České vysoké učení technické v Praze

Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky

Obor: Inženýrská informatika

Zaměření: Softwarové inženýrství

Iterativní lokální dynamické programování pro návrh duálního řízení

Iterative local dynamic programming for dual control

BAKALÁŘSKÁ PRÁCE

Vypracoval: Michal Vahala

Vedoucí práce: Ing. Václav Šmídl, Ph.D.

Rok: 2010

zadání práce

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

V Praze dne ....

Michal Vahala

Poděkování

Děkuji ... za ...

Michal Vahala

Název práce:
Iterativní lokální dynamické programování pro návrh duálního řízení

Autor: Michal Vahala

Obor: Inženýrská informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Václav Šmídl, Ph.D.

Konzultant: —

Abstrakt: abstrakt

Klíčová slova: klíčová slova

Title:
Iterative local dynamic programming for dual control

Author: Michal Vahala

Abstract: abstrakt

Key words: klíčová slova

Obsah

Seznam použitého označení

iLDP	iterativní lokální dynamické programování
LQG	lineáně kvadraticky gaussovské řízení (Linear-Quadratic-Gaussian)
iLQG	iterativní LQG

Úvod

Skutečný svět se nikdy nechová přesně podle matematických rovnic, protože ty jsou vždy jen jakýmsi zjednodušením nebo přiblížením. V reálném světě se vyskytuje mnoho neznámých veličin, poruch, nepředvídatelných vlivů a ani naše měřící přístroje nejsou přesné. Chceme-li efektivně řídit nějaký systém, musíme si být těchto vlivů vědomi a zahrnout je do našich uvažování. Situace se však ještě může zkomplikovat, když jeden nebo více parametrů neznáme. To může nastat z různých důvodů, například příšlušné čidlo nebo měřící přístroj nemůžeme nebo nechceme (například z důvodů vysoké ceny) instalovat a tedy o velikosti příslušné hodnoty můžeme jen usuzovat ze známých dat. Ještě složitější situace nastane, když uvažujeme neznámý parametr proměnný.

Máme tedy dva cíle, musíme systém co nejlépe řídit a současně se snažit o co nejpřesnější určení neznámých parametrů. Tyto dva postupy jsou však obecně v rozporu, protože parametry se nejlépe určují, když je systém vybuzen a nechová se optimálně. Právě tento rozpor a nalezení kompromisu, který povede k jeho řešení, je podstatou duálního řízení.

Pro přiblížení ilustrujme problém na jednoduchém příkladě: Uvažujme elektromotor s možností řídit napětí na vstupu motoru a měřit příslušné proudy. Jedná se tedy o systém se dvěma vstupy a dvěma výstupy. Cílem našeho řízení je dosažení požadovaných otáček rotoru. Ovšem otáčky a ani polohu hřídele měřit nemůžeme. Máme o nich však znalost v podobě středních hodnot a variancí. Naší snahou je co nejpřesněji určit hodnotu otáček a polohy hřídele a současně systém řídit tak, abychom dosáhly požadované hodnoty otáček. Tyto dvě snahy jsou ale v rozporu, protože nejvíce informací o neznámých parametrech získáme, když je motor vybuzen. Tedy například se prudce rozjíždí, brzdí, rychle mění rychlost nebo kmitá, což se projevuje v proudech, které máme možnost měřit. Ale právě vybuzení motoru je v rozporu se snahou o dobré řízení, protože chyba, které se dopustíme je většinou nepřijatelná. Naopak, když se systém snažíme řídit, bez dostatečné znalosti jeho parametrů, s velkou pravděpodobností selžeme.

Námětem této bakalářské práce je algoritmus *iterativního lokálního dynamického programování* (iLDP) jako jedna z metod pro řešení problému duálního řízení. Algoritmus byl navržen a popsán v článku [7]. Jak už prozrazuje název algoritmu, jedná se o iterační metodu. Tedy stručně řečeno, algoritmus vyjde od nějakého počátečního řízení, které je ovšem nutno dodat jako apriorní informaci a v cyklech (iteracích) tuto řídící strategii vylepšuje, za účelem získání řízení optimálního. Dále se jedná o metodu lokální, což v můžeme jednoduše chápat tak, že kandidáti na „vylepšení“ řízení jsou vybírání z jistého, zatím blíže nespecifikovaného, okolí původní řídící strategie. Nakonec algoritmus využívá obecné schéma dynamického programování, které bude blíže popsáno v dalším textu.

Cílem této práce bylo seznámit se s obecnou tématikou duálního řízení a detailněji s konkrétním algoritmem - iterativním lokálním dynamickým programováním. Následně

tento algoritmus implementovat a aplikovat na jednoduchý systém. Otestovat jeho funkčnost a schopnost řídit a to i v porovnání s jinými metodami a algoritmy. Dále se pokusit implementovat algoritmus iLDP pro složitější systém blíže praktické aplikaci, konkrétně se jedná o synchronní motor s permanentními magnety. Otestovat funkčnost a případně srovnat s dostupnými výsledky jiných řídících strategii. Na základě získaných výsledků posoudit výhody a nevýhody algoritmu a jeho použitelnost na další úlohy.

Hlavním přínosem práce je otestování vlastností algoritmu iLDP na jiných problémech, než pro které byla vyvinuta autory. Objevení kladů a záporů algoritmu a dále díky srovnání s jinými algoritmy získání přehledu, pro které praktické aplikace je vhodnější respektive méně vhodný než srovnávané metody. Prvotní očekávání pro srovnání algoritmu iLDP a řízení získaného pomocí principu separace jsou, že iLDP bude pomalejší co do výpočetního času, avšak přesnost získaných výsledků bude lepší. Dále je očekávána nezanedbatelná závislost výsledného řízení na volbě použitých aproximací a apriorní řídící strategie.

1 Teorie duálního řízení

1.1 Základní pojmy

1.1.1 Systém a řízení

Systém

Základním pojmem, se kterým budeme v textu pracovat je *systém*. Obdobně jako základní pojmy zejména v matematických vědách (bod, množina, algoritmus,...), nelze tento pojem exaktně definovat. Systém si můžeme představit jako jistý „objekt“, často bude reprezentovat objekt skutečného světa. Hlavní vlastností systému je, že má zpravidla jeden nebo více vstupů, pomocí kterých mu můžeme předávat informaci – řízení a jeden nebo více výstupu, což jsou hodnyty, které pozorujeme. Co se odehrává uvnitř systému však obecně nevíme. Řízení, které budeme dodávat systému na vstup bude v textu značeno písmenem u. Analogicky bude písmenem y označena pozorovaná hodnota na výstupu.

Chování systému, to je jakým výstupem reaguje na vstup, popisujeme dle [3] obecně diferenciální rovnicí respektive soustavou diferenciálních rovnic vyšších řádů. Jedná se o takzvaný vnější popis. Tento druh popisu, pohlíží na systém „zvenku“ bez skutečné znalosti, co se odehrává uvnitř systému a jaká je jeho podstata. Vnější popis obvykle obdržíme při odvození modelu systému z fyzikálních rovnic. Omezení, která z něj plynou, se snažíme odstranit zavedením vnitřního (stavového) popisu, kdy (soustavu) diferenciálních rovnic vyššího řádu, převedeme vhodnou volbou nových proměnných x na soustavu diferenciálních rovnic prvního řádu. Proměnné x označujeme jako stavové proměnné.

Řízení

Naším úkolem je pro zadaný systém nalézt regulátor, tedy obecně řízení u takové, které dodané na vstup způsobí, že systém se bude „chovat podle našich požadavků“. To zpravidla znamená, že hodnoty výstupní veličiny y dosáhnou (nebo se přiblíží s danou přesností) požadované hodnotě v podobě referenčního signálu, který regulátor dostává z vnějšku a současně dodrží předem stanovená omezení. Práce je ovšem zaměřena na řízení složitějších systémů, u kterých jeden nebo více parametrů neznáme přesně. Tedy některý (více) z koeficientů v rovnicích popisujících systém není znám. Máme však o něm jistou statistickou informaci v podobě jeho očekávané hodnoty a variance. Dále je-li systém nelineární, jsou výsledné rovnice příliš složité a tedy analyticky neřešitelné. Pro numerické řešení, jsou rovnice systému zpravidla převáděny do diskrétního tvaru.

Řízení obecně dělíme podle [3] na dva typy: *Přímovazební řízení* užíváme v případě, kde je k dispozici přesný matematický model systému a je vyloučen výskyt neurčitostí.

Toto řízení nevyužívá žádné zpětné informace od systému a regulátor pracuje pouze s referenčním signálem. Naproti tomu *zpětnovazební řízení* využívá i informace o skutečném výstupu systému a snaží se tak eliminovat chyby v důsledku neurčitostí a chyb způsobenych nepřesností modelu.

Duální řízení

Chceme navrhnout regulátor pro zadaný systém s neznámými parametry. Úkoly jsou tedy dva: 1. *opatrnost* - efektivně systém řídit a 2. *buzení* - určit neznáme parametry. Tyto dva přístupy jsou ale obecně v rozporu. Abychom mohli systém dobře řídit, potřebujeme znát parametry co nejpřesněji. Nejvíce informací o parametrech však získáme, když je systém vybuzen a nechová optimálně. Tyto pojmy není snadné kvantifikovat, ale velmi často se projevují v konkrétních řídících schématech. Naším úkolem je pokusit nalézt nějaký kompromis mezi oběma úkoly. Právě tento přístup je označován jako *duální řízení* [2].

1.2 Dynamické programování

1.2.1 Formulace problému

V textu budeme pracovat zpravidla s diskrétním systémem, ve smyslu systému s diskrétním časem, protože výpočty jsou prováděny ve většině případů problematiky duálního řízení numericky. Rovnice popisující systém jsou však zpravidla ve spojitém tvaru, (model často vychází ze skutečnosti, popřípadě fyzikálních zákonů). V tomto případě provádíme diskretizaci.

Základní problém je formulován podle [2] následovně:

Uvažujme stavový popis diskrétního dynamického systému

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, \ldots, N-1 \ , \tag{1.1}$$

kde x_k je stavová proměná, u_k řízení a w_k náhodná porucha, vše v čase k při celkovém časovém horizontu N. Na řízení u_k klademe omezení, že může nabývat pouze hodnot z neprázdné monožiny $U_k(x_k)$ závislé na stavu x_k. Náhodná porucha w_k je charakterizována rozdělením pravděpodobnosti P_k, které může explicitně záviset na x_k a u_k, ne však na předchozích poruchách $w_{k-1}, \ldots, w_0$.

Dále uvažujme množinu řízení, jedná se o posloupnost funkcí

$$\pi = \{\mu_0, \ldots, \mu_{N-1}\},$$

kde μ_k přiřazuje stavu x_k přípustné řízení $u_k = \mu_k(x_k)$, to je takové, že $\mu_k(x_k) \in U_k(x_k)$, množinu přípustných řešení označme Π. Máme-li dány počáteční stav x_0 a přípustné řešení π můžeme stavy x_k a poruchy w_k považovat za náhodné veličiny s rozdělemím definovaným systémem rovnic 1.1, kde za řízení u_k dosadíme hodnotu funkce μ_k v x_k.

Pro dané ztráty v jednotlivých časech – funkce g_k, pak definujeme očekávanou ztrátu π v x_0 jako

$$J_\pi(x_0) = \mathbf{E}\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k\left(x_k, \mu_k(x_k), w_k\right) \right\}$$

kde je očekávaná hodnota počítána přes náhodné veličiny w_k a x_k. Optimální řízení π^*je právě to, které minimalizuje ztrátu

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0).$$

Optimální ztrátu označme $J^*(x_0)$.

1.2.2 Přístup dynamického programování

Dynamické programovní dle [9] je jedním ze způsobů návrhu algoritmů pro řešení jistých typu optimalizačních problémů. Konkrétně se uplatňuje v případě, že jde o diskrétní optimalizační úlohu, na řešení daného problému můžeme nahlížet jako na konečnou posloupnost rozhodnutí a platí *princip optimality.*

Princip optimality

říká, že optimální posloupnost rozhodnutí musí mít následující vlastnost: *Jestliže jsme už udělali* k *rozhodnutí, musí být všechna následující rozhodnutí optimální vzhledem k výsledkům rozhodnutí předchozích, jinak nemůžeme dostat optimální řešení* [9].

Princip optimality v teorii řízení

Nechť $\pi^* = \left\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\right\}$ je optimální řídící strategie pro základní problém a předpokládejme, že když aplikujeme řízení π^*, daný stav x_i se vyskytne v čase i s pozitivní pravděpodobností. Uvažujme podproblém, kdy ve stavu x_i a čase i chceme minimalizovat *náklady na pokračování* (v anglické literatuře označováno jako „cost-to-go“) od času i do N

$$\mathbf{E}\left\{ g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}$$

Potom úsek strategie $\left\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\right\}$ je optimální pro tento podproblém.

Intuitivně je princip optimality velmi jednoduchý. Jestliže úsek strategie $\left\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\right\}$ nebude optimální, budeme schopni dále zredukovat cenu přechodem k optimální strategii pro podproblém.

Princip optimality umožňuje optimální strategii konstruovat postupně. Nejdříve nalezneme optimální strategii pro koncový podproblém zahrnující poslední krok. Poté rozšiřujeme podproblém od konce přidáním předposledního kroku a tak dále. Takto může být vytvořena optimální strategie pro celý problém.

Algoritmus dynamického programování je tedy založen na následující myšlence: Algoritmus pracuje iterativně a řeší koncové podproblémy pro daný časový úsek, při tom

využívá řešení předchozích koncových podproblémů pro kratší časové úseky. Převzato z [2].

Formulace algoritmu dynamického programování

Podle [2], pro každý počáteční stav x_0, je optimální cena $J^*(x_0)$ základního problému rovna $J_0(x_0)$, získané z posledního kroku následujícího algoritmu, který prochází zpět časy od $N-1$ do 0:

$$J_N(x_N) = g_N(x_N)$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k) w_k} \mathbf{E}\left\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\right\} \tag{1.2}$$

$$k = 0, 1, \ldots, N-1$$

kde je očekávaná hodnota počítána podle náhodné veličiny w_k, která obecně závisí na x_k a u_k. Dále, když $u_k^* = \mu_k^*(x_k)$ minimalizuje pravou stranu rovnice (1.2) pro každé x_k a k, stretegie $\pi* = \left\{\mu_1^*, \ldots, \mu_{N-1}^*\right\}$ je optimální.

Hodnotu $J_k(x_k)$ je možno interpretovat jako optimální cenu pro $(N-k)$-tý krok problému začínajícího ve stavu x_k a čase k, a končícího v čase N. Následně označujeme $J_k(x_k)$ náklady na pokračování („cost-to-go“) ve stavu x_k a čase k, a J_k označujeme jako funkci nákladů na pokračování („cost-to-go function“) v čase k.

Ideálně bychom chtěli využít algoritmus dynamického programování k získání J_k vyjádřené v uzavřeném tvaru nebo k získání optimální strategie. Existuje mnoho případů, kdy je daná úloha řešitelná analyticky, obzvláště za zjednodušujících předpokladů. To je velmi užitečné zejména pro lepší náhled do problematiky a jako vodítko pro složitější modely. Avšak ve většině případů není analytické řešení možné, pak je třeba použít numerické řešení pomocí algoritmu dynamického programování. Tento přístup může být časově velmi náročný, zejména minimalizaci v rovnici (1.2) je třeba provést pro každou hodnotu x_k. Stavový prostor musí být diskretizován, nejedná-li se o konečnou množinu a výpočetní nároky pak narůstají proporcionálně k počtu možných hodnot x_k. Nicméně dynamické programování je pouze obecný přístup pro iterativní optimalizaci při uvažování nejistoty v systému.

1.3 Vliv neznalosti na systém

1.3.1 Úplná a neúplná stavová informace

V optimálním případě by bylo možno měřit všechny stavové veličiny systému a na jejich základě libovolným způsobem upravovat jeho dynamické vlastnosti. Ve skutečnosti ale zpravidla není možné všechny stavy změřit a musíme se rozhodovat pouze na základě informací, které máme k dispozici, pak mluvíme o *neúplné informaci o stavu systému* [5, 2]. Může to být způsobeno například nedostupností hodnot některých stavů, použité měřící přístroje mohou být nepřesné nebo náklady na získání přesné hodnoty stavu mohou být příliš omezující. Případy tohoto typu modelujeme zpravidla tak, že v každém

kroku regulátor obdrží jisté pozorování skutečné hodnoty stavu, které ovšem může být ovlivněno a narušeno stochastickou nejistotou. Teoreticky se však problém s neúplnou informací o stavu neodlišuje od úloh s úplnou stavovou informací, protože existují způsoby, jak převést (redukovat) systém s neúplnou informací na systém s úplnou. Tyto postupy obecně vedou na algoritmy využívající dynamické programování, ale jsou výpočetně mnohem náročnější, než v případě úplné informace. Dva možné postupy redukce převzaté z [2] budou následovat po formulaci problému:

Formulace problému s neúplnou informací o stavu

Nejdříve formulujme základní problém s neúplnou stavovou informací, který následně redukujeme na systém s informací úplnou. Uvažujme rozšíření základního problému 1.1, kde ale regulátor, namísto přístupu ke stavu systému, získává pouze pozorování z_k ve tvaru

$$z_0 = h_0(x_0, v_0), \quad z_k = h_k(x_k, u_{k-1}, v_k), \quad k = 1, 2, \ldots, N-1, \tag{1.3}$$

kde v_k reprezentuje náhodnou poruchu pozorování charakterizovanou rozdělením pravděpodobnosti P_{v_k}, která závisí na současném stavu a všech předchozích stavech, řízeních a poruchách. Dále také počáteční stav x_0 považujeme za náhodnou veličinu s rozdělením P_{x_0}.

Soubor informací dostupných regulátoru v čase k označme I_k informačním vektorem. Tedy

$$\begin{aligned} I_k &= (z_0, \ldots, z_k, u_0, \ldots, u_{k-1}), \quad k = 1, \ldots, N-1, \\ I_0 &= z_0. \end{aligned}$$

Uvažujme množinu přípustných řízení jako posloupnost funkcí $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, kde každá funkce μ_k přiřazuje informačnímu vektoru I_k řízení $\mu_k(I_k) \in U_k$, pro všechna I_k, kde $k = 0, \ldots, N-1$. Chceme najít přípustnou řídící strategii, to jest posloupnost π, která minimalizuje očekávanou ztrátu

$$J_\pi = \mathbf{E}\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k\left(x_k, \mu_k(I_k), w_k\right) \right\},$$

kde je očekávaná hodnota počítána přes náhodné veličiny x_0 a w_k, v_k pro $k = 0, \ldots, N-1$. Veličiny x_k a z_k se vypočítají z rovnic 1.1 respektive 1.3, přičemž v nich položíme $u_k = \mu_k(I_k)$.

Redukce na systém s úplnou stavovou informací

Tento postup je založen na myšlence definovat nový systém, jehož stav v čase k je množina všech hodnot, kterých může využít regulátor při tvorbě řízení. Jako stav nového systému tedy volíme informační vektor I_k a získáme systém

$$I_{k+1} = (I_{k,} z_{k+1}, u_k), \quad I_0 = z_0, \quad k = 0, \ldots, N-2. \tag{1.4}$$

Na tento systém povahy základního problému s úplnou informací můžeme pohlížet tak, že I_k je stav. Řízení u_k a pozorování z_k lze pak chápat jako náhodné poruchy. Dále rozdělení

pravděpodobnosti z_{k+1} závisí explicitně pouze na stavu I_k a řízení u_k. Ztrátovou funkci vyjádřenou pro nový systém je možno zapsat jako

$$\mathbf{E}\left\{g_k\left(x_k, u_k, w_k\right)\right\} = \mathbf{E}\left\{\mathbf{E}_{x_k,w_k}\left\{g_k\left(x_k, u_k, w_k\right) \mid I_k, u_k\right\}\right\}.$$

Tedy ztráta během jednoho kroku vyjádřená jako funkce nového stavu I_k a řízení u_k je

$$\tilde{g}_k(I_k,u_k) = \mathbf{E}_{x_k,w_k}\left\{g_k\left(x_k, u_k, w_k\right) \mid I_k, u_k\right\}. \tag{1.5}$$

Původní základní problém s neúplnou stavovou informací byl tedy převeden na úlohu s úplnou stavovou informací s rovnicí popisující systém 1.4 a ztrátou během jednoho kroku 1.5. Nyní je pro něj možno napsat algoritmus dynamického programování.

Postačující statistika

Při užití algoritmu dynamického programování za neúplné stavové informace je hlavní problém v jeho vyhodnocování ve stavovém prostoru, jehož dimenze neustále roste. S každým dalším měřením dimenze stavu a tedy informační vektor I_k narůstá, proto se snažíme redukovat množství dat skutečně potřebných pro účely řízení. Hledáme tedy popis známý jako *postačující statistika*, který bude mít menší dimenzi než I_k ale současně zahrne veškerý důležitý obsah I_k potřebný pro řízení. Jako postačující statistiku označme funkci S_k informačního vektoru I_k, tedy $S_k(I_k)$ takovou, že minimalizuje ztrátu v algoritmu dynamického programování přes všechna přípustná řízení. Což můžeme zapsat pro vhodnou funkci H_k jako

$$J_k(I_k) \quad = \quad \min_{u_k \in U_k} H_k(S_k(I_k), u_k).$$

Po funkci S_k samozřejmě chceme, aby byla charakterizována menší množinou čísel, než informační vektor I_k, abychom získaly výhody z jejího použití. Obecně existuje mnoho funkcí, které mohou sloužit jako postačující statistika. Triviálním příkladem může být identita $S_k(I_k) = I_k$.

Závisí-li rozdělení pravděpodobnosti poruchy pozorování v_k explicitně pouze na bezprostředně předcházejícím stavu, řízení a poruše systému, tedy na x_k, u_k, w_k a nezávisí na předchozích hodnotách $x_{k-1}, \ldots, x_0, u_{k-1}, \ldots, u_0, w_{k-1}, \ldots, w_0, v_{k-1}, \ldots, v_0$ můžeme za postačující statistiku volit podmíněné rozdělení pravděpodobnosti $P_{x_k|I_k}$, o kterém lze ukázat (viz [2]), že

$$J_k(I_k) = \min_{u_k \in U_k} H_k(P_{x_k|I_k}, u_k) = \overline{J}_k(P_{x_k|I_k}),$$

kde H_k a $\overline{J}_k$ jsou vhodné funkce. Optimální řízení pak získáme ve tvaru funkcí podmíněného rozdělení pravděpodobnosti $\mu_k(I_k) = \overline{\mu}_k(P_{x_k|I_k})$ pro $k = 0, \ldots, N-1$. Tato reprezentace může být velmi užitečná, protože nám umožňuje rozložit optimální řízení na dvě nezávislé časti:

1. *pozorovatel* (estimátor), který v čase k použije měření z_k a řízení u_{k-1} k vygenerování rozdělení pravděpodobnosti $P_{x_k|I_k}$

2. *regulátor* (akurátor), který generuje vstupy (řízení) pro systém jako funkci rozdělení pravděpodobnosti $P_{x_k|I_k}$

Tento rozklad pak umožňuje navrhovat každou z částí samostatně podle charakteru konkrétní úlohy.

1.3.2 Kalmanův filtr

Chceme řešit následující problém, viz [5]: Máme lineární systém s neúplnou stavovou informací a snažíme se odhadnout (rekonstruovat, estimovat) stav systému z měřitelných vstupních a výstupních veličin. Dále předpokládejme, že měření výstupu a popřípadě i vstupu je zatíženo chybou měření. Tyto nepřesnosti měření můžeme modelovat jako aditivní šum. Odhadování (rekonstrukci, estimaci) potom navrhujeme pomocí stochastických metod. Řešení vede na takzvaný *Kalmanův filtr*.

Následující formulace problému a popis algoritmu Kalmanova filtru je převzat z [2], kde lze také nalézt odvození příslušných rovnic: Máme dva náhodné vektory x a y, které jsou svázány sdruženým rozdělením pravděpodobnosti tak, že hodnota jednoho poskytuje informaci o hodnotě druhého. Známe hodnotu y a chceme určit (odhadnout) hodnotu x tak, aby střední kvadratická odchylka mezi x a jeho odhadem byla minimální.

Takový odhad můžeme zístat v nejjednodušším případě metodou nejmenších čtverců, ale pro tento způsob je třeba velkého počtu měření. Jako lepší způsob se ale jeví využít sekvenční struktury problému a iterativně použít Kalmanův filtr, kdy odhad v čase $k+1$ získáme na základě jednoduchých rovnic pouze z předchozího odhadu a nového měření v čase k, žádná předchozí měření nejsou explicitně zahrnuta.

V dalším textu označme $\hat{x}_{k|k-1}$ apriorní odhad stavu, tedy odhad stavu v čase k na základě informací až do času $k-1$. Analogicky $\Sigma_{k|k-1}$ označuje apriorní kovarianční matici. Aposteriorní odhad stavu označme $\hat{x}_{k|k}$, to jest odhad v čase k na základě informačí až do času k. Aposteriorní kovarianční matice je pak označena $\Sigma_{k|k}$.

System

Uvažujme lineární dynamický systém bez řízení ($u_k \equiv 0$) ve tvaru

$$x_{k+1} = A_k x_k + w_k,\ k = 0, 1, \ldots, N-1,$$

kde x_k je vektor stavu, w_k vektor náhodné poruchy a matice A_k předpokládáme známé. Dále rovnice měření je

$$z_k = C_k x_k + v_k,\ k = 0, 1, \ldots, N-1,$$

kde z_k je vektor pozorování (měřených veličin) a v_k vektor šumu. Nechť $x_0, w_0, \ldots, w_{N-1}, v_0, \ldots, v_{N-1}$ jsou vektory nezávislých náhodných veličin s daným rozdělením pravděpodobnosti, takovým, že

$$\mathrm{E}\{w_k\} = \mathrm{E}\{v_k\} = 0,\ k = 0, 1, \ldots, N-1.$$

Označme

$$S = \mathrm{E}\left\{(x_0 - \mathrm{E}\{x_0\})\,(x_0 - \mathrm{E}\{x_0\})^T\right\},\ M_k = \mathrm{E}\{w_k w_k^T\},\ N_k = \mathrm{E}\{v_k v_k^T\},$$

a nechť matice N_k pozitivně definitní pro všechny časy k.

Algoritmus Kalmanova filtru

Předpokládejme, že máme spočítaný odhad $\hat{x}_{k|k-1}$ společně s kovarianční maticí $\Sigma_{k|k-1} = \mathrm{E}\left\{\left(x_k - \hat{x}_{k|k-1}\right)\left(x_k - \hat{x}_{k|k-1}\right)^T\right\}$. V čase k získáme další měření $z_k = C_k x_k + v_k$. Nyní můžeme získat aposteriorní odhad stavu $\hat{x}_{k|k}$ v čase k jako

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + \Sigma_{k|k-1} C_k^T \left(C_k \Sigma_{k|k-1} C_k^T + N_k\right)^{-1} \left(z_k - C_k \hat{x}_{k|k-1}\right), \tag{1.6}$$

dále pak apriorní odhad stavu $\hat{x}_{k+1|k}$ v čase $k+1$, tedy $\hat{x}_{k+1|k} = A_k \hat{x}_{k|k}$. Apriorní kovarianční matici v čase $k+1$ vypočítáme z

$$\Sigma_{k+1|k} = A_k \Sigma_{k|k} A_k^T + M_k,$$

kde aposteriorní kovarianční matici $\Sigma_{k|k} = \mathrm{E}\left\{\left(x_k - \hat{x}_{k|k}\right)\left(x_k - \hat{x}_{k|k}\right)^T\right\}$ můžeme získat z rovnice

$$\Sigma_{k|k} = \Sigma_{k|k-1} - \Sigma_{k|k-1} C_k^T \left(C_k \Sigma_{k|k-1} C_k^T + N_k\right)^{-1} C_k \Sigma_{k|k-1}.$$

Přidáním počátečních podmínek $\hat{x}_{0|-1} = \mathrm{E}\{x_0\}$ a $\Sigma_{0|-1} = S$ získáme *algoritmus Kalmanova filtru*, který ve své podstatě rekurzivně generuje posloupnost lineárních odhadů založených na metodě nejmenších čtverců.

Dále je možno vyjádřit rovnici 1.6 ve tvaru

$$\hat{x}_{k|k} = A_{k-1} \hat{x}_{k-1|k-1} + \Sigma_{k|k} C_k^T N_k^{-1} \left(z_k - C_k A_{k-1} \hat{x}_{k-1|k-1}\right),$$

který při uvažování systému se vstupem

$$x_{k+1} = A_k x_k + B_k u_k + w_k,\ k = 0, 1, \ldots, N-1,$$

umožňuje vypočítat rekurzivně aposteriorní odhady stavů $\hat{x}_{k|k}$ v časech k z rovnice

$$\hat{x}_{k|k} = A_{k-1} \hat{x}_{k-1|k-1} + B_{k-1} u_{k-1} + \Sigma_{k|k} C_k^T N_k^{-1} \left(z_k - C_k A_{k-1} \hat{x}_{k-1|k-1}\right),$$

přičemž rovnice pro výpočet aposteriorní kovarianční matice $\Sigma_{k|k}$ zůstávají nezměněny.

1.4 Spojité systémy

1.4.1 Deterministické systémy se spojitým časem

I když zpravidla pracujeme s diskrétními systémy, zejména z důvodů výpočtů na počítači, teorie optimálního řízení spojitých systémů může být velmi užitečná. Poskytuje totiž důležité principy, které jsou velmi často používány při návrhu algoritmů pro duální řízení. Konkrétně se jedná o Hamilton-Jacobi-Bellmanovu rovnost a Pontryaginův princip minima.

Spojitý systém

Dynamický systém se spojitým časem uvažujeme dle [2] ve tvaru

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)),\ 0 \leq t \leq T, \\ x(0) &= x_0, \end{aligned} \tag{1.7}$$

kde $x(t)$ je stavový vektor v čase t, $\dot{x}(t)$ je vektor prvních derivací podle času v čase t, $u(t) \in U$ je řídící vektor v čase t, U je množina omezení řízení a T je časový horizont. O funkci f předpokládáme, že je spojitě diferencovatelná vzhledem k x a spojitá vzhledem k u. Rovnice 1.7 představuje soustavu n diferenciálních rovnic prvního řádu. Naším cílem je nalézení přípustné řídící trajektorie $\{u(t) \mid t \in [0,T]\}$ a odpovídající stavové trajektorie $\{x(t) \mid t \in [0,T]\}$ takové, že minimalizují ztrátovou funkci ve tvaru

$$h(x(T)) + \int_0^T g\left(x(t), u(t)\right) dt,$$

o funkcích g a h předpokládáme, že jsou spojitě diferencovatelné vzhledem k x a g je spojitá vzhledem k u.

Hamilton-Jacobi-Bellmanova rovnost

Hamilton-Jacobi-Bellmanova rovnost je parciální diferenciální rovnicí, která je splněna optimální funkcí nákladů na pokračování $J^*(t,x)$. Tato rovnice je analogií algoritmu dynamického programování ve spojitém čase. Rovnici lze psát podle [2] ve tvaru

$$\begin{aligned} 0 &= \min_{u \in U} \left[g(x,u) + \nabla_t J^*(t,x) + \nabla_x J^*(t,x)^T f(x,u) \right], \quad \forall t, x, \\ J^*(T,x) &= h(x). \end{aligned} \tag{1.8}$$

Jedná se tedy o parciální diferenciální rovnici s okrajovou podmínkou. O funkci $J^*(t,x)$ jsme předpokládali diferencovatelnost, apriorně ale její diferencovatelnost neznáme a tedy nevíme, jestli $J^*(t,x)$ řeší rovnici 1.8. Můžeme však použít následující tvrzení, jehož formulaci i důkaz lze nalézt v [2]:

Věta o dostatečnosti:

Nechť je funkce $V(t,x)$ spojitě diferencovatelná vzhledem k t a x a nechť je řešením Hamilton-Jacobi-Bellmanovy rovnosti:

$$\begin{aligned} 0 &= \min_{u \in U} \left[g(x,u) + \nabla_t V(t,x) + \nabla_x V(t,x)^T f(x,u) \right], \quad \forall t, x, \\ V(T,x) &= h(x), \quad \forall x. \end{aligned} \tag{1.9}$$

Předpokládejme dále, že $\mu^*(t,x)$ dosáhne minima v rovnosti 1.9 pro všechna t a x. Nechť $\{x^*(t) \mid t \in [0,T]\}$ označuje stavovou trajektorii získanou při dané počáteční podmínce $x^*(0) = x_0$ a řídící trajektorii $u^*(t) = \mu^*(t, x^*(t)),\ t \in [0,T]$. Pak V je rovno optimální funkci nákladů na pokračování, tedy

$$V(t,x) = J^*(t,x), \quad \forall t, x.$$

Navíc řídící trajektorie $\{u^*(t) \mid t \in [0,T]\}$ je optimální.

Pontryaginův princip minima

Pontryaginův princip minima je důležitým teorémem optimálního řízení. Poskytuje nutnou (ne však postačující) podmínku pro optimální trajektorii, je úzce spřízněn s Hamilton-Jacobi-Bellmanovou rovností a lze ho z ní podle [2] také odvodit. Princip minima je výhodné formulovat pomocí Hamiltoniánu. Označme p gradient optimální funkce nákladů na pokračování pro optimální stavovou trajektorii $p(t) = \nabla_x J^* (t, x^*(t))$ a definujme Hamiltonián jako funkci zobrazující trojice vektorů (x, u, p) do reálných čísel

$$H(x, u, p) = g(x, u) + p^T f(x, u).$$

Rovnice pro systém pak může být zapsána v kompaktním tvaru

$$\dot{x}^*(t) = \nabla_p H \left(x^*(t), u^*(t), p(t)\right).$$

Obdobně může být zapsána pro p takzvaná *adjungovaná rovnice*

$$\dot{p}(t) = -\nabla_x H \left(x^*(t), u^*(t), p(t)\right).$$

Pontryaginův princip minima je podle [2] formulován následovně:

Princip minima:

Nechť $\{u^*(t) \mid t \in [0, T]\}$ je optimální řídící trajektorie a nechť $\{x^*(t) \mid t \in [0, T]\}$ je odpovídající stavová trajektorie, to jest

$$\dot{x}^*(t) = f \left(x^*(t), u^*(t)\right), \quad x^*(0) = x_0.$$

Nechť dále $p(t)$ je řešením adjungované rovnice

$$\dot{p}(t) = -\nabla_x H \left(x^*(t), u^*(t), p(t)\right),$$

s okrajovou podmínkou $p(T) = \nabla h \left(x^*(T)\right)$. Pak pro všechna $t \in [0, T]$

$$u^*(t) = \arg\min_{u \in U} H \left(x^*(t), u, p(t)\right).$$

Navíc existuje konstanta C taková, že

$$H \left(x^*(t), u^*(t), p(t)\right) = C, \quad \forall t \in [0, T].$$

1.5 Algoritmy pro duální řízení

Metody pro nalezení optimálního řízení lze obecně rozdělit do dvou základních kategorií na *globální* a *lokální* viz [8, 7].

Globální metody, používané zejména v posilovaném učení („Reinforcement Learning“), jsou založeny na na Bellmanově principu optimality, Hamilton-Jacobi-Bellmanově rovnosti a dynamickém programování. Tyto algoritmy hledají globálně optimální zpětnovazební řízení pro všechny stavy obecného stochastického systému a proto podléhají nebezpečí

„problému dimenzionality“ nebo také rozměrnosti (z anglického “curse of dimensionality” doslovně - *kletba rozměrnosti*). Jednoduše můžeme tento problém chápat tak, že při numerickém řešení úlohy jsou počítačem procházeny všechny body diskretizovaného stavového a řídícího prostoru jejichž počet s rostoucím počtem dimenzí extrémně (exponenciálně) rychle roste. Výpočet pro mnohadimenzionální úlohy se pak stává co do paměťových nároků, ale hlavně z hlediska výpočetního času prakticky nerealizovatelným.

Lokální metody, častěji studované v teorii řízení, souvisí s Pontryaginovým principem maxima. Jejich podstatou je nalezení řízení, které je pouze lokálně optimální v okolí nějaké „extremalní“ trajektorie. Většinou je užito deterministických prostředků jako řešení soustavy obyčejných diferenciálních rovnic (například střelbou nebo relaxací). Tento přístup ale vede na přímovazební řízení a nezle užít pro stochastické úlohy, vyhýbá se ale problému dimenzionality, což umožňuje řešit i komplexnější problémy.

V poslední době je snaha vyvíjet nové algoritmy, které kombinují výhody obou výše zmíněných přístupů. Příkladem může být *diferenciální dynamické programování* (DDP). Tento algoritmus zůstává lokální metodou ve smyslu, že uchovává pouzve jedinou trajektorii, která je lokálně vylepšována. Vylepšení však není založeno na řešení soustavy obyčejných diferenciálních rovnic, ale na dynamickém programování aplikovaném na okolí - “trubici” podél současné trajektorie. Jedná se o algoritmus s konvergencí druhého řádu. Ještě efektivnější je metoda podobná DDP, *iterativní LQG* (iLQG). Tento algoritmus je založen na linearizaci nelineární úlohy v každém bodě reprezentativní trajektorie a následném řešení modifikované Riccatiho rovnice. Výhodou DDP i iLQG je, že jejich výsledkem je zpětnovazební řízení. Obě metody jsou ale stále deterministické a nedokáží se vypořádat s nekvadratickými ztrátovými funkcemi a požadavky na omezené řízení.

S výše zmíněnými problémy se snaží vypořádat modifikovaná iLQG, která bude použita pro srovnání s ústřední metodou této práce iLDP. Dále pak do kategorie smíšených metod spadá právě i metoda iLDP, která bude podrobně popsána v následující kapitole.

2 Algoritmy pro návrh řízení

2.1 Výběr algoritmů pro srovnání

2.1.1 Princip separace

Princip separace nebo také *separační teorém pro lineární systémy s kvadratickou ztrátovou funkcí* zaujímá důležité místo v moderní teorii řízení. Intuitivně je velmi jednoduchý. Podle [2] je formulován následovně:

Optimální řízení pro lineární systém může být rozděleno do dvou částí:

1. *pozorovatel* (estimátor), který využívá měřená data k odhadu stavu systému,
2. *regulátor* (akurátor), který generuje ze stavu, respektive jeho odhadu, řízení pro systém.

Navíc část optimálního řízení označená jako *pozorovatel* je optimálním řešením problému určování (estimace) stavu nezávisle na uvažování řízení a část označená jako *regulátor* je optimální řešení řídícího problému za předpokladu úplné stavové informace. Každá část tedy může být navrhována nezávisle na sobě jako optimální řešení příslušných problémů estimace a regulace.

2.1.2 LQG

Řízení LQG (z anglického „Linear-Quadratic-Gaussian“) je primárně navrženo pro řízení lineárních systémů s kvadratickou ztrátovou funkci a Gaussovským šumem. Existují však různé modifikace i pro nelineární systémy. Algoritmus LQG je založen právě na *principu separace* kdy pozorovatel a regulátor jsou navrhovány zvlášť. Optimálním pozorovatelem je zde Kalmanův filtr a lze jej užít například ve tvaru, jak byl uveden v části 1.3.2. Optimálním regulátorem pak řízení označované jako LQ regulátor, které může být formulováno podle [2] následovně:

LQ regulátor pro lineární systém

$$x_{k+1} = A_k x_k + B_k u_k + w_k, \quad k = 0, 1, \ldots, N-1,$$

s kvadratickou ztrátovou funkcí

$$\mathbf{E}\left\{ x_N^T Q_N x_N + \sum_{k=0}^{N-1} \left(x_k^T Q_k x_k + u_k^T R_k u_k \right) \right\},$$

při uvažování neúplné stavové informace je optimálním řízením v každém čase rovno

$$\mu_k^*(I_k) = L_k \mathrm{E}\left\{x_k \mid I_k\right\},$$

kde matice L_k je dána rovností

$$L_k = -\left(R_k + B_k^T K_{k+1} B_k\right)^{-1} B_k^T K_{k+1} A_k,$$

přičemž matice K_k získáme rekurzivně z Riccatiho rovnice

$$\begin{aligned} K_N &= Q_N, \\ K_k &= A_k^T \left(K_{k+1} - K_{k+1} B_k \left(R_k + B_k^T K_{k+1} B_k\right)^{-1} B_k^T K_{k+1}\right) A_k + Q_k. \end{aligned}$$

2.1.3 Zobecněné iterativní LQG řízení

V článku [8] je popsán algoritmus *zobecněného iterativního LQG* řízení (iLQG) pro účely nalezení lokálního zpětnovazebního řízení nelineárních stochastických systémů s kvadratickou ztrátou, ale navíc lze požadovat i omezené vstupy. Obecně zahrnutí požadavku na omezené vstupy do ztrátové funkce způsobí porušení její kvadratičnosti, zmiňovaný algoritmus však řeší problém jinak, konkrétně následnou korekcí rovnic pro výpočet řízení. Dále s nelinearitou se algoritmus vypořádává tak, že systém v každém časovém kroku linearizuje vzhledem k reprezentativní trajektorii. Linearizovaný systém je pak řešen klasickým přístupem LQG, avšak v jeho průběhu je do výpočtů ještě zasahováno. Jsou prováděny úpravy dílčích výsledků a opravy chyb z důvodu práce s linearozovaným nelineárním systémem pro zajištění konvergence algoritmu. Samotný algoritmus je odvozen a detailně popsám v [8] odkud je převzat následující zestručněný popis:

iLQG lokální řízení pro obecně nelineární stochastický systém

$$\begin{aligned} x_{k+1} &= x_k + f(x_k, u_k) \cdot \Delta k + F(x_k, u_k) \cdot e_k,\ k = 0, 1, \ldots, N-1, \\ x_{(k=0)} &= x_0, \end{aligned} \tag{2.1}$$

se ztrátovou funkcí

$$\mathrm{E}\left\{h(x_N) + \sum_{k=0}^{N-1} l(k, x_k, u_k)\right\}$$

je lokálně optimální řízení, které konstruujeme iterativně. Každá iterace začíná s posloupností přímovazebních řízení $\overline{u}_k$ a odpovídající bezšumovou trajektorií $\overline{x}_k$ získanou aplikací $\overline{u}_k$ na systém 2.1 s nulovým šumem. To je možno provést například pomocí Eulerovy integrace. Pak linearizujeme systém a kvadratizujeme ztrátu podél trajektorií $\overline{x}_k$ a $\overline{u}_k$. Následně získaný lineární systém s kvadratickou ztrátou vyjádříme v odchylkách stavových a řídících veličin od bezšumové trajektorie $\delta x_k = x_k - \overline{x}_k$ a $\delta u_k = u_k - \overline{u}_k$.

Veličiny charakterizující modifikovaný problém získané v každém čase k z $(\overline{x}_k, \overline{u}_k)$ jsou

$$\begin{aligned}
A_k &= I + \frac{\partial f}{\partial x} \cdot \Delta k, \quad B_k = \frac{\partial f}{\partial u} \cdot \Delta k, \\
\mathbf{c}_{i,k} &= F^{[i]} \cdot \sqrt{\Delta k}, \quad C_{i,k} = \frac{\partial F^{[i]}}{\partial u} \cdot \sqrt{\Delta k}, \\
q_k &= l \cdot \Delta k, \quad \mathbf{q}_k = \frac{\partial l}{\partial x} \cdot \Delta k, \\
Q_k &= \frac{\partial^2 l}{\partial x \partial x} \cdot \Delta k, \quad P_k = \frac{\partial^2 l}{\partial u \partial x} \cdot \Delta k, \\
\mathbf{r}_k &= \frac{\partial l}{\partial u} \cdot \Delta k, \quad R_k = \frac{\partial^2 l}{\partial u \partial u},
\end{aligned}$$

kde $F^{[i]}$ označuje i-tý sloupec matice F a veličiny „q“ se počítají v čase $k = N$ z funkce h namísto l.

Dále zaveďme označení

$$\begin{aligned}
\mathbf{g}_k &= \mathbf{r}_k + B_k^T \mathbf{s}_{k+1} + \sum_i C_{i,k}^T S_{k+1} \mathbf{c}_{i,k}, \\
G_k &= P_k + B_k^T S_{k+1} A_k, \\
H_k &= R_k + B_k^T S_{k+1} B_k + \sum_i C_{i,k}^T S_{k+1} C_{i,k}.
\end{aligned}$$

Zpětnovazební řízení pak hledáme ve tvaru $\delta u_k(\delta x) = \mathbf{l}_k + L_k \delta x$, kde $\mathbf{l}_k = -H_k^{-1} \mathbf{g}_k$ a $L_k = -H_k^{-1} G_k$. Přičemž parametry S_k a $\mathbf{s}_k$ jsou počítány rekurzivně z rovnic

$$\begin{aligned}
S_k &= Q_k + A_k^T S_{k+1} A_k + L_k^T H_k L_k + L_k^T G_k + G_k^T L_k, \\
\mathbf{s}_k &= \mathbf{q}_k + A_k^T \mathbf{s}_{k+1} + L_k^T H_k \mathbf{l}_k + L_k^T \mathbf{g}_k + G_k^T \mathbf{l}_k.
\end{aligned} \tag{2.2}$$

V důsledku linearizace obecně nelineárního systému mohou vyjít některá vlastní čísla matice H nulová nebo záporná. Řešení tohoto problému spolu s ošetřením požadavku na omezené vstupy u je detailně popsáno v [8]. Pokud však nepotřebujeme vyhovět požadavku na nekladná vlastní čísla matice H a omezené vstupy, lze rovnice 2.2 zjednodušit a pokud dále šum nezávisí na řízení (tedy $C_{i,k} = 0$) rovnice 2.2 se redukuje na Riccatiho rovnici klasického LQ regulátoru.

2.2 Algoritmus iterativního lokálního dynamického programování

Algoritmus iLDP byl vytvořen pro účely nalezení stochastického optimálního řízení v mnohadimenzionálních stavových a řídících prostorech. Tento případ je častý zejména při řízení biologických pohybů. Metoda je popsána autory v článku [7] a z tohoto zdroje je také převzata.

Základní popis algoritmu, tak jak ho autoři podali, je však pouze šablonou a mnoho detailů a dílčích částí je ponecháno na vyřešení při konkrétní realizaci. To se týká hlavně

použitých aproximací pro jednotlivé funkce, zejména aproximace Bellmanovy funkce a aproximace hledaného regulátoru. Dále, protože algoritmus využívá hledání minima, není v základním popisu algoritmu vyřešen konkrétní typ minimalizace. Použitý minimalizační algoritmus se samozřejmě liší podle konkrétního problému, zejména jedná-li se o minimalizaci omezenou nebo neomezenou. Ještě je třeba zmínil, že pro algoritmus je nutno zvolit parametr „velikosti“ okolí, protože se jedná o lokální metodu.

2.2.1 Formulace problému

Naším úkolem je nalézt řízení $u = \pi(t, x)$, které minimalizuje očekávanou ztrátu

$$J(\pi) = E_\omega \left(h(x) + \int_0^T l(x, \pi(t, x))dt \right),$$

obecně pro spojitý systém:

$$\begin{aligned} d\mathbf{x} &= f(x, u)dt + F(x, u)d\omega, \\ x(0) &= x_0, \\ t &\in [0, T], \end{aligned} \tag{2.3}$$

v diskrétním tvaru:

$$\begin{aligned} x_{k+1} - x_k &= f(x, u) \cdot \Delta k + F(x, u)e_k, \\ x_{(k=0)} &= x_0, \\ k &\in \{0, 1, \ldots, N\}, \\ \Delta k &= \frac{T}{N}, \end{aligned} \tag{2.4}$$

kde hledáme řízení $u = \pi(k, x)$, které minimalizuje očekávanou ztrátu

$$J(\pi) = E \left(h(x) + \sum_{k=0}^{N-1} l_k(x, \pi(k, x)) \cdot \Delta k \right).$$

2.2.2 Osnova algoritmu

Algoritmus pracuje iteračně, každá iterace začne s řízením π a vytvoří zlepšení π'. Přičemž prvotní řešení π_0 musíme algoritmu dodat jako apriorní informaci. Pro zajištění globální konvergence je možno nové řešení hledat jako konvexní kombinaci starého a algoritmem nalezeného řešení

$$\pi^{nové} = \alpha\pi' + (1 - \alpha)\pi;\ 0 < \alpha \leq 1;\ J(\pi^{nové}) < J(\pi).$$

V každé iteraci proběhne nejprve přípravná fáze, kdy z řízení $\pi(k,x)$ generuje průměrnou trajektorii $\bar{x}(k)$ řešením rovnice 2.3 respektive 2.4. Následně se počítá aproximace $\tilde{V}(k,x)$ Bellmanovy funkce $V(k,x)$ v čase odzadu, tj. od N k 1. Současně počítáme i aproximaci řízení $\pi'(k,x)\dots\pi'(N-1,x)$. Tedy pro každý čas k takový, že $k = N-1\dots 1$ jdeme zpět, přičemž pokládáme v koncovém čase N hodnotu aproximace Bellmanovy funkce $\tilde{V}(N,x) = h(x)$ a provádíme následující čtyři kroky:

1. Generujeme množinu stavů $\left\{x^{(n)}\right\}_{n=1\dots M}$ shromážděných kolem průměrného stavu $\bar{x}(k)$.

2. Pro každé $x^{(n)}$ vypočítáme optimální řízení $u^{(n)}$ minimalizací Hamiltoniánu

$$H(k,x,u) = l(x,u) + f(x,u)^T\tilde{V}_x(k+1,x) + \frac{1}{2}\mathbf{tr}\left(\sum(x,u)\tilde{V}_{xx}(k+1,x)\right)$$

   s inicializačním bodem $\pi(k,x^{(n)})$. Kde $\Sigma(x,u) = F(x,u)F(x,u)^T$. Tedy optimální řízení v čase k pro stav n hledáme jako

$$u^{(n)} = \arg\min_u H(k,x,u).$$

3. Pro každé $x(k)$ aproximovat $v^{(n)} = V(k,x^{(n)})$ použitím Hamolton-Jacobi-Bellmanovi rovnosti

$$V(k,x^{(n)}) \approx \Delta k \cdot H(k,x^{(n)},u^{(n)}) + \tilde{V}(k+1,x^{(n)}).$$

4. Vypočítat novou aproximaci funkce $\tilde{V}(k,x)$ z množiny bodů $\left\{x^{(n)},v^{(n)}\right\}$ a aproximaci řízení $\pi'(k,x^{(n)})$ definované pro všechna x jako z množiny bodů $\left\{x^{(n)},u^{(n)}\right\}$.

2.2.3 Detaily implementace

Uvedený obecný popis algoritmu může být aplikován mnoha způsoby v závislosti na konkrétních volbách v každém z kroků algoritmu. Jedná se zejména o následující případy:

Volba okolí v *bodě 1.*

Zde se projevuje lokálnost metody. Množina stavů $\left\{x^{(n)}\right\}$ je vybrána z okolí průměrného stavu $\bar{x}(k)$. Toto okolí a způsob výběru množiny je třeba konkrétně specifikovat. Pro účely implementace algoritmu bylo okolí specifikováno parametrem ρ^2. Množina stavů $\left\{x^{(n)}\right\}$ pak byla generována náhodně jako náhodná veličina s normálním rozdělením se střední hodnotou rovnou průměrnému stavu $\bar{x}(k)$ a rozptylem specifikovaným parametrem ρ^2.

Počet vzorků M je nutno zvolit při implementaci algoritmu. Obecně je nejlepší volit maximální možné číslo, ovšem s rostoucím počtem vzorků rostou i paměťové nároky a výpočetní čas algoritmu.

Minimalizace v *bodě 2.*

Pro minimalizaci lze použít například minimalizační funkce programu *Matlab* z balíku *Optimization Toolbox*, konkrétně se jedná o funkce `fminunc` respektive `fmincon` pro neomezenou respektive omezenou minimalizaci. V případě, kdy je možno spočítat minimalizaci analyticky, jedná se samozřejmě o nejlepší způsob.

Použití aproximací v *bodě 4.*

Aproximace je třeba zvolit ještě před zahájením výpočtu algoritmu, avšak právě v *bodě 4.* je třeba je vypočítat z množiny párů hodnot. Konkrétně se jedná o aproximaci Bellmanovy funkce $\tilde{V}$ a aproximaci řízení π. Volíme aproximace v jednodušším tvaru z důvodu výpočetní náročnosti, protože jsou počítány opakovaně. Dále je nutno vygenerovat dostatečný počet M vzorků $\left\{x^{(n)}\right\}$ v *bodě 1.* abychom měli dostatek dat pro určení koeficientů aproximací. I když nám volnost ve volbě aproximací přináší relativně velkou svobodu při návrhu algoritmu iLDP, jedná se současně i o největší slabinu, protože autoři explicitně neuvadějí jaké aproximace volit. Následně, při implementaci algoritmu pro systém s větším počtem dimenzí, může být Bellmanova funkce velmi složitá a právě její vhodnou aproximaci se nemusí podařit nalézt.

2.2.4 Konkrétní použité aproximace

Výpočet hodnot a aproximace $\tilde{V}$ ($\tilde{V}_x, \tilde{V}_{xx}$) je opakovaný. Je tedy třeba vysoké optimalizace, proto je použita lineární aproximace ve tvaru lineární kombinace dvakrát diferencovatelných základních funkcí $\phi(x) \in \mathbf{R}^P$ kde $P < N$. Jako základní funkce mohou být voleny například funkce 1, x_i, $x_i x_j$, $x_i^2 x_j$. Aproximace je volena jako časově proměnná, kdy $\tilde{V}(k, x) = \phi(x - \bar{x}(k))^T \mathbf{w}(k)$, kde $\mathbf{w}(k)$ je parametrický vektor závislý na čase k.

Označme $\tilde{V}_x = \phi_x^T \mathbf{w}$ a $\tilde{V}_{xx} = \phi_{xx}^T \mathbf{w}$ první a druhou derivaci aproximace Bellmanovy funkce podle proměnné x respektive *vektor* a *matici* parciálních derivací podle složek vektoru x. Parametry aproximace pro jednotlivé časy $\mathbf{w}$ se určí lineární regresí. Pro $\mathbf{v} = \left[v^{(1)} \dots v^{(M)}\right]$ vektor cílových hodnot a matici $\mathbf{\Phi} = \left[\phi(x^{(1)} - \bar{x}(k)) \dots \phi(x^{(M)} - \bar{x}(k))\right]$ je minimální kvadratická odchylka $\| \mathbf{v} - \mathbf{\Phi}^T \mathbf{w} \|^2$ pro volbu parametru $\mathbf{w} = \left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1} \mathbf{\Phi}\mathbf{v}$.

Protože je průměrná trajektorie $\bar{x}(k)$ konstantní v iteraci algoritmu, je z důvodu urychlení výpočtu aproximace vycentrována v tomto bodě. Množina $\left\{x^{(n)}\right\}$ je časově proměnná, abychom nemuseli v každém kroku počítat $\left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1} \mathbf{\Phi}$, položíme $x^{(n)} = \bar{x}(k) + \varepsilon^{(n)}$, kde $\left\{\varepsilon^{(n)}\right\}$ je stejná pro všechny časy k. Množina$\left\{x^{(n)}\right\}$ se pak jakoby pohybuje podél trajektorie $\bar{x}(k)$. Tedy $\tilde{V}(k, x^{(n)}) = \phi(\varepsilon^{(n)})^T \mathbf{w}(k)$ a Φ je konstantní v nejen čase, ale i v iteracích algoritmu a matici $\left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1} \mathbf{\Phi}$ je možno předpočítat (což by nešlo při závislosti na stavech).

2.2.5 Předběžný odhad vlatností algoritmu

V tomto odstavci jsou uvedeny předběžné odhady vlastností algoritmu, jeho výhody a nevýhody. Tyto odhady byly učiněny na základě popisu algoritmu, dále podle samotného hodnocení autorů v článku [7] a následně i v průběhu implementace metody. Později budou konfrontovány s pozorováními získaných výsledků a závěry simulací, aby bylo zřejmé, která očekávání byla naplněna, a která nikoliv. Tento postup může být velmi užitečný zejména z důvodu posouzení, které charakteristické vlastnosti algoritmu iLDP jsou patrny pouze při letmém prostudovaní a naopak, pro které je nutno algoritmus implementovat a otestovat.

Výhody

- duální metoda (lépe se vypořádá s neznalostí oproti neduálním metodám, například LQG)
- lepší zvládnutí šumu
- rychlejší dosažení požadované hodnoty
- možnost aplikace na mnoharozměrové stavové a řídící prostory
- univerzálnost (vychází z obecných principů)
- svoboda ve výběru konkrétních aproximací a minimalizací

Nevýhody

- vyšší časová náročnost
- numerické výpočty (minimalizace)
- nepřesnost v důsledku aproximace klíčových funkcí v algoritmu
- implementační složitost
- problémy s volbou aproximací
- lokálnost metody a tedy i nalezeného řešení
- volba okolí (lokální metoda)

3 Systémy pro testování

3.1 Jednoduchý systém

3.1.1 Popis problému

Tato úloha byla převzata z článku [6]. Sami autoři [6] pak přejali tento problém z [1].

Jedná se o integrátor s neznámým ziskem, tedy lineární časově invariantní systém s jedním vstupem a jedním výstupem

$$\begin{aligned} y_{k+1} &= y_k + bu_k + \sigma e_k, \\ b &\sim N(\hat{b}, P), \\ e_k &\sim N(0,1), \\ \mathrm{cov}(e_k, b_k) &= 0,\ \forall k. \end{aligned} \tag{3.1}$$

kde y_k je výstup nebo také stav procesu v čase k, u_k je řízení v čase k. Varianci šumu σ^2 předpokládáme známou, stejně jako počáteční hodnoty systému y_0, $\hat{b}_0$ a P_0. Úkolem je nalézt zpětnovazební řízení

$$u_k^* = u_k^*(y_k, y_{k-1}, \ldots, y_0, u_{k-1}, u_{k-2}, \ldots, u_0),\ 0 \leq k \leq N-1$$

minimalizující očekávanou ztrátu

$$\begin{aligned} J_0 &= \left\{ \sum_{k=0}^{N-1} g_k \right\}, \\ g_k &= (y_{k+1} - r_{k+1})^2, \end{aligned} \tag{3.2}$$

pro daný časový horizont N a referenční signál, tj. požadovanou hodnotu výstupu, ve formě posloupnosti $\{r_k\}_{k=1}^N$. Diskrétní časový krok Δk pokládáme roven jedné časové jednotce, tedy $\Delta k = 1$.

3.1.2 Úpravy rovnic

Při řešení tohoto problému je výhodné podle [1] nahlížet na systému jako úlohu s postačující statistikou $H_k = [y_k, \hat{b}_k, P_k]$. Kde y_k reprezentuje stav původní y_k, dále $\hat{b}_k$ je střední hodnota neznámého parametru b a P_k jeho variance.

Pak první rovnici v 3.1 doplníme rovnicemi, ze kterých mohou být rekurzivně napočítány parametry $\hat{b}_k$ a P_k

$$\begin{aligned} \hat{b}_{k+1} &= \hat{b}_k + K_k(y_{k+1} - y_k - \hat{b}_k u_k), \\ P_{k+1} &= (1 - K_k u_k)P_k, \\ K_k &= \frac{u_k P_k}{u_k^2 P_k + \sigma^2}. \end{aligned} \tag{3.3}$$

Ztráta v čase k je

$$J_k = \min_{u_k} \mathrm{E}_{e_k,b} \left\{ g_k + J_{k+1} \mid y_k, y_{k-1}, \ldots, u_{k-1}, u_{k-2}, \ldots \right\},$$

kde se stredni hodnota pocita pres e_k a b. Systém 3.1 je lineární, Gaussovský a máme k dispozici sdruženou hustotu rozdělení pravděpodobnosti $f(b_k) = N(\hat{b}_k, P_k)$ jejíž parametry se vyvíjejí rekurzivně podle 3.3. Je tedy možno upravit ztrátovou funkci J_k dosadíme-li za $g_k = (y_{k+1} - r_{k+1})^2$ z 3.2 a následně z 3.1 za $y_{k+1} = y_k + bu_k + \sigma e_k$:

$$\begin{aligned} J_k &= \min_{u_k} \mathrm{E}_{e_k,b} \left\{ (y_k + bu_k + \sigma e_k - r_{k+1})^2 + J_{k+1} \mid y_k, y_{k-1}, \ldots, u_{k-1}, u_{k-2}, \ldots \right\} \\ &= \min_{u_k} \mathrm{E}_{e_k} \left\{ (y_k + \hat{b}u_k + \sigma e_k - r_{k+1})^2 + P_k u_k^2 \mid y_k, y_{k-1}, \ldots, u_{k-1}, u_{k-2}, \ldots \right\} + \\ &\quad + \min_{u_k} \mathrm{E}_{e_k,b} \left\{ J_{k+1} \mid y_k, y_{k-1}, \ldots, u_{k-1}, u_{k-2}, \ldots \right\} \end{aligned}$$

A ztráta v čase k je pak vyjádřena ve tvaru

$$g_k = (y_k - r_{k+1})^2 + P_k u_k^2.$$

Následně lze zadání úlohy formulovat jako:

$$\text{Rovnice systému :} \quad \begin{bmatrix} y_{k+1} \\ \hat{b}_{k+1} \\ P_{k+1} \end{bmatrix} = \begin{bmatrix} y_k + \hat{b}_k u_k \\ \hat{b}_k + \frac{u_k P_k}{u_k^2 P_k + \sigma^2}(y_{k+1} - y_k - \hat{b}_k u_k) \\ (1 - \frac{u_k P_k}{u_k^2 P_k + \sigma^2} u_k) P_k \end{bmatrix} + \begin{bmatrix} \sigma e_k \\ 0 \\ 0 \end{bmatrix}$$

$$\text{Ztráta v čase } k\text{:} \qquad g_k = (y_k - r_{k+1})^2 + P_k u_k^2 \tag{3.4}$$

3.1.3 Aplikace metody CE

Princip metody označené jako CE (z anglického „Certainty Equivalence“) je velmi jednoduchý. Neznámé parametry v systému nahradíme jejich očekávanými hodnotami a dále všechny výpočty provádíme, jako kdyby byly parametry známé. Takto získáné řízení samozřejmě není duální a pokud se skutečná hodnota neznámého parametru výrazněji odchyluje od očekávané hodnoty, se kterou počítáme, dopouštíme se značné chyby. Zmiňovaná metoda je použita jako první přiblížení a hlavně pro srovnání s dalšími algoritmy.

Triviální CE regulátor

Při návrhu prvního, nejjednoduššího regulátoru uvažujeme pouze rovnici 3.1 a nahradíme v ní parametr b jeho očekávanou hodnotou $\hat{b}$, což vede na

$$y_{k+1} = y_k + \hat{b}u_k + \sigma e_k.$$

Se ztrátovou funkcí nebudeme explicitně počítat. Místo toho předpokládáme, že ztráta bude minimální, dosáhlneme-li požadované hodnoty r_{k+1} v jednom kroku. Položíme tedy $y_{k+1} = r_{k+1}$, šum neuvažujeme (respektive jej nahradíme jeho střední hodnotou, což je nula) a z rovnice vyjádříme řízení u_k v čase k jako

$$u_k = \frac{r_{k+1} - y_k}{\hat{b}}.$$

Zde je samozřejmě nutné předpokládat, že očekávaná hodnota $\hat{b}$ není rovna nule. Tento předpoklad může být omezující, protože z pohledu původní rovnice s neznámým parametrem b nastane problém pouze, když samotný parametr b nabývá hodnoty nula. To pak zřejmě řízení nemá na systém žádný vliv. Chceme-li tento přístup použít pro libovolné $\hat{b}$ (tedy i pro $\hat{b} = 0$), je možno například volit jmenovatel zlomku ve výrazu pro řízení místo $\hat{b}$ jako $\hat{b} + \varepsilon$ s vhodným parametrem ε, následně pak

$$u_k = \frac{r_{k+1} - y_k}{\hat{b} + \varepsilon}, \quad \hat{b} + \varepsilon \neq 0.$$

3.1.4 Algoritmus LQG

Algoritmus LQG („Linear-Quadratic-Gaussian“) je vhodný k nalezení řízení pro lineární systémy s kvadratickou ztrátovou funkci a gaussovským šumem. To je sice případ zde uvažovaného *jednoduchého systému*, ale algoritmus LQG není duální. Nedokáže si tedy poradit s neznámým parametrem b a je nutné použít nějaké aproximace. Opět tedy využijeme principu CE a nahradíme parametr b jeho očekávanou hodnotou $\hat{b}$. LQG algoritmus využívá Kalmanova filtru a dokáže tedy lépe zvládat šumy a nepřesnosti měření.

LQG regulátor

Jak již bylo zmíněno v předchozím textu, řízení LQG je založeno na principu separace, tedy estimátor a regulátor jsou navrhovány zvlášť. Máme-li k dispozici matice, popisující systém, stačí pro nalezení řízení pouze dosadit do rovnic v částech 1.3.2 a 2.1.2. Tento postup můžeme aplikovat na matice získané z rovnice 3.1, pak získáme jednoduché řízení, které ale předpokládá parametr b známý a jedná se tedy o princip CE. Matice systému budou v tomto případě

$$\begin{aligned} A &= 1, \quad B = \hat{b}, \\ C &= 1, \quad N = \sigma. \end{aligned}$$

A úpravou ztrátové funkce

$$\begin{aligned}\mathbf{E}\left\{\sum_{k=0}^{N-1} g_k\right\} &= \mathbf{E}\left\{\sum_{k=0}^{N-1} (y_{k+1} - r_{k+1})^2\right\} \\ &= \mathbf{E}\left\{\sum_{k=0}^{N-1} \psi_{k+1}^2\right\} = \mathrm{E}\left\{\sum_{k=0}^{N-1} \left(\psi_{k+1}^T Q_k \psi_{k+1}\right)\right\},\end{aligned}$$

kde ψ_k reprezentuje rozdíl $y_k - r_k$, získáme matice Q a R ve tvaru

$$Q = 1, \quad R = 0.$$

Nebo se můžeme pokusit o aplikaci na systém 3.4, který vznikl úpravou systému 3.1 a odhaduje očekávanou hodnotu a varianci neznámého parametru b, ale není lineární. Je tedy třeba systém 3.4 linearizovat, nejlépe v každém časovém kroku k. Potřebujeme tedy nějakou reprezentativní trajektorii a systém následně linearizujeme rozvojem do prvního řádu do Taylorovy řady se středem v této trajektorii a tento postup opakujeme pro každý čas k. Následně získáme matice linearizovaného systému

$$A_k = \frac{\partial}{\partial(y_k,\hat{b}_k,P_k)} \begin{bmatrix} y_{k+1} \\ \hat{b}_{k+1} \\ P_{k+1} \end{bmatrix} = \begin{pmatrix} 1 & u_k & 0 \\ -\frac{u_k P_k}{u_k^2 P_k + \sigma^2} & 1 - \frac{u_k^2 P_k}{u_k^2 P_k + \sigma^2} & \frac{u_k \sigma^2 (y_{k+1} - y_k - \hat{b} u_k)}{\left(u_k^2 P_k + \sigma^2\right)^2} \\ 0 & 0 & 1 - \frac{u_k^2 P_k \left(u_k^2 P_k + 2\sigma^2\right)}{\left(u_k^2 P_k + \sigma^2\right)^2} \end{pmatrix},$$

$$B_k = \frac{\partial}{\partial u_k} \begin{bmatrix} y_{k+1} \\ \hat{b}_{k+1} \\ P_{k+1} \end{bmatrix} = \begin{pmatrix} \hat{b} \\ \frac{\left(P_k \sigma^2 - u_k^2 P_k^2\right)\left(y_{k+1} - y_k + 2\hat{b} u_k\right)}{\left(u_k^2 P_k + \sigma^2\right)^2} \\ -\frac{2 u_k P_k^2 \sigma^2}{\left(u_k^2 P_k + \sigma^2\right)^2} \end{pmatrix}.$$

Matice pro výpočet Kalmanova filtru jsou v čase konstantní a rovny

$$C_k = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \quad N_k = \sigma.$$

Pro ztrátovou funkci upravíme ztrátu systému 3.4 následovně

$$\begin{aligned}\mathbf{E}\left\{\sum_{k=0}^{N-1} g_k\right\} &= \mathbf{E}\left\{\sum_{k=0}^{N-1} \left((y_k - r_{k+1})^2 + P_k u_k^2\right)\right\} \\ &= \mathbf{E}\left\{\sum_{k=0}^{N-1} \left(\psi_k^2 + P_k u_k^2\right)\right\} = \mathrm{E}\left\{\sum_{k=0}^{N-1} \left(\psi_k^T Q_k \psi_k + u_k^T R_k u_k\right)\right\}\end{aligned}$$

kde ψ_k reprezentuje rozdíl $y_k - r_{k+1}$. Pak matice pro kvadratickou ztrátovou funkci Q a R jsou

$$\begin{aligned} Q_N &= \theta \\ Q_k &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ R_k &= P_k \end{aligned}$$

3.1.5 iLQG

Metoda iLQG je v podstatě rozšířením základního algoritmu pro nalezení LQ řízení a v triviálním případě se na tento algoritmus i redukuje. Proto většinu z veličin charakterizujících systém, potřebných pro výpočet iLQG řízení, můžeme převzít z předchozí části o aplikaci LQG regulátoru. Postup jejich výpočtu je totiž prakticky totožný.

iLQG řízení

Veličiny budou uvedeny pouze pro případ jednoduchého systému s postačující statistikou, odhadující parametr b. Přičemž obecný tvar parametrů vychází z systému definovaného v 2.1.

$$
\begin{aligned}
A_k &= I + \frac{\partial f}{\partial x}\cdot\Delta k = \begin{pmatrix} 1 & u_k & 0 \\ -\frac{u_k P_k}{u_k^2 P_k+\sigma^2} & 1-\frac{u_k^2 P_k}{u_k^2 P_k+\sigma^2} & \frac{u_k\sigma^2\left(y_{k+1}-y_k-\hat{b}u_k\right)}{\left(u_k^2 P_k+\sigma^2\right)^2} \\ 0 & 0 & 1-\frac{u_k^2 P_k\left(u_k^2 P_k+2\sigma^2\right)}{\left(u_k^2 P_k+\sigma^2\right)^2} \end{pmatrix}, \\
B_k &= \frac{\partial f}{\partial u}\cdot\Delta k = \begin{pmatrix} \hat{b} \\ \frac{\left(P_k\sigma^2-u_k^2P_k^2\right)\left(y_{k+1}-y_k+2\hat{b}u_k\right)}{\left(u_k^2P_k+\sigma^2\right)^2} \\ -\frac{2u_kP_k^2\sigma^2}{\left(u_k^2P_k+\sigma^2\right)^2} \end{pmatrix}, \\
\mathbf{c}_{i,k} &= F^{[i]}\cdot\sqrt{\Delta k} = \begin{pmatrix} \sigma \\ 0 \\ 0 \end{pmatrix}, \\
C_{i,k} &= \frac{\partial F^{[i]}}{\partial u}\cdot\sqrt{\Delta k} = 0, \\
q_k &= l\cdot\Delta k = (y_k - r_{k+1})^2 + P_k u_k^2, \\
\mathbf{q}_k &= \frac{\partial l}{\partial x}\cdot\Delta k = \begin{pmatrix} 2(y_k-r_{k+1}) \\ 0 \\ u_k^2 \end{pmatrix}, \\
Q_k &= \frac{\partial^2 l}{\partial x\partial x}\cdot\Delta k = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \\
P_k &= \frac{\partial^2 l}{\partial u\partial x}\cdot\Delta k = \begin{pmatrix} 0 \\ 0 \\ 2u_k \end{pmatrix}, \\
\mathbf{r}_k &= \frac{\partial l}{\partial u}\cdot\Delta k = 2P_k u_k, \\
R_k &= \frac{\partial^2 l}{\partial u\partial u} = 2P_k.
\end{aligned}
$$

3.1.6 iLDP

Algoritmus implementujeme podle základní osnovy uvedené v 2.2.2 přičemž detaily implementace jsou voleny následovně:

Volba okolí

Množina stavů $\left\{x^{(n)}\right\}$ je volena jako náhodná veličina s normálním rozdělením se střední hodnotou rovnou průměrnému stavu $\bar{x}(k)$ a rozptylem specifikovaným parametrem ρ^2. Tedy $x_k^{(n)} = \overline{x}(k) + \varepsilon_k^{(n)}$, kde $\varepsilon_k^{(n)} \sim N(0, \rho^2)$. Samotný parametr ρ^2 pak volíme v řádu šumu, popřípadě o řád větší, aby okolí postihlo možné změny trajektorie v důsledku šumu, ale současně nezasahovalo příliš daleko.
Počet vzorků M je zde konkrétně volen 100 což se ukazuje jako dostatečné množství dat pro výpočet koeficientů aproximací.

Minimalizace

Zde použitá minimalizace je neomezená, je tedy užito minimalizační funkce programu *Matlab* (*Optimization Toolbox*) `fminunc`.

Aproximace řízení

Aproximace zpětnovazebního řízení v tomto případě vychází z *triviálního CE regulátoru* navrženého v 3.1.3, který rozšiřuje

$$\begin{aligned} \text{CE regulátor :} \qquad u_k &= \frac{r_{k+1} - y_k}{\hat{b} + \varepsilon}, \quad \hat{b} + \varepsilon \neq 0, \\ \text{Aproximace řízení:} \quad \pi(k,x) &= \frac{r_{k+1} - K_1 y_k}{K_2 \hat{b}_k + K_3 P_k + K_4}. \end{aligned}$$

Koeficienty aproximace $K_{1\ldots4}$ vypočítáme v každém čase k z množiny hodnot $\left\{x^{(n)}, u^{(n)}\right\}$ lineární regresí, tedy metodou nejmenších čtverů. Provedeme následující úpravy

$$\begin{aligned} \left(K_2\hat{b}_k + K_3 P_k + K_4\right)\pi(k,x) &= r_{k+1} - K_1 y_k, \\ \begin{pmatrix} y_k & \hat{b}_k\pi(k,x) & P_k\pi(k,x) & \pi(k,x) \end{pmatrix} \begin{pmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \end{pmatrix} &= r_{k+1}. \end{aligned}$$

Rovnici označíme jako

$$\Psi K = R.$$

Následně dosadíme do Ψ vypočítaná x_k za $\begin{pmatrix} y_k & \hat{b}_k & P_k \end{pmatrix}^T$ a odpovídající vypočítaná u za $\pi(k,n)$, kdy dosazujeme celé vektory v n. Tedy výsledné Ψ je maticí rozměru $n\times4$. Aby mohla být rovnice splněna, položíme $R = r_{k+1}\begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix}^T$, tedy sloupcový vektor ze samých r_{k+1}. A koeficienty K vypočítáme metodou nejmenších čtverců jako

$$K = \left(\Psi^T\Psi\right)^{-1}\Psi R.$$

Aproximace Bellmanovy funkce

Aproximace Bellmanovy funkce je volena po vzoru dle 2.2.4 jako lineární kombinace devíti základních funkcí

$$1,\ y_k,\ \hat{b}_k,\ \ln P_k,\ y_k^2,\ y_k\hat{b}_k,\ y_k \ln P_k,\ \hat{b}_k^2,\ \hat{b}_k \ln P_k.$$

Kdy se koeficienty aproximace určují lineární regresí podle vzorce uvedeného v 2.2.4. Proměnná P_k vystupuje v souboru základních funkcí v logaritmu z výpočetních důvodů. Nejdříve bylo užito základnich funkcí pro P_k bez logaritmů, ale výpočet koeficientů aproximace selhával, protože matice $\Phi\Phi^T$ vystupující ve vzorci pro lineární regresi byla blízko singulární matici. To způsobilo problémy, při její následné inverzi, proto bylo P_k nahrazeno v bázových funkcích $\ln P_k$.

3.2 Synchronní motor s permanentními magnety

3.2.1 Popis systému

Následující model popisuje synchronní elektromotormotor s rotorem tvořeným permanentními magnety. Systém je popsán standartními rovnicemi synchronního stroje s permanentními magnety ve stacionárním tvaru

$$\begin{aligned}
\frac{di_\alpha}{dt} &= -\frac{R_s}{L_s} i_\alpha + \frac{\Psi_{PM}}{L_s}\omega \sin\vartheta + \frac{u_\alpha}{L_s}, \\
\frac{di_\beta}{dt} &= -\frac{R_s}{L_s} i_\beta - \frac{\Psi_{PM}}{L_s}\omega \cos\vartheta + \frac{u_\beta}{L_s}, \\
\frac{d\omega}{dt} &= \frac{k_p p_p^2 \Psi_{PM}}{J}\left(i_\beta \cos\vartheta - i_\alpha \sin\vartheta\right) - \frac{B}{J}\omega - \frac{p_p}{J} T_L, \\
\frac{d\vartheta}{dt} &= \omega.
\end{aligned} \tag{3.5}$$

Zde $i_{\alpha,\beta}$ reprezentují proudy a $u_{\alpha,\beta}$ napětí na statoru. Poloha (úhel otočení) rotoru je označen ϑ a ω je pak rychlost otáčení. Dále R_s je rezistance a L_s induktance statoru. Ψ_{PM} má význam magnetického toku permanentních magnetů rotoru, B tření a T_L je zatěžovací moment. Konstanta p_p označuje počet párů polů a k_p Parkovu konstantu.

Cílem je návrh řízení bez senzorů, kdy čidla pro měření polohy a otáček nejsou (z různých důvodů) přítomna. Tedy jediné měřitelné veličiny jsou:

$$y(t) = \left(i_\alpha(t), i_\beta(t)\right), u_\alpha(t), u_\beta(t).$$

Které samozřejmě můžeme měřit jen s určitou přesností. Dále předpokládáme, že vstupy u_α a u_β jsou omezené a tato omezení jsou známa. Nyní chceme dosáhnout požadovaných otáček rotoru $\overline{\omega}(t)$ (skutečnou hodnotu $\omega(t)$ neznáme, pouze ji odhadujeme ze známých hodnot $y(t)$).

3.2.2 Úprava rovnic

Diskretizace

Provedení diskretizace modelu 3.5 pomocí Eulerovy metody vede na následující diskrétní popis:

$$\begin{aligned}
i_{\alpha,k+1} &= \left(1 - \frac{R_s}{L_s}\Delta k\right) i_{\alpha,k} + \frac{\Psi_{PM}}{L_s}\Delta k\omega_k \sin\vartheta_k + \frac{\Delta k}{L_s}u_{\alpha,k},\\
i_{\beta,k+1} &= \left(1 - \frac{R_s}{L_s}\Delta k\right) i_{\beta,k} - \frac{\Psi_{PM}}{L_s}\Delta k\omega_k \cos\vartheta_k + \frac{\Delta k}{L_s}u_{\beta,k},\\
\omega_{k+1} &= \left(1 - \frac{B}{J}\Delta k\right) \omega_k + \frac{k_p p_p^2 \Psi_{PM}}{J}\Delta k \left(i_{\beta,k}\cos\vartheta_k - i_{\alpha,k}\sin\vartheta_k\right) - \frac{p_p}{J}T_L\Delta k,\\
\vartheta_{k+1} &= \vartheta_k + \omega_k\Delta k.
\end{aligned}$$

Kde Δk označuje diskrétní časový okamžik. Předpokládáme, že paremetry modelu známe, můžeme tedy provést následující substituci za účelem zjednodušení: $a = 1 - \frac{R_s}{L_s}\Delta k$, $b = \frac{\Psi_{PM}}{L_s}\Delta k$, $c = \frac{\Delta k}{L_s}$, $d = 1 - \frac{B}{J}\Delta k$, $e = \frac{k_p p_p^2 \Psi_{PM}}{J}\Delta k$. Pro jednoduchost uvažujme model bez zatížení, tedy zatěžovací moment T_L je nulovy a zjednodušený model je:

$$\begin{aligned}
i_{\alpha,k+1} &= a i_{\alpha,k} + b\omega_k \sin\vartheta_k + c u_{\alpha,k},\\
i_{\beta,k+1} &= a i_{\beta,k} - b\omega_k \cos\vartheta_k + c u_{\beta,k},\\
\omega_{k+1} &= d\omega_k + e\left(i_{\beta,k}\cos\vartheta_k - i_{\alpha,k}\sin\vartheta_k\right),\\
\vartheta_{k+1} &= \vartheta_k + \omega_k\Delta k.
\end{aligned} \tag{3.6}$$

Tyto rovnice můžeme chápat jako popis systému se stavem $x_k = (i_{\alpha,k}, i_{\beta,k}, \omega_k, \vartheta_k)$, kde předchozí soustavu rovnic zapíšeme jako $x_{k+1} = g(x_k, u_k)$.

Odhad stavu

O skutečném stavu systému x_k máme informaci pouze v podobě měření $y_k = (i_{\alpha,k}, i_{\beta,k})$. Vlastní vývoj stavových proměnných může být ovlivněn šumem, pro jednoduchost předpokládáme Gaussovský šum s nulovou střední hodnotou a kovarianční maticí M_k. Pozorování stavu, tedy výstup y_k je zatížen chybou měření, která je způsobena zaokrouhlením skutečné hodnoty na rozlišovací hodnotu stupnice přístroje. Z důvodu zjednodušení ale předpokládáme, že chyba měření bude mít ve výsledku normální rozdělení s nulovou střední hodnotou a kovarianční maticí N_k. K stejnému závěru bychom mohli dojít i použitím *centrální limitní věty* z teorie pravděpodobnosti. Tedy na vnitřní stav systému i na výstup můžeme pohlížet jako na náhodné veličiny s normálním rozdělením

$$\begin{aligned}
x_{k+1} &\sim N\left(g(x_k), M_k\right),\\
y_k &\sim N\left(\begin{pmatrix} i_{\alpha,k} \\ i_{\beta,k} \end{pmatrix}, N_k\right).
\end{aligned}$$

Nyní využijeme toho, že Kalmanův filtr je optimálním pozorovatelem lineárního systému s Gaussovským šumem. Zde uvažovaný systém 3.6 není lineární, ale můžeme využít

nelineární verze Kalmanova filtru, označované jako *rozšířený Kalmanův filtr* (Extended Kalman filter), který systém linearizuje v každém časovém kroku. Rovnice pro výpočet odhadu stavu pak budou následující

$$\begin{aligned} \hat{x}_{k+1} &= g(\hat{x}_k) - K\left(y_{k+1} - h(\hat{x}_k)\right), \\ K &= P_k C_k^T \left(C_k P_k C_k^T + N_k\right)^{-1}, \\ P_{k+1} &= A_k \left(P_k - P_k C_k^T \left(C_k P_k C_k^T + N_k\right)^{-1} C_k P_k\right) A_k^T + M_k, \end{aligned} \tag{3.7}$$

kde funkce h je $h(x_k) = (i_{\alpha,k}, i_{\beta,k})^T$ a matice A_k a C_k získáme linearizecí systému v každém kroku, tedy

$$\begin{aligned} A_k = \frac{d}{dx_k} g(x_k, u_k) &= \begin{pmatrix} a & 0 & b\sin\vartheta_k & b\omega_k\cos\vartheta_k \\ 0 & a & -b\cos\vartheta_k & b\omega_k\sin\vartheta_k \\ -e\sin\vartheta_k & e\cos\vartheta_k & d & -e\left(i_{\alpha,k}\cos\vartheta_k + i_{\beta,k}\sin\vartheta_k\right) \\ 0 & 0 & \Delta k & 1 \end{pmatrix}, \\ C_k = \frac{d}{dx_k} h(x_k) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \end{aligned}$$

Ztrátová funkce

Cílem je dosáhnout požadovaných otáček rotoru $\overline{\omega}$. Pro zjednodušení uvažujme aditivní kvadratickou ztrátovou funkci

$$J = \mathrm{E}\left\{\sum_{k=0}^{N-1} l(x_k, u_k)\right\},$$

kdy ztráta v každém časovém kroku k je

$$l(x_k, u_k) = (\omega_k - \overline{\omega}_k)^2 + r(u_{\alpha,k}^2 + u_{\beta,k}^2),$$

kde r je vhodný parametr penalizace za vstupy, který je ovšem potřeba doladit. Tento výraz můžeme upravit do maticové podoby

$$\begin{aligned} l(x_k, u_k) &= (\omega_k - \overline{\omega}_k)\, Q\, (\omega - \overline{\omega}_k) + (u_{\alpha,k}, u_{\beta,k}) \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix} \begin{pmatrix} u_{\alpha,k} \\ u_{\beta,k} \end{pmatrix} \\ &= \psi_k^T Q \psi_k + u_k^T R u_k, \end{aligned}$$

kde ψ_k značí rozdíl vektoru stavu a pořadované hodnoty $\psi_k = x_k - \overline{x}_k$ a matice Q a R pak mají tvar

$$\begin{aligned} Q &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \\ R &= \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix}. \end{aligned}$$

3.2.3 Aplikace iLDP

K implementaci iLDP algoritmu, je nutno podotknout, že jsem zatím nevytvořil funkční verzi. Je to zejména z důvodu, že se nepodařilo nalézt vhodnou aproximaci Bellmanovy funkce. Přesto zde uvedu postup aplikace tohoto algoritmu.

Postačující statistika

Pro aplikaci iLDP metody je vhodné nejdříve zavést postačující statistiku. Volme tedy $\tilde{S}_k = (\hat{x}_k, P_k)$, kde $\hat{x}_k$ má význam odhadu stavu a P_k kovarianční matice, přičemž tyto parametry se vyvíjejí v čase podle rovnic 3.7. Následně, kdybychom chtěli zahrnout do aproximace Bellmanovy funkce všechny členy $\tilde{S}_k$, jednlo by se o příliš velké množství dat. Samotný vektor $\hat{x}_k$ má v každém čase k čtyři složky a kovarianční matice P_k pak šestnáct složek. Hledáme-li aproximaci Bellmanovy funkce po vzoru 2.2.4, získáme dvacet členů prvního řádu a mnohonásobně víc členů druhého řádu. V takovémto případě je implementace algoritmu prakticky nemožná, omezíme se tedy na postačující statistiku ve tvaru $S_k = \left(\hat{x}_k, P_k^{(3,3)}, P_k^{(4,4)}\right)$, odhadu stavu a variancí odhadů složek rychlosti a otáček, které právě nemůžeme měřit.

Detaily implementace algoritmu

Základní návrh implementace vychází z verze algoritmu pro jednoduchý systém viz 3.1.6, kterou modifikuje a rozšiřuje.

Okolí $\left\{x^{(n)}\right\}$ je voleno opět jako náhodná veličina s normálním rozdělením se střední hodnotou rovnou průměrnému stavu $\bar{x}(k)$ a rozptylem specifikovaným parametrem ρ^2. Počet vzorků M je ponechán na hodnotě 100, i když byly testovány i jiné hodnoty.

Protože se úloha řízení synchronního motoru snaží do jisté míry přiblížit realitě, uvažujeme vstupy jako omezené. Tedy předpokládáme, že zdroj nemůže dodat na vstup libovolné napětí, ale je třeba dodržet jistá omezení. Zde budou omezení vstupů reprezentována podmínkou

$$u_\alpha^2 + u_\beta^2 \leq u_{max}^2,$$

kde u_{max} předpokládáme jako zadanou konstantu. Pro minimalizaci v algoritmu iLDP je tedy třeba užít omezené minimalizace, zde je použita minimalizační funkce programu *Matlab* (*Optimization Toolbox*) `fmincon`.

Volba aproximací

Aproximaci Bellmanovy funkce vytvoříme na základě postačující statistiky $S_k = \left(\hat{x}_k, P_k^{(3,3)}, P_k^{(4,4)}\right)$, tedy dle 2.2.4 volíme lineární kombinace základních funkcí a na základě zkušeností s jednoduchým systémem použijeme místo variancí jejich logaritmy. Soubor základních

funkcí je pak

$$1,\ \hat{x}_k^{(1)},\dots,\hat{x}_k^{(4)},\ \ln P_k^{(3,3)},\ \ln P_k^{(4,4)},\ \left(\hat{x}_k^{(1)}\right)^2,\ \hat{x}_k^{(1)}\hat{x}_k^{(2)},\dots,\hat{x}_k^{(1)}\hat{x}_k^{(4)},$$
$$\hat{x}_k^{(1)}\ln P_k^{(3,3)},\ \hat{x}_k^{(1)}\ln P_k^{(4,4)},\ \left(\hat{x}_k^{(2)}\right)^2,\ \hat{x}_k^{(2)}\hat{x}_k^{(3)},\ \hat{x}_k^{(2)}\hat{x}_k^{(4)},$$
$$\hat{x}_k^{(2)}\ln P_k^{(3,3)},\ \hat{x}_k^{(2)}\ln P_k^{(4,4)},\ \left(\hat{x}_k^{(3)}\right)^2,\ \hat{x}_k^{(3)}\hat{x}_k^{(4)},\ \hat{x}_k^{(3)}\ln P_k^{(3,3)},$$
$$\hat{x}_k^{(3)}\ln P_k^{(4,4)},\ \left(\hat{x}_k^{(4)}\right)^2,\ \hat{x}_k^{(4)}\ln P_k^{(3,3)},\ \hat{x}_k^{(4)}\ln P_k^{(4,4)}.$$

Ale i takový soubor základních funkcí může být příliš velký, proto byla zkoušena i možnost s vynecháním prvních dvou členů $\hat{x}_k$, tedy proudů i_α a i_β. Naopak byly přidány kvadráty logaritmů variancí. Druhý možný soubor je tedy

$$1,\ \hat{x}_k^{(3)},\ \hat{x}_k^{(4)},\ \ln P_k^{(3,3)},\ \ln P_k^{(4,4)},\ \left(\hat{x}_k^{(3)}\right)^2,\ \hat{x}_k^{(3)}\hat{x}_k^{(4)},$$
$$\hat{x}_k^{(3)}\ln P_k^{(3,3)},\ \hat{x}_k^{(3)}\ln P_k^{(4,4)},\ \left(\hat{x}_k^{(4)}\right)^2,\ \hat{x}_k^{(4)}\ln P_k^{(3,3)},$$
$$\hat{x}_k^{(4)}\ln P_k^{(4,4)},\ \left(\ln P_k^{(3,3)}\right)^2,\ \ln P_k^{(3,3)}\ln P_k^{(4,4)},\ \left(\ln P_k^{(4,4)}\right)^2.$$

Aproximace řízení byly volena a zkoušena v několika různých tvarech. Jednalo se o přímovazební řízení $\pi(k,x)=\overline{u}_k$, kde hodnotu řízení $\overline{u}_k$ získáme jako střední hodnotu přes vzorky n všech řízení $\left\{u^{(n)}\right\}$ v čase k. Dále, protože se jedná o točivý stroj, byla testována zpětnovazební aproximace řízení ve tvaru lineární kombinace funcí $\sin\vartheta,\ \cos\vartheta,\ \sin^2\vartheta,\ \cos^2\vartheta$. Nakonec byla ještě zkoušena aproximace získaná vyjádřením veličiny u_k z rovnic systému a doplnění o koeficienty po vzoru nalezení aproximace řízení v 3.1.6.

Problém aplikace iLDP

Žádný z výše uvedených postupů nevedl k nalezení funkčního řízení, pro zadaný problém synchronního motoru s permanentními magnety. Jako zásadní problém zde shledávám netriviální úkol nalezení vhodných aproximací. V případě vícerozměrného nelineárního systému to může být velmi náročné a nahodilé zkoušení volby různých aproximací zřejmě nemusí vést k cíli. Jednou z možností je, vyjít z jednodušší metody, například LQG nebo modifikované iLQG, a aproximace vytvořit po vzoru jejích funkcí. Pak bychom však obdrželi v podstatě stejně „přesnou“ metodu, jako je ta, ze které jsme vyšli, jenom by byl náš algoritmus iLDP časově náročnější z důvodu numerických výpočtů. Vhodným kandidátem na metodu z níž by bylo možné vyjít je algoritmus LQG, pomocí kterého se podařilo implementovat funkční řízení.

3.2.4 Algoritmus LQG

Zde navržený algoritmu LQG není duální, neurčitosti v systému tedy zvládá hůře než případná duální metoda. Dále algoritmus předpokládá lineární systém a kvadratickou ztrátu. Ztrátu jsme, z důvodu jednoduchosti, jako kvadratickou volili již na počátku, je

ale třeba linearizovat systém v každém časovém kroku. Dále LQG je založeno na principu separace, tedy estimátor a regulátor navrhujeme zvlášť. Estimátorem zvolíme rozšířený Kalmanův filtr, jehož rovnice jsou uvedeny v části 3.2.2. Jako regulátor použijeme LQ regulátor, který je popsán v 2.1.2.

Požadovaná hodnota

Protože jednoduchý systém v 3.1 byl lineární, bylo prakticky jedno, na jakou požadovanou hodnotu jej řídíme. Díky linearitě můžeme totiž hodnoty vždy posunout. Regulátor LQ je navržen pro lineární systém, předpokládá tedy linearitu a hledá řízení pouze na nulovou hodnotu. Tedy snaží se minimalizovat odchylku od nuly. Zde uvažovaný systém je ale nelineární a když chceme řídit na nenulovou požadovanou hodnotu, v tomto případě jde o požadované otáčky $\overline{\omega}$, nelze pouze nalézt LQ řízení na nulu a následně řešení posunout. Je proto třeba požadovanou hodnotu již od počátku zahrnout do našich uvažování a přidat ji do systému jako novou stavovou proměnou, byť může být v celém časovém vývoji systému konstantní.

Provedeme tedy substituci. Chceme řídit na nulu $\omega_k - \overline{\omega}_k$ rozdíl skutečných a požadovaných otáček, tuto veličinu tedy označíme jako ψ_k a následně $\psi_k = \omega_k - \overline{\omega}_k$. Z tohoto výrazu si můžeme vyjádřit stavovou proměnou otáček jako $\omega_k = \psi_k + \overline{\omega}_k$. Nyní v rovnicích 3.6 dosadíme za ω_k a přidáním další rovnice pro vývoj požadované hodnoty $\overline{\omega}_k$ získáme rovnice nového systému v pěti stavových proměnných

$$\begin{aligned}
i_{\alpha,k+1} &= a i_{\alpha,k} + b\left(\psi_k + \overline{\omega}_k\right)\sin\vartheta_k + c u_{\alpha,k}, \\
i_{\beta,k+1} &= a i_{\beta,k} - b\left(\psi_k + \overline{\omega}_k\right)\cos\vartheta_k + c u_{\beta,k}, \\
\psi_{k+1} &= d\left(\psi_k + \overline{\omega}_k\right) - \overline{\omega}_{k+1} + e\left(i_{\beta,k}\cos\vartheta_k - i_{\alpha,k}\sin\vartheta_k\right), \\
\vartheta_{k+1} &= \vartheta_k + \left(\psi_k + \overline{\omega}_k\right)\Delta k, \\
\overline{\omega}_{k+1} &= \overline{\omega}_k.
\end{aligned}$$

Současně se nám ale ztráta v každém časovém kroku k změní na

$$l(x_k, u_k) = \psi_k^2 + r(u_{\alpha,k}^2 + u_{\beta,k}^2) = \psi_k^T Q \psi_k + u_k^T R u_k.$$

LQG řízení

Nyní můžeme na matice popisující systém

$$\begin{aligned}
A_k &= \frac{d}{dx_k} g(x_k, u_k) \\
&= \begin{pmatrix}
a & 0 & b\sin\vartheta_k & b\left(\psi_k + \overline{\omega}_k\right)\cos\vartheta_k & b\sin\vartheta_k \\
0 & a & -b\cos\vartheta_k & b\left(\psi_k + \overline{\omega}_k\right)\sin\vartheta_k & -b\cos\vartheta_k \\
-e\sin\vartheta_k & e\cos\vartheta_k & d & -e\left(i_{\alpha,k}\cos\vartheta_k + i_{\beta,k}\sin\vartheta_k\right) & (d-1) \\
0 & 0 & \Delta k & 1 & \Delta k \\
0 & 0 & 0 & 0 & 1
\end{pmatrix},
\end{aligned}$$

$$
\begin{aligned}
B_k = \frac{d}{du_k} g(x_k, u_k) &= \begin{pmatrix} c & 0 \\ 0 & c \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \\
C_k = \frac{d}{dx_k} h(x_k) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \\
Q &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \\
R &= \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix},
\end{aligned}
$$

aplikovat rovnice pro rozšířený Kalmanův filtr a LQ regulátor. Přičemž konkrétní hodnoty počátečních hodnot a parametrů budou specifikovány v kapitole 4.

4 Výsledky

4.1 Metodika zpracování a získávání výsledků

Pro testované systémy viz kapitola3, byly jednotlivé algoritmy implementovány jako funkce programu *Matlab*. Vstupními hodnotami byla nastavení jednotlivých počátečních podmínek a parametrů pro výpočet. Jednotlivé algoritmy pak byly volány ze skriptu programu *Matlab* se stejným nastavením hodnot. Návratovými hodnotami jednotlivých funkcí reprezentujících algoritmy pak byla dosažená hodnota ztráty a posloupnost reprezentující diskrétní trajektorii systému ve stavovém prostoru. Hodnota ztráty J_{alg} byla pro účely porovnání algoritmů formulována stejně, jako součet kvadrátů odchylek výstupu od požadované hodnoty, tedy

$$J_{alg} = \sum_{k=0}^{N-1} \left(y_{k+1} - r_{k+1}\right)^2 .$$

Při porovnávání jednotlivých algoritmů nebyly samozřejmě uvažovány výsledky jednoho běhu výpočtu, ale provedlo se běhů více a následně byly výsledky zpracovány statisticky.

4.1.1 Funkce pro jednoduchý systém

Všechny konkrétní funkce programu *Matlab* reprezentující jednotlivé algoritmy mají stejnou strukturu. Na počátku dostanou jako své vstupní hodnoty parametry:

y_0 počáteční hodnota proměnné y, tedy hodnota y_k v čase $k = 0$;

y_r požadovaná hodnota y, referenční signál;

Eb střední hodnota neznámého parametru b;

P variance neznámého parametru b;

σ^2 variance šumu;

K časový horizont, pro který navrhujeme řízení;

N počet vzorkových trajektorií pro simulaci.

Následně proběhne výpočet řízení na základě konkrétního algoritmu, zpravidla podle rovnic a vzorců popsaných v části 3.1. Dále je provedena simulace běhu systému s použitím řízení získaného v předchozím kroku. V průběhu této simulace je vypočítána hodnota dosažené ztráty a posloupnost odpovídající trajektorie systému. Tyto veličiny charakterizující použití daného algoritmu na jednoduchý systém jsou na závěr vráceny jako návratové hodnoty funkce.

4.2 Výsledky algoritmu iLDP

4.2.1 Různá počáteční nastavení

(testování možná i jen pro iLDP bez ostatních)
(použité počáteční hodnoty)

4.3 Výsledky ostatních použitých metod

4.3.1 Pozorované výsledky

(získané výsledky v podobě tabulek, grafů a bar-grafů)
(slovní závěry pro jednotlivé metody)
(charakteristické rysy budou rekapitulovány v závěru)

4.3.2 CE

4.3.3 LQG

4.3.4 iLQG

4.4 Srovnání

4.4.1 Získané výsledky

4.4.2 Porovnání algoritmů

4.4.3 Konfrontace s prvotními očekáváními

4.5 Diskuze pro metodu iLDP

Závěr

Literatura

[1] Aström K. J.; Helmersson A.: Dual control of an integrator with unknown gain. *Computers and Mathematics with Applications*, 1986: s. 12A, 653–662.

[2] Bertsekas D. P.: *Dynamic Programming and Optimal Control*, ročník I. Belmont, Massachusetts: Athena Scientific, třetí vydání, 2005.

[3] Melichar J.: Lineární systémy 1: Učební texty. [online], 2010, [cit. 2010-04-03] Dostupné z: <http://www.kky.zcu.cz/uploads/courses/ls1/LS1-Ucebni-texty-2010.pdf>.

[4] Nagy I.; Pavelková L.; Suzdaleva E.; aj.: *Bayesian decision making: Theory and examples*. Prague: ÚTIA AVČR, 2005.

[5] Štecha J.: *Teorie dynamických systémů: transparenty a přednášky*. Praha: ČVUT, Elektrotechnická fakulta, 2003, ISBN 80-01-02744-9.

[6] Thompson A. M.; Cluett W. R.: Stochastic iterative dynamic programming: a Monte Carlo approach to dual control. *Automatica*, 2005: s. 41:767–778.

[7] Todorov E.; Tassa Y.: Iterative local dynamic programming. *Proceedings of the 2nd IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2009: s. 90–95.

[8] Todorov E.; Weiwei L.: A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. *In proceedings of the American Control Conference*, 2005.

[9] Virius M.: *Základy algoritmizace*. Praha: ČVUT, Fakulta jaderná a fyzikálně inženýrská, 2008, ISBN 978-80-01-04003-4.