SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY STU

Ústav robotiky a kybernetiky

Návrh evolučných algoritmov pre riadenie procesov

Dizertačná práca

Bratislava, jún 2016 **Branislav Kadlic**

Obsah

Obsah 1

1 Úvod 3

2 Evolučné algoritmy 5

2.1 Genetické algoritmy 5

2.2 Genetické programovanie 8

2.3 Gramatická evolúcia 11

2.4 Karteziánske genetické programovanie 13

2.5 Spoločné črty a rozdiely uvedených typov evolučných algoritmov 17

3 Evolučné algoritmy v riadení 18

3.1 Princíp návrhu regulátora 19

3.2 Voľba účelovej funkcie 21

3.3 Návrh robustifikovaného regulátora 24

3.4 Návrh vnútornej štruktúry regulátora – praktické príklady 25

3.5 Návrh diskrétneho regulátora 25

3.6 Navzájom prepájaná sieť regulátorov 27

3.7 Gramatická evolúcia 29

4 Karteziánske genetické programovanie pre návrh regulačných obvodov 33

4.1 Formulácia problému 33

4.2 Základné stavebné jednotky regulátorov 34

4.3 Reprezentácia jedinca 36

4.4 Evolučné operácie 42

4.5 Ilustračný príklad evolúcie regulátora 48

5 Experimentálne výsledky 52

5.1 Jednoduchý spätnoväzobný SISO regulačný obvod 53

5.2 Návrh robustifikovaného regulátora 58

5.3 Návrh riadenia vodnej turbíny 68
5.4 Návrh MIMO regulačného obvodu 73
6 Záver 80
Použitá literatúra 82

1 Úvod

Evolučné algoritmy (EA) [22,36,28,57] sú efektívny nástroj riešenia širokého spektra praktických problémov, ktoré sa dajú formulovať ako optimalizačné alebo prehľadávacie úlohy. Ich potenciál sa dá úspešne využiť v oblasti návrhu regulačných obvodov a algoritmov riadenia. Takéto aplikácie môžeme z istého pohľadu rozdeliť na dve triedy. Prvá trieda aplikácií je charakterizovaná tým, že štruktúra regulačného obvodu alebo riadiaceho algoritmu je vopred známa a pevne definovaná. To znamená, že vnútorná štruktúra regulátora alebo riadiacej jednotky je určená a objektom návrhu alebo optimalizácie je hľadanie hodnôt vopred známeho a nemeniaceho sa počtu parametrov. Na takto definovanú úlohu sa spomedzi EA zvyčajne používajú prístupy ako Genetické algoritmy (GA), Evolučné stratégie (ES), Diferenciálna evolúcia (DE), Kŕdľové algoritmy (PSO), Umelý imunitný systém (UIS), prípadne iné numerické optimalizačné metódy [7,28,57,64]. Do druhej triedy aplikácií patria také problémy, kde okrem hodnôt parametrov nie je známy ani ich počet. Čiže nie je definovaná vnútorná štruktúra, vzájomné prepojenie stavených blokov a ani ich počet v riadiacej jednotke či v regulátore. Na takéto problémy je orientovaná aj táto práca. Takéto úlohy sú riešiteľné pomocou prístupov, ktoré majú nástroje na vytváranie a modifikáciu štrukturálnych väzieb optimalizovaných objektov, ako aj hľadanie parametrov objektov s novovzniknutými štruktúrami. Medzi nástroje schopné hľadať a tvoriť nové štruktúry patria prístupy ako Genetické programovanie, Gramatická evolúcia, Karteziánske genetické programovanie a niektoré iné [22,36,61,64]. Genetické programovanie a Gramatická evolúcia sú pomerne všeobecné prístupy s veľkým potenciálom. Štruktúry, ktoré tieto prístupy sú schopné generovať nemajú spravidla žiadne obmedzenia. To znamená, že definujeme iba elementárne stavené bloky, ktoré sú medzi sebou spájané väzbami s minimálnymi obmedzeniami. Takto môžu vznikať rôznorodé objekty, často aj neintuitívne, ako by ich „prirodzene inteligentný“ návrhár nikdy nekonštruoval. Ich výsledky môžu mať nečakané, až prekvapivé vlastnosti. Na druhej strane nevýhodou takýchto „umelo inteligentných“ prístupov je vysoká výpočtová (časová) náročnosť. Prehľadávaný priestor je totiž obrovský a obmedzení v spájaní stavených

kameňov je málo. V prípade ich použitia v kybernetických aplikáciách výpočtový čas narastá tak výrazne, že to predstavuje zásadný problém. Iba pre ilustráciu môžeme uviesť, že kým parametrizácia PID regulátora pomocou genetického algoritmu je otázka sekúnd až desiatok sekúnd, návrh štruktúry SISO regulátora pomocou genetického programovania môže trvať hodiny až dni. Z tohto dôvodu je potrebné hľadať cesty a zjednodušenia, ktoré sú na úkor príliš veľkorysej všeobecnosti schopné znížiť výpočtovú náročnosť. Jedným z úspešných prístupov, ktorý bol využitý v iných aplikáciách je Karteziánske programovanie (alebo Karteziánske genetické programovanie, KGP) [61], ktorý je použitý aj v tejto práci. Jeho podstatou je, že nárast nových štruktúr a prepojení elementárnych stavených blokov, ktorý pôvodne nebol nijak limitovaný sa tu redukuje do štruktúry, kde stavebné bloky sú jednak lokalizované v pevnej pravouhlej (karteziánskej) mriežke, ktorá je naviac vopred obmedzená zvolenou veľkosťou. Algoritmus hľadá typ stavebných blokov, prepojenia medzi nimi a hodnoty ich parametrov. Cieľom tejto práce je prezentovať tento prístup pri návrhu resp. optimalizácii riadenia dynamických systémov, predovšetkým pri návrhu štruktúry regulačných obvodov resp. algoritmov regulácie.

Druhá časť tejto práce veľmi stručne opisuje vybraných reprezentantov EA. Tretia ich existujúce aplikácie pri návrhu riadenia pomocou EA, ktoré vznikli na našom pracovisku ÚRK FEI STU. Štvrtá časť je venovaná opisu nami navrhnutého prístupu, ktorý voľne nadväzuje na tieto výsledky a využíva Karteziánske programovanie pri implementácii návrhu riadenia. Piata časť dokumentuje dosiahnuté výsledky a výsledky simulácií. Zhodnotenie prínosov práce je v poslednej šiestej kapitole.

2 Evolučné algoritmy

Evolučné algoritmy [7,28,57,64] sú nedeterministické optimalizačné metódy, ktoré vychádzajú zo základných prístupov Darwinovej evolučnej teórie, založenej na prirodzenom výbere, náhodných zmenách jedincov a prežití úspešnejších. Stochastickými zmenami vo vlastnostiach reprezentantov (jedincov) z množiny potenciálnych riešení (tzv. populácie) a uprednostňovaním úspešnejších jedincov spomedzi ostatných je možné konvergovať ku globálnemu extrému danej úlohy alebo aspoň kvalitnému suboptimálnemu výsledku.

2.1 Genetické algoritmy

Genetický algoritmus (GA) [10,7,57,28,64] je univerzálnym prístupom optimalizácie/prehľadávania, ktorý má schopnosť (na rozdiel od mnohých exaktných analytických a deterministických numerických metód) opustiť lokálny extrém a smerovať ku globálnemu optimu. Má potenciál sa vysporiadať s nelinearitami, s nekonvexnosťou úlohy či s veľkou dimenziou prehľadávaného priestoru. Uplatňujú sa tu z prírody odpozorované princípy prežitia tých schopnejších jedincov, pričom počas dlhodobého vývoja dochádza k vymieraniu menej schopných, čo má za následok, že daná populácia sa stáva silnejšou. Zjednodušene možno povedať, že genetické algoritmy simulujú evolúciu, počas ktorej v populácii viacerých jedincov zanikajú tie menej úspešné a tie úspešnejšie majú možnosť sa ďalej vyvíjať a prispôsobovať sa. Podmienkou realizovateľnosti tohto prístupu je možnosť vyhodnotenia účelovej funkcie (tzv. fitness funkcie) v ľubovoľnom bode prehľadávaného priestoru. Základné znaky genetických algoritmov sú [57]:

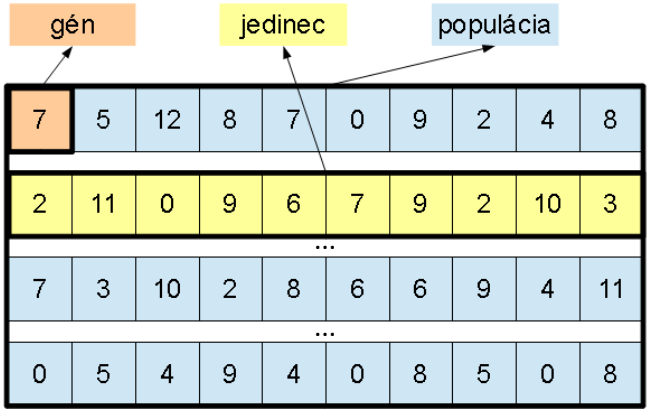
- sú schopné vyviaznuť z okolia lokálneho extrému a priblížiť sa ku globálnemu extrému,
- realizujú paralelné prehľadávanie priestoru vo viacerých smeroch zároveň,
- nevyžadujú pomocné informácie o vývoji riešenia,
- nie sú obmedzené požiadavkami spojitosti alebo konvexnosti účelovej funkcie,

- sú schopné riešiť optimalizačné problémy s desiatkami až stovkami premenných,
- sú jednoducho aplikovateľné na široké spektrum úloh,
- ale patria k časovo najnáročnejším riešeniam.

Základné dátové štruktúry GA

- **gén** – je jeden parameter optimalizovaného objektu, spolu s ostatnými génmi tvorí reťazec, gény sú elementárne stavebné prvky reťazcov
- **reťazec (chromozóm, jedinec)** – je množina hľadaných parametrov optimalizovaného problému, z tohto reťazca je možné vypočítať hodnotu účelovej funkcie (z angl. fitness funkcie, funkcie úspešnosti)
- **populácia** – je množina vhodného počtu reťazcov

Ilustrácia základných dátových štruktúr je na Obr. 2.1.1.

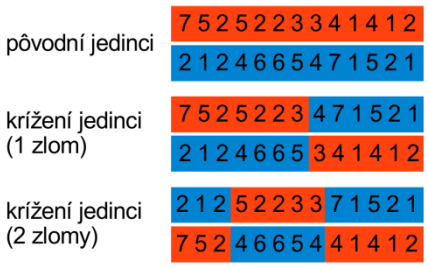


Obrázok 2.1.1: Príklad štruktúry populácie

Základné operácie GA

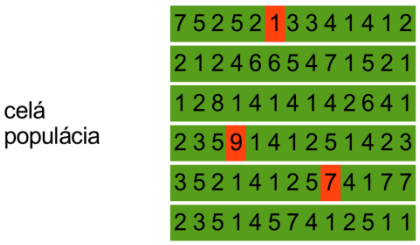
Základné operácie GA sú výber, mutácia a kríženie.

- **kríženie** – je operácia, pri ktorej sa náhodne rozdelí 2 a niekedy aj viac jedincov a tieto si navzájom vymenia 2 alebo viaceré časti reťazcov. Môže tak dochádzať k výhodnej kombinácii viacerých, potenciálne úspešných častí jedincov (Obr. 2.1.2).



Obrázok 2.1.2: Príklad kríženia

- **mutácia** – táto operácia nahradí jeden alebo skupinu génov novými, náhodne vygenerovanými, táto operácia má za účel diverzifikovať informácie v populácii. Početnosť mutácií génov v populácii je jedným z najdôležitejších parametrov GA (Obr. 2.1.3).

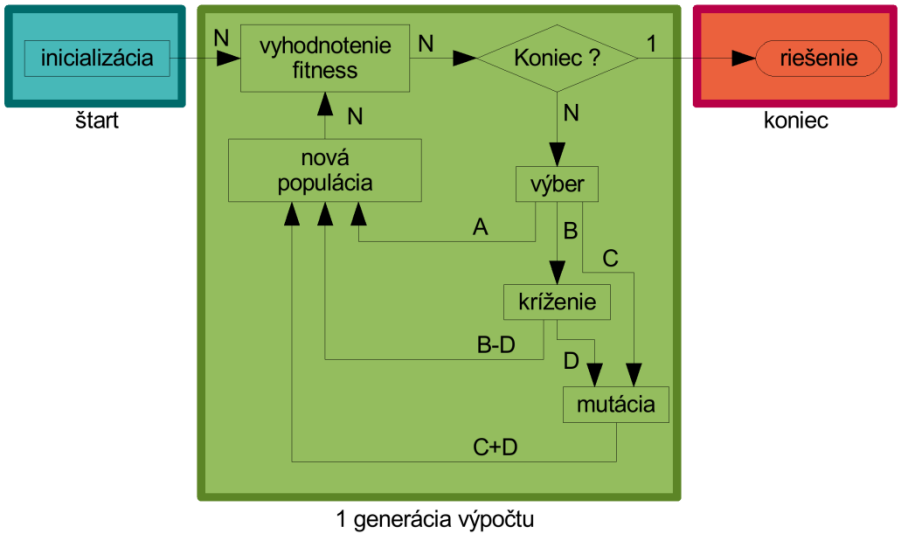


Obrázok 2.1.3: Príklad mutácie – červené gény boli náhodne modifikované - mutované

- **výber** – je operácia, pri ktorej na základe zvolenej stratégie vyberieme skupinu jedincov do novej populácie pričom časť z nich môže prejsť procesom kríženia, časť procesom mutácie a časť by mala byť ponechaná bez zmeny, vrátane najlepšieho jedinca, aby bola zabezpečená monotónna konvergencia algoritmu. Pri výbere tiež platí, že pravdepodobnosť pre výber úspešnejších jedincov je väčšia ako pravdepodobnosť pre výber menej úspešných.

Jednotlivé operácie môžu byť vzájomne kombinované, aby pri konkrétnom probléme dokázali hľadať čo najefektívnejšie a aby sa dokázali priblížiť ku globálnemu extrému s vysokou rýchlosťou a presnosťou.

Jednoduchý genetický algoritmus môže vyzerať ako na Obr. 2.1.4. V úvode sa náhodne inicializuje populácia berúc do úvahy obmedzenia jednotlivých génov. Prebehne vyhodnotenie relatívnej úspešnosti (fitness) jednotlivých reťazcov. Ak sú splnené požadované kritéria, vyberie sa z populácie najlepší reťazec, ktorý je riešením. Ak kritéria splnené nie sú, vyberie sa zvolený počet reťazcov z pôvodnej populácie použitím rôznych metód ako napríklad: výberu podľa úspešnosti, náhodného výberu, turnajového výberu, ruletového výberu a mnoho iných [57]. Časť postupuje priamo do novej populácie. Nad ostatnými reťazcami sa realizujú operácie kríženia a mutácie. Všetky tieto reťazce spolu vytvoria novú populáciu.



Obrázok 2.1.4: Všeobecný genetický algoritmus

2.2 Genetické programovanie

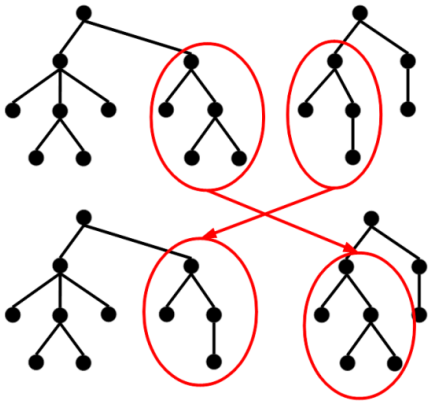
Základy genetického programovania (GP) podobne ako GA vychádzajú z princípov napodobenia evolúcie. Prvotnou potrebou bolo automatizované zlepšovanie kvality počítačových programov alebo vlastností dátových štruktúr pomocou náhodných zmien. Dôležitou súčasťou postupov, ktoré mali za úlohu tento ciel dosiahnuť, bolo definovanie nového spôsobu reprezentácie jedinca,

ktorý už v tomto prípade nemal iba neznáme parametre, ale aj neznámu vnútornú štruktúru [36,3].

Genetické algoritmy používajú metódy, ktoré majú za úlohu optimalizovať najmä parametre objektu. Oproti tomu genetické programovanie rozširuje proces optimalizácie aj o návrh štruktúry. Toto rozšírenie so sebou ale nesie mnoho komplikácií. Jednou z nich je enormný nárast dimenzie prehľadávaného priestoru. Druhou je, ako zakódovať premenlivú štruktúru do lineárneho reťazca, na ktorý sa potom dajú aplikovať elementárne genetické operácie. Vzhľadom na fakt, že táto štruktúra môže obsahovať rôzne množstvo funkčných blokov, vzniká potreba variabilného reťazca, ktorá má ďalej za následok potrebu modifikácií v genetických operáciách [22]. Reprezentáciu jedinca v GP je možné v závislosti od aplikácie realizovať viacerými spôsobmi, ako sú stromové štruktúry, tabuľky, lineárne reťazce registrových operácií a prípadne aj iné [25, 3]. Bez ujmy na všeobecnosti tu ilustrujme iba prvý prípad. Potenciálne riešenia sú reprezentované stromami, ktorých uzly predstavujú funkcie alebo operandy a hrany medzi nimi určujú väzby medzi operáciami [57]. Jedince reprezentované stromami sú potom transformované do lineárnych postupností symbolov pomocou vhodnej transformácie ako je napr. Readov lineárny kód [57, 22] alebo inej. Nad takýmito reťazcami sa potom realizujú podobné genetické operácie ako v GA.

Genetické operácie v GP pomocou stromových štruktúr

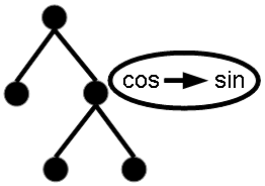
- **kríženie** – je podobné kríženiu, ktoré sa používa v genetických



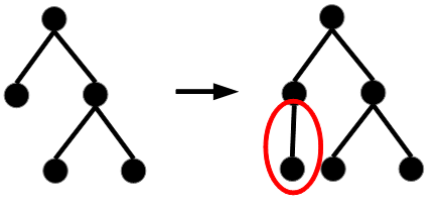
Obrázok 2.2.1: Kríženie

algoritmoch no dochádza pri ňom k výmene častí podstromov (Obr. 2.2.1), teda reťazcov s variabilnou dĺžkou

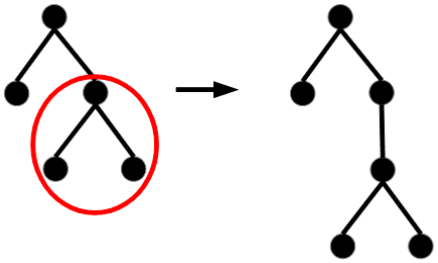
- **mutácia** – v tomto prípade môže mať mutácia mnoho podôb akými sú: zmena obsahu uzla (Obr. 2.2.2), pridanie (Obr. 2.2.3), zmena (Obr. 2.2.4) či odstránenie podstromu (Obr. 2.2.5).



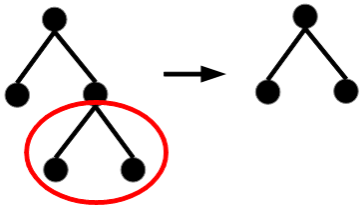
Obrázok 2.2.2: Mutácia zmenou funkcie uzla



Obrázok 2.2.3: Mutácia pridaním podstromu



Obrázok 2.2.4: Mutácia zmenou časti podstromu



Obrázok 2.2.5: Mutácia odstránením podstromu

Algoritmus genetického programovania je v mnohých prípadoch veľmi podobný, ako ten použitý pri genetickom algoritme. Pri tomto algoritme však treba brať do úvahy fakt, že dĺžka reťazca sa počas evolúcie mení. Tomu je potrebné prispôsobiť kritériá výberu tak, aby zabraňovali nadbytočnému narastaniu stromov. Rovnako je dôležité inicializovať počiatočnú populáciu s dostatočnou rôznorodosťou kombinácií spojení a typov uzlov. V genetickom programovaní je často vhodné voliť populáciu väčšiu než pri GA, bežne aj stovky alebo tisíc jedincov [57]. Viac o aplikácií GP pri návrhu riadenia bude uvedené neskôr v tejto práci.

2.3 Gramatická evolúcia

Gramatická evolúcia je alternatívou genetického programovania, ktoré je založené na reprezentácii jedincov formou gramatiky a je vhodné na istú triedu úloh, kde jedince je možné vyjadriť rozvojom štruktúry formou gramatiky. Tento prístup modifikuje pohľad na reprezentáciu jedincov pomocou stromových štruktúr alebo iných flexibilných verzií reprezentácií, ktoré sú zaužívané predovšetkým v genetickom programovaní. Takýto postup sa v podstate snaží napodobniť základné biologické princípy prekladu génov na vonkajšie znaky. Gény každého jedinca sú reprezentované lineárnym reťazcom s variabilnou dĺžkou tvorenou celými číslami. Nad takto vytvorenými reťazcami sú následne aplikované genetické operácie ako sú kríženie alebo mutácia. Na to aby bolo možné vyhodnotiť fitness takéhoto reťazca, je potrebné tento genotyp v podobe reťazca preložiť do programom zrozumiteľnej štruktúry. Preklad sa, ako už bolo naznačené, realizuje za pomoci pravidiel, ktoré sú definované v gramatike [42, 41]. Pravidlá samotnej gramatiky sa môžu meniť. To znamená, že evolučné postupy a reťazce môžu byť síce identické, ale výsledok sa môže líšiť v závislosti na použitej množine gramatických pravidiel, čo bude mať po preložení za následok aj odlišný výsledok [34].

Stromová štruktúra sa používa iba ako dočasne vytvorená štruktúra v procese postupného uplatňovania pravidiel zadefinovaných v gramatike. Celý lineárny reťazec sa zoskupí do celkov s pevným počtom génov.

Gramatika môže byť definovaná ako množina ukončujúcich výrazov, neukončujúcich výrazov, počiatočného symbolu a pravidiel pre tvorbu (Obr. 2.3.1).

Neukončujúce výrazy	expr, op, pre-op
Ukončujúce výrazy	sin, +, -, /, *, x, 1.0, (,)
Počiatočný symbol	<expr>
Pravidlá tvorby	1 <expr>::= <expr><op><expr> (0) (<expr><op><expr>) (1) <pre-op>(<expr>) (2) <var> (3)
	2 <op>::= + (0) - (1) / (2) * (3)
	3 <pre-op>::= sin (0)
	4 <var>::= x (0) 1.0 (1)

Obrázok 2.3.1: Backus–Naurova forma gramatiky

Proces prekladu funguje nasledovne. Vyberie sa ľavý krajný neukončujúci výraz (*<expr>*). Potom vydelíme prvý prvok genotypu počtom možných gramatických pravidiel. (*R = C mod N*, *220 mod 4 = 0)* Následne odoberieme zvyšok po tomto delení a aplikujeme pravidlo s týmto číslom (delenie 4 = N je kvôli tomu, že *<expr>* obsahuje štyri pravidlá). Tento postup sa opakuje pokiaľ už nie je možné aplikovať ďalšie pravidlá (Obr. 2.3.2).

Výraz	**C**	**N**	**R**	**Pravidlo**
<expr>	220	4	0	<expr>::=<expr><op><expr>
<expr><op><expr>	240	4	0	<expr>::=<expr><op><expr>
<expr><op><expr><op><expr>	220	4	0	<expr>::=<expr><op><expr>
<expr><op><expr><op><expr><op><expr>	203	4	3	<expr>::=<var>
<var><op><expr><op><expr><op><expr>	101	2	1	<var>::=1.0
1.0<op><expr><op><expr><op><expr>	53	4	1	<op>::=-
1.0-<expr><op><expr><op><expr>	202	4	2	<expr>::=<pre-op>(<expr>)
1.0-<pre-op>(<expr>)<op><expr><op><expr>				<pre-op>::=sin
1.0-sin(<expr>)<op><expr><op><expr>	203	4	3	<expr>::=<var>
1.0-sin(<var>)<op><expr><op><expr>	102	2	0	<var>::=x
1.0-sin(x)<op><expr><op><expr>	55	4	3	<op>::=*
1.0-sin(x)*<expr><op><expr>	223	4	3	<expr>::=<var>
1.0-sin(x)*<var><op><expr>	202	2	0	<var>::=x
1.0-sin(x)*x<op><expr>	243	4	3	<op>::=*
1.0-sin(x)*x*<expr>	134	4	2	<expr>::=<pre-op>(<expr>)
1.0-sin(x)*x*<pre-op>(<expr>)				<pre-op>::=sin
1.0-sin(x)*x*sin(<expr>)	35	4	3	<expr>::=<var>
1.0-sin(x)*x*sin(<var>)	202	2	0	<var>::=x
1.0-sin(x)*x*sin(x)				

Obrázok 2.3.2: Spôsob prekladu

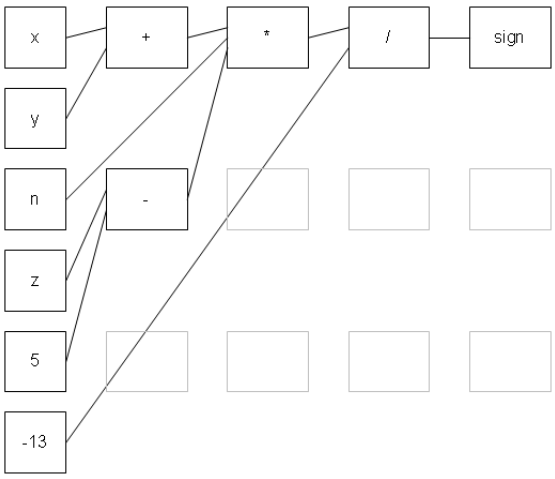
V prípadoch, kde sa nachádza pre výraz len jedno jediné pravidlo aplikuje sa toto priamo, lebo by sme zbytočne znižovali počet použiteľných prvkov z genotypu (Obr. 2.3.3). V závislosti na požiadavkách je možné použiť aj cyklickú formu genotypu, kedy pri dosiahnutí posledného prvku sa pokračuje od prvého [36].

220	240	220	203	101	53	202	203	102	55	223	202	243	134	35	202	203	140	39	202	203	102

Obrázok 2.3.3: Príklad lineárneho reťazca, genotypu

2.4 Karteziánske genetické programovanie

Karteziánske programovanie je pomerne mladá časť genetického programovania [36]. Vzniklo z metód, ktoré slúžili pre vývoj číslicových obvodov. Ako už samotný názov napovedá, jedná sa o algoritmy, ktoré pracujú s pravouhlou mriežkou a v tomto prípade môžeme hovoriť o mriežke, ktorá obsahuje uzly reprezentované funkciami. Na Obr. 2.4.1 je príklad takejto mriežky.



Obrázok 2.4.1: Príklad funkcii rozmiestnených do mriežky

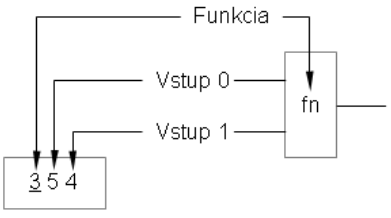
Základná verzia karteziánskeho programovania

V karteziánskom genetickom programovaní je väčšina algoritmov reprezentovaná vo forme grafov. Grafy sú reprezentovane ako dvojrozmerná mriežka uzlov. Gény, označme spolu ako genotyp, sú tvorené dátami, ktoré

bližšie popisujú spojenia medzi jednotlivými uzlami a operácie, ktoré sa v každom z uzlov vykonávajú. Pri prepise do konečného algoritmu môžu byť niektoré časti vynechané. Väčšinou sa vynechávajú časti, ktoré nemajú žiaden vplyv na konečný výsledok riešenia, ale môžu nastať aj prípady, kedy sú nerealizovateľné, alebo neúmerne náročné na výpočtový čas spolu so zanedbateľným vplyvom na výsledok. Dôvody sa môžu líšiť od tipu, požiadaviek a spôsobu realizácie samotnej aplikácie.

Gény základnej verzie karteziánskeho programovania majú nemennú dĺžku a teda pred samotným začiatkom výpočtov je dobré odhadnúť náročnosť procesu a stanoviť túto dĺžku. Túto dĺžku je ale možné voliť aj bez predchádzajúcej analýzy, no pri riešení sa môže vyskytnúť zopár problémov. Napríklad voľbou priveľkej dĺžky pre riešenie jednoduchého problému sa môže stať, že výsledný algoritmus bude omnoho náročnejší na výpočtový čas, realizáciu alebo bude obsahovať zbytočne zložité konštrukcie. Naopak pri voľbe malého počtu génov pre zložitý problém by mal byť síce výpočtový čas nižší no finálne riešenie nemusí spĺňať požadované kritériá.

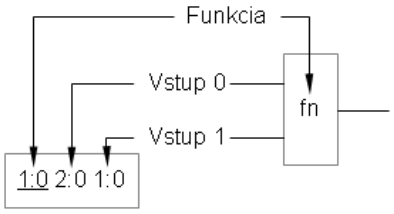
Na začiatku je tiež potrebné si stanoviť množinu funkcii, ktoré sa budú môcť uplatniť v každom z uzlov (Obr. 2.4.2). Túto množinu treba voliť s prihliadnutím na konečnu realizovateľnosť riešenia. Teda ak v konečnom riešení nesmie byť operácia delenia, tak prirodzene nemá význam ju do tejto množiny zahrnúť. Jednou zo základných požiadaviek pre riešenie problémov za pomoci princípov karteziánskeho genetického programovania je poznať vstupy a výstupy, ktoré je možné poskytnúť procesu optimalizácie. Čím viac vstupov a výstupov má algoritmus k dispozícii, tým viac možností získa na ich vhodné zapojenie, ale rovnako aj rastie náročnosť na ich správne zapojenie pre riešenie celého problému [36].



Obrázok 2.4.2: Uzol a jeho reprezentácia

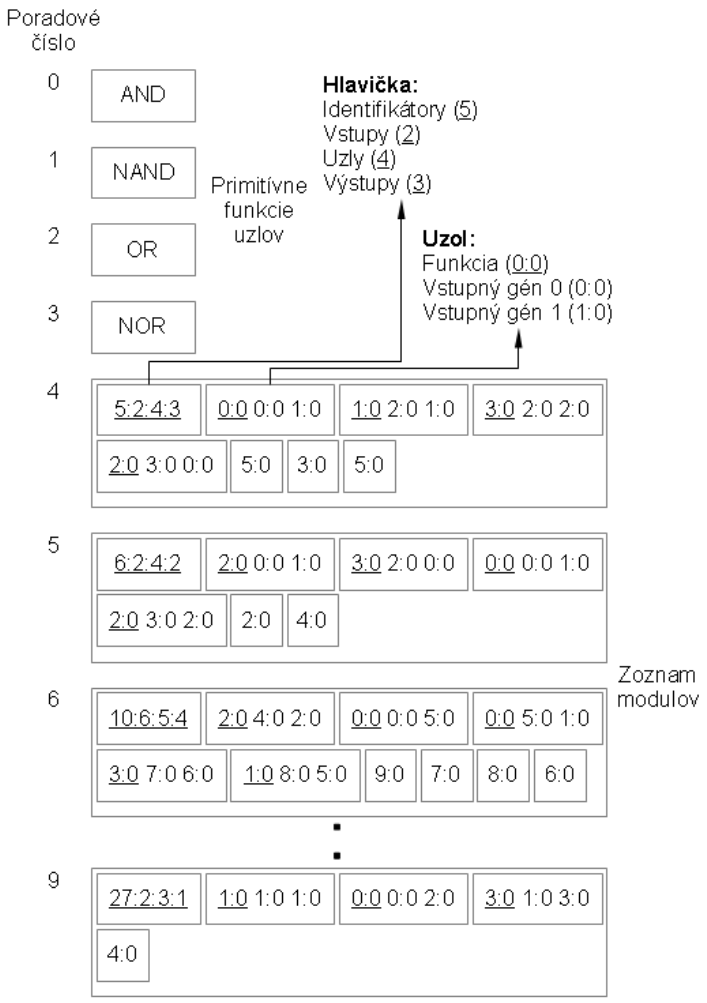
Karteziánske programovanie s vkladaním modulov

Tento prístup je akýmsi vylepšením pre základnú verziu. Oproti základnému prístupu využíva myšlienky zberu modulov, teda kusov už navrhnutých častí, ktoré sa ukázali veľmi vhodne pre výpočet celkového výsledku v predchádzajúcich iteráciách. Modulom (Obr. 2.4.3) v karteziánskom programovaní s vkladaním modulov teda chápeme skupinu uzlov zo základnej verzie, ktorá môže byť použitá ako funkcia. Tieto moduly môžu byť počas procesu evolúcie, za pomoci špeciálnych operácii, postupne pridávané a následne tie menej úspešné aj odstraňované (Obr. 2.4.4). Vkladanie modulov vychádza zo získavania a viacnásobného používania užitočných úsekov. Zmena prináša nevyhnutnú modifikáciu základného prístupu. Pre vkladanie modulov sa môžu uplatniť ale aj iné zákonitosti. Tiež je možné znovu používať aj bloky, ktoré na prvý pohľad nemusia prispievať k výraznému zlepšeniu výsledku, môžu mať nastavený akýsi čas exspirácie a ak sa riešenie s ich použitím nezlepší môžu byť tieto moduly postupne modifikované alebo odstraňované [36].



Obrázok 2.4.3: Modul a jeho reprezentácia

Rovnako sa dá pristúpiť aj k myšlienke, že všetky uzly sa stanú akýmisi modulmi, ktoré pozostávajú z uzlov, poprípade ďalších modulov. Tu sa môže prejaviť dekompozícia zložitejších problémov za pomoci evolúcie a ďalej riešiť čiastkové optimalizácie [36].



Obrázok 2.4.4: Príklad sady modulov – prvé štyri moduly 0 - 3 sú primitívne funkcie, pričom ostatné moduly 4 – 9 sú zložené z iných modulov

Modifikácie karteziánskeho programovania

Základný model karteziánskeho programovania je možné ďalej škálovať, jednou z možných modifikácii je pracovať s variabilnou dĺžkou reťazca génov (genotypom). Táto zmena môže vyžadovať modifikáciu reprezentácie genotypu alebo základných genetických operácii ako je kríženie alebo mutácia. Rozširuje však možnosť pridávať rôzne procedúry, ktoré sa aplikujú priamo nad genotypom, ako pridať alebo odstrániť vstup alebo výstup, meniť dynamicky počet uzlov. Rozširovať množinu funkcií priamo počas procesu hľadania riešenia. Tieto prístupy je vhodné používať pre riešenie zložitejších problémov, kedy zabraňujú opakovanému znovu vytváraniu úspešných skupín

uzlov, modulov, zapájaniu vstupov, ktoré nemajú vplyv na celkový výsledok a tým urýchľujú hľadanie [36].

2.5 Spoločné črty a rozdiely uvedených typov evolučných algoritmov

Hlavným cieľom genetických algoritmov je optimalizovať sadu definovaných parametrov. Vo všeobecnosti sa ich počet, ktorý je počtom génov v chromozóme počas behu algoritmu nemení a závisí na nemennej štruktúre problému. Veľké množstvo úloh túto požiadavku spĺňa. Avšak v mnohých aplikáciách je hľadaná štruktúra objektu neznáma a môže byť časťou problému, ktorú je potrebné vyriešiť. Takéto problémy nemôžu byť reprezentované konštantnou dĺžkou reťazca. Pre tieto problémy je možné použiť genetické programovanie, gramatickú evolúciu či karteziánske genetické programovanie. Pre spomenuté evolučné prístupy je možné použiť viacero reprezentácií jedinca ako stromovú štruktúru, tabuľkovú reprezentáciu či lineárny kód. Reprezentácia jedinca je hlavným rozdielom medzi genetickými algoritmami a ostatnými spomenutými evolučnými algoritmami. Tieto ostatné evolučné prístupy môžu byť použité pre konštrukčné úlohy s vopred neznámou štruktúrou ako sú návrh programov, návrh riadiacich algoritmov, návrh elektrických obvodov a mnoho iných problémov. Gény sú v týchto prípadoch:

- sada stavebných blokov (konštrukčné časti robota, konštrukčné časti stavieb a iné)
- inštrukcie programovacích jazykov (podmienky, cykly, výnimky a iné)
- funkcie, sekvencie inštrukcií (posuň objekt, zvýš rýchlosť a iné)
- stavebné bloky regulátorov (zosilnenie, integrátor, derivátor, saturácia, násobička, relé a iné)

elektrické súčiastky, logické hradlá (logickým súčet, logický súčin, exkluzívny logický súčet a iné)

3 Evolučné algoritmy v riadení

Metódy pre návrh regulátorov musia rešpektovať rôzne požiadavky kladené na výkon riadiacich systémov. Regulátory často obsahujú mnoho parametrov a obmedzení. Optimalizačný proces môže byť komplikovaný, nespojitý alebo nekonvexný a často nie je možné analytickými metódami dosiahnuť uspokojivé výsledky. Ako už bolo spomenuté, optimalizačné techniky sú schopné tiež tvoriť nové pravidlá riadenia a neintuitívne riešenia. V nedávnej dobe boli evolučné algoritmy aplikované v riadení procesov na vyriešenie širokého spektra optimalizačných problémov. Táto kapitola sa zameriava na návrh regulátorov pre spojité systémy.

Zhrňme si doposiaľ známe postupy využívajúce evolučné algoritmy do dvoch skupín. Prvá skupina využíva evolučné algoritmy ako výkonnú optimalizačnú stratégiu pre analyticky formulované metódy návrhu riadenia. Parametre regulátorov alebo rôznych dynamických systémov sú navrhnuté analyticky s ohľadom na stabilitu alebo požadovaný výkon systémov [20, 26, 33, 55]. Druhá skupina využíva vyhodnotenie výsledkov simulácie modelu uzavretého regulačného obvodu [13, 21, 31, 39, 62, 63]. Objavuje sa aj prístup optimalizácie viacerých kritérií naraz, kde je medzi kritéria zahrnutá analytická formulácia ale aj výsledok časovej odozvy [63].

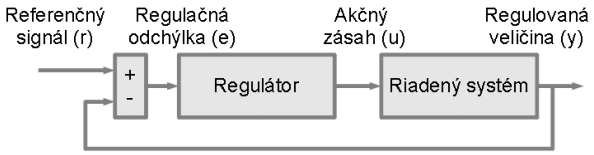
Ak návrh zahŕňa nielen hľadanie parametrov vopred pripravenej štruktúry riadenia ale aj hľadanie štruktúry samotnej, je možné použiť genetické programovanie [25, 3, 23, 57, 46]. Pre optimalizáciu štruktúry riadenia môže byť použitý aj hierarchický genetický algoritmus [33]. Genetické programovanie bolo použité ku generovaniu Lyapunovových funkcií [11]. Prehľad návrhov riadení založených na evolučných princípoch [8, 31].

V práci je priamo začlenený prístup založený na vyhodnocovaní výsledkov simulácií časovej odozvy v uzavretej regulačnej slučke. Predstavený prístup sa zaoberá priamym použitím optimalizácie založenej na evolučných stratégiách v hľadaní v priestore parametrov regulátora s rozsiahlymi počítačovými simuláciami navrhnutých systémov v uzavretej regulačnej slučke [60, 59, 57, 46]. Výsledky simulácií sú zásadnou časťou

minimalizovanej účelovej funkcie. Neskôr bude ukázané ako je pomocou tohto prístupu návrh optimálnych parametrov pre dynamické systémy pretvorený na štandardný viacrozmerný optimalizačný problém.

3.1 Princíp návrhu regulátora

Ako už bolo spomenuté, cieľom návrhu riadenia je poskytnutie požadovaných statických a dynamických vlastností riadeného procesu, zväčša vo forme dobre známych charakteristických vlastností: maximálne preregulovanie, trvalá regulačná odchýlka, doba nábehu, doba ustálenia alebo iné integrálne ukazovatele kvality [5, 27, ...].



Obrázok 3.1.1: Jednoduchý spätnoväzobný regulačný obvod

S ohľadom na všestrannosť si predstavme jednoduchú regulačnú slučku (Obr. 3.1.1), kde *y* je riadená veličina *u* je riadiaci zásah, *r* je referenčný signál a *e* je regulačná odchýlka (*e*=*r*-*y*). Uvažujme, že je model riadeného systému známy. Kvalita riadenia v uzavretej regulačnej slučke bude vyhodnotená pomocou jednoduchého integrálneho kritéria (3.1.1),

$$I_{AE} = \int_0^T |e(t)|dt \qquad \textbf{(3.1.1)}$$

kde *T* je čas simulácie. Diskrétna forma tohto kritéria kvality (3.1.2),

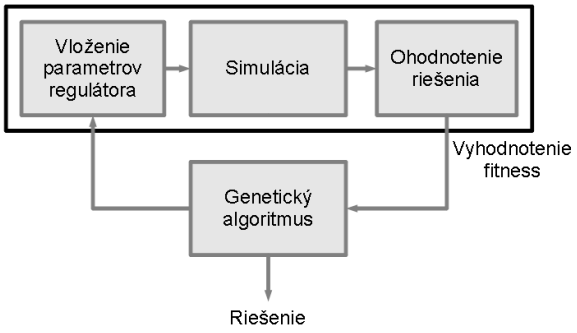
$$I_{AE} = \sum_{k=1}^{N} T_{s,k}|e_k| \qquad \textbf{(3.1.2)}$$

kde $T_{s,k}$ je krok simulácie a *N* je počet krokov simulácie.

Návrh regulátora je v podstate optimalizačná úloha. Hľadanie parametrov regulátora z vopred definovaného priestoru parametrov sa uskutočňuje pomocou genetických algoritmov, ktoré sa snažia minimalizovať účelovú

funkciu (3.1.1, 3.1.2). Vyhodnotenie účelovej funkcie pozostáva z dvoch krokov. Prvým krokom je simulácia časovej odozvy regulačnej slučky. Druhým je vyhodnotenie účelovej funkcie. Pri návrhu systémov s viacerými vstupmi a viacerými výstupmi (MIMO) alebo inými typmi regulátorov (fuzzy regulátory, regulátory na bázy neurónových sietí) je rozmer priestoru hľadaných parametrov pomerne veľký.

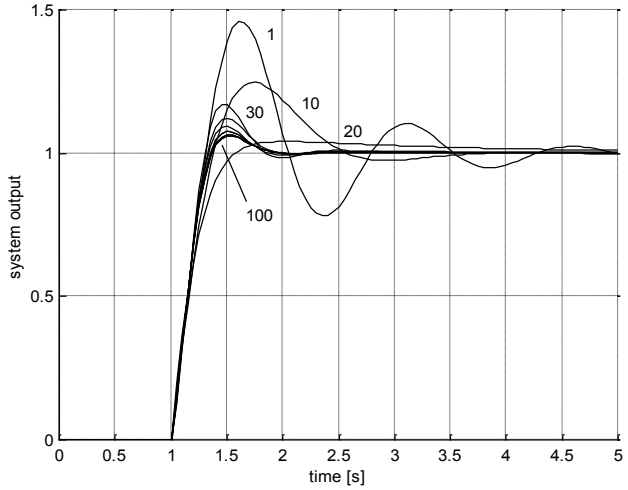
Bloková schéma popisujúca princíp návrhu regulátora za pomoci evolučných algoritmov je znázornená na Obr. 3.1.2.



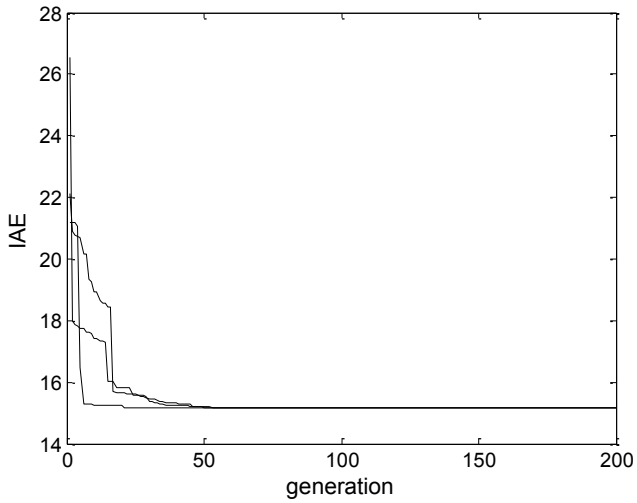
Obrázok 3.1.2: Bloková schéma návrhu regulátora pomocou evolučných algoritmov

Príklad na Obr. 3.1.3 znázorňuje evolúciu PID regulátora s využitím integrálneho kritéria (3.1.1, 3.1.2). Po niekoľkých generáciách je najlepšie riešenie z aktuálnej populácie zobrazované na grafe. Ak sa po 100 generáciách riešenie zásadne nezmení môže byť genetický algoritmus ukončený. Konvergencia účelovej funkcie počas troch nezávislých behov genetického algoritmu je znázornená na Obr. 3.1.4.

Návrh regulátora pre komplexné systémy môže mať viacero problémov, ako existencia viacerých riešení, či trvanie optimalizačného procesu, kde je často uspokojivé aj riešenie blízke optimu. Prístup k rýchlosti konvergencie návrhu riadenia je podobný ako v prípade iných numerických problémov, v ktorých sa využívajú genetické algoritmy. Závisí od priestoru parametrov, rozmerov prehľadávaného priestoru a výkonu genetického algoritmu.



Obrázok 3.1.3: Evolúcia parametrov PID regulátora po 1., 10., 20., 30. a 100. generácii



Obrázok 3.1.4: Konvergencia účelovej funkcie návrhu PID regulátora pre tri nezávislé behy pre veľkosť populácie 30 jedincov – účelová funkcia aktuálne najlepšieho jedinca v populácií pre číslo generácie

3.2 Voľba účelovej funkcie

Uvážme, že genetický algoritmus (GA) našiel optimálne riešenie alebo riešenie blízke optimu v užívateľom definovanom priestore parametrov regulátora. Voľba účelovej funkcie má zásadný vplyv na dynamiku regulačnej slučky. Štandardne sa pri použití základných integrálnych kritérií (3.1.1, 3.1.2) dosahuje rýchla riadiaca odozva s nízkym preregulovaním medzi 2-5%

(Obr. 3.1.3). Pre dosiahnutie rôznych cieľov môžu byť použité aj iné kritéria [60, 57].

Ak je potrebné zmierniť preregulovanie alebo zamedziť osciláciám je odporúčané vložiť dodatočné integrálne členy, ktoré zahŕňajú absolútne hodnoty prvej alebo aj druhej derivácie regulačnej odchýlky (3.2.1).

$$J = \int_0^T \left(\alpha|e(t)| + \beta|e'(t)| + \gamma|e''(t)|\right)dt \qquad \textbf{(3.2.1)}$$

Zvyšovať hodnoty β and γ je možné iba s ohľadom na parameter α kde α, β, γ sú váhové koeficienty. Derivácie regulačnej odchýlky môžu byť nahrádzané deriváciami riadenej veličiny. Pre diskrétny prípad je integrál nahrádzaný sumou a derivácie diferenciami. Dobré výsledky môžu byť dosiahnuté aj použitím energetického kritéria (3.2.2),

$$J = \alpha\eta + (1-\alpha)t_s \qquad \textbf{(3.2.2)}$$

kde η je preregulovanie, t_s je doba regulácie a $0<\alpha<1$ je váhovací koeficient. Sledovanie referenčného signálu je dosiahnuté pomocou minimalizácie funkcie (3.2.3).

$$J = \int (y_r(t) - y(t))^2 dt \qquad \textbf{(3.2.3)}$$

Minimalizácia energie, ktorá je potrebná na riadenie môže byť dosiahnutá použitím vzťahu (3.2.4),

$$J = \int (\alpha e^2(t) + (1-\alpha)u^2(t))dt \qquad \textbf{(3.2.4)}$$

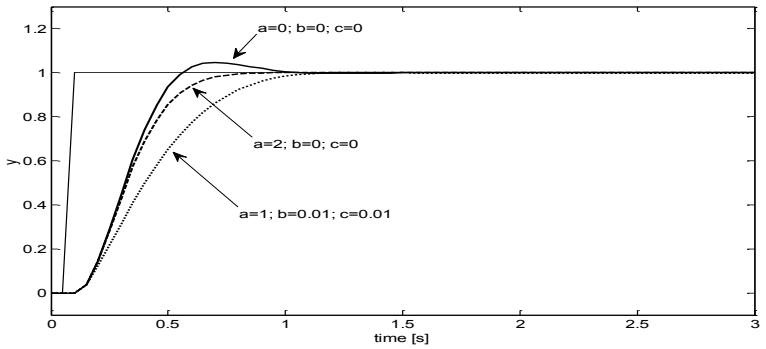
kde u je riadiaca veličina (akčný zásah). Univerzálne kritérium, ktoré

$$J = \int_0^T \left(|e(t)| + a|e'(t)| + b|u(t)| + c|u'(t)|\right)dt \qquad \textbf{(3.2.5)}$$

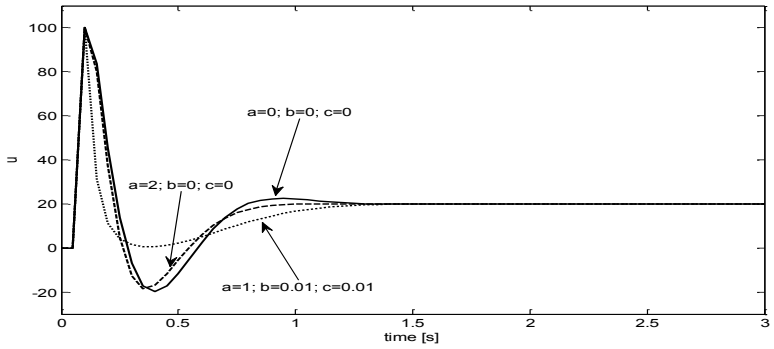
kombinuje viaceré požiadavky je v tvare funkcie (3.2.5).

Toto kritérium zahŕňa minimalizáciu oscilácií, pomocou zvyšovania hodnoty *a* minimalizáciu absolútnej hodnoty riadiacej veličiny *u*, pomocou zvyšovania hodnoty *b* a minimalizáciu zmeny riadiacej veličiny pomocou parametra *c*.

Na grafoch (Obr. 3.2.1, Obr. 3.2.2) je možné vidieť, aký majú vplyv rôzne konfigurácie parametrov *a*, *b* a *c* vo funkcií (3.2.5) ak ich vplyv na výstupnú (riadenú) veličinu uzavretej slučky a riadiacu veličinu.



Obrázok 3.2.1: Časové odozvy riadenej veličiny v uzavretej regulačnej slučke pre rôzne nastavenie parametra a, b a c.



Obrázok 3.2.2: Časové odozvy riadiacej veličiny v uzavretej regulačnej slučke pre rôzne hodnoty parametrov a,b,c.

V návrhu pre systémy s viacerými vstupmi a viacerými výstupmi (MIMO) je postup rovnaký, iba pre každú výstupnú časť je zvolená funkcia s prislúchajúcimi váhovými koeficientami (3.2.6).

$$J = \sum_{i=1}^{N} w_i J_i \qquad \textbf{(3.2.6)}$$

3.3 Návrh robustifikovaného regulátora

Uvažujme $c = \{c_1, c_2, \ldots, c_q\}$ ako sadu navrhnutých parametrov regulátora a nech $s = \{s_1, s_2, \ldots, s_r\}$ je sada parametrov riadeného systému. Počas behu systému sa môžu parametre s_i pohybovať v istom neurčitom priestore (3.3.1),

$$s_{i,\min} \leq s_i \leq s_{i,\max}\ ;\quad i = 1,2,\ldots,r \qquad \textbf{(3.3.1)}$$

kde $s_{i,min}$ and $s_{i,max}$ sú minimálne a maximálne hodnoty *i*-teho systému.

Uvažujme, že *W* predstavuje rozdielne pracovné body riadeného procesu, definovaného rozdielnymi vektormi *s*, ktoré majú byť riadené robustifikovaným regulátorom. Pre tento prípad si predstavme účelovú funkciu (3.3.2)

$$J = \sum_{i=1}^{W} J_i \qquad \textbf{(3.3.2)}$$

zahŕňajúcu hodnotu účelovej funkcie vo všetkých pracovných bodoch. Odporúča sa zahrnúť do simulačného modelu nameraný šum z reálneho systému alebo iné možné rušenia, či iné očakávané situácie, ktoré môžu v reálnom systéme nastať. Alternatívne do sady pracovných bodov (*W*) môžeme použiť sadu vektorov parametrov systému umiestnených vo vrcholoch mnohostena reprezentujúceho hranice priestoru parametra *S*.

Poznámka ku stabilite uzavretej slučky

Vzhľadom na minimalizáciu použitej účelovej funkcie, je stabilita v uzavretej regulačnej slučke vlastnosť, ktorá je už v každom riešení zahrnutá. Počas evolučného procesu sú nestabilné chromozómy eliminované kvôli svojej vysokej hodnote účelovej funkcie a riešenie je smerované do oblasti stabilných parametrov. Napriek tomuto faktu, je možné zahrnúť test stability do každého vyhodnotenia hodnoty účelovej funkcie. Nestabilné riešenia budú dodatočne ovplyvnené pokutovými hodnotami.

3.4 Návrh vnútornej štruktúry regulátora – praktické príklady

Nasledujúce časti nie sú pôvodným výsledkom tejto dizertačnej práce [52, 46, 47]. Výsledky tu prezentované vznikli na našom pracovisku ÚRK FEI STU, niektoré aj za nepriameho prispenia autora tejto dizertačnej práce. Považujeme však za zmysluplné túto časť zaradiť do textu tejto práce, nakoľko naše pôvodné výsledky uvedené v neskorších kapitolách nadväzujú na túto kapitolu. To môže jednak prispieť k lepšiemu pochopeniu nami dosiahnutých výsledkov, ako aj spoluvytvára jeden kompaktný a logický celok.

V predchádzajúcich častiach návrhu bolo cieľom optimalizovať parametre definovaného regulátora. V tejto sekcii je tiež optimalizovaná aj interná štruktúra regulátora. Pretože je štruktúra regulátora neznáma nie je známa ani množina parametrov. Kvôli tomuto dôvodu je dĺžka reťazca premenlivá. Pre takéto prípady je používané Genetické programovanie (GP) [25, 3]. Neskôr sú predstavené dva ďalšie prístupy [25, 3]. V prvom prístupe je použitý diskrétny algoritmus riadenia. V druhom navzájom prepájaná sieť základných statických a dynamických stavebných blokov, použitých pre návrh regulátora.

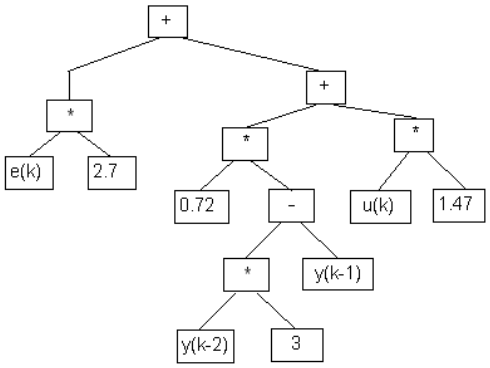
3.5 Návrh diskrétneho regulátora

V navrhovanom prístupe je použitý diskrétny rekurentný algoritmus riadenia ako funkcia časovo oneskorených vstupných premenných [57]. Boli uvažované nasledovné premenné: *e(k), e(k-1), ..., e(k-m), y(k), y(k-1), ..., y(k-n), u(k-1), u(k-2), ..., u(k-p), r(k)*, kde *e* je regulačná odchýlka, *y* je riadiaca veličina, *u* je riadiaca veličina a *k* je referenčný signál (skok požadovanej hodnoty). Cieľom je nájsť optimálnu formu funkcie regulátora (3.5.1).

$$F(\theta)=?$$
$$\theta=\{e(k),\ldots, e(k-m), y(k),\ldots, y(k-n), u(k-1),\ldots, u(k-p),r(k),c\} \quad \textbf{(3.5.1)}$$

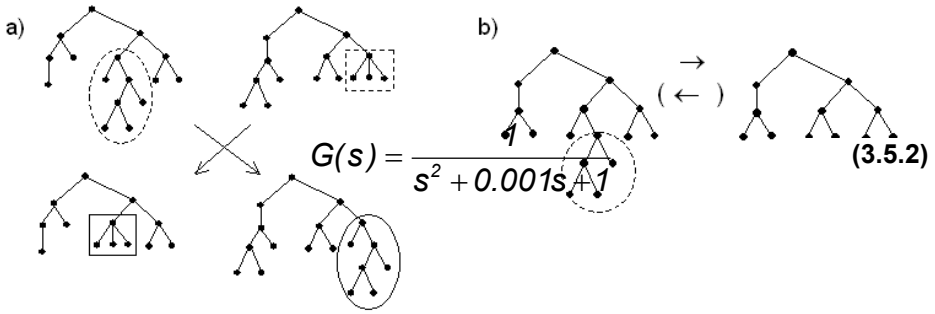
pričom je minimalizovaná účelová funkcia (3.2.1), kde *c* sú reálne konštanty. Pre reprezentáciu funkcie *F* je pre Genetické programovanie (GP) bežne používaná stromová reprezentácia. Funkčné uzly obsahujú operácie: +, -, *

a ukončovacie uzly obsahujú argumenty operácie z vektora θ. Príklad takéhoto stromu je na Obr. 3.5.1.



Obrázok 3.5.1: Príklad stromovej štruktúry riadiacej funkcie

Nasledujúce genetické operácie sú aplikované na jedincov v stromovej reprezentácii. Prvá je kríženie, ktorá spočíva vo výmene náhodne vybraného podstromu z dvoch rodičovských stromov (Obr. 3.5.2). Druhou je mutácia, ktorej boli použité nasledujúce typy: náhrada náhodne vybraného podstormu iným náhodne vygenerovaným podstromom, odstránenie náhodne vybraného podstromu alebo pridanie náhodne vygenerovaného podstormu (Obr. 3.5.2). Posledným typom mutácie je mutácia ukončovacieho uzla v strome, ktorý je náhodne zmenený z konštanty alebo premennej na inú konštantu či premennú.



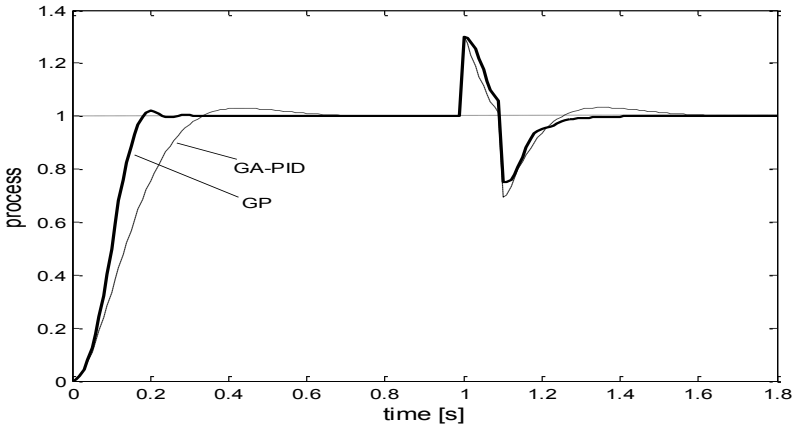
Obrázok 3.5.2: Genetické operácie: a, kríženie dvoch stromov, b, mutácia odstránením (pridaním) podstromu

Demonštrujme si prístup návrhu na návrhu regulátora pre jednoduchý lineárny systém v tvare prenosovej funkcie (3.5.2).

Bola použitá funkcia (3.2.1). Po 3000 generáciách vznikol regulátor v nasledujúcej rekurentnej forme (3.5.3).

$$u(k) = 6.74([y(k) - y(k-2) - e(k)][e(k-2) - 10.88] [23.25 + y(k-1) + y(k-2)]) - -87.82e(k-2)[9.33 + y(k-2)] + y(k) + e(k) - e(k-1) \quad \textbf{(3.5.3)}$$

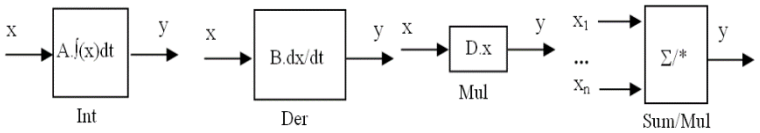
Na Obr. 3.5.3 je porovnaný výstup uzavretej slučky na referenčný skok a vonkajšiu poruchu s výstupom pre PID regulátor navrhnutý genetickým algoritmom (GA) za použitia rovnakej účelovej funkcie.



Obrázok 3.5.3: Porovnanie časových odoziev systému pre regulátor navrhnutý Genetickým programovaním (GP) s PID regulátorom

3.6 Navzájom prepájaná sieť regulátorov

Táto reprezentácia regulátora je založená na navzájom prepojenej sieti, ktorá obsahuje nasledujúce spojité dynamické a statické funkčné bloky: integrátor, derivátor, zosilnenie (násobenie konštantou), súčtový člen a násobiaci člen (Obr. 3.6.1), kde *A*, *B*, a *D* sú reálne konštanty [46]. Cieľom je nájsť optimálnu riadiacu sieť, ktorá sa skladá z takýchto elementárnych funkčných blokov a ich vzájomných prepojení, ktoré minimalizujú účelovú funkciu (3.2.1)



Obrázok 3.6.1: Knižnica použitých elementárnych blokov

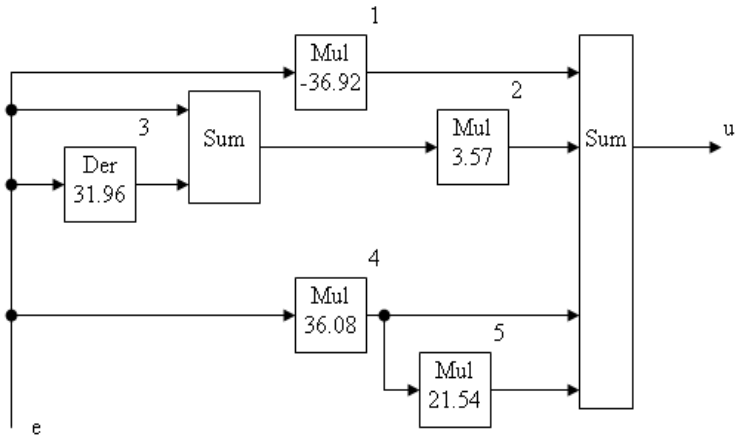
Pre zmeny v chromozómoch boli navrhnuté špeciálne operácie kríženia a mutácie. V prípade kríženia, sú dva vybrané rodičovské chromozómy rozdelené, obidva na náhodné pozície medzi stĺpcami tabuľky a následne sú

ich príslušné časti vymenené. Pre mutáciu boli použité nasledujúce postupy: prvým je náhodná zmena typu bloku, (napr. integrátor) je zamenený za iný typ bloku (napr. derivátor). Ďalším typom mutácie je náhodné zmazanie alebo pridanie bloku do schémy. Posledným typom mutácie je náhodná zmena konštanty, ktorá je súčasťou každého bloku. Po každej modifikácii môžu byť niektoré prepojenia chýbajúce alebo presahujú rozsah. Pre tieto prípady sú nepotrebné odstránené alebo sú nové náhodne pridané a náhodne pripojené k náhodným blokom. Návrhový postup bol overený na návrhu regulátora pre lineárny dynamický systém (3.6.1).

$$G(s) = \frac{s^2 + 3s + 1}{4s^5 + 8s^4 + 10s^3 + 6s^2 + 3s + 1} \qquad \textbf{(3.6.1)}$$

Blok	**1**	**2**	**3**	**4**	**5**
Parametre	Mul	Mul	Der	Mul	Mul
	0	0	2	0	0
	-1	3	-1	-1	4
	36.92	3.57	31.96	36.08	21.54

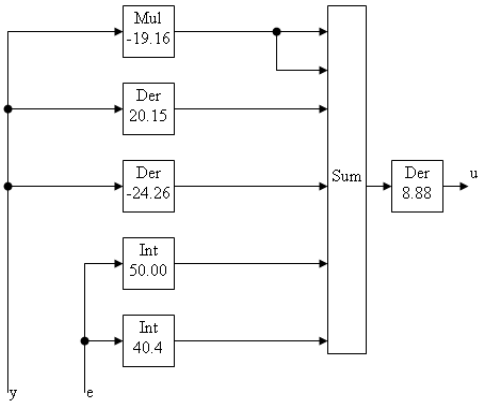
Tabuľka 3.6.1: Zápis chromozómu



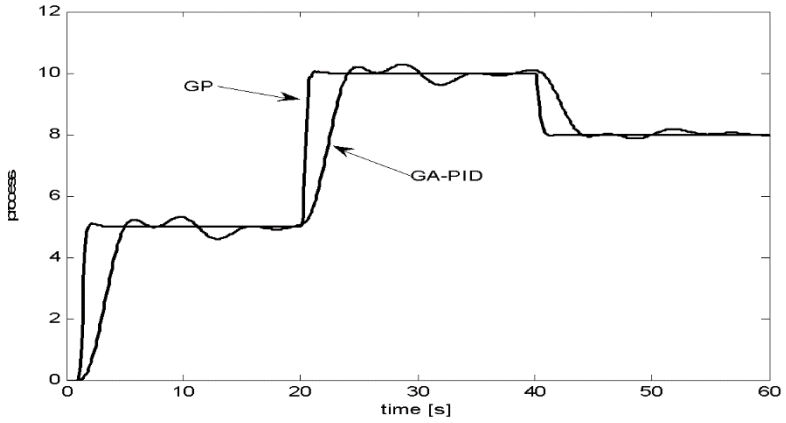
Obrázok 3.6.2: Príklad riadiacej siete

Cieľom bolo minimalizovať účelovú funkciu (3.2.1). Navrhnutá štruktúra riadenia bola získaná po 3000 generáciách s veľkosťou populácie 80 jedincov, mierou mutácie 0,3 a mierou kríženia 0,3 je znázornený na

Obr. 3.6.3. Výstup uzavretej slučky získaného regulátora (GP) na referenčný signál je na Obr. 3.6.4 porovnaný s PID regulátorom, ktorý bol optimalizovaný použitím genetického algoritmu (GA).



Obrázok 3.6.3: Štruktúra regulátora



Obrázok 3.6.4: Časová odozva navrhnutého regulátora pre skoky referenčného signálu z hodnoty 0 na 5, 10 a 8

3.7 Gramatická evolúcia

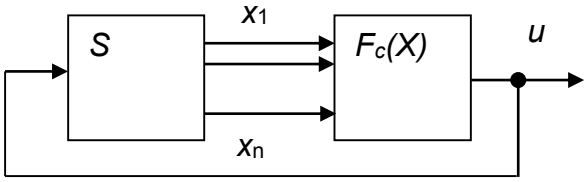
Riadiaca štruktúra (riadiaci algoritmus) je reprezentovaný vo forme funkcie, ktorej argumenty sú dynamické signály (riadená veličina, referenčný signál, regulačná odchýlka alebo iné signály a ich derivácie) a výstup riadiacej funkcie je aktuálna riadiaca hodnota. V nasledujúcej časti je popísaný princíp návrhu, reprezentácia jedinca a vyhodnocovanie účelovej funkcie. Následne je navrhovaný prístup demonštrovaný na príklade návrhu regulátora pre nelineárny dynamický systém.

Definícia problému a reprezentácia jedinca

Regulátor spojitého dynamického systému je reprezentovaný vo forme spojitej funkcie F_c, ktorej argumenty sú vybrané premenné riadiacej slučky (Obr. 3.7.1). Cieľom návrhu je nájsť takú funkciu, ktorá zachová najlepší možný výkon – takú, ktorá maximalizuje definovanú fitness funkciu. Funkcia F_c reprezentuje predpis definujúci, ktoré argumenty x_i sú vybrané, ktoré matematické vzťahy sú medzi nimi a ktoré parametre matematických operácií. Uvažovaná sada vstupných premenných (3.7.1),

$$\mathrm{xi} \in \{e_i,\ ie_i,\ de_i,\ d^2e_i,\ y_i,\ dy_i,\ \ldots\ iné\} \qquad \textbf{(3.7.1)}$$

e je regulačná odchýlka ($e=r-y$), r je referenčný signál, i_{ei} je integrál regulačnej odchýlky (e), $d^n e_i$ sú derivácie regulačnej odchýlky (e), y je riadená veličina a možné sú aj iné vstupné premenné regulačnej slučky.



Obrázok 3.7.1: Riadiaca slučka, Fc – riadiaca funkcia, S – riadený dynamický systém

Uvažujme nasledujúcu sadu matematických operácií: *{+, -, *, /, ^}*. Vo všeobecnosti môžu byť použité iné operácie a funkcie. Každé potenciálne riešenie (jedinec populácie) je reprezentovaný vo forme 4 génov (3.7.2),

$$jedinec = \{f_1,f_2,\ldots,f_n,\ x_1,\ x_2,\ldots,x_n,\ p_1,p_2,\ldots,p_n,\ b_1,b_2,\ldots,b_n\} \qquad \textbf{(3.7.2)}$$

f_i je kód matematickej operácie x_i je argument vstupnej premennej, p_i je parameter reprezentujúci násobiaci koeficient každého vhodného vstupu premennej x_i a b_i je koeficient operácie umocnenia (ak je potrebná). Kódovanie matematických operácií f_i v uvažovanej gramatike je nasledovné (Obr. 3.7.2). Symbol λ je substituovaný: $\lambda_i=p_i x_i$.

0: λ →λ
1: λ →(λ+ λ)
2: λ →(λ- λ)
3: λ →(λ* λ)
4: λ →(λ/ λ)
5: λ →(λ)^b

Obrázok 3.7.2: Prvky použitej gramatiky

Výsledky experimentov

Návrh založený na gramatickej evolúcii bol testovaný na niekoľkých lineárnych a nelineárnych systémoch. V prípade lineárnych systémov nebolo dosiahnuté obzvlášť výrazné zlepšenie v porovnaní s metódami návrhu regulátorov pomocou genetických algoritmov. Na druhú stranu v prípade nelineárnych systémov boli výhody z nelineárnych vlastností regulátorov navrhovaných gramatickou evolúciou (GE) značne lepšie. Na Obr. 3.7.3 je porovnanie návrhu PID regulátora pomocou genetických algoritmov (GA) s regulátorom navrhnutým gramatickou evolúciou (GE). Nelineárny riadený objekt je definovaný vo forme diferenciálnej rovnice (3.7.3).

$$\ddot{y} = 10u - 1\dot{y} - 2y - 4y^3 \qquad \textbf{(3.7.3)}$$

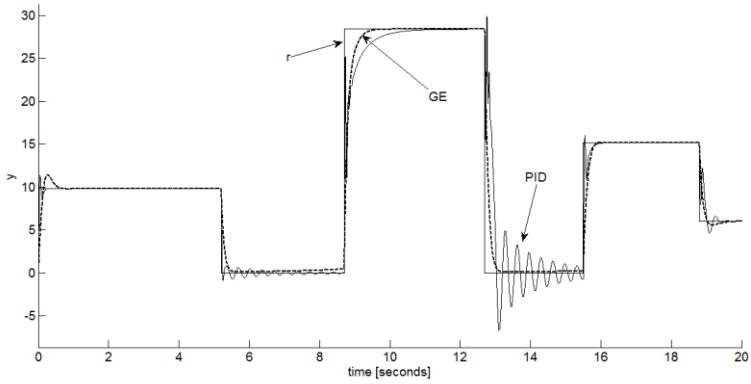
Účelová funkcia mala tvar (3.2.1) s parametrom α *=0.01*. Parametre PID regulátora sú *P=23*, *I=1882*, *D = 5,6*. Fenotyp regulátora (3.7.4)

$$\begin{aligned} F_c: \quad & u = 178.98e - 17.35y' \\ & + 1.4988(\int e)(4.79e + 64.72\int e + 90.52y) \end{aligned} \qquad \textbf{(3.7.4)}$$

a jeho genotyp (Tab. 3.7.1) sú znázornené.

+	+	-	*	-	+	λ	λ	λ	λ
98.21	80.77	-17.33	-1.4988	4.79	-64.72	90.52	-	-	-
e	e	dy	ie	e	ie	y	-	-	-

Tabuľka 3.7.1: Genotyp regulátora



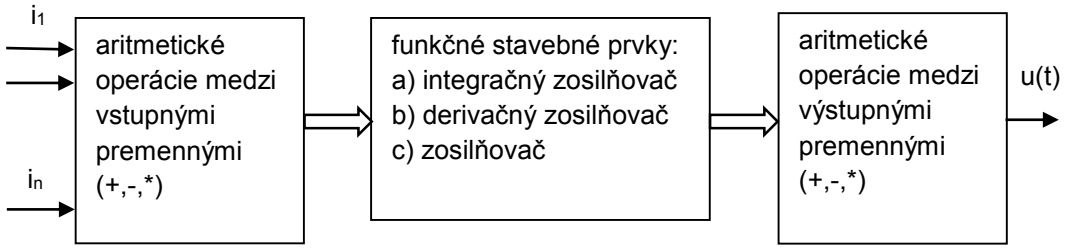
Obrázok 3.7.3: Porovnanie časovej odozvy výstupnej veličiny PID regulátora a regulátora navrhnutého gramatickou evolúciou (GE)

4 Karteziánske genetické programovanie pre návrh regulačných obvodov

Táto kapitola vysvetľuje princíp nášho návrhu aplikácie karteziánskeho genetického programovania pre návrh regulátorov.

4.1 Formulácia problému

Cieľom je aplikácia karteziánskeho genetického programovania pri návrhu regulátorov vo všeobecnosti pre nelineárne dynamické systémy, ktoré sú vytvárané vyberaním a spájaním elementárnych stavebných blokov, ich vzájomných prepojení a hľadaním ich parametrov. Každý regulátor pozostáva z elementárnych jednotiek: blokov a prepojení. Viaceré bloky rôzneho druhu sú prepájané navzájom s cieľom realizovať štruktúru, ktorá generuje riadiacu veličinu pre určený objekt riadenia.



Obrázok 4.1.1: Jeden základný stavebný blok

Vstupy blokov sú spracované použitím aritmetických operácií (OP): súčet, rozdiel, násobenie, ktoré definujú matematické vzťahy medzi všetkými vstupnými signálmi konkrétneho bloku. Následne je signál spracovaný dynamickým operátorom, ako: integrátor, derivátor či jednotkové zosilnenie a na konci je násobený zosilnením. Takéto stavebné bloky sú organizované do stĺpcových vektorov, kde výstup z jedného bloku môže byť spojený s ľubovoľným vstupom iného alebo iných blokov. Vstupné signály regulátora (ako napríklad regulačná odchýlka, riadená veličina, merateľná porucha a iné) môžu byť spojené so vstupom každého iného stavebného bloku. Ľubovoľne výstupy blokov môžu reprezentovať výstup(y) regulátora. Takto vytvorená sieť môže byť považovaná za ortogonálnu (karteziánsku) mriežku navzájom

prepojených stavebných blokov, kde každý uzol mriežky obsahuje elementárnu operáciu. Prepojenie uzlov nie je úplne ľubovoľné. Je limitované vyššie uvedenými pravidlami. Cieľom návrhu regulátora je nájsť optimálnu alebo aspoň vyhovujúcu sieť stavebných blokov, ktoré maximalizujú/minimalizujú zvolenú kriteriálnu funkciu v definovanej regulačnej slučke. Sieť môže byť chápaná ako nelineárna funkcia (4.1.1),

$$u(t)=F(\theta)$$

$$\theta=\{i_1,i_2,\ldots,i_n\};\ i_i \in \{e, e', e'', y, y', y'', s_1 \ldots s_m, d_1 \ldots d_k\} \quad \textbf{(4.1.1)}$$

kde vektor θ obsahuje všetky uvažované vstupné premenné, $u(t)$ je riadiaca veličina (akčný zásah) regulátora, i_i sú vstupné signály regulátora e, e', e'' je regulačná odchýlka a jej prvá a druhá derivácia, y, y', y'' je výstupná veličina a jej prvá a druhá derivácia, s_i sú vnútorné stavové premenné riadeného systému a d_i sú vonkajšie poruchy alebo informácie z prostredia (Obr. 4.4.9). Je možné použiť akékoľvek signály, ktoré sú k dispozícii.

4.2 Základné stavebné jednotky regulátorov

Jedinec v karteziánskom genetickom programovaní (KGP) je potenciálne riešenie, ktoré je realizované ako funkcia (4.1.1) a reprezentuje kompletnú informáciu regulátora vrátane jej vnútornej štruktúry a parametrov. Každý jedinec je členom populácie. Populácia obsahuje množinu jedincov a reprezentuje základnú dátovú štruktúru karteziánskeho genetického programovania.

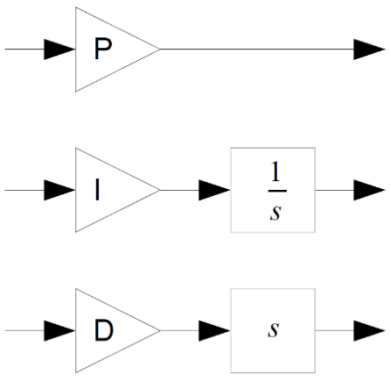
Na Obr. 4.2.1 je znázornený princíp návrhu regulátora (algoritmu riadenia) pomocou evolučného algoritmu.

Každý jedinec obsahuje základné stavebné prvky: bloky a prepojenia.

Bloky - elementárne funkčné prvky

Sú parametrizovateľnými prvkami, kde sa vykonáva časť najdôležitejších matematických operácií so vstupnými dátami, pozostávajú z nasledujúcich matematických operácii (Obr. 4.2.2):

- **zosilnenie** - má jeden parameter, ktorým je veľkosť zosilnenia
- **integrátor** - má jeden parameter, ktorý vyjadruje veľkosť zosilnenia prislúchajúce operácii integrovania
- **derivátor** - má jeden parameter vyjadrujúci veľkosť zosilnenia prislúchajúceho operácii derivovania



Obrázok 4.2.2: Znázornenie funkčných blokov v laplaceovej transformácií

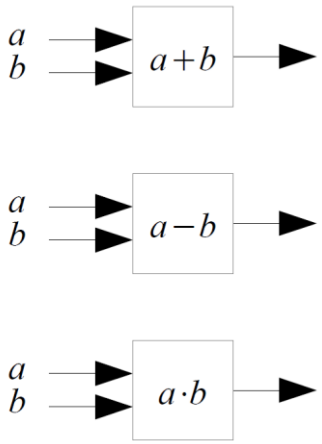
Prepojenia a matematické operácie

Sú jedným z typov stavebných objektov, ktoré slúžia na vzájomne prepojenie vstupov, výstupov a blokov. Každé prepojenie má typ operácie, s ktorými vychádza zo vstupného bloku, funkčného bloku alebo vstupuje do výstupného bloku, funkčného bloku. Teda prepojenia zabezpečujú správne spojenie rôznorodých zdrojov informácií do jedného tak, aby bol k dispozícii pre blok, do ktorého vstupuje. Zoznam ich operácií je nasledovný (Obr. 4.2.3):

- **súčet** – pre prípad jedného alebo prvého prepojenia vstupujúceho do funkčného bloku alebo výstupného bloku sa táto operácia neprejaví, keďže neexistuje prvý sčítanec, teda prebehne sčítanie nuly a hodnoty

z bloku, z ktorého toto prepojenie vychádza, pre prípad viacerých vstupov sa pripočíta k hodnote predchádzajúceho prepojenia

- **rozdiel** – pre prípad jedného spojenia vstupujúceho do funkčného bloku alebo výstupného bloku sa táto operácia prejaví tak, že hodnote z bloku, z ktorého vychádza prepojenie zmení hodnotu na opačnú, pre prípad viacerých vstupov sa odpočíta od hodnoty predchádzajúceho prepojenia
- **súčin** – pre prípad jedného alebo prvého prepojenia vstupujúceho do funkčného bloku alebo výstupného bloku sa táto operácia ignoruje, pre viaceré vstupy sa prejaví ako násobenie hodnoty predchádzajúceho prepojenia



Obrázok 4.2.3: Znázornenie prepojení a ich operácií

4.3 Reprezentácia jedinca

Ako bolo uvedené, jeden jedinec (ďalej používame pojem reťazec) populácie je jedno potenciálne riešenie regulátora (riadenia). Pozostáva z viacerých prepojených stavebných blokov. Každý stavebný blok pozostáva z elementárnych prvkov podľa Obr. 4.2.2 a matematických operácií medzi vstupmi a výstupmi bloku (Obr. 4.2.3). Z programátorského hľadiska (z hľadiska reprezentácie genómu jedinca v KGP) bol navrhnutý nasledovný spôsob zakódovania jedinca.

Hlavička

Je použitá na popis vektorov tela, pre statickú dĺžku reťazca je konštantná. Obsahuje nasledujúce prvky:

- počet vstupov regulátora (I)
- počet výstupov regulátora (0)
- maximálny počet blokov (B)
- maximálny počet prepojení (C)

Hlavička je tvorená 4 parametrami ({I, O, B, C}).

Telo

Je tvorené vektorom parametrov, ktoré popisujú jednotlivé bloky a prepojenia.

Pre popis všetkých funkčných blokov sú potrebné dva vektory ($\{\{B_T\}, \{B_G\}\}$), každý z nich má dĺžku B:

- operácia funkčného bloku (B_T)
- veľkosť zosilnenia (B_G)

Pre popis všetkých prepojení sú potrebné tri vektory ($\{\{C_I\}, \{C_O\}, \{C_T\}\}$), každý z nich má dĺžku C.

- index bloku, z ktorého prepojenie vychádza (C_I)
- index bloku kam prepojenie vstupuje (C_O)
- operácia, s ktorou hodnoty vstupujú do iného bloku (C_T)

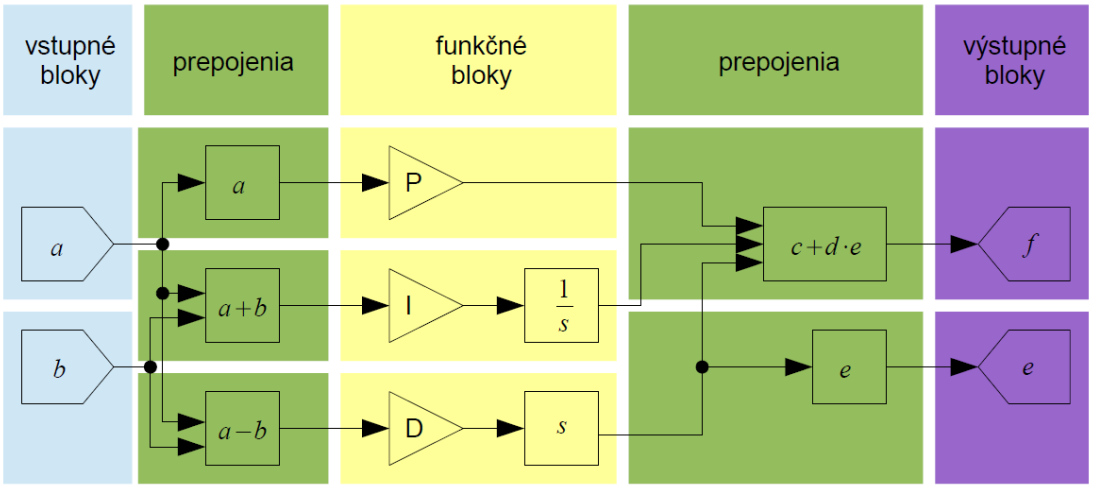
Spolu je teda telo tvorené 5 vektormi ($\{\{B_T\}, \{B_G\}, \{C_I\}, \{C_O\}, \{C_T\}\}$).

Všetky vstupné bloky, výstupné bloky a funkčné bloky sú indexované, aby bolo možné jednoznačne definovať prepojenie. Indexácia prebieha automaticky.

- vstupné bloky – sú indexované od 1 po I
- funkčné bloky – sú indexované od $1+I$ po $B+I$

- výstupné bloky – sú indexované od *1+I+B* po *O+I+B*

Spolu všetky tieto prvky tvoria reťazec s dĺžkou *D=4+B+B+C+C+C*, ktorý je možné graficky znázorniť (Obr. 4.3.1).



Obrázok 4.3.1: Príklad jedinca (2 vstupné bloky, 2 výstupné bloky, 3 funkčné bloky, 5 prepojení)

Príklad č. 1

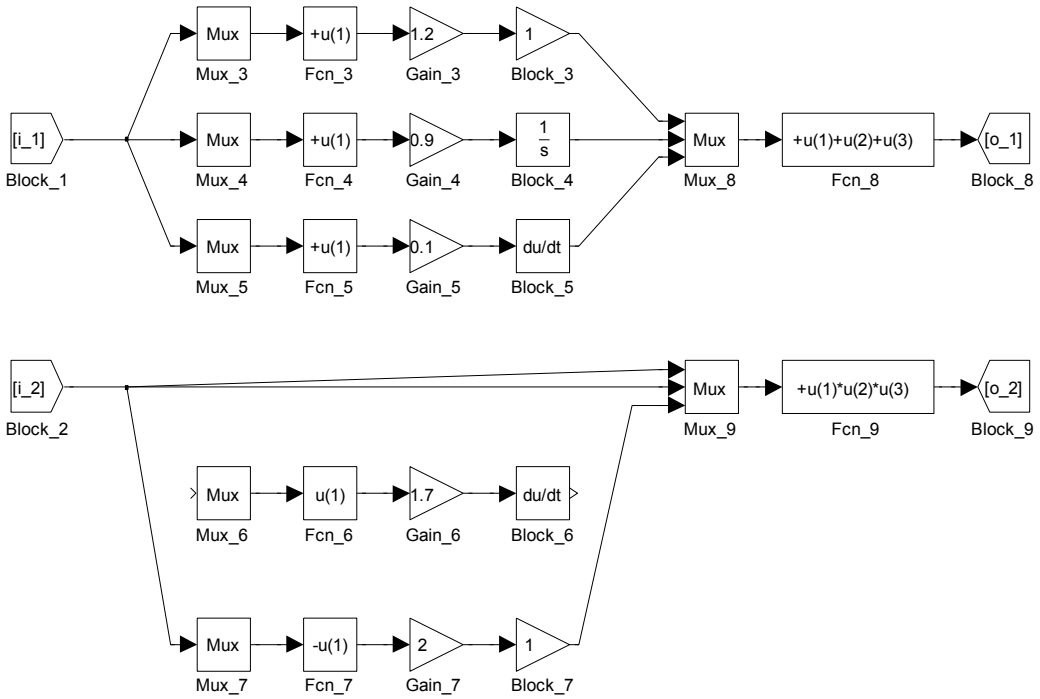
Uveďme si jednoduchý príklad reprezentácie jedinca z lineárneho reťazca v prostredí Matlab/Simulink. Uvažujme ako príklad regulátor, ktorý má 2 vstupy , 2 výstupy, 5 funkčných blokov a 10 prepojení, ako sme už spomenuli, tieto základné dáta sú opísané v hlavičke. Potom sa následne v tele nachádzajú dva vektory s parametrami, ktoré určujú funkčné bloky a ďalšie 3 vektory, ktoré určujú spojenia.

Reťazec: *{{2 2 5 10}{{1 2 3 3 1}{1,2 0,9 0,1 1,7 2,0}{1 1 1 3 4 5 2 2 2 7}{3 4 5 8 8 8 9 9 7 9}{1 1 1 1 1 1 1 3 2 3}}}* (Obr. 4.3.2)

Pre lepšiu prehľadnosť si reťazec zapíšme do pre čitateľa prehľadnejšej podoby:

- *{2 2 5 10}*: hlavička
- *{1 2 3 3 1}*, *{1,2 0,9 0,1 1,7 2,0}*: časť tela, parametre funkčných blokov

- *{1 1 1 3 4 5 2 2 2 7}, {3 4 5 8 8 8 9 9 7 9}, {1 1 1 1 1 1 1 3 2 3}*: časť tela, parametre prepojení



Obrázok 4.3.2: Reprezentácia jedinca pre príklad č. 1

Celá schéma vyzerá byť rozdelená na dve časti, nie je to náhoda. Je možné vytvárať aj dva samostatne fungujúce regulátory pre množinu vstupov. Prirodzene o tejto konfigurácií vo väčšine prípadoch rozhodne evolučný proces, ale zaujímavosťou je, že je možné vziať z reálneho systému konfiguráciu, vytvoriť schému a pokúsiť sa evolučným procesom nájsť možné vylepšenia.

Rozoberme si najprv prvú časť schémy. Zo vstupného bloku č. 1 vychádzajú tri prepojenia, pričom každé prepojenie vstupuje do samostatného funkčného bloku s operáciou súčtu. Vzhľadom k faktu, že žiadne iné prepojenie nevstupuje do týchto funkčných blokov, táto operácia nemá žiaden vplyv na dáta prechádzajúce cez tieto prepojenia. Tieto dáta, ktoré prechádzajú k samotnému jadru funkčného bloku, kde sa následne prenásobia zosilnením, čo má pri hodnotách blížiacich sa k nule utlmujúci charakter a pokračujú k samotnej operácií bloku. Pre blok č. 3 je touto operáciou jednotkové zosilnenie, v jednoduchosti by sa dalo aj povedať, že táto operácia nijak nepozmeňuje zosilnené prípadne zoslabené dáta. Z funkčných blokov č. 3, 4,

5 vychádzajú ďalšie tri prepojenia, ktoré vchádzajú do výstupného bloku č. 8 s operáciami súčtu, čo má za následok že dáta z blokov č. 3, 4, 5 sa navzájom sčítajú a prechádzajú do výstupného bloku č 8.

V druhej časti schémy sa nachádza blok č. 6, do ktorého nevchádza prepojenie a ani z neho žiadne prepojenie nevyúsťuje, teda tento funkčný prvok nemá žiaden vplyv na výsledok, takýto prípad prirodzene v evolučnom procese nastáva a dalo by sa povedať, že sa jedná o časť génov, ktoré nemajú žiadny vplyv na jedinca. Vstupný blok č 2 je jediným zdrojom vstupných informácií, z ktorého vychádzajú tri spojenia. Dve spojenia vchádzajú priamo do výstupného bloku s operáciou násobenia, čo má za následok vytvorenie druhej mocniny nad vstupnými dátami. Jedno prepojenie vchádza do bloku 7 s operáciou rozdielu. Dáta v tomto vstupe sa tým pádom otáčajú na opačné, následne sa prenásobujú zosilnením a keďže je operáciou tohto bloku jednotkové zosilnenie vchádzajú viac nezmenené do bloku č. 9 prepojením s operáciou násobenia. To má za následok, že sa druhá mocnina vstupu zmení na tretiu mocninu a celý výsledok sa ešte prenásobí zosilnením.

Príklad č. 2

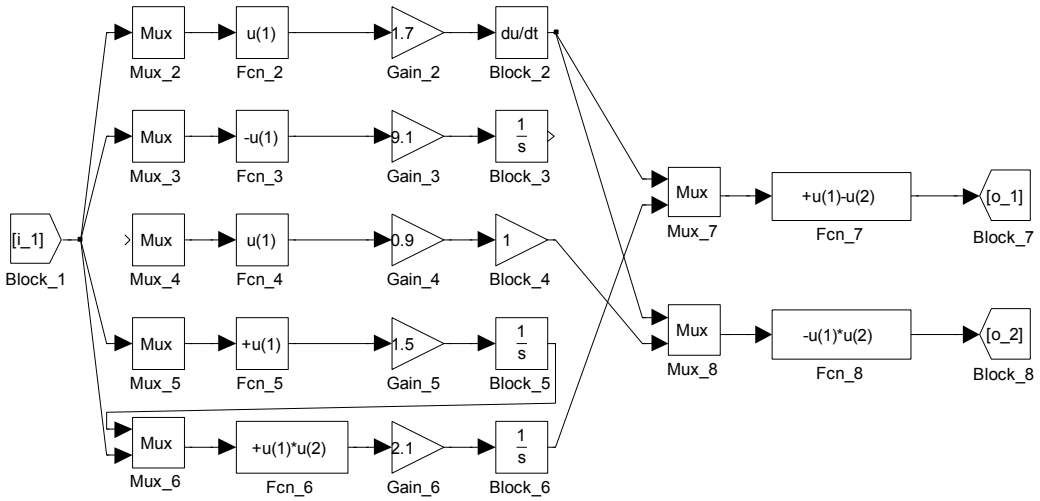
Pre lepšie priblíženie niektorých situácií, ktoré sa môžu objaviť, si uveďme ešte aj nasledovný príklad. Tu sa jedná o schému, ktorá má 1 vstupný blok, 2 výstupné bloky, 5 funkčných blokov a 10 prepojení.

Reťazec: *{{1 2 5 10}{{3 2 1 2 2}{1,7 9,1 0,9 1,5 2,1}{1 1 0 2 2 4 1 5 1 6}{2 3 4 7 8 8 5 6 6 7}{3 2 3 1 2 3 1 1 3 2}}}* (Obr. 4.3.3)

Po rozpísaní:

- *{1 2 5 10}*: hlavička
- *{3 2 1 2 2}*, *{1,7 9,1 0,9 1,5 2,1}*: časť tela, parametre funkčných blokov

- *{1 1 0 2 2 4 1 5 1 6}*, *{2 3 4 7 8 8 5 6 6 7}*, *{3 2 3 1 2 3 1 1 3 2}*: časť tela, parametre prepojení



Obrázok 4.3.3: Reprezentácia jedinca pre príklad č. 2

V úvode uvádzame, že je prepojení celkovo 10 no na Obr. 4.3.3 je vidieť iba 9. Dôvodom je to, že sme sa snažili poskytnúť evolučnému procesu jednoduchú schopnosť nezapojiť všetky prepojenia. Táto časť je pomerne dôležitá kvôli faktu, že vo väčšine prípadov neočakávame, že všetky prepojenia budú aktívne, lebo by nemuseli mať opodstatnenosť. Túto funkcionalitu je možné vidieť pri prepojení č. 3. Index vychádzajúceho bloku je 0. Jedná sa o fiktívny index, ktorý len napovedá, že toto prepojenie, bez ohľadu na jeho ďalšiu konfiguráciu, nemá byť v schéme zahrnuté.

Prepojenia môžu výstup z bloku priviesť na vstupy viacerých blokov, čo je možné vidieť medzi blokom č.2 a blokmi č. 7, 8. Ale tiež je možné výstupy viacerých blokov spájať do vstupu jedného bloku ako je vidieť medzi blokmi č. 2, 4 a blokom č. 8.

Funkčné bloky nemusia byť vždy pripojené na obidvoch koncoch k inému bloku, ako napríklad bloky č. 3, 4. Na prvý pohľad by sa mohlo zdať, že je situácia z pohľadu príspevku do systému skoro rovnaká. No v skutočnosti to nie je celkom tak. Blok č. 3 nemá žiaden výstup, teda tento blok sa nijak nepodieľa na celkovom výsledku, keďže neexistuje z neho žiadne prepojenie na výstup. Blok č. 4 nemá síce žiaden vstup, ale tento stav je pokladaný za prípad, ako by tam neustále prichádzal nulový signál. Čo má v kombinácií

s operáciou násobenia v prepojení medzi blokom č. 4 a blokom č. 8 za následok, že celý výstup z bloku dva je násobený nulovými hodnotami a teda v konečnom dôsledku bude na výstup prechádzať iba nulový signál.

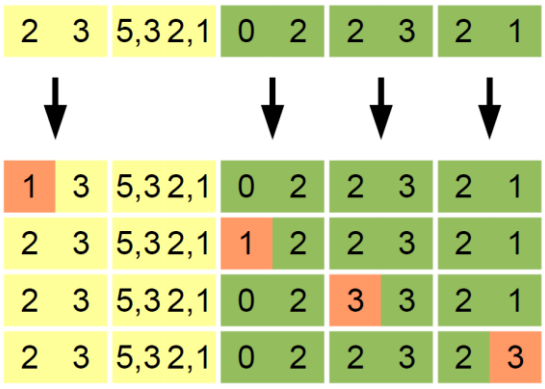
Ako je vidieť funkčné bloky môžu byť prepojené aj s inými funkčnými blokmi čo je bližšie ilustrované medzi blokom č. 5 a blokom č. 6, kde je vďaka operáciám prepojení realizované násobenie signálu medzi výstupmi z blokov č. 1, 5.

4.4 Evolučné operácie

Mutácie

Mutácia je operácia nad jedným jedincom, kedy dochádza k náhodnej zmene obsahu (konštanta, typ operácie) alebo ku náhodnej zmene štruktúry (typ jednotky, prepojenie). Rozlišujeme nasledovné operácie mutácie:

- **mutácia štrukturálnych parametrov** – pri štrukturálnych parametroch ako sú typy blokov, typy operácií a indexy blokov prepojení dochádza k náhodnej zmene daného parametra na iný, bez ohľadu na jeho predchádzajúcu hodnotu. Vektory parametrov, ktoré môžu byť ovplyvnené týmto typom mutácie sú: B_T, C_I, C_O, C_T (Obr. 4.4.1).

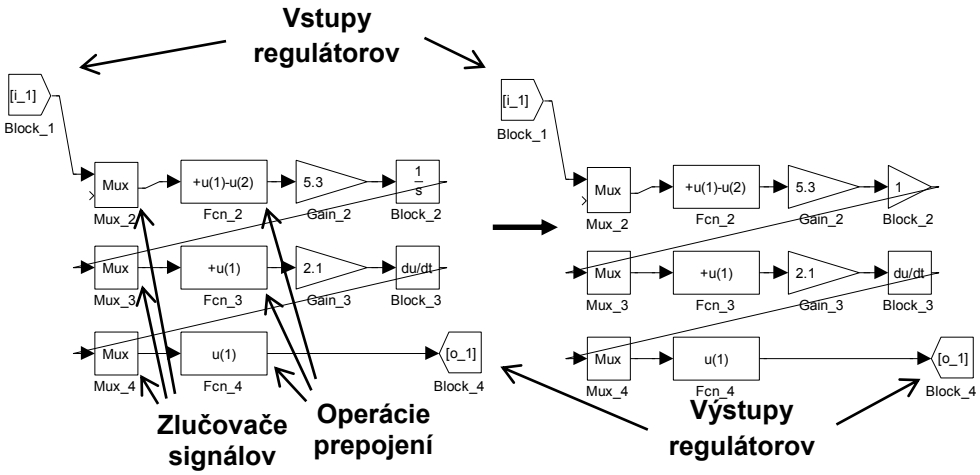


Obrázok 4.4.1: Príklady mutácií štrukturálnych parametrov, mutácia je zvýraznená červenou farbou

Príklad č. 1 (Obr. 4.4.2):

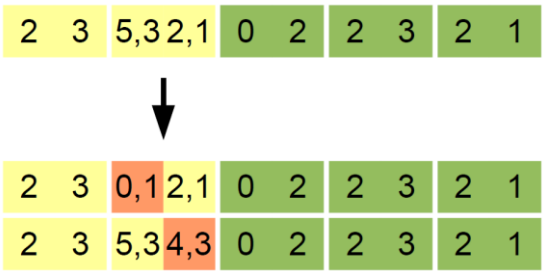
Telo: *{2 3}{5,3 2,1}{1 0 2 3}{2 2 3 4}{1 2 1 3}* mutujeme.

Výsledné telo: *{1 3}{5,3 2,1}{1 0 2 3}{2 2 3 4}{1 2 1 3}*



Obrázok 4.4.2: Mutácia štrukturálnych parametrov pre príklad č. 1

- **mutácia parametrov funkčných blokov** – pre parametre blokov využívame dva typy mutácie: aditívnu - kedy náhodne zväčšíme vybranú hodnotu o hodnotu zo vo zvoleného rozsahu, čo slúži najmä na jemnejšie ladenie parametrov a všeobecnú mutáciu - kedy náhodne generujeme hodnotu parametra v celom definovanom rozsahu. Vektor parametrov, ktorý môže byť ovplyvnený týmto typom mutácie je iba: BG (Obr. 4.4.3).

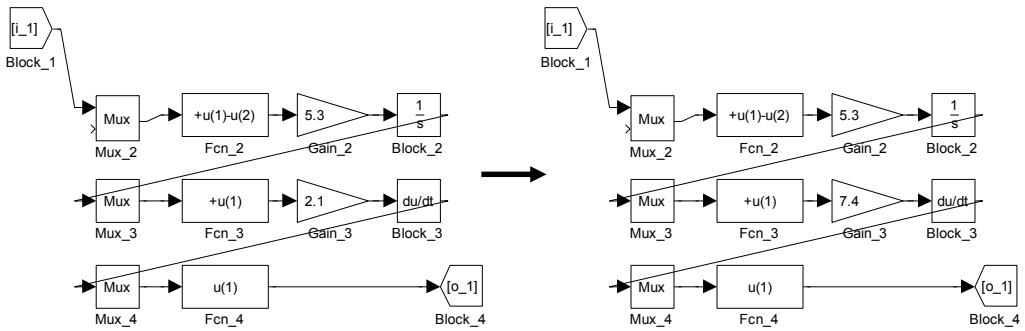


Obrázok 4.4.3: Príklady mutácií parametrov funkčných blokov, mutácia je zvýraznená červenou farbou

Príklad č. 2 (Obr. 4.4.4):

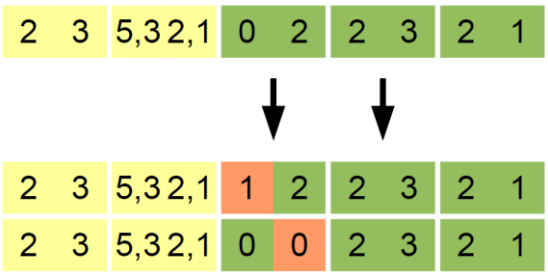
Telo: *{2 3}{5,3 2,1}{1 0 2 3}{2 2 3 4}{1 2 1 3}* mutujeme.

Výsledné telo: *{2 3}{5,3 4,3}{1 0 2 3}{2 2 3 4}{1 2 1 3}*



Obrázok 4.4.4: Mutácia parametrov funkčných blokov pre príklad č. 2

- **náhodné pridanie alebo vynechanie prepojenia** – jedná sa o taký typ mutácie, ktorá náhodne zmení existujúce prepojenia medzi blokmi. Dôležitá vlastnosť je tá, že metóda vyhodnocuje koľko prepojení je práve v jedincovi aktívnych a nevykoná nič, ak požadujeme náhodné pripojenie a všetky prepojenia sú zapojené alebo ak požadujeme náhodné odpojenie a všetky prepojenia sú odpojené. Tiež je možné nastavovať presný počet odpojení alebo pripojení, ktoré má metóda realizovať. Vektory parametrov, ktoré môžu byť ovplyvnené týmto typom mutácie sú: C_I, C_O (Obr. 4.4.5).

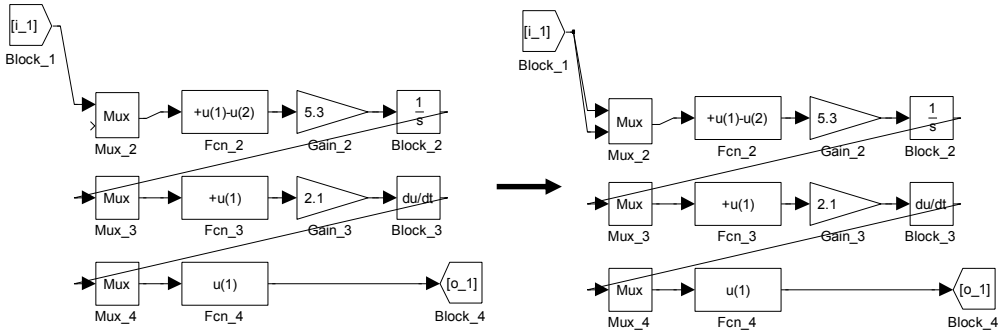


Obrázok 4.4.5: Príklady zapájania alebo rozpájania prepojení

Príklad č. 3 (Obr. 4.4.6):

Telo: *{2 3}{5,3 2,1}{1 0 2 3}{2 2 3 4}{1 2 1 3}* mutujeme.

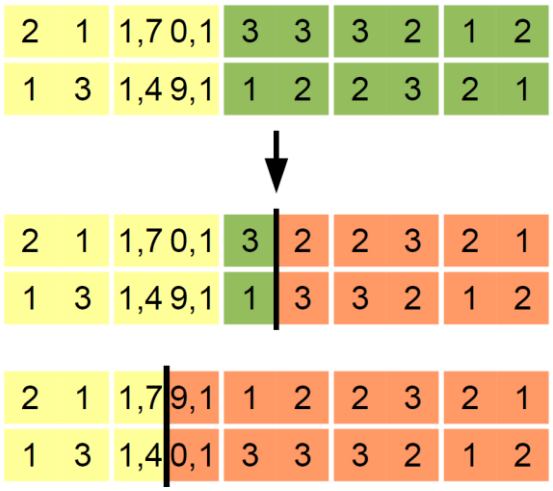
Výsledné telo: *{2 3}{5,3 2,1}{1 1 2 3}{2 2 3 4}{1 2 1 3}*



Obrázok 4.4.6: Mutácia zapájania pre príklad č. 3

Kríženia

Vzhľadom k faktu, že nevyužívame variabilnú dĺžku reťazca sa používa štandardné kríženie, ktoré poznáme z genetických algoritmov. Využívame kríženie, kde si dvaja jedinci náhodne vymenia náhodne vybrané časti svojich genómov (Obr. 4.4.7).



Obrázok 4.4.7: Príklady kríženia dvoch jedincov

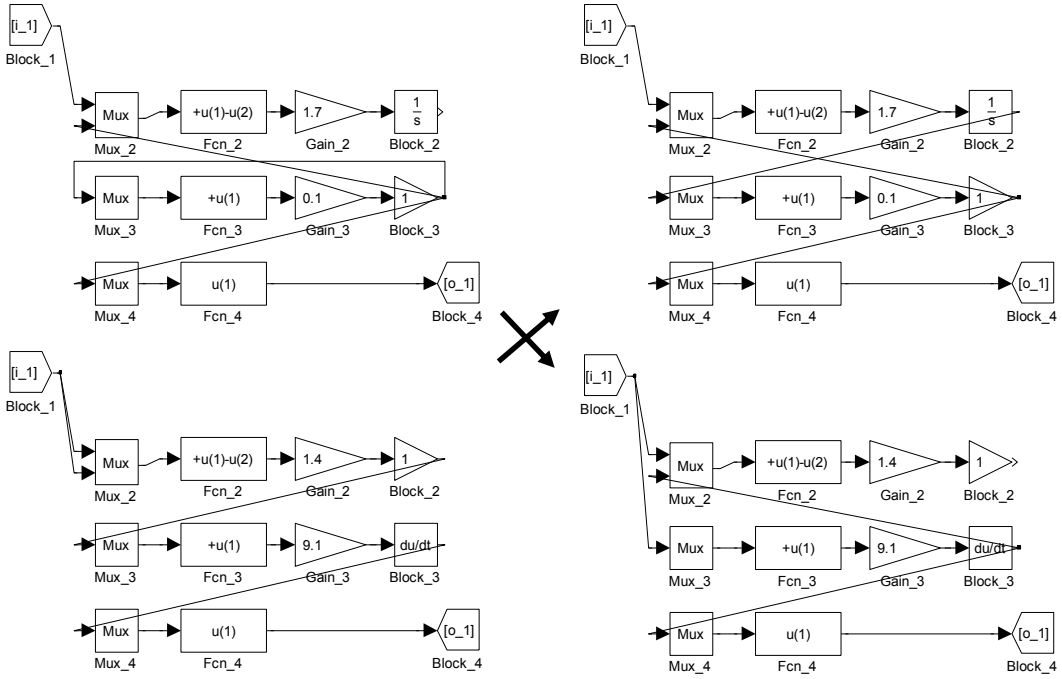
Príklad č. 4 (Obr. 4.4.8):

Telo a: *{2 1}{1,7 0,1}{1 3 3 3}{2 3 2 4}{1 1 2 3}* sa skríži s reťazcom b.

Telo b: *{1 3}{1,4 9,1}{1 1 2 3}{2 2 3 4}{1 2 1 3}* sa skríži s reťazcom a.

Výsledné telo a: *{2 1}{1,7 0,1}{1 3 2 3}{2 2 3 4}{1 2 1 3}*

Výsledné telo b: *{1 3}{1,4 9,1}{1 1 3 3}{2 3 2 4}{1 1 2 3}*



Obrázok 4.4.8: Kríženie dvoch reťazcov pre príklad č. 4

Inicializácia populácie

Cieľom inicializácie populácie je vygenerovať počiatočnú množinu jedincov, ktorá bude základom evolúcie. Jedince počiatočnej populácie budú obsahovať náhodné štruktúry a hodnoty, ktoré sú spravidla z hľadiska definovaného cieľa nevyhovujúce. V priebehu evolúcie sa tieto začínajú meniť v prospech určených kritérií. V našom prípade sa vygeneruje *B* stavebných blokov a *C* spojení s náhodnými parametrami, ktoré vyhovujú vopred určeným ohraničeniam.

Výber

- **výber najlepších jedincov z populácie (elitarizmus)** – tento výber sa používa v obmedzenej miere, iba z dôvodu, aby bol zabezpečený monotónny nárast úspešnosti populácie (prežitie najúspešnejšieho riešenia). Minimalizácia elitarizmu môže eliminovať príliš veľký selektívny tlak - výber väčšieho počtu (krátkodobo) najlepších jedincov. Vysoký

elitarizmus totiž môže viesť k stagnovaniu (uviaznutiu) algoritmu v lokálnom optime. V našom prípade vyberáme iba približne 3 % jedincov týmto typom výberu.

- **turnajový výber** – Jedná sa o najviac používaný typ výberu, známy z genetických algoritmov [10, 35, 57]. Približne 57 % jedincov vyberáme do novej populácie práve touto metódou.
- **náhodný výber** – jeho používaním sa snažíme zachovať jedincov, ktorí nemusia byť úspešní, ale môžu niesť produktívne gény, ktoré z krátkodobého hľadiska neprinášajú jedincovi vysokú úspešnosť na to, aby uspeli vo vzájomnom porovnaní s aktuálne najúspešnejšími jedincami. Tým sa snažíme znižovať selektívny tlak. V našom prípade sa tento výber používa pre 40 % jedincov.

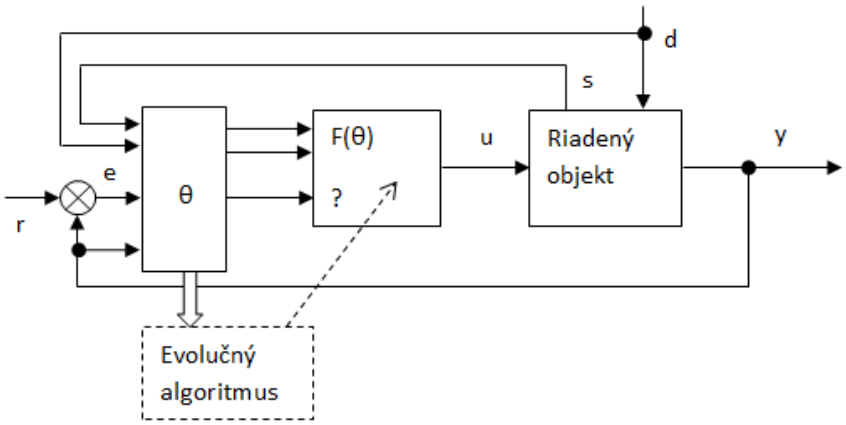
Ukončovacie podmienky algoritmu

Vzhľadom na dlhé výpočtové časy a fakt, že vopred nepoznáme najlepšiu možnú hodnotu kriteriálnej funkcie (fitness), využívame ako ukončovaciu podmienku dosiahnutie vopred určeného počtu generácií (výpočtových cyklov). Toto číslo stanovujeme spravidla experimentálne, v prípade potreby ho predlžujeme, resp. ho určíme s ohľadom na predchádzajúce skúsenosti. Táto metóda je vzhľadom na naše dlhodobé skúsenosti najspoľahlivejšia a najjednoduchšia. Používateľ v priebehu evolúcie posúdi, či mu riešenie už vyhovuje, alebo ešte nie.

Princíp evolúcie riadenia pomocou KGP

Princíp návrhu regulátora, ktorý bol opísaný v predchádzajúcich častiach tejto kapitoly je ilustrovaný na Obr. 4.4.9. Evolučný algoritmus postupne generuje náhodné zmeny v populácii jedincov. Jedince, ktoré sú úspešnejšie, majú väčšiu šancu v evolúcii prežiť. Hodnotenie úspešnosti každého jedinca predstavujú dva kroky. Prvý krok je simulácia riadiaceho procesu (uzavretého regulačného obvodu) vo vhodnom simulačnom nástroji (v našom prípade Simulink). Druhý krok je vyčíslenie zvolenej kriteriálnej funkcie. Tento proces

sa opakuje, kým riešenie nevyhovuje, prípadne sa vykonáva stanovený počet generácií.



Obrázok 4.4.9: Schematické znázornenie princípu návrhu riadenia evolučným algoritmom. y – riadené (výstupné) veličiny, u – riadiace veličiny, e – regulačné odchýlky, r – žiadané veličiny, s – vnútorné stavy riadeného objektu, d – externé poruchy resp. informácie o prostredí

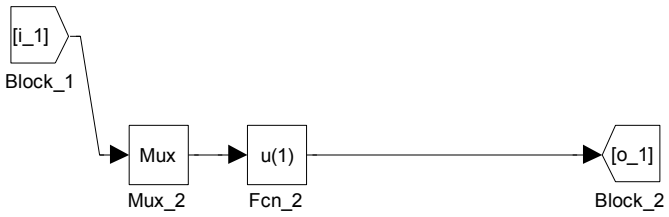
4.5 Ilustračný príklad evolúcie regulátora

Pre ilustráciu hore uvedených genetických operácií si uveďme jednoduchý príklad evolúcie regulátora s jedným vstupom a jedným výstupom.

Priebeh evolúcie bol zachytený iba v štyroch krokoch: po 1., po 10., po 50. a po 100. Generácii výpočtu.

Regulátor v 1. generácii (Obr. 4.5.1):

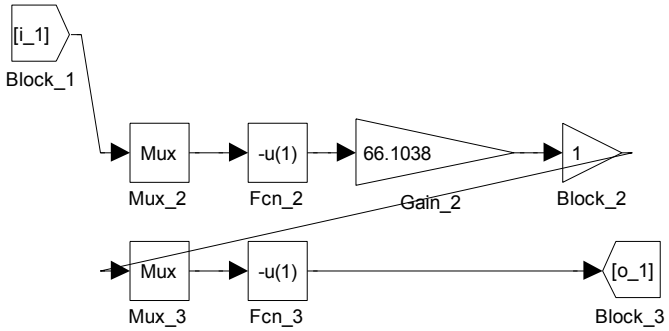
{{1 1 0 1}{}{}{}{1}{2}{3}}}



Obrázok 4.5.1: Regulátor v 1. generácii

Regulátor v 10. generácii (Obr. 4.5.2):

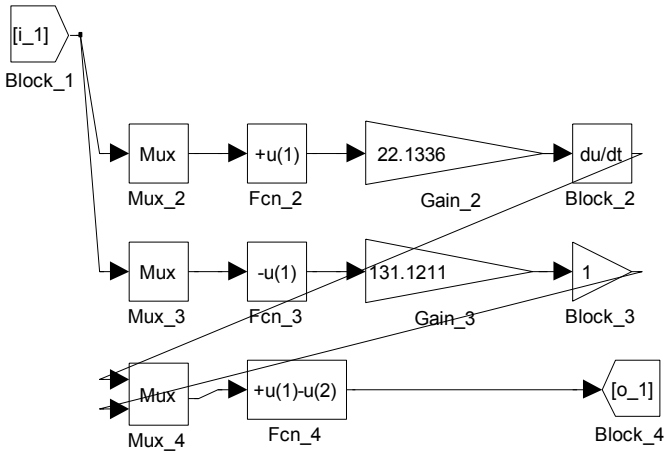
{{1 1 1 2}{{1}{66,1038}{1 2}{2 3}{2 2}}}



Obrázok 4.5.2: Regulátor v 10. generácii

Regulátor v 50. generácii (Obr. 4.5.3):

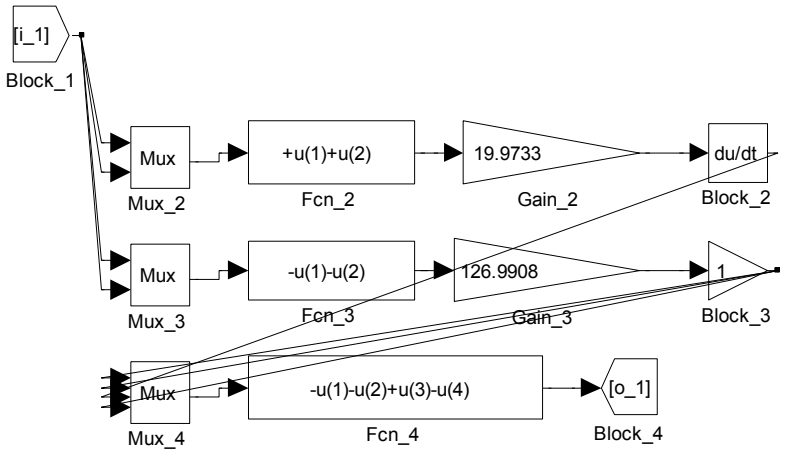
{{1 1 2 4}{{3 1}{22,1336 131,1211}{1 1 2 3}{3 2 4 4}{2 1 1 2}}}



Obrázok 4.5.3: Regulátor v 50. generácii

Regulátor v 100. generácii (Obr. 4.5.4):

{{1 1 2 8}{{3 1}{19,9733 126,9908}{1 3 1 3 1 1 2 3}{3 4 2 4 3 2 4 4}{2 2 1 2 2 1 1 2}}}

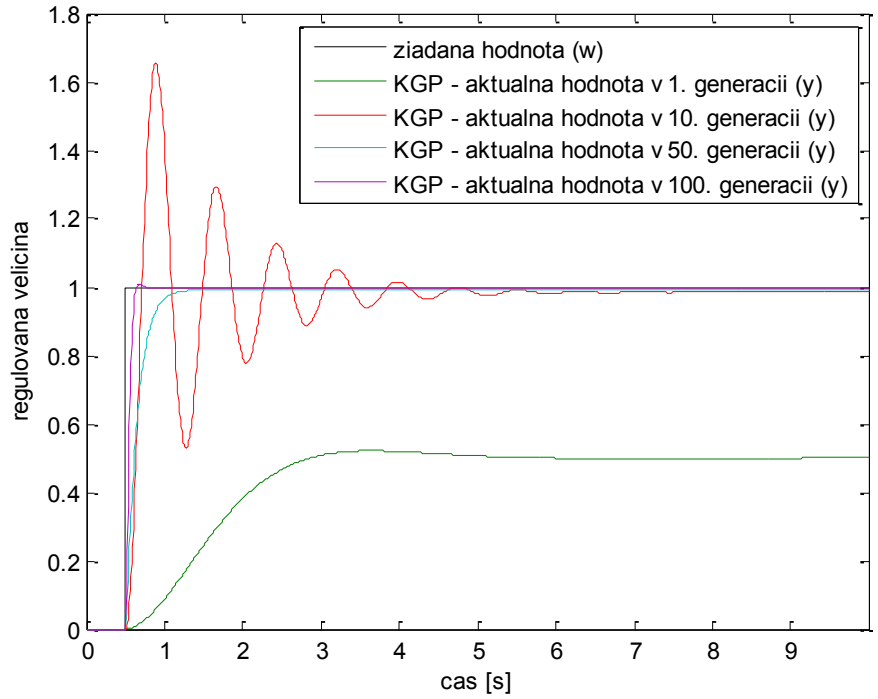


Obrázok 4.5.4: Regulátor v 100. generácii

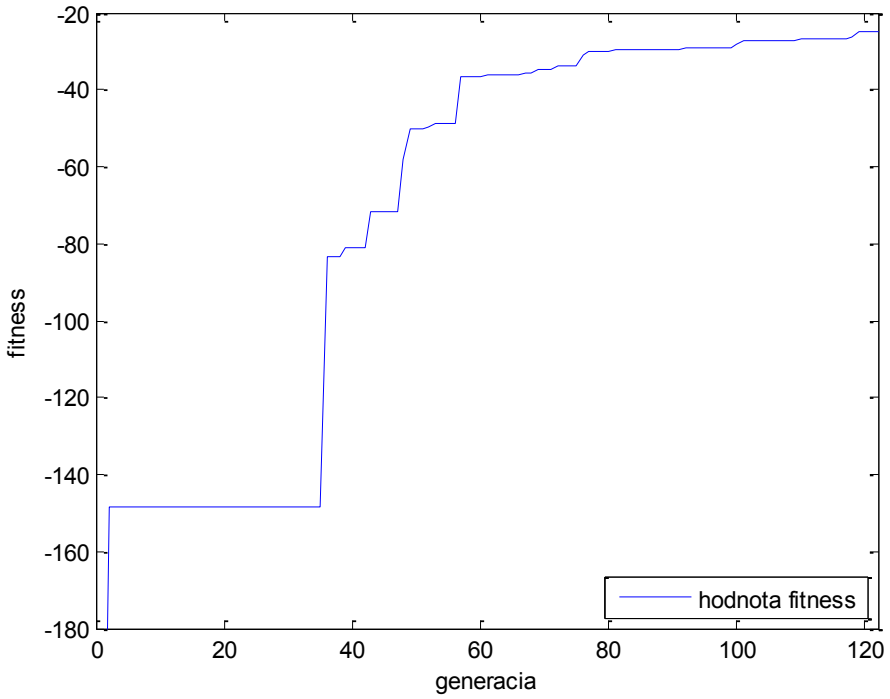
Hodnoty kriteriálnej funkcie (fitness) sa postupne zvyšujú podľa Tab. 4.5.1. V tomto prípade bolo použité integrálne kritérium v tvare (3.2.1). Optimálna hodnota kritéria nie je známa, ale hodnoty fitness v priebehu času musia konvergovať smerom k nule. Výsledky dosiahnuté v experimente sú uvedené v Tab. 4.5.1.

Generácia	Fitness
1.	-1041,2016
10.	-148,3170
50.	-50,0774
100.	-27,9770

Tabuľka 4.5.1: Porovnanie fitness v jednotlivých generáciách



Obrázok 4.5.5: Priebeh regulácie pre regulátory s najvyššou hodnotou fitness získané z populácie vo vybraných generáciách



Obrázok 4.5.6: Vývoj fitness

5 Experimentálne výsledky

V tejto práci navrhnutý a doteraz opísaný prístup návrhu riadenia sme experimentálne overili pri vybraných typoch dynamických systémov. V ďalšom ukážeme výsledky návrhu riadenia nelineárneho dynamického SISO systému, robustifikovaného riadenia, riadenia systému s viacerými vstupmi a výstupmi a riadenia vodnej turbíny.

Pri simuláciách, ktoré sme realizovali, uvádzame ako hlavné výsledky časové priebehy regulovaných veličín, riadiacich veličín, výsledný návrh štruktúry regulátorov a grafy priebehu evolúcie fitness funkcie v závislosti od počtu generácií. Posedne menovaný graf možno považovať za graf priebehu evolúcie návrhu riadenia. Evolučný proces predstavuje minimalizáciu zvolenej kriteriálnej funkcie – kvality regulácie. Pre zosúladenie zo všeobecne zaužívanými postupmi v literatúre, kde sa uvažuje spravidla maximalizácia fitness funkcie (čiže maximalizácia miery úspešnosti) sú ale vo výsledkoch zobrazované hodnoty fitness funkcie maximalizované. To znamená, že fitness funkcia je v našom prípade obrátenou hodnotou minimalizovanej hodnoty kriteriálnej funkcie. Čiže úloha minimalizácie kriteriálnej funkcie (5.0.1),

$$x_{opt}=?;\ f(x_{opt}) = min(f(x)) \qquad \textbf{(5.0.1)}$$

kde x_{opt} je optimum – minimum, $f(x)$ je kriteriálna funkcia, sa prevedie na maximalizačnú úlohu (5.0.2).

$$F(x) = - f(x);\ xopt=?;\ F(xopt) = max(F(x)) \qquad \textbf{(5.0.2)}$$

Teda snažíme sa maximalizovať kriteriálnu funkciu so záporným znamienkom. Z toho dôvodu všetky grafy evolúcie fitness funkcie, ktoré uvádzame v ďalšom narastajú zo záporných hodnôt smerom k nule. Ale nulu nemôžu dosiahnuť, pretože nie je fyzikálne možné, aby kvalita regulácie (spravidla integrálne kritérium regulačnej odchýlky, atď.) dosiahlo nulovú hodnotu.

5.1 Jednoduchý spätnoväzobný SISO regulačný obvod

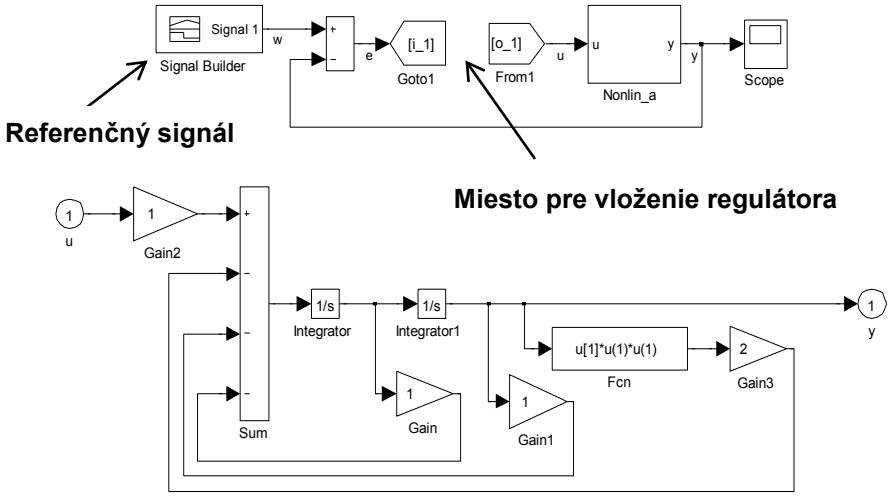
Cieľom je navrhnúť regulátor pre nelineárny dynamický systém(5.1.1),

$$y'' + y' + y + 2y^3 - u = 0 \qquad \textbf{(5.1.1)}$$

ktorý je konštruovaný použitím optimalizačnej metódy na báze kartezián skeho genetického programovania. Bolo použité kritérium (5.1.2), ktoré vychádza z absolútnej regulačnej plochy s malou obmenou podľa vzťahu (3.2.1).

$$J = -\int_0^T (|e(t)| + 0.1 \cdot |e'(t)|)dt \qquad \textbf{(5.1.2)}$$

Na báze algoritmu karteziánskeho genetického programovania (KGP) bola maximalizovaná kriteriálna funkcia (5.1.2). Aby bolo možné toto kritérium vyčísliť, bola uskutočnená simulácia uzavretého regulačného obvodu, ktorého schéma v prostredí Matlab/Simulink je zobrazená na Obr. 5.1.1. V dolnej časti



Obrázok 5.1.1: Schéma regulačného obvodu (hore) s miestom pre vkladanie regulátorov meniacej sa štruktúry a nelineárny riadený objekt (dolu)

obrázka je riadený nelineárny systém a v hornej časti celý uzavretý regulačný obvod. Jeho súčasti sú riadený objekt (označený: *Nonlin_a*) a regulačný obvod. Regulátor, ktorého štruktúra je hľadaná sa vloží pred simuláciou medzi bloky *I_1* (výstup regulačnej odchýlky) a blok *O_1* (riadiaca veličina

regulátora). Medzi tieto bloky vkladá KGP algoritmus meniace sa schémy regulátorov. Na konci evolúcie bol získaný regulátor, ktorého štruktúra je na Obr. 5.1.4. Táto schéma obsahuje aj bloky, ktoré neplnia žiadnu funkciu (nefunkčné časti genómu) a tie boli manuálne odstránené pre lepšiu názornosť na Obr. 5.1.5. Jedná sa označené bloky, ktoré nemajú vstup, resp. výstup a teda ich vplyv na riadenie je nulový.

Poznámka: Podobné nefunkčné časti genómu, ktoré sa počas evolúcie dostali do chromozómov sa vyskytujú aj v živej prírode, napr. aj v genóme človeka.

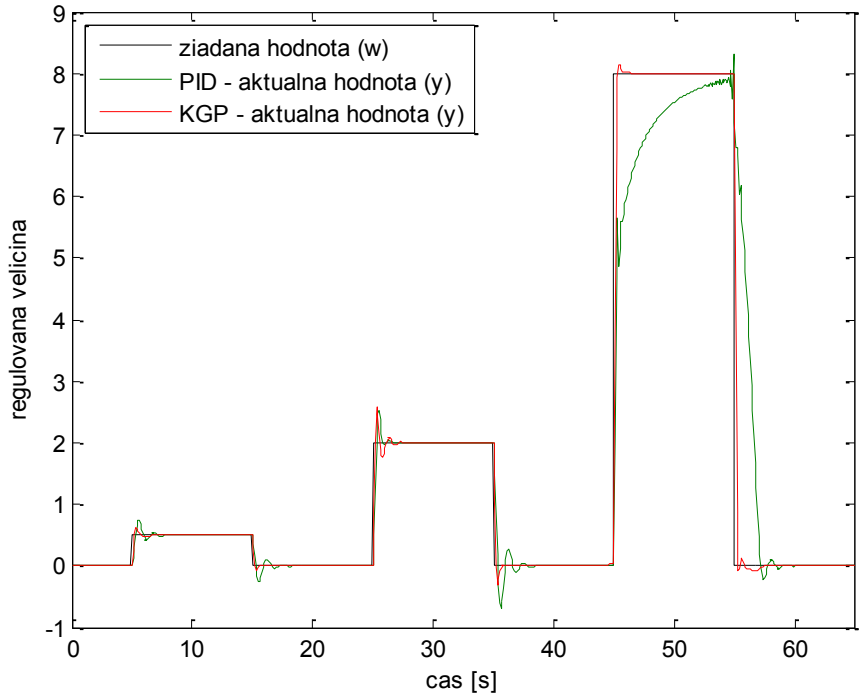
Priebeh regulácie je zobrazený na Obr. 5.1.2. Tu je porovnaný priebeh KGP regulátora s PID regulátorom, ktorého parametre boli navrhnuté pomocou genetického algoritmu [57, 56]. Je zrejmé, že výsledok návrhu s KGP dosahuje evidentne lepšiu kvalitu regulácie. Časové priebehy riadiacej veličín sú porovnané na Obr. 5.1.3. Priebeh evolúcie fitness funkcie je zobrazený na Obr. 5.1.6.

Parameter	Hodnota
P	30,9976
I	117,9831
D	5,5650

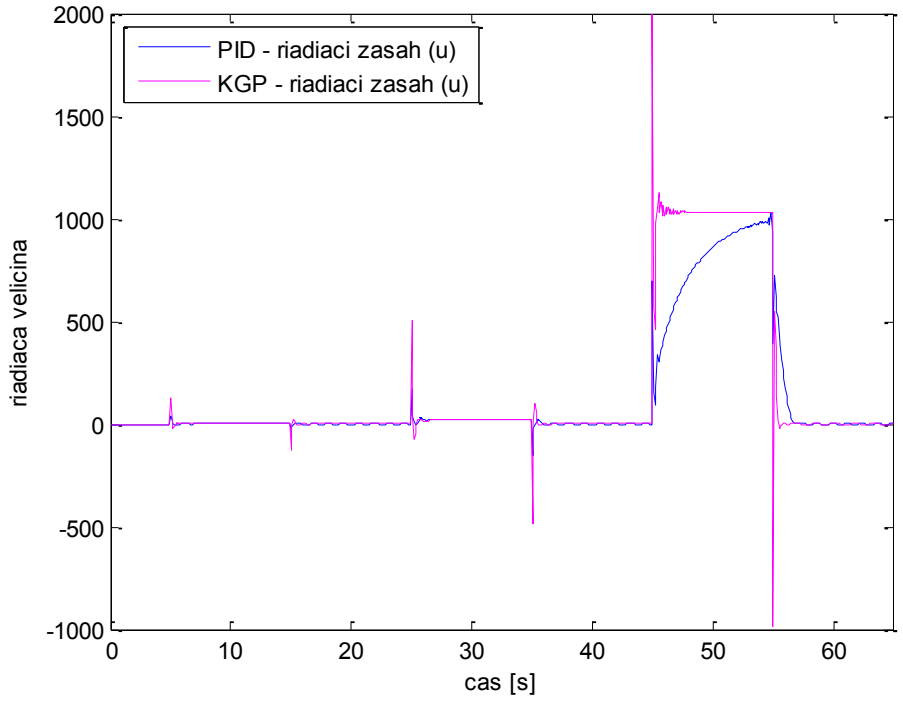
Tabuľka. 5.1.1: Parametre PID regulátora

Typ	Fitness
PID	-249,7698
Karteziánske genetické programovanie	-86,0865

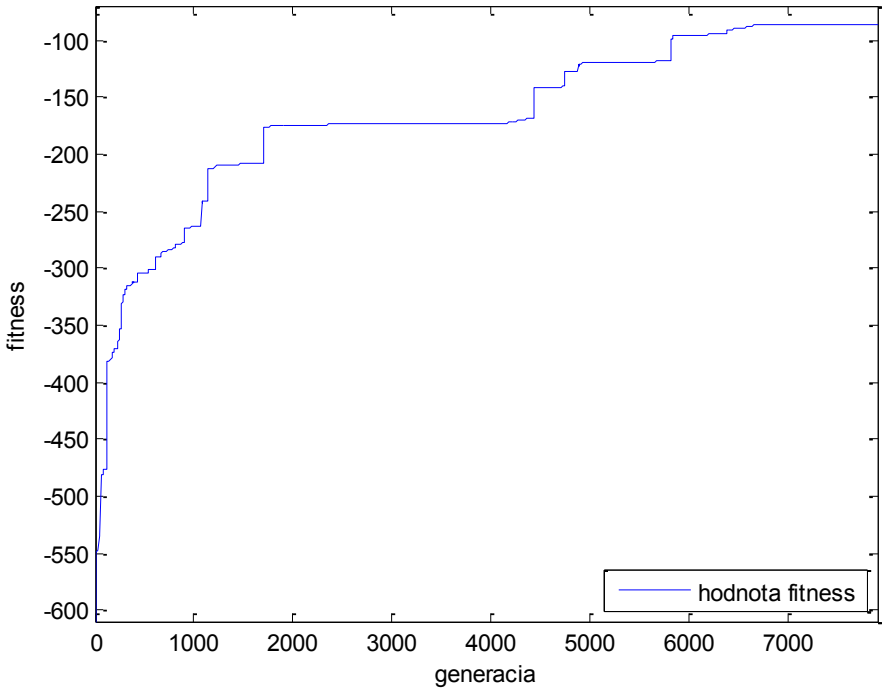
Tabuľka. 5.1.2: Porovnanie fitness jednotlivých metód



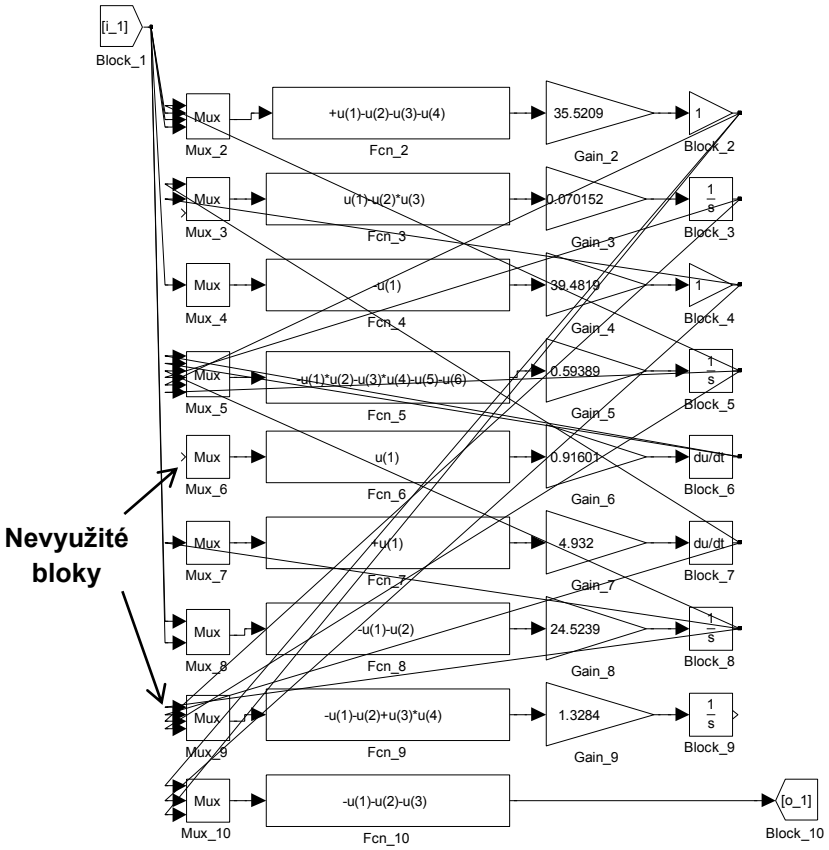
Obrázok 5.1.2: Porovnanie priebehov regulácie nelineárneho systému s PID regulátorom navrhnutým pomocou genetického algoritmu a regulátora navrhnuté pomocou karteziánskeho genetického programovania (KGP)



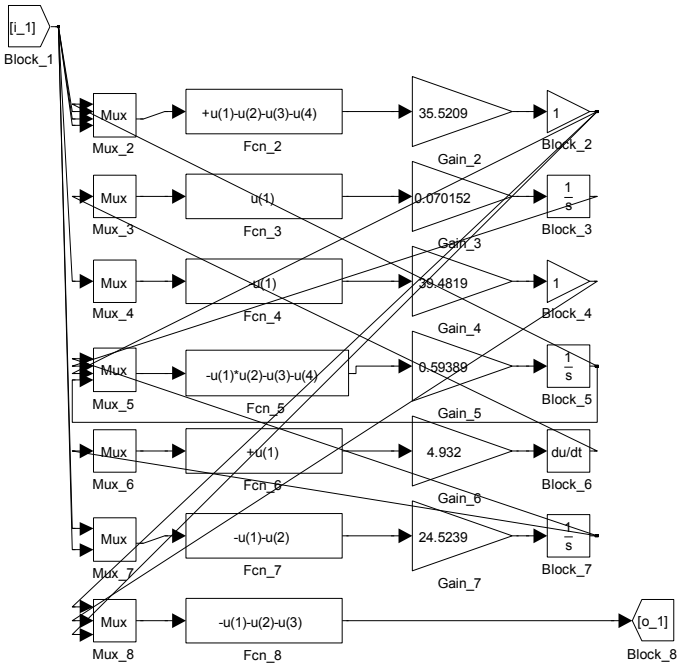
Obrázok 5.1.3: Porovnanie priebehov riadiacich zásahov PID regulátora a regulátora navrhnutého karteziánskym genetickým programovaním (KGP)



Obrázok 5.1.6: Priebeh fitness pre optimalizáciu regulátora navrhnutého pomocou karteziánskeho genetického programovania (KGP)



Obrázok 5.1.4: Bloková schéma regulátora navrhnutého karteziánskym programovaním



Obrázok 5.1.5: Bloková schéma redukovaná o nevyužívané bloky

5.2 Návrh robustifikovaného regulátora

Ďalším typom úlohy, ktorým sa snažíme overiť vlastnosti KGP je zohľadnenie robustnosti. Úlohou je nájsť robustný regulátor, ktorý bude schopný

$$y'' + my' + y + y^3 - u = 0 \qquad \textbf{(5.2.1)}$$

dosahovať dostatočnú kvalitu regulácie vo viacerých pracovných bodoch dynamického systému. Jedná sa tu o minimalizáciu kritéria (3.2.1) pre viac určených pracovných bodov prevádzky systému. Predpokladajme dynamický systém definovaný diferenciálnou rovnicou (5.2.1),

kde parameter *m* môže nadobúdať hodnoty z rozsahu *0.1<m<10.* Nelineárny systém má premenlivé zosilnenie v závislosti od polohy výstupu *y* a tiež jeho dynamika závisí od hodnoty parametra *m*. Zvoľme 4 pracovné body definované hraničnými hodnotami parametrov *y* a *m* podľa Tab. 5.2.1.

Pracovný bod	***y***	***m***
1	0	0,01
2	0	10
3	10	0,01
4	10	10

Tabuľka 5.2.1: Určenie pracovných bodov systému

Vyhodnotenie fitness funkcie tu pozostáva zo 4 simulácii, pre každý pracovný bod a následného vyhodnotenia kriteriálnej funkcie typu (3.2.1 – kapitola robustný návrh) vo forme (5.2.2),

$$J = \sum_{i=1}^{4} J_i \qquad \textbf{(5.2.2)}$$

kde J_i je uvažované v tvare (5.2.3).

$$J_i = -\int_0^T (|e_i(t)| + 0.1 \cdot |e_i'(t)|)dt \qquad \textbf{(5.2.3)}$$

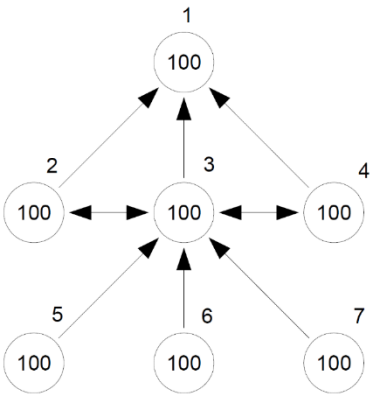
Za účelmi porovnania sme uskutočnili návrh konvenčného regulátora PID pre rovnaké 4 pracovné body použitím rovnakej kriteriálnej funkcie podľa vzťahov (5.2.2) a (5.2.3). Porovnanie priebehov dosiahnutých regulátorom KGP a PID v jednotlivých pracovných bodoch je na Obr. 5.2.2 až Obr. 5.2.9.

Typ	Pracovný bod	Integrál e	integrál de/dt	Fitness spolu
PID	m=0,01, w=0 - 1	5,1585	104,4057	-15,5991
	m=0,01, w=0 - 10	53,7726	1000,9383	-153,8664
	m=10, w=0 - 1	5,9246	104,9296	-16,4176
	m=10, w=0 - 10	53,7691	1000,0000	-153,7691
	celkovo	118,6248	2210,2735	**-339,6522**
Karteziánske genetické programovanie	m=0,01, w=0 - 1	4,0743	101,9745	-14,2717
	m=0,01, w=0 - 10	49,8858	1005,9129	-150,4771
	m=10, w=0 - 1	4,5110	102,1125	-14,7222
	m=10, w=0 - 10	51,4377	1000,8319	-151,5209
	celkovo	109,9088	2210,8317	**-330,9920**

Tabuľka 5.2.2: Porovnanie fitness jednotlivých metód návrhu

Špecifikom evolučných výpočtových metód je veľký počet vyhodnotení účelovej funkcie. V priebehu evolúcie riešenia je potrebné uskutočniť $N=P.G$ vyhodnotení účelovej funkcie, kde P je veľkosť populácie evolučného algoritmu a G je počet generácií výpočtu. Naviac, ak výpočet účelovej funkcie obsahuje výpočtovo náročnú procedúru, akou je určite aj simulácia regulačného obvodu, výpočtová / časová náročnosť návrhu môže byť vysoká. Z toho dôvodu je zmysluplné evolučný algoritmus paralelizovať. To znamená, rozdeliť ho do viacerých čiastočne izolovaných sub-populácií (tzv. ostrovov), pričom na každom ostrove prebieha samostatný výpočet EA. Jednotlivé ostrovy spravidla bežia na vlastných procesoroch/jadrách, ktoré navzájom komunikujú prostredníctvom migračných väzieb [57].

Pre hľadanie robustného regulátora navrhnutého karteziánskym genetickým programovaním využívame paralelný algoritmus na uvedenom princípe. Architektúra nami použitého paralelného evolučného algoritmu (PEA) je na Obr. 5.2.1. Je tu použitých 7 ostrovov, každý so 100 jedincami. Migračné väzby predstavujú šípky. Nami zvolená stratégia migrácií je nasledovná.

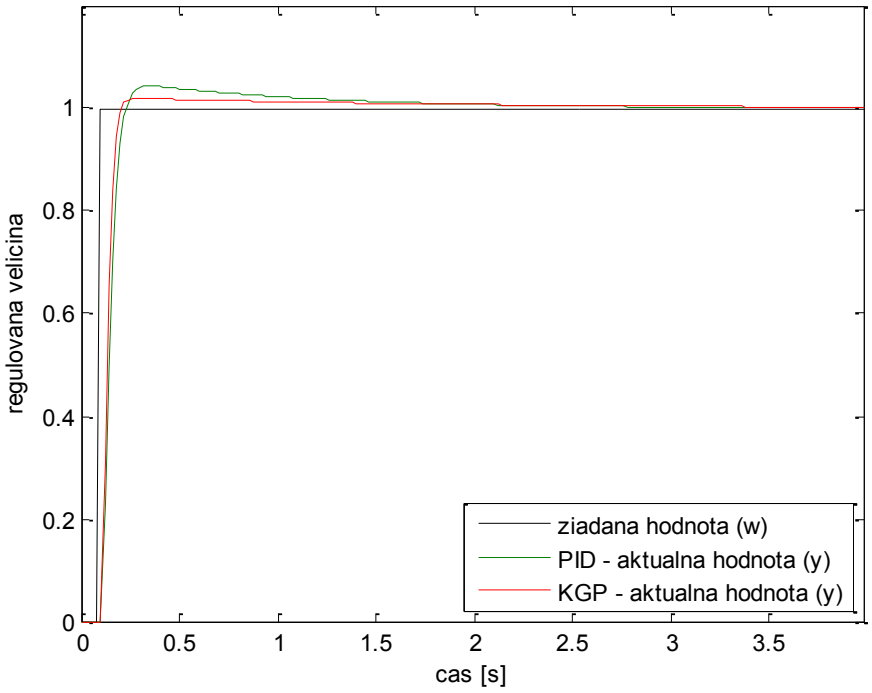


Obrázok 5.2.1: Schéma paralelného algoritmu

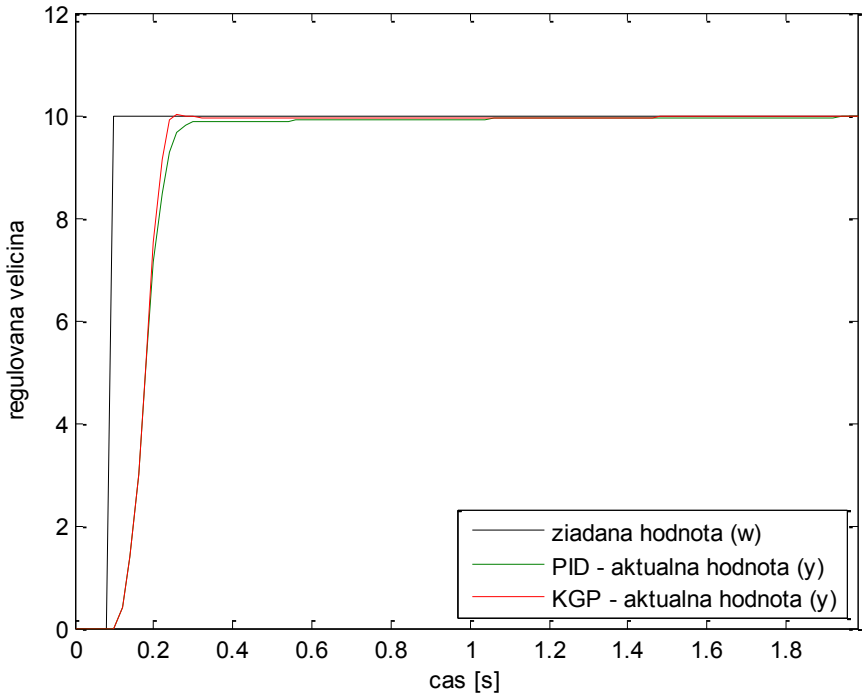
Na začiatku vygenerujeme náhodne všetky gény pre 700 jedincov, ktorých rozdelíme do siedmich skupín (ostrovov). 4 ostrovy (1, 5, 6, 7) budú v prvej skupine a 3 ostrovy (2, 3, 4) v skupine druhej. Migrácia prebieha nasledovne. Z ostrova 2 prebieha migrácia troch najlepších jedincov na ostrov 1 každú 30. 130. 230. .. (n * 100 + 30) generáciu Z ostrova 3 na ostrov 1 sa presunú traja najlepší jedinci každú 60., 160., 260. ... (n * 100 + 60). Z ostrova 4 sa presunú rovnako traja najlepší jedinci na ostrov 1 každú 90., 190., 290. ... (n * 100 + 90). Na ostrov 3 sa každú 50. generáciu presunie najlepší jedinec z ostrova 5, 6 a 7. Tu prichádza zaujímavá zmena. Každú desiatu generáciu prebehne migrácia najlepšieho jedinca z ostrova 2 a 4 na ostrov 3 pričom pri migrácii z ostrova 2 náhodne odoberieme jedno prepojenie a pri migrácií z ostrova 4 dôjde k náhodnému zapojeniu prepojenia. Z ostrova 3 prebieha rovnako každú desiatu generáciu migrácia najlepšieho jedinca na ostrov 2 a 4, pričom na ostrov 2 dôjde k náhodnému pripojeniu prepojenia a na ostrov 4 dôjde k náhodnému odpojeniu prepojenia. Každých 500 generácií prebehne kompletná reinicializácia na ostrovoch 5, 6 a 7.

Výsledky jednotlivých kriteriálnych funkcií sú uvedené v Tab. 5.2.2 V porovnaní s návrhom robustného PID regulátora bolo docielené mierne

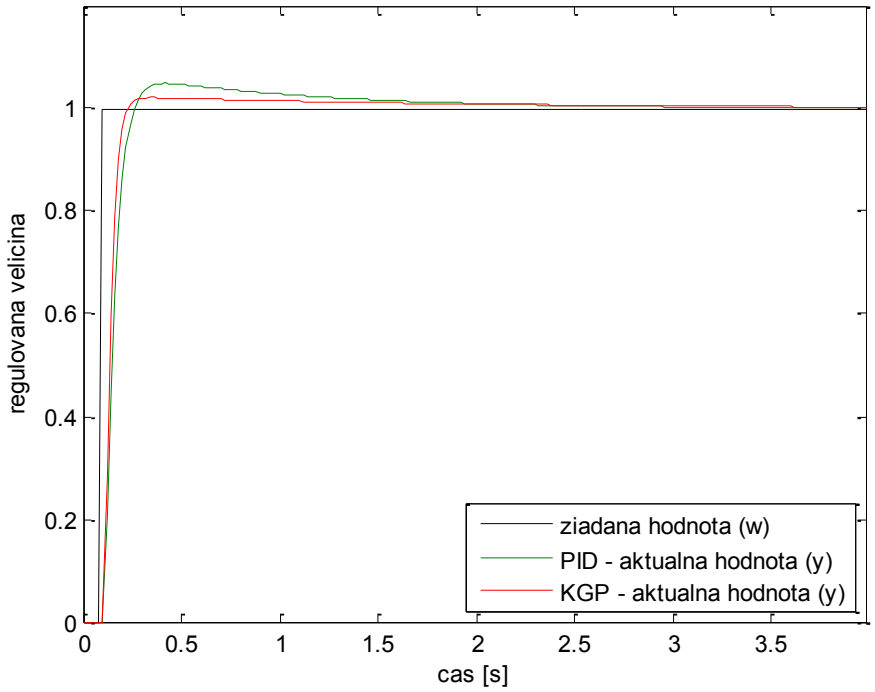
zlepšenie o 2,5%. Obr. 5.2.2 až Obr. 5.2.9) zobrazujú porovnania priebehov regulácie aj riadiacich zásahov PID regulátora navrhnutého pomocou GA verzus návrhy KGP. Obr. 5.2.10 znázorňuje priebehy evolúcie v jednotlivých ostrovoch PGA a evolúciu v centrálnom ostrove (modrý hrubý graf), kde sa sústreďujú všetky najlepšie riešenia celého paralelného genetického algoritmu (PGA). Súčasne je možné na Obr. 5.2.10 pozorovať proces reinicializácie jednotlivých ostrovov, keď fitness funkcia opakovane začína z veľkých záporných hodnôt. Nakoniec výsledná štruktúra robustného regulátora je znázornená na Obr. 5.2.11 a štruktúra očistená od neúčinných blokov je na Obr. 5.2.12.



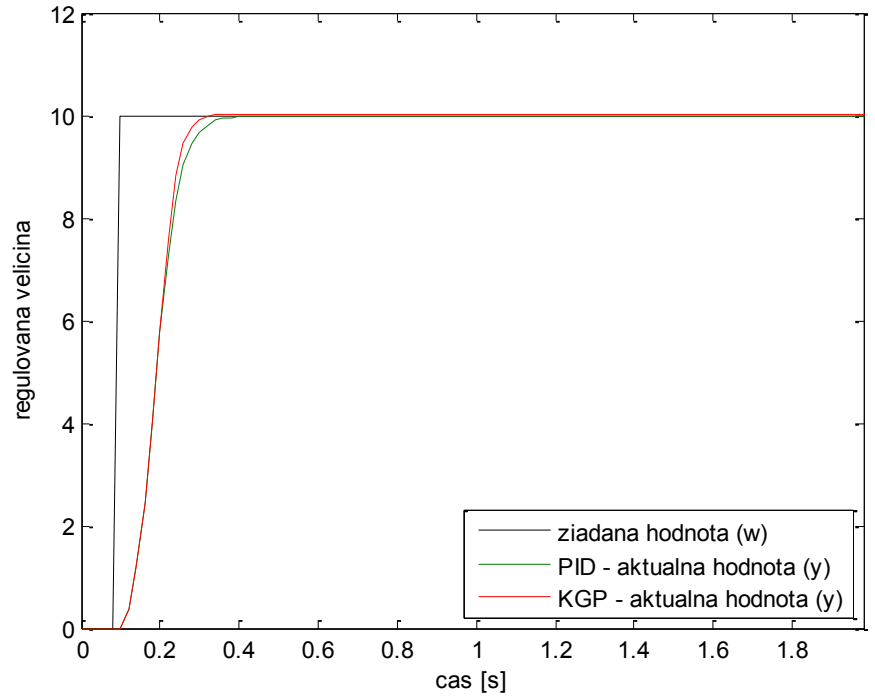
Obrázok 5.2.2: Porovnanie priebehov pre pracovný bod číslo 1 (skok žiadanej hodnoty z 0 na 1 a s parametrom m = 0,01)



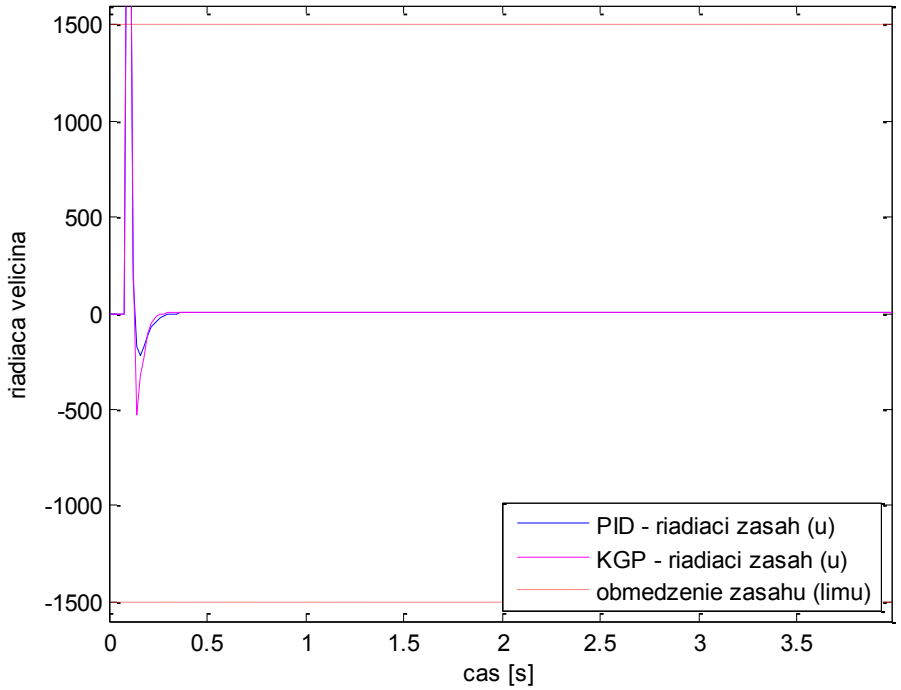
Obrázok 5.2.3: Porovnanie priebehov pre pracovný bod číslo 2 (skok žiadanej hodnoty z 0 na 10 a s parametrom m = 0,01)



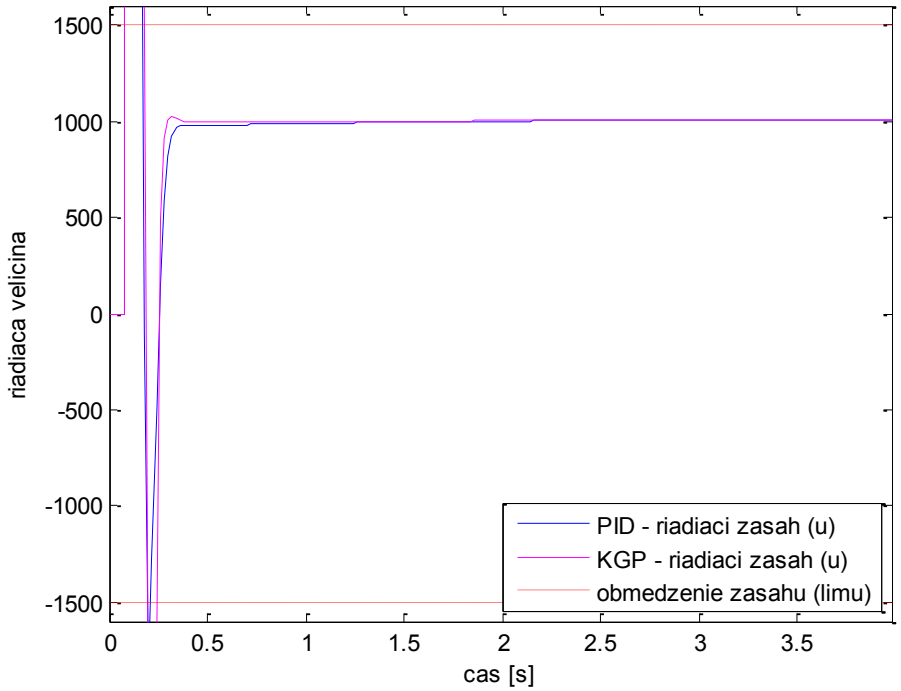
Obrázok 5.2.4: Porovnanie priebehov pre pracovný bod číslo 3 (skok žiadanej hodnoty z 0 na 1 a s parametrom m = 10)



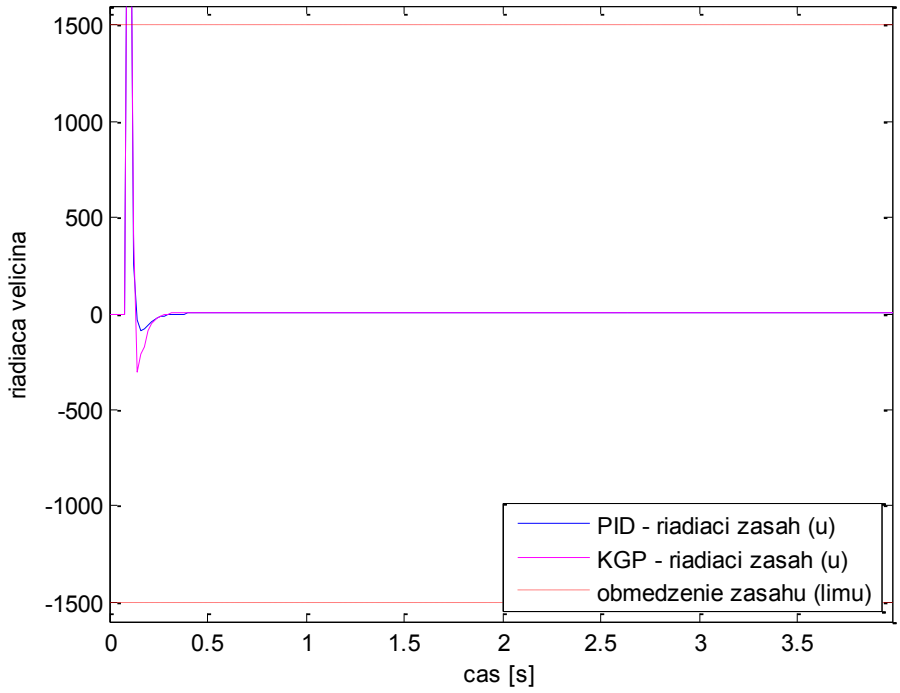
Obrázok 5.2.5: Porovnanie priebehov pre pracovný bod číslo 4 (skok žiadanej hodnoty z 0 na 10 a s parametrom m = 10)



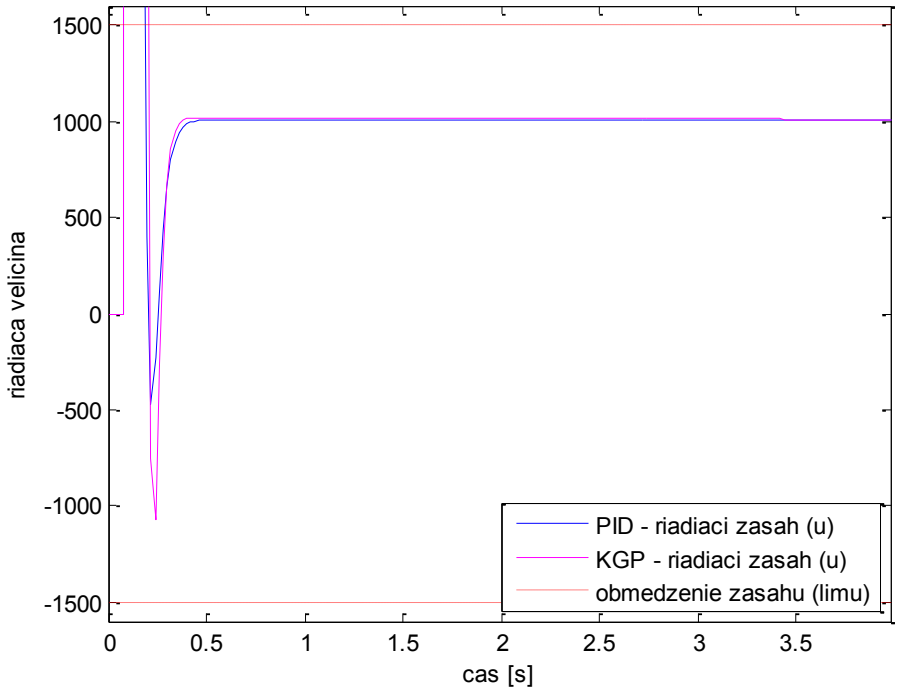
Obrázok 5.2.6: Porovnanie riadiacich zásahov pre pracovný bod číslo 1 (skok žiadanej hodnoty z 0 na 1 a s parametrom m = 0,01)



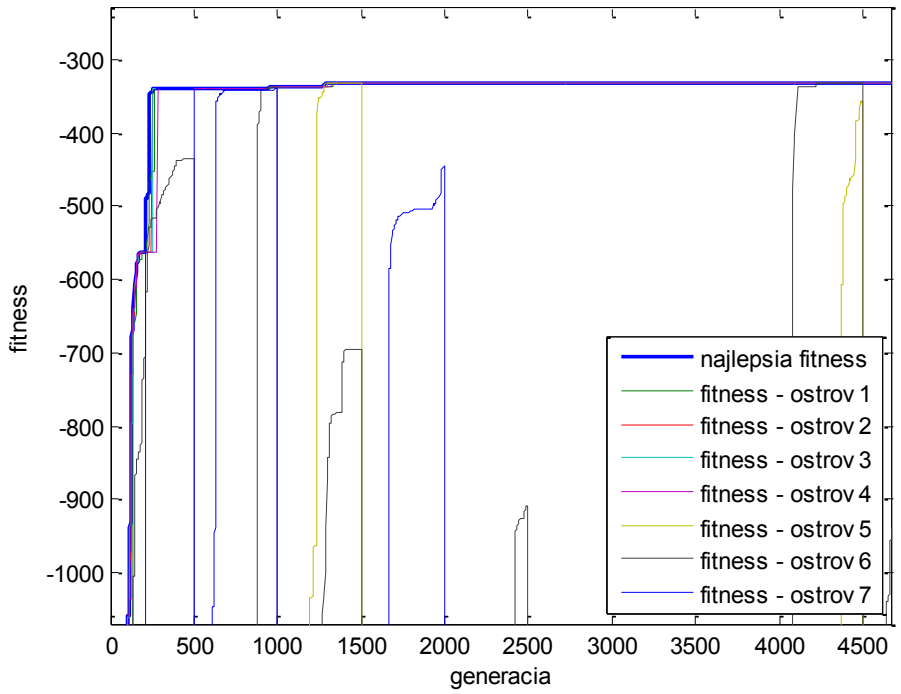
Obrázok 5.2.7: Porovnanie riadiacich zásahov pre pracovný bod číslo 2 (skok žiadanej hodnoty z 0 na 10 a s parametrom m = 0,01)



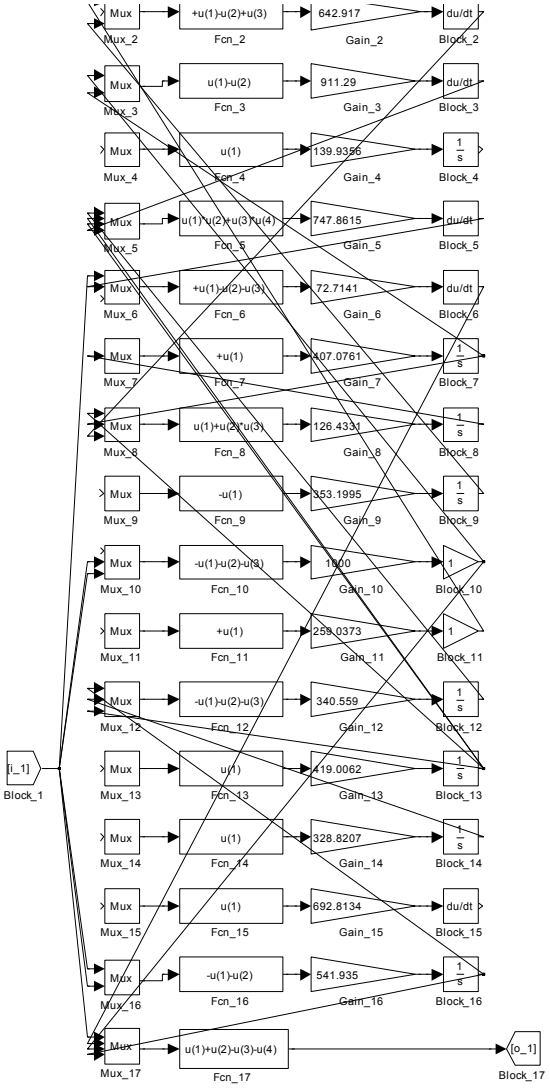
Obrázok 5.2.8: Porovnanie riadiacich zásahov pre pracovný bod číslo 3 (skok žiadanej hodnoty z 0 na 1 a s parametrom m = 10)



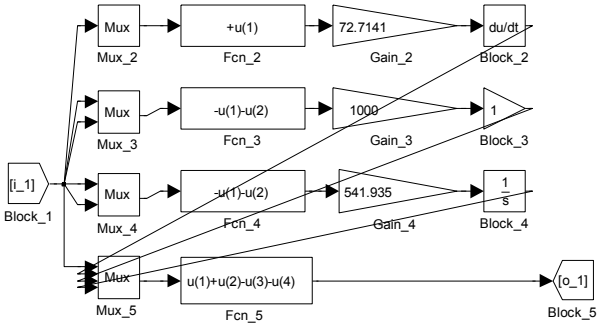
Obrázok 5.2.9: Porovnanie riadiacich zásahov pre pracovný bod číslo 4 (skok žiadanej hodnoty z 0 na 10 a s parametrom m = 10)



Obrázok 5.2.10: Priebeh fitness pre optimalizáciu regulátora navrhnutého pomocou karteziánskeho genetického programovania (KGP)



Obrázok 5.2.11: Bloková schéma regulátora navrhnutého karteziánskym programovaním



Obrázok 5.2.12: Bloková schéma očistená o nevyužívané objekty

5.3 Návrh riadenia vodnej turbíny

Model turbíny a tlakového potrubia je definovaný tromi rovnicami závislými na rýchlosti prúdenia vody v tlakovom potrubí, mechanickým výkonom turbíny a zrýchlením vodného stĺpca. Rýchlosť vody v tlakovom potrubí je (5.3.1),

$$U = k_u G\sqrt{H} \tag{5.3.1}$$

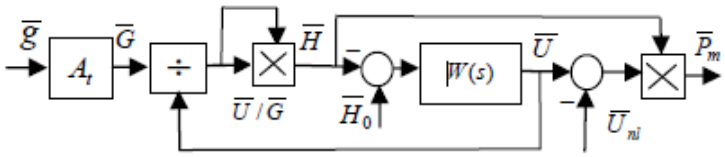
kde U je rýchlosť vody, G je parameter vtoku vody, H je spád hydraulického vtoku, k_u je proporcionálna konštanta. Mechanický výkon turbíny je proporcionálny ku výsledku násobenia tlaku a toku (5.3.2).

$$P_m = k_p HU \tag{5.3.2}$$

Zrýchlenie vodného stĺpca po zmene v spáde turbíny je popísaný Newtnovým druhým pohybovým zákonom a môže byť vyjadrený v podobe (5.3.3),

$$\rho LA\frac{dU}{dt} = -A\rho a_g(H - H_0), \tag{5.3.3}$$

kde H_0 je počiatočná ustálená hodnota hodnoty H, A je plocha prierezu potrubia, L je dĺžka potrubia, ρ je hmotnostná hustota, a_g je gravitačné zrýchlenie. Model elektrárne je zobrazený na Obr. 5.3.1 Rovnice (5.3.1), (5.3.2), (5.3.3) sú v normalizovanej forme, kde je uvažovaný nepružný



Obrázok 5.3.1: Bloková schéma hydraulickej turbíny

charakter vodného stĺpca.

Ak predpokladáme, nepružný charakter vodného stĺpca bude prevodová funkcia W(s) vo forme (5.3.4),

$$W(s) = \frac{1}{T_w s} \tag{5.3.4}$$

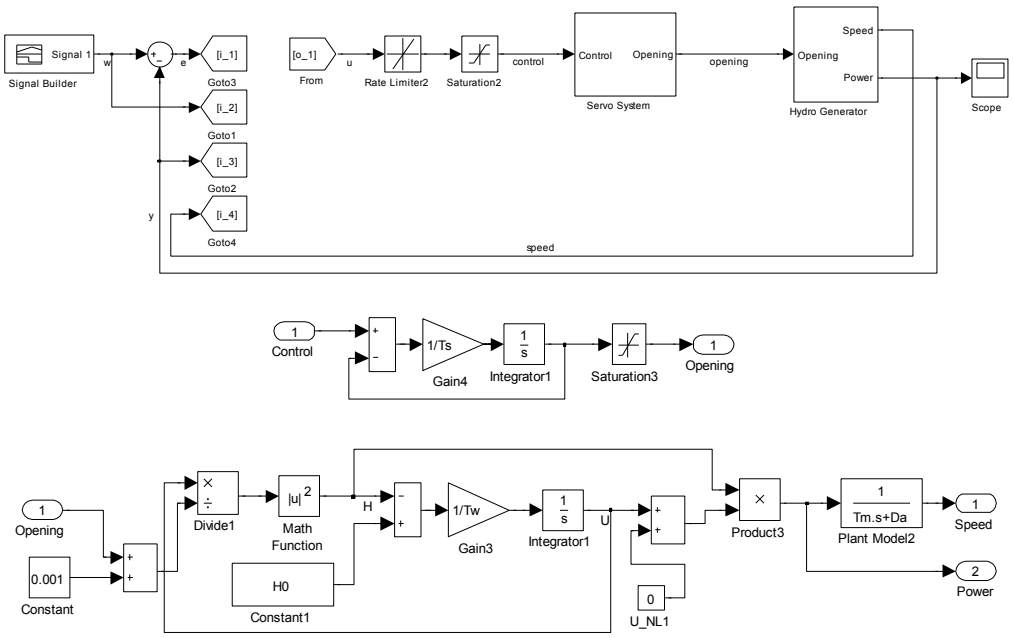
kde T_w je čas nábehu pri záťaži. Má konštantnú hodnotu pre dané potrubie a je stanovená podľa (5.3.5),

$$T_w = \frac{LG_r}{a_g AH_r} \qquad \textbf{(5.3.5)}$$

kde H_r a Q_r sú nominálne hodnoty hydraulického spádu a vtoku a respektíve miery toku. Nasledujúce dáta súvisiace s turbínou, potrubím a generátorom sú nasledujúce: dĺžka potrubia *700 m*, miera hydraulického spádu je *180 m*, prierez potrubia je *10,25 m*$_2$ rýchlosť prietoku vody pri nominálnom zaťažení je *75 m*^{3}*/s*, vtok pri nominálnom zaťažení je *0,96*, vtok bez zaťaženia je *0,04*, výkon turbíny je *150 MW*, nominálny výkon *140 MVA* a časová konštanta nábehu vody pri nominálnom výkone je T_w*=2,9 s.*

Simulačná schéma systému je znázornená na Obr. 5.3.2. Na Obr. 5.3.3 je znázornené porovnanie priebehu regulácie pre PID regulátor a regulátor navrhnutý karteziánskym genetickým programovaním. Priebehy riadiacich veličín sú porovnané na Obr. 5.3.4. Nasledujúce výsledky boli dosiahnuté pri optimalizácii funkcie (5.3.6)

$$J = -\int_0^T |e(t)| + 0.1 \cdot |e'(t)|\, dt \qquad \textbf{(5.3.6)}$$



Obrázok 5.3.2: Schéma zapojenia

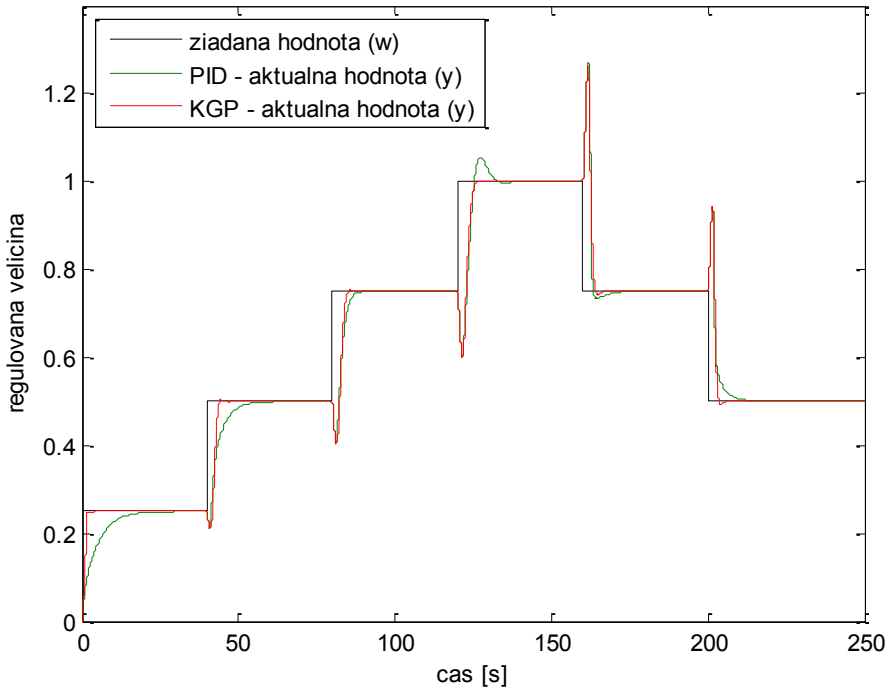
Výsledný regulátor je znázornený na Obr. 5.3.6 a Obr. 5.3.7. Pre systém (Obr. 5.3.2) bolo možné dosiahnuť lepšiu kvalita riadenia, menšie preregulovanie a kratšiu dobu regulácie.

Parameter	Hodnota
P	0,4081
I	0,2430
D	0,1809

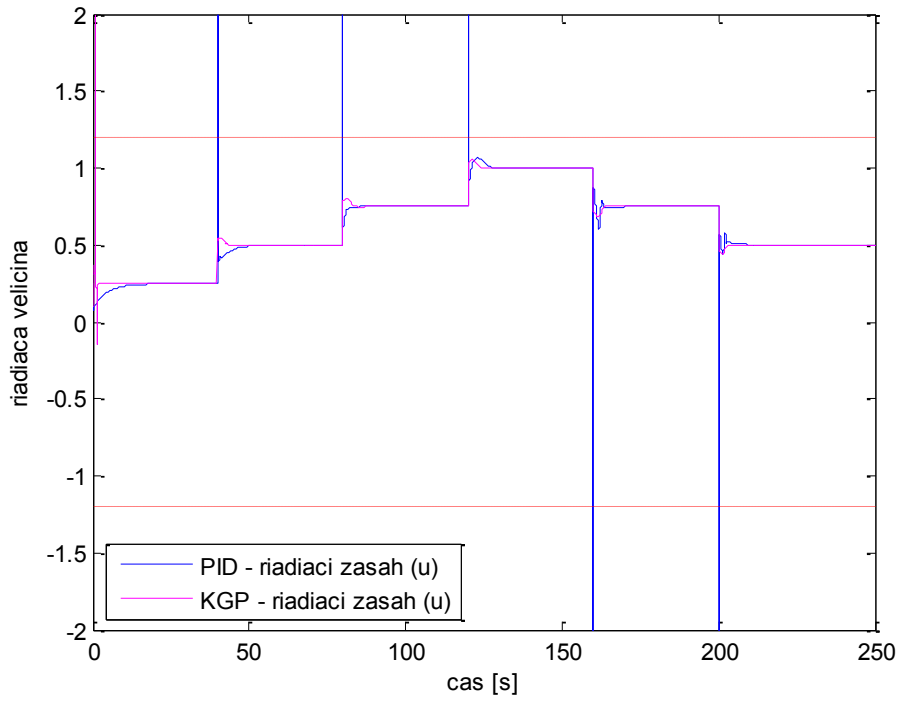
Tabuľka 5.3.1: Parametre PID regulátora

Typ	Fitness
PID	-720.4961
Karteziánske genetické programovanie	-555,7672

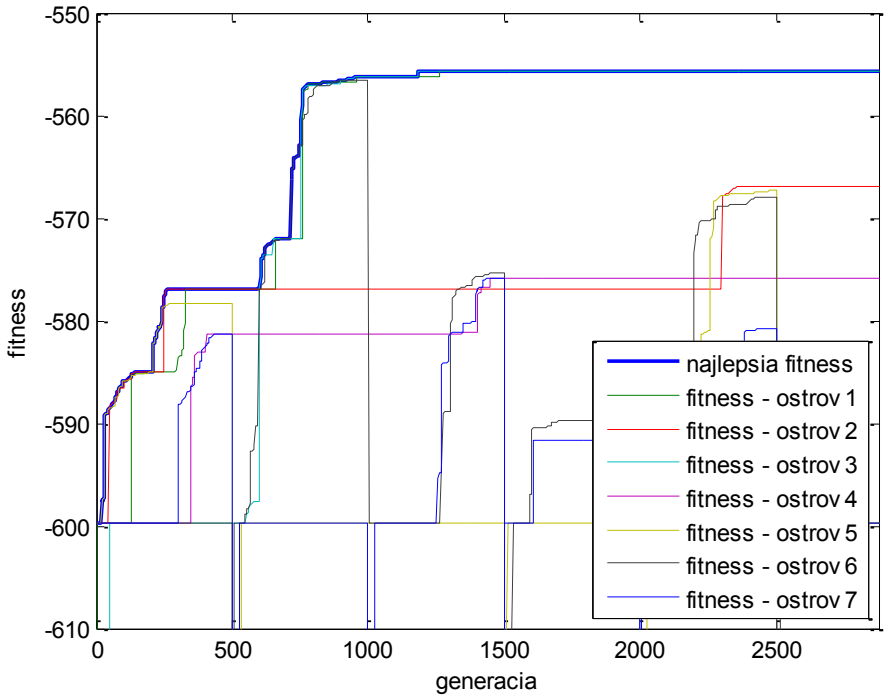
Tabuľka 5.3.2: Porovnanie fitness jednotlivých metód



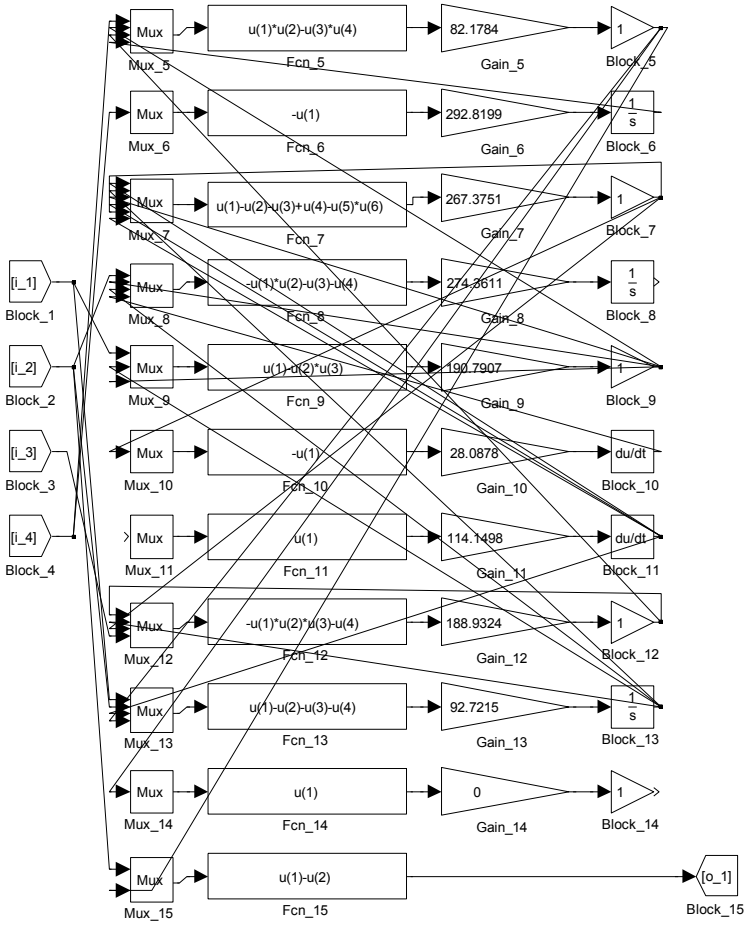
Obrázok 5.3.3: Porovnanie priebehov regulácie nelineárneho systému s PID regulátorom navrhnutým pomocou genetického algoritmu a regulátora navrhnuté pomocou karteziánskeho genetického programovania (KGP)



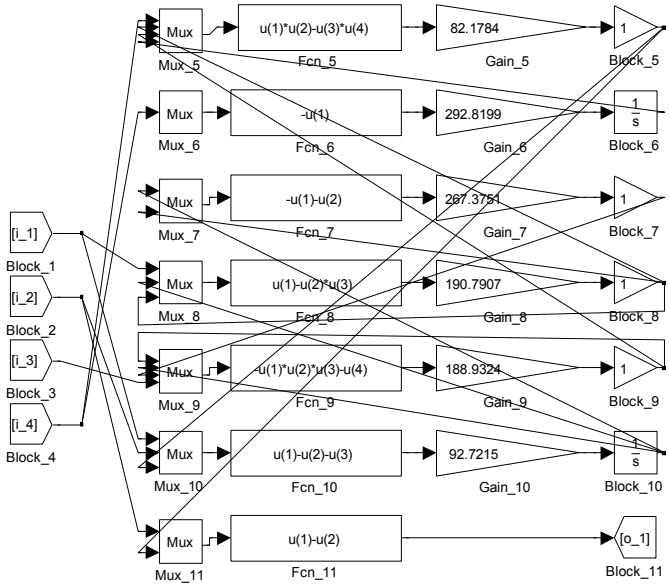
Obrázok 5.3.4: Porovnanie priebehov riadiacich zásahov PID regulátora a regulátora navrhnutého karteziánskym genetickým programovaním (KGP)



Obrázok 5.3.5: Priebeh fitness pre optimalizáciu regulátora navrhnutého pomocou karteziánskeho genetického programovania (KGP)



Obrázok 5.3.6: Bloková schéma regulátora navrhnutého karteziánskym programovaním



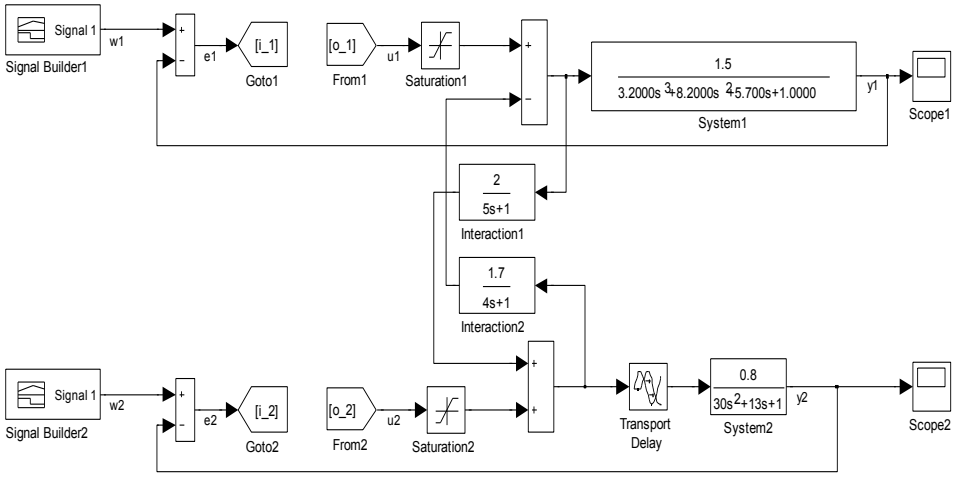
Obrázok 5.3.7: Bloková schéma očistená o nevyužívané objekty

5.4 Návrh MIMO regulačného obvodu

Ďalším krokom bola snaha aplikovať nami navrhnutý prístup na návrh zložitejšej regulačnej štruktúry, ako je jednoduchý spätnoväzobný regulačný obvod. Snažíme sa ukázať, že pri novom prístupe nezáleží od zložitosti, topológie regulačného obvodu a počtu jeho vstupov a výstupov. Bez ujmy na všeobecnosti uvažujme objekt riadenia, ktorý obsahuje dva vstupy, dva výstupy a interakčné väzby medzi subsystémami (Obr. 5.4.1). Hľadáme regulátor, ktorý je schopný riadiť tento systém s požadovanou kvalitou. Tá je

$$J = -\int_0^T (|e_1(t)| + 0.1 \cdot |e_1'(t)|)\, dt - \int_0^T (|e_2(t)| + 0.1 \cdot |e_2'(t)|)\, dt \qquad \textbf{(5.4.1)}$$

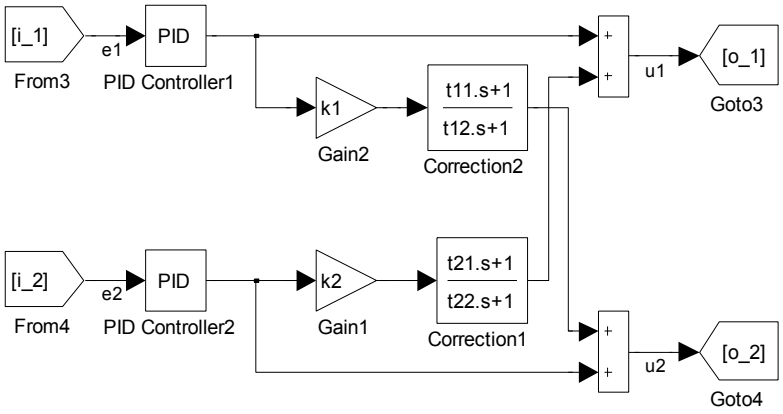
Schéma MIMO regulátora použitá pre návrh konvenčného typu riadenia za účelom porovnania je na Obr. 5.4.2. Obsahuje dva hlavné PID regulátory v priamych vetvách a dva korekčné členy na kompenzáciu vzájomných



Obrázok 5.4.1: Schéma zapojenia

interakcií. Schéma MIMO regulátora (Obr.5.4.2) sa prepojí so schémou riadeného objektu (Obr. 5.4.1) v nasledujúcich prepojeniach: *I_1* - *I_1*, *I_2* – *I_2*, *o_1* – *o_1*, *o_2* – *o_2*. Priebehy regulácie sú na Obr. 5.4.3 a Obr. 5.4.4, kde sú porovnané s reguláciou KGP. Porovnanie hodnôt kriteriálnych funkcií konvenčného riadenia a riadenia s KGP je v Tab. 5.4.2. Grafy priebehu evolúcie vo všetkých ostrovoch pomocou paralelného algoritmu KGP sú zobrazené na Obr. 5.4.7. Na koniec bloková schéma regulátora, ktorý je

výsledkom návrhu pomocou KGP v plnej verzii, ako aj v zjednodušenej verzii bez nefunkčných blokov je na Obr. 5.4.8 a Obr. 5.4.9.



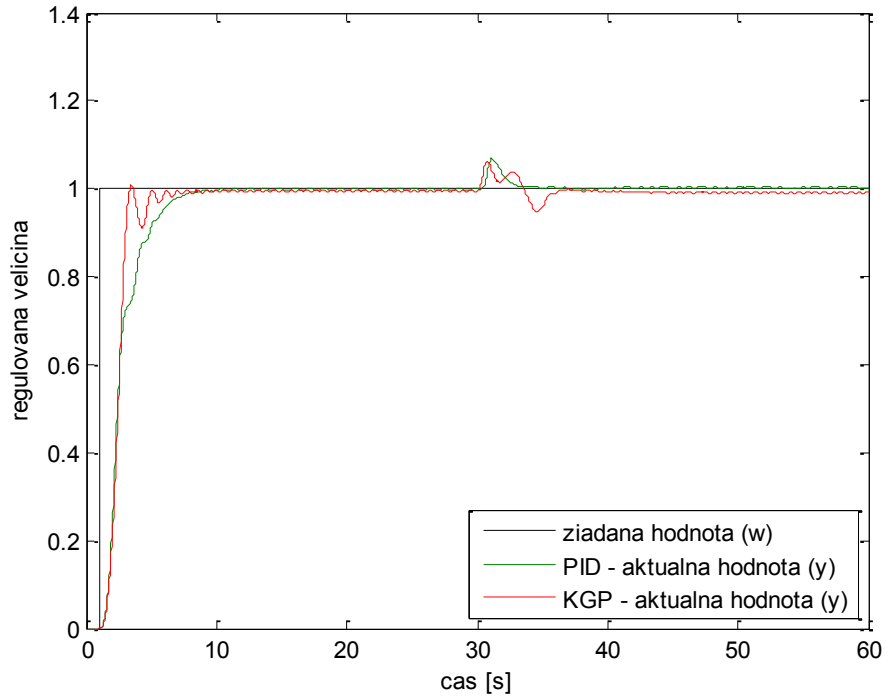
Obrázok 5.4.2: Zapojenie pre konvenčný prístup – s využitím PID regulátorov

Parameter	Hodnota
P1	652,5226
I1	0
D1	870,9029
k1	5,6622
t11	111,1628
t12	679,1911
P2	50,9643
I2	0
D2	96,6445
k2	59,6563
t21	51,0911
t22	954,4391

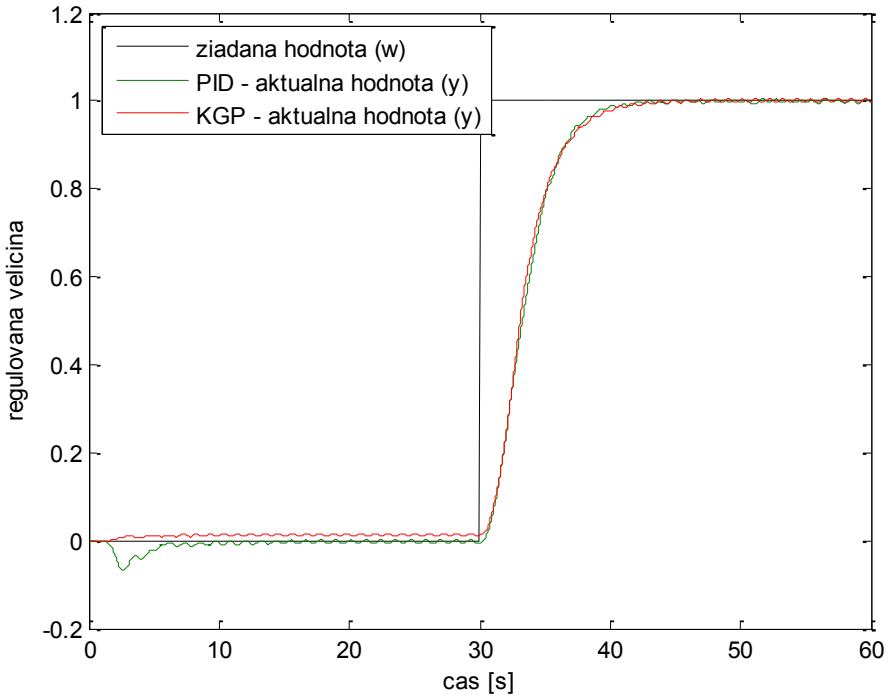
Tabuľka 5.4.1: Parametre konvenčného riadiaceho systému

Typ	Pracovný bod	Integrál e	Integrál de/dt	Fitness spolu
PID	podsystém 1	84,0653	100,4364	-94,1089
	podsystém 2	158,6765	109,4927	-169,6258
	celkovo	242,7418	209,9290	**-263,7347**
Karteziánske genetické programovanie	podsystém 1	80,0447	129,7511	-93,0198
	podsystém 2	160,0934	101,6834	-170,2617
	celkovo	240,1381	231,4346	**-263,2816**

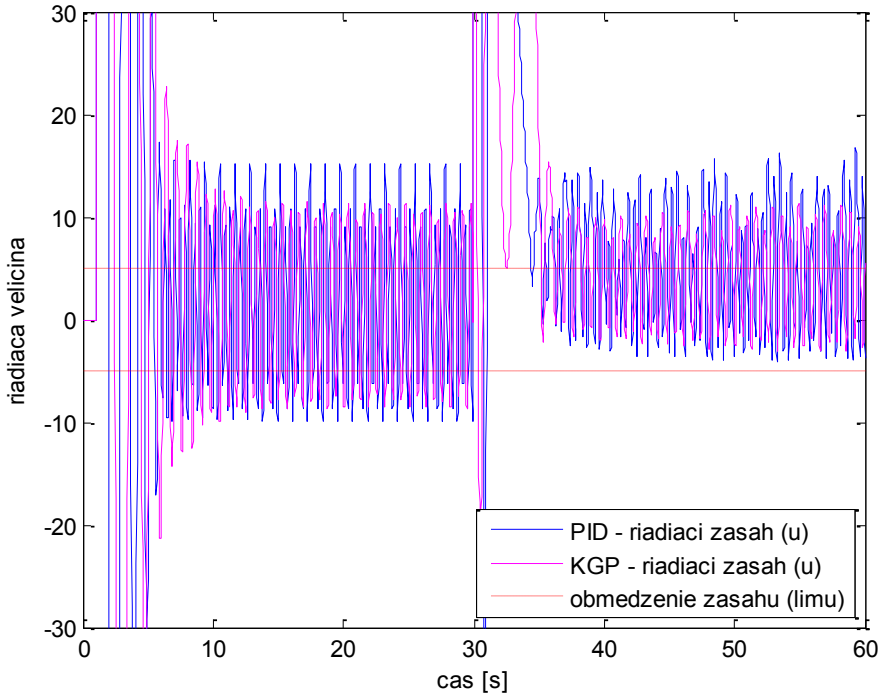
Tabuľka 5.4.2: Porovnanie fitness získaných pomocou konvenčného prístupu a pomocou KGP



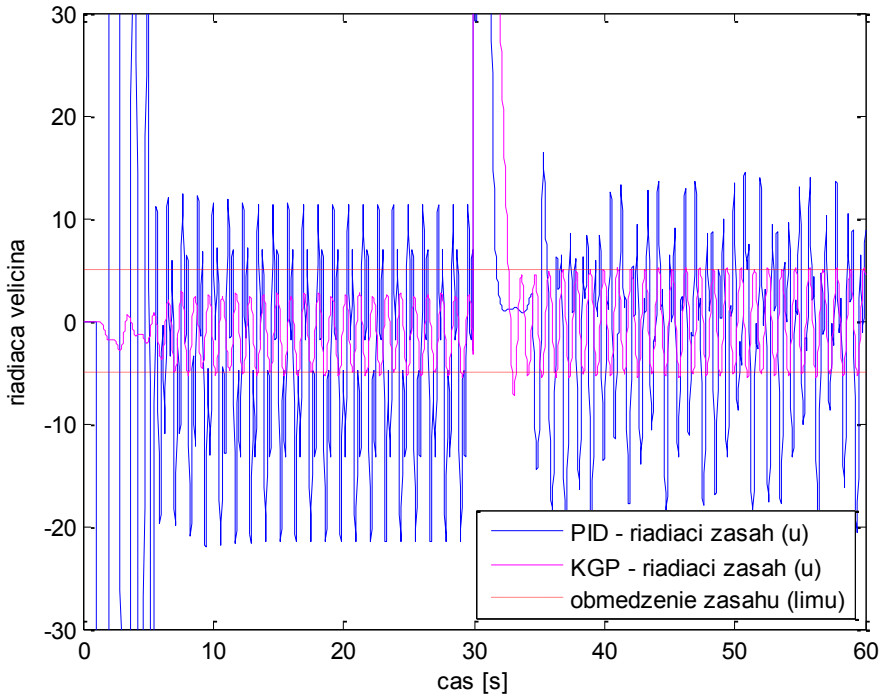
Obrázok 5.4.3: Porovnanie priebehov pre podsystém 1



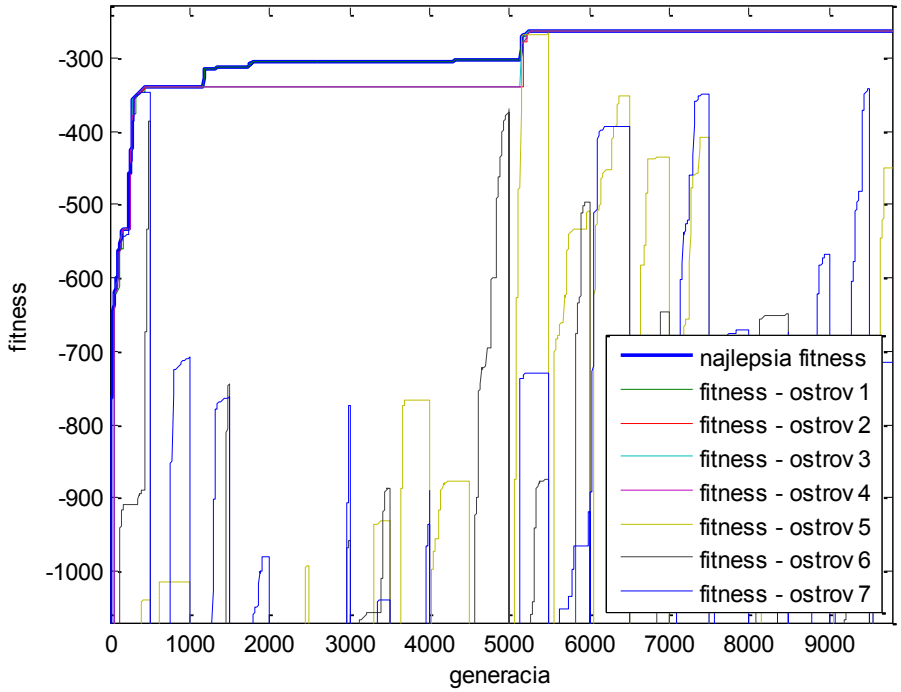
Obrázok 5.4.4: Porovnanie priebehov pre podsystém 2



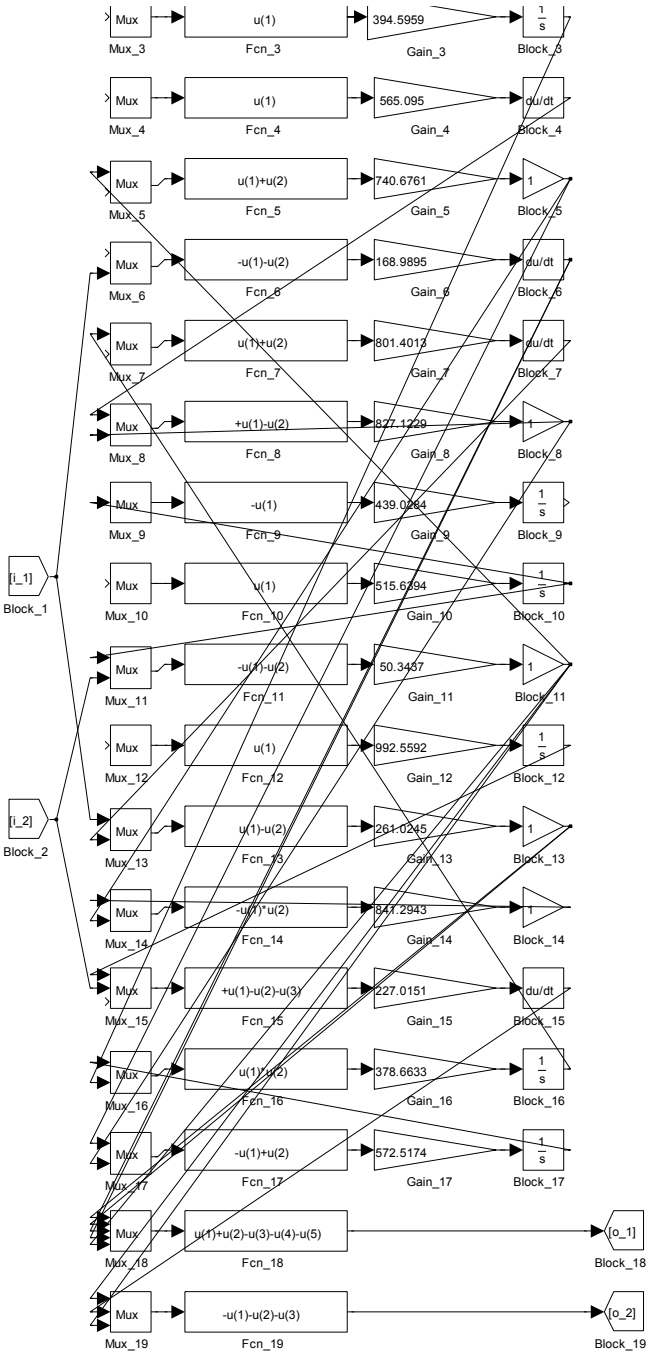
Obrázok 5.4.5: Porovnanie riadiacich zásahov pre podsystém 1



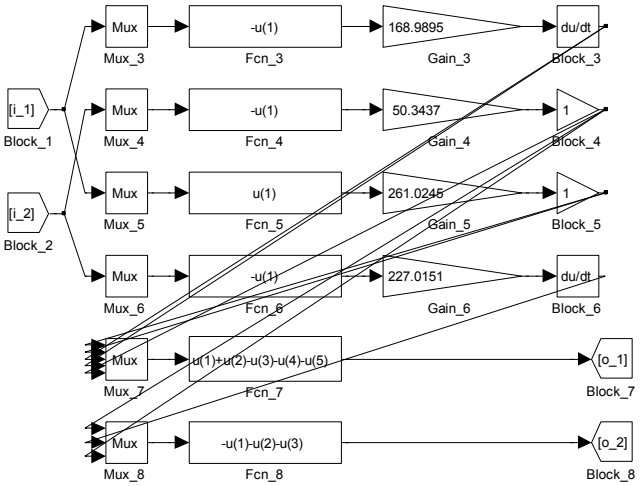
Obrázok 5.4.6: Porovnanie riadiacich zásahov pre podsystém 2



Obrázok 5.4.7: Priebeh fitness pre optimalizáciu regulátora navrhnutého pomocou karteziánskeho genetického programovania (KGP)



Obrázok 5.4.8: Bloková schéma regulátora navrhnutého karteziánskym genetickým programovaním (KGP)



Obrázok 5.4.9: Bloková schéma regulátora navrhnutého karteziánskym genetickým programovaním (KGP) očistená o nevyužívané objekty

6 Záver

V priebehu posledných zhruba dvoch desiatok rokov sa ukázalo, že evolučné algoritmy sú veľmi efektívny optimalizačný prístup, schopný riešiť náročné úlohy z rôznych aplikačných domén. Pritom sa jedná často o úlohy s mnohými optimalizovanými parametrami, s neznámou vnútornou štruktúrou, ktoré sú inými prístupmi riešiteľné iba ťažko, alebo vôbec nie.

Tieto prístupy boli okrem iného úspešne aplikované aj v oblasti riadenia a pri návrhu regulačných obvodov. Medzi úspešných reprezentantov EA v tejto sfére použitia patria predovšetkým Genetické algoritmy. Ako sme už skôr spomenuli, GA sú vhodné na riešenie úloh, keď vnútorná štruktúra problému, v našom prípade regulátora alebo algoritmu riadenia atď. je známa a neznáme sú iba hodnoty ich parametrov. Naproti tomu, keď nepoznáme vnútornú štruktúru navrhovaného/optimalizovaného problému, je potrebné použiť iné evolučné techniky ako sú Genetické programovanie či v niektorých prípadoch aj Gramatická evolúcia. Veľkou prekážkou týchto prístupov je ale veľká (extrémne veľká) výpočtová náročnosť spôsobená veľkou dimenziou prehľadávaného priestoru. Naviac si treba uvedomiť, že vyhodnotenie účelovej funkcie, ktoré je uskutočňované veľmi veľa krát, v prípade návrhu riadenia dynamického systému je v uvažovaných aplikáciách spojené so simuláciou procesu. Toto spôsobuje, že výpočtové časy pri použití napríklad Genetického programovania boli často neprípustne dlhé (desiatky hodín, dni) [46]. Hlavnou motiváciou tejto dizertačnej práce bolo eliminovať tento nedostatok. Vhodným kandidátom sa ukázalo byť Karteziánske genetické programovanie. Tento prístup bol v minulosti úspešne použitý na niektoré iné aplikácie [19]. To dovoľuje redukovať prehľadávaný priestor a tým aj potrebný počet vyhodnotení účelovej funkcie a teda aj výpočtovú náročnosť a výpočtový čas na prijateľnejšie hodnoty. Napriek tomu ponecháva dostatočnú flexibilitu pre nájdenie vhodného riešenia.

Skúsenému kybernetikovi, ktorý si pozrie výsledky regulátorov získané pomocou KGP v tejto práci neunikne fakt, že navrhnuté regulačné štruktúry sú „neintuitívne“. Človek so skúsenosťami v riadení by ich takto nenavrhol. Napriek tomu sú tieto výsledky takmer vždy lepšie, než výsledky získané

konvenčnými, bežne zaužívanými regulačnými štruktúrami. Toto je výrazným špecifikom aj Genetického programovania. Okrem toho, dá sa predpokladať, že ani riešenia nájdené pomocou KGP nie sú globálnymi optimami zvolených kriteriálnych funkcií. Kvalita regulácie, robustnosť, miera stability, prípadne energetické aspekty a iné kvalitatívne ukazovatele sa postupne zlepšujú s počtom výpočtových cyklov (generácií). Nezriedka ale uviaznu v lokálnom extréme, keďže nájsť globálny extrém môže byť ťažká, až nerealizovateľná úloha.

Napriek tomu, výsledky dosiahnuté v tejto práci ukázali, že KGP má potenciál riešiť úlohy z oblasti riadenia pre rôzne typy problémov. Pôvodné výsledky, ktoré tu prezentujeme sú všeobecne opísané v kapitole 4 a experimentálne výsledky sú dokumentované v kapitole 5. Navrhli sme pôvodný prístup využitia KGP pre návrh regulačných obvodov a ukázali sme úspešné riešenie návrhu štruktúry aj parametrov riadenia pre nelineárne dynamické systémy, systémy s rozvetvenými (netriviálnymi) regulačnými obvodmi, ktoré obsahujú viac vstupov aj výstupov. Ukázali sme aj riešenie aspektu robustnosti. Na základe našich skúseností si trúfneme konštatovať, že nami navrhnutý prístup má potenciál riešiť aj iné typy úloh z oblasti kybernetiky, kde štruktúra hľadaného objektu nie je vopred známa. Môže ísť o oblasti identifikácie dynamických systémov, o návrh algoritmov riadenia spojitých, diskrétnych systémov, o logické riadenie o aplikácie v robotike, mechatronike a podobne. Hlavným obmedzením nášho prístupu je výpočtový čas, ktorý býva dlhší, než pri „konvenčných“ metódach návrhu. Na druhej strane obmedzením nie je typ riadeného objektu. Môže sa jednať lineárny/nelineárny systém, systém so zložitou štruktúrou, mnohými vstupmi a výstupmi. Kritériom návrhu môže byť ľubovoľná miera kvality regulácie, obmedzenia energie, veľkosti vstupných a výstupných veličín. Do úvahy môžeme brať existujúce poruchy, režimy prevádzky, šum a podobne. Jedinou podmienkou návrhu je existencia adekvátneho modelu riadeného objektu, ktorého simulácia je súčasťou vyhodnotenia kriteriálnej funkcie.

Použitá literatúra

[1] Aarstad A.S.: Evolving Locomotion for a Quadruped Robot using Cartesian Genetic Programming and Physics Simulation, University of Oslo, 2012

[2] Ashmore L., Miller J.F.: Evolutionary Art with Cartesian Genetic Programming. Department of Informatics, University of Sussex, Falmer, BN1 9QH, Veľká Británia, 2004

[3] Banzhaf W., Nordin P., Keller R. E., Francone F. D.: Genetic Programming: An introduction. Morgan Kaufmann, San Francisco, 1999

[4] Castro L.N., Timmins J.: An Artificial Immune Network for Multimodal Function Optimisation. In Proceedings on Congress on Evolutionary Computation 2002, IEEE Press, pp. 699-704, 2002

[5] Dorf R. C.: Modern Control Systems, Addison-Wesley publishing Company, 5th edition, 1990

[6] Eberhart R. C., Kennedy J.: A new optimizer using particle swarm theory. In Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan. pp. 39-43, 1995

[7] Eiben A. E., Smith J. E.: Introduction to Evolutionary Computing. Springer, 2003

[8] Fleming P. J., Purshouse R. C.: Evolutionary Algorithms in Control System Engineering: A Survey. In Control Engineering Practice, 10(11), 1223, 2002

[9] Garipov E., Puleva T., Haralanova E.: Modeling and simulation of hydraulic turbine power control., Proc. Int. Conf. on Modeling and Simulation (MS'10 Prague), Chech Republic, 2010

[10] Goldberg D. E.: Genetic Algorithms in Search, Optimisation and Machine Learning. Addisson-Wesley, 1989

[11] Grosman B.; Lewin D.R.: Automatic Generation of Lyapunow Functions Using Genetic Programming, In: 16th World Congress of IFAC, Prague, July 3-8, 2005

[12] Harding S., Miller J. F.: Evolution of Robot Controller Using Cartesian Genetic Programming, EuroGP 2005, LNCS 3447:62–73, 2005

[13] Herrero J.M., Blasco X., Martínez M., Salcedo J.V.: Optimal PID Tuning with Genetic Algorithms for Non Linear Process Models. In Proceedings on the 15th World Congress of IFAC, Barcelona, July 21-2, 2002

[14] Hirayama Y., Clarke T., Miller J.F.: Fault Tolerant Control using Cartesian Genetic Programming., University of York, Veľká Británia, 2008

[15] Kadlic B., Sekaj I., Pernecký D.: Design of continuous-time controllers using cartesian genetic programming. In Proceedings of the 19th World Congress of the International Federation of Automatic Control (elektronický zdroj) : IFAC 2014: 19th World Congress of the International Federation of Automatic Control. Cape Town, South Africa, 24-29 August 2014. (s.l.) : IFAC, 2014, 6982-6987. ISBN 978-3-902823-62-5, 2014

[16] Kadlic B., Sekaj I.: Cartesian Genetic Programming Based Controller Design, Conference. MENDEL ’12, Brno, 2012

[17] Kadlic B., Sekaj I.: Controller Design Based on Cartesian Genetic Programming in Matlab, Conference: TECHNICAL COMPUTING PRAGUE 2011. Praha, 2011

[18] Kadlic B., Sekaj I.: Controller Design Based on Cartesian Genetic Programming in Matlab. Conference: ELITECH ’12, Bratislava, 2012

[19] Kadlic B.: Písomná práca k dizertačnej skúške, Slovenská technická univerzita v Bratislave - Fakulta elektrotechniky a informatiky, 2013

[20] Kawabe T., Tagami T., Katayama T.: A Genetic Algorithm based Minimax Optimal Design of Robust I-PD Controller. In UKACC Int. Conference on Control ’96, Conf. Publication No. 427, IEE, pp.436-441, 1996

[21] Khatib W., Silva V., Chipperfield A., Fleming P.: Multidisciplinary Optimisation with Evolutionary Computing for Control Design. In Proceedings on the 14thWorld Congress of IFAC. Beijing, July 5-9, 1999

[22] Koza J. R..: Genetic Programming. Cambridge, MA, MIT Press, 1992

[23] Koza J.R., Yu J., Keane M.A., Mydlowec W.: Evolution of a controller with a free variable using genetic programming. In Proceedings on the European Conference EuroGP 2000. Edinburgh, Scotland, UK, Lecture Notes in Computer Science, Volume 1802. Berlin, Germany: Springer-Verlag, pp. 91–105, ISBN 3-540-67339-3, April 2000

[24] Koza J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge Massachusetts, 1994

[25] Koza J.R.: Genetic Programming. Cambridge, MA, MIT Press, 1992

[26] Krohling R.A., Rey J.P.: Design of Optimal Disturbance Rejection PID Controllers Using Genetic Algorithms. InIEEE Trans. On Evolutionary Computation, Vol.5, No.1, 2001

[27] Kuo B.C.: Automatic Control Systems, Prentice-Hall International Editions, 1991

[28] Kvasnička V., Pospíchal J., Tiňo P.: Evolučné algoritmy. Vydavateľstvo STU, Bratislava, 2000

[29] Lazarus C., Hu H.: Using genetic programming to evolve robot behaviours. In Proc. of the 3rd British Workshop on Towards Intelligent Mobile Robots, 2001

[30] Leitner J., Harding S., Förster A., Schmidhuber J.: Mars Terrain Image Classification Using Cartesian Genetic Programming. Dalle Molle Institute for Artificial Intelligence (IDSIA), SUPSI and Universit`a della Svizzera Italiana (USI), Lugano, Švajčiarsko, 2012

[31] Lewin D.R.: Evolutionary Algorithms in Control System Engineering. In Proceedings on the 16th World Congress of IFAC, July 3-8, Prague, 2005

[32] Lipson H., Pollack J. B.: The golem project. In NATURE volume 406. pages 974-978, 2000 (http://demo.cs.brandeis.edu/golem/ (2010-09-29))

[33] Man K.F.; Tang K.S.; Kwong S.: Genetic Algorithms, Concepts and Deign, Springer, 2001

[34] McKay R., Hoai N. Whigham P., Shan Y., O'Neill M.: Grammar-based Genetic Programming: a survey. Genetic Programming and Evolvable Machines, 2010

[35] Michalewicz Z.: Genetic Algorithms + Data Structures = Evolutionary Programs. Springer, 1996

[36] Miller J. F. a kol.: Cartesian Genetic Programming. Springer, 2011 (ISBN: 978-3-642-17309-7, Dept. of Electronics, The University of York, Heslington, YO10 5DD, UK, jfm7@ohm.york.ac.uk, e-ISBN 978-3-642-17310-3, ISSN 1619-7127 Natural Computing Series, DOI 10.1007/978-3-642-17310-3, Library of Congress Control Number: 2011938670

[37] Miller J. F., Thomson P., Fogarty T. C.: Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study. (Genetic Algorithms and Evolution Strategies in Engineering and Computer Science), Wiley, 1997

[38] Miller. J.: An empirical study of the e±ciency of learning boolean functions using cartesian genetic programming approach. volume 2 of Proc. of GECCO, page 1135-1142. Morgan Kaufmann, 1999

[39] Mitsukura Y., Yamamoto T., Kaneda M., Fujii K.: Evolutionary Computation in Designing a PID Control System. In Proceedings on the 14thIFAC World Congress, Beijing, 5-9 july, P.R.China, pp. 497-502, 1999

[40] Nidel, M.: Návrh regulačných štruktúr pomocou nástrojov genetického programovania. Diplomová práca, Slovenská Technická Univerzita, Fakulta Elektrotechniky a informatiky, Bratislava, 2005

[41] Nohejl A.: Grammar-based genetic programming. Diplomová práca, Univerzita Karlova v Praze, Praha, 2011 (http://nohejl.name/age/pdf/Adam-Nohejl-2011-Grammar-based-GP.pdf)

[42] Nohejl A.: Grammatical Evolution. Bakalárska práca, Univerzita Karlova v Praze, Praha, 2009 (http://nohejl.name/age/pdf/AGE-Documentation-1.0.2.pdf)

[43] O'Neill M., Ryan C.: Grammatical Evolution. IEEE Transactions on Evolutionary Computation, 2001

[44] O'Neill M., Ryan C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Springer, 2003

[45] Páleník, T.: Návrh regulátorov pomocou genetického programovania. Diplomová práca, Slovenská Technická Univerzita, Fakulta Elektrotechniky a informatiky, Bratislava, 2006

[46] Perkácz J.: Automatická syntéza štruktúry riadenia pomocou genetického programovania. Dizertačná práca, FEI STU Bratislava, 2007

[47] Pernecký D., Sekaj I.: Grammatical evolution based controller design. Príspevok: MENDEL ’13, Brno, 2013

[48] Puleva T., Garipov E., Ruzhekov G.: Adaptive Power Control Modeling and Simulation of a Hydraulic Turbine, ISAP, 2011

[49] Reynolds C.W.: An evolved, vision-based behavioral model of obstacle avoidance behaviour. In Christopher G. Langton, editor, Artificial Life III, volume 16 of SFI Studies in the Sciences of Complexity. Addison-Wesley, 1993

[50] Ryan, C., Collins, J.J., O’Neill, M.: Grammatical Evolution: Evolving Programs for an Arbitrary Language, EuroGP',98: Proceedings of the First European Workshop on Genetic Programming, vol. 1391, pp.83 - 95 1998: Springer-Verlag, 1998

[51] Searson D., Willis M., Montague G.: Chemical Process Controller Design Using Genetic Programming. University of Newcastle, Veľká Británia, 1998

[52] Sekaj I., Perkácz J., Nídel M.: Controller design based on genetic programming. In Proceedings of the Int. conference Mendel'05. Brno, Czech Republic, June 15-17, 2005

[53] Sekaj I., Perkácz J.: Genetic Programming - Based Controller Design. IEEE Congress on Evolutionary Computation, September 25-28, 2007

[54] Sekaj I., Šrámek M.: Robust Controller Design Based on Genetic Algorithms and System Simulation. In Proceedings on the conference CDC-ECC'05, Seville, Spain, December 12-15, 2005

[55] Sekaj I., Veselý V.: Robust output feedback controller design: Genetic Algorithm approach. In IMA Journal of Mathematical Control and Information, 22, pp. 257-263, 2005

[56] Sekaj I.: Control algorithm design based on evolutionary algorithms. In: Chugo, D., Yokota, S. a kol.: Introduction to Modern Robotics. iConcept

Press, Hong Kong, 2011 (ISBN 978-0980733068, http://www.iconceptpress.com/books/introduction-to-modern-robotics)

[57] Sekaj I.: Evolučné výpočty. IRIS, Bratislava, 2005

[58] Sekaj I.: Evolutionary Controller design. In: Wellington Pinheiro dos Santos. Evolutionary Computation, In-Teh, Vukovar, Chroatia, 2009 (www.intechopen.com)

[59] Sekaj I.: Genetic Algorithm Based Controller Design. In 2nd IFAC conference Control System Design'03. Bratislava, Slovak Republic, September 7-10, 2003

[60] Sekaj I.: Genetic Algorithm-based Control System Design and System Identification. In Proceedings on the Int. Conference Mendel'99, June 9-12, Brno, Czech Republic, pp.139-144, 1999

[61] Sekanina L. a kol.: Evoluční hardware. Academia, Praha, 2009 (ISBN: 978-80-200-1729-1)

[62] Sweriduk G.D., Menon P.K., Steinberg M.L.: Design of a pilot-activated recovery system using genetic search methods. In:Optimal Synthesis, 1999

[63] Yang Z.Y., Chan C.W., Xue M.S., Luo G.J.: On-line Temperature Control of an Oven Based on Genetic Algorithms. In Proceedings on the 16th World Congress of IFAC, Prague, July 3-8Molina-Cristóbal, A.; Griffin, I.A.; Fleming, P.J.; Owens, D.H. (2005). Multiobjective Controller Design: Optimising Controller Structure with GA, In: Proceedings on the 16th World Congress of IFAC, July 3-8, Prague, 2005

[64] Zelinka I. a kol.: Evoluční výpočetní techniky, principy a aplikace. Ben, Praha 2009

[65] http://www.cartesiangp.co.uk/

[66] http://www.wikipedia.org/

[67] http://www.genetic-programming.com/

[68] http://link.springer.com/book/10.1007/978-3-642-18609-7/page/1

[69] http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/LukasSekanina.html