Sinteza optimalnog regulatora stanja linearnih sustava primjenom algoritma povratnog prostiranja kroz vrijeme

Trbara, Klara

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Klara Trbara

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Izv. prof. dr. sc. Vladimir Milić, mag.ing.

Student:

Klara Trbara

Zagreb, 2023.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i prijašnje stečenog obrazovanja uz pomoć navedene literature.

Iznimno hvala i mom mentoru izv. prof. dr. sc. Vladimir Milić na dostupnosti, pomoći te korisnim savjetima tijekom izrade ovog rad.

Jedno veliko hvala mojoj obitelji na neizmjernoj podršci, razumijevanju i strpljenju kroz sve godine mog studiranja.

Klara Trbara

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 23 - 6 / 1**	
Ur.broj: **15 - 23 -**	

DIPLOMSKI ZADATAK

Student: **Klara Trbara** JMBAG: 0035211817

Naslov rada na hrvatskom jeziku: **Sinteza optimalnog regulatora stanja linearnih sustava primjenom algoritma povratnog prostiranja kroz vrijeme**

Naslov rada na engleskom jeziku: **Optimal state controller synthesis for linear systems using the back-propagation through time algorithm**

Opis zadatka:

Optimalno upravljanje linearnim dinamičkim sustavima predstavlja jedno od najintenzivnije istraživanih područja iz teorije upravljanja. Zbog matematičke kompleksnosti analitičkih metoda za rješavanje nužnih uvjeta optimalnosti neophodna je primjena numeričkih metoda za sintezu optimalnog regulatora kako bi se omogućila njegova praktična implementacija. Budući da dinamički upravljački sustavi imaju povratnu vezu, za računanje derivacija funkcije cilja optimizacijskog problema može se primijeniti lančano pravilo za obične derivacije s prostiranjem pogreške unazad u vremenu.
U diplomskom radu je potrebno:

1. Provesti teorijska razmatranja o metodama sinteze regulatora stanja linearnih dinamičkih sustava prema kvadratnom kriteriju optimalnosti s uključenim ograničenjima na varijablama upravljanja.
2. Postaviti problem sinteze optimalnog regulatora stanja pri čemu je zakon upravljanja jednoslojna neuronska mreža sa saturacijskom aktivacijskom funkcijom. Primjena ovakve aktivacijske funkcije omogućit će da ograničenja varijabli upravljanja budu uvijek zadovoljena.
3. Izvesti algoritam podešavanja težina neuronske mreže, a time i parametara regulatora. Algoritam u prvom redu treba biti temeljen na algoritmu povratnog prostiranja kroz vrijeme (engl. *Back-Propagation Through Time* – BPTT) za rekurzivno računanje gradijenata funkcije cilja. Nadalje, u algoritamsku proceduru trebaju biti integrirane: Eulerova metoda za vremensku diskretizaciju, metoda konjugiranog gradijenta za podešavanje težina neuronske mreže te kompleksna koračna metoda za računanje Jacobiana funkcija dinamike sustava s obzirom na vektor stanja i vektor ulaza.
4. Izvedeni algoritam napisati u nekom višem programskom jeziku kao što su npr. matematički programski alati MATLAB ili NumPy/SciPy u Pythonu.
5. Provesti analizu i usporedbu rezultata izvedenog algoritma s nekom poznatom numeričkom metodom za sintezu linearnog kvadratnog regulatora (engl. *Linear Quadratic Regulator* – LQR) standardno ugrađenom u nekom matematičkom programskom alatu na nekim standardnim primjerima linearnih sustava, npr. na mehaničkim sustavima masa-prigušnica-opruga.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predvideni datumi obrane:
28. rujna 2023.	30. studenoga 2023.	4. – 8 . prosinca 2023.

Zadatak zadao: Izv. prof. dr. sc. Vladimir Milić

Predsjednik Povjerenstva: Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
A	-	Matrica koeficijenata sustava
B	-	Matrica ulaza sustava
b	-	Vektor ulaza sustava
C	-	Matrica izlaza sustava
c	-	Vektor izlaza sustava
c	Ns/m	Faktor prigušenja
D	-	Matrica prijenosa sustava
$d^{()}$	-	Dilatacijski parametri
E	-	Matrica upravljivosti
$\boldsymbol{e}_j$	-	Jedinični vektor
$\mathbf{F}_x, \mathbf{F}_u$	-	Prošireni oblik Jacobiana
H	-	Hamiltonova funkcija
H	-	Matrica osmotrivosti
$G(s)$	-	Prijenosna funkcija sustava
$\boldsymbol{I}$	-	Jedinična matrica
I	-	Funkcija cilja
i	-	Imaginarna jedinica
J	-	Funkcija cilja
J_C	-	Kvadrat funkcije cilja
k	-	Vektor stanja povratne veze
k	N/m	Faktor krutosti opruge
K	-	Matrica stanja povratne veze
p(s)	-	Karakterisitčni polinom zatvorenog sustava
q(s)	-	Karakteristični polinom otvorenog sustava
Q	-	Matrica težina stanja linearnog kvadratičnog regulatora
R*(s)*	-	Rezolventna matrica
R	-	Matrica težina ulaza linearnog kvadratičnog regulatora
S	-	Matrica pretraživanja
t_f	-	Konačan broj točaka
x	-	Vektor stanja sustava
$\dot{x}$	-	Derivacija vektora stanja
u	-	Vektor upravljanja
$u(s)$	-	Laplaceove transformacija ulaza

$\mathbf{U_t}$	-	Nesingularna trokutna matrica
v	-	Vanjski ulaz u sustav
$V(x)$	-	Lyapunovljeva funkcija stabilnosti
$\dot{V}(x)$	-	Derivacija Lyapunovljeve funkcije
$\mathbf{W}$	-	Matrica težina neuronske mreže
z(t)	-	Vektor kaznena varijabla
y	-	Vektor izlaza
$y(s)$	-	Laplaceove transformacija izlaza
$\boldsymbol{\sigma}(\cdot)$	-	Jednoslojna neuronska mreža
ξ_j	-	Suma umnoška težina i vektora stanja
$\tilde{x}$	-	Proširena varijabla stanja
$\dot{\chi}$		Derivacija proširene funkcije stanja
$\boldsymbol{\lambda}$	-	Vektor lagrangeovih multiplikatora
τ	-	Duljina vremenskog koraka
η_i	-	Stopa učenja u *i*-tom koraku
Adj()	-	Adjunktivna matrica
det()	-	Determinanta matrice
$\|\cdot\|$	-	Općenita norma vektora
$\min_W$	-	Minimizacijski problem
$\otimes$	-	Kroneckerov produkt
∂	-	Parcijalna derivacija
Tr()	-	Trag matrice

SAŽETAK

Diplomski rad istražuje optimalno upravljanje linearnim dinamičkim sustavima kroz primjenu numeričkih metoda, s posebnim naglaskom na sintezu optimalnog regulatora stanja. Zbog kompleksnosti analitičkih metoda, koriste se numeričke metode kako bi se omogućila praktična implementacija optimalnog regulatora. Fokus rada je na upotrebi jednoslojne neuronske mreže sa saturacijskom aktivacijskom funkcijom kao zakonom upravljanja, osiguravajući pritom zadovoljavanje ograničenja varijabli upravljanja. Rad uključuje algoritam podešavanja težina neuronske mreže, temeljen na algoritmu povratnog prostiranja kroz vrijeme (eng. *Back-Propagation Through Time* – BPTT), uz integraciju Eulerove metode za vremensku diskretizaciju, metode konjugiranog gradijenta, te kompleksne koračne metode za računanje Jacobijana funkcije dinamike sustava. Navedene metode su integrirane u jedan numerički algoritam za sintezu regulatora prema kvadratnom kriteriju optimalnosti. Izvedeni algoritam je provjeren na sintezi regulatora mehaničkog sustava masa-prigušnica-opruga.

Ključne riječi: linearni vremenski invarijantni sustavi, linearni kvadratni regulator, neuronske mreže, algoritam povratnog prostiranja kroz vrijeme, konjugirani gradijent, SuperSAB algoritam

SUMMARY

The master's thesis explores optimal control of linear dynamic systems through the application of numerical methods, with a particular focus on the synthesis of an optimal state regulator. Due to the complexity of analytical methods, numerical methods are used to enable the practical implementation of the optimal regulator. The focus of the thesis is on the use of a single-layer neural network with a saturation activation function as the control law, ensuring the satisfaction of control variable constraints. The work includes a weight adjustment algorithm for the neural network based on the Backpropagation Through Time (BPTT) algorithm, integrating Euler's method for time discretization, conjugate gradient method, and complex step method for computing the system's Jacobian function. These methods are integrated into a single numerical algorithm for synthesizing the regulator according to the quadratic optimality criterion. The derived algorithm is validated in the synthesis of a regulator for a mechanical system consisting of mass-damper-spring components.

Key words: linear time-invariant systems, linear quadratic regulator, neural networks, backpropagation through time algorithm, conjugate gradient, SuperSAB algorithm

1. UVOD

U raznim suvremenim inženjerskim područjima, upravljanje dinamičkim sustavima ključno je za postizanje željenih performansi, stabilnosti i optimalne potrošnje resursa. U tom kontekstu, sinteza optimalnih regulatora predstavlja ključnu komponentu u postizanju učinkovitog upravljanja. Linearni sustavi predstavljaju široko primijenjenu klasu dinamičkih sustava, a optimalno upravljanje ovim sustavima ima značajan utjecaj na različite inženjerske discipline, uključujući robotiku, sustave za upravljanje letjelicama, industrijske procese i energetiku. Neki od primjera se mogu vidjeti i u [12-15].

Ovaj diplomski rad razmatra analizu i primjenu algoritma povratnog prostiranja kroz vrijeme (eng. *Backward Propagation Through Time* - BPTT) [6] u kontekstu sinteze optimalnog regulatora stanja za linearni dinamički sustav. Osnovna razmatranja primjene samog algoritma su temeljena prema [1], gdje se promatrani min-max optimalni problem upravljanja za nelinearne sustave rješava s pomoću jednoslojne neuronske mreže kojoj je za izlaz (eng. *output*) dan zakon upravljanja. Podešavanje težina neuronskih mreža je temeljeno na minimizaciji funkcije cilja, dok se istovremeno vektor poremećaja računa prema maksimizaciji iste funkcije. Za sam izračun derivacija funkcije cilja te minimuma i maksimuma korištena je metoda konjugiranog gradijenta. Nadalje, kako su vektor stanja, poremećaja i upravljanja povezani preko jednadžbi stanja sustava, cijeli algoritam ima oblik povratnog prostiranja kroz vrijeme sa sličnom strukturom kao i BPTT. U svrhu ovog diplomskog rada cijeli sustav je prilagođen na linearne sustave bez poremećaja te je problem sveden na minimizaciju funkcije cilja kao pravilo podešavanja težina neuronske mreže.

U prvom dijelu ovog rada provedena su neka opća razmatranja vezana za linearne dinamičke sustave i predstavljene su glavne karakteristike moderne linearne teorije upravljanja. One koje obuhvaća ovaj rad su opis sustava u prostoru stanja i metode sinteze regulatora stanja linearnih dinamičkih sustava prema kvadratnom kriteriju optimalnosti s uključenim ograničenjima na varijablama upravljanja. Predstavljeni su ključni koncepti upravljivosti, osmotrivosti i stabilnosti u kontekstu LTI sustava. Upravljivost sustava je bitna u postizanju željene performanse sustava putem vanjskog upravljana. Izvedena je matrica upravljivosti kojom

možemo analizirati stanje našeg sustava. Osmotrivost je ključna jer omogućava praćenje unutarnjeg stanja sustava na temelju vanjskih mjerenja. Kako stanja sustava ne bi ostala skrivena ili nedostupna za praćenje, izvedena je matrica osmotrivosti kojom možemo analizirati naš sustav. Aleksandar Mihailović Ljapunov postavio je definicije i koncepte stabilnosti koji se oslanjaju na skalarnu funkciju $V(x)$, koja se može smatrati općom funkcijom energije. Za dokazivanje stabilnosti linearnih sustava Ljapunovljevim pristupom potrebno je ispuniti dva uvjeta: funkcija $V(x)$mora biti pozitivno definitna, a vremenska derivacija $\dot{V}(x)$ mora biti negativno semidefinitna.

Drugi dio rada obuhvaća dubinsku analizu algoritma povratnog prostiranja kroz vrijeme (BPTT). Fokus je na teorijskom istraživanju ovog algoritma, posebno njegove primjene u sintezi optimalnog regulatora stanja. Kroz detaljnu analizu matematičkih aspekata BPTT-a, istražit će se kako ovaj algoritam pridonosi optimizaciji i upravljanju sustavima. Nadalje, bit će dana opća usporedba različitih metoda optimizacije (Dai-Yuan, Fletcher-Reeves, Polak-Ribiere, Hestenes-Stiefel) u odnosu na obični BPTT algoritam. Dodatno, istražit će se utjecaj izmjene početnih parametara na funkcionalnost BPTT algoritma. Analizirat će se kako varijacije početnih parametara utječu na konvergenciju i performanse algoritma, pružajući dublje razumijevanje osjetljivosti algoritma na inicijalne uvjete. Za kraj će se provesti analiza i usporedbu rezultata izvedenog algoritma s nekom poznatom numeričkom metodom za sintezu LQR regulatora standardno ugrađenom u Matlab matematičkom programskom alatu za mehanički sustav masa-prigušnica-opruga.

Cilj je razviti duboko razumijevanje ovog algoritma i demonstrirati njegovu primjenu u rješavanju konkretnih problema upravljanja, uzimajući u obzir različite aspekte kao što su stabilnost, performanse sustava i ograničenja resursa (vremenskih, tehničkih,..).

1.1. Pregled literature

U ovom poglavlju će se ukratko prikazati pregled literature korištene u izradi ovog diplomskog rada. Znanstvena područja koja obrađuje navedena literatura su podijeljena u dvije skupine. Prva skupina literature razmatra općenito teoriju upravljanja i regulacije dinamičkih sustava [2,3,4,5], a druga skupina se bavi modeliranjem neuronskih mreža pomoću popratnih algoritama [1,11,7] te njihove izvedbe [6,8,9,10].

Prvi dio literature koja obrađuje općenita razmatranja teorije upravljanja i regulacije dinamičkih sustava bavi se analizom i sintezom kako linearnih tako i nelinearnih kontinuiranih i diskretnih sustava. Obrađena su područja optimalnog i robusnog upravljanja linearnim sustavima. Uvodi se metode prostora stanja. Obrađuju se generalizirani Cayley-Hamiltonov teorem koji pruža temeljna matematička saznanja o strukturi sustava i implikacijama za analizu i sintezu kontrolnih sustava. Razmatraju se i osobine sustava bitne za analizu i sintezu regulatora, a one su : upravljivost, osmotrivost i stabilnost.

U literaturi [6,8] je predstavljen detaljan uvid u upotrebu BPTT algoritma u neuronskim mrežama. Predstavljena su teorijska razmatranja prilagođavanja težina neuronske mreže te matematički modeli računanja derivacija. Njegova široka primjenjivost na različite kategorije dinamičkih sustava omogućuje njegovu široku primjenu. Predstavljene su ključne jednadžbe za sam algoritam s različitim stupnjem kompleksnosti.

Reference najčešće korištene za izradu ovog rada su [1] i [11]. One predstavljaju pristup uporabe neuronskih mreža za optimalno upravljanje nelinearnim sustavima. Najprije je definiran nelinearni sustav s poremećajima u prostoru stanju. Zatim je predstavljen općenit model jednoslojne neuronske mreže koja za izlaz ima zakon upravljanja, a funkcijom cilja rješava min-max problem optimizacije. Predstavljena je i upotreba algoritma povratnog širenja kroz vrijeme zasnovanog na konjugiranom gradijentu. Primjena Adamsove metode diskretizacije za rješavanje diferencijalnih jednadžbi u kombinaciji s automatskom diferencijacijom je ono što ovaj pristup čini učinkovitiji. Teorijska razmatranja su popraćena s numeričkim primjerima i simulacijama na dinamičkom modelu robotskog manipulatora. Rezultati su pokazali povoljne osobine algoritma u smislu točnosti i dosljednosti numeričke stabilnosti. Dobiveni algoritam se lako može primijeniti na sustave višeg reda veće složenosti.

U [10] je detaljnije opisana sama konjugirano gradijentna metoda koja konvergira globalno pod uvjetom da zadovoljava standardne Wolfove uvjete. Zbog računske zahtjevnosti standardnog računanja koraka učenja uveden je SuperSAB algoritam. U [9] je prikazano kako može postići bržu konvergenciju ne utječući pritom previše na stabilnost. Također je pokazno i da je algoritam neosjetljiv na izbor vrijednosti parametara.

2. LTI SUSTAVI

Za opisivanje fizikalnih sustava na način da matematički modeliramo vezu između ulaza i izlaza istih pomoću fizikalnih zakona razlikujemo dva pristupa:

1. prijenosna funkcija (engl. *Transfer function*) - prikazana u frekvencijskoj domeni,
2. stanja sustava (engl. *State space*) - prikazano u vremenskoj domeni.

Linearni vremenski-invarijabilni kontinuirani dinamički sustavi se nalaze, kako im i ime kaže, u vremenskoj domeni stanja sustava. Za sustav kažemo da je vremenski invarijantan, ako veza između ulaz i izlaz ne ovisi o vremenu. Možemo ga prikazati sljedećim linearnim diferencijalnim jednadžbama[2]:

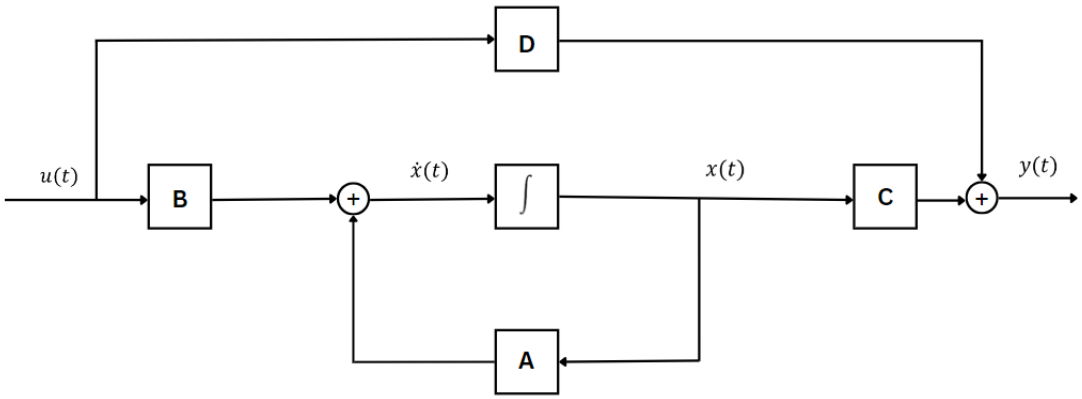
$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t), \quad x(t_0) = x_0 \tag{2.1}$$

$$\mathrm{y}(t) = \mathbf{C}x(t) + \mathbf{D}u(t) \tag{2.2}$$

gdje su:

- $x(t) \in \mathbb{R}^n$ - vektor stanja ,
- $u(t) \in \mathbb{R}^m$ - vektor upravljanja,
- $y(t) \in \mathbb{R}^p$ - vektor izlaza,
- $x_0 \in \mathbb{R}^n$ - vektor početnih stanja sustava
- $\mathbf{A} \in \mathbb{R}^{n\times n}$ - matrica koeficijenata sustava,
- $\mathbf{B} \in \mathbb{R}^{n\times m}$ - matrica ulaza sustava,
- $\mathbf{C} \in \mathbb{R}^{p\times n}$ - matrica izlaza sustava,
- $\mathbf{D} \in \mathbb{R}^{p\times m}$ - matrica prijenosa sustava.

Ukoliko matrice **A, B, C** i **D** vremenski ovisne govorimo o vremenski varijantnim sustavima no fokus u ovom radu će biti na vremenski invarijantne sustave, a blokovski prikaz jednog takvog sustava je dan na slici 2.1.



Slika 2.1 Blokovski prikaz LTI susta

Jednadžbe (2.1) i (2.2) se nazivaju jednadžba stanja i jednadžba izlaza. Autonomni ili homogeni linearni sustav se dobije kada je $u(t) = 0$, a prikazujemo ga sljedećim jednadžbama:

$$\dot{x}(t) = \mathbf{A}x(t)\,, \quad x(t_0) = x_0 \tag{2.3}$$

$$\mathrm{y}(t) = \mathbf{C}x(t)\,. \tag{2.4}$$

Poznavanjem početnog stanja sustava u $x(0)$ i prijelazne (fundamentalne) matrice $e^{\boldsymbol{At}}$ možemo odrediti stanje sustava u bilo kojem kasnijem trenutku $x(t)$.[2]

Rješenje nehomogenog linearno invarijantnog neprekidnog sustava dano je s jednadžbama:

$$\mathrm{x}(t) = e^{\boldsymbol{A(t-t_o)}}x_0 + \int_{\boldsymbol{t_o}}^{t} e^{\boldsymbol{A(t-s)}}\mathbf{B}u(s)\,ds \tag{2.5}$$

$$\mathrm{y}(t) = \mathbf{C}e^{\boldsymbol{A(t-t_o)}}x_0 + \int_{\boldsymbol{t_o}}^{t} \mathbf{C}e^{\boldsymbol{A(t-s)}}\mathbf{B}u(s)\,ds + \mathbf{D}u(t). \tag{2.6}$$

Nakon definiranih jednadžbi za prostor stanja još ćemo definirati i matricu prijenosnih funkcija $G(s) = Y(s)/U(s)$. Ona povezuje ulaz i izlaz nekog sustava te nam daje informacije o dinamičkom ponašanju sustava. Primjenom Laplaceove transformacije na jednadžbe (2.1) i (2.2) se dobije:

$$s\mathbf{X}(s) - x_0 = \mathbf{A}\mathbf{X}(s) + \mathbf{B}\mathbf{U}(s) \tag{2.7}$$

$$\mathbf{Y}(s) = \mathbf{C}\mathbf{X}(s) + \mathbf{D}\mathbf{U}(s)\ . \tag{2.8}$$

Daljnjim sređivanjem se dobije:

$$\mathbf{X}(s) = \mathbf{R}(s)x_0 + \mathbf{R}(\mathrm{s})\mathbf{B}\mathbf{U}(s) \tag{2.9}$$

$$\begin{aligned}\mathbf{Y}(s) &= \mathbf{C}\big(\mathbf{R}(s)x_0 + \mathbf{R}(\mathrm{s})\mathbf{B}\mathbf{U}(s)\big) + \mathbf{D}\mathbf{U}(s)\\ &= \mathbf{C}\,\mathbf{R}(s)x_0 + \mathbf{G}(s)\mathbf{U}(s)\ \ .\end{aligned} \tag{2.10}$$

Matrica **R***(s)* se naziva rezolventa, a matrica **G***(s)* se naziva matrica prijenosnih funkcija. Definirane su jednadžbama:

$$\mathbf{R}(s) = (s\mathbf{I} - \mathbf{A})^{-1} \tag{2.11}$$

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\ + \mathbf{D} \tag{2.12}$$

Stabilnost LTI multivarijablnih sustava će ovisiti o polovima matrice prijenosnih funkcija, ako su polovi matrice **G***(s)*, odnosno svojstvene vrijednosti matrice **A**, smješteni u lijevoj polovini kompleksne s-ravnine sustav je stabilan.

2.1. Upravljivost

Jedno od glavnih pitanja prije početka sinteze regulacijskih sustava je pitanje upravljivosti. Ono što nas zanima je mogućnost da zadani sustav prebacimo iz proizvoljnog početnog stanja u proizvoljno konačno stanje.

Sustav definiran s (2.1) i (2.2) je potpuno upravljiv u zatvorenom vremenskom intervalu $t\epsilon[t_0, t_1]$, ako je moguće za zadane t_0 i t_1 svako početno stanje $x(t_o)$ prevesti u svako željeno konačno stanje preko vektora upravljanja $u(t)$ u konačnom vremenskom intervalu.[2] Također je potrebno dobiti i neki kriterij na osnovu kojeg ćemo moći definirati samu upravljivost. Linearni kontinuirani vremenski-invarijantnih sustav je reprezentiran jednadžbama stanja:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t). \tag{2.13}$$

Pomoću (2.5) možemo odrediti vektor stanja u vremenskom trenutku t_1:

$$\mathrm{x}(t_1) = e^{\boldsymbol{At_1}} x_0 + \int_{\mathbf{0}}^{t_1} e^{\boldsymbol{A}(t_1-\boldsymbol{\tau})} \mathbf{B} u(\tau)\, d\tau. \tag{2.14}$$

Množenjem prethodnog izraza s $e^{-\boldsymbol{At_1}}$ i primjenom Cayley-Hamiltonovog teorema [2] na istu tu matričnu funkciju slijedi njen razvoj u polinom reda $n-1$ po matrici **A** :

$$e^{-\boldsymbol{At_1}}\mathrm{x}(t_1) - x(0) = \sum_{k=1}^{m}\sum_{j=0}^{n-1} \mathbf{A}^j b_k \int_{\mathbf{0}}^{t_1} \alpha_j(\tau) u_k(\tau)\, d\tau. \tag{2.15}$$

Integral na lijevoj strani možemo označiti s v_{jk} od prethodnog izraza nam preostane:

$$e^{-\boldsymbol{At}_1}\mathrm{x}(t_1) - x(0) = \sum_{k=1}^{m}\sum_{j=0}^{n-1} v_{jk}\mathbf{A}^j b_k, \tag{2.16}$$

$$j = 0,1,\dots,n-1;\ k = 1,2,\dots,m.$$

Budući da su početno i konačno stanje proizvoljni, dobiveni izraz interpretiramo kao proizvoljnu dekompoziciju n-dimenzionalnog vektora $e^{-\boldsymbol{At}_1}\mathrm{x}(t_1) - x(0)$ po vektorima $\mathbf{A}^j b_k$. Kako bi to bilo moguće *n* vektora $\mathbf{A}^j b_k$ mora biti linearno nezavisno odnosno:

$$rank(\mathbf{E}) = rank[\mathbf{A^{n-1}B} \quad \mathbf{A^{n-2}B} \quad \dots \quad \mathbf{A^2B} \quad \mathbf{AB} \quad \mathbf{B}] = n. \tag{2.17}$$

Ovo je ujedno i uvjet potpune upravljivosti stanja linearnih kontinuiranih sustava.

2.2. Osmotrivost

Potreba za rekonstrukcijom varijabli stanja, koje ne moraju biti uvijek mjerljive, na temelju mjerenja izlaznih varijabli sustava je još jedna od praksi prilikom upravljanja sustavima. Ona će biti moguća samo ako sustav ima svojstvo osmotrivosti ili mjerljivosti.

Sustav je potpuno osmotriv po stanjima (uz $u = 0$) u zatvorenom vremenskom intervalu $t\epsilon[t_0, t_1]$,, ako za zadani $t_0\ i\ t_1$ svako početno stanje $x(t_o)$ možemo egzaktno odrediti na osnovu poznavanja vektora izlaza $y(t)$ u tom istom intervalu [3]. Kao i za slučaj upravljivosti trebamo dobiti kriterij na osnovu kojeg ćemo provjeravati osmotrivost sustava. U ovom slučaju se promatra homogeni linearni sustav opisan matričnim sustavom diferencijalnih jednadžbi:

$$\dot{x}(t) = \mathbf{A}x(t)\,, \tag{2.18}$$

$$\mathrm{y}(t) = \mathbf{C}x(t). \tag{2.19}$$

Rješenje gornjeg sustava je definirano izrazom

$$\mathrm{x}(t) = e^{\mathbf{A}t}x(0), \tag{2.20}$$

a izlazna varijabla je

$$\mathrm{y}(t) = \mathbf{C}e^{\mathbf{A}t}x(0). \tag{2.21}$$

Cayley-Hamiltonov teorem će biti primijenjen na matričnu funkciju u svrhu razvijanja polinoma reda $n-1$ po matrici **A**:

$$\mathrm{y}(t) = \sum_{k=0}^{n-1} \alpha_k(t)\mathbf{C}\mathbf{A}^k\, x(0), \tag{2.22}$$

Prošireni matrični zapis gornje jednadžbe za *N* mjerenja izlaznog vektora u *N* različitih vremenskih trenutaka izgleda:

$$\begin{bmatrix} y(t_1) \\ y(t_2) \\ \vdots \\ y(t_N) \end{bmatrix} = \begin{bmatrix} \alpha_0(t_1)\mathbf{I} & \alpha_1(t_1)\mathbf{I} & \cdots & \alpha_{n-1}(t_1)\mathbf{I} \\ \alpha_0(t_2)\mathbf{I} & \alpha_1(t_2)\mathbf{I} & & \alpha_{n-1}(t_2)\mathbf{I} \\ \vdots & & \ddots & \vdots \\ \alpha_0(t_N)\mathbf{I} & \alpha_1(t_N)\mathbf{I} & \cdots & \alpha_{n-1}(t_N)\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathrm{C} \\ \mathrm{CA} \\ \mathrm{CA}^2 \\ \vdots \\ \mathrm{CA}^{n-1} \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_2(0) \\ \vdots \\ x_n(0) \end{bmatrix} \tag{2.23}$$

$$Np \times np \qquad\qquad np \times n$$

Kako bi se zadovoljila linearna nezavisnost jednadžbi u osnovu na zadanu dimenziju matrica i mogli odrediti početni uvjeti , druga matrica na desnoj strani mora imati rang *n*.

$$rank(\mathbf{H}) = rank\left[\mathrm{C}^T\mathbf{A}^{\boldsymbol{T}}\mathrm{C}^T \ldots \mathbf{A}^{2T}\mathrm{C}^T\mathbf{A}^{(n-1)T}\mathrm{C}^T\right] = n \tag{2.24}$$

Ovo je ujedno i uvjet potpune osmotrivosti stanja linearnih kontinuiranih sustava.

2.3. Stabilnost i Lyapunovljeva teorija stabilnosti

Pod općim pojmom stabilnosti podrazumijevamo nešto što u odsustvu poremećaja vraća u svoj ravnotežni položaj. Kada pričamo o sustavima i njihovoj stabilnosti to podrazumijeva da

rješenja sustava teži ravnotežnom stanju kada vrijeme teži u beskonačnost. Kod dinamičkih sustava problem stabilnost podrazumijeva traženje asimptotski rješenja diferencijalnih jednadžbi [3]. Dinamički sustav možemo prikazati jednadžbom:

$$\dot{x}(t) = \mathbf{f}(x)\,. \tag{2.25}$$

Funkcija $\mathbf{f} \in \mathbb{R}^n$ je nelinearna kontinuirana vektorska funkcija, a $x \in \mathbb{R}^n$ je vektor stanja sustava.

Za ravnotežno stanje $x = 0$ kažemo da je stabilno ako za neki $\varepsilon > 0$ postoji $\delta > 0$ tako da iz $\|x(t_0)\| < \delta$, slijedi $\|x(t)\| < \varepsilon$ za sve $t \geq t_0$. Kako bi ono bilo asimptotski stabilno mora zadovoljavat i dodatni uvjet da za neki $\delta > 0$ i za $\|x(t)\| < \delta$ slijedi da $x(t) \to 0\ kada\ t \to \infty$.[2]

Sada kad je definirana stabilnost preostaje nam samo da definiramo Lyapunovljev kriterij stabilnosti. Ako imamo skalarnu funkciju $V(x)$ sa kontinuiranim parcijalnim derivacijama prvog reda tako da vrijedi:

- $V(x)$ je pozitivno definitna ($V(x) > 0$ za $x > 0$; $V(x)=0$ za $x = 0$)
- $\dot{V}(x)$ je negativno semidefinitna ($\dot{V}(x) \leq 0\ za\ x > 0; \dot{V}(x) = 0\ za\ x = 0$)
- $V(x) \to \infty$ kada $\|x(t)\| \to \infty$ ($V(x)$ je radijalno neograničena).

Onda je ravnotežno stanje $x = 0$ globalno stabilno. Funkcija koja zadovoljava samo prva dva uvjeta je lokalno stabilna, a ona koja zadovoljava sva tri uvjeta je globalno stabilna. Gornji uvjet za stabilnost će biti zadovoljen ukoliko sve svojstvene vrijednosti matrice **A** imaju pozitivne realne vrijednosti tj. nalaze se u lijevo dijelu kompleksne ravnine [3].

Ako definiramo pozitivno definitnu funkciju *V(t)* kao:

$$V(t) = x^T(t)\mathbf{P}x(t), \qquad \mathbf{P} = \mathbf{P^T} > 0. \tag{2.26}$$

Deriviranjem gornjeg izraza po vremenu se dobije:

$$\dot{V}(t) = \dot{x}^T(t)\mathbf{P}x(t) + x^T(t)\mathbf{P}\dot{x}(t), \tag{2.27}$$

uvrštavanjem (**2.18**) u (**2.27**) dobivamo:

$$\dot{V}(t) = x^T(t)[\mathbf{A^T P + PA}]x(t). \tag{2.28}$$

Za stabilne sustave $\dot{V}(t)$ je negativno definitna funkcija stoga:

$$[\mathbf{A^T P + PA}] = -\mathbf{Q}, \quad \mathbf{Q = Q^T} > 0. \tag{2.29}$$

Prethodnu jednadžbu zovemo Lyapunovljeva jednadžba stabilnosti. Ukoliko $\mathbf{Q = Q^T} > 0$ i sve svojstvene vrijednosti matrice **A** imaju negativne realne vrijednosti Lyapunovljeva jednadžba ima jedinstveno rješenje za matricu **P**:

$$\mathbf{P} = \int_{\mathbf{0}}^{\infty} e^{\mathbf{A}^T \cdot t}\mathbf{Q}e^{\mathbf{A}t}dt \ , \quad \mathbf{P = P^T} > 0. \tag{2.30}$$

3. REGULACIJA I OPTIMIZACIJA POVRATNE VEZE STANJA

U ovom poglavlju ćemo za početak razmatrati konstrukciju povratne veze za SISO (eng. *Single input/output system*) sustav, a zatim će se izvest potrebni uvjeti za optimalnu kontrolu te ih upotrijebiti za razvoj linearne kvadratne teorije upravljanja.

3.1. Utjecaj povratne veze na nule i polove sustava

Promotrimo zadani sustav stanja zadan na sljedeći način:

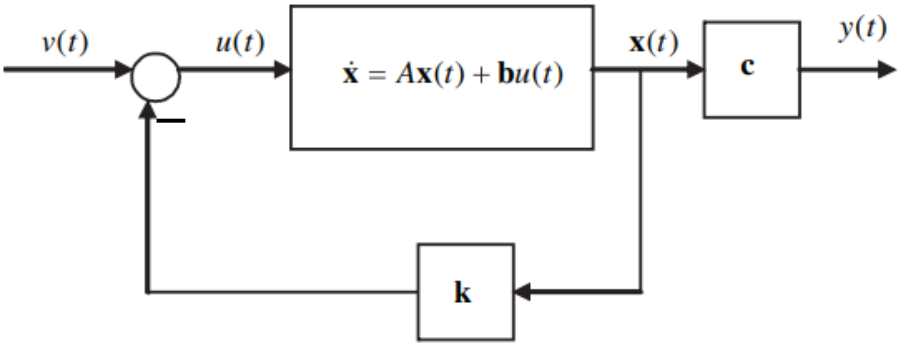
$$\frac{dx}{dt} = \mathbf{A}x(t) + \mathbf{b}u(t), \tag{3.1}$$

$$y(t) = \mathbf{c}x(t). \tag{3.2}$$

Prijenosna funkcija otvorenog kruga dana je s:

$$\frac{y(s)}{u(s)} = \mathbf{c}(sI - \mathbf{A})^{-1}\mathbf{b} = \frac{\mathbf{c}Adj(sI - \mathbf{A})\mathbf{b}}{\det(sI - \mathbf{A})}. \tag{3.3}$$

Sustav povratne veze varijable stanja prikazan je blok dijagramom na slici 3.1,



Slika 3.1 Sustav kontrole stanja s povratnom spregom [4]

u kojem je zakon upravljanja opisan idućom jednadžbom:

$$u(t) = v(t) - \mathbf{k}x(t) \tag{3.4}$$

gdje *v(t)* predstavlja vanjski ulaz, a **k** predstavlja vektor stanja povratne veze. Uvrštavanjem (3.4) u (3.1) dobivamo jednadžbu koja predstavlja dinamiku zatvorenog kruga povratne veze vektora stanja. Uvrštavanjem (3.4) u (3.1) se dobije:

$$\frac{dx}{dt} = (\mathbf{A} - \mathbf{bk})x(t) + \mathbf{b}v(t). \tag{3.5}$$

Prijenosna funkcija zatvorene petlje je:

$$\frac{y(s)}{v(s)} = \mathbf{c}(s\mathbf{I} - \mathbf{A} + \mathbf{bk})^{-1}\mathbf{b} = \frac{\mathbf{c}Adj(s\mathbf{I} - \mathbf{A} + \mathbf{bk})\mathbf{b}}{\det(s\mathbf{I} - \mathbf{A} + \mathbf{bk})}. \tag{3.6}$$

Ono što je bitno kod korištenja povratne veze je njezin utjecaj na nule i polove samog sustava jer oni su ti koji određuju tj. opisuju dinamiku samog sustava. Prema [4] polovi povratne veze zatvoreno kruga mogu biti smješteni bilo gdje u s-ravnini (kompleksnoj ravnini koja prikazuje grafički prikaz Laplaceove transformacije prijenosne funkcije) pod uvjetom da je stanje sustava upravljivo. Što zapravo znači da će uvijek postojati mogućnost pronaći **k** za svaku poziciju polova povratne veze zatvorenog kruga.

Željeni karakteristični polinom zatvorenog sustava (3.6) je:

$$p(s) = \det(s\mathbf{I} - \mathbf{A} + \mathbf{bk}) = s^n + p_1 s^{n-1} + p_2 s^{n-2} + \cdots + p_{n-1}s + p_n, \tag{3.7}$$

gdje su $p_1, p_2, \cdots, p_n$ su proizvoljni realni brojevi. Na isti način dobijemo i karakteristični polinom otvorenog kruga:

$$q(s) = \det(s\mathbf{I} - \mathbf{A}) = s^n + q_1 s^{n-1} + q_2 s^{n-2} + \cdots + q_{n-1}s + q_n. \tag{3.8}$$

Matričnim zapisom možemo polinom iz (3.7) pomoću (3.8) zapisati kao:

$$\mathbf{p} - \mathbf{q} = \mathbf{kCU_t}, \tag{3.9}$$

gdje je **C** upravljiva matrica, **p** i **q** su vektori koeficijenata, a $\mathbf{U_t}$ je trokutna nesingularna matrica oblika:

$$\mathbf{U_t} = \begin{bmatrix} 1 & q_1 & q_2 & \cdots & & q_{n-1} \\ 0 & 1 & q_1 & q_2 & \cdots & q_{n-2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & & 0 & 1 \end{bmatrix}. \tag{3.10}$$

Kako bi polovi zatvorenog sustava odgovarali željenim položajima, osiguravajući željene performanse i stabilne karakteristike u sustavu upravljanja, koristi se *Bass-Gurra* formula (3.9) koja poveziva ta dva polinoma:

$$\mathbf{k} = (\mathbf{p} - \mathbf{q})\mathbf{U}_\mathbf{t}^{-\mathbf{1}}\mathbf{C}^{-\mathbf{1}}. \tag{3.11}$$

Ona omogućuje da se u povratnoj vezi kontroliraju vlastite vrijednosti sustava, kako bi se dobile željene karakteristike. Za detaljan izvode gornje formule pogledati [4].

Pod uvjetom da je upravljiv, sustava jednadžbi (3.1) i (3.2) možemo prevesti u kanonsku formu kontrolera $\{\mathbf{A}_\mathbf{c}, \mathbf{b}_c, \mathbf{c}_c\}$. Matrica sustava zadane povratne veze je

$$\mathbf{A}_\mathbf{c} - \mathbf{b}_c\mathbf{k} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_n - k_1 & -a_{n-1} - k_2 & \dots & & -a_1 - k_n \end{bmatrix}. \tag{3.12}$$

Što ne mijenja kanonsku formu te njezinu prijenosnu funkciju možemo zapisati kao:

$$\mathbf{c}_c(s\mathbf{I} - \mathbf{A}_\mathbf{c} + \mathbf{b}_c\mathbf{k})^{-1}\mathbf{b}_c = \frac{b_1 s^{n-1} + \cdots + b_{n-1}s + b_n}{s^n + (q_1 + k_n)s^{n-1} + \cdots + (q_n + k_1)}. \tag{3.13}$$

Prema gornjoj jednadžbi je vidljivo da brojnik ostaje ne promijenjen što znači da povratna veza nema nikakav utjecaj na nule prijenosne funkcije [4].

3.2. Osnove teorije optimalnog upravljanja

Za potrebe razrade osnova optimalnog upravljanja uzet će se općeniti dinamički sustav *n*-tog reda:

$$\frac{dx}{dt} = f\big(x(t), u(t)\big); \quad x(0) = x_o \tag{3.14}$$

gdje su $x(t)$ *i* $u(t)$ *n*-dimenzonalni vektor stanja i *m*-dimenzionlani vektor ulaza. Cilj je pronaći takav $u(t)$ u vremenskom intervalu $0 \leq t \leq t_f$ da se funkcija cilja minimizira ili maksimizira [4]:

$$I\big(u(t)\big) = G\left(x\big(t_f\big)\right) + \int_0^{t_f} F(x,u)\, dt. \tag{3.15}$$

Jednadžbe stanja služe kao ograničenja za optimizaciju funkcije *I*, a dodatno će se primijeniti ograničenja na varijable upravljanja tj. ulaza:

$$u_{min} \leq u_i \leq u_{max}. \tag{3.16}$$

Prema [4] ukoliko je $u^*(t)$ kandidat za optimalni vektor ulaza i $x^*((t)$ je odgovarajući vektor stanja, kako bi se utvrdilo da je to uistinu optimalno rješenje za dani sustav dodat ćemo mu malu vrijednost $\delta u(t)$:

$$u(t) = u^*(t) + \delta u(t). \tag{3.17}$$

Promjene u funkciji cilja i rješenju s u sljedeće:

$$\delta I = I\big(u^*(t) + \delta u(t)\big) - I(u^*(t)) \tag{3.18}$$

$$\frac{d(x^*(t) + \delta x(t))}{dt} = f\big((x^*(t) + \delta x(t), u^*(t) + \delta u(t)\big). \tag{3.19}$$

Linearizacijom jednadžbe (3.19) se dobije:

$$\frac{d(\delta x)}{dt} = \left(\frac{\partial f}{\partial \mathrm{x}}\right)\delta x(t) + \left(\frac{\partial f}{\partial \mathrm{u}}\right)\delta u(t). \tag{3.20}$$

Nadalje, množimo jednadžbu s proizvoljnim *n*-dimenzionalnim vektorom $\boldsymbol{\lambda}^T$ i integracijom od 0 do t_f te uvrštavanjem dobivenog u (3.18) :

$$\begin{aligned}\delta I = &\left[F(t_f) + \left(\frac{\partial \mathrm{G}}{\partial \mathrm{x}}\right)f(t_f)\right]\delta t_f + \boldsymbol{\lambda}^T(0)\delta x(0) + \left[\left(\frac{\partial \mathrm{G}}{\partial \mathrm{x}}\right)\boldsymbol{\lambda}^T(t_f)\right]\delta x(t_f) \\ &+ \int_0^{t_f}\left[\left(\frac{\partial \mathrm{H}}{\partial \mathrm{x}}\right)\delta x + \left(\frac{\partial \mathrm{H}}{\partial \mathrm{u}}\right)\delta u(t) + \frac{d\boldsymbol{\lambda}^T}{dt}\delta x\right]dt\end{aligned} \tag{3.21}$$

gdje je funkcija *H* poznata kao Hamiltonova funkcija:

$$H = F(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^T f(\boldsymbol{x}, \boldsymbol{u}). \tag{3.22}$$

Također $\boldsymbol{\lambda}(t)$ zadovoljava izabrani uvjet:

$$\frac{d\boldsymbol{\lambda}^T}{dt} = -\left(\frac{\partial \mathrm{H}}{\partial \mathrm{x}}\right). \tag{3.23}$$

Članovi izvan integrala u (3.21) poznati su kao termini rubnih uvjeta, koji se uklanjaju zbog navedenog problema ($\delta t_f = \delta x = 0$ za poznate vrijednosti t_f *i* $x(0)$). Isto tako ostatak integrala nestane uz uvjet $\boldsymbol{\lambda}^T(t_f) = \left(\frac{\partial \mathrm{G}}{\partial \mathrm{x}}\right)_{t=t_f}$. Tako jednadžbu (3.21) možemo pisati kao:

$$\delta I = \int_o^{t_f} \left(\frac{\partial \mathrm{H}}{\partial \mathbf{u}}\right) \delta \boldsymbol{u} dt. \tag{3.24}$$

Optimalno rješenje $u_0(t)$ mora onda za minimizacijski problem zadovoljit uvjet:

$$\delta I \geq 0 \; za \; svaki \; \delta \boldsymbol{u}(\boldsymbol{t}). \tag{3.25}$$

U skladu s nametnutim uvjetima za ulaznu varijablu (3.16):

$$ako \; je \; u_{io} = u_{i,\max} \; , \quad \left(\frac{\partial \mathrm{H}}{\partial u_i}\right) \leq 0. \tag{3.26}$$

$$ako \; je \; u_{io} = u_{i,\min} \; , \quad \left(\frac{\partial \mathrm{H}}{\partial u_i}\right) \geq 0. \tag{3.27}$$

Zbog konciznosti rada neki koraci u izvodima su preskočeni te se treba referirati na [3] za detaljne izvode.

3.3. LQR – Linearno kvadratni optimizacijski regulator

Prilikom stabilizacije linearnog kontrolnog sustava nije uvijek dovoljno pronaći matricu **K** tako da matrica (**A-BK**) bude stabilna već postoji potreba za podešavanjem svojstvenih vrijednosti iste. Taj problem se zove problem podešavanja polova ili problem dodjeljivanja svojstvenih vrijednosti.

U praksi, zbog nedostatka informacija gdje postaviti svojstvene vrijednosti, često se napamet moraju pogađati u svrhu dobivanja željenih rezultata. Kako bi izbjegli takav način rada, promatraju su različite performanse sustava i na osnovu njih stabiliziramo regulacijski sustav. Jedna od takvih metoda je LQR metoda ili metoda linearnog kvadratnog optimizacijskog sustava.

Cilj je dovesti vektor stanja $\boldsymbol{x}(t)$ u ishodište stanja sustava (nulti vektor stanja) iz bilo koje nenultne pozicije sustava. Ukoliko koristimo zakon upravljanja povratne veze, $x(t)$ će poprimiti to stanje samo ako su polovi zatvorene petlje smješteni daleko u lijevoj polovici s-ravnine. Međutim elementi vektora pojačanja mogu poprimit velike vrijednosti što će direktno utjecat na povećanje funkcije cilja. S druge strane, ako su polovi zatvorenog kruga smješteni blizu

polova otvorenog kruga brzina propadanja vektora $\boldsymbol{x}(t)$ će se povećati i neće postojat potreba za velikim upravljanjem. S obzirom na tu činjenicu podešavanje polova postaje kompromis između brzine propadanja $\boldsymbol{x}(t)$ i veličine upravljačkog ulaza [4].

S obzirom na linearni sustav:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t), \quad x(0) = x_0 \tag{3.28}$$

zadatak LQR regulator je pronaći signal $u(t) = -\mathbf{K}x(t)$ na način da se minimizira kvadrat funkcije cilja dan jednadžbom:

$$J_C(x) = \int_0^\infty [x(t)^T \mathbf{Q} x(t) + u(t)^T \mathbf{R} u(t)] dt. \tag{3.29}$$

Za zadani problem pretpostavlja se za matricu **Q** da je simetrična i pozitivno semidefinitna**,** a za matricu **R** da je simetrična i pozitivno definitna. Kvadratna forma $x^T \mathbf{Q}\boldsymbol{x}$ označava devijaciju stanja *x* od početnog, dok $u^T \mathbf{R} u$ predstavlja cijenu ulaznog signala.

Zadani sustav (3.28) se rješava pomoću vremenski neprekidnih algebarskih Riccatijevih jednadžbi (CARE):

$$\begin{gathered} \mathbf{X}\mathbf{A} + \mathbf{A}^{\mathrm{T}}\mathbf{X} + \boldsymbol{Q} - \boldsymbol{X}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathrm{T}}\mathbf{X} = \mathbf{0} \\ \mathbf{X}\mathbf{A} + \mathbf{A}^{\mathrm{T}}\mathbf{X} + \boldsymbol{Q} - \boldsymbol{X}\mathbf{S}\mathbf{X} = \mathbf{0}, \qquad \mathbf{S} = \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathrm{T}}. \end{gathered} \tag{3.30}$$

LQR teorem glasi [5]:

Pretpostavimo da se par (**A, B**) *može stabilizirati te da je par* (**A, Q**) *detektabilan. Tada postoji jedinstveno optimalno upravljanje* $u_{op}(t)$ *koje minimizira* $J_G(x)$*. Vektor* $u_{op}(t)$ *je dan* $u_{op}(t) = -\mathbf{K}x(t)$*, gdje je* $\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^{\mathbf{T}}\mathbf{X}$**.** **X** *je jedinstvena pozitivno semidefinitna matrica, koja je ujedno rješenje CARE od* (*3.28*)*. Matrica zatvorene petlje* (**A** − **BK**) *je stabilna, a minimaln vrijednost od* $J_G(x)$ *je* $x_{op}^T \mathbf{X} x_{op}$*, gdje je* $x_{op} = x(0)$*.*

Deriviranjem rješenja dobivamo:

$$\begin{gathered} \frac{d}{dt}({x_0}^T \mathbf{X} x_0) = (u^T + x^T \boldsymbol{X}\boldsymbol{B}\boldsymbol{R}^{-1})\mathbf{R}(u + \boldsymbol{R}^{-1}\boldsymbol{B}^{\boldsymbol{T}}\mathbf{X}x) - (x^T \mathbf{Q}\boldsymbol{x} + \\ u^T \mathbf{R} u)\mathbf{.} \end{gathered} \tag{3.31}$$

Daljnjim sređivanjem i integriranjem po *t* na intervalu [0, T] dobivamo:

$$\int_0^T (x^T \mathbf{Q}\boldsymbol{x} + u^T \mathbf{R}u)\, dt = -x^T(T)\, \boldsymbol{X}\boldsymbol{x}(\boldsymbol{t}) + {x_0}^T \mathbf{X} x_0 + \int_0^T (u + \boldsymbol{R^{-1}B^T}\mathbf{X}x)^{\boldsymbol{T}}\mathbf{R}(u + \boldsymbol{R^{-1}B^T}\mathbf{X}x)\, dt. \tag{3.32}$$

Uzimajući u obzir da $\mathbf{X} = \mathbf{X^T} \geq \mathbf{0}$ i $\mathbf{R} = \mathbf{R^T} > \mathbf{0}$, kada $T \to \infty$ onda će i $x(T) \to \mathbf{0}$**.** Nadalje se dobiva:

$$J_C(x) = {x_0}^T \mathbf{X} x_0 + \int_0^\infty (u + \boldsymbol{R^{-1}B^T}\mathbf{X}x)^{\boldsymbol{T}}\mathbf{R}(u + \boldsymbol{R^{-1}B^T}\mathbf{X}x). \tag{3.33}$$

Budući da je **R** simetrična i pozitivno semidefinitna, dobivamo da za svaki *u* i svaki x_0 funkcija cilja mora biti $J_C(x) \geq {x_0}^T \mathbf{X} x_0$. Ona ne ovisi o *u* pa njezina minimalna vrijednost iznosi:

$$J_C(x) = {x_0}^T \mathbf{X} x_0, \tag{3.34}$$

a postiže se za upravljački signal:

$$u_0(t) = -\boldsymbol{R^{-1}B^T}\mathbf{X}x(t) = -\mathbf{K}x(t). \tag{3.35}$$

Ovim smo dokazali jedinstvenu optimalnu upravljivost uz pretpostavku da rješenje CARE postoji.

4. PRIMJENA NEURONSKIH MREŽA U TEORIJI UPRAVLJANJA

U prethodnim poglavljima obrađeni su neki osnovni pojmovi i jednadžbe, vezane za LTI sustave i teoriju optimalnog upravljanja s pomoću LQR regulatora, koje će se koristit u samoj formulaciji zadatka za ovaj diplomski rad. U nastavku će se postaviti zadatak diplomskog rada u skladu s istim te će se razraditi sami algoritam upravljanje i njegovi pojedini dijelovi.

Sama uporaba neuronskih mreža je predstavljanja zbog teškoće pronalaska analitičkog rješenja Hamilton-Jacobi-Bellmanove jednadžbe za problem optimalnog upravljanja, odnosno Hamilton-Jacobi-Isaacs jednadžbe u slučaju djelovanja i varijable poremećaja. Kako bi izbjegli problem računskog opterećenja zbog dimenzionalnosti stanja sustava dobivena aproksimiranjem HJI jednadžbi kao u [7], u ovom radu će se koristit direktniji pristup. Upravljačka varijabla je dana kao linearna kombinacija izvorne funkcije u svrhu aproksimiranja iste (pogotovo korisno kod nelinearnih problema). Konjugiranim gradijentom se minimizira funkcija cilja, a njezini gradijenti se računaju primjenom lančanog pravila uređenih derivacija [1]. Ovisnost upravljačke varijable i vektora stanja, opisane jednadžbama sustava, omogućava korištenje strukture algoritma propagiranja unatrag kroz vrijeme, koji je sličan BPTT algoritmu [8].

Cijeli izvod i formulacija problema je izvedena prema [1] i [11] te je prilagođena za minimizirajući problem bez poremećaja na varijablu stanja, a vremenska diskretizacija je provedena Eulerovom metodom..

4.1. Formulacija problema

Kao i do sad promatrat ćemo linearni vremenski invarijantni dinamički sustav u obliku:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t)\ , \quad x(0) = x_0 \tag{4.1}$$

$$z(t) = \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}, \tag{4.2}$$

gdje su:

$x(t) \in \mathbb{R}^{n_0}$ - vektor stanja ,

$u(t) \in \mathbb{R}^{n_u}$ - vektor ulaza,

$x_0 \in \mathbb{R}^{n_0}$ - vektor početnih stanja sustava

$z(t) \in \mathbb{R}^{n_z}$ - vektor kojim će se upravljati (kaznena varijabla).

Također se pretpostavlja da je $u(t)$ ograničen prema u intervalu:

$$u_{min} \leq u(t) \leq u_{max}. \tag{4.3}$$

Isto tako se smatra za sva stanja sustava (4.1) i (4.2) da su upravljiva prema (2.17).

Cilj ovog rada je utvrditi regulator stanja povratne veze u obliku:

$$u(t) = \boldsymbol{\sigma}\big(x(t)\big), \tag{4.4}$$

kada su sva stanja sustava dostupna, tako da se sustav zatvorene petlje sastoji od (4.1) i (4.2) i da je (4.4) stabilan. U predstavljenom zakonu upravljanja $\boldsymbol{\sigma}(\cdot)$ je jednoslojna neuronska mreža sa saturacijskom aktivacijskom funkcijom. Uporabom saturacijske aktivacijske funkcije ograničenje (4.3) na vektor upravljanja biva uvijek zadovoljeno. Zakon upravljanja temeljen na neuronskoj mreži postaje:

$$u(t) = \boldsymbol{\sigma}(x(t), \mathbf{W}), \tag{4.5}$$

gdje **W** predstavlja matricu težina neuronske mreže i:

$$u_j(t) = \boldsymbol{\sigma_j}(\xi_j), \quad j = 1,2,\ldots,n_u, \tag{4.6}$$

$$\xi_j(t) = \sum_{\boldsymbol{k=1}}^{\boldsymbol{n_0}} w_{jk} x_k(t), \tag{4.7}$$

$$\boldsymbol{\sigma_j}\big(\xi_j\big) = \begin{cases} u_{j,max}\,; & \xi_j > u_{j,max} \\ \xi_j\,; & -u_{j,max.} < \xi_j < u_{j,max} \\ u_{j,min}\,; & \xi_j < -u_{j,max.} \end{cases} \tag{4.8}$$

Parametar w_{jk} nam predstavlja sinaptičke težine. Nadalje u radu se razmatra problem pronalaska zakona upravljanja takvog da funkcija cilja, podložna linearnim jednadžbama (4.1) i (4.2), minimizirana za sve $t_f \geq 0$:

$$J = \int_o^{t_f} F(x(t), u(t))dt, \tag{4.9}$$

$$F\big(x(t),u(t)\big) = \|z(t)\|^2_{Q,R} = x^T\mathbf{Q}x + u^T\mathbf{R}u,$$
$$\mathbf{Q}^{\mathrm{T}} = \mathbf{Q} \geq 0, \quad \mathbf{R}^{\mathbf{T}} = \mathbf{R} \geq 0\,. \tag{4.10}$$

Kriterij izvedbe (4.9) zakona upravljanja implicitno ovisi o težinama neuronske mreže w_{jk}. Stoga se iznos same matrice **W,** čiji su elementi same težine w_{jk}, može odrediti prema kriteriju izvedbe (4.9).

U svrhu pojednostavljenja izračuna višeg reda integralnog izraza u funkciji troška (4.9) uvodi se dodatna varijabla stanja. Nova varijabla ima sljedeći oblik:

$$\dot{\chi} = F\big(x(t),u(t)\big), \qquad \chi(0) = 0. \tag{4.11}$$

Početni sustav s dodatnim stanjem (4.11) se može napisati u sljedećem obliku:

$$\frac{d\tilde{x}}{dt} = \tilde{f}\ (\tilde{x}(t),\ u(t)), \quad \tilde{x}(0) = \tilde{x}_0, \tag{4.12}$$

gdje je $\tilde{x}$ novi $n = (n_0 + 1)$-dimenzionalni vektor stanja

$$\tilde{x}(t) = [x_1 \quad x_2 \quad \cdots \quad x_{n_0} \quad \chi]^{\mathrm{T}}, \tag{4.13}$$

i nova funkcija troška :

$$\tilde{f}(\tilde{x}(t),u(t)) = \Big[\big(\mathbf{A}x(t) + \mathbf{B}u(t)\big)^{\boldsymbol{T}} \ F\Big]^T. \tag{4.14}$$

Sada problem vremenski kontinuirane optimizacije postaje:

$$\min_W J\big(t_f\big) = \chi\big(t_f\big)$$
$$\text{podvrgnut (4.13).} \tag{4.15}$$

4.2. Vremenska diskretizacija Eulerovom metodom

Upotreba BPTT algoritma zahtijeva i vremensku diskretizaciju dinamike sustava (4.12). Za potrebe ovog rada koristit ćemo se najjednostavnijom vremenskom diskretizacijom, a to je Eulerova metoda vremenske diskretizacije.

Promatramo vremenski interval $[0, t_f]$ koji je podijeljen na $N-1$ jednakih podintervala.

Tada se vremenski raspon sastoji od konačnog broja točaka $t_f = i\tau$ za $i = 0, 1, 2, \ldots, N-1$

gdje je duljina vremenskog koraka $\tau = t_f/N$.

Eulerova aproksimacija jednadžbe stanja sustava (4.12) u kontinuiranom vremenu je dana s:

$$\tilde{x}(i+1) = \tilde{x}(i) + \tau\tilde{f}(i), \qquad \tilde{x}(0) = \tilde{x}_0, \tag{4.16}$$

gdje je $\tilde{f}(i) = \tilde{f}\ (\tilde{x}(t),\ u(t))$ na cijelom intervalu $0, 1, 2, \ldots, N-1$.

```
48
49      #Eulerova diskretizacija
50      def Update_xtilda(self, xtilda, u, tau, sus_func, F):
51          x_dot = sus_func(xtilda[:-1, :], u)
52          ix_dot = F(xtilda[:-1, :], u)
53
54          xtilda_dot = np.vstack([x_dot, ix_dot])
55          xtilda = xtilda + xtilda_dot * tau
56          return xtilda
```

Slika 4.1 Python kod za Eulerovu diskretizaciju

Na slici 4.1. je prikazan kod koji implementira Eulerovu diskretizaciju za ažuriranje stanja sustava. Funkcija *Update_xtilda* prima trenutno stanje *xtilda*, ulaz *u*, vremenski korak *tau*, funkciju sustava *sus_func* koja izračunava prvu derivaciju stanja, i funkciju *F* koja izračunava drugu derivaciju stanja. Izračunate derivacije se spajaju u *xtilda_dot*, a konačno novo stanje *xtilda* se ažurira koristeći Eulerovu diskretizaciju, gdje se trenutno stanje povećava za derivaciju pomnoženu vremenskim korakom *tau*.

Diskretizacija jednadžbi (4.6) i (4.7) je dana s:

$$u_j(\mathrm{i}) = \boldsymbol{\sigma_j}(\xi_j(\boldsymbol{i}), \tag{4.17}$$

$$\xi_j(i) = \sum_{\boldsymbol{s=1}}^{\boldsymbol{n_0}} w_{js} x_s(t), \tag{4.18}$$

za $j = 1, 2, \ldots, n_u$.

Diskretizacija funkcije cilja će shodno tom imati oblik:

$$J = \chi(N). \tag{4.19}$$

Za kraj naš problem optimalnog upravljanja je dan kao:

$$\min_{W} J = \chi(N) \tag{4.20}$$

te podliježe (4.16). Zbog problema poput netočnosti i nestabilnosti, koji se javljaju prilikom korištenja ove metode, posebnu pažnju treba posvetiti prilikom izbora vremenskog koraka. Smanjenjem vremenskog koraka točnost i stabilnost se povećavaju no izvršenje samog algoritma duže traje. Detalji izbora će biti opisani u idućim poglavljima.

4.3. Metoda konjugiranog gradijenta

U ovom poglavlju predstavit će se poboljšani algoritam konjugiranog gradijenta s prilagodbom stope konvergencije koji pruža bolji performans nad standardnom gradijentnom metodom [9]. Izvodi u nastavku su preuzeti iz [11] ukoliko nije navedeno drukčije.

Za početak razmatra se obični gradijenti algoritam za optimalni problem bez ograničenja:

$$\min_{x} f(x), \tag{4.21}$$

gdje se za funkcija troška uzima glatka funkcija $f: \mathbb{R}^n \to \mathbb{R}$ i x je vektor nezavisnih varijabli:

$$x = [x_1 \quad x_2 \quad \cdots \quad x_n]^{\mathrm{T}} \in \mathbb{R}^n. \tag{4.22}$$

Oblik gradijentnog algoritma možemo zapisati kao:

$$x(k+1) = x(k) + \Delta x(k), \tag{4.23}$$

a vektor smjera pretraživanja je:

$$\Delta x(k) = -\eta \nabla f\big(x(k)\big). \tag{4.24}$$

Stopa učenja η je pozitivan broj koji predstavlja koeficijent konvergiranja algoritma i

$$\nabla f\big(x(k)\big) = \left[\frac{\partial f}{\partial x_1(k)} \quad \frac{\partial f}{\partial x_2(k)} \quad \cdots \quad \frac{\partial f}{\partial x_n(k)}\right], \tag{4.25}$$

je gradijent funkcije troška u *k*-toj iteraciji. Poseban oblik gradijentnog algoritma koristi moment drugog reda i njegov zapis ima oblik:

$$\Delta x(k) = -\eta \nabla f(x(k)) + \alpha \Delta x(k-1) + \beta \Delta x(k-2). \tag{4.26}$$

Dodatni skalarni parametri koji se pojavljuju su $\alpha\ i\ \beta$. Parametar α može poprimiti vrijednosti između 0.5 i 1, a preporučana vrijednost parametra β iznosi $(\alpha - 1)/3$.

4.3.1. Parametri konjugirane gradijentne metode

Konjugirano gradijentna metoda ima oblik

$$x(k+1) = x(k) + \eta d(k), \tag{4.27}$$

gdje je η pozitivan broj i vektor smjera pretraživanja iznosi

$$d(k+1) = -g(k+1) + \beta(k)d(k), \tag{4.28}$$

a skalarni parametar β_k je dan idućom formulom

$$\beta(k) = \frac{\lambda(k)g(k+1)\cdot g(k+1)^T + [1-\lambda(k)]g(k+1)\cdot[g(k+1)-g(k)]^T}{\mu g(k)\cdot g(k)^T + [1-\mu(k)]d(k)\cdot[g(k+1)-g(k)]^T}, \tag{4.29}$$

dok $\lambda_k \in [0,1]$ i $\mu_k \in [0,1]$. Ako dani skalari $\mu_k\ i\ \lambda_k$ poprimaju samo svoje krajnje vrijednosti 0 i 1, jednadžba (4.29) se može izračunati iz četiri poznate metode.

Za vrijednosti $\lambda(k) = 1\ i\ \mu(k) = 1$ koristimo Fletcher-Reeves metodu:

$$\beta(k) = \frac{g(k+1)\cdot g(k+1)^T}{g(k)\cdot g(k)^T}. \tag{4.30}$$

Za vrijednosti $\lambda(k) = 0\ i\ \mu(k) = 1$ koristimo Polak-Ribere metodu:

$$\beta(k) = \frac{g(k+1)\cdot[g(k+1)-g(k)]^T}{g(k)\cdot g(k)^T}. \tag{4.31}$$

Za vrijednosti $\lambda(k) = 0\ i\ \mu(k) = 0$ koristimo Hestenes-Stiefel metodu:

$$\beta(k) = \frac{g(k+1)\cdot[g(k+1)-g(k)]^T}{d(k)\cdot[g(k+1)-g(k)]^T}. \tag{4.32}$$

Za vrijednosti $\lambda(k) = 1\ i\ \mu(k) = 0$ koristimo Dai-Yaun metodu:

$$\beta(k) = \frac{g(k+1)\cdot g(k+1)^T}{d(k)\cdot[g(k+1)-g(k)]^T}. \tag{4.33}$$

4.3.2. SuperSAB metoda

Standardna gradijentna metoda često je prespora za praktične primjene, a ima i slabiju mjerljivost kako kompleksnost zadatka raste te se u tu svrhu predstavlja nova metoda prilagodbe specifičnog koraka učenja optimiziranih varijabli SuperSAB [9]. Ova metoda prati ponašanje predznaka gradijenta funkcije cilja u dvije uzastopne iteracije, ako imaju isti predznak stopa učenja se povećava u svrhu ubrzanja konvergencije. U slučaju promjene predznaka derivacija izračun je prešao lokalni minimum i stopa učenja se smanjiva. Jednadžbe koje opisuju danu metodu su sljedeće:

$$x_i(k+1) = x_i(k) + \Delta x_i(k), \tag{4.34}$$

$$\Delta x_i(k) = -\eta_i(k)\frac{\partial f}{\partial x_i(k)} + \alpha \Delta x_i(k-1), \tag{4.35}$$

$$\eta_i(k)\begin{cases} d^+ \cdot \eta_i(k-1), & ako \quad \dfrac{\partial f}{\partial x_i(k)} \cdot \dfrac{\partial f}{\partial x_i(k-1)} > 0 \\ d^- \cdot \eta_i(k-1), & ako \quad \dfrac{\partial f}{\partial x_i(k)} \cdot \dfrac{\partial f}{\partial x_i(k-1)} < 0 \\ \eta_i(k-1), & ako \quad \dfrac{\partial f}{\partial x_i(k)} \cdot \dfrac{\partial f}{\partial x_i(k-1)} = 0 \end{cases} \tag{4.36}$$

Ovaj je algoritam lokalne adaptivnosti, jer se optimizirana varijabla x_i ažurira samo lokalnim komponentama gradijenta funkcije $\frac{\partial f}{\partial x_i(k)}$ i $\frac{\partial f}{\partial x_i(k-1)}$. Iako pruža brzu konvergenciju, pojavljuje se problem diskontinuiteta upravljačke varijable. Kako bi izbjegli taj problem napravit će se izmjene na dani postupak. Izmijenjeni algoritam omogućava globalnu adaptaciju jedne stope učenja u svakoj iteraciji *k*, što dovodi do optimalnog kontinuiranog upravljačkog rješenja.

4.3.3. Računanje gradijenta za zadani problem

Optimizacijski pristup temelji se na algoritmu konjugiranog gradijentnog spusta za matricu težina neuronske mreže $\mathbf{W} = [w_{pq}]$

$$\mathbf{W}^{(l+1)} = \mathbf{W}^{(l)} + \eta^{(l)}\mathbf{S}^{(l)} \tag{4.37}$$

gdje je $l = 1, 2, \dots, M,\ p = 1, 2, \dots, n_u,\ q = 1, 2, \dots, n_0$. *N* je broj vremenskih trenutaka i *M* je broj iteracija u gradijentnom algoritmu, a η je stopa učenja. Matrica pretraživanja je:

$$\mathbf{S}^{(l+1)} = -\nabla J\left(\mathbf{W}^{(l+1)}\right) + \beta^{(l)}\mathbf{S}^{(l)}. \tag{4.38}$$

Standardan metoda računanja $\eta^{(l)}$ je metoda najstrmiji spusta ili algoritam linijskog pretraživanja koji zahtijeva jednodimenzionalnu minimizaciju funkcije cilja. Odmah u startu nailazimo na probleme, gdje zbog mnogobrojnih evaluacija funkcije cilja tijekom jedne iteracije gradijentnog algoritma, računsko opterećenja računala postaje veliko. Nadalje, zbog lošeg skaliranja funkcije cilja, svojstvo konvergiranja postaje lošije [1]. Kako je spomenuto u prethodnim poglavljima, u svrhu izbjegavanja ovakvih problema uveden je *SuperSAB algoritam* koji prati informaciju o smjeru gradijenta kroz dvije uzastopne iteracije. Također se postiže i globalna konvergencija algoritma kao u [9]. Modificirani *SuperSAB* algoritam za računanje težina neuronske mreže je

$$\eta^{(l)} = \begin{cases} d^{+} \cdot \eta^{(l-1)}, & ako \quad \mathrm{Tr}\left\{\left[\dfrac{\partial J}{\partial \mathbf{W}^{(l)}}\right]^{T} \dfrac{\partial J}{\partial \mathbf{W}^{(l-1)}}\right\} \geq 0 \\ d_1^{-} \cdot \eta^{(l-1)}, & ako \quad \mathrm{Tr}\left\{\left[\dfrac{\partial J}{\partial \mathbf{W}^{(l)}}\right]^{T} \dfrac{\partial J}{\partial \mathbf{W}^{(l-1)}}\right\} < 0 \\ d_2^{-} \cdot \eta^{(l-1)}, & ako \quad J(\mathbf{W}^{(l)}) \geq J\left(\mathbf{W}^{(l-1)}\right), \end{cases} \tag{4.39}$$

gdje je $0 < d_2^{-} < d_1^{-} < 1 < d^{+}$ i $\eta^{(0)}$ je početna vrijednost stope učenja.

```
24      #SuperSAB algoritam
25      def SuperSab(self, J_wi, J_wprev, J, J_prev, eta):
26          if np.trace(J_wi.T @ J_wprev)<0:
27              eta = eta*0.95
28          if J>=J_prev:
29              eta = eta*0.4
30          if np.trace(J_wi.T @ J_wprev)>=0:
31              eta = eta*1.2
32          return eta
```

Slika 4.2 Python kod *SuperSAB* algoritma

Na slici 4.2 se nalazi kod u kojem je implementiran *SuperSAB* algoritam za prilagođavanje veličine koraka učenja pomoću dilatacijskih koeficijenata. Uvjeti su određeni prema veličini funkcije cilja s obzirom na prethodni koraka ili tragu umnoška trenutnog i prošlog koraka Jacobijevih matrica težina neuronske mreže. Skalarna vrijednost $\beta^{(l)}$ se može odrediti različitim metodama: Fletcher-Reeves, Polak-Ribiere, Hestenes-Stiefel, Dai-Yuan. U ovom radu će se koristiti primarno Dai-Yuan metoda:

$$\beta^l = min\left\{\frac{\mathrm{Tr}\left\{\left[\frac{\partial J}{\partial \mathbf{W}^{(l+1)}}\right]^T \frac{\partial J}{\partial \mathbf{W}^{(l+1)}}\right\}}{\mathrm{Tr}\left\{\left[\frac{\partial J}{\partial \mathbf{W}^{(l+1)}} - \frac{\partial J}{\partial \mathbf{W}^{(l)}}\right]^T \mathbf{S}^l\right\}}, \beta_{max}\right\}. \tag{4.40}$$

Parametar je limitiran s β_{max} jer u (4.39) kod stope učenja se može pojaviti $\left\|\frac{\partial J}{\partial \mathbf{W}^{(l)}}\right\| \geq \left\|\frac{\partial J}{\partial \mathbf{W}^{(l-1)}}\right\|$ kada je $\left[\frac{\partial J}{\partial \mathbf{W}^{(l)}}\right]^T \frac{\partial J}{\partial \mathbf{W}^{(l-1)}} < 0$ što vodi do nestabilnosti u algoritmu ako β^l nije zasićen. Ukoliko parametar β^l ima konstantnu vrijednost ($0 < \beta^l < 1$), onda govorimo o ekvivalentnu običnom gradijentnom algoirtmu s momentom [1].

Računanje gradijenta funkcije cilja (4.15) u skladu s težinama u *l*-toj iteraciji gradijentnog algoritma i *i*-tom intervalu uzorkovanja se vrši:

$$\frac{\partial J}{\partial w_{pq}} = \sum_{r=1}^{n} \frac{\partial J}{\partial \tilde{x}_r} \frac{\partial \tilde{x}_r(i)}{\partial w_{pq}} \ . \tag{4.41}$$

Parcijalna derivacija $\frac{\partial \tilde{x}_r(N)}{\partial w_{pq}}$ se računa kao:

$$\frac{\partial \tilde{x}_r(i)}{\partial w_{pq}} = \sum_{j=1}^{n} \frac{\partial \widetilde{f}_r(i-1)}{\partial \tilde{x}_j(i-1)} \frac{\partial \tilde{x}_j(i-1)}{\partial w_{pq}} + \sum_{m=1}^{n_u} \frac{\partial \widetilde{f}_r(i-1)}{\partial u_m(i-1)} \frac{\partial u_m(i-1)}{\partial w_{pq}}. \tag{4.42}$$

Nadalje, prema (4.18) parcijalna derivacija $\frac{\partial u_m(i)}{\partial w_{pq}}$ se računa kao:

$$\frac{\partial u_m(i)}{\partial w_{pq}} = D_m(\xi_m(i)) \sum_{j=1}^{n} \left[\frac{\partial w_{\boldsymbol{mj}}}{\partial w_{pq}} \tilde{x}_r(i) + w_{\boldsymbol{mj}} \frac{\partial \tilde{x}_r(i)}{\partial w_{pq}}\right], \tag{4.43}$$

gdje je $D_m(\xi_m(i))$ derivacija funkcije $\sigma_m(\xi_m(i))$ iz jednadžbe (4.8)

$$D_m(\xi_m(i)) = \begin{cases} 0\,; & \xi_m > u_{j,max} \\ 1\,; & -u_{j,max.} < \xi_m < u_{j,max} \\ 0\,; & \xi_m < -u_{j,max.} \end{cases} \tag{4.44}$$

Sljedeći korak je određivanje početnog stanja za gore spomenuti rekurzijski izraz. Početni uvjet vektora stanja je neovisan o težinama w_{jk} tako da vrijedi

$$\frac{\partial \tilde{x}_m(0)}{\partial w_{pq}} = 0. \tag{4.45}$$

Drugi početni uvjet glasi

$$\frac{\partial u_m(0)}{\partial w_{pq}} = D_m(\xi_m(0)) \sum_{j=1}^{n} \frac{\partial w_{\boldsymbol{mj}}}{\partial w_{pq}} \tilde{x}_r(0)\,. \tag{4.46}$$

Konačni gradijent funkcije cilja (4.20), uzimajući u obzir težinsku matricu, se može izračunati prema rekurzijskom algoritmu

$$\begin{aligned} \mathbf{X}_W(i) &= \mathbf{F}_x(i-1)\mathbf{X}_W(i-1) + \mathbf{F}_u(i-1)\mathbf{U}_W(i-1), \\ \mathbf{U}_W(i) &= \mathbf{D}_i\left[\tilde{x}^{\mathrm{T}}(i) \otimes \mathrm{I}_{n_u \times n_u} + \mathbf{W}\mathbf{X}_W(i)\right], \\ \mathbf{J}_W &= \mathbf{J}_{\boldsymbol{x}}(N)\mathbf{X}_W(N). \end{aligned} \tag{4.47}$$

Pojedini članovi se računaju prema

$$\begin{aligned} \mathbf{X}_W(i) &= \frac{\partial \tilde{x}_i(i)}{\partial vec(\mathbf{W})}, \qquad \mathbf{U}_W(i) = \frac{\partial u_i(i)}{\partial vec(\mathbf{W})}, \\ \mathbf{F}_x(i) &= \frac{\partial \tilde{f}(i)}{\partial \tilde{x}_i(i)}, \qquad \mathbf{F}_u(i) = \frac{\partial \tilde{f}(i)}{\partial u(i)}, \\ \mathbf{J}_W &= \frac{\partial J}{\partial vec(\mathbf{W})}, \qquad \mathbf{J}_W = \frac{\partial J}{\partial \tilde{x}(N)}. \end{aligned} \tag{4.48}$$

4.4. Računanje Jacobiana

Osim analitičkim putem gradijent se može izračunati i numerički primjenom formula konačnih diferencija (eng. *Automatic differentiation*) ili primjenom metode kompleksnih varijabli. Dobro je poznato da izrazi u brojnicima formula konačnih diferencija postaju problematični, jer računala ne prepoznaju razliku između dva približno jednaka broja, što dovodi do problema

kraćenja (engl. *cancellation*). Iz tog razloga primorani smo koristiti konzervativne korake diferenciranja, iako to može umanjiti točnost algoritma.

Jedan od takvih pristupa je i primjena kompleksnih brojeva za računanje derivacija realnih funkcija. Primjenom kompleksne varijable možemo ukloniti problem kraćenja, a i derivacija možemo izračunati do strojne preciznosti.

Razvojem $f(x + ih\boldsymbol{e}_j)$ u Taylorov niz za neki mali iznos h

$$f(x + ih\boldsymbol{e}_j) = f(x) + ih\nabla_x f(x) \cdot \boldsymbol{e}_j + \frac{(ih)^2}{2!}\boldsymbol{e}_j{}^{\mathrm{T}} \cdot \nabla_x f(x) \cdot \boldsymbol{e}_j + \cdots, \tag{4.49}$$

ako uzmemo samo imaginarni dio dobiva se

$$\nabla_x f(x) \cdot \boldsymbol{e}_j = \frac{Im\{f(x + ih\boldsymbol{e}_j)\}}{h} + O(h^2). \tag{4.50}$$

U gornjem izrazu $\boldsymbol{e}_j$ predstavlja jedinični vektor (odgovarajuće dimenzije vektora s *j*-tim elementom 1), a i predstavlja jedinični vektor $i^2 = -1$. U brojniku izraza se ne pojavljuje oduzimanje tj. naš problem kraćenja je uklonjen. Korak h u ovom slučaju može biti izuzetno mali broj (npr. $h = 10^{-100}$) što direknto omogućava strojnu preciznost pri računanju.

Implementaciju kompleksne metode se može izvesti na jednostavan način primjenom nekog programskog jezika koji ne zahtijeva prethodno definiranje tipa varijabli, nego direktno definiramo kompleksne varijable putem ugrađenih funkcija. Kompleksnu metodu možemo zapisati u obliku sljedećeg algoritma:

Algoritam 1 Računanje gradijenta $\Delta_x f(x)$ primjenom kompleksne metode

Ulaz: *n, h, x,* $f(x)$

Izlaz: $\Delta_x f(x)$

1: $x1 \leftarrow x$

2: **for** $j = 1$ to n **do**

3: $x1_j \leftarrow complex\ (x_j, h)$

4: $f1 \leftarrow f(x1)$

5: $\frac{\partial f(x_j)}{\partial x_j} \leftarrow \frac{imag(f1)}{h}$

6: $x1_j \leftarrow x$

7: **end for**

4.4.1. Računanje Jacobiana za zadani problem

Prošireni oblik Jacobiana $\mathbf{F}_x(i)$ *i* $\mathbf{F}_u(i)$ se mogu računati kao funkcije osnovnih Jacobiana na način

$$\mathbf{F}_x(i) = \mathbf{I} + \tau \frac{\partial \tilde{f}(i)}{\partial \tilde{x}(i)},$$
$$\mathbf{F}_u(i) = \tau \frac{\partial \tilde{f}(i)}{\partial u(i)} \quad . \tag{4.51}$$

Derivacije prvog reda od $\tilde{f}$ u odnosu na $\tilde{x}$ pomoću aproksimacije koračno kompleksnom metodom je postignuta aproksimacijom njegove komponente (recimo *k*-atom komponentom) kompleksnom varijablom pomoću razvoja u Taylorov niz

$$\tilde{f}_k(\tilde{x} + ih\boldsymbol{e}_j) = \tilde{f}_k(\tilde{x}) + ih\Delta_{\tilde{x}}\tilde{f}_k(\tilde{x}) \cdot \boldsymbol{e}_j + \frac{(ih)^2}{2!}\boldsymbol{e}_j{}^{\mathrm{T}} \cdot \Delta_{\tilde{x}}\tilde{f}_k(\tilde{x}) \cdot \boldsymbol{e}_j + \cdots, \tag{4.52}$$

ako uzmemo samo imaginarni dio

$$\Delta_{\tilde{x}}\tilde{f}_k(\tilde{x}) \cdot \boldsymbol{e}_j = \frac{Im\{\tilde{f}(\tilde{x} + ih\boldsymbol{e}_j)\}}{h} + O(h^2). \tag{4.53}$$

U ovom slučaju *i* koristimo kao oznako imaginarne jedinice, a do sad smo ga koristili kao *i*-tu vremensku točku. Zbog tog u gornjem izrazu nije rabljena notacija $\tilde{x}(i)$, ali se podrazumijeva. U ovom radu će se koristiti *Python* programski jezik i u njemu se vrlo lako preko ugrađene funkcije *complex()* definira imaginarni broj *i* koji će nam koristiti prilikom definiranja Algoritma 1. Na slici 4.3 je prikazana primjena kompleksne metode u *Pythonu*.

```python
# Jacobian od f po x pomocu kompleksne metode
import numpy as np

def Jacobian_x(func, x, x2, h=None):
    if h is None:
        h = np.finfo(float).eps
    df_dx = []
    for i in range(len(x)):
        x1 = x.astype(complex)
        x1[i] = complex(x[i], h)
        f = func(x1, x2)
        df_dx.append(f.imag/h)
    return np.array(df_dx).reshape(len(df_dx[0]), len(df_dx))


# Jacobian od f po u pomocu kompleksne metode

def Jacobian_u(func, x, x2, h=1e-100):
    df_du = []
    for i in range(len(x)):
        x1 = x.astype(complex)
        x1[i] = complex(x[i], h)
        f = func(x2, x1)
        df_du.append(f.imag/h)
    return np.array(df_du).reshape(len(df_du[0]), len(df_du))

```

Slika 4.3 ***Python*** **kod za kompleksnu metodu računanja Jacobijana**

Broj *h* se može definirati eksplicitno nekom malom vrijednošću ili iskoristit funkciju *finfo(float)* koja vraća najmanju moguću pozitivnu brojnu vrijednost za *float* tip podataka.

4.5. Sumiranje algoritma

U idućim koracima bit će sumiran do sad objašnjeni algoritam iz 4. poglavlja:

Tablica 4.1 Sumiran algoritam po koracima [1]

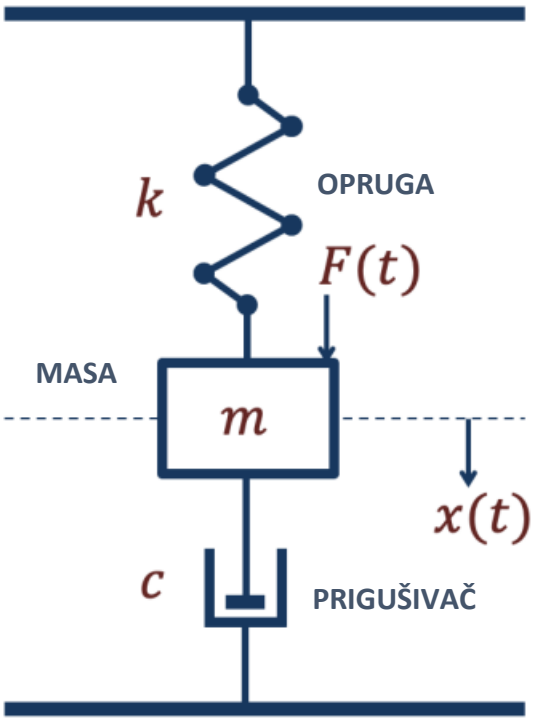
1. Korak	Inicijalizacija podataka: početak u l = 1 s početnim vektorom stanja $\boldsymbol{x_0}$; izbor inicijalnih vrijednosti matrice težina **W**; postaviti konačno vrijeme $\boldsymbol{t_f}$, broj vremenskih intervala *N*, duljina vremenskog koraka $\boldsymbol{\tau}$; postaviti broj iteracija gradijentnog algoritma *M*, inicijalna stopa konvergencije $\boldsymbol{\eta_0}$; postaviti početne dilatacijske parametre $\boldsymbol{d^+}$ *i* $\boldsymbol{d^-}$ i $\boldsymbol{\beta_{max}}$.
2. Korak	Računanje vektora stanja (4.16) upotrebom Eulerove metode.
3. Korak	Računanje proširenih Jacobijana (4.51) pomoću kompleksne koračne metode.

4. Korak	Računanje algoritma propagacije unatrag od (4.42) do (4.46).
5. Korak	Računanje gradijenta (4.41).
6. Korak	Računanje novih težina koristeći konjugiranu gradijentnu metodu(4.37) i (4.38) s Dai-Yuan metodom (4.40) i SuperSAB algoritmom (4.39).
7. Korak	Povećavanje indeks $l = l + 1$ i vraćanje na korak 2.

5. SIMULACIJSKI PRIMJER

5.1. Dinamički model MDS sustava

Kako bi proveli analizu i usporedbu rezultata dobivenog algoritma s numeričkom metodom za sintezu linearnog kvadratnog regulatora (LQR) u Pythonu potrebno je izabrati i potrebni linearni model sustava. Standardni primjer linearnog sustava je i MDS sustav (eng. *mass-spring- damping*) sa slici 5.1.



Slika 5.1 MDS sustav

Sustav možemo opisati idućom jednadžbom

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t), \tag{5.1}$$

gdje je t vrijeme, $F(t)$ je vanjska sila nametnuta na sustav, c je faktor prigušenja, k je faktor krutosti opruge. Nadalje, $x(t)$ označava poziciju mase m u datom trenutku, $\dot{x}(t)$ je brzina mase m, a $\ddot{x}(t)$ označava ubrzanje mase m.

Diferencijalne jednadžbe višeg reda zatim možemo prebaciti u prostor stanja s pomoću sljedećih jednadžbi

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx),$$

$$x = x_1 \rightarrow \dot{x}_1 = x_2, \tag{5.2}$$

$$\dot{x} = x_2 \rightarrow \dot{x}_2 = -\frac{k}{m}x_1 - \frac{c}{m}x_2 + \frac{1}{m}F\,.$$

Nadalje naš model prema (2.1) i (2.2) kraće možemo zapisati kao

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 1\\ -\frac{k}{m} & -\frac{c}{m}\end{bmatrix}}_{A}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ \frac{1}{m}\end{bmatrix}}_{B}F \tag{5.3}$$

$$\mathrm{y}(t) = \mathbf{C}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Polazne vrijednosti za dane faktore iznose $m = 1\ kg, k = 1\ \frac{N}{m}$ i $c = 0.2\frac{Ns}{m}$. Za dani problem matrica $\mathbf{C} = [1 \quad 0]$.

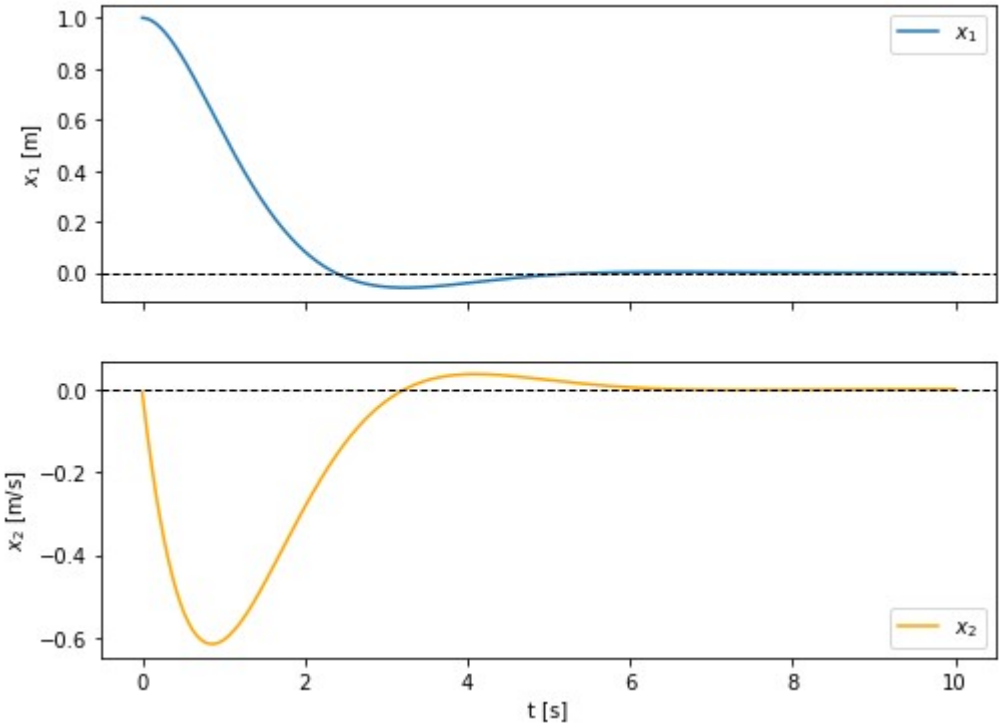
5.2. Optimizacijski problem i rezultati simulacije

Za numeričko računanje rješenja danog problema će se primijeniti algoritam iz prethodnog poglavlja, a izvest će se u Python programskom jeziku. Vrijeme izvedbe je $t_f = 10\ s$, a broj optimizacijskih intervala je $N = 2000$ tako da interval uzorkovanja iznosi $\tau = 0.005\ s$. Konjugirano kompleksni gradijent s Dai-Yuanom je korištena, a broj iteracija izvođenja gradijenta je $M = 100$. Numeričke vrijednosti parametara relevantnih za algoritam iznose: $\beta_{max} = 0.85, d^+ = 1.2,\ d_1^- = 0.95,\ d_2^- = 0.4,\ \eta_0 = 1.2.$

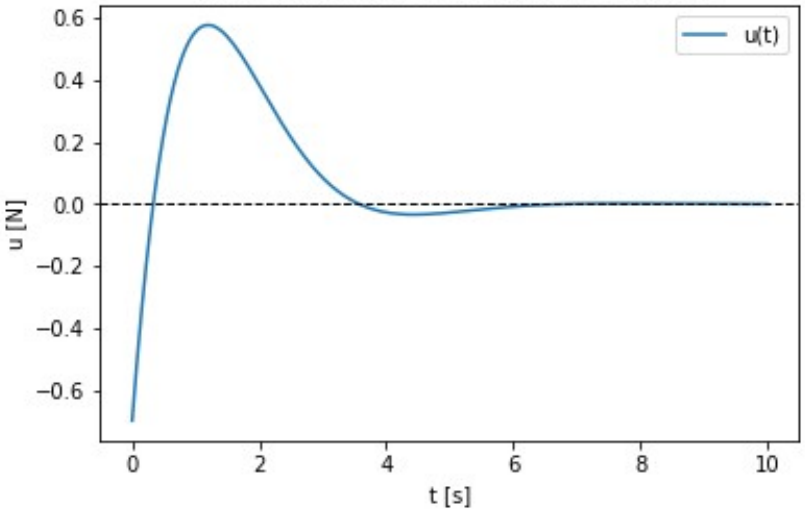
Članovi matrice težina u početku su postavljeni na $w_{11} = w_{12} = -1$. Iako je broj iteracija bio postavljen na 100, algoritam je konvergirao već nakon 23. iteracije i nove vrijednosti težina upravljačke varijable u postaju $w_{11} = -0.69995\ i\ w_{12} = -1.53537$.

Vektor početnih uvjeta varijable stanja iznosi $x_0 = \begin{bmatrix}1\\ 0\end{bmatrix}$, a na slici 5.1 prikazana je vremenska ovisnost varijable stanja. Slika 5.2 prikazuje vremensku ovisnost upravljačke varijable za Eulerovu metodu diskretizacije u zadanom algoritmu. Nakon prijelazne pojave, zbog pogreške u početnim uvjetima, pogreške funkcije cilja asimptotski teže nekoj maloj konačnoj vrijednosti

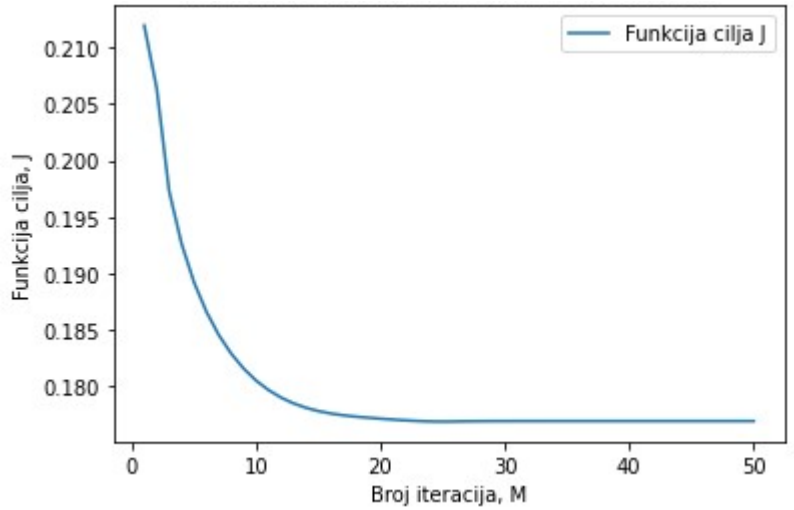
kako je prikazano na slici 5.3. Konvergencija u J = 0,1769 ostaje konstantnom i nakon povećavanja broja iteracija na 250. Na slici 5.1 su prikazane varijable stanja u ovisnosti o vremenu.



Slika 5.1 Vremenska ovisnost varijabli stanja



Slika 5.2 Vremenska ovisnost upravljačke varijable



Slika 5.3 Funkcija cilja u odnosu na broj iteracija

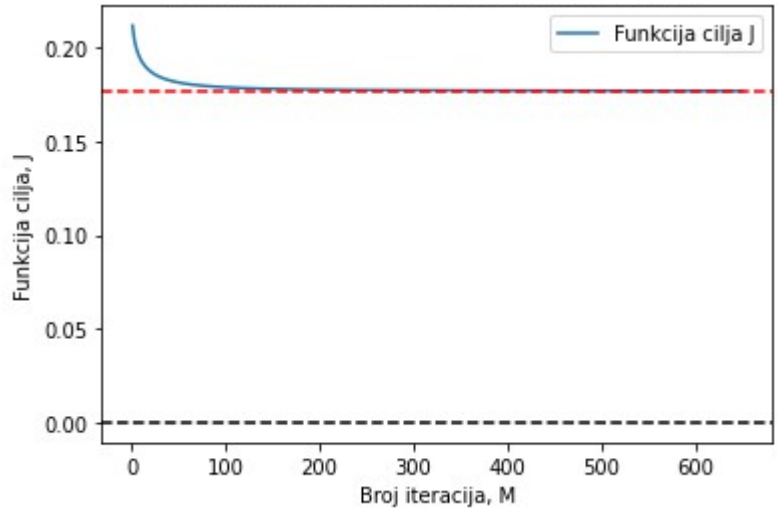
U obzir za optimizaciju algoritma osim Dai-Yuan metode uzete su u obzir i metoda od (4.31) do (4.33), no zbog jednostavnosti primjera nema skoro nikakve razlike između njih i konvergencija se postiže do 27. iteracije za prve tri metode. Dai-Yuan postiže najbržu konvergencije i ona se događa već nakon 23. iteracije, ali je i najsporija u kompletnoj izvedbi algoritma.

Tablica 5.1 Usporedba različitih optimizacijskih metoda

Ime metode	Konfiguracija	Greška funkcije cilja, j	Vrijeme izvedbe, [s]
Dai-Yaun	$t_f = 10$ $N = 2000$ $M = 50$ τ =0,005 $\beta_{max} = 0{,}85$ $\eta = 1{,}2$	0,1769	41,7029
Fletcher-Reeves		0,1769	37,4441
Polak-Ribere		0,1769	38,6094

Na tablici 5.1 dana je usporedba različitih metoda za optimizaciju. Fletcher-Reeves ima konvergenciju u 24. iteraciji te Polak-Ribere u 27. iteraciji. Za dane metode simulacija je ponovljena i za 100 iteracija i konvergencija je bila postojana za sve osim Hestenes-Stiefel metode pa ona nije uključena u usporedbu. Na brzinu izvođenja algoritma će utjecati dosta i sama izmjena β_{max} i intervala uzorkovanja što je prikazano u idućim poglavljima.

Ukoliko iskoristimo isti problem za obični gradijenti algoritam uz početne uvjete $\eta_0 = 0.85$, isto vrijeme izvedbe $t_f = 10\ s$, broj optimizacijskih intervala $N = 2000$, interval uzorkovanja $\tau = 0.005$ i $M = 650$ dobivaju se slični rezultati, ali funkcija cilja, prikazana na slici 5.4, će konvergirati tek nakon 618 iteracija. Za tu konvergenciju će joj biti potrebno 634,17 sekunde.

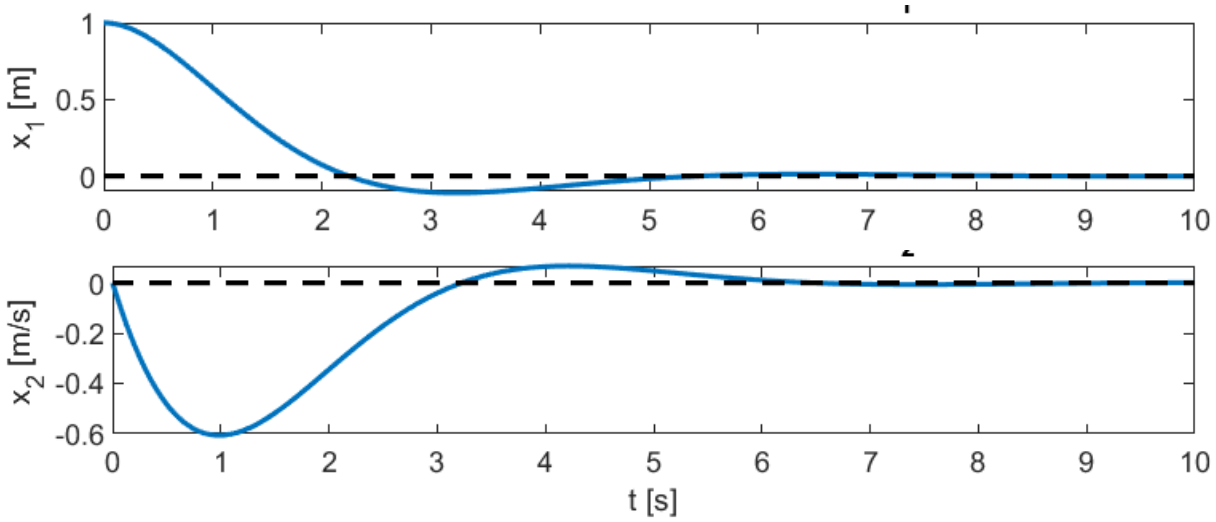


Slika 5.4 Funkcija cilja obične gradijentne metode u odnosu na M

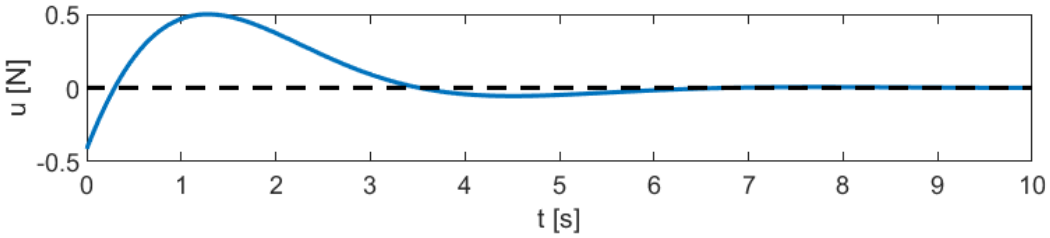
Na ovom primjeru se jako dobro vidi kako SuperSAB algoritam omogućava puno bržu konvergenciju bez velikog utjecaja na stabilnost. Matrica težina poprima vrijednosti $\mathbf{W} = [-0.71348 \quad -1.47572]$. Razlika između ove dvije metode će drastično rasti kako se budu mijenjali ulazni parametri. Jedan od parametara koji jako utječe na stabilnost obične gradijentne metode je stopa učenja. Ona kod običnog gradijenta ima velik utjecaj na konvergiranje i stabilnost. Ukoliko stopu učenja smanjimo od navedene ona sporo konvergira prema konstantnoj vrijednosti, ali broj iteracija se mora znatno povećati.

5.2.1. Usporedba – LQR

Za sintezu linearnog regulatora LQR metodom, za definirani MDS sustav, se koristi programski jezik *Matlab* i njegova funkcija *lqr(sys,Q,R,N)*. Upravljačka varijabla je opisana formulom $u = -Kx$. Iznos dobivene matrice pojačanja je $\mathbf{K} = [0.4142 \quad 1.1669]$, a dobiveni rezultati za varijable stanja su prikazani na slici 5.5. Ovisnost upravljačke varijable o vremenu je prikazana na slici 5.6. Vrijeme potrebno za izvedbu koda LQR kontrolera iznosi 0.03966 sekundi, a vrijeme za izvođenje simulacije 0.027571sekundi. Vrijeme je izvedbe je znatno brže nego za metode u tablici 5.1.



Slika 5.5 Vremenska ovisnost varijabli stanja



Slika 5.6 Ovisnost upravljačke varijable o vremenu

Iz prikazanih rezultata se može vidjeti da smo dobili veliko poklapanje s onima dobivenim pomoću neuronske mreže. Naša mreža jako dobro optimizira zadani problem, što je bilo i za očekivati s obzirom na samu jednostavnost sustava. Tamo gdje se javlja problem kod modela s neuronskom mrežom je taj što prilikom izmjena vrijednosti matrica **Q** i **R** sustav postaje nestabilan i metoda počinje divergirati ili se javlja problem nestajućeg gradijenta čime LQR problem postaje nerješiv. Kod LQR metode za nove vrijednosti matrica $\mathbf{Q_1} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$ i $\mathbf{R_1} = [10]$ sustavu se samo neznatno poveća. Vrijednost izvršavanja algoritma su prikazane u tablici 5.2.

Eulerova metoda je prvog reda točnosti i može imati numeričke probleme, a posebno u situacijama gdje se vremenski koraci moraju odabrati vrlo malima kako bi se održala stabilnost. To može rezultirati sporim treniranjem i potrebom za pažljivim podešavanjem hiperparametara. S druge strane kod LQR metode takvih problema nema, no ona je ograničena samo na linearne sustave. Neuronska mreža uz potrebne modifikacije može biti pogodna za nelinearne sustave [1].

Tablica 5.2 Matlab – LQR za različite vrijednosti Q i R matrica

Ime metode	Konfiguracija	Vrijeme izvedbe, [s]	Vrijeme simulacije, [s]
LQR: $\mathbf{Q} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$ $\mathbf{R} = [0.1]$.	$t_f = 10$ $N = 2000$ $M = 50$ τ =0,005 $\beta_{max} = 0{,}85$ $\eta = 1{,}2$	0.03966	0.027571
LQR: $\mathbf{Q_1} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$ $\mathbf{R_1} = [10]$		0.35915	0.12764

5.3. Prilagođavanje ulaznih parametara

Kako je već i spomenuto veliki zahtjev kod ovakvog algoritma je i sama inicijalizacija hiperparametara. Algoritam posjeduje nekoliko takvih parametara, a oni su: interval uzorkovanja τ, stopa učenja η_0, dilatacijski parametri d^+, d_1^-, d_2^- i β_{max} (Dai-Yuan). Smanjenje intervala uzorkovanja generalno vodi ka točnijem rezultatu, ali ono uzrokuje povećano računsko opterećenje. Ukoliko je prevelika vrijednost intervala uzorkovanja može doći do numeričke nestabilnosti Eulerove metode tj. metoda postaje kruta pri velikim τ.

Postavljanje dilatacijskih parametara i β_{max} je dobiveno prema [1], dok stopa učenja η_0 ovisi o izboru metode optimizacije. Neke okvirne vrijednosti dilatacijskih parametara su $0.85 \leq d^+ \leq 0.95, 1.05 \leq d^- \leq 1.2$, a preporučena veza između dilatacijskih parametara je dana s $d^+ \approx 1/d^-{}_1$. Pokazalo se za konjugirano gradijentni algoritam da nije previše osjetljiv na inicijalno postavljanje stope učenja. Kod programiranja Dai-Yuan metode u *Pythonu* prilikom računanja skalarnih vrijednosti β se pojavio problem dijeljena s nulom. Prema (4.33) ukoliko nije velika razlika između gradijenta funkcije cilja u trenutnom i prethodnom trenutku doći će do dijeljenja nulom. Kao rješenje je dodan denominator u kod prema slici 5.7. Na ovaj način dodajemo jako malu vrijednost u nazivniku te ona sama ne mijenja fundamentalno algoritam.

```python
#Dai-Yuan metoda
def DaiYuan(self, beta_max, Jw, Jw_plus, S):
    deltaJ = Jw_plus - Jw

    # jako mali broj
    mali_br = 1e-10

    # provjera
    denominator = np.trace(deltaJ.T @ S)
    if denominator < mali_br:
        denominator += mali_br

    beta = np.trace((Jw_plus.T @ Jw_plus)/(denominator))
    return min(beta, beta_max)
```

Slika 5.7 Dai-Yuan - računanje skalara β

Vrijeme izvršavanja i stabilnost ovisi dosta o samom izboru β_{max}, ali i o izboru intervala uzorkovanja. Prema tablici 5.3 vidimo da su se vremena izvršavanja povisila, ali su se i međusobno metode izjednačile. Također možemo vidjeti da za isti interval uzorkovanja i malom promjenom β_{max} sustav može postati nestabilan. Smanjivanjem intervala uzorkovanja, za isti β_{max}, vrijeme izvođenja se znatno povećava.

Tablica 5.3 Rezultati algoritma s novim $\beta_{max} = 0,7$

Ime metode	Konfiguracija	Greška funkcije cilja, J	Vrijeme izvedbe, [s]
Dai-Yaun	$t_f = 10$ $N = 2000$ $M = 50$ τ =0,005 $\beta_{max} = 0,7$ $\eta = 1,2$	0,1769	44,0352
Fletcher-Reeves		0,1769	44.8306
Polak-Ribere		0,1769	44.6435
Dai-Yaun	τ =0,0005, $\beta_{max} = 0,9$	0,1880	195,3358
Dai-Yaun	τ =0,0005, $\beta_{max} = 0,85$	divergencija	-
Dai-Yaun	τ =0,0027, $\beta_{max} = 0,85$	0,1885	25,2509
Dai-Yaun	τ =0,01, $\beta_{max} = 0,85$	0,1780	23.5828

Smanjene stope učenja u odnosu na početnu je prema vrijednostima iz tablice 5.4 uzrokovala povećanje vremena izvršavanja algoritma za Dai-Yaun i Fletcher-Reeves, a za Polak-Ribere je smanjeno. Rezultati su uspoređivani s vrijednostima iz tablice 5.1.

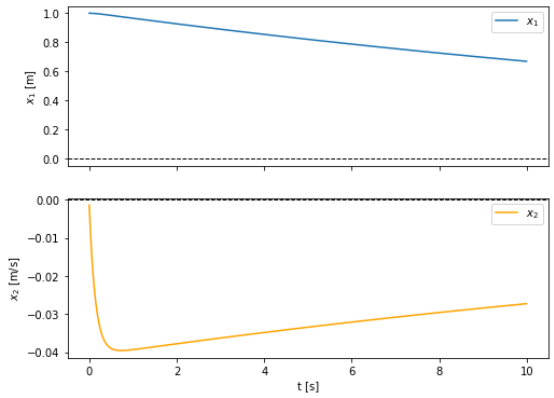
Tablica 5.4 Rezultati nakon promjene stope učenja

Ime metode	Konfiguracija	Greška funkcije cilja, j	Vrijeme izvedbe, [s]
Dai-Yaun	$t_f = 10$ $N = 2000$ $M = 50$ τ =0,005 $\beta_{max} = 0{,}85$ $\eta = 0{,}9$	0.1770	47.1878
Fletcher-Reeves		0,1769	38.2714
Polak-Ribere		0,1769	37.5583

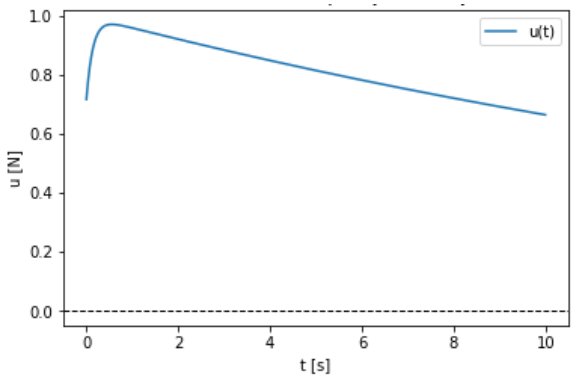
MDS sustav zahtijeva također i inicijalizaciju težinskih matrice **Q** i **R**. Eksperimentalnim namještanjem matrica dobivene su sljedeće vrijednosti:

$$\mathbf{Q} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad \mathbf{R} = [\mathbf{0}.1] \tag{5.4}$$

Jedino za ovaj par matrica je sustav davao stabilna rješenja koja su točno opisivala zadani problem, ako bi se koristile neke druge vrijednosti došlo bi do pucanja Eulerove metode i čime LQR problem postaje nerješiv. Na slici 5.8 i slici 5.9 su prikazani rezultati za matricu $\mathbf{Q_1} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$ i $\mathbf{R_1} = [10]$.

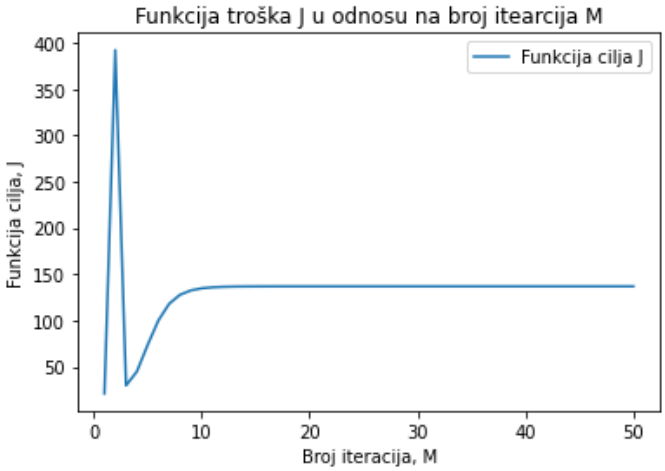


Slika 5.8 Varijable stanja za $\mathbf{Q_1}$ i $\mathbf{R_1}$



Slika 5.9 Upravljačka varijabla za Q_1 i R_1

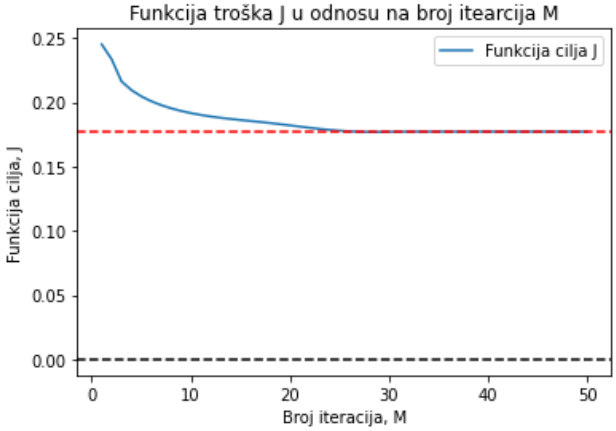
Funkcija troška na slici 5.10 konvergira, ali u neku besmislenu vrijednost. Rješenje ovog problema se može pronaći u korištenju Adamsove metode višeg reda i automatske diferencijacije. Kako je pokazno u [1] korištenjem tih dviju metoda efektivno se postiže robusniji algoritam za rješavanje optimalne sinteze regulatora stanja za min-max problem upravljanja nelinearnih sustava. Za svaku konfiguraciju nove dinamike sustava potrebno je ponovno parametre namještati.



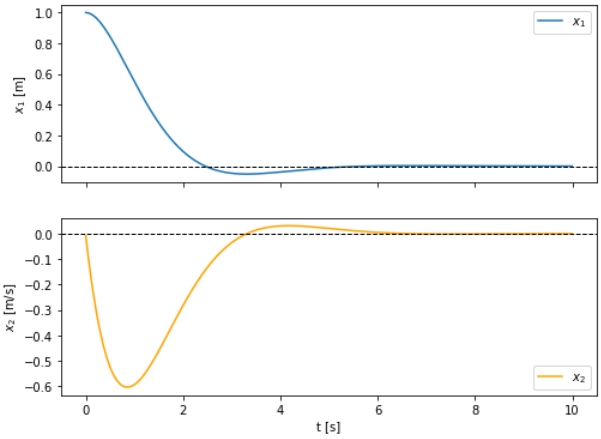
Slika 5.10 Funkcija cilja za Q_1 i R_1

Inicijalizacija samih početnih vrijednosti matrica težina je jako velik faktor prilikom modeliranja ovog algoritma. U početku su isprobane nasumične metode inicijalizacije podataka poput He inicijalizacije [6]. Rezultati dobiveni s njom su jako varirali i nisu bili stabilni. Zatim su težine postavljene u 0 i tad su se počeli dobivati neka stabilnija rješenja, ali ne dovoljno dobra. Daljnjim eksperimentiranjem težine su konačno postavljana između -1 i 0. za te vrijednosti su se već počeli dobivati smisleni rezultati. Konačno izbor je zaustavljen na

inicijalnim vrijednostima **W** = [-1,-1]. Rezultati za početni sustav $\tau = 0{,}005$ i početne težine postavljene na $\mathbf{W_1}$ = [-1,-0,75] su prikazani na slici 5.11 i slici 5.12. Sustav za manje vrijednosti od -0.75 počinje divergirati i uz promjenu početnih parametara.



Slika 5.11 Funkcija cilja za $\mathbf{W_1}$



Slika 5.12 Varijable stanja za $\mathbf{W_1}$

6. ZAKLJUČAK

U prvom dijelu ovog rada predstavljena je teorijska podloga za LTI sustave te su objašnjeni pojmovi upravljivosti i osmotrivosti. Nadalje, kada govorimo o sustavima i njihovoj stabilnosti to podrazumijeva da rješenja sustava teži ravnotežnom stanju kada vrijeme teži u beskonačnost. Zatim smo definirali Lyapunovljevu funkcije i uvijete stabilnosti. U slučaju linearnog vremenski invarijantnog sustava izravna Lyapunovljeva metoda daje dovoljne i nužne uvjete stabilnosti. Kada govorimo o LTI sustavima, mogu se razmotriti projektiranje regulatora, kao što su proporcionalno-integrirajući-derivirajući (PID) kontroleri, sustavi s linearnim kvadratnim regulatorom (LQR), ili sustavi s H∞ optimalnim kontrolerima. U ovom radu fokus je bio na linearnim kvadratnim regulatorima. Postupak uključuju optimizaciju parametara kontrolera kako bi se postigla željena svojstva sustava tj. poboljšala stabilnost i performanse sustava.

Za rješavanje problema optimalnog upravljanja u ovom radu je predložena jednoslojna neuronska mreža sa saturacijskom aktivacijskom funkcijom. Unutar neuronske mreže je korištena struktura algoritma propagiranja unatrag kroz vrijeme slična BPTT koja koristi konjugiranu gradijentnu metodu poboljšanu *SuperSAB* algoritmom. Na klasičnom linearnom sustavu masa-opruga-prigušenje je provedena simulacija izvedenog algoritma u Pythonu i simulacija LQR metodom ugrađenom kao funkcija u Matlab. Rezultati optimalne sinteze regulatora su pokazali da izvedeni algoritam neuronske mreže nakon podešavanja početnih parametara daje zadovoljavajuće rezultate. Postignuta je konvergencija funkcije cilja, a postignuta je i upravljivost varijabli stanja. Simulacija pokušaja bržeg odziva sustava je također napravljena, no ispostavilo se da daljnjim utjecajem na matrice **Q** i **R** dolazi do divergencije i LQR problem postaje nerješiv. Dobiveni rezultati vrijede samo za tu specifičnu konfiguraciju LTI sustava što nam ukazuje na veliku osjetljivost samog sustava. Zbog korištenja Eulerove metode te način na koji utječe na robusnost algoritma i upravljivost, predložena su i poboljšanja u vidu korištenja Adamsove metode višeg reda i automatske diferencijacije obrađene u [1]. Također se može implementirati rano zaustavljanje gradijentnog algoritma učenja. Ranim zaustavljanjem procesa treninga, cilj je pronaći model koji dobro generalizira nove, do tad nepoznate podatke uspostavljajući ravnotežu između prilagođavanja podataka procesu učenja i izbjegavanju takozvanog *overfittinga*.

LITERATURA

[1] Milić V, Kasać J, Majetić D. A back propagation through time-like min-max optimal control algorithm for nonlinear systems. Optimal Control Applications and Methods. 2012 Mar 28;34(3):364–78.

[2] Mertzios B, Christodoulou M. On the generalized Cayley-Hamilton theorem. IEEE Transactions on Automatic Control 1986.; 31: 156–157.

[3] Friedland B. Control System Design: An Introduction to State-Space Methods . New York: Dover Publications; 2005.

[4] Sinha A. Linear Systems: Optimal and Robust Control. Boca Raton: CRC Press;2007.

[5] Datta B.N., Numerical methods for linear control systems design and analysis, Elsevier Science Publishing Co Inc, 2003.; 10:364.

[6] Müller, B., Reinhardt, J., Strickland, M.T. (1995). BTT: Back-Propagation Through Time. In: Neural Networks. Physics of Neural Networks. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-57760-4_28

[7] M. Abu-Khalaf, F. L. Lewis, and J. Huang. Neurodynamic programming and zerosum games for constrained control systems. IEEE Transactions on Neural Networks, 19(7):1243–1252, 2008.

[8] P. J. Werbos. Backpropagation through time: What it does and how to do it. In Proceedings of IEEE, volume 78, pages 1550–1560, October 1990.

[9] T. Tollenaere. SuperSAB: Fast adaptive backpropagation with good scaling properties. Neural Networks, 3(5):561–573, 1990.

[10] Dai YH, Yuan Y. A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property. SIAM Journal on Optimization. 1999 Jan;10(1):177–82.

[11] Milić, V.; Numerical algorithm for robust optimal control of electromechanical systems. Internal report. HRZZ projekt 03.01/170, 2011.

[12] Kreim H, Kugelmann B, Pesch HJ, Breitner MH. Minimizing the maximum heating of a re-entering space shuttle: an optimal contol problem with multiple control constraints. Optimal Control Applications and Methods 1996; 17(1):45–69.

[13] Qingguo L, Payandeh S. Planning for dynamic multiagent planar manipulation with uncertainty: a game theoretic approach. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 2003; 33(5):620–626.

[14] Gruber JK, Ramirez DR, Alamo T, Bordons C, Camacho CF. Control of a pilot plant using QP based min–max predictive control. Control Engineering Practice 2009; 17(11):1358–1366.

[15] Nonhoff M, Köhler J, Müller MA. Online convex optimization for constrained control of linear systems using a reference governor*. IFAC-PapersOnLine [Internet]. [Citirano:28.11.2023.];56(2):2570–5. Dostupno na:https://www.sciencedirect.com/science/article/pii/S2405896323017469#abs0001

PRILOZI

I. Python skripta

Python skripta

SKRIPTA 1

```
# Jacobian of f po u pomocu kompleksne metode
import numpy as np

def Jacobian_u(func, x, x2, h=1e-100):
    df_du = []
    for i in range(len(x)):
        x1 = x.astype(complex)
        x1[i] = complex(x[i], h)
        f = func(x2, x1)
        df_du.append(f.imag/h)
    return np.array(df_du).reshape(len(df_du[0]), len(df_du))

# Jacobian od f po x pomocu kompleksne metode
import numpy as np

def Jacobian_x(func, x, x2, h=None):
    if h is None:
        h = np.finfo(float).eps
    df_dx = []
    for i in range(len(x)):
        x1 = x.astype(complex)
        x1[i] = complex(x[i], h)
        f = func(x1, x2)
        df_dx.append(f.imag/h)
    return np.array(df_dx).reshape(len(df_dx[0]), len(df_dx))

# Derivacija saturacijske funkcije
import numpy as np

def d_sat_func(u, umax:float, umin:float):
    if u >= umax:
        return 0
    elif u <= -umin:
        return 0
    else:
        return 1

d_sat_func_vec = np.vectorize(d_sat_func)

## Saturacijska funkcija
import numpy as np

def sat_func(u, umax:float, umin:float):
    if u >= umax:
        return umax
    elif u <= -umin:
        return -umin
    else:
        return u

sat_func_vec = np.vectorize(sat_func)
```

SKRIPTA 2

```
import numpy as np
from sat_func import sat_func_vec
from d_sat_func import d_sat_func_vec
from Jacobian_x import Jacobian_x
from Jacobian_u import Jacobian_u
import matplotlib.pyplot as plt

class BPTT_method:
    def __init__(self, sus_func=None, A=None, B=None):
        self.A = A
        self.B = B
        self.n = A.shape[0]
        self.m = B.shape[1]
        #self.sus_func = lambda x,u: np.matmul(self.A, x) + np.matmul(self.B, u)
        self.sus_func = lambda x, u: self.A @ x + self.B @ u

    def F(self, x, u):
        Q = self.Q
        R = self.R
        #return np.matmul(np.matmul(x.T, Q), x) + np.matmul(np.matmul(u.T, R), u)
        return x.T @ Q @ x + u.T @ R @ u

    #SuperSAB algoritam
    def SuperSab(self, J_wi, J_wprev, J, J_prev, eta):
        if np.trace(J_wi.T @ J_wprev)<0:
            eta = eta*0.95
        if J>=J_prev:
            eta = eta*0.4
        if np.trace(J_wi.T @ J_wprev)>=0:
            eta = eta*1.2
        return eta

    #Dai-Yuan metoda
    def DaiYuan(self, beta_max, Jw, Jw_plus, S):
        deltaJ = Jw_plus - Jw

        # jako mali broj
        mali_br = 1e-10

        # provjera
        denominator = np.trace(deltaJ.T @ S)
        if denominator < mali_br:
            denominator += mali_br

        beta = np.trace((Jw_plus.T @ Jw_plus)/(denominator))
        return min(beta, beta_max)

    #Euler
    def Update_xtilda(self, xtilda, u, tau, sus_func, F):
        x_dot = sus_func(xtilda[:-1, :], u)
        ix_dot = F(xtilda[:-1, :], u)

        xtilda_dot = np.vstack([x_dot, ix_dot])
        xtilda = xtilda + xtilda_dot * tau
        return xtilda
```

```
    #BPTT algoritam
    def main_alg(self, x0, tf:float, N:int, M:int, eta:float, bmax:float, Q, R, umax:float,
umin:float, SupSab = True, DaiYuan = True, FletcherReeves= False, PolakRibiere= False,
HestenesStiefel= False):
        self.Q = Q
        self.R = R
        W=-np.ones([self.m, self.n])

        W_optim = W
        tau = tf/N
        J_wprev = np.zeros([self.m, self.n])
        J_prev = 0
        J = 0
        J_list = []

        #varijable za plotanje
        state_variable_values_mth_iteration = []
        time_values=[]
        control_input_values_mth_iteration = []

        for l in range(M):
            xtilda = np.vstack([x0, 0])
            X_w = np.zeros([self.n+1, self.m*self.n])
            J_x = np.zeros([1, self.n])
            J_u = np.zeros([1, self.m])
            iteration_time_values = []

            for i in range(N):

                epsilon = W @ xtilda[:-1, :]
                u = sat_func_vec(epsilon, umax, umin)
                #Euler
                xtilda = self.Update_xtilda(xtilda, u, tau, self.sus_func, self.F)
                # spremanje varijabli u M-toj iteraciji
                if l == M - 1:
                    state_variable_values_mth_iteration.append(xtilda[:-1,
:].flatten().tolist())
                    control_input_values_mth_iteration.append(u.flatten().tolist())

                iteration_time_values.append(i * tau)
                #Derivacije
                Jacobian_xx = Jacobian_x(self.sus_func, xtilda[:-1, :], u)
                Jx = Jacobian_x(self.F, xtilda[:-1,:], u)

                Jacobian_xx = np.vstack([Jacobian_xx, Jx])
                Jacobian_xx = np.hstack([Jacobian_xx, np.zeros([self.n+1, 1])])

                Jacobian_uu = Jacobian_u(self.sus_func, u, xtilda[:-1, :])
                Jacobian_uu = np.vstack([Jacobian_uu, Jacobian_u(self.F, u, xtilda[:-1, :])])

                F_x = np.eye(self.n+1)+tau*Jacobian_xx
                F_u = tau*Jacobian_uu
                D = np.array(d_sat_func_vec(epsilon, umax, umin)).reshape([self.m, 1])
```

```
            U_w = D*(np.kron(xtilda[:-1, :].T, np.eye(self.m))+(W @ X_w[:-1,:]))
            X_w = F_x @ X_w + F_u @ U_w

            J_x += tau*Jacobian_x(self.F, xtilda[:-1, :], u)
            J_u += tau*Jacobian_u(self.F, u, xtilda[:-1, :])
            J += tau*self.F(xtilda[:-1, :], u)

        J_x = np.hstack([J_x, np.array([[0]])])
        J_w = J_x @ X_w + J_u @ U_w
        J_w = J_w.reshape(self.m, self.n)
        J = xtilda[-1, -1]

        time_values.append(iteration_time_values)

        if SupSab and l >= 1:
            eta = self.SuperSab(J_w, J_wprev, J, J_prev, eta)
        if l != 1:
            S = -J_w
        if DaiYuan and l>=1:
            beta = self.DaiYuan(bmax, J_wprev, J_w, S)
            S = -J_w + beta*S

        if FletcherReeves and l >= 1:
            beta_fr = np.trace(J_w.T @ J_w) / np.trace(J_wprev.T @ J_wprev)
            S = -J_w + beta_fr * S

        if PolakRibiere and l >= 1:
            numerator_pr = J_w.T @ (J_w - J_wprev)
            denominator_pr = J_wprev.T @ J_wprev
            # beta_pr = max(0, numerator_pr / denominator_pr)
            beta_pr = np.max(np.maximum(0, numerator_pr / denominator_pr))
            S = -J_w + beta_pr * S

        if HestenesStiefel and l >= 1:
            numerator_hs = J_w.T @ (J_w - J_wprev)
            denominator_hs = S.T @ (J_w - J_wprev)
            mali_br = 1e-10
            denominator_hs = np.maximum(mali_br, denominator_hs)
            beta_hs = np.maximum(0, numerator_hs / denominator_hs)
            S = -J_w + beta_hs * S

        W = W + eta*S
        if J<J_prev:
            W_optim = W
        J_wprev = J_w
        J_prev = J
        J_list.append(J)

        time_values.append(iteration_time_values)
        print('Number of iteration %d, eta = %.2f' % (l+1, eta))
        print('Cost function J = %.4f' % (J))

    # omogucena lakša manipulacija
    state_variable_values_mth_iteration = np.array(state_variable_values_mth_iteration)
    control_input_values_mth_iteration = np.array(control_input_values_mth_iteration)
```

```
        time_values = np.array(time_values)

        # Plot
        for i in range(self.n):
            plt.plot(time_values[M - 1], state_variable_values_mth_iteration[:, i],
label=f'x{i + 1}')

        plt.axhline(0, color='black', linestyle='--', linewidth=1)
        plt.xlabel('Vrijeme')
        plt.ylabel('Varijabla stanja')
        plt.legend()
        plt.title(f'Vremenska ovisnost varijable stanja')
        plt.show()

        #svaki zasebno plot za x1 and x2
        fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(8, 6))

        # Plot x1
        ax1.plot(time_values[M - 1], state_variable_values_mth_iteration[:, 0], label='x1')
        ax1.axhline(0, color='black', linestyle='--', linewidth=1)
        ax1.set_ylabel('x1')
        ax1.legend()

        # Plot x2
        ax2.plot(time_values[M - 1], state_variable_values_mth_iteration[:, 1], label='x2',
color='orange')  # Adjust color as needed
        ax2.axhline(0, color='black', linestyle='--', linewidth=1)
        ax2.set_xlabel('Vrijeme')
        ax2.set_ylabel('x2')
        ax2.legend()

        plt.suptitle(f'Vremenska ovisnost varijable stanja')
        plt.show()

        # Plot upravljacka
        for i in range(self.m):
            plt.plot(time_values[M - 1], control_input_values_mth_iteration[:, i], label=f'u{i
+ 1}')

        plt.axhline(0, color='black', linestyle='--', linewidth=1)
        plt.xlabel('Vrijeme')
        plt.ylabel('Upravljačka varijabla')
        plt.legend()
        plt.title(f'Vremenska ovisnost upravljačke varijable')
        plt.show()
        print('Finished')
        return W, W_optim, J_list
```

SKRIPTA 3

```
import numpy as np
import BPTT_method as bptt
import matplotlib.pyplot as plt

A = np.array([[0, 1], [-1, -0.2/1]])
```

```
B = np.array([[0], [1]])

model = bptt.BPTT_method(A = A, B = B)

x0 = np.array([[1], [0]])
Q = np.array([[10, 0], [0, 10]])
R = np.array([[10]])

W, Wopt, J_list = model.main_alg(x0 = x0, tf = 10, N =2000, M = 50, eta = 1.2, bmax = 0.85, Q
= Q, R = R, umax = 3.0, umin = 3.0, SupSab = True, DaiYuan = True,FletcherReeves= False,
PolakRibiere= False, HestenesStiefel= False)

# #CRTANJE GRAFOVA###############
iteration_numbers = np.arange(1, 51)

# Plot J_list
plt.plot(iteration_numbers, J_list, label='Funkcija cilja J')
plt.xlabel('Broj iteracija, M')
plt.ylabel('Funkcija cilja, J')
plt.legend()
plt.title('Funkcija troška J u odnosu na broj itearcija M')
plt.show()
```