie platná pre tento časový krok, t.j.:

$$\begin{aligned} \tilde{J}_{k+1} &= \sum_{i=1}^{n_p} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p+1}^T \mathbf{P} \mathbf{x}_{k+n_p+1} \\ &= \sum_{i=1}^{\infty} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) \\ &= \sum_{i=0}^{\infty} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) - \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right) \\ &= J_k - \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right). \end{aligned} \quad (8.11)$$

Zvyšok  $\tilde{\mathbf{u}}_{k+1}$  bude v skutočnosti v danom čase suboptimálny, keďže sa zakladá na optimálnych predikciách v predchádzajúcom kroku  $k$ . Optimálna hodnota účelovej

funkcie bude preto spĺňať [26]

$$J_{k+1} \leq \tilde{J}_{k+1} = J_k - \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right), \quad (8.12)$$

čo viedie na podmienku danú (8.9).

Tento dôležitý aspekt analýzy stability je užitočný pri formulácii algoritmov na zaručenie stability spätnovázbového riadiaceho systému. Uvedené podmienky si môžeme zhrnúť takto:

Ak  $J_k$  je kladne definitná účelová funkcia s nekonečným horizontom, potom  $\mathbf{x}_k = \mathbf{0}$  je stabilným rovnovážnym bodom daného spätnovázbového systému a  $\mathbf{x}_k$  asymptoticky konverguje k nule, za predpokladu, že zvyšok predikovanej sekvencie vstupov,  $\tilde{\mathbf{u}}_{k+1}$ , je zlučiteľný pre všetky  $k > 0$ .



Druhá metóda dôkazu stability MPC s obmedzeniami, nazývaná tiež *nepriama stabilita*, pochádza z práce [30] a možno ju vyjadriť ako:

$$J_k - J_{k+1} > 0 \quad \text{pre} \quad \mathbf{x} \neq \mathbf{0}. \quad (8.13)$$

Ak je možné dokázať, že ľavá strana výrazu (8.13) je kladná, potom je tiež dokázaná stabilita riadenia. Tento prístup je bližšie opísaný v práci [30] a iných [15, 122].

### 8.3 Stabilita pomocou koncových obmedzení

V predchádzajúcej podkapitole sme si ukázali, že spätnovázbové prediktívne riadenie bude asymptoticky stabilné, ak bude tzv. zvyšok sekvencie predikovaných vstupov vypočítaný v aktuálnom časovom kroku vyhovovať obmedzeniam v nasledujúcich krokoch. Obvykle teda nie je potrebné, aby systém rešpektoval dané obmedzenia od aktuálneho času až do nekonečna, keďže takýto problém by nebolo možné formulovať s uvažovaním konečného počtu obmedzení.

Uvedená požiadavka zlučiteľnosti zvyšku predikovaného vektora vstupov,  $\tilde{\mathbf{u}}_{k+1}$ , je aspoň sčasti splnená samotnou formuláciou problému. Rešpektovanie obmedzení sekvenciou  $\tilde{\mathbf{u}}_{k+1}$  je už totiž zaručené v prvých  $n_p - 1$  krokoch  $n_p$  krokov dlhého horizontu predikcie, čo je zrejmé, keďže optimálne predikcie vypočítané v kroku  $k$  musia splniť obmedzenia. Vektor  $\tilde{\mathbf{u}}_{k+1}$  má však ešte jeden dodatočný prvok, t.j.  $\mathbf{u}_{k+n_p} = \mathbf{K} \mathbf{x}_{k+n_p}$ , definovaný spätnovázbovým riadiacim zákonom režimu 2 zloženého riadiaceho zákona.

Preto, aby sme zaručili zlučiteľnosť tohto posledného prvku a tým aj celého zvyšku  $\tilde{\mathbf{u}}_{k+1}$ , musíme si v kroku  $k$  definovať dodatočné obmedzenie. Ako už bolo

spomenuté, táto množina dodatočných obmedzení sa nazýva obmedzenia koncového stavu, alebo jednoducho koncové obmedzenia. Koncové obmedzenia sú definované v druhom režime zloženého riadiaceho zákona, v ktorom je aktívny pevný spätnoväzbový riadiaci zákon, a preto sú aj definované v zmysle predikcie koncového stavu, t. j.  $\mathbf{x}_{k+n_p}$ .

Definujme si preto oblasť—podmnožinu stavového priestoru, v ktorej sa musí nachádzať predikcia koncového stavu,  $\mathbf{x}_{k+n_p}$ , aby boli splnené koncové obmedzenia. Ak máme systém s obmedzeniami vstupu  $\underline{\mathbf{u}} \leq \mathbf{u}_k \leq \bar{\mathbf{u}}$  a obmedzeniami stavu  $\underline{\mathbf{x}} \leq \mathbf{x}_k \leq \bar{\mathbf{x}}$ , tak potom na to, aby sme zaručili, že zvyšok optimálnej sekvencie vstupov,  $\tilde{\mathbf{u}}_{k+1}$ , bude splňať dané obmedzenia na celom horizonte, musíme do optimalizačného problému MPC zahrnúť koncové obmedzenia v tvare

$$\mathbf{x}_{k+n_p} \in \Omega \quad \Rightarrow \quad \begin{cases} \underline{\mathbf{u}} \leq \mathbf{K}\mathbf{x}_{k+n_p} \leq \bar{\mathbf{u}}, \\ \underline{\mathbf{x}} \leq \mathbf{x}_{k+n_p} \leq \bar{\mathbf{x}}, \end{cases} \quad (8.14)$$

definujúce už spomínanú *cieľovú množinu*  $\Omega$ , do ktorej chceme riadením do konca horizontu priviesť stav systému. Prislúchajúce koncové obmedzenia musia byť vypočítané tak, aby bola zaručená rekurzívna zlučiteľnosť problému MPC, a teda predikcie  $\tilde{\mathbf{u}}_{k+1}$  zabezpečia splnenie obmedzení vrátane koncových obmedzení.

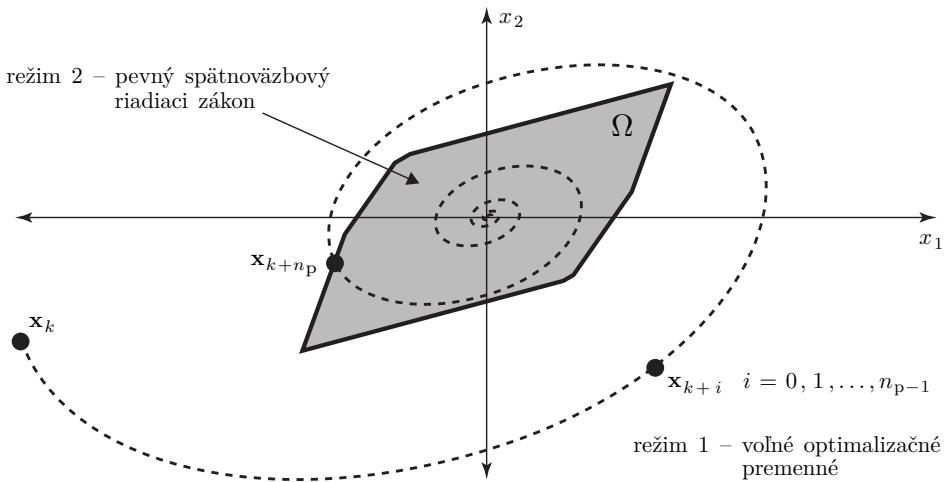
Toto však nestačí, pretože predikcie generované zvyškom  $\tilde{\mathbf{u}}_{k+1}$  v kroku  $k+1$  musia splňať tiež nasledujúce koncové obmedzenie, inými slovami koncový stav  $\mathbf{x}_{k+n_p+1}$  v čase  $k+1$  musí taktiež patriť do cieľovej množiny  $\Omega$ , t. j.  $\mathbf{x}_{k+n_p+1} \in \Omega$ .

Cieľová množina je na príklade dvojrozmerného stavového priestoru ilustrovaná na Obr. 8.4. V oblasti mimo cieľovú množinu vstupy uvažujeme ako voľné optimalizačné premenné, zatiaľ čo v jej vnútri sú vstupy počítané pevným spätnoväzbovým riadiacim zákonom. Akonáhle sa stavová trajektória dostane do množiny  $\Omega$ , už ju nemôže opustiť. Uvedená vlastnosť tzv. invariantnosť množiny  $\Omega$  môže byť interpretovaná tak, že táto množina je kontraktívna, resp. „zmršťujúca sa“. Ak je teda stav v čase  $k$  súčasťou invariantnej cieľovej množiny, musí ňou byť aj v nasledujúcom a ďalších časových krokoch.

Na zaručenie stability MPC v uzavretej slučke musíme teda v prípade zlučiteľnosti MPC problému v čase  $k$  zaručiť zlučiteľnosť zvyšku predikcií vstupov,  $\tilde{\mathbf{u}}_{k+1}$ , v čase  $k+1$ . Aby sme toto dosiahli, musíme teda zaručiť rekurzívnu zlučiteľnosť, čo dosiahneme splnením nasledujúcich podmienok:

1. všetky stavy patriace do množiny  $\Omega$  spĺňajú obmedzenia,
2. množina  $\Omega$  je v režime 2 invariantná.

Prvá podmienka je v podstate reformuláciou (8.14). Invariantnosť množiny  $\Omega$  ďalej znamená, že ak do nej patrí koncový stav, musí to isté platiť aj pre jeho predikciu v



Obr. 8.4. Ilustrácia invariantnej cieľovej množiny  $\Omega$  spolu so stavovou trajektóriou.

uzavretej slučke podľa stabilizujúceho spätnoväzbového riadiaceho zákona [17, 16]:

$$\mathbf{x}_{k+n_p} \in \Omega \Rightarrow (\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x}_{k+n_p} \in \Omega. \quad (8.15)$$

Úlohu prediktívneho riadenia s obmedzeniami, ale bez garancie stability, sme si definovali pomocou MPC problému (5.60). Uvažujme preto teraz pridanie koncových obmedzení tak, aby stabilita daného MPC problému bola zaručená. Ak chceme daný systém priviesť do počiatku, t. j.  $\mathbf{x} = \mathbf{0}$ , musíme nájsť optimálnu postupnosť vstupov riešením nasledujúceho MPC problému:

$$\min_{\overrightarrow{\mathbf{u}}_k} \left[ \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \right], \quad (8.16a)$$



vzhľadom na podmienky:

$$\underline{\mathbf{u}} \leq \mathbf{u}_{k+i} \leq \bar{\mathbf{u}} \quad \text{pre } i = 0, 1, \dots, n_p - 1, \quad (8.16b)$$

$$\underline{\mathbf{x}} \leq \mathbf{x}_{k+i} \leq \bar{\mathbf{x}} \quad \text{pre } i = 1, 2, \dots, n_p - 1, \quad (8.16c)$$

$$\mathbf{x}_{k+n_p} \in \Omega, \quad (8.16d)$$

$$\mathbf{x}_k = \mathbf{x}(k), \text{ resp. } \mathbf{x}_k = \hat{\mathbf{x}}(k) \quad (8.16e)$$

$$\mathbf{x}_{k+i+1} = \mathbf{A}\mathbf{x}_{k+i} + \mathbf{B}\mathbf{u}_{k+i} \quad \text{pre } i = 0, 1, 2, \dots, n_p - 1, \quad (8.16f)$$

$$\mathbf{u}_{k+i} = \mathbf{K}\mathbf{x}_{k+i}, \quad \text{pre } i \geq n_p. \quad (8.16g)$$

Ak teda cieľová množina  $\Omega$  splňa obmedzenia kladené na daný systém a zároveň je invariantná v režime 2 zloženého riadiaceho zákona, potom ak je uvedený MPC problém riešiteľný v kroku  $k$ , zostane rekurzívne riešiteľný aj v časových krokoch  $k = 1, 2, 3, \dots, \infty$ .

Na základe uvedeného si môžeme tiež definovať tzv. množinu riešiteľných stavov<sup>7</sup>, ktorá predstavuje podmnožinu stavového priestoru obsahujúcu všetky počiatočné podmienky (stavy), z ktorých je možné riadením priviesť predikcie stavu do množiny  $\Omega$  počas  $n_p$  krokov dlhého horizontu režimu 1 zloženého riadiaceho zákona [76]. Označme si túto množinu ako  $S_\Omega$ ; formálne teda ide o množinu všetkých možných počiatočných stavov  $\mathbf{x}_0$ , pre ktoré existuje taký vektor  $\vec{\mathbf{u}}_k$ , že na konci horizontu bude  $\mathbf{x}_{k+n_p}$  patrila do množiny  $\Omega$ :

$$S_\Omega = \left\{ \mathbf{x}_0 : \text{existuje také } \vec{\mathbf{u}}_k, \text{ že} \begin{array}{l} \mathbf{x}_{k+n_p} \in \Omega \\ \underline{\mathbf{u}} \leq \mathbf{u}_{k+i} \leq \bar{\mathbf{u}} \\ \underline{\mathbf{x}} \leq \mathbf{x}_{k+i+1} \leq \bar{\mathbf{x}} \end{array} \right\}, \quad (8.17)$$

kde  $i = 0, 1, \dots, n_p - 1$ . Množina riešiteľných počiatočných stavov teda predstavuje pracovnú oblasť MPC regulátora. Stavy ležiace mimo tejto množiny nie sú riešiteľné, a práve preto je v našom záujme, aby bola čo najväčšia. Platí, že zväčšením dĺžky predikčného horizontu  $n_p$  sa zväčší tiež veľkosť množiny riešiteľných počiatočných stavov  $S_\Omega$ . Iným spôsobom je zväčšiť veľkosť cieľovej množiny  $\Omega$ .

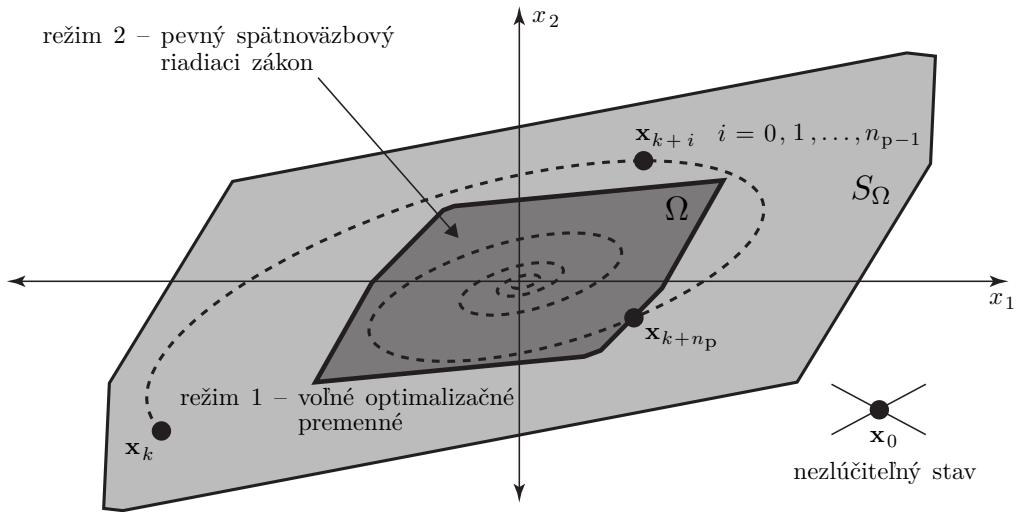
Množina riešiteľných stavov  $S_\Omega$  je ilustrovaná na Obr. 8.5 ako väčšia, sivá oblasť. Všetky stavy obsiahnuté v tejto množine sú riešiteľné, čo znamená, že je možné ich riadením v  $n_p$  krokoch priviesť do cieľovej množiny. Naopak, stavy mimo tejto množiny sú neriešiteľné a porušili by tak obmedzenia. Cieľová množina, znázornená tmavosivou farbou, je podmnožinou množiny riešiteľných stavov. Stavová trajektória je na obrázku znázornená prerušovanou špirálovitou krivkou. Režim 1 zloženého MPC riadiaceho zákona je aktívny v množine riešiteľných stavov, zatiaľ čo pevný spätnoväzbový riadiaci zákon v režime 2 uvažujeme v cieľovej množine.

## 8.4 Maximálna invariantná cieľová množina

Z predchádzajúcej diskusie vyplýva, že pre danú konečnú dĺžku predikčného horizontu je potrebné zabezpečiť najväčšiu možnú množinu riešiteľných stavov a tým zväčšiť pracovnú oblasť prediktívneho regulátora. Pre tento účel je preto potrebné koncové obmedzenia formulovať tak, aby sme získali čo najväčšiu cieľovú množinu. Problematika návrhu tzv. maximálnej prípustnej invariantnej (cieľovej) množiny<sup>8</sup>

<sup>7</sup>angl. region of attraction, set of feasible states, feasible set

<sup>8</sup>angl. maximal admissible set



Obr. 8.5. Ilustrácia množiny všetkých riešiteľných, a teda prípustných počiatočných stavov,  $S_\Omega$ . Cieľová množina  $\Omega$  je podmnožinou množiny riešiteľných stavov, a je znázornená tmavšou farbou.

bola podrobne spracovaná v publikácii [54], pričom jej elipsoidálna approximácia pre spojité systémy bola navrhnutá v [107].

Ako si ďalej ukážeme, maximálnu cieľovú množinu, z pohľadu geometrie teda množinu s najväčším možným objemom, získame napr. formulovaním takej množiny obmedzení koncového stavu, ktoré zaručia splnenie obmedzení kladených na systém počas tzv. *horizontu koncových obmedzení*<sup>9</sup>  $n_c$ , ako dodatku k predikčnému horizontu  $n_p$ . V prípade lineárnych časovo invariantných systémov, ktoré uvažujeme aj my, je dĺžka tohto horizontu konštantná a je možné ju určiť offline.

Rekurzívna riešiteľnosť vyžaduje, aby obmedzenia boli splnené na celej dĺžke horizontu režimu 2 zloženej účelovej funkcie. To znamená, že obmedzenia stavov a vstupov musia byť splnené aj v režime 2 zloženého MPC riadiaceho zákona, t. j.:

$$\bar{\mathbf{u}} \leq \mathbf{u}_{k+i} \leq \underline{\mathbf{u}}, \quad i = n_p, n_p + 1, n_p + 2, \dots, \infty, \quad (8.18a)$$

$$\bar{\mathbf{x}} \leq \mathbf{x}_{k+i} \leq \underline{\mathbf{x}}, \quad i = n_p, n_p + 1, n_p + 2, \dots, \infty, \quad (8.18b)$$

z čoho ďalej vyplýva, že maximálnu prípustnú cieľovú množinu dostaneme, ak zavedieme obmedzenia na horizonte režimu 2 s predpokladom, že  $\Omega$  je definovaná nasledovne:

$$\Omega = \left\{ \mathbf{x} : \underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{u}}, \underline{\mathbf{x}} \leq (\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{x}} \right\} \quad (8.19)$$

<sup>9</sup>angl. constraint checking horizon [26]

pre  $i = n_p, n_p + 1, n_p + 2, \dots, \infty$ . Našťastie, ako sme už naznačili, namiesto množiny nekonečne veľa obmedzení je možné obmedzenia na celom nekonečnom horizonte režimu 2 zohľadniť uvažovaním  $n_c$  krokov dlhého horizontu koncových obmedzení. Množina (8.19) bude preto namiesto nekonečného horizontu definovaná pre  $i = n_p, n_p + 1, \dots, n_c - 1, n_c$ .

Aby sme dokázali, že pomocou horizontu koncových obmedzení s konečným počtom krokov je možné zabezpečiť splnenie obmedzení na nekonečnom horizonte režimu 2 zloženého MPC riadiaceho zákona, uvažujme MPC problém iba s obmedzeniami vstupov,  $\bar{\mathbf{u}}$  a  $\underline{\mathbf{u}}$ . Taktiež si ako  $\Pi_i$  označme množinu počiatočných podmienok, t. j. stavov, pre ktoré sú tieto obmedzenia splnené na určitom horizonte dĺžky  $n$  pri uvažovaní spätnoväzbového stavového riadiaceho zákona  $\mathbf{u} = \mathbf{Kx}$  predpokladaného v režime 2, pričom platí

$$\Pi_n = \left\{ \mathbf{x} : \underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{BK})^i \mathbf{x} \leq \bar{\mathbf{u}}, i = 0, 1, \dots, n \right\}. \quad (8.20)$$

Predpokladajme ďalej, že množinu počiatočných podmienok spĺňajúcich obmedzenia na nekonečne dlhom horizonte režimu 2 možno nahradit množinou počiatočných podmienok určených horizontom koncových obmedzení konečnej dĺžky:

$$\Pi_\infty = \Pi_{n_c}. \quad (8.21)$$

Ako už vieme, spätnoväzbový systém daný maticou dynamiky uzavretého režima obvodu,  $(\mathbf{A} + \mathbf{BK})$ , je stabilný, ak všetky jej vlastné čísla sú menšie ako 1. Je zrejmé, že keď zvyšujeme jej mocninu  $i$  do nekonečna, prvky matice sa búdú blížiť k nule, t. j.:

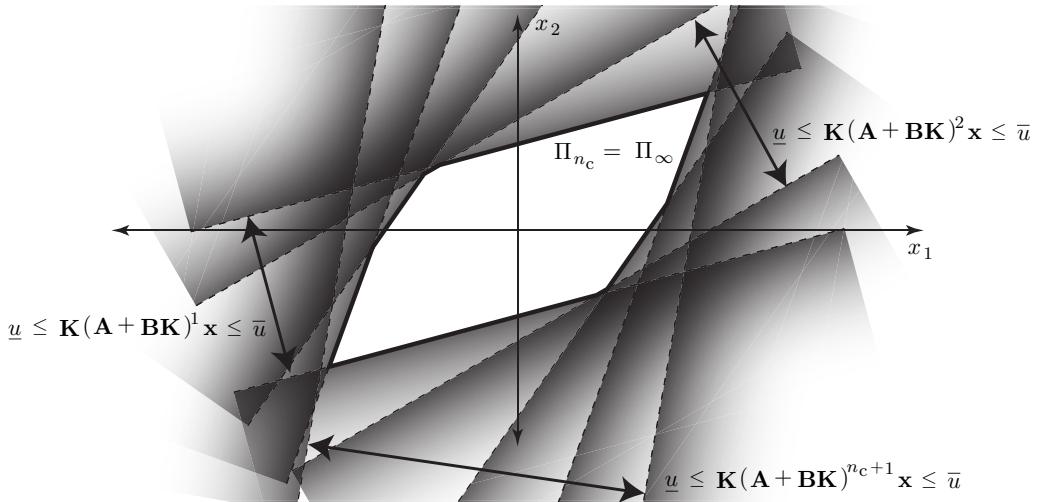
$$(\mathbf{A} + \mathbf{BK})^i \rightarrow \mathbf{0} \quad \text{pre} \quad i \rightarrow \infty. \quad (8.22)$$

Označme si kolmú vzdialenosť nadroviny definovanej ako  $\mathbf{K}(\mathbf{A} + \mathbf{BK})^i \mathbf{x} = \bar{\mathbf{u}}$  od počiatku  $\mathbf{x} = \mathbf{0}$  ako  $d_i$ . Je zrejmé, že pre ľubovoľné  $\mathbf{x}$  sa bude vzdialenosť  $d_i$  pre  $i \rightarrow \infty$  blížiť k nekonečnu [26]:

$$d_i = \frac{\bar{\mathbf{u}}}{\|\mathbf{K}(\mathbf{A} + \mathbf{BK})^i\|_2} \rightarrow \infty \quad \text{pre} \quad i \rightarrow \infty, \quad (8.23)$$

kedže menovateľ výrazu sa bude so zvyšujúcim  $i$  blížiť k nule, lebo systém je v uzavretej slučke stabilný. Z geometrického pohľadu toto znamená, že horné a dolné obmedzenia vstupov,  $\bar{\mathbf{u}}$  a  $\underline{\mathbf{u}}$ , definujú nadroviny prislúchajúce zväčšujúcemu sa horizontu. Oblasť medzi dvoma nadrovinami prislúchajúcimi jednému obmedzeniu z množiny  $\underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{BK})^i \mathbf{x} \leq \bar{\mathbf{u}}$  sa pri rastúcom  $i$  bude stále viac zväčšovať.

Poznamenajme, že pre daný systém s obmedzeniami je vždy možné vybrať taký počiatočný stav, ktorého budúci vývoj skôr či neskôr poruší obmedzenia, čo súvisí



Obr. 8.6. Ilustrácia maximálnej invariantnej cielovej množiny  $\Pi_{n_c} = \Pi_{\infty}$  v dvojrozmernom stavovom priestore, ktorá je definovaná ako prienik koncových obmedzení na horizonte  $n_c$ .

s tým, že obmedzenia v režime 2 vytvárajú konečnú podmnožinu celého stavového priestoru. To znamená, že ak je stav  $\mathbf{x}_0$  dostatočne veľký, musí v určitom budúcom čase dôjsť k porušeniu obmedzení, a preto množina  $\Pi_{\infty}$  musí byť konečná, za predpokladu, že páár  $(\mathbf{A} + \mathbf{B}\mathbf{K}, \mathbf{K})$  je pozorovateľný. Oblast v stavovom priestore definovaná horným a dolným obmedzením na vstup

$$\left\{ \mathbf{x} : \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{u} \right\} \quad (8.24)$$

je nadmnožinou množiny  $\Pi_{\infty}$  pre všetky  $i > n_c$  a horizont koncových obmedzení  $n_c$ . To znamená, že musí existovať konečný horizont  $n_c$ , ktorý zaručí zlučiteľnosť obmedzení na nekonečnom horizonte režimu 2 zloženého MPC riadiaceho zákona. Všetky stavy nachádzajúce sa v  $\Pi_{n_c}$  sa preto nachádzajú tiež v množine  $\Pi_{\infty}$ .

Maximálna invariantná cielová množina  $\Pi_{n_c} = \Pi_{\infty}$  pre systém s dvoma stavmi ( $n_x = 2$ ) a symetrickými obmedzeniami jednorozmerného vstupu ( $n_u = 1$ ),  $\underline{u}$  a  $\bar{u}$ , je ilustrovaná na Obr. 8.6. Ako je zrejmé, geometricky je daná ako prienik podmnožín stavového priestoru získaných postupným iterovaním koncových obmedzení pozdĺž horizontu koncových obmedzení,  $n_c$ .

Pokúsme sa túto skutočnosť ilustrovať graficky v nasledujúcom príklade.

**Príklad 8.2.** Pre systém s obmedzeniami vstupu z Pr. 8.1 graficky znázorníte obmedzenia koncového stavu podľa (8.24). Uvažujte váhovanie  $\mathbf{Q} = \mathbf{C}^T \mathbf{C}$  a  $\mathbf{R} = 1$ ,

a stabilizujúci LQ riadiaci zákon v režime 2 zloženého MPC riadiaceho zákona. Na základe grafického výstupu určte horizont koncových obmedzení,  $n_c$ .

Vytvorime si v MATLABe skript s názvom *prikladkoncoveobmedzenia.m*, ktorý si uvedieme definovaním modelu a parametrov MPC problému, analogicky ako v Pr. 8.1. Vypočítame si tiež LQ zosilnenie v zápornej spätej väzbe,  $\mathbf{K}$ , a tiež maticu dynamiky uzavretého regulačného obvodu,  $\Psi = \mathbf{A} + \mathbf{B}\mathbf{K}$ .

Zdrojový kód 8.2. *prikladkoncovebmedzenia.m*

```
clear; close all; clc
A=[1.1 2;0 0.95]; B=[0;0.0787]; C=[-1 1]; % Stavovy model
uh=1; ul=-uh; % Obmedzenia vstupu
Q=C'*C; R=1; np=2; % Vahy a horizont
[K,P]=iterdlqr(A,B,Q,R,100); % Vahov. koncov. stavu
K=-K; % Zaporna spatna vazba
Psi=A+B*K; % Dynamika uzavr. reg. obvodu
```

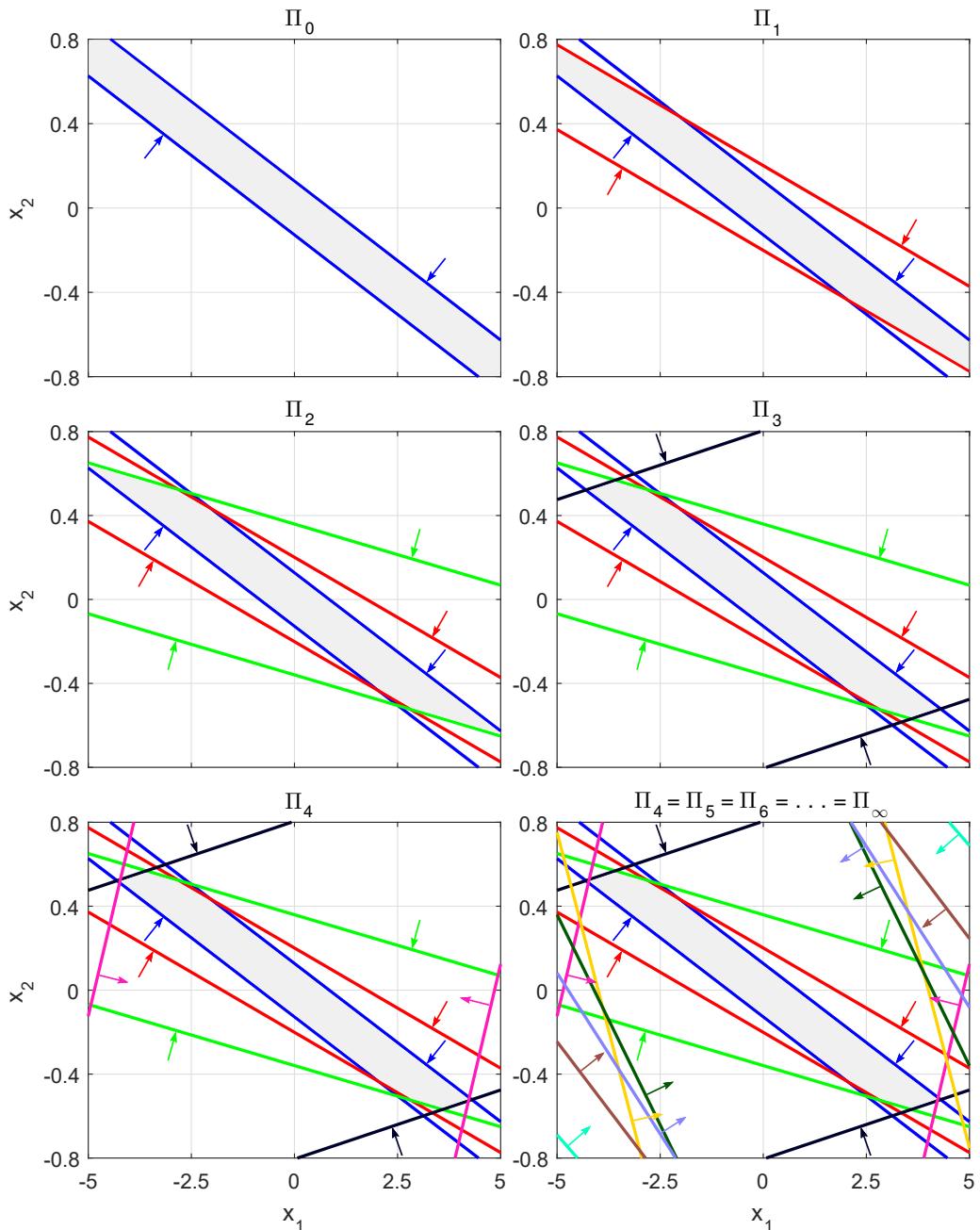
Ďalej si otvorime nové grafické okno a v cykle podľa (8.24) formulujme obmedzenia koncového stavu (resp. v tomto prípade všeobecne stavu  $\mathbf{x}$ ) v tvare:

$$\begin{bmatrix} \mathbf{K}\Psi^i \\ -\mathbf{K}\Psi^i \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} \bar{u} \\ -u \end{bmatrix}, \quad \text{pre } i = 0, 1, 2, \dots, n.$$

Tieto sa budú opakovane prepočítavať pre rastúci horizont  $i$  a zároveň dokreslovať do stavového priestoru na obrázku pomocou funkcie *vykreslitobmedzenia()* (viď Kap. 6, Zdrojový kód 6.6). Celkový počet iterácií slučky si zvolíme napríklad ako  $n = 10$ . Na záver si ohraničíme a označíme osi grafu.

```
figure('color','w') % Graficke okno
n=9; % Pocet iteracii
for i=0:n
    Ac=[ K*Psi^i; % Lava strana obmedzeni vstupu
          -K*Psi^i];
    bc=[ uh; % Prava strana obmedzeni vstupu
          -ul];
    vykreslitobmedzenia(Ac,bc,[-5 5]) % Vykresl. nadrovin
end
axis([-5 5 -0.8 0.8]) % Hranice grafu
xlabel('x_1'); ylabel('x_2'); % Oznacenie osi
```

Obr. 8.7 znázorňuje grafické výstupy programu pre  $n \in \{0, 1, 2, 3, 4, 9\}$ .



Obr. 8.7. Grafické znázornenie koncových obmedzení, ktorých prienik predstavuje množiny  $\Pi_n$ ,  $n = 0, 1, 2, \dots, \infty$ , znázornené sivou farbou. Pre maximálnu invariantnú cieľovú množinu z posledného grafu platí  $\Omega = \Pi_4 = \Pi_5 = \dots = \Pi_\infty$ .

V jednotlivých grafoch na Obr. 8.7 je naviac sivou farbou ilustrovaný prienik daných obmedzení, t. j. polpriestorov  $\mathbf{K}\Psi^i \mathbf{x} \leq \bar{u}$  a  $-\mathbf{K}\Psi^i \mathbf{x} \leq -\underline{u}$  pre  $i = 0, 1, 2, \dots, n$ . Tieto prieники predstavujú množiny  $\Pi_n$  definované ako (8.20) a z Obr. 8.7 je zrejmé, že pre ne platí:

$$\begin{aligned}\Pi_0 &= \mathbb{R}^{n_x} \cap \{\mathbf{x} : \underline{u} \leq \mathbf{K}\mathbf{x} \leq \bar{u}\}, \quad \Pi_0 \subset \mathbb{R}^{n_x}, \\ \Pi_1 &= \Pi_0 \cap \{\mathbf{x} : \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x} \leq \bar{u}\}, \quad \Pi_1 \subset \Pi_0, \\ \Pi_2 &= \Pi_1 \cap \left\{ \mathbf{x} : \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^2 \mathbf{x} \leq \bar{u} \right\}, \quad \Pi_2 \subset \Pi_1, \\ \Pi_3 &= \Pi_2 \cap \left\{ \mathbf{x} : \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^3 \mathbf{x} \leq \bar{u} \right\}, \quad \Pi_3 \subset \Pi_2, \\ \Pi_4 &= \Pi_3 \cap \left\{ \mathbf{x} : \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^4 \mathbf{x} \leq \bar{u} \right\}, \quad \Pi_4 \subset \Pi_3,\end{aligned}\tag{8.25}$$

pričom  $\Pi_n = \Pi_4, \forall n > 4$ , z čoho vyplýva, že v tomto prípade je horizont koncových obmedzení  $n_c = 4$ .  $\Pi_4$  teda predstavuje maximálnu invariantnú cieľovú množinu, t. j.  $\Omega = \Pi_4$ . Ako si ďalej ukážeme, vlastnosť množín  $\Pi_n$  vyjadrenú Rov. (8.25) je možné formalizovať do jednoduchého algoritmu pre určenie horizontu  $n_c$ .

□

Ďalej sa preto pokúsme nájsť množinu  $\Pi_\infty = \Pi_{n_c}$  za predpokladu, že pre horizont  $n_c$  sa nasledujúca množina bude rovnať predchádzajúcej, a teda pre  $\Pi_{n_c+1} = \Pi_{n_c}$  už nezískame ďalší nárast v objeme cieľovej množiny. Toto tvrdenie môžeme formalizovať nasledujúcou vetou [26]:  $\Pi_\infty = \Pi_{n_c}$  vtedy a len vtedy, ak  $\Pi_{n_c+1} = \Pi_{n_c}$ . Pre jej dôkaz uvažujme, že ak  $\mathbf{x} \in \Pi_{n_c}$ , potom  $\mathbf{x}$  patrí aj do nasledujúcej množiny, t. j.  $\mathbf{x} \in \Pi_{n_c+1}$  pre nejakú hodnotu  $n_c$ , čo môžeme matematicky zapísat ako<sup>10</sup>

$$\mathbf{x} \in \Pi_{n_c} \Rightarrow \mathbf{x} \in \Pi_{n_c+1} \Rightarrow \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+1} \mathbf{x} \leq \bar{u}.\tag{8.26}$$

Nahradíme teraz  $\mathbf{x}$  výrazom  $\mathbf{x} = (\mathbf{A} + \mathbf{B}\mathbf{K})\check{\mathbf{x}}$ , a dosadíme tento do predchádzajúcej nerovnosti, na základe čoho dostávame:

$$\underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+1} \mathbf{x} \leq \bar{u} \Rightarrow \underline{u} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+2} \check{\mathbf{x}} \leq \bar{u}.\tag{8.27}$$

Toto znamená, že vždy keď platí, že stav  $\mathbf{x} \in \Pi_{n_c+1}$ , platí tiež, že  $\mathbf{x} \in \Pi_{n_c+2}$ . Ak túto skutočnosť aplikujeme rekurzívne, dostaneme sa k záveru, že implikácia (8.26) tiež hovorí, že pre určité  $n_c$  stav v režime 2 zloženého MPC riadenia zostane v danej cieľovej množine.

Praktický význam uvedeného tvrdenia a implikácie (8.26) je, že v prípade uvažovania obmedzení vstupu postačuje overiť, či obmedzenia definované na horizonte  $n_c$  režimu 2 spĺňajú tiež obmedzenia na horizonte  $n_c + 1$ . Ak toto platí, potom

<sup>10</sup>Ďalej pre všeobecnosť uvažujeme ľubovoľný počet vstupov.

$n_c$  je hľadaný horizont koncových obmedzení. V každej iterácii je teda potrebné overiť, či je splnená nasledujúca podmienka:

$$\underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{u}}, i = 0, 1, \dots, n_c \Rightarrow \underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+1} \mathbf{x} \leq \bar{\mathbf{u}} \quad (8.28)$$

Pre prípad s uvažovaním iba obmedzení vstupu môžeme algoritmus pre určenie dĺžky horizontu koncových obmedzení zhrnúť do nasledujúcich krokov, ktoré inicializujeme s  $n_c = 0$  [26]:

1. Vyrieš  $2n_u$  lineárnych programov, pre dané dolné a horné obmedzenie vstupov,  $\underline{\mathbf{u}}$  a  $\bar{\mathbf{u}}$ :

$$u_{\max,j} = \max_{\mathbf{x}} \mathbf{K}_j(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+1} \mathbf{x} \text{ vzhľ. na } \underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{u}}, \quad (8.29)$$

$$u_{\min,j} = \min_{\mathbf{x}} \mathbf{K}_j(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+1} \mathbf{x} \text{ vzhľ. na } \underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{u}}, \quad (8.30)$$

kde ohraničenia uvedených problémov sú formulované pre  $i = 0, 1, 2, \dots, n_c$  pre jednotlivé obmedzenia vstupu indexované s  $j = 1, 2, \dots, n_u$ , kde  $\mathbf{K}_j$  označuje  $j$ -tý blokový riadok stavového zosilnenia  $\mathbf{K}$ .

2. Otestuj, či  $u_{\max,j} \leq \bar{u}_j$  a  $u_{\min,j} \geq \underline{u}_j$  pre každé  $j = 1, 2, \dots, n_u$ . Ak uvedené platí, ukonči algoritmus a označ  $n_c$  ako dodatočný horizont; ak nie, pokračuj.
3. Zväčši horizont ako  $n_c = n_c + 1$  a pokračuj odznova krokom 1.

Inými slovami, v každej iterácii algoritmu pre každý ohraničený vstup skontrolujeme, či obmedzenia v prvých  $n_c$  krokoch budú tiež splňať obmedzenia v kroku  $n_c + 1$ . Túto úlohu vykonáme tak, že jednoducho vypočítame maximálnu (minimálnu) hodnotu vstupu, ktorú by bolo možné získať pomocou spätnoväzbového riadiaceho zákona  $\mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^{n_c+1} \mathbf{x}$  v ďalšom kroku  $n_c + 1$ , pričom výpočet je ohraničený v zmysle  $\underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{u}}$  pre  $i = 0, \dots, n_c$ . Ak je vypočítaná maximálna (minimálna) hodnota väčšia (menšia) ako príslušné obmedzenie, uvedená procedúra sa zopakuje a dĺžka horizontu navýší o 1. V opačnom prípade, ak sú dané podmienky splnené, algoritmus sa ukončí a aktuálna hodnota  $n_c$  označuje horizont koncových obmedzení prislúchajúci režimu 2 zloženého riadiaceho zákona.

**Príklad 8.3.** Podľa uvedeného algoritmu navrhnite v programe MATLAB funkciu na výpočet dĺžky horizontu koncových obmedzení  $n_c$  pre systém s obmedzeniami vstupu, kde  $\mathbf{u} \in \mathbb{R}^{n_u}$ .

Vytvorime si funkciu `horizontkoncobmedzeni()`, ktorej vstupom bude LQ zosilnenie  $\mathbf{K}$ , matica dynamiky uzavretého regulačného obvodu  $\Psi = \mathbf{A} + \mathbf{B}\mathbf{K}$ , a horné a dolné obmedzenie vstupnej veličiny,  $\bar{\mathbf{u}}$  a  $\underline{\mathbf{u}}$ . V rámci funkcie si vyčítame počet vstupov systému, do prvých dvoch blokových riadkov premennej  $\mathbf{K}$  si uložíme matice  $\mathbf{K}$  a  $\mathbf{K}\Psi$ , a inicializujme horizont  $n$  a premenné  $u_{\max}$  a  $u_{\min}$ .

Zdrojový kód 8.3. horizontkoncobmedzeni.m

```
function [n,KPsi] = horizontkoncobmedzeni(K,Psi,uh,ul)
o=optimoptions('linprog','Display','none');% Vypis
KPsi=K; KPsi(nu+1:2*nu,:)=K*Psi; % Obmedzenia LP
n=0; umax=uh+1; umin=ul-1; nu=length(uh); % Inicializacia
```

Nasleduje samotný algoritmus, v každom kroku ktorého sformulujeme obmedzenia a účelovú funkciu lineárneho programu (LP) (8.29), respektíve (8.30). Argumentom maximalizácie v (8.29) je stavový vektor  $\mathbf{x}$ , pričom daný problém riešime pre každý zo vstupov  $\mathbf{u} \in \mathbb{R}^{nu}$  pomocou LP riešiča linprog, ktorý je súčasťou balíka Optimization Toolbox programu MATLAB. Riešenie pre každý vstup  $u_{\max,j}$ ,  $j = 1, \dots, n_u$  si označíme ako  $u_{\max,j}$ ,  $j = 1, \dots, n_u$  a uložíme do premennej umax. V prípade, ak sú obmedzenia vstupu symetrické, t. j.  $\mathbf{u} = -\bar{\mathbf{u}}$ , druhý LP (8.30) nie je potrebné riešiť, keďže  $u_{\min,j} = -u_{\max,j}$ ,  $j = 1, \dots, n_u$ . Algoritmus bude bežať, pokiaľ bude splnená podmienka  $umax > uh$  alebo  $umin < ul$ , v opačnom prípade bude zastavený, a to v konečnom počte krokov. Posledná hodnota  $n$  získaná pri ukončení algoritmu predstavuje horizont koncových obmedzení  $n_c$ .

```
while (umax>uh | umin<ul) % Podmienka behu cyklu
    n=n+1; % Inkrementuj horizont
    f=K*Psi^(n+1); % Gradient pre LP
    Ac=[ KPsi(1:(n+1)*nu,:); % Lava str. obm. v LP
          -KPsi(1:(n+1)*nu,:)]; % ...
    b=[ repmat(eye(nu),n+1,1)*uh; % Prava str. obm. v LP
        -repmat(eye(nu),n+1,1)*ul]; % ...
    for j=1:nu % Ries. LP pre kazde u
        x=linprog(-f(j,:),Ac,b,[],[],[],[],o); %LP1
        umax(j,1)=f(j,:)*x; % Vypocet umax,j
        if ul===-uh; % Asymetr. obm. vstupu
            x=linprog(f(j,:),Ac,b,[],[],[],[],o); %LP2
            umin(j,1)=f(j,:)*x; % Vypocet umin,j
        else % Symetr. obm. vstupu
            umin(j,1)=-umax(j,1); % Vypocet umin,j
        end
    end
    KPsi(nu*(n+1)+1:nu*(n+2),:) = f; % K*Phi^(n+1) do KPhi
end % Ukoncenie cyklu
KPsi=KPsi(1:nu*(n+1),:); % Uloz KPsi^i, i=0,...,n
end % Ukoncenie funkcie
```

□

**Príklad 8.4.** Určte dĺžku horizontu koncových obmedzení pre ohraničený MPC problém z Pr. 8.2.

MPC problém si definujeme identicky ako v Pr. 8.2:

Zdrojový kód 8.4. prikladhorizontNc.m

```
clear; clc
A=[1.1 2;0 0.95]; B=[0;0.0787]; C=[-1 1]; % Stavovy model
uh=1; ul=-uh; % Obmedzenia vstupu
Q=C'*C; R=1; np=2; % Vahy a horizont
[K,P]=iterdlqr(A,B,Q,R,100); % Vahov. koncov. stavu
K=-K; % Zaporna spatna vazba
Psi=A+B*K; % Dynamika uzavr. reg. obvodu
```

Ďalej zavoláme vytvorenú funkciu horizontkoncobmedzeni(), ktorej výstupom okrem horizontu  $n_c$  sú aj príslušné hodnoty  $\mathbf{K}\Psi^i, i = 0, \dots, n_c$ , potrebné pre neskôršiu formuláciu koncových obmedzení. V tomto prípade sa v každej iterácii algoritmu rieši iba jeden lineárny program, keďže daný systém má jeden vstup, ktorý musí splňať symetrické obmedzenia.

```
[nc,KPsi]=horizontkoncobmedzeni(K,Psi,uh,ul); % Vypoc. nc
disp(['nc = ', num2str(nc)]) % Zobraz nc
```

Po spustení skriptu získavame dĺžku horizontu koncových obmedzení  $n_c = 4$ , čo súhlasí s grafickým výstupom na 8.6 v Pr. 8.2.  $\square$

Pripomeňme si, že uvedený algoritmus na určenie horizontu koncových obmedzení, ako aj jeho programové spracovanie, boli odvodené pre prípad iba obmedzení vstupu. Ak by tiež stavy riadeného systému podliehali obmedzeniam, potom je pri výpočte horizontu  $n_c$  potrebné zaručiť splnenie  $\underline{\mathbf{x}} \leq (\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x} \leq \bar{\mathbf{x}}$  pre  $i = n_p, n_p + 1, \dots, \infty$  (pozri (8.19)), čo je možné dosiahnuť rozšírením algoritmu.

Prienik obmedzení vstupu a stavu (8.18) v režime 2 zloženého MPC riadiaceho zákona teda predstavuje maximálnu invariantnú cieľovú množinu  $\Omega \subset \mathbb{R}^{n_x}$ , pre ktorú platí  $\Omega = \Pi_{n_c}$ , kde  $n_c$  je offline vypočítaný horizont koncových obmedzení.

## 8.5 Problém MPC so zaručenou stabilitou

Na základe predchádzajúcej diskusie môžeme teda definovať nasledujúci ohraničený MPC problém so zaručenou stabilitou a zlučiteľnosťou: Na základe predchádzajúcej diskusie môžeme teda definovať nasledujúci ohraničený MPC problém so zaručenou stabilitou a riešiteľnosťou:



$$\min_{\vec{\mathbf{u}}_k} \left[ \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{k+i}^T \mathbf{Q} \mathbf{x}_{k+i} + \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \right] \quad (8.31a)$$

vzhľadom na podmienky:

$$\underline{\mathbf{u}} \leq \mathbf{u}_{k+i} \leq \bar{\mathbf{u}} \quad \text{pre } i = 0, 1, \dots, n_p - 1, \quad (8.31b)$$

$$\underline{\mathbf{x}} \leq \mathbf{x}_{k+i} \leq \bar{\mathbf{x}} \quad \text{pre } i = 1, 2, \dots, n_p - 1, \quad (8.31c)$$

$$\underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x}_{k+n_p} \leq \bar{\mathbf{u}} \quad \text{pre } i = 0, 1, 2, \dots, n_c, \quad (8.31d)$$

$$\underline{\mathbf{x}} \leq (\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x}_{k+n_p} \leq \bar{\mathbf{x}} \quad \text{pre } i = 0, 1, 2, \dots, n_c, \quad (8.31e)$$

$$\mathbf{x}_k = \mathbf{x}(k), \text{ resp. } \mathbf{x}_k = \hat{\mathbf{x}}(k) \quad (8.31f)$$

$$\mathbf{x}_{k+i+1} = \mathbf{A}\mathbf{x}_{k+i} + \mathbf{B}\mathbf{u}_{k+i} \quad \text{pre } i = 0, 1, 2, \dots, n_p - 1, \quad (8.31g)$$

$$\mathbf{u}_{k+i} = \mathbf{K}\mathbf{x}_{k+i}, \quad \text{pre } i \geq n_p, \quad (8.31h)$$

kde (8.31d)–(8.31e) predstavujú obmedzenia koncového stavu  $\mathbf{x}_{k+n_p} \in \Omega = \Pi_{n_c}$ .



Cieľová množina  $\Omega$ , definovaná koncovými obmedzeniami  $\mathbf{x}_{k+n_p} \in \Omega$  a váhovanie koncového stavu  $\mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p}$  zaručujú rekurzívnu riešiteľnosť a stabilitu systému v uzavretej slučke s prediktívnym regulátorom.

Uvedený problém prediktívneho riadenia si upravíme dosadením predikčného modelu (8.31g) do účelovej funkcie (8.31a), a ďalej zoskupením obmedzení vstupov a stavov (8.31b)–(8.31e) do jednej množiny nerovností v zmysle  $\vec{\mathbf{u}}_k$ , čím ho transformujeme na ekvivalentný problém kvadratického programovania v tvare:

$$\min_{\vec{\mathbf{u}}_k} \left( \frac{1}{2} \vec{\mathbf{u}}_k^T \mathbf{H} \vec{\mathbf{u}}_k + \underbrace{\mathbf{x}_k^T \mathbf{G}^T}_{\mathbf{g}^T} \vec{\mathbf{u}}_k \right) \quad (8.32a)$$

vzhľadom na obmedzenia:

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \underbrace{\mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k}_{\mathbf{b}_c}. \quad (8.32b)$$

Optimalizačný problém (8.32) riešime priebežne, t. j. v každom kroku  $k$  pre aktuálnu hodnotu meraného, respektíve odhadnutého stavu,  $\mathbf{x}_k$ . Získame tak postupnosť optimálnych vstupov  $\vec{\mathbf{u}}_k^* = [\mathbf{u}_k^{*T} \ \mathbf{u}_{k+1}^{*T} \ \dots \ \mathbf{u}_{k+n_p}^{*T}]^T$ , z ktorej prvý prvok  $\mathbf{u}_k^*$  implementujeme ako optimálny akčný zásah na riadený systém.

Platí, že ak existuje riešenie MPC problému (8.31) v počiatočnom čase  $k = 0$ , potom daný problém bude riešiteľný v nasledujúcich časových okamihoch  $k > 0$  a systém preto možno stabilizovať zo všetkých počiatočných podmienok – stavov  $\mathbf{x}(0) \in S_\Omega$ . Zdôrazníme si tiež nasledujúce skutočnosti:

- Nárast výpočtovej náročnosti kvadratického programu (8.31), resp. (8.32) v dôsledku pridania koncových obmedzení (8.31d)–(8.31e) je nepatrny, keďže ide o lineárne obmedzenia a rozhodujúci vplyv na ňu má počet optimalizačných premenných,  $n_p \cdot n_u$ ;
- veľkosť (objem) množiny riešiteľných počiatočných stavov,  $S_\Omega$  (8.17), sa so zväčšujúcim predikčným horizontom  $n_p$  (ako aj s veľkosťou cieľovej množiny  $\Omega$ ) zväčšuje;
- optimálna predikovaná hodnota účelovej funkcie,  $J^*(k)$  sa so zväčšujúcim predikčným horizontom  $n_p$  zmenšuje.

Ilustrujme si formuláciu MPC problému (8.31) a jeho transformáciu a riešenie v tvare (8.32) na nasledujúcim príklade.

**Príklad 8.5.** Navrhnite a implementujte prediktívny regulátor podľa (8.31) pre systém, váhovanie a obmedzenia z Pr. 8.2. Vyšetrite tiež vplyv dĺžky predikčného horizontu na kvalitu riadenia v uzavretej slučke.

Ešte pred samotným simulačným skriptom si vytvorime funkciu, pomocou ktorej k daným obmedzeniam vstupov (8.31b) pridáme koncové obmedzenia (8.31d)–(8.31e), ktoré sme si formulovali v Pr. 8.4. Pre všeobecnosť uvažujme  $n_u$  vstupov. Ako vidíme, obmedzenia vstupov sú formulované v zmysle  $\mathbf{u}_{k+i}$ ,  $i = 0, 1, \dots, n_p - 1$ , a koncové obmedzenia v zmysle  $\mathbf{x}_{k+n_p}$  (pričom ich výsledkom je vstup), t.j.

$$\underline{\mathbf{u}} \leq \mathbf{u}_{k+i} \leq \bar{\mathbf{u}} \quad \text{pre } i = 0, 1, \dots, n_p - 1, \quad (8.33a)$$

$$\underline{\mathbf{u}} \leq \mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x}_{k+n_p} \leq \bar{\mathbf{u}} \quad \text{pre } i = 0, 1, 2, \dots, n_c. \quad (8.33b)$$

Kedže našou optimalizačnou premennou je  $\vec{\mathbf{u}}_k$ , musíme obmedzenia (8.33b) využadiť v zmysle  $\vec{\mathbf{u}}_k$ . Postupujeme pritom analogicky ako v prípade obmedzení stavov, pozri Kap. 5.3.3. Zoberme si horné obmedzenie z (8.33b)

$$\mathbf{K}(\mathbf{A} + \mathbf{B}\mathbf{K})^i \mathbf{x}_{k+n_p} \leq \bar{\mathbf{u}}, \quad i = 0, 1, \dots, n_c,$$

kde dynamiku uzavretého regulačného obvodu si označíme ako  $\Psi = \mathbf{A} + \mathbf{B}\mathbf{K}$ , a zároveň pre koncový stav podľa (3.21) platí  $\mathbf{x}_{k+n_p} = \mathbf{M}_{n_p} \mathbf{x}_k + \mathbf{N}_{n_p} \vec{\mathbf{u}}_k$ , kde  $\mathbf{M}_{n_p}$  a  $\mathbf{N}_{n_p}$  označujú posledné,  $n_p$ -té, blokové riadky predikčných matíc  $\mathbf{M}$  a  $\mathbf{N}$ . Po úprave dostávame

$$\mathbf{K}\Psi^i (\mathbf{M}_{n_p} \mathbf{x}_k + \mathbf{N}_{n_p} \vec{\mathbf{u}}_k) \leq \bar{\mathbf{u}}, \quad i = 0, 1, \dots, n_c,$$

a potom

$$\mathbf{K}\Psi^i \mathbf{N}_{n_p} \vec{\mathbf{u}}_k \leq \bar{\mathbf{u}} - \mathbf{K}\Psi^i \mathbf{M}_{n_p} \mathbf{x}_k, \quad i = 0, 1, \dots, n_c. \quad (8.34)$$

Podobne, ak si zoberieme horné obmedzenie z (8.33a), t.j.

$$\mathbf{u}_{k+i} \leq \bar{\mathbf{u}}, \quad i = 0, 1, \dots, n_p - 1.$$

Podľa (5.30) z Kap. 5.3.2 môžeme túto nerovnosť zapísat kompaktne ako

$$\mathbf{I} \vec{\mathbf{u}}_k \leq \mathbf{1} \bar{\mathbf{u}}, \quad (8.35)$$

kde  $\mathcal{I} \in \mathbb{R}^{n_u \cdot n_p \times n_u \cdot n_p}$  je matica s jednotkovými maticami  $\mathbf{I} \in \mathbb{R}^{n_u \times n_u}$  na diagonále a  $\mathbf{1} \in \mathbb{R}^{n_u \cdot n_p \times n_u}$  je blokový stĺpcový vektor pozostávajúci z matíc  $\mathbf{I}$ .

Obmedzenia (8.34) a (8.35) môžeme spoločne zapísat v nasledujúcom tvare:

$$\underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \\ \mathbf{K}\mathbf{N}_{n_p} & & & & \\ \mathbf{K}\Psi\mathbf{N}_{n_p} & & & & \\ \vdots & & & & \\ \mathbf{K}\Psi^{n_c-1}\mathbf{N}_{n_p} & & & & \\ \mathbf{K}\Psi^{n_c}\mathbf{N}_{n_p} & & & & \end{bmatrix}}_{\mathbf{A}_c} \underbrace{\begin{bmatrix} \mathbf{u}_k \\ \mathbf{u}_{k+1} \\ \vdots \\ \mathbf{u}_{k+n_p-2} \\ \mathbf{u}_{k+n_p-1} \\ \vdots \\ \bar{\mathbf{u}}_k \end{bmatrix}}_{\vec{\mathbf{u}}_k} \leq \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \\ \mathbf{I} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix}}_{\bar{\mathbf{b}}_0} \bar{\mathbf{u}} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \\ -\mathbf{K}\mathbf{M}_{n_p} \\ -\mathbf{K}\Psi\mathbf{M}_{n_p} \\ \vdots \\ -\mathbf{K}\Psi^{n_c-1}\mathbf{M}_{n_p} \\ -\mathbf{K}\Psi^{n_c}\mathbf{M}_{n_p} \end{bmatrix}}_{\bar{\mathbf{B}}_0} \mathbf{x}_k, \quad (8.36)$$

kde  $\mathbf{0}$  je matica núl príslušného rozmeru ( $n_u \times n_u$  na ľavej strane a  $n_u \times n_x$  na pravej strane nerovnosti (8.36)). Je zrejmé, že matice  $\bar{\mathbf{A}}_c$ ,  $\bar{\mathbf{b}}_0$  a  $\bar{\mathbf{B}}_0$  majú v zmysle (8.33a)–(8.33b)  $(n_p + n_c) \cdot n_u$  riadkov, resp.  $n_p + n_c$  blokových riadkov.

Analogicky si odvodíme kombinované obmedzenia stavov a koncového stavu pri zohľadnení iba dolných obmedzení v (8.33a) a (8.33b). Ich spojením s obmedzeniami (8.36) získame výslednú množinu obmedzení vstupov MPC problému so zloženou účelovou funkciou, ktorú po úprave môžeme zapísat ako:

$$\underbrace{\begin{bmatrix} \bar{\mathbf{A}}_c \\ -\bar{\mathbf{A}}_c \end{bmatrix}}_{\mathbf{A}_c} \vec{\mathbf{u}}_k \leq \underbrace{\begin{bmatrix} \bar{\mathbf{b}}_0 \bar{\mathbf{u}} \\ -\bar{\mathbf{b}}_0 \bar{\mathbf{u}} \end{bmatrix}}_{\mathbf{b}_0} + \underbrace{\begin{bmatrix} \bar{\mathbf{B}}_0 \\ -\bar{\mathbf{B}}_0 \end{bmatrix}}_{\mathbf{B}_0} \mathbf{x}_k, \quad (8.37a)$$

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \underbrace{\mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}_k}, \quad (8.37b)$$

$$\mathbf{A}_c \vec{\mathbf{u}}_k \leq \mathbf{b}_c. \quad (8.37c)$$

Uvedený postup formulácie obmedzení vstupov na horizonte  $n_p + n_c$  si v MATLABe implementujeme do funkcie `obmedzeniastabilita()`, ktorej vstupnými argumentmi sú matice vstupu a dynamiky systému, horizont predikcie a koncových obmedzení, a blokový stĺpcový vektor pozostávajúci z výrazov  $\mathbf{K}\Psi^i$ ,  $i = 0, \dots, n_c$ , ktorý je výstupom funkcie `horizontkoncobmedzeni()`.

Zdrojový kód 8.5. `obmedzeniastabilita.m`

```

function [Ac,b0,B0] = obmedzeniastabilita(A,B,np,nc,KPsi
    ,ul,uh)
[nx,nu]=size(B); % Pocet stavov a vstupov
[M,N] = predikcia(A,B,np); % Predikcne matice M a N
Mnp = M(nx*np-(nx-1):nx*np,:); % Posledny blok. riadok M
Nnp = N(nx*np-(nx-1):nx*np,:); % Posledny blok. riadok N
Ac=eye(np*nu); % Inic. lavej strany obm.
B0=zeros(np*nu,nx); % Inic. pravej strany obm.
for i=0:nc % Pridaj konc. obm. na horizonte nc
    Ac(nu*(np+i)+1:nu*(np+i+1),:) = KPsi(i*nu+1:(i+1)*nu
        ,:) * Nnp; % Lava str. konc. obmedzeni
    B0(nu*(np+i)+1:nu*(np+i+1),:) = -KPsi(i*nu+1:(i+1)*nu
        ,:) * Mnp; % B0 na pravej str. konc. obm.
end
Ac=[Ac;-Ac]; % Pridaj dolne obm. u a xnp
B0=[B0;-B0]; % ...
b0=[ repmat(eye(nu),np+nc+1,1)*uh; % b0 obm. na u a xnp
    -repmat(eye(nu),np+nc+1,1)*ul]; % ...
end % Ukoncenie funkcie

```

Výstupom funkcie sú matice príslušných obmedzení v tvare (8.37b), ktoré môžeme priamo použiť na formuláciu a riešenie MPC problému v tvare QP (8.32).

Pristúpme teraz k riešeniu príkladu v zmysle zadania. Začnime ako v Pr. 8.2:

Zdrojový kód 8.6. `MPCstabilita.m`

```

clear; close all; clc
A=[1.1 2;0 0.95]; B=[0;0.0787]; C=[-1 1]; % Stavovy model
nx=length(A); % Pocet stavov
uh=1; ul=-uh; % Obmedzenia vstupu
Q=C'*C; R=1; np=5; % Vahy a horizont
[K,P]=iterdlqr(A,B,Q,R,100); % Vahovanie koncoveho stavu
K=-K; Psi=A+B*K; % Dynamika uzav. reg. obvodu

```

Pokračujeme výpočtom matíc účelovej funkcie zloženého MPC riadiaceho zákona, výpočtom dĺžky horizontu koncových obmedzení,  $n_c$  a ďalej výpočtom ľavej a pravej strany obmedzení vstupov na horizonte  $n_p + n_c$  pomocou predvytvorennej funkcie `obmedzeniastabilita()`

```
[H, G, F]=ucelovafunkcia(A, B, np, Q, R, P); % Ucelova f
[nc, KPsi]=horizontkoncobmedzeni(K, Psi, uh, ul); % Horiz. nc
[Ac, b0, B0]=obmedzeniastabilita(A, B, np, nc, KPsi, ul, uh);
```

Vypočítaný Hessián si tradične prevedieme na jeho symetrickú formu a vypneme informatívne výpisu riešiča.

```
H=(H+H')/2; % Symetr. H
o=optimoptions('quadprog','Display','none'); % Vyp. vypis
```

Prediktívne riadenie systému simulujeme v uzavretej slučke z počiatočného stavu  $\mathbf{x}(0) = [-7 \ 0.5]^T$ , pričom optimálny zásah vypočítame riešením MPC problému v tvare QP (8.32) pomocou rutiny `quadprog`. Uvažujme  $n = 25$  krokov simulácie.

```
n=25; % Trvanie simulacie
x=[-7 0.5]'; % Pociat. stav x(0)
for k=1:n % Simulacna slucka
    u (:,k)=quadprog(H, G*x (:,k), Ac, b0+B0*x (:,k)
    ,[],[],[],[],[],o); % Riesenie QP MPC
    x (:,k+1)=A*x (:,k)+B*u (1,k); % Simulacia modelu
end
```

Premenná  $\mathbf{x}$  obsahuje simulované priebehy jednotlivých stavov systému v uzavretej slučke. Na porovnanie si tiež vypočítajme predikciu stavov z identickej počiatočnej podmienky  $\mathbf{x}_0 = \mathbf{x}(0)$ . Predikované stavy na horizonte  $n_p$ ,  $\vec{\mathbf{x}}_k$ , získame pomocou predikčnej rovnice (3.12), t.j.

$$\vec{\mathbf{x}}_0 = \mathbf{M}\mathbf{x}_0 + \mathbf{N}\vec{\mathbf{u}}_0,$$

```
[M, N]=predikcia(A, B, np); % Predikcne matice
xpr=M*x (:,1)+N*u (:,1); % Predikcia z x(0)
```

kde posledný blok vektora  $\vec{\mathbf{x}}_0$  predstavuje koncový stav  $\mathbf{x}_{0+n_p}$  a získame ho pomocou:

```
xnp=xpr (end-nx+1:end); % Koncovy stav
```

Pre predikované stavy za horizontom  $n_p$ , t.j. v režime 2 zloženého MPC riadiaceho zákona, platí  $\mathbf{x}_{0+n_p+i} = \Psi^i \mathbf{x}_{0+n_p}$ ,  $i \geq 1$ . Kedže predikovanú trajektóriu budeme ďalej porovnávať so skutočnou trajektóriou stavov, obmedzme si celú predikciu

na  $n$  krokov simulácie, z čoho  $i = 0, 1, \dots, n - n_p$ , a teda si nájdeme predikovanú postupnosť stavov na horizonte  $n_p + 1, \dots, n - n_p$  ako

$$\vec{x}_{0+n_p} = \begin{bmatrix} \Psi \\ \Psi^2 \\ \vdots \\ \Psi^{n-n_p} \end{bmatrix} \mathbf{x}_{0+n_p},$$

a zaraďme ju do premennej  $x_{\text{pr}}$  za predikciu  $\vec{x}_k$  a počiatočný stav  $\mathbf{x}_0$ :

```
for i=1:n-np
    xpr=[xpr;Psi^(i)*xnp];
    % Predik. traj. x
end
xpr=[x(:,1);xpr];
    % Pridaj poc. stav
```

Získame tak predikciu stavov v  $k = 0$  v tvare  $\begin{bmatrix} \mathbf{x}_0^T & \mathbf{x}_{0+1}^T & \dots & \mathbf{x}_{n-n_p-1}^T & \mathbf{x}_{n-n_p}^T \end{bmatrix}^T$ .

Aby sme si porovnali kvalitu riadenia systému v uzavretej slučke s kvalitou riadenia v zmysle stavov a vstupov predikovaných v čase  $k = 0$ , vyčíslime si tak tiež kumulatívne hodnoty príslušných účelových funkcií. Hodnotu účelovej funkcie MPC v uzavretej slučke vypočítame ako

$$J_{\text{us}} = \sum_{k=0}^n \left( \mathbf{x}(k)^T \mathbf{Q} \mathbf{x}(k) + \mathbf{u}(k)^T \mathbf{R} \mathbf{u}(k) \right),$$

čo v MATLABe implementujeme ako

```
Jus=0;
for j=1:n
    % Hodnota J v uzavretej slucke
    Jus=Jus+x(:,j)'*Q*x(:,j)+u(1,j)'*R*u(1,j);
end
```

Optimálnu predikovanú hodnotu účelovej funkcie v čase  $k = 0$  získame ako

$$\begin{aligned} J_0 &= \sum_{i=0}^{n_p-1} \left( \mathbf{x}_{0+i}^T \mathbf{Q} \mathbf{x}_{0+i} + \mathbf{u}_{0+i}^T \mathbf{R} \mathbf{u}_{0+i} \right) + \mathbf{x}_{k+n_p}^T \mathbf{P} \mathbf{x}_{k+n_p} \\ &= \vec{\mathbf{u}}_0^T \mathbf{H} \vec{\mathbf{u}}_0 + 2\mathbf{x}_0^T \mathbf{G}^T \vec{\mathbf{u}}_0 + \mathbf{x}_0^T \mathbf{F} \mathbf{x}_0, \end{aligned}$$

čo v MATLABe implementujeme ako

```
Jpr0=u(:,1)'*H*u(:,1)+2*x(:,1)'*G'*u(:,1)+x(:,1)'*F*x
    % Predikovana J v x(0)
```

Získané hodnoty  $J_{\text{us}}$  a  $J_0$  si následne necháme vypísať v príkazovom okne:

```
disp(['Jus = ',num2str(Jus)])
    % Vypis Jcl
disp(['J(0) = ',num2str(Jpr0)])
    % Vypis J(0)
```

Nakoniec si graficky znázorníme stavovú trajektóriu predikovanú v  $\mathbf{x}_0 = \mathbf{x}(0)$  a skutočnú stavovú trajektóriu získanú simuláciou systému z rovankého počiatočného stavu v uzavretej slučke so stabilizujúcim MPC regulátorom:

```

figure('color','w') % Obrazok
plot(xpr(1:nx:end-1),xpr(2:nx:end),'-bo') % Pred. traj. x
hold on; grid on % Podrž graf, mriezka
plot(x(1,:),x(2,:),'-r+') % Skutoc. stav. traj.
xlabel('x_1'); ylabel('x_2'); % Oznacenie osi x a y
legend('Predikcia v x_0','Uzavreta slucka') % Legenda
axis([-7 1 -0.1 0.8]) % Hran. grafu

```

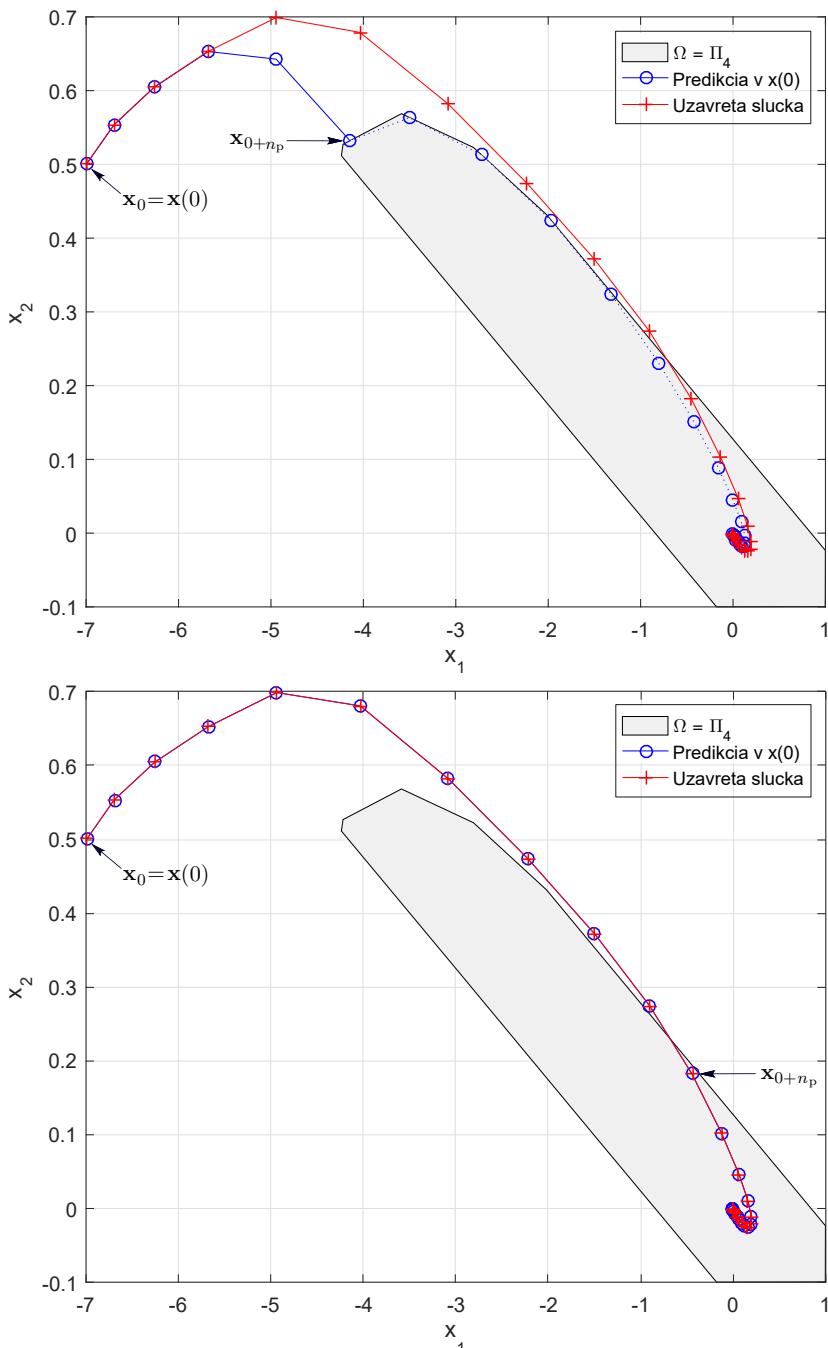
Grafický výstup je znázornený na Obr. 8.8, v hornom grafe pre predikčný horizont  $n_p = 5$  a v dolnom grafe pre  $n_p = 10$ . Oba grafy sú tiež pre ilustráciu doplnené o maximálnu invariantnú cielovú množinu  $\Omega = \Pi_4$ , ktorú sme získali v Pr. 8.2. Ako je zrejmé, prediktívny regulátor úspešne priviedie systém do žiadaneho, rovnovážneho stavu. Všimnime si, že v prípade  $n_p = 5$  sa predikovaná stavová trajektória líši od skutočnej, zatiaľ čo pri uvažovaní  $n_p = 10$  sú tieto identické. Túto skutočnosť možno interpretovať aj numericky, pomocou hodnoty účelovej funkcie predstavujúcej ukazovateľ kvality riadenia.

Tab. 8.1 uvádza hodnoty účelovej funkcie, predikovanú a skutočnú v uzavretej slučke, pre rôzne dĺžky  $n_p$ . Platí, že kvalita riadenia (daná  $J_{us}$ ) sa s rastúcim  $n_p$  potenciálne zlepšuje, v dôsledku poklesu predikovanej hodnoty účelovej funkcie. Pre daný počiatočný stav  $\mathbf{x}_0 = \mathbf{x}(0)$  pritom existuje určitá dĺžka horizontu,  $n_{p,\infty}$ , pri ktorej už nie je možné dosiahnuť zlepšenie kvality riadenia v uzavretej slučke pre  $n_p = n_{p,\infty}$ . Je to tak, lebo koncové obmedzenia musia byť od určitej hodnoty  $n_p$  neaktívne a pri ďalšom zvyšovaní  $n_p$  preto nemôže dôjsť k poklesu hodnoty účelovej funkcie. Kvalita riadenia pri  $n_p = n_{p,\infty}$  je preto rovnaká ako keby sme uvažovali MPC problém s nekonečným počtom optimalizačných premenných, t. j. nekonečne dlhou optimalizovanou postupnosťou vstupov  $\vec{u}_k$ .

Tabuľka 8.1. Porovnanie hodnôt účelovej funkcie – získaných predikciou v  $\mathbf{x}(0) = [-7 \ 0.5]^T$  (prvý riadok) a skutočných, získaných simuláciou v uzavretej slučke z  $\mathbf{x}(0)$  (druhý riadok) – pre rôzne dĺžky predikčného horizontu.

$n_p$	5	6	7	10	$>10$
$J_0$	295.23	287.67	286.89	286.65	286.65
$J_{us}$	286.65	286.65	286.65	286.65	286.65

□



Obr. 8.8. Grafické znázornenie stavových trajektórií: predikovanej v stave  $\mathbf{x}(0)$  a skutočnej, simulovanej v uzavretej slučke. Horný graf:  $n_p = 5$ , spodný graf:  $n_p = 10$ . Cieľová množina  $\Omega = \Pi_4$  je pre ilustráciu znázornená sivou farbou.



Poznamenajme, že maximálnu invariantnú zlúčiteľnú množinu  $\Omega$  možno pre daný MPC problém vypočítať aj efektívnejším spôsobom, s využitím nástrojov teórie invariantných množín, pozri napr. publikácie [77, 18, 19]. Uvedený spôsob určenia koncových obmedzení pomocou horizontu koncových obmedzení je sice priamočiary, ale je zrejmé, že veľká časť týchto obmedzení môže byť redundantná. t. j. nepotrebná na definovanie minimálnej reprezentácie výslednej cieľovej množiny. Počet obmedzení je tak väčší ako je nevyhnutné. Vo všeobecnosti výhodnejšie je preto siahnuť po algoritmoch výpočtu invariantných množín využívajúcich nástroje výpočtovej geometrie, ktoré sú implementované napr. v softvérovom balíku Multi Parametric Toolbox [65], navrhnutom pre použitie v MATLABe. Pomocou týchto možno pri výpočte cieľovej množiny  $\Omega$  efektívne zohľadniť nielen obmedzenia vstupov, stavov a iné, ale aj nenulové referenčné úrovne stavov a vstupov a na základe množiny  $\Omega$  tiež rekurzívne vypočítat príslušnú množinu riešiteľných počiatočných stavov,  $S_\Omega$ .

Poznamenajme tiež, že polyedrické cieľové množiny môžu byť v určitých prípadoch veľmi zložité, najmä ak uvažujeme predikčné modely vyšších rádov, pričom táto zložitosť sa samozrejme prenika aj do výpočtovej náročnosti online riešenia príslušného MPC problému. Z tohto dôvodu existuje mnoho techník aproximácie danej cieľovej množiny jednoduchšími invariantnými polytopickejmi, prípadne elipsoidálnymi množinami, definovanými spravidla menším počtom ohraničení. Výhodou je spomínaná nižšia náročnosť výpočtu riešenia MPC problému, avšak na úkor teoreticky konzervatívnejšieho riadenia. Praktické aspekty návrhu a implementácie týchto tzv. cieľových množín nižšej zložitosti<sup>11</sup> sú bližšie a prehľadne opísané v publikácii [74].

<sup>11</sup>angl. low-complexity terminal sets

# Literatúra

- [1] Double Pendulum on a Cart. <https://www.youtube.com/watch?v=B6vr1x6KDaY>. Video. [citované 15.05.2018].
- [2] Triple Pendulum on a Cart. <https://www.youtube.com/watch?v=cyN-CRNrb3E>. Video. [citované 15.05.2018].
- [3] B. Anderson, A. Bennick, a M. Salciccioli. The Michigan Chemical Process Dynamics and Controls Open Textbook: Model Predictive Control. [online], 2007. [citované 17.09.2015]; Dostupné z <https://controls.engin.umich.edu/wiki/index.php/MPC>.
- [4] E. Arnold, O. Sawodny, J. Neupert, a K. Schneider. Anti-sway system for boom cranes based on a model predictive control approach. In *2005 IEEE International Conference on Mechatronics and Automation*, volume 3, s. 1533–1538 Vol. 3, 2005.
- [5] W. Ashby a R. Conant. Every good regulator of a system must be a model of that system. *International Journal of Systems Science*, 1(2):89–97, 1970.
- [6] AspenTech. Aspen DMCplus: Take Control and Drive Better Results. [online], 2018. [citované 09.05.2018]; Dostupné z <https://www.aspentechn.com/en/products/pages/aspen-dmcplus>.
- [7] A. Y. Barraud. A numerical algorithm to solve  $\mathbf{A}^T \mathbf{X} \mathbf{A} - \mathbf{A} = \mathbf{Q}$ . In *1977 IEEE Conference on Decision and Control including the 16th Symposium on Adaptive Processes and A Special Symposium on Fuzzy Set Theory and Applications*, s. 420–423, Dec 1977.
- [8] R. Bartels a G. W. Stewart. Solution of the matrix equation  $\mathbf{AX} + \mathbf{XB} = \mathbf{C}$ . *Communications of the ACM*, 15(9):820–826, 1972.

- [9] C. Belavý. *Teória Automatického Riadenia II: Návody na cvičenia*. Slovenská vysoká škola technická v Bratislave: Strojnícka Fakulta, Bratislava, Slovakia, 1. vydanie, 1990.
- [10] A. Bemporad. Model predictive control: Basic concepts. [online], 2009. Controllo di Processo e dei Sistemi di Produzione - A.a. 2008/08; [citované 03.09.2015]; Dostupné z <http://www.seas.upenn.edu/~ese680/papers/IntroductionMPC.pdf>.
- [11] A. Bemporad, N. A. Bozinis, V. Dua, M. Morari, a E. N. Pistikopoulos. Model predictive control: A multi-parametric programming approach. In S. Pierucci, editor, *European Symposium on Computer Aided Process Engineering-10*, volume 8 of *Computer Aided Chemical Engineering*, s. 301 – 306. Elsevier, 2000.
- [12] A. Bemporad, L. Chisci, a E. Mosca. On the stabilizing property of the zero terminal state receding horizon regulation. *Automatica*, 30(12):2013–2015, December 1994.
- [13] A. Bemporad, M. Morari, V. Dua, a E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3 – 20, 2002.
- [14] J. E. Bertram a R. E. Kalman. Control systems analysis and design via “second method” of Lyapunov. *Journal of Basic Engineering*, 82(2):371–400, 1960.
- [15] R. R. Bitmead, M. Gevers, a V. Wertz. *Adaptive optimal control: The thinking man’s GPC*. Prentice Hall, 1990.
- [16] F. Blanchini. Ultimate boundedness control for uncertain discrete-time systems via set-induced Lyapunov functions. *IEEE Transactions on Automatic Control*, 39(2):428–433, 1994.
- [17] F. Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, November 1999.
- [18] F. Blanchini a S. Miani. *Set-Theoretic Methods in Control*. Birkhäuser Boston, 2008.
- [19] F. Borrelli, A. Bemporad, a M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, Cambridge, UK, 2017.

- [20] F. Borrelli, C. Jones, a M. Morari. Model Predictive Control: Part I - Introduction. [online], 2009. Fall Semester 2014 (revised August 2014); [citované 04.09.2015]; Dostupné z [http://www.mpc.berkeley.edu/mpc-course-material/MPC2014\\_PartI\\_Intro\\_handsout.pdf](http://www.mpc.berkeley.edu/mpc-course-material/MPC2014_PartI_Intro_handsout.pdf).
- [21] S. Boyd. Linear quadratic Lyapunov theory. [online], 2008. Prezentácia pre EE363, Lekcia 13. [citované 23.05.2018]; Dostupné z <https://stanford.edu/class/ee363/lectures/lq-lyap.pdf>.
- [22] S. Boyd a L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [23] M. Brdicka a A. Hladík. *Teoretická mechanika*. Academia, Praha, 1987.
- [24] C. Brezinski. *Computational Aspects of Linear Control*. Kluwer Academic Publisher, Dordrecht, Netherlands, 2002.
- [25] J. V. Burke. Quadratic programming. [online], 2014. MATH 408 Lecture Notes. [citované 07.06.2018]; Dostupné z <https://sites.math.washington.edu/~burke/crs/408/notes/qp/qp1.pdf>.
- [26] M. Cannon. Model predictive control. Lecture notes. Michaelmas Term 2005 (4 Lectures), Course code 4ME44. University of Oxford, Oxford, United Kingdom., October 2005.
- [27] M. Cannon a B. Kouvaritakis. Optimizing Prediction Dynamics for Robust MPC. *IEEE Transactions on Automatic Control*, 50(11):1892–1597, November 2005.
- [28] F. E. Cellier a E. Kofman. *Continuous System Simulation*. Springer US, New York, NY, 2006.
- [29] E. Cheever. State Space Representations of Linear Physical Systems. [online], 2015. [citované 04.06.2018]; Dostupné z <http://lpsa.swarthmore.edu/Representations/SysRepSS.html>.
- [30] C. C. Chen a L. Shaw. On receding horizon feedback control. *Automatica*, 18:349–352, 1982.
- [31] P. D. Christofides, R. Scattolini, D. M. de la Peña, a J. Liu. Distributed model predictive control: A tutorial review and future research directions. *Computers & Chemical Engineering*, 51:21–41, 2013. CPC VIII.

- [32] D. W. Clarke, C. Mohtadi, a P. S. Tuffs. Generalized predictive control, Part I: The basic algorithm. *Automatica*, 23(2):137–148, February 1987.
- [33] D. W. Clarke, C. Mohtadi, a P. S. Tuffs. Generalized predictive control, Part II: Extensions and interpretations. *Automatica*, 23(2):149–160, February 1987.
- [34] D. W. Clarke a R. Scattolini. Constrained receding-horizon predictive control. *IEE Proceedings Part D*, 138(4):347–354, 1991.
- [35] D. W. Clarke a L. Zhang. Long-range predictive control using weighting-sequence models. *IEE Proceedings Part D*, 134(3):187–195, 1987.
- [36] R. W. Cottle. William Karush and the KKT Theorem. *Documenta Mathematica*, Extra volume: Optimization Stories(-):255–269, 2012.
- [37] C. R. Cutler a B. C. Ramaker. Dynamic matrix control - a computer control algorithm. In *Automatic Control Conference*, San Francisco, United States of America, 1980.
- [38] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.
- [39] R. M. C. de Keyser a A. R. V. Cauwenberghhe. Extended prediction self-adaptive control. In *IFAC Symposium on Identification and System Parameter Estimation*, s. 1317–1322, York, Great Britain, 1985.
- [40] P. Del Moral. Non linear filtering: Interacting particle solution. *Markov Processes and Related Fields*, 2(4):555–580, 1996.
- [41] “Duckduckstop” a Wikimedia. Head shot of deceased academic, Harold W. Kuhn. [online], 2015. Historický obrázok. [citované 19.06.2018]; Dostupné z [https://en.wikipedia.org/wiki/File:Harold\\_W.\\_Kuhn.jpg](https://en.wikipedia.org/wiki/File:Harold_W._Kuhn.jpg).
- [42] “Ellywa” a Wikimedia. Lagrange portrait. [online], 2005. [citované 19.06.2018]; Dostupné z [https://commons.wikimedia.org/wiki/File:Lagrange\\_portrait.jpg](https://commons.wikimedia.org/wiki/File:Lagrange_portrait.jpg).
- [43] Engineering a T. H. Wiki. First-Hand:The Unscented Transform. Website. [http://ethw.org/First-Hand:The\\_Unscented\\_Transform](http://ethw.org/First-Hand:The_Unscented_Transform).
- [44] ETH Life. Rudolf Kalman recognized for filter. [online], 2008. Fotografia. [citované 15.06.2018]; Dostupné z [http://www.ethlife.ethz.ch/archive\\_articles/080103-draperprize/kalmannew-1-hires.jpeg](http://www.ethlife.ethz.ch/archive_articles/080103-draperprize/kalmannew-1-hires.jpeg).

- [45] P. Eykhoff. *System Identification*. Wiley and Sons, London, 1994.
- [46] P. Falcone, H. E. Tseng, F. Borrelli, J. Asgari, a D. Hrovat. MPC-based yaw and lateral stabilisation via active front steering and braking. *Vehicle System Dynamics*, 46(sup1):611–628, 2008.
- [47] I. Fantoni a R. Lozano. *Non-linear Control for Underactuated Mechanical Systems*. Springer-Verlag, London, 2. vydanie, 2002.
- [48] R. Fletcher. *Practical Methods of Optimization*. Wiley, New York, NY, United States of America, 2000.
- [49] M. G. Forbes, R. S. Patwardhan, H. Hamadah, a R. B. Gopaluni. Model predictive control in industry: Challenges and opportunities. *IFAC-PapersOnLine*, 48(8):531 – 538, 2015. 9th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2015.
- [50] A. Forsyth. *Calculus of Variations*. Dover Publications, New York, NY, 1960. str. 17–20 a 29.
- [51] G. F. Franklin, J. D. Powell, a M. L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, Boston, MA, 3. vydanie, 1997.
- [52] Gameroom Show. Zoltan fortune teller machine. Online, 2006. [citované 16.03.2018]; Licencované na základe CC BY-SA 4.0. Dostupné z [https://commons.wikimedia.org/wiki/File:Zoltan\\_Fortune\\_Teller.jpg](https://commons.wikimedia.org/wiki/File:Zoltan_Fortune_Teller.jpg).
- [53] C. E. García, D. M. Prett, a M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335 – 348, 1989.
- [54] E. G. Gilbert a K. T. Tan. Linear systems with state and control constraints: The theory and application of maximal output admissible sets. *IEEE Transactions on Automatic Control*, 36(9):1008–1020, 1991.
- [55] H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 2. vydanie, 1960. str. 44.
- [56] G. Golub, S. Nash, a C. V. Loan. A hessenberg-schur method for the problem  $\mathbf{AX} + \mathbf{XB} = \mathbf{C}$ . *IEEE Transactions on Automatic Control*, 24(6):909–913, December 1979.
- [57] A. Gray, M. Ali, Y. Gao, J. Hedrick, a F. Borrelli. A unified approach to threat assessment and control for automotive active safety. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1490–1499, Sept 2013.

- [58] M. S. Grewal a A. P. Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems*, 30(3):69–78, 2010.
- [59] M. S. Grewal a A. P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. John Wiley & Sons, Inc., New York, NY, 4. vydanie, 2015.
- [60] M. Gulan. *Nonlinear model predictive control of underactuated mechatronic systems*. PhD thesis, 2015.
- [61] M. Gulan, G. Takács, N. A. Nguyen, S. Olaru, P. Rodríguez-Ayerbe, a B. Rohal-Ilkiv. Efficient embedded model predictive vibration control via convex lifting. *IEEE Transactions on Control Systems Technology*, s. 1–15, 2018. V tlači.
- [62] F. Gustafsson a G. Hendeby. Some relations between extended and unscented kalman filters. *IEEE Transactions on Signal Processing*, 60(2):545–555, 2012.
- [63] M. Halčin. Vnorené interpolačné riadenie reálneho mechatronického systému. Master’s thesis, Slovenská Technická Univerzita v Bratislave, 2006. Diplomová práca, SjF-5226-54996. Vedúci práce: M. Gulan.
- [64] V. Havlena a J. Štěcha. *Moderní teorie řízení*. ČVUT, Praha, 1999. Skriptum.
- [65] M. Herceg, M. Kvasnica, C. Jones, a M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, s. 502–510, Zürich, Switzerland, July 17–19 2013. <http://control.ee.ethz.ch/~mpt>.
- [66] S. Iles, J. Matusko, a F. Kolonic. Real-time Predictive Control of 3D tower crane. In *23rd IEEE International Symposium on Industrial Electronics*, s. 224–230, June 2014.
- [67] InSapphoWeTrust. United airlines—N797UA. Online, 2013. [citované 20.07.2018]; Licencované na základe CC BY-SA 2.0 Generic. Dostupné z <https://www.flickr.com/photos/skinnylawyer/9278357641/>.
- [68] E. Jarlebring. Lecture notes in numerical linear algebra: Numerical methods for Lyapunov equations. [online], 2017. Prezentácia pre predmet “Numerical linear algebra” SF3580, Lekcia 13. [citované 24.05.2018]; Dostupné z <https://people.kth.se/~eliasj/NLA/matrixeqs.pdf>.

- [69] P. A. Jensen a J. F. Bard. *Operations Research Models and Methods*. John Wiley & Sons, Hoboken, NJ, 2014.
- [70] S. J. Julier a J. K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *In Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls*, s. 182–193, Orlando, FL, 1997.
- [71] S. J. Julier a J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.
- [72] R. E. Kalman. A new approach to linear filtering and prediction problems. 82 (Series D):35–45, 01 1960.
- [73] A. Karas. *Stabilizujúce prediktívne riadenie systémov s obmedzeniami*. PhD thesis, Slovak University of Technology in Bratislava, Bratislava, Slovakia, 2002.
- [74] A. Karas, B. Roháč-Ilkiv, a C. Belavý. *Praktické aspekty prediktívneho riadenia*. Slovak University of Technology in Bratislava / Slovenská E-Akadémia, n.o., Bratislava, Slovakia, 1. vydanie, November 2007.
- [75] S. S. Keerthi a E. G. Gilbert. Optimal, infinite horizon feedback law for a general class of constrained discrete time systems: Stability and moving-horizon approximations. *Journal of Optimization Theory and Application*, 57:265–293, 1988.
- [76] E. C. Kerrigan. *Robust Constraint Satisfaction: Invariant Sets and Predictive Control*. PhD thesis, Control Group, Department of Engineering, University of Cambridge, Great Britain, November 2000.
- [77] E. C. Kerrigan. *Robust constraint satisfaction: invariant sets and predictive control*. Ph.D. dissertation, University of Cambridge, UK, 2000.
- [78] E. C. Kerrigan, J. L. Jerez, S. Longo, a G. A. Constantinides. Number representation in predictive control. *IFAC Proceedings Volumes*, 45(17):60 – 67, 2012. 4th IFAC Conference on Nonlinear Model Predictive Control.
- [79] M. Kinnaert. Adaptive generalized predictive controller for MIMO systems. *Journal of Process Control*, 50(1):161–172, 1989.
- [80] G. Kitagawa. An algorithm for solving the matrix equation  $\mathbf{X} = \mathbf{F}\mathbf{X}\mathbf{F}^T + \mathbf{S}$ . *International Journal of Control*, 25(5):745–753, 1977.

- [81] B. Kouvaritakis, J. A. Rossiter, a A. O. T. Chang. Stable generalised predictive control: An algorithm with guaranteed stability. *IEE Proceedings Part D*, 139(4):349–362, July 1992.
- [82] H. W. Kuhn a S. Nasar. *The Essential John Nash*. Princeton University Press, Princeton, NJ, 2007.
- [83] M. Kvasnica, J. Hledík, I. Rauová, a M. Fikar. Complexity reduction of explicit model predictive control via separation. *Automatica*, 49(6):1776–1781, 2013.
- [84] M. Kvasnica, B. Takács, J. Holaza, a S. D. Cairano. On region-free explicit model predictive control. In *Proc. 54th Conf. Decision and Control*, s. 3669–3674, Osaka, Japan, 2015.
- [85] H. Kwakernaak a R. Sivan. *Linear Optimal Control Systems*. Wiley-Interscience / John Wiley & Sons, New York, NY, United States of America, 1972.
- [86] W. H. Kwon a D. G. Byun. Receding horizon tracking control as a predictive control and its stability properties. *International Journal of Control*, 50(5):1807–1824, 1989.
- [87] W. H. Kwon, H. H. Choi, D. G. Byun, a S. B. Noh. Recursive solution of generalized predictive control and its equivalence to receding horizon tracking control. *Automatica*, 28(6):1235–1238, 1992.
- [88] W. H. Kwon a A. E. Pearson. On the stabilisation of a discrete constant linear system. *IEEE Transactions on Automatic Control*, 20(6):800–801, 1975.
- [89] W. H. Kwon a A. E. Pearson. On feedback stabilization of time-varying discrete linear systems. *IEEE Transactions on Automatic Control*, 23:479–481, 1978.
- [90] “Lamro” a Wikimedia. Historical photo of Canadian mathematician Albert W. Tucker. [online], 2014. Historický obrázok. [citované 19.06.2018]; Dostupné z [https://en.wikipedia.org/wiki/File:Albert\\_W.\\_Tucker.gif](https://en.wikipedia.org/wiki/File:Albert_W._Tucker.gif).
- [91] J. H. Lee. A lecture on model predictive control. [online], 2005. School of Chemical and Biomolecular Engineering, Center for Process Systems Engineering, Georgia Inst. of Technology; Prepared for Pan American Advanced Studies Institute Program on Process Systems Engineering

- ; [citované 03.09.2015]; Dostupné z <http://cepac.cheme.cmu.edu/pasilectures/lee/LectureonMPC-JHL.pdf>.
- [92] LIEBHERR. Cycoptronic: Accurate sway free load motion. [online], 2014. Video. [citované 09.05.2018]; Dostupné z [https://www.youtube.com/watch?v=U1IDFS\\_eMac](https://www.youtube.com/watch?v=U1IDFS_eMac).
- [93] LIEBHERR. LIEBHERR LHM 550 Mobile Harbour Crane. [online], 2014. [citované 09.05.2018]; Dostupné z [https://www.liebherr.com/external/products/products-assets/232681/IMG\\_390x390/liebherr-lhm-550-mobile-harbour-crane-bulk-handling-pactronic-agam-kuwait-1.jpg](https://www.liebherr.com/external/products/products-assets/232681/IMG_390x390/liebherr-lhm-550-mobile-harbour-crane-bulk-handling-pactronic-agam-kuwait-1.jpg).
- [94] LIEBHERR. Anti-sway system Cycoptronic. [online], 2018. [citované 09.05.2018]; Dostupné z <https://www.liebherr.com/en/sau/products/maritime-cranes/maritime-technology/cycoptronic/cycoptronic.html>.
- [95] A. Liniger. ORCA - MPCC with static and dynamic obstacle avoidance. [online], Automatic Control Laboratory, ETH Zurich, 2015. Video. [citované 09.05.2018]; Dostupné z <https://www.youtube.com/watch?v=JoHfJ6LEKVo>.
- [96] A. Liniger, A. Domahidi, a M. Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2014.
- [97] L. Ljung. *System Identification: Theory for the User*. PTR Prentice Hall, Upper Saddle River, NJ., 2. vydanie, 1999.
- [98] A. M. Lyapunov. *Problème général da la stabilité du mouvement*. Reprinted in 1966 by Academic Press, New York, United States of America, 1893.
- [99] Y. Ma, F. Borrelli, B. Hencey, B. Coffey, S. Bengea, a P. Haves. Model predictive control for the operation of building cooling systems. *Control Systems Technology, IEEE Transactions on*, 20(3):796–803, May 2012.
- [100] Y. Ma, F. Borrelli, B. Hencey, A. Packard, a S. Bortoff. Model predictive control of thermal energy storage in building cooling systems. In *Proceedings of the 48th IEEE Conference on Decision and Control 2009 held jointly with the 2009 28th Chinese Control Conference.*, s. 392–397, 2009.
- [101] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, Upper Saddle River, NJ, 2000.

- [102] T. Mathworks. Choose an ODE solver. Website. <https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html>, [citované 23.05.2018].
- [103] T. Mathworks. Grey-box model estimation. Website. <https://www.mathworks.com/help/ident/grey-box-model-estimation.html>, [citované 03.06.2018].
- [104] T. Mathworks. ode45. Website. <https://www.mathworks.com/help/matlab/ref/ode45.html>, [citované 23.05.2018].
- [105] D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.
- [106] D. Q. Mayne a H. Michalska. Receding horizon control of non-linear systems. *IEEE Transactions on Automatic Control*, 35(5):814–824, 1990.
- [107] D. Q. Mayne, J. B. Rawlings, C. V. Rao, a P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, June 2000.
- [108] B. Messner, D. Tilbury, R. Hill, a J. D. Taylor. Control Tutorials for MATLAB and Simulink (CTMS). [online], 2018. [citované 30.05.2018]; Dostupné z <http://ctms.engin.umich.edu/CTMS>.
- [109] H. Michalska a D. Q. Mayne. Robust receding horizon control of constrained nonlinear systems. *IEEE Transactions on Automatic Control*, 38(11):1623–1633, 1993.
- [110] J. Mikleš a M. Fikar. *Modelovanie, identifikácia a riadenie procesov 2—Identifikácia a optimálne riadenie*. Vydavateľstvo STU v Bratislave, 2004.
- [111] C. Moler a C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.
- [112] E. Mosca a J. Zhang. Stable redesign of predictive control. *Automatica*, 28(6):1229–1233, June 1992.
- [113] Y. Nesterov a A. Nemirovskii. *Interior-Point Polynomial Methods in Convex Programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, Pennsylvania, United States of America, 1994.
- [114] I. Newton. *Philosophiae Naturalis Principia Mathematica*. Edmond Halley / Royal Society, London, 1. vydanie, July 1867.

- [115] J. Nocedal a S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.
- [116] V. Obetková, A. Mamrillová, a A. Košinárová. *Teoretická mechanika*. Alfa, Bratislava, 1990.
- [117] J. J. O'Connor a E. F. Robertson. MacTutor History of Mathematics archive: Jacopo Riccati. [online], 2018. School of Mathematics and Statistics University of St Andrews, Scotland. [citované 25.05.2018]; Dostupné z <http://www-history.mcs.st-andrews.ac.uk/Biographies/Riccati.html>.
- [118] A. W. Ordys a D. W. Clarke. A state-space description for GPC controllers. *International Journal of Systems Science*, 24(9):1727–1744, 1993.
- [119] T. Pappas, A. Laub, a N. R. Sandell. On the numerical solution of the discrete-time algebraic Riccati equation. *IEEE Transactions on Automatic Control*, 25(4):631–641, 1980.
- [120] T. Penzl. Numerical solution of generalized Lyapunov equations. *Advances in Computational Mathematics*, 8(1):33–48, Jan 1998.
- [121] F. A. Potra a S. J. Wright. *Primal-dual interior-point methods*. SIAM, 1997.
- [122] J. A. Primbs a V. Nevistić. Constrained finite receding horizon linear quadratic control. Technical Report CIT-CDS 97-002, California Institute of Technology, Pasadena, CA, United States of America, 1997.
- [123] S. J. Qin a T. A. Badgwell. An overview of industrial model predictive control technology. In *Proceedings of Chemical Process Control - V.*, volume 93 of *AICHE Symposium Series*, s. 232–256, Tahoe City, CA, United States of America, 1997.
- [124] C. V. Rao, J. B. Rawlings, a J. H. Lee. Constrained linear state estimation—moving horizon approach. *Automatica*, 37(10):1619–1628, 2001.
- [125] C. V. Rao, J. B. Rawlings, a D. Q. Mayne. Constrained state estimation for nonlinear discrete-time systems: stability and moving horizon approximations. *Markov Processes and Related Fields*, 48(2):246–258, 2003.
- [126] C. V. Rao, S. J. Wright, a J. B. Rawlings. Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications*, 99(3):723–757, Dec 1998.

- [127] J. B. Rawlings a R. Amrit. *Optimizing Process Economic Performance Using Model Predictive Control*, s. 119–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
- [128] J. Richalet, A. Rault, J. L. Testud, a J. Papon. Model predictive heuristic control: Application to industrial processes. *Automatica*, 14(2):413–428, February 1978.
- [129] W. Rugh. *Linear System Theory*. Prentice Hall, Upper Saddle River, NJ, 1996.
- [130] F. Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation. Master’s thesis, Royal Institute of Technology (Kungliga Tekniska högskolan), Stockholm, Sweden, 2015. Master’s Degree Project.
- [131] P. O. M. Scokaert a J. B. Rawlings. Constrained linear quadratic regulation. *IEEE Transactions on Automatic Control*, 43(8):1163–1169, 1998.
- [132] S. L. Shah, C. Mohtadi, a D. W. Clarke. Multivariable adaptive control without a prior knowledge of the delay matrix. *Systems and Control Letters*, 9:295–306, 1987.
- [133] L. F. Shampine a M. W. Reichelt. The MATLAB ODE suite. *SIAM Journal on Scientific Computing*, 18:1–22, 1997.
- [134] D. Simon. *Optimal State Estimation: Kalman,  $H_\infty$ , and Nonlinear Approaches*. Wiley–Interscience, Hoboken, NJ, 1. vydanie, 2006.
- [135] R. Soeterboek. *Predictive control: a unified approach*. Prentice Hall, New York, United States of America, 1992.
- [136] Space Exploration Technologies Corp. (SpaceX). Falcon 9 first stage landing on droneship “Of Course I Still Love You”. [online], 2017. [citované 09.05.2018]; Dostupné z <http://www.spacex.com/media-gallery/detail/149441/9516>.
- [137] G. Takács, J. Vachálek, a B. Rohal-Ikkiv. *Identifikácia sústav*. Slovenská Technická Univerzita v Bratislave, 2014.
- [138] G. Takács, P. Zometa, R. Findeisen, a B. Rohal’-Ikkiv. Efficiency and performance of embedded model predictive control for active vibration attenuation. In *2016 European Control Conference (ECC)*, s. 1334–1340, June 2016.

- [139] The Mathworks, Inc. Optimization Toolbox Release Notes. [online], 2018. [citované 20.06.2018]; Dostupné z <https://www.mathworks.com/help/optim/release-notes.html>.
- [140] US Dept of Commerce, National Oceanic and Atmospheric Administration (NOAA), National Weather Service. Surface Analysis: Hi Res Version. [online], 2018. Detail predpovedi počasia zo dňa . [citované 15.06.2018]; Dostupné z [http://www.nws.noaa.gov/outlook\\_tab.php](http://www.nws.noaa.gov/outlook_tab.php).
- [141] R. Waldvogel. MPC Control of a Ball on Plate System: Theory and Implementation. Master's thesis, Eidgenössische Technische Hochschule (ETH) Zürich, Zurich, Switzerland, 2010. Master thesis.
- [142] G. Welch a G. Bishop. An introduction to the kalman filter. 8:1–16, 2001.
- [143] S. Wright. Applying new optimization algorithms to model predictive control. In *Fifth International Conference on Chemical Process Control (CPC V)*, s. 147–155. CACHE Publications, 1997.
- [144] L. Wunderli. MPC based Trajectory Tracking for 1:43 scale Race Cars. Master's thesis, Eidgenössische Technische Hochschule (ETH) Zürich, Zurich, Switzerland, 2011. Semester thesis.
- [145] L. Wunderli. MPC based Trajectory Tracking for 1:43 scale Race Cars. [online], 2011. Prezentácia. [citované 18.04.2015]; Dostupné z <https://goo.gl/EQuc2t>.
- [146] B. E. Ydstie. Extended horizon adaptive control. In *Proceedings of 9th IFAC World Congress*, Budapest, Hungary, 1984.
- [147] T. W. Yoon. *Robust adaptive predictive control*. PhD thesis, Department of Engineering Science, Oxford University, Parks Road, Oxford, Great Britain, 1994.
- [148] X. Zhang. Stochastic MPC Group. [online], 2014. ETH Zürich; [citované 16.09.2015]; Dostupné z <http://control.ee.ethz.ch/~smpc/>.



# Zdroj vybraných obrázkov

Nasledujúce obrázky a ilustrácie majú postavenie verejného vlastníctva alebo sú voľným dielom, a preto nie sú chránené autorským právom: Obr. 4.16 (s. 209).; Obr. 4.15 (s. 207); Space Exploration Technologies Corp. (SpaceX): Obr. 1.18 (s. 60) [136]; US Dept of Commerce, National Oceanic and Atmospheric Administration (NOAA), National Weather Service: Obr. 6.1 (s. 282).

Historické obrázky (portréty) pripadajú do doktríny spravodlivého používania (fair use policy) v právnom systéme Spojených štátov: Obr. 6.21(a), (s. 314); Obr. 6.21(b), (s. 314); Obr. 6.21(c), (s. 314); Obr. 2.22, (s. 128).

Nasledujúci obrázok je reprodukovaný v zmysle licencie Creative Commons 4.0 share-alike: CC BY-SA 4.0: Obr. 3.1 (s. 148) [52].

Nasledujúci obrázok je reprodukovaný v zmysle licencie Creative Commons 2.0 share-alike: CC BY-SA 2.0: Obr. 7.8 (s. 345) [67].

Nasledujúce obrázky sú reprodukované v tejto publikácii písomným súhlasom a explicitným povolením ich pôvodných autorov alebo právnych zástupcov:

- Michal Halčin: Obr. 1.17 (s. 59) [63];
- Lukas Wunderli: Obr. 1.16(b) (s. 58) [144];
- Alex Liniger a Automatic Control Laboratory, ETH Zurich Obr. 1.16(a) (s. 58) [95];
- LIEBHERR: Obr. 1.13(a) (s. 54) [93]; a Obr. 1.13(b) (s. 54) [92];
- René Waldvogel Obr. 1.1 (s. 36) [141]; a Obr. 1.19 (s. 60) [141].

Zároveň autorom ďakujem za túto ich pomoc v príprave tejto publikácie.

Ilustrácie a fotografie tu nevymenované sú pôvodnou prácou autorov tejto knihy.



# Register

- APVV, 27, 31, 32
- diskretizácia, 100, 102, 105, 111, 171
- GitHub, 33
- gradient, 186, 198, 224, 287, 295, 309
- Gulan, M., 32
- horizont
- koncových obmedzení, 399, 405
  - pohyblivý, 43, 51, 146, 170
  - predikcie, 39, 150, 187, 247
- identifikácia, 32, 42, 69, 99
- integračná zložka, 42, 348, 353
- invertované kyvadlo, 72, 84, 105, 129, 141, 304, 364
- Kalmanov filter, 72, 118, 121, 123
  - nevoňavý, 146
  - rozšírený, 72, 138, 140, 141, 377
- Kálmán, R.E., 128
- KKT podmienky, 313, 315
- konvexnosť, 62, 179, 299
- kvadratické programovanie, 277, 296
  - algoritmus, 320
  - v MATLABe, 296
- kvalita riadenia, 45, 178, 188, 193, 409
- Lagrange, J.-L., 72, 307
- Lagrangeove multiplikátory, 307, 312
- linearizácia, 62, 83, 85, 89, 90, 93, 96
- Ljapunov, A.M., 208
- LQ riadenie, 36, 126, 205, 241, 364
- mechatronika, 31, 32, 72
- metóda
- aktívnych množín, 303, 320
  - vnútorného bodu, 325
- množina
- aktívnych obmedzení, 303, 320
  - cieľová, 386, 387, 396, 398
  - invariantná, 398, 401
  - riešiteľných počiatočných stavov, 398
- model, 69, 73
- modelovanie, 69
- obmedzenia, 180, 255, 290
  - koncové, 385, 395, 398
  - nezlučiteľné, 291
  - prírastkov vstupu, 355, 359
  - stavov, 265, 269, 399
  - výstupov, 273
  - vstupov, 260, 399
  - zlučiteľnosť, 396, 407
  - zmiešané, 270
- odchýlka riadenia, 188, 344
- odhad stavu, 118, 138, 377
- optimalizácia, 178, 181, 277
- optimalizačná úloha, 178
- PID riadenie, 35, 38, 41, 58, 364
- pozorovateľ stavu, 71, 118
- pozorovateľnosť, 114, 117

- predikčné matice, 155, 161, 166  
 predikcia, 147  
     výstupov, 172  
     voľnej odozvy, 152  
     vynútenej odozvy, 160  
 prediktívne riadenie, 224, 333  
     akademické aplikácie, 57  
     algoritmus, 340, 341  
     analógie, 39  
     bez obmedzení, 224, 241  
     budúcnosť, 61  
     ekonomické, 66  
     explicitné, 63  
     história, 48  
     myšlienka, 38  
     nelineárne, 61  
     pojmy, 42  
     priemyselné aplikácie, 54  
     rýchla dynamika, 62  
     rozložené, 66  
     s obmedzeniami, 276, 333  
     stochastické, 65  
     v MATLABe, 333  
     v Simulinku, 361  
     vnorené, 64  
 prírastky vstupu, 274  
 problém  
     MPC, 227, 276, 334, 407  
     QP, 278, 299, 334  
 programová podpora, 33  
 rad  
     mocninový, 102  
     Taylorov, 86  
 recenzenti, 30  
 riaditeľnosť, 114, 117  
 Riccati, J., 206  
 riešiteľnosť, 383  
 rovnica  
     dynamiky, 78, 102, 148  
     Ljapunova, 208, 214  
     merania, 78, 102, 148  
     predikčná, 160  
     Riccatiho, 210  
 rovnice  
     Lagrangeove, 75  
     pohybové, 72, 76  
 saturácia vstupov, 256  
 Simulink, 82, 136, 361  
 sledovanie referencie, 58, 343  
 stabilita, 111, 383, 388, 407  
     Ljapunova, 111, 388  
 stavový model  
     diskrétny, 102  
     lineárny, 83, 89  
     linearizovaný, 89  
     nelineárny, 78  
     spojitý, 78, 89  
 STU, 29, 31, 32  
     Strojnícka fakulta, 29, 31, 32  
     ÚAMAI, 29, 31, 32  
 šum  
     merania, 121, 138, 377  
     procesný, 121, 138, 377  
 Takács, G., 31  
 Taylorov rad, 100  
 tvarovač nultého rádu, 100, 101  
 účelová funkcia, 177, 187, 195, 234  
     zložená, 202, 220  
 uzavretá slučka, 230, 339  
 váhovanie  
     koncového stavu, 206, 214, 386  
     stavov, 190  
     vstupov, 192  
 VEGA, 27, 31, 32  
 vrstevnice, 281, 287  
 vzorkovanie, 100

doc. Ing. Gergely Takács, PhD.—Ing. Martin Gulan, PhD.

## **ZÁKLADY PREDIKTÍVNEHO RIADENIA**

Vydala Slovenská technická univerzita v Bratislave vo Vydavatelstve  
SPEKTRUM STU,  
Bratislava, Vazovova 5, v roku 2018.

Edícia učebníc

Rozsah 435 strán, 167 obrázkov, 4 tabuľky, X,XXX AH, X,XXX VH, 1. vydanie, náklad 100 výtlačkov, edičné číslo XXXX, tlač ForPress NITRIANSKE TLAČIARNE, s. r. o.

85-223-2018

ISBN 978-80-227-4826-1





# Chceš spoznať budúcnosť?

...lebo táto učebnica je práve o predpovedaní budúcnosti. Na predpoved' budúcnosti—inými slovami na predikciu—však nebudeme potrebovať postupy tradičného čínskeho veštenia z ruky, ani tarotové karty, astrológiu, jasnovidectvo alebo iné paranormálne schopnosti.

V oblasti automatického riadenia máme k dispozícii niečo oveľa lepšie: matematický opis fyzikálnych dejov, mechatronických systémov a priemyselných procesov. Pomocou týchto modelov dokážeme sformulovať strategiu riadenia, ktorá v záujme čo najlepšej kvality zahŕňa do procesu rozhodovania aj vplyv budúceho vývoja dynamiky sústavy, a tak vypočíta akčné, zásahy, ktoré sú v danom čase a situácii optimálne. Prediktívne riadenie (MPC) je moderná metóda spätnoväzbového riadenia, ktorá sa vďaka neustálemu vývoju výpočtového hardvéru dostáva do popredia pri návrhu high-tech produktov a pri riadení zložitých procesov a systémov v priemysle. Prediktívne riadenie zo svojej podstaty dokáže rešpektovať fyzikálne, bezpečnostné a ekonomické obmedzenia riadenej sústavy, a často poskytuje lepší výkon ako tradičné algoritmy riadenia. Táto učebnica je určená pre dovoľštekným pre tých čitateľov, ktorí doteraz nemali žiadnu skúsenosť s prediktívnym riadením, ale majú základné vedomosti z oblasti teórie riadenia a identifikácie sústav.

Nové poznatky v učebnici „Základy prediktívneho riadenia“ sú pre úplnosť didaktického prístupu bohatu graficky ilustrované, doplnené príkladmi a počítačovými simuláciami. Učebnica sa vyhýba zbytočne zložitému výkladu niektorých konceptov a dáva dôraz na pedagogickú kvalitu textu. Cieľom je poskytnúť čitateľovi potrebné základy MPC, tak, aby v budúcnosti mohol sebavedome otvoriť aj práce s hlbším záberom problematiky, a v konečnom dôsledku tak realizovať moderné interpretácie algoritmov prediktívneho riadenia.



ISBN 978-80-227-4826-1

