Primjena hiperdualnih brojeva za računanje derivacija prvog i drugog reda funkcija više varijabli s primjenama u MATLAB-u

Jonjić, Jakov

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:142030

Rights / Prava: Attribution-NonCommercial-NoDerivatives 4.0 International/Imenovanje-Nekomercijalno-Bez prerada 4.0 međunarodna

Download date / Datum preuzimanja: **2024-05-18**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

DIPLOMSKI RAD

Jakov Jonjić

Zagreb, 2024.

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

DIPLOMSKI RAD

Mentor:
izv. prof. dr. sc. Vladimir Milić, mag. ing.

Student:
Jakov Jonjić

Zagreb, 2024.

Zahvaljujem se svom mentoru prof. dr. sc. Vladimiru Miliću na savjetima, pomoći pri nabavci literature i ukazanom povjerenju.
Također, zahvaljujem se svojoj obitelji na podršci i strpljenju tijekom studiranja.

Izjava

Izjavljujem da sam ovaj rad radio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zagreb, ožujak 2024. Jakov Jonjić

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 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Jakov Jonjić** JMBAG: 0035215218

Naslov rada na hrvatskom jeziku: **Primjena hiperdualnih brojeva za računanje derivacija prvog i drugog reda funkcija više varijabli s primjenama u MATLAB-u**

Naslov rada na engleskom jeziku: **Application of hyper-dual numbers to compute first and second order derivatives of multivariate functions with applications in MATLAB**

Opis zadatka:
Većina algoritama za numeričko rješavanje nelinearnih jednadžbi i nelinearnu optimizaciju zahtijevaju poznavanje derivacija. Mnoge metode za računanje derivacija podložne su greškama zaokruživanja i kraćenja, a posebno u slučaju računanja derivacija drugog reda. Relativno nedavno razvijena je aritmetika hiperdualnih (HD) brojeva čime je omogućeno računanje derivacija prvog i drugog reda sve do strojne preciznosti.
U diplomskom radu je potrebno:

1. Izvesti sve relevantne matematičke izraze i opisati svojstva aritmetike HD brojeva.
2. Istražiti dostupne programske alate u kojima je aritmetika HD brojeva implementirana u obliku klasa za preopterećenje operatora u programskom jeziku MATLAB.
3. Detaljno opisati koncepte i tehnike programiranja korištene klase HD brojeva u MATLAB-u sa stanovišta objektno orijentirane programske paradigme.
4. S klasom dualnih brojeva u MATLAB-u provesti niz numeričkih eksperimenata:
   a. Na dovoljno složenoj funkciji više varijabli provesti usporedbu računanja prvih i drugih derivacija primjenom metode HD brojeva s metodom konačnih diferencija i koračnom kompleksnom metodom.
   b. Za rješavanje sustava nelinearnih jednadžbi Newtonovom metodom pri čemu se matrica Jacobijan računa primjenom HD brojeva.
   c. Za traženje minimuma potencijalne energije tijela obješenog u vertikalnoj ravnini o dva elastična pera Newtonovom metodom pri čemu se matrica Hessijan računa primjenom HD brojeva.
   d. Na primjeru optimalnog upravljanja nelinearnim sustavom s analitičkim rješenjem provesti usporedbu točnosti primjenom HD brojeva s metodom konačnih diferencija za računanje gradijenta funkcije cilja. Ovdje se podrazumijeva za optimizaciju koristiti funkcije koje su ugrađene u MATLAB-ove optimizacijske programske alate.
   e. Kombinacijom primjene klase HD brojeva i MATLAB-ovih optimizacijskih alata za sustav robotskog manipulatora u horizontalnoj ravnini s dva stupnja slobode gibanja provesti sintezu optimalnog upravljanja za slučajeve željenog pozicioniranja u konačnom vremenu i slijeđenja referentne trajektorije.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
18. siječnja 2024.	21. ožujka 2024.	25. – 29. ožujka 2024.

Zadatak zadao: Predsjednik Povjerenstva:

Izv. prof. dr. sc. Vladimir Milić Prof. dr. sc. Ivica Garašić

Sadržaj

Sadržaj v

Popis slika viii

Popis oznaka ix

Sažetak xii

Summary xiii

1. Uvod 1

1.1. Pregled rada po poglavljima 1

1.2. Pregled literature 3

2. Izvod algoritma baziranog na hiperdualnim brojevima za računanje derivacija funkcije 5

2.1. Hiperdualni brojevi 5

2.2. Opis algoritma za računanje derivacija hiperdualnim brojevima 6

3. Tehnike objektnog programiranja korištene u klasi hiperdualnih brojeva 12

3.1. Klasa, objekti i atributi 12

3.2. Metode i pristup atributima 14

3.3. Preopterećenje operatora i funkcija 15

3.4. Mehanizam rada klase za računanje derivacija 17

3.4.1. Izazovi i rješenja . . . 17
3.4.2. Računanje Jacobijana algoritmom 7 . . . 21

4. Usporedba metode hiperdualnih brojeva s metodom konačnih diferencija i koračnom kompleksnom metodom 23
4.1. Usporedba klase HD brojeva za računanje Jacobijana s tradicionalnim metodama . . . 24
4.1.1. Metoda konačnih diferencija . . . 24
4.1.2. Koračna kompleksna metoda . . . 24
4.2. Usporedba klase HD brojeva za računanje Hessijana s tradicionalnim metodama . . . 26
4.2.1. Metoda konačnih diferencija . . . 27
4.2.2. Koračna kompleksna metoda . . . 28
4.2.3. Usporedba metoda u MATLAB-u . . . 30

5. Primjena HD brojeva za računanje derivacija u Newtonovoj metodi traženja nultočke 35
5.1. Računanje sustava nelinearnih jednadžbi primjenom Newtonove metode . 36
5.2. Traženje minimuma potencijalne energije Newton-Raphsonovom metodom . . . 39

6. Primjena hiperdualnih brojeva u optimizacijskim algoritmima u MATLAB-u 44
6.1. Primjena hiperdualnih brojeva u optimizaciji kemijskog procesa . . . 45
6.2. Primjena hiperdualnih brojeva u optimalnom upravljanju 2 DOF robota 49
6.2.1. Transformacija robota iz inicijalnog u željeno stanje . . . 51
6.2.2. Transformacija robota iz inicijalnog u željeno stanje uz praćenje referentne trajektorije . . . 56

7. Zaključak 62

A. Prvi prilog 64
A.1. Kodovi za računanje derivacija iz poglavlja 4 . . . 64
A.2. Kodovi kod korištenja klase HD brojeva u Newtonovoj metodi . . . 69
A.3. Kodovi za rješavanje optimizacijskih problema u MATLAB-u . . . 72
A.3.1. Primjer reaktanta . . . 72

A.3.2. SCARA robot s 2-DOF 73

Literatura **78**

Popis slika

4.1 Usporedba metoda za računanje Jacobijana 26
4.2 Usporedba metoda za računanje Hessijana 31

5.1 Masa obješena na dvije opruge [1] 39
5.2 Vrijednost potencijalne energije ovisno o xy koordinatama sustava 42
5.3 Izolinije vrijednosti potencijalne energije sustava mase na oprugama, 5.1 . 43

6.1 Usporedba analitičkog s rezultatima optimizacije u MATLAB-u 48
6.2 2 DOF SCARA robot, [2] 49
6.3 Varijable stanja SCARA robota za slučaj 1 53
6.4 Varijable stanja SCARA robota za slučaj 1 53
6.5 Varijable stanja SCARA robota za slučaj 1 za vrijeme $t_f = 1.5$ s 54
6.6 Upravljačke varijable SCARA robota za slučaj 1 za vrijeme $t_f = 1.5$ s . . 55
6.7 Varijable stanja SCARA robota za slučaj 2 za vrijeme $t_f = 2$ s. 59
6.8 Upravljačke varijable SCARA robota za slučaj 2. 60
6.9 Praćenje referentne trajektorije za slučaj 2. 61

Popis oznaka

χ dodatna varijabla stanja . . . 52
$\dot{\chi}$ derivacija dodatne varijable stanja . . . 52
$\mathbf{C}$ Coriolisova matrica, [Nm/s ili Ns] . . . 46
E_{trunc} prvi član greške odbacivanja . . . 26
$\mathcal{F}$ dozvoljeni skup . . . 40
$\mathbb{H}$ skup hiperdualnih brojeva . . . 6
$\mathbf{H}$ Hessijan vektorske funckije . . . 6
I_1, I_2 momenti inercije za centre masa robotskih linkova, [kg m^2] . . . 45
$\mathbf{J}$ Jacobijan vektorske funckije . . . 6
J funkcija cilja . . . 46
$K_{b,k}$ koeficijenti funckije cilja . . . 46
$\mathbf{M}$ matrica inercija, [kg/ kgm^2] . . . 46
$\mathbb{N}$ skup prirodnih brojeva . . . 6
$\mathcal{O}(h)$ greška odbacivanja . . . 22
$\mathbb{R}$ skup realnih brojeva . . . 6
T_1, T_2 Momenti motora, [Nm] . . . 46
$V(x, y)$ potencijalna energija mase m, [J] . . . 37
$\mathcal{X}$ projektni skup . . . 40
X_d željena trajektorija u x-osi . . . 52
X_r pozicija vrha linka 2 u x-osi . . . 52
Y_d željena trajektorija u y-osi . . . 52
Y_r pozicija vrha linka 2 u y-osi . . . 52

a	brzina zvuka, [m/s]	59
d	razmak objesišta, [m]	37
f	opća skalarna funckija	6
$f(\mathbf{x})$	Viševarijabilna funckija	6
$\mathbf{f}$	diferencijalna jednadžba sustava	46
$\vec{g}$	opća vektorska funckija	9
g	gravitacijsko ubrzanje, [ms^{-2}]	37
$\mathbf{g}$	ograničenja tipa nejednakosti	41
h	perturbacija za numeričku derivaciju	22
$\mathbf{h}$	ograničenja tipa jednakosti	41
i	imaginarna jedinica	22
$\mathbf{e}_j$	j-ti kartezijski jedinični vektor	8
k_1, k_2	konstante elastičnih pera, [N/m]	37
l_1, l_2	duljine elastičnih pera, [m]	37
l_{c1}, l_{c2}	udaljenosti osi do centara mase linkova, [m]	45
l_1	duljina prvog robotskog linka, [m]	45
m	masa objekta	37
m_1, m_2	mase robotskih linkova, [m]	45
ω	frekvencija, [Hz]	52
$\mathbf{q}$	vektor kuteva zakreta stanja, [rad]	46
q_1, q_2	kutevi zakreta linkova, [rad]	45
$\dot{\mathbf{q}}$	vektor kutnih brzina, [rads^{-1}]	46
$\ddot{\mathbf{q}}$	vektor kutnih ubrzanja, [rads^{-2}]	46
$\mathbf{u}$	vektor upravljačkih varijabli, [Nm]	46
$\mathbf{x}$	vektor varijabli stanja sustava	46
$\dot{\mathbf{x}}$	vektor derivacija varijabli stanja sustava	46
$\widehat{\mathbf{x}}$	Vektorski hiperdualni broj	6
$\mathbf{x}$	projektne varijable	41
$\mathbf{x}^*$	optimalno rješenje	41
δ	konstantni kut robotskog linka, [rad]	45
$\nabla^2 f$	Hessijan funckije f	6
∇f	Jacobijan funckije f	6

ϕ potencijal brzine, $[m^2/s]$ . . . 59
θ fazni kut kompleksne varijable . . . 27

Kratice

HD Hiperdualni . . . 5

Akcenti

$''$ druga derivacija funckije . . . 25
$'$ prva derivacija funckije . . . 22
n, k dimenzija prostora . . . 6
T transponirana matrica . . . 6

Sažetak

U ovom diplomskom radu ostvarena je implementacija aritmetike hiperdualnih brojeva u programskom jeziku MATLAB za računanje prve i druge derivacije funkcije. Pošto je naglasak u radu na praktičnoj primjeni hiperdualnih brojeva, implementacija je ostvarena objektnim programiranjem i uvođenjem algoritma za računanje derivacija. Točnost algoritma je pokazana detaljnim matematičkim izvodom, te su opisane sve tehnike programiranja korištene u izradi klase objekata. Točnost metode je uspoređena s metodom centralne razlike i s koračno kompleksnom metodom i rezultati su prikazani na grafovima. Numerički eksperimenti uključuju klasu hiperdualnih brojeva u računanju Jacobijana i Hessijana u Newtonovoj metodi traženja nultočke, te u ugrađenim optimizacijskim algoritmima u MATLAB-u za računanje prve derivacije funkcije cilja.

Ključne riječi: hiperdualni brojevi, objektno orijentirano programiranje, automatsko računanje derivacija, metoda konačnih diferencija, koračno kompleksna metoda, Newtonova metoda, ugrađeni optimizacijski algoritmi

Summary

In this master's thesis, an implementation of the arithmetic of hyperdual numbers is realized in MATLAB programming language for computing the first and second derivatives of a function. Since the emphasis in the work is on the practical application of hyperdual numbers, the implementation is realised using object-orianted programming and introducing an algorithm for computing the derivatives. The correctness of the method is shown by a detailed mathematical derivation and all programming techniques used in the class of objects are described. The accuracy of the algorithm is compared to the central difference method and the complex-step method and the results are shown on graphs. The numerical experiments include the class of hyperdual numbers in the evaluation of the Jacobian and Hessian in Newton's method of finding the zero and in MATLAB's built-in optimization methods for evaluating the first derivative of the cost function.

Keywords: hyper-dual numbers, object oriented programming, automatic differentiation, finite difference method, complex-step method, Newton's method, built-in optimization algorithms

1 Uvod

U ovom diplomskom radu bavimo se implementacijom hiperdualnih brojeva za računanje derivacija matematičkih funkcija u računalnim programima. Način na koji smo ugradili hiperdualne (ili skraćeno HD brojeve) u programski jezik je preko klase objekata s metodama koje se odnose na osnovne matematičke funckije ili operatore. U ovom radu će se kroz poglavlja predstaviti svi koraci potrebni za razvoj ove klase objekata te će se i numeričkim eksperimentima provjeriti točnost ove metode za numeričko računanje derivacija.

Najveća razlika metode hiperdualnih brojeva za računanje prve i druge derivacije funkcije u odnosu na ostale metode je u tome što je greška svedena na nulu, što znači da je rezultat jednak uvrštavanju vrijednosti varijabli u izraze za derivacije dobivene simboličkim putem. To znači da je ova metoda bazirana na simboličkim operacijama u kombinaciji sa numeričkim vrijednostima varijabli, a implemntacija je ostvarena u MATLAB-u pomoću tehnike objektno orijentiranog programiranja poznate kao preopterećenje operatora (eng. *operator overloading*).

1.1. Pregled rada po poglavljima

U drugom poglavlju ovog rada kreće se od jednostavnog računalnog programa koji automatskim putem, osim vrijednosti funckije kao izlaz daje i vrijednosti derivacija funckije. Ovo služi kao motivacija za uvođenje konkretnog algoritma koji bi funkcionirao i na složenim matematičkim funkcijama. Uz detaljan izvod za dokazivanje točnosti metode za računanje derivacija bazirane na aritmetici HD brojeva, opisan je i

algoritam računanja derivacija skalarne viševarijabilne funkcije $f(\mathbf{x})$ kako bi se mogla izvesti primjena u stvarnom programskom jeziku s mogućnostima objektnog programiranja za preopterećenje operatora. Generalizacija algoritma na složene analitičke funkcije je dokazana matematičkim izvodom nakon čega preostaje ispravno definirati klasu proizvoljnih objekata koji bi tada sa sigurnošću obavljali sve korake algoritma na što efikasniji način.

Budući da je naglasak na praktičnoj primjeni hiperdualnih brojeva u problemima računanja derivacija funkcija, treće poglavlje rada biti će posebno posvećeno analizi i klasifikaciji svih tehnika objektno orijentirane paradigme koje su potrebne za kvalitetnu implementaciju ove metode u MATLAB-u, a može poslužiti i kao podloga za implementaciju ovog paketa u drugim programskim jezicima poput Pythona, C++ ili Julia programskog jezika.

U četvrtom poglavlju uspoređujemo klasu HD brojeva koju smo implementirali u programskom jeziku MATLAB s tradicionalnim metodama za računanje derivacija u nadograđenim verzijama gdje smo im smanjivali grešku oduzimanja i kraćenja. Metoda konačnih diferencija može se poboljšati time što se računa vrijednost funckije u što više različitih točaka i time se red greške odbacivanja povećava, a u ovom radu je odabrana 5-point stencil metoda koja će poslužiti za usporedbu točnosti za računanje druge derivacije sa klasom hiperdualnih brojeva. Nadalje, koračno kompleksna metoda također se može poboljšati u odnosu na svoj osnovni oblik uvođenjem kompleksnog koraka sa faznim kutom koji ovisi o potenciji tj. o redu derivacije funkcije. Na taj način neki članovi kompleksnog koraka iščezavaju i moguće je odbaciti niže članove u redu greške odbacivanja. Kasnije će se pokazati da i u najboljim slučajevima s obzirom na izbor veličine (kompleksnog) koraka, tradicionalne metode nisu u mogućnosti ostvariti strojnu preciznost poput klase hiperdualnih brojeva.

Nadalje, klasa hiperdualnih brojeva ispituje se i na traženju rješenja sustava jednadžbi pomoću Newtonove metode. Poznato je da je Newtonova metoda iteracijski postupak u kojem svaka iduća točka ovisi o vrijednosti derivacije funkcije u prethodnoj točki i da su poželjne što točnije vrijednosti derivacija zbog brzine konvergencije. U ovom primjeru također je pomoću algoritma opisan i stvarni postupak računanja Jacobijana viševarijabilne funkcije metodom HD brojeva. Uz to smo i samu Newtonovu metodu traženja nultočke opisali pomoću algoritma radi lakšeg razumijevanja čitatelju.

Jedna od najvažnijih primjena klase za računanje derivacija je u optimizacijskim algoritmima. Cilj je naći optimalnu vrijednost upravljačke varijable za koju je funkcija cilja minimalna, uz unaprijed zadane tolerancije na optimizacijski algoritam. U MATLAB-u je za ugrađene solvere poput `fmincon` i `fminunc` moguće kao izlaze iz funkcije cilja opskrbiti algoritam i matricama Jacobijan i Hessijan. Ako se one računaju na efikasan način s obzirom na vrijeme izvršavanja, onda je algoritam točniji i brže konvergira željenom rješenju i obično je potrebno manje evaluacija funkcije. To je zato što su ugrađene metode za računanje derivacija u optimizacijskim algoritmima u MATLAB-u osnovane na metodi konačnih diferencija za koju je pokazano da ima ugrađene greške odbacivanja i zaokruživanja. Tehnika deriviranja pomoću hiperdualnih brojeva ispitana je na dva primjera, u prvom se traži optimalna upravljačka varijabla za dobivanje željenih iznosa koncentracija u kemijskom reaktoru, a u drugom primjeru se za robot sa dva stupnja slobode gibanja želi pronaći optimalna upravljačka varijabla koja osigurava najprije pozicioniranje u željeno stanje a u drugom slučaju i praćenje referentne trajektorije vrha drugog linka robotske ruke.

1.2. Pregled literature

U ovom podpoglavlju će se dati pregled literature korištene u izradi diplomskog rada. Najviše referenca u tekstu odnosi se na literaturu [3] i [4] u kojima je zapravo i prvi puta izvedena metoda računanja derivacija bazirana na HD brojevima koja je i korištena u ovom radu. Ipak, to nije prvi pokušaj korištenja HD brojeva za računanje derivacija jer je u radu [5] izvorna metoda predstavljena i poslužila je zapravo u radovima [3] i [4] kao temelj za razvoj aritmetike.

Tehnika automatskog deriviranja također je tema rada u [6] gdje je implementacija objekata koji sadrže vrijednost varijable i derivacije također ostvarena koristeći objektno programiranje definiranjem klase i metoda. Razlika između metode u [6] i metode korištene u ovom radu je što je računanje druge derivacije funckije lakše ostvareno koristeći HD brojeve.

U poglavlju gdje se uspoređuje metoda HD brojeva na primjeru računanja prve i druge derivacije funkcije koristili smo literaturu [7] za pomoć pri načinu usporedbe s tradicionalnim metodama u kojoj smo tražili načine za nadogradnju osnovnih oblika metode kompleksne varijable i metode konačnih diferencija. U literaturi [8] fokus je na

unaprijeđenju osnovnih oblika metoda za računanje druge derivacije funkcije (metoda konačnih diferencija i koračno kompleksna metoda) gdje se greška oduzimanja i odbacivanja smanjila za nekoliko redova veličine i značajno je utjecalo na točnost. Algoritam za opis metode kompleksne varijable za računanje prve derivacije funckije preuzet je iz [9].

U poglavljima s numeričkim eksperimentima gdje se koristi HD klasa za računanje matrice Jacobijan u Newtonovoj metodi traženja nultočke najprije smo uzeli sustav nelinearnih jednadžbi iz predavanja [10]. Uz algoritam za opis Newtonove metode preuzet iz [10], također smo dali i algoritam za opis računanja Jacobijana klasom HD brojeva u MATLAB-u.

Nadalje, numerički eksperiment s Newton-Rhapsonovom metodom zahtjeva računanje Hessijana potencijalne energije sustava mase na oprugama tražeći pritom minimum funckije ovisno o prostornim koordinatama gdje je originalni sustav preuzet iz literature [1] koja razmatra niz fizikalnih pojava s jednadžbama kontinuuma.

U zadnjem poglavlju s numeričkim eksperimentima gdje se klasa HD brojeva koristi u optimizacijskim algoritmima za računanje Jacobijana koristili smo primjer robota iz knjige [2] koja obuhvaća širok spektar metoda sinteze regulatora temeljenih na modelima robota. Primjeri kontrolera objašnjenih u knjizi su PD, PID regulatori, adaptivni i proporcionalni regulatori. Knjiga je podjeljena u četiri djela: dinamika robota, modelsko upravljanje, regulatori za praćenje trajektorije te adaptivni regulatori bez ovisnosti o brzini praćenja.

Na kraju, literatura koja podrobnije obrađuje teme iz numeričke optimizacije je [11]. U ovoj knjizi predstavljene su metode koje su najkorištenije u slučajevima iz prakse. U knjizi su predstavljene i optimizacijske metode koje ne zahtjevaju izračun derivacija i predmet su istraživanja. Još jedna knjiga citirana u ovom radu je [12] u kojoj je naglasak na numeričkoj optimizaciji. Polazi se od osnovnih koncepata poput konveksnosti, uvjeta optimalnosti i Lagrangeovih multiplikatora. U knjizi se obrađuju i tri glavna pristupa rješavanju problema optimalnog upravljanja: Hamilton Jacobi Bellman jednadžba, neposredni pristup preko Pontryagin-ovog principa minimuma i direktni pristupi.

2 Izvod algoritma baziranog na hiperdualnim brojevima za računanje derivacija funkcije

U ovom poglavlju bavimo se razvojem aritmetike i izvodom algoritma baziranog na hiperdualnim brojevima za računanje vrijednosti derivacija funkcija do strojne preciznosti. U radu [5] prvi je puta razvijena aritmetika pomoću hiperdualnih brojeva kojom se uspješno izračunala prva i druga derivacija složene funkcije. Cilj tog rada je bio sačuvati prednosti koračno kompleksne metode kod računanja prve derivacije poput jednostavnosti implementacije i strojne preciznosti ali koja nije mogla zadržati ista svojstva kod računanja druge derivacije funckije zato što ima ugrađenu grešku kraćenja. Stoga se krenulo u razvoj hiperdualnih brojeva kojima (hiper)dualna jedinica kvadriranjem postaje nula što znači da je greška kraćenja jednaka nula i time je vrijednost derivacije točna što će se i pokazati u nastavku poglavlja kroz numerički primjer.

2.1. Hiperdualni brojevi

Hiperdualni ili skraćeno HD brojevi (eng. *hyper-dual numbers*) su proširenje dualnih brojeva, a razvijeni su u radu [5] za računanje druge derivacije funkcije. Prednosti korištenja ove metode u odnosu na koračno kompleksnu ili metodu konačnih diferencija je u tome što se izbjegavaju greške zaokruživanja i kraćenja, a i sama

implementacija je jednostavnija nego u slučaju metode automatskog deriviranja u [6] kod računanja druge derivacije. Razlog zašto je metoda točna do strojne preciznosti kod računanja čak i druge derivacije funckije krije se u činjenici da je dualni dio broja uvijek jednak nula pri potenciranju na potenciju dva ili više, $\epsilon^2 = 0$. To znači da članovi Taylorovog razvoja za neki dualni korak su uvijek nula od treće derivacije. Hiperdualni brojevi koriste također svojstvo da je umnožak dualne i hiperdualne komponente različit od nula: $\epsilon_1\epsilon_2 \neq 0$. Kako je u ovom poglavlju naglasak na praktičnoj primjeni hiperdualnih brojeva u računanju prve i druge derivacije funkcije, koristit će se izmijenjena notacija HD (hiperdualnih) brojeva poput one u radu [3], naspram one predstavljene u izvornome radu [5]. Neka od područja u kojima se traži prva i druga derivacija funkcije su numerička optimizacija i integracija, analiza stabilnosti, rizika i osjetljivosti (eng. *sensitivity analysis*), [4].

U ovom poglavlju detaljno će se obrazložiti i opisati postupak uvođenja automatskog deriviranja u računalnim programima koji su zapravo implementacije matematičkih funkcija. Na primjeru funkcije potenciranja će se algoritmom prikazati princip rada klase HD brojeva za računanje derivacija funkcije. Kako bi se uvjerili u ispravnost ove metode na primjeru viševarijabilne funckije s ugniježđenim pozivima, izvest će se i matematički dokaz da svaka funckija koja prati algoritam HD klase će davati ispravne rezultate bez obzira na složenost, broj varijabli ili strukturu funckije.

2.2. Opis algoritma za računanje derivacija hiperdualnim brojevima

U općem smo slučaju zainteresirani za dobivanje točnih numeričkih vrijednosti derivacija skalarne viševarijabilne funkcije

$$f : \mathbb{R}^n \to \mathbb{R}, \tag{2.1}$$

gdje je $n \in \mathbb{N}$, a $\mathbb{R}$ je skup realnih brojeva, za koju postoje prva i druga derivacija u nekoj točki $\mathbf{x} \in \mathbb{R}^n$. Notacija za zapis Jacobijana skalarne funkcije f je u obliku:

$$\mathbf{J}^\intercal = \nabla f(\mathbf{x})^\intercal = \begin{bmatrix} \dfrac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \dfrac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1\times n}, \tag{2.2}$$

kao i matrice Hessijan

$$\mathbf{H} = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \in \mathbb{R}^{n \times n}. \tag{2.3}$$

Implementacija hiperdualnih brojeva omogućuje izračun Jacobijana i Hessijana funkcije automatskim putem. To znači da je samo potrebno definirati ulazni vektor $\mathbf{x}$ te pozvati funkciju `f` u nekom programskom jeziku koju smo prethodno nadogradili na način da istovremeno računa prvu i drugu derivaciju matematičke funkcije $f(\mathbf{x})$ čiji će izlaz uz samu vrijednost funkcije biti i vrijednost prve odnosno druge derivacije bez dodatnog posredovanja korisnika takve klase. Hiperdualni brojevi sastoje se od četiri člana gdje prvi član predstavlja vrijednost varijable, drugi i treći član sadrže iznos derivacije prvog reda funkcije, dok četvrti član označava vrijednost derivacije funkcije drugog reda. Tako se uvodi sljedeća notacija za zapis vektorske hiperdualne varijable:

$$\begin{aligned} \widehat{\mathbf{x}} = (\mathbf{x}, \delta\mathbf{x}_1, \delta\mathbf{x}_2, \delta\delta\mathbf{x}) &\in \mathbb{H}^n, \\ \mathbf{x}, \delta\mathbf{x}_1, \delta\mathbf{x}_2, \delta\delta\mathbf{x} &\in \mathbb{R}^n, \end{aligned} \tag{2.4}$$

gdje je $k \in \mathbb{N}$, a $\mathbb{H}$ skup hiperdualnih brojeva. Pozivom funkcije `f` (koja je implementacija matematičke funkcije f u nekom programskom jeziku) sa ulaznim argumentom $\widehat{\mathbf{x}}$ očekujemo kao izlaz iz programa hiperdualni skalarni broj:

$$\begin{aligned} \widehat{y} &= f(\widehat{\mathbf{x}}), \\ (y, \delta y_1, \delta y_2, \delta\delta y) &= f(\mathbf{x}, \delta\mathbf{x}_1, \delta\mathbf{x}_2, \delta\delta\mathbf{x}), \end{aligned} \tag{2.5}$$

gdje sada članovi $\delta y_1, \delta y_2$ sadrže vrijednost derivacije funkcije po nekoj od varijabli u $\mathbf{x}$, dok $\delta\delta y$ sadrži vrijednost druge derivacije funckije po nekim varijablama iz $\mathbf{x}$, a naznačene su preko $\delta\mathbf{x}_1, \delta\mathbf{x}_2$. Pritom je potrebno definirati pravila za dodjeljivanje vrijednosti svim članovima hiperdualnog broja $\widehat{y}$ kao izlaza iz programa.

Definicija 1 (Pravilo za preopterećenje funkcija i operatora za rad s hiperdualnim brojevima, [3])**.** *Za skalarnu funkciju f iz (2.1), sljedeće jednadžbe definiraju funkciju*

$f : \mathbb{H}^n \to \mathbb{H}$ *kod poziva* $\widehat{y} = f(\widehat{\mathbf{x}})$*:*

$$y = f(\mathbf{x}), \tag{2.6}$$
$$\delta y_1 = \nabla f(\mathbf{x})^\intercal \cdot \delta \mathbf{x}_1, \tag{2.7}$$
$$\delta y_2 = \nabla f(\mathbf{x})^\intercal \cdot \delta \mathbf{x}_2, \tag{2.8}$$
$$\delta\delta y = \delta \mathbf{x}_1^\intercal \cdot \nabla^2 f(\mathbf{x}) \cdot \delta \mathbf{x}_2 + \nabla f(\mathbf{x})^\intercal \cdot \delta\delta \mathbf{x}. \tag{2.9}$$

Primjena hiperdualnih brojeva na ovaj način zahtjeva definiranje nove klase objekata kao i redefiniranje (eng. *overloading*) svih elementarnih matematičkih funkcija i operatora koji se namjeravaju koristiti s novom klasom objekata. Pritom se služimo MATLAB-ovim mogućnostima za objektno orijentirano programiranje te već gotovim paketom predstavljenim u radu [3].

Na primjeru funkcije potenciranja predstavlja se metoda opisana jednadžbama (2.6-2.9):

$$f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \quad (a, b) \mapsto a^b. \tag{2.10}$$

Nakon računanja same vrijednosti funkcije, algoritam obavlja i računanje matrice Jacobijan i Hessijan:

$$\nabla f(a, b)^\intercal = \begin{bmatrix} b \cdot a^{b-1} & ab \cdot \log(a) \end{bmatrix}, \tag{2.11}$$
$$\nabla^2 f(a, b) = \begin{bmatrix} a^b \cdot \left(\log(a)\right)^2 & a^{b-1} \cdot \left(1 + b \cdot \log(a)\right) \\ a^{b-1} \cdot \left(1 + b \cdot \log(a)\right) & b \cdot (b-1) \cdot a^{b-2} \end{bmatrix}. \tag{2.12}$$

Zatim se prema definiciji 1 računaju vrijednosti izlaza iz programa, $\widehat{y}$. Cijelu medotu na primjeru funkcije potenciranja možemo svesti na sljedeći algoritam iz [4]:

Algoritam 1 Automatsko računanje derivacija funkcije potenciranja

procedure POWER($\langle a \rangle, \langle b \rangle$)
  $\widehat{a} \leftarrow hyperdual(a)$
  $\widehat{b} \leftarrow hyperdual(b)$
  $\widehat{\mathbf{x}} \equiv (\mathbf{x}, \delta\mathbf{x}_1, \delta\mathbf{x}_2, \delta\delta\mathbf{x}) := \langle \begin{pmatrix} a \\ b \end{pmatrix}, \begin{pmatrix} \delta a_1 \\ \delta b_1 \end{pmatrix}, \begin{pmatrix} \delta a_2 \\ \delta b_2 \end{pmatrix}, \begin{pmatrix} \delta\delta a \\ \delta\delta b \end{pmatrix} \rangle$
  Compute $y := f(a,b)$, $\mathbf{J} := \nabla f(a,b)$, $\mathbf{H} := \nabla^2 f(a,b)$.
  $\delta y_1 = \mathbf{J} \cdot \delta\mathbf{x}_1$,
  $\delta y_2 = \mathbf{J} \cdot \delta\mathbf{x}_2$,
  $\delta\delta y = \delta\mathbf{x}_1^\intercal \cdot \mathbf{H} \cdot \delta\mathbf{x}_2 + \mathbf{J} \cdot \delta\delta\mathbf{x}$.
  return $\widehat{y} \equiv \langle y, \delta y_1, \delta y_2, \delta\delta y \rangle$
end procedure

Za funkciju iz (2.10) sada se može dobiti numerički točan iznos matrice Jacobijan i Hessijan zbog toga što su vrijednosti za derivacije definirane analitičkim izrazima. Cjelovite matrice Jacobijan i Hessijan dobivamo višestrukim pozivom funkcije sa argumentima u obliku

$$\widehat{y} = f(\langle \mathbf{x}, \mathbf{e}_j, \mathbf{e}_l, \mathbf{0} \rangle), \tag{2.13}$$

gdje su $\mathbf{e}_j, \mathbf{e}_l$ j-ti i l-ti kartezijski jedinični vektori. Služeći se svojstvom simetričnosti matrice Hessijan otprilike se polovica poziva funkcije može izuzeti na način da pozivamo funkciju f za $j = 1, ..., n$, $l = 1, ..., j$, odnosno broj pozivanja funkcije f je jednak $n(n+1)/2$, gdje je n broj varijabli.

Teorem 1 (Rekurzivno pravilo hiperdualnih brojeva)**.** *Sve funkcije koje se pozivaju unutar matematičke funkcije f (odnosi se na operatore, osnovne funkcije i sve podoperacije unutar istih) moraju biti definirane na način da zadovoljavaju jednadžbe iz Definicije 1. Tada će i f zadovoljavati jednadžbe (2.6-2.9).*

Dokaz. Uzmimo u obzir sljedeće funkcije.

$$f : \mathbb{H}^k \to \mathbb{H}, \tag{2.14}$$

$$g_j : \mathbb{H}^n \to \mathbb{H} \quad j = 1, ..., k, \tag{2.15}$$

$$\psi := f \circ \vec{g} : \mathbb{H}^n \to \mathbb{H}. \tag{2.16}$$

Dovoljno je razmotriti sljedeći ugniježđeni poziv:

$$\widehat{z} = f(g_1(\widehat{\mathbf{x}}), ..., g_k(\widehat{\mathbf{x}})). \tag{2.17}$$

Razlog je što ugniježđeni poziv pokriva sva pravila diferencijalnog računa poput umnoška derivacija, deriviranje razlomaka i lančanog deriviranja.

Vektorska funkcija $\vec{g}$ sastoji se od komponenti $g_j,\ j = 1, ..., k$ te se svaka poziva unutar f. Uz to, uvodimo ψ kao kompoziciju funkcija f i $\vec{g}$ na koju možemo direktno primijeniti jednadžbe (2.6-2.9), dok za funkcije f i g_j jednadžbe iz definicije 1 služe kao definicijski izrazi.

Dokaz se provodi tako da se za ugniježđeni poziv $\widehat{z} := f(\vec{g}(\widehat{\mathbf{x}}))$ dokaže jednakost rezultatu funkcije $\psi(\widehat{\mathbf{x}})$.

Dodjeljujući vrijednost $\widehat{\mathbf{y}} := \vec{g}(\widehat{\mathbf{x}})$, prema jednadžbama (2.6-2.9) dobivamo:

$$\begin{aligned}\widehat{\mathbf{y}} = \langle &\begin{pmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_k(\mathbf{x}) \end{pmatrix}, \begin{pmatrix} \nabla g_1(\mathbf{x})^\intercal \cdot \delta\mathbf{x}_1 \\ \vdots \\ \nabla g_k(\mathbf{x})^\intercal \cdot \delta\mathbf{x}_1 \end{pmatrix}, \begin{pmatrix} \nabla g_1(\mathbf{x})^\intercal \cdot \delta\mathbf{x}_2 \\ \vdots \\ \nabla g_k(\mathbf{x})^\intercal \cdot \delta\mathbf{x}_2 \end{pmatrix}, \\ &\begin{pmatrix} \delta\mathbf{x}_1^\intercal \cdot \nabla^2 g_1(\mathbf{x}) \cdot \delta\mathbf{x}_2 + \nabla g_1(\mathbf{x})^\intercal \cdot \delta\delta\mathbf{x} \\ \vdots \\ \delta\mathbf{x}_1^\intercal \cdot \nabla^2 g_k(\mathbf{x}) \cdot \delta\mathbf{x}_2 + \nabla g_k(\mathbf{x})^\intercal \cdot \delta\delta\mathbf{x} \end{pmatrix}\rangle \\ = \langle \vec{g}(\mathbf{x}), &\mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_1, \mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_2, \delta\mathbf{x}_1^\intercal \cdot \nabla^2\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_2 + \mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\delta\mathbf{x}\rangle. \end{aligned} \tag{2.18}$$

Nadalje, sada je $\widehat{\mathbf{y}}$ ulazni argument za funkciju f. Slijedeći opet jednadžbe iz definicije 1 dobivamo:

$$\begin{aligned}\widehat{z} =&\langle f(\mathbf{y}), \mathbf{J}f(\mathbf{y}) \cdot \delta\mathbf{y}_1, \mathbf{J}f(\mathbf{y}) \cdot \delta\mathbf{y}_2, \delta\mathbf{y}_1^\intercal \cdot \nabla^2 f(\mathbf{y}) \cdot \delta\mathbf{y}_2 + \mathbf{J}f(\mathbf{y}) \cdot \delta\delta\mathbf{y}\rangle \\ =&\langle f(\mathbf{y}), \mathbf{J}f(\mathbf{y}) \cdot \mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_1, \mathbf{J}f(\mathbf{y}) \cdot \mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_2, \\ &\delta\mathbf{x}_1^\intercal \cdot \mathbf{J}\vec{g}(\mathbf{x})^\intercal \cdot \nabla^2 f(\mathbf{y}) \cdot \mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_2 + \mathbf{J}f(\mathbf{y}) \cdot \mathbf{J}\vec{g}(\mathbf{x}) \cdot \delta\delta\mathbf{x} \\ &+ \mathbf{J}f(\mathbf{y}) \cdot \left(\delta\mathbf{x}_1^\intercal \cdot \nabla^2\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_2\right)\rangle, \end{aligned} \tag{2.19}$$

gdje smo koristili notaciju

$$\left(\delta\mathbf{x}_1^\intercal \cdot \nabla^2\vec{g}(\mathbf{x}) \cdot \delta\mathbf{x}_2\right) = \begin{pmatrix} \delta\mathbf{x}_1^\intercal \cdot \nabla^2 g_1(\mathbf{x}) \cdot \delta\mathbf{x}_2 \\ \vdots \\ \delta\mathbf{x}_1^\intercal \cdot \nabla^2 g_k(\mathbf{x}) \cdot \delta\mathbf{x}_2 \end{pmatrix}. \tag{2.20}$$

Služeći se algebrom vektora i diferencijalnim računom možemo napisati sljedeću jednakost:

$$\begin{aligned}\mathbf{J}f(\mathbf{y})\cdot\left(\delta\mathbf{x}_1^\intercal\cdot\nabla^2\vec{g}(\mathbf{x})\cdot\delta\mathbf{x}_2\right) &= \delta\mathbf{x}_1^\intercal\cdot\left(\sum_{j=1}^{k}\frac{\partial f}{\partial x_j}(\vec{g}(\mathbf{x})\cdot\nabla^2 g_j(\mathbf{x}))\right)\cdot\delta\mathbf{x}_2\\ &= \delta\mathbf{x}_1^\intercal\cdot\nabla^2\psi(\mathbf{x})\cdot\delta\mathbf{x}_2. \end{aligned}\tag{2.21}$$

Pojednostavljivanjem izraza (2.19) dobivamo:

$$\widehat{z} = \langle\psi(\mathbf{x}), \mathbf{J}\psi(\mathbf{x})\cdot\delta\mathbf{x}_1, \mathbf{J}\psi(\mathbf{x})\cdot\delta\mathbf{x}_2, \delta\mathbf{x}_1^\intercal\cdot\nabla^2\psi(\mathbf{x})\cdot\delta\mathbf{x}_2 + \mathbf{J}\psi(\mathbf{x})\cdot\delta\delta\mathbf{x}\rangle.\tag{2.22}$$

Ovime smo dokazali jednakost primjenjivanju definicije 1 direktno na kompoziciju ψ. Dakle, kada funkcije f i $\vec{g}$ zadovoljavaju jednadžbe iz definicije 1, tada i kompozicija funkcija $\psi(\mathbf{x}) = f\circ\vec{g}$ također zadovoljava jednadžbe (2.6-2.9) i time je dokaz završen, [3]. □

3 Tehnike objektnog programiranja korištene u klasi hiperdualnih brojeva

U ovom poglavlju detaljno će se opisati koncepti i tehnike korištene u izradi klase hiperdualnih brojeva u MATLAB-u sa stanovišta programiranja. Za početak, može se reći da se ovdje radi o objektno orijentiranoj programskoj paradigmi jer se računanje svodi na operacije nad objektima koji sadrže podatke u poljima koja se još nazivaju i atributi. Metode služe kao upute kojima se manipulira objektima, odnosno podacima sadržanim u poljima prilikom njihove interakcije. Cilj razvoja klase hiperdualnih brojeva je ostvariti praktičnu implementaciju jednadžbi (2.6-2.9), pa će se većina metoda fokusirati na redefiniranje (preopterećenje) osnovnih matematičkih funkcija za rad s hiperdualnim objektima. Ova tehnika također je poznata i kao preopterećenje operatora (eng. *operator overloading*) i predstavlja jednu od najvažnijih tehnika objektnog programiranja. U nastavku će se opisati i potkrijepiti primjerima iz koda sljedeće tehnike objektnog programiranja: klase, objekti, atributi, metode, enkapsulacija, konstruktor klase, preopterećenje operatora i funkcija te polimorfizam. Također, na primjeru funkcije potenciranja čitatelju će se još jasnije predstaviti način rada klase i dodatno povezati s definicijskim jednadžbama. Na kraju, jednadžba (2.13) će se potrkrijepiti primjerima iz koda za slučaj računanja Jacobijana kako bi se objasnio razlog n pozivanja funckije za računanje cjelovitog Jacobijana.

3.1. Klasa, objekti i atributi

Definiranje klase podrazumijeva da se objektima u nekom programskom jeziku pridodaju svojstva i pritom opišu metode u kojima su definirani principi nad podacima

sadržanim u poljima objekta. Kod hiperdualnih brojeva možemo razlikovati četiri polja (atributa): polje za realni rio, dva dualna i jedan hiperdualni, nad kojima kasnije definiramo metode.

Sljedeći kod prikazuje naredbu za definiranje klase i dodjeljivanje svojstava u MATLAB-u:

```
classdef hyperdual

  properties(Access=public)
    x           %realni dio
    dx1         %prva dualna komponenta
    dx2         %druga dualna komponenta
    dx1x2       %hiperdualna komponenta
  end
```

Kod 3.1: klasa u MATLAB-u

U prikazanom kodu može se primjetiti da je opcija za pristup poljima (atributima) klase postavljena kao javno (`'Access=public'`). Razlog je što želimo korisnicima klase dopustiti da kreiraju i mijenjaju vrijednosti u poljima hiperdualnog (HD) objekta preko bilo kojeg vanjskog koda. Suprotno, pristup bi se mogao postaviti na `'Access=private'` i time bi se atributima moglo pristupiti samo unutar klase što je također bitna značajka objektog programiranja, poznata i kao enkapsulacija gdje se podaci klase štite od neželjenog vanjskog utjecaja.

Sljedeća bitna značajka u kreiranju klase objekata je definiranje konstruktora klase. Ta metoda zaslužna je za inicijalizaciju instance klase, te upravo ona povezuje podatke s atributima objekta. Konstruktor klase je funkcija koja ima isto ime kao i klasa, a u nastavku je prikazan kod iz klase u MATLAB-u:

```
methods (Access=public) % class def

  % constructor
  function [lhs] = hyperdual(x,dx1,dx2,dx1x2)
    if(isempty(x) || size(x,1)==0)
      error('input for x must be at least one numerical value.');
    end
    size_x          = size(x);
    if(nargin<2)
      dx1             = zeros(size_x);
```

```
    else
      size_dx1         = size(dx1);
      if(norm(size_dx1-size_x,'inf')~=0)
        error('dx1 has another size than x.');
      end
    end
    lhs.x            = x;
    lhs.dx1          = dx1;
    lhs.dx2          = dx2;
    lhs.dx1x2        = dx1x2;
  end
```

Kod 3.2: Konstruktor klase MATLAB-u

Vidljivo je da je pristup konstruktoru klase zadan kao javno, jer želimo iz proizvoljnog eksternog koda moći kreirati nove instance klase. Ako bismo htjeli na primjer kreirati instancu klase HD brojeva uz postavku pristupa postavljenu na privatno, MATLAB bi javio grešku u izvršavanju jer tada nije moguće stvarati instance klase iz nekog eksternog koda. Razumljivo je kako unosom samo jedne vrijednosti u konstruktor inicijaliziramo konstantu umjesto prave varijable jer su svi ostali članovi hiperdualnog broja jednake nuli, a prema jednadžbama (2.6-2.9) slijedi da su i članovi koji označavaju vrijednost derivacije po konstanti jednaki nuli jer množe nulu.

3.2. Metode i pristup atributima

Nakon što smo uspješno definirali klasu i atribute, konstruktor klase i inicijalizirali objekt, potrebno je definirati način na koji pristupamo podacima tj. atributima hiperdualnih brojeva u ovoj implementaciji. U općem slučaju želimo dohvatiti numeričku vrijednost nekog atributa vektorskog hiperdualnog broja. Metoda zadužena za ponašanje objekata kojima se dohvaćaju podaci naziva se `'subsref'`. U sljedećem kodu prikazan je samo jedan način pristupu podacima, a to je preko točkaste notacije s imenom atributa, nakon kojeg u zagradi unosimo poziciju tražene vrijednosti na način isti kao pristupu elementima vektora.

```
function varargout = subsref(obj,s)
  switch s(1).type
    case '.'
```

```
4       if length(s) == 1
5         % Implement obj.PropertyName
6         varargout{:}     = obj.(s.subs);
7       elseif length(s) == 2 && strcmp(s(2).type,'()')
8         % Implement obj.PropertyName(indices)
9         tmp              = obj.(s(1).subs);
10        varargout{:}     = tmp(s(2).subs{:});
11      else
12        varargout        = {builtin('subsref',obj,s)};
13      end
```

Kod 3.3: Dohvaćanje podataka objekata MATLAB-u

Metoda `'subsref'` ima dva ulazna argumenta, objekt čiji se podaci dohvaćaju i strukturu `'s'` koja sadrži polja `'type'` i `'subs'`. Polje `'type'` može imati vrijednosti `'()'` ili `'.'` što znači da možemo pristupiti podacima preko indeksiranja ili služeći se imenom samog atributa. Polje `'subs'` sadrži vrijednosti ovisne o polju `'type'`. U slučaju zagrada ono sadrži indekse za pristup, a u slučaju točke sadrži ime atributa kao niz znakova (eng. *character array*) tj. string. Drugi `'if'` uvjet pokriva slučaj u kojem želimo dohvatiti podatke vektorskog HD broja na način da prvo označimo atribut, a zatim u zagrade upišemo sve retke i stupce od interesa. U slučaju da se podacima unutar atributa pristupa na neki način kojeg ne obuhvaćaju prva dva `'if'` uvjeta, služimo se ugrađenom `'subsref'` metodom kako bi se korisnicima omogućio pristup na neki drugi način.

3.3. Preopterećenje operatora i funkcija

Pojam preopterećenja (eng. *overloading*) odnosi se na redefiniranje osnovnih matematičkih operacija kako bi korisniku bilo što jednostavnije poslužiti se ovom klasom s postojećim operacijama npr. $(+, -, *, /)$, bez potrebe za učenjem novih funkcija što usput pridonosi jednostavnosti korištenja i efikasnosti klase. Prednost tehnike preopterećenja operatora je što njihovo ponašanje se ne mijenja u slučaju rada sa standardnim objektima tipa double, float, integer i ostali, već ponašanje operatora ovisi o objektima. Tako će funkcije i operatori ostati nepromijenjeni u slučaju rada sa standardnim objektima, ali ćemo mi posebno definirati njihovo ponašanje u slučaju kada su njihovi argumenti hiperdualni objekti. Ova značajka naziva se još i

polimorfizam jer se ponašanje funkcija mijenja ovisno o tipu objekata. Tu se pretpostavlja da su funkcije u klasi ispravno redefinirane i daju točne rezultate s hiperdualnim brojevima sljedeći pritom pravila aritmetike HD brojeva.

Za primjer uzet ćemo relacijski operator veće ili jednako (>=) čiji je izlaz logičko 1 ili 0. U MATLAB-u se funkcija za izvođenje ovog relacijskog operatora naziva `'ge'` i obično namjeravamo uspoređivati samo dva objekta istog tipa, npr. dva hiperdualna broja. Objektno programiranje nam omogućuje da ovaj operator možemo preopteretiti u klasi na način da daje točan rezultat umjesto greške čak i ako uspoređujemo na primjer double i hiperdualni objekt, a način izvedbe u klasi je prikazan u kodu ispod:

```
function [lhs] = ge(rhs1,rhs2)% >=
  if( ~isa(rhs1,'hyperdual') )
    rhs1  = hyperdual(rhs1);
  end
  if( ~isa(rhs2,'hyperdual') )
    rhs2  = hyperdual(rhs2);
  end
  lhs     = (rhs1.x) >= (rhs2.x);
end
```

Kod 3.4: Preopterećenje binarnog operatora

U slučaju da jedan ili drugi objekt nisu hiperdualni brojevi, funkcija `'ge'` osigurava da se kreiraju instance klase, te da se kao izlaz iz programa daje rezultat usporedbe realnih dijelova. Međutim, nakon izvršenja operacije objekt i dalje ostaje istog tipa kao i prije usporedbe, što znači da metoda samo kratkotrajno promjeni tip objekta u hiperdualni broj, ali nakon same operacije varijabla ostaje istog tipa što se može vidjeti u radnom okruženju (eng. *workspace*) u MATLAB-u.

Primjer 1 (*Usporedba dva različita objekta u MATLAB-u*)**.** *Definirajmo prvo vektor s dvije double vrijednosti i hiperdualni broj istih dimenzija te koristimo operator veće ili jednako:*

```
x=[1;2];
y=hyperdual([2;-0.5])
x>=y
ans =
  2x1 logical array

```

```
7    0
8    1
9  >>
```

Kod 3.5: Usporedba HD objekta i vektora s vrijednostima double

Nakon provođenja operacije možemo provjeriti klasu objekta `'x'` naredbom `'class()'` i potvrditi da tip objekta ostaje nepromijenjen:

```
1  >> class(x)
2
3  ans =
4
5  'double'
```

3.4. Mehanizam rada klase za računanje derivacija

Glavni izazov u kreiranju klase hiperdualnih brojeva za automatsko deriviranje funkcija više varijabli bio je razviti mehanizam koji osigurava da se zadovoljavaju jednadžbe (2.6-2.9) jer prema teoremu 1 će i ugniježdene funkcije zadovoljavati iste. To znači da svaku osnovnu funkciju i operaciju koju namjeravamo koristiti moramo preopteretiti na način da je izlaz hiperdualni (skalarni) broj sa odgovarajućim vrijednostima atributa.

3.4.1. Izazovi i rješenja

Jedan od izazova bio je dobro definirati (binarnu) funkciju potenciranja iz jednadžbe (2.10) u klasi HD brojeva. Program treba prepoznati radi li se o funkciji jedne ili dvije varijable. Prema tome razlikujemo tri slučaja:

1. potencijal je varijabla $f(x) = a^x$,
2. baza je varijabla $f(x) = x^b$,
3. i baza i potencijal su varijable $f(x, y) = x^y$.

Objektno programiranje u MATLAB-u nam omogućuje da funkciju potenciranja `'power'` definiramo pomoću tri dodatne funkcije u samoj klasi, ovisno o kojem se

slučaju radi. Za početak, kada korisnik upotrijebi funkciju potenciranja sa barem jednim hiperdualnim argumentom MATLAB poziva funkciju `'power'` u klasi i nameće se da je njena prva zadaća sada odrediti o kojem se od tri slučaja radi. Rješenje ovog koraka prikazano je u kodu ispod:

```
function [lhs] = power(rhs1,rhs2)
  b_hd1           = isa(rhs1,'hyperdual');
  b_hd2           = isa(rhs2,'hyperdual');
  b_exponial      = b_hd2 && (~b_hd1);
  b_monomial      = b_hd1 && (~b_hd2);
  b_powerial      = b_hd1 && b_hd2;
  if( b_exponial )
    lhs             = exponial(rhs1,rhs2);
  end
  if( b_monomial )
    lhs             = monomial(rhs1,rhs2);
  end
  if( b_powerial )
    lhs             = powerial(rhs1,rhs2);
  end
end
```

Kod 3.6: preopterećenje funkcije potenciranja

Varijable `'b_hd1'` i `'b_hd2'` su logičke varijable s vrijednostima `'true'` ili `'false'`. Nakon određivanja tipova argumenata funkcije potenciranja možemo i odrediti o kojem se slučaju radi. Time se služimo `'short-circuit and'` operatorom koji prekida evaluaciju čim je vrijednost prvog argumenta `'false'`.

Najzanimljiviji nam je upravo treći slučaj kada se radi o dvije varijable, te je prilika da prikažemo način rada algoritma 1 u klasi. Prema algoritmu 1 prvi korak je stvoriti izraze za izračunavanje matrica Jacobijan i Hessijan te nekoj drugoj funkciji u klasi prepustiti računanje vrijednosti izlazne varijable $\widehat{y}$.

Kod ispod prikazuje funkciju `'powerial'` čija je zadaća stvaranje objekata iz trećeg koraka algoritma 1, to jest stvaranje izraza za članove matrica Jacobijan i Hessijan:

```
methods (Access=private) % private power functions
  function [lhs] = powerial(rhs1,rhs2)
    % f(x,y):= x^y
    fun_f       = @(x,y)  x.^y;
```

```
    fun_DfDx     = @(x,y)  y.*x.^(y-1);
    fun_DfDy     = @(x,y)  x.^y .* log(x);
    fun_DDfDxx   = @(x,y)  y.*(y-1).*x.^(y-2);
    fun_DDfDxy   = @(x,y)  x.^(y-1).*( 1+y.*log(x) );
    fun_DDfDyy   = @(x,y)  x.^y .* log(x).^2;
    opt_simd     = 'elemental';
    lhs          = fun_scalar_bivariate( fun_f,fun_DfDx,fun_DfDy,...
      fun_DDfDxx,fun_DDfDxy,fun_DDfDyy,rhs1,rhs2,opt_simd );
end
```

Kod 3.7: Funkcija prilagođena slučaju 3

Funkcija `'powerial'` kreira anonimne funkcije za članove matrica derivacija i šalje ih funkciji `'fun_scalar_bivariate'` čija je zadaća prosto načiniti vektore i matrice pomoću uzalnih argumenata koje čine anonimne funkcije iz `'powerial'`. Zatim, od dviju varijabli `'rhs1, rhs2'` kreirati vektorski hiperdualni broj kao što i sugerira korak 2 u algoritmu 1. Razlog zašto je ova metoda postavljena na `'Access=private'` je zato što želimo samo unutar klase koristiti ovu funkciju. Korisnik ne treba znati kako ona funkcionira jer nam služi samo u posebnom slučaju pri potenciranju dvije hiperdualne varijable.

To znači da preostaje izvršiti korake 6-8 i izračunati vrijednost funkcije. U funkciji `'fun_scalar_bivariate'` pozivamo `'fun_scalar'` na način:

```
lhs          = fun_scalar(fun_f,fun_Df,fun_Hf,rhs,opt_simd);
```

Kod 3.8: Poziv funkcije za računanje izlaza iz programa

gdje su `'fun_f'` vrijednost funkcije $y = f(x, y)$, `'fun_Df'` je Jacobijan, a `'fun_Hf'` Hessijan iz koraka 2 algoritma 1. Princip rada funkcije `'fun_scalar'` je prikazan u iscječcima kodova ispod gdje su dani samo najvažniji djelovi za izvođenje algoritma 1.

Funkcija `'fun_scalar'` također je dio privatne metode jer služi samo kao interna funckija klase:

```
function [lhs] = fun_scalar(fun_f,fun_Df,fun_Hf,rhs,opt_simd)
  x                = rhs.x;
  dx1              = rhs.dx1;
  dx2              = rhs.dx2;
  dx1x2            = rhs.dx1x2;
  [n,~,M]          = size(x);    % n is input dimension; M is SIMD page
    dimension
```

```
  % initialise 0th, 1st and 2nd derivative arrays
  f               = zeros(1,1,M);
  Df              = zeros(1,n,M);
  Hf              = zeros(n,n,M);
```

Kod 3.9: Kreiranje izlaznih varijabli

U ovom dijelu koda kreiramo prazne matrice odgovarajućih dimenzija prema broju ulaznih varijabli $n = 2$ u ovom slučaju.

U prazne matrice sada je potrebno spremiti vrijednosti derivacija nakon čega možemo računati komponente izlaza iz programa $\widehat{y}$:

```
f( 1,1,:)       = fun_f(  x(:,1,:) );
Df(1,:,:)       = fun_Df( x(:,1,:) );
Hf(:,:,:)       = fun_Hf( x(:,1,:) );
% inicijaliziranje y,dy1,dy2,ddy
z               = f;
dz1             = zeros(m,1,M);
dz2             = zeros(m,1,M);
dz1z2           = zeros(m,1,M);
% racunanje vrijednosti iz koraka 4-7 algoritma 1
for k1=1:n
  dz1             = dz1   + Df(1,k1,:) .*   dx1(k1,1,:);
  dz2             = dz2   + Df(1,k1,:) .*   dx2(k1,1,:);
  dz1z2           = dz1z2 + Df(1,k1,:) .* dx1x2(k1,1,:);
  for k2=1:n
    dz1z2(1,1,:)    = dz1z2(1,1,:) + dx1(k1,1,:) * Hf(k1,k2,:) * dx2(
   k2,1,:);
  end
end
% izlaz iz programa
lhs         = hyperdual(z,dz1,dz2,dz1z2);
```

Kod 3.10: Računanje koraka 4-7 algoritma 1

Dvije for petlje prikazane u kodu iznad zapravo prikazuju izvođenje koraka 6-8 algoritma 1 gdje se vidi da se dobiva derivacija funkcije po onoj varijabli na čiju je poziciju pohranjena vrijednost 1 u nekom od dualnih dijelova hiperdualnog argumenta $\widehat{\mathbf{x}}$.

3.4.2. Računanje Jacobijana algoritmom 7

Na kraju, uz pretpostavku da je funkcija potenciranja $f(x, y)$ napravljena kao zasebna m-funkcija u MATLAB-u, možemo prikazati računanje matrice Jacobijan kako i sugerira jednadžba (2.13), a preko algoritma 7.

Funkcija potenciranja spremljena je u direktorij kao zasebna m-funkcija:

```
function f = potencija_f(x,y)
f=x(1).^x(2);
end
```

Kod 3.11: Funkcija za računanje potencije

Zatim se u radnom prostoru kreira vektor koji sadrži vrijednosti varijabli x, y te se poziva funkcija 'HD_Jacobian_Call' koja automatskim putem računa vrijednost matrice Jacobian slijedeći algoritam 7 koji se u MATLABU-u realizira na sljedeći način:

```
function Dy = HD_Jacobian_Call(func,x)
  hx          = hyperdual(x);
  n           = size(x,2);
  Dy          = zeros(1,n);
  for j=1:n
    hxj         = hx;
    hxj.dx1(j)  = 1;
    hyj         = func(hxj);
    Dy(1,j)     = hyj.dx1;
  end
end
```

Kod 3.12: Algoritam 7 u MATLAB-u

U kodu ispod prikazano je radno okruženje za navedenu operaciju:

```
x=[2;3]

x =

2
3

>>  HD_Jacobian_Call(@potencija_f,x)
```

```
9
10 ans =
11
12 12.0000    5.5452
```

Kod 3.13: Računanje matrice Jacobijan

4 Usporedba metode hiperdualnih brojeva s metodom konačnih diferencija i koračnom kompleksnom metodom

U praksi je čest problem odabira metode za računanje derivacija funkcije. Metode se razlikuju po stupnju točnosti, težini implementacije u programskim jezicima te u broju operacija (računska efikasnost), [7]. Nakon detaljnog izvoda i dokaza točnosti algoritma temeljenog na hiperdualnim brojevima za računanje derivacija, u MATLAB-u smo uspješno implementirali novu klasu koja ima nadograđene sve matematičke funkcije i operatore na način da slijede jednadžbe (2.6-2.9). U ovom poglavlju ćemo usporediti točnost ove tehnike s ostalim metodama za računanje derivacija funkcije kao što su koračna kompleksna i metoda konačnih diferencija. Prvo će se kod računanja prve derivacije viševarijabilne funkcije prezentirati algoritmi na osnovu metode konačnih diferencija i kompleksne metode, zatim opisati njihove ugrađene greške te na grafu usporediti točnost. Nadalje, kod računanja druge derivacije koristit ćemo istu funkciju te će se također prikazati algoritmi korišteni u MATLAB-u bazirani na tradicionalnim metodama. U ovom djelu detaljnije će se objasniti prirode ugrađenih grešaka za obje tradicionalne metode te će se one poboljšati koristeći formule iz literature. Iznos greške će se prikazati na grafu i usporediti sa analitičkim te rješenjem preko HD klase.

4.1. Usporedba klase HD brojeva za računanje Jacobijana s tradicionalnim metodama

Klasa HD brojeva ovdje će se ispitati na primjeru računanja Jacobijana skalarne multivarijabilne funkcije sa metodom konačnih diferencija ili s obzirom na vrstu razlike koju računamo u brojniku izraza točniji naziv bi bio metoda centralne razlike te sa metodom kompleksne varijable koja ima uklonjenu grešku kraćenja u brojniku.

4.1.1. Metoda konačnih diferencija

Ova metoda zapravo radi na principu definicije derivacije funckije, s tom razlikom što umjesto infinitezimalno male perturbacije h se u programskom jeziku se koriti neka dovoljno mala veličina. Formula koja će se koristiti u ovom radu naziva se centralna razlika:

$$f' = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h), \tag{4.1}$$

Ideja je da se derivacija aproksimira računajući vrijednosti funckije u dvije različite točke i djeljenjem sa ukupnim korakom dobivamo aproksimaciju derivacije. Ova metoda zbog svoje konstrukcije ima ugrađene dvije vrste grešaka, a to su greška oduzimanja (oduzimanje bliskih brojeva u brojniku izraza) te greška zaokruživanja koja se odnosi na ostatak Taylorovog reda razvoja funckije, a detaljnije će o greškama biti opisano u sljedećem potpoglavlju gdje se računa matrica Hessijan.

4.1.2. Koračna kompleksna metoda

Nešto manje intuitivan pristup računanju prve derivacije funkcije je primjenom kompleksnih koraka hi gdje je $i^2 = -1$. Ako skalarnu multivarijabilnu funckiju f razvijamo u Taylorov red s imaginarnim korakom dobivamo sljedeće:

$$f(\mathbf{x} + ih\mathbf{e}j) = f(\mathbf{x}) + ih\nabla_{\mathbf{x}} f(\mathbf{x}) \cdot \mathbf{e}_j + \frac{(ih)^2}{2!}\mathbf{e}_j^T \cdot \nabla_{\mathbf{x}}^2 f(\mathbf{x}) \cdot \mathbf{e}_j + ..., \tag{4.2}$$

a rješavanjem po gradijentu $\nabla_{\mathbf{x}} f(\mathbf{x})$:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) \cdot \mathbf{e}_j = \frac{\mathfrak{Im}\{f(\mathbf{x} + ih\mathbf{e}_j)\}}{h} + \mathcal{O}(h^2). \tag{4.3}$$

gdje je $\mathbf{e}_j$ jedinični vektor odgovarajuće dimenzije u kojemu je $j-$ti element jednak 1.

Sada se u jednadžbi (4.3) u brojniku ne pojavljuje oduzimanje i time je greška kraćenja uklonjena u ovoj metodi. Korak h se može odabrati tako da bude izuzetno mali ($h = 10^{-100}$) i derivacije se može izračunati do strojne preciznosti, [9].

Algoritam baziran na metodi centralne razlike koji se koristi u ovom radu, a napisan je u MATLAB-u je prikazan ispod:

Algoritam 2 Algoritam temeljen na metodi centralne razlike

Ulaz: f, x, h, n

Izlaz: $dfdx$

$dx \leftarrow zeros(n, 1)$
for $i = 1 : n$ **do**
    $ss \leftarrow zeros(n, 1)$
    $ss(i) \leftarrow 1$
    $f_1 \leftarrow f(x - h \cdot ss)$
    $f_2 =\leftarrow f(x + h \cdot ss)$
    $dfdx(i) \leftarrow (f_2 - f_1)/(2h)$
end for

Implementacija kompleksne metode može biti vrlo jednostavno ostvarena u nekom višem programskom jeziku gdje su kompleksne varijable unaprijed definirane. U ovom radu koristi se sljedeći algoritam baziran na metodi kompleksne varijable:

Algoritam 3 Algoritam temeljen na metodi kompleksne varijable

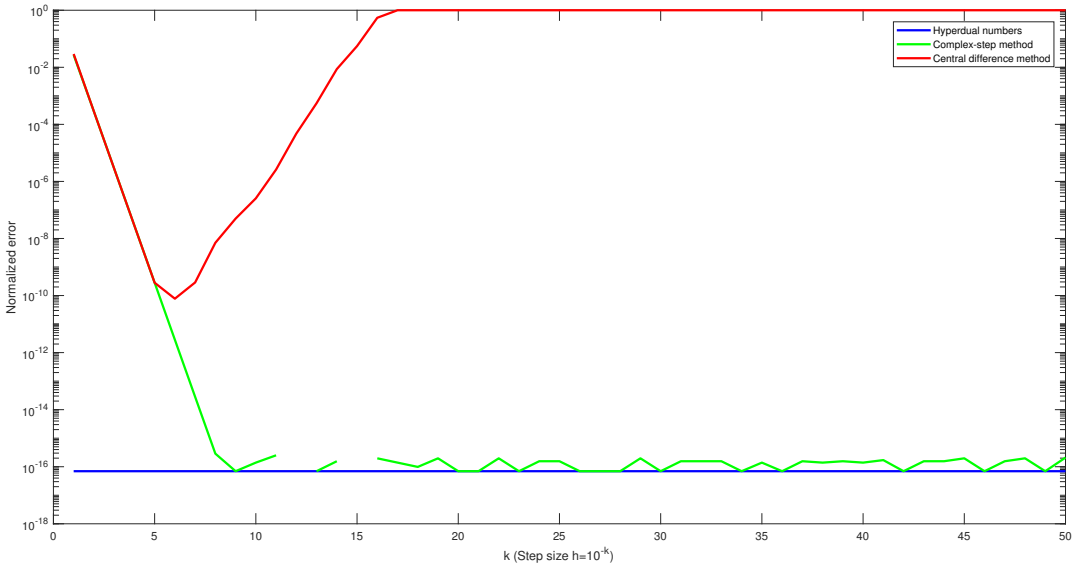
Ulaz: f, x, h, n

Izlaz: $dfdx$

$dx \leftarrow zeros(n, 1)$
$\mathbf{x}_1 \leftarrow \mathbf{x}$
for $i = 1 : n$ **do**
    $x1_j \leftarrow complex(x_j, h)$
    $f1 \leftarrow f(\mathbf{x}1)$
    $dfdx(i) \leftarrow \frac{imag(f1)}{h}$
    $\mathbf{x}1 \leftarrow \mathbf{x}$
end for

Računanje Jacobijana koristeći HD brojeve obavlja se algoritmom 7 te se metode

ispituju na računanju parcijalnih derivacija funckije (4.19) za raspon koraka $h = 10^{-k},\ k = 1, ..., 50$. Rezultati usporedbe su prikazani na slici ispod:



Slika 4.1: Usporedba metoda za računanje Jacobijana

Na slici 4.1 može se iščitati da se rješenje preko HD metode podudara s rješenjem koristeći simboličku derivaciju s uvrštavanjem vrijednsoti varijabli u MATLAB-u. Nadalje, koračno kompleksna metoda poboljšava točnost s manjim izborom koraka h te se oko vrijednosti $k = -9$ počinje podudarati s analitičkim rješenjem. Razlog zašto metoda centralne razlike daje lošije rezultate što se vrijednost koraka smanjiva ispod vrijednosti $h = 10^{-6}$ je zato što greška kraćenja prevladava u odnosu na grešku odbacivanja i stoga postoji optimalna vrijednost izbora koraka kod ove metode.

4.2. Usporedba klase HD brojeva za računanje Hessijana s tradicionalnim metodama

U nastavku će se usporediti metoda hiperdualnih brojeva s metodom konačnih diferencija i koračnom kompleksnom metodom na primjeru računanja druge derivacije funkcije više varijabli.

4.2.1. Metoda konačnih diferencija

Metoda konačnih diferencija (eng. *finite-difference method*) uključuje korištenje koraka h kojim se računa vrijednost funkcije u (najmanje) dvije različite točke i time se aproksimira vrijednost druge derivacije funkcije u točki x. Dvije od najčešćih varijacija ove metode su s pozitivnim korakom:

$$f''(x) = \frac{f(x+2h) - 2f(x+2h) + f(x)}{h^2} + \mathcal{O}(h), \tag{4.4}$$

i aproksimacija preko centralne razlike (eng. *central-difference approximation*):

$$f''(x) = \frac{f(x+2h) - 2f(x) + f(x-2h)}{4h^2} + \mathcal{O}(h^2). \tag{4.5}$$

Izvod ovih relacija dobiva se razvojem Taylorovog reda:

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{1}{2!}h^2 f''(x) \pm \frac{1}{3!}h^3 f'''(x) + \dots . \tag{4.6}$$

Ova metoda je jednostavna za ostvariti u programskom jeziku ali uključuje dva izvora grešaka. Prvi je povezan s greškom odbacivanja, a to su svi članovi višeg reda koje odbacujemo. Greška odbacivanja jednaka je $\mathcal{O}(h)$ za metodu s pozitivnim korakom, (4.4), te $\mathcal{O}(h^2)$ za aproksimaciju preko centralne razlike, (4.5). Na ovu vrstu greške možemo utjecati odabirom jako malog koraka h pri čemu brzina pada greške odbacivanja ovisi o njenom redu. Na primjer, za metodu centralne razlike (4.5) smanjivanjem koraka h za jedan red veličine, greška bi pala za dva reda veličine. Na ovu grešku također možemo utjecati i tako što računamo vrijednosti funkcije u više točaka i pravilnim izborom koeficijenata u brojniku izraza za računanje druge derivacije se mogu pokratiti članovi nižeg reda pa time greška odbacivanja dosta brže opada kako se smanjuje vrijednost koraka h. Drugi izvor greške naziva se greška zaokruživanja, a događa se kada se oduzimaju dva bliska broja. Poznato je da se brojevi u memoriji računala zapisuju preko konačnog broja znamenaka, a pošto bliksi brojevi u brojniku formule imaju sve više jednakih znamenaka što se korak h smanjuje, to će i rezultat operacije imati manji broj znamenaka preciznosti. Krajnji nepoželjan slučaj je kada se za vrijednost koraka h uzima vrijednost koja je manja od strojne preciznosti kada je zapis vrijednosti x i $x + h$ identičan u računalu, kao i vrijednosi funkcije u tim točkama što rezultira s vrijednošću nula za iznos derivacije. S obzirom na ove dvije greške potrebno je odabrati optimalnu vrijednost koraka h kako bi se

izbjegla greška zaokruživanja ali i osiguralo da je greška odbacivanja istovremeno što manja jer su u oprečnom odnosu s obzirom na izbor koraka h.

Varijacija ove metode koju koristimo u ovom radu za usporedbu točnosti s klasom hiperdualnih brojeva nastoji povećati red greške odbacivanja. Kako smo već rekli, ona iznosi $\mathcal{O}(h)$ i $\mathcal{O}(h^2)$ za jednadžbe (4.4) i (4.5), respektivno. Metoda je poznata kao 5-point stencil i u njoj se računa vrijednost funkcije u 5 različitih točaka i tako daje aproksimaciju za drugu derivaciju u obliku:

$$f''(x) = \frac{-f(x-2h) + 16f(x-h) - 30f(x) + 16f(x+h) - f(x+2h)}{12h^2}, \tag{4.7}$$

$$E_{trunc}(h) = \frac{h^6}{90} f^{(6)}(x), \tag{4.8}$$

prema [8], gdje E_{trunc} označava prvi član greške odbacivanja koja je u ovom slučaju $\mathcal{O}(h^6)$. U nastavku će se prikazati i algoritam koji se koristi za računanje derivacija viševarijabilne funkcije koristeći ovu metodu.

4.2.2. Koračna kompleksna metoda

Iduća tehnika za usporedbu računanja numeričke vrijednosti derivacije funkcije je koračna kompleksna metoda (eng. *complex-step method*). U ovom slučaju se za korak uzima kompleksna vrijednost hi, gdje je i imaginarna jedinica: $i^2 = -1$. Razvojem Taylorovog reda za ovakav izbor koraka dobivamo:

$$f(x+hi) = f(x) + ihf'(x) - \frac{h^2}{2!}f''(x) - i\frac{h^3}{3!}f^{(3)}(x) + \frac{h^4}{4!}f^{(4)}(x) + \dots . \tag{4.9}$$

Ako bismo grupirali ovaj red na realni i imaginarni dio dobivamo sljedeće:

$$f(x+hi) = \left(f(x) - \frac{h^2}{2!}f''(x) + \frac{h^4}{4!}f^{(4)}(x) + \dots \right) + h\left(f'(x) - \frac{h^2}{3!}f^{(3)}(x) + \dots \right) i. \tag{4.10}$$

Da bismo dobili numeričku vrijednost druge derivacije funkcije u točki x potrebno je uzeti realni dio Taylorovog razvoja:

$$\Re\{f(x+hi)\} = f(x) - \frac{h^2}{2!}f''(x) + \frac{h^4}{4!}f^{(4)}(x) + \dots . \tag{4.11}$$

Rješavanjem po $f''(x)$ dobivamo

$$f''(x) = \frac{2(f(x) - \mathrm{Re}[f(x+hi)])}{h^2}, \; E_{trunc}(h) = \frac{h^2}{12}f^{(4)}(x), \tag{4.12}$$

gdje je E_{trunc} prvi član greške odbacivanja koji služi za mjeru reda greške odbacivanja u numeričkim tehnikama deriviranja. Ova tehnika ipak ima ugrađenu grešku oduzimanja bliskih brojeva i smanjivanjem kompleksnog koraka hi možemo dobiti u najgorem slučaju oduzimanje dva jednaka broja u zapisu računala te je vrijednost druge derivacije pogrešna. Ovo pak nije slučaj u računanju prve derivacije funkcije za isti kompleksni korak. Ako pogledamo jednadžbu (4.10) onda vidimo da je prva derivacija funkcije vodeća vrijednost u imaginarnom dijelu razvoja te za njeno računanje nije potrebno oduzimanje već samo gledanje imaginarnog dijela vrijednosti funkcije u točki $(x + hi)$. Također, sve iduće vrijednosti u razvoju množe se sa korakom h na neku potenciju, što znači da odabirom dovoljno male veličine koraka h na strojnu se preciznost može dobiti vrijednost prve derivacije što ovu metodu čini *efektivno egzaktnom*, ali samo u slučaju dobivanja vrijednosti prve derivacije funkcije. Ovakvim razmatranjem uvodi se potreba za odabirom metode koja uključuje korištenje druge vrste brojeva s nekoliko *ne-realnih* jedinica na čijem bi vodećem mjestu bile vrijednosti željenih derivacija, a to su upravo hiperdualni brojevi, jednadžba (2.5).

Na grešku odbacivanja kod ove metode može se utjecati odabirom povoljnijeg oblika kompleksnog koraka. Cilj je da se zbrajanjem/oduzimanjem vrijednosti funkcija u različitim točkama s povoljnim oblikom kompleksong koraka pokuša pokratiti što više članova razvoja i tako povećati red greške odbacivanja. Tako se uvodi korak s imaginarnom jedinicom na potenciju u obliku razlomka dva racionalna broja $i^{\frac{p}{q}}$. U kompleksnoj ravnini ova potencija označava i fazni kut gdje je $\theta = \frac{p}{q}90° = \frac{p}{2q}\pi$ rad. Iz trigonometrije imamo relaciju $i^{\frac{p}{q}} = e^{i\theta}$, [8]. Razvoj taylorovog reda za vrijednost funkcije u točki x sa korakom $i^{\frac{p}{q}}h$ je

$$f(x + e^{i\theta}h) = f(x) + \sum_{n=1}^{\infty} e^{ni\theta}\frac{h^n}{n!}f^{(n)}(x), \tag{4.13}$$

$$f(x + e^{i(\theta+\pi)}h) = f(x) + \sum_{n=1}^{\infty} e^{ni(\theta+\pi)}\frac{h^n}{n!}f^{(n)}(x), \tag{4.14}$$

gdje je $e^{i(\theta\pm\pi)} = -e^{i(\theta)}$. Također, postoji i još jedan koristan oblik zapisa imaginarne jedinice na potenciju, a to je preko Eulerove jednadžbe $e^{i\theta} = \cos(\theta) + i\sin(\theta)$. Na ovaj način može se odmah razdvojiti imaginarni od realnog dijela razvoja reda. Sumiranjem

jednadžbi (4.13,4.14) i uvođenjem Eulerove jednadžbe dobivamo sljedeće:

$$f(x+e^{i\theta}h)+f(x+e^{i(\theta+\pi)}h)=2f(x)+2\sum_{n=1}^{\infty}\left[\cos(2n\theta)+i\sin(2n\theta)\right]\frac{h^{2n}}{(2n)!}f^{(2n)}(x). \tag{4.15}$$

Rješavanjem jednadžbe (4.15) za $f''(x)$ dobivamo

$$f''(x)=\frac{f(x+e^{i\theta}h)-2f(x)+f(x+e^{i(\theta+\pi)}h)}{[\cos(2\theta)+i\sin(2\theta)]h^2}+\ldots \tag{4.16}$$
$$-2\sum_{n=2}^{\infty}\left[\cos[(2n-2)\theta]+i\sin[(2n-2)\theta]\right]\frac{h^{2n-2}}{(2n)!}f^{(2n)}(x).$$

Ako odvajanje realnog od imaginarnog dijela nije od značaja, dostupna nam je i kompaktnija jednadžba u obliku

$$f''(x)=\frac{f(x+e^{i\theta}h)-2f(x)+f(x-e^{i\theta}h)}{(e^{i\theta}h)^2}-2\sum_{n=2}^{\infty}\frac{(e^{i\theta}h)^{2n-2}}{(2n)!}f^{(2n)}(x). \tag{4.17}$$

Sada je moguće izračunati drugu derivaciju funckije uzimajući samo imaginarni dio i uz odabir faznog kuta u iznosu $\theta=45°$:

$$f''(x)=\frac{\Im[f(x+e^{i\theta}h)+f(x+e^{i(\theta+\pi)}h)]}{h^2},\ E_{trunc}(h)=\frac{h^4}{360}f^{(6)}(x). \tag{4.18}$$

I time je ostvarena zadaća uvođenja kompleksnog koraka u odnosu na standardnu koračnu kompleksnu metodu jer smo postigli da je greška kraćenja sada reda jednakog $\mathcal{O}(h^4)$, u odnosu na grešku kraćenja iz jednadžbe (4.12) koja iznosi $\mathcal{O}(h^2)$. Razlog je što gledajući samo imaginarni dio jednadžbe (4.15) za vrijednost $n=2$ je $\sin(4\theta)=0$ za vrijednost kuta $\theta=45°$ i prvi član koji se pojavljuje je za vrijednost $n=3$. Iako numerički je ova metoda napredak u odnosu na osnovnu koračnu kompleksnu jednadžbu, ona je i dalje podvrgnuta greški kraćenja bliskih brojeva.

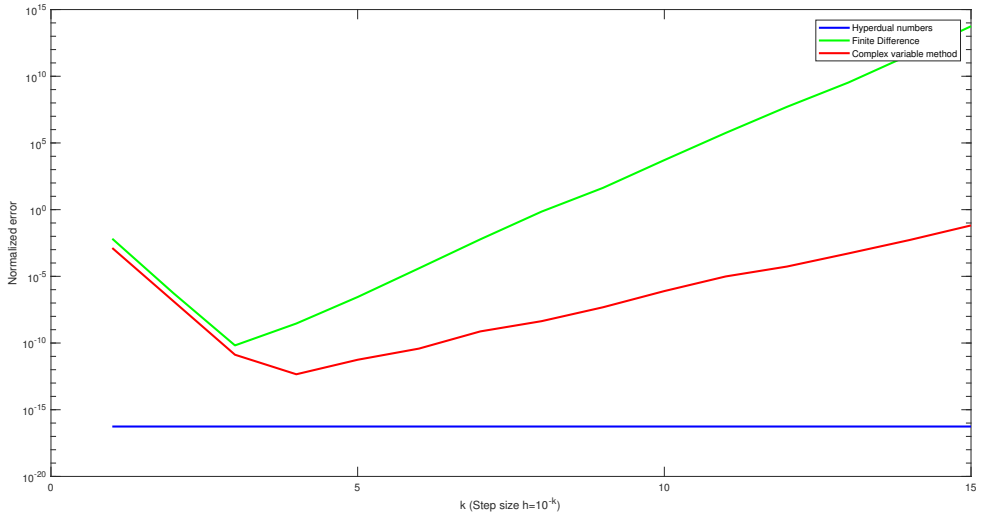
4.2.3. Usporedba metoda u MATLAB-u

U MATLAB-u su implementirane koračna kompleksna i metoda konačnih diferencija i uspoređuju se s klasom hiperdualnih brojeva za računanje druge derivacije viševarijabilne funkcije

$$f(x_1,x_2,x_3)=\frac{x_1x_2\sin(x_3)+e^{x_1x_2}}{x_3}. \tag{4.19}$$

U MATLAB-u je moguće dobiti i analitičke izraze za derivacije sa simboličkim varijablama, a naredbom '`subs()`' moguće je uvrstiti i dobiti numeričke vrijednosti za članove matrice Hessijan. Time imamo i točnu vrijednost s kojom možemo uspoređivati pojedine metode. Način usporedbe je da se izračuna maskimalna singularna vrijednost matrice dobivene oduzimanjem matrice Hessijan dobivene analitičkim putem od matrice izračunate preko pojedine metode, te da se podjeli sa maksimalnom singularnom vrijednošću simboličke matrice s numeričkim vrijednostima.

Prema jednadžbama (4.7) i (4.16) točnost ovisi o koraku h, pa će se prema tome uzeti korak $h = 10^{-i}$, gdje je $i = 1, ..., 15$. Način prikaza rezultata je da se na x-osi nalaze vrijednosti koraka h, a na y-osi je u logaritamskoj skali prikazana greška u odnosu na analitičku metodu.



Slika 4.2: Usporedba metoda za računanje Hessijana

Na slici 4.2 prikazani su rezultati usporedbe triju metoda i očekivano, klasa hiperdualnih brojeva se ne razlikuje od rješenja preko uvrštavanja vrijednosti varijabli u analitičke izraze dobivene simboličkim putem u MATLAB-u. Kod metode konačnih diferencija u početku imamo pad greške a povezan je uz smanjivanje greške odbacivanja kako smanjujemo korak h. Ipak, kod vrijednosti $k = 3$ imamo i optimalnu vrijednost koraka jer se daljnjim smanjivanjem koraka povećava greška zaokruživanja ondosno oduzimamo brojeve sa sve više jednakih znamenaka u zapisu računala. Kod koračne kompleksne metode pad greške također je povezan sa smanjivanjem koraka i

optimalna vrijednost se postiže za vrijednost $h = 10^{-4}$. Daljnjim smanjivanjem koraka opet imamo sve veću grešku kraćenja koja sporije raste nego kod metode konačnih diferencija.

Algoritmi u kojima su ostvarene metode konačnih diferencija i koračno kompleksna metoda prikazani su ispod a povezani su s jednadžbama (4.7) i (4.18), respektivno:

Algoritam 4 Računanje Hessijana metodom konačnih diferencija

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

Izlaz: H

$H \leftarrow zeros(n, n)$
$f \leftarrow f(x)$
for $l = 1$ to n **do**
    $ss \leftarrow zeros(n, 1)$
    $ss(l) \leftarrow 1$
    $ff1 \leftarrow f(x - 2h \cdot ss)$
    $ff2 \leftarrow f(x - h \cdot ss)$
    $ff3 \leftarrow f(x + h \cdot ss)$
    $ff4 \leftarrow f(x + 2h \cdot ss)$
    $H(l, l) \leftarrow 1/(12h^2) \cdot (-ff1 + 16ff2 - 30f + 16ff3 - ff4)$
end for
$lam \leftarrow 1$
$l \leftarrow n - 1$
while l>0 **do**
    for $p = 1 : l$ **do**
        $vector \leftarrow zeros(n, 1)$
        $vector(p : p + lam) \leftarrow 1$
        $ff1 \leftarrow f(x - 2h \cdot vector)$
        $ff2 \leftarrow f(x - h \cdot vector)$
        $ff3 \leftarrow f(x + h \cdot vector)$
        $ff4 \leftarrow f(x + 2h \cdot vector)$
        $H(p, p + lam) \leftarrow (1/(12h^2) \cdot (-ff1 + 16ff2 - 30f + 16ff3 - ff4) - sum(sum(H(p : p + lam, p : p + lam)))/2$
        $H(p + lam, p) \leftarrow H(p, p + lam)$
    end for
    $l \leftarrow l - 1$
    $lam \leftarrow lam + 1$
end while

Algoritam 5 Računanje Hessijana metodom kompleksnog koraka

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

Izlaz: H

$H \leftarrow zeros(n, n)$
for $l = 1$ to n **do**
    $ss \leftarrow zeros(n, 1)$
    $ss(l) \leftarrow 1$
    $xx1 \leftarrow x + hi^{(1/2)} \cdot ss$
    $xx2 \leftarrow x + hi^{(5/2)} \cdot ss$
    $ff1 \leftarrow f(xx1)$
    $ff2 \leftarrow f(xx2)$
    $H(l, l) \leftarrow 1/h^2 \cdot \mathrm{Im}[ff1 + ff2]$
end for
$lam \leftarrow 1$
$l \leftarrow n - 1$
while l>0 **do**
    for $p = 1 : l$ **do**
        $vector \leftarrow zeros(n, 1)$
        $vector(p : p + lam) \leftarrow 1$
        $ff1 \leftarrow f(x + hi^{(1/2)} \cdot vector)$
        $ff2 \leftarrow f(x + hi^{(5/2)} \cdot vector)$
        $H(p, p + lam) \leftarrow (1/h^2 \cdot \mathrm{Im}[ff1 + ff2] - \mathrm{sum}(\mathrm{sum}(H(p : p + lam, p : p + lam))))/2$
        $H(p + lam, p) \leftarrow H(p, p + lam)$
    end for
    $l \leftarrow l - 1$
    $lam \leftarrow lam + 1$
end while

5 Primjena HD brojeva za računanje derivacija u Newtonovoj metodi traženja nultočke

Tehniku HD brojeva ispitujemo na dva primjera traženja minimuma funkcija. U prvom slučaju koristimo Newtonovu metodu za traženje rješenja sustava nelinearnih jednadžbi gdje se matrica Jacobijan računa koristeći klasu HD brojeva. Newtonova metoda daje dobra rješenja uz uvjet da je početna točka iteracije dovoljno blizu konačnom rješenju. U našem slučaju se kao zadovoljavajuće rješenje smatra ono u kojem je norma razlike zadnjih dviju točaka iteracije manja od iznosa tolerancije. Metoda će također biti opisana i algoritmom kako bi se čitatelju jednostavnije prikazala Newtonova metoda, a korisno je i što je u tom slučaju lakše napraviti vlastitu implementaciju u bilo kojem programskom jeziku.

U drugom slučaju koristimo Newton-Rhapsonovu metodu za traženje minimuma potencijalne energije sustava mase na oprugama. U ovom problemu razmatramu masu obješenu u vertikalnoj ravnini o dva elastična pera s određenim razmakom te postavljamo jednadžbu potencijalne energije mase m. Koristeći definicije o stabilnoj ravnoteži iz predavanja Računalne matematike, potencijalna energija će ovisiti o koordinatama u vertikalnoj ravnini, a njen minimum je u točki za koju vrijedi da je prva derivacija jednaka nuli. To znači da je algoritam kojim se služimo po prirodi isti kao onaj u prvom slučaju, s tim da je ovdje potrebno računati i drugu derivaciju viševarijabilne funckije (Hessijan). Također ćemo prikazati algoritam kako bi čitatelj lakše razumio metodu traženja rješenja ili napravio vlasitiu implementaciju.

5.1. Računanje sustava nelinearnih jednadžbi primjenom Newtonove metode

Metodu deriviranja pomoću hiperdualnih brojeva ispitujemo na zadatku pronalaženja rješenja sustava nelinearnih jednadžbi, [10], zadanih u obliku

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0, \\ f_2(x_1, x_2, ..., x_n) &= 0, \\ &\vdots \\ f_m(x_1, x_2, ..., x_n) &= 0. \end{aligned} \tag{5.1}$$

Jacobijan vektorske funkcije $\mathbf{f}(\mathbf{x})$ sada je matrica čiji su elementi $\frac{\partial f_i(\mathbf{x})}{\partial x_j}$, $i = 1, ..., m$, $j = 1, ..., n$, ili matrično

$$\nabla_x \mathbf{f} = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \nabla_x f_1(\mathbf{x}) \\ \nabla_x f_2(\mathbf{x}) \\ \vdots \\ \nabla_x f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \frac{\partial f_m(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix}. \tag{5.2}$$

Iteracijski postupak Newtonove metode je sada definiran u matričnom obliku na sljedeći način

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\frac{\partial \mathbf{f}(\mathbf{x}_k)}{\partial \mathbf{x}_k}\right]^{-1} \cdot \mathbf{f}(\mathbf{x}_k),\ k = 0, 1, 2, ..., \tag{5.3}$$

pri čemu su

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \tag{5.4}$$

dok je $\left[\frac{\partial \mathbf{f}(\mathbf{x}_k)}{\partial \mathbf{x}_k}\right]^{-1}$ inverz matrice Jacobijan uz $n = m$. Sustav nelinearnih jednadžbi zadan je kao:

$$\begin{aligned} 3x_1 - \cos(x_2 x_3) - \frac{1}{2} &= 0, \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 &= 0, \\ e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0. \end{aligned} \tag{5.5}$$

Dok je za početnu točku iteracije zadano $x_0 = \begin{bmatrix}0.1, 0.1, 0.1\end{bmatrix}^\intercal$. Način rješavanja zadatka može se svesti na sljedeći algoritam:

Algoritam 6 Rješavanje sustava nelinearnih jednadžbi Newtonovom metodom

Ulaz: f, x_0, tol

Izlaz: x

$x \leftarrow x_0$
while $err > tol$ **do**
    $J \leftarrow \nabla f(x)$ ▷ Ovdje se koristi HD metoda
    $f_1 \leftarrow f(x)$
    $x_1 = x - J^{-1} \cdot f_1$
    $err = norm_2(x_1 - x)$
    $x \leftarrow x_1$
end while

Računanje matrice Jacobijan metodom hiperdualnih brojeva također se može opisati algoritmom po uzoru na jednadžbu (2.13).

Algoritam 7 Računanje Jacobijana metodom HD brojeva

Ulaz: f, x

Izlaz: J

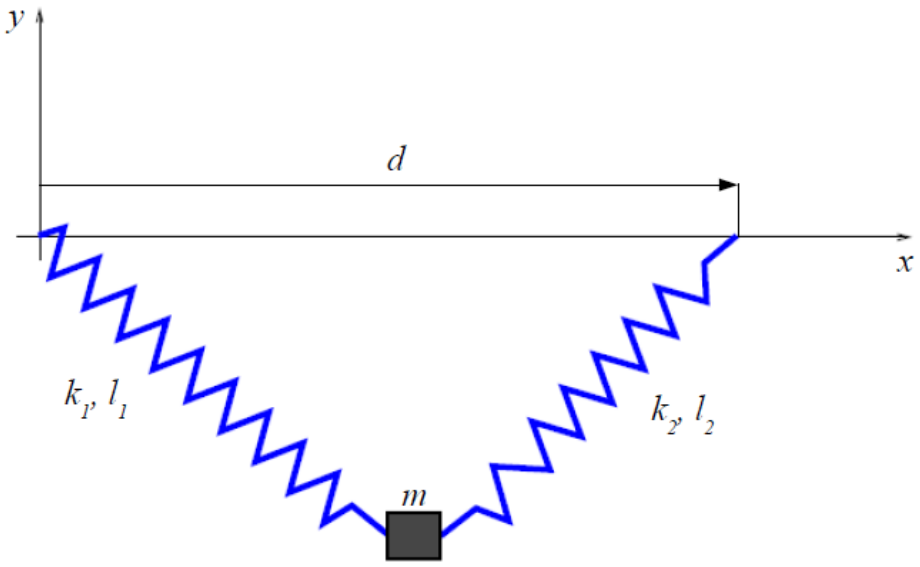
$\widehat{x} \leftarrow hyperdual(x)$
$n \leftarrow size(x, 1)$
$J \leftarrow zeros(1, n)$
for $j = 1$ to n **do**
    $\widehat{xj} \leftarrow \widehat{x}$
    $\delta xj_1(j) \leftarrow 1$
    $\widehat{y} \leftarrow f(\widehat{xj})$
    $J(1, j) \leftarrow \delta y_1$
end for

Rješenje:

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
x =

0.500000000007109
0.000000000776739
-0.523598775577995
```

5.2. Traženje minimuma potencijalne energije Newton-Raphsonovom metodom

Za tijelo mase m obješeno u vertikalnoj ravnini o dva elastična pera zanemarive mase s konstantama k_1, k_2 i duljinama l_1, l_2 i uz razmak objesišta jednak d, potrebno je izračunati ravnotežni položaj [1],[12].



Slika 5.1: Masa obješena na dvije opruge [1]

Problem rješavamo postavljanjem jednadžbe potencijalne energije mase m za položaj (x, y):

$$V(x,y) = mgy + \frac{k_1}{2}\left(l_1 - \sqrt{x^2+y^2}\right)^2 + \frac{k_2}{2}\left(l_2 - \sqrt{(d-x)^2+y^2}\right)^2, \tag{5.6}$$

Definicija 2 (Stabilna ravnoteža [10])**.** *Ako glatka funckija $V(x,y)$ poprima u točki (x^*, y^*) minimum onda vrijedi*

$$\frac{\partial V}{\partial(x,y)}(x^*,y^*) = \mathbf{0}, \quad \frac{\partial^2 V}{\partial(x,y)^2}(x^*,y^*) \geq \mathbf{0}. \tag{5.7}$$

Obratno, ako vrijedi

$$\frac{\partial V}{\partial(x,y)}(x^*,y^*) = \mathbf{0}, \quad \frac{\partial^2 V}{\partial(x,y)^2}(x^*,y^*) > \mathbf{0}, \tag{5.8}$$

onda funkcija V poprima u točki (x^*, y^*) strogi lokalni minimum.

Numeričko rješavanje jednadžbi ravnoteže $\frac{\partial V}{\partial(x,y)}(x^*, y^*) = \mathbf{0}$ ćemo ostvariti korištenjem Newton-Raphsonove metode koja generira niz točaka iteracijama

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \left[\nabla^2 V(\mathbf{q}_k)\right]^{-1} \nabla V(\mathbf{q}_k)^\intercal, \tag{5.9}$$

gdje je $\mathbf{q} = \begin{bmatrix} x & y \end{bmatrix}^\intercal$. Gradijent funckije V je

$$\nabla V(\mathbf{q}) = \begin{bmatrix} \frac{\partial V}{\partial x} & \frac{\partial V}{\partial xy} \end{bmatrix}, \tag{5.10}$$

a matrica Hessijan funkcije V po vektorskoj varijabli $\mathbf{q}$ je

$$\nabla^2 V(\mathbf{q}) = \begin{bmatrix} \frac{\partial^2 V}{\partial x^2} & \frac{\partial^2 V}{\partial x \partial y} \\ \frac{\partial^2 V}{\partial y \partial x} & \frac{\partial^2 V}{\partial y^2} \end{bmatrix}. \tag{5.11}$$

Traženje nultočke vektorske funkcije $\frac{\partial V}{\partial(x,y)}(x^*, y^*)$ ostvaruje se algoritmom 6, dok se računanje matrice Hessijan metodom hiperdualnih brojeva provodi sljedećim algoritmom:

Algoritam 8 Računanje Hessijana metodom HD brojeva

Ulaz: f, x

Izlaz: H

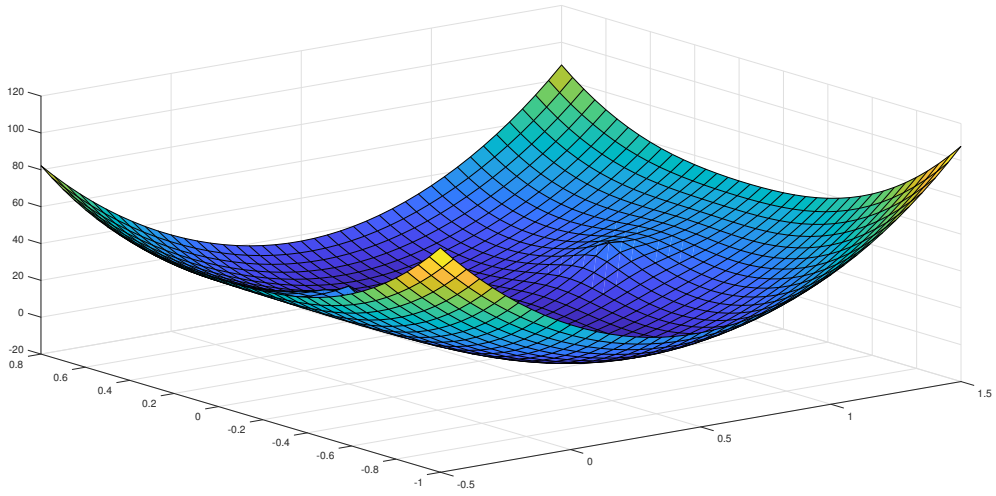
1: $\widehat{x} \leftarrow hyperdual(x)$
2: $n \leftarrow size(x, 1)$
3: $H \leftarrow zeros(n, n)$
4: **for** $j = 1$ to n **do**
5: **for** $k = 1$ to n **do**
6: $\widehat{xjk} \leftarrow \widehat{x}$
7: $\delta xjk_1(j) \leftarrow 1$
8: $\delta xjk_2(k) \leftarrow 1$
9: $\widehat{yjk} \leftarrow f(\widehat{xjk})$
10: $H(j,k) \leftarrow \delta\delta yjk$
11: $H(k,j) = H(j,k)$
12: **end for**
13: **end for**

Rješenje:

```
First-order
Iteration  Func-count       f(x)        Step-size       optimality
0           3          53.7415                          106
1           6          6.07662     0.00940796           48.9
2           9          2.64175              1           26.7
3          12          1.17883              1           3.59
4          18         -0.723377            10           6.13
5          24         -0.747516       0.10297           5.84
6          27         -0.844538              1           1.41
7          30         -0.850535              1         0.0284
8          33          -0.85054              1       0.000996
9          36          -0.85054              1        7.9e-07
Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the optimality tolerance.
x_optimalno =

```

```
0.499183554669675
-0.361134937393740


fval =

-0.850539941110294
```

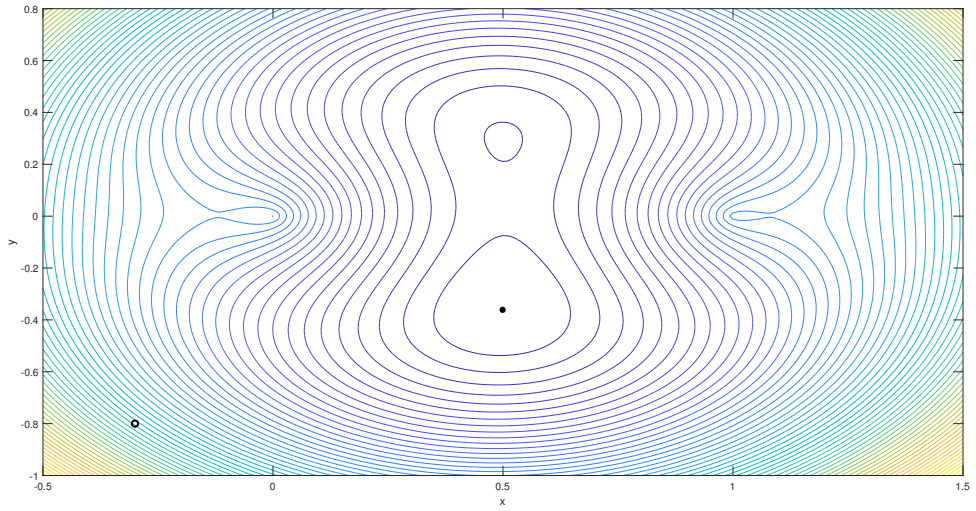
Na sljedećoj slici prikazana je ovisnost potencijalne energije mase na oprugama u ovisnosti o koordinatama xy:



Slika 5.2: Vrijednost potencijalne energije ovisno o xy koordinatama sustava

Tako se na slici 5.2 može iščitati da se vizualno rješenje zapravo podudara s numeričkim koje kaže da se minimum potencijalne energije sustava nalazi u točki $x = 0.499183554669675$ m, $y = -0.361134937393740$ m.

Također, na slici 5.3 s ucrtanim izolinijama vrijednosti potencijalne energije može se vidjeti točka s lokalnim ekstremima sustava kao i pozicija minimuma potencijalne energije.



Slika 5.3: Izolinije vrijednosti potencijalne energije sustava mase na oprugama, 5.1

6 Primjena hiperdualnih brojeva u optimizacijskim algoritmima u MATLAB-u

U ovom poglavlju aplikaciju klase HD brojeva nalazimo u računanju Jacobijana koristeći funkcije koje su ugrađene u MATLAB-ove optimizacijske programske alate.

Optimizacija sadrži skup aktivonosti čiji je cilj pronaći "najbolje rješenje", optimum, uz poštivanje zadanih ograničenja. Pojam "matematičko programiranje" često je korišten kao sinonim optimiranju, pa tako imamo i pojmove: linearno programiranje, kvaratno programiranje, nelinearno programiranje, semi-definitno programiranje itd, [10].

Formulacija problema optimizacije

Formulacija optimizacijskog problema zahtjeva definiranje:

- $\mathcal{X}$ projektnog skupa (skup projektnih varijabli, skup varijabli odluke, *engl. decision set*),
- $\mathcal{F} \subseteq \mathcal{X}$ dozvoljenog skupa (*engl. feasible set, constraint set*),
- $f : \mathcal{F} \to \mathbb{R}$ funkcije cilja (*engl. objective function, cost function*).

Cilj je odrediti vrijednost projektnih varijabli $x \in \mathcal{F}$ za koju je funkcija cilja $f(x)$ minimalna .

Optimizacijski problem :

$$\min_{\mathbf{x}} f(\mathbf{x}),$$

uz ograničenja

$$\mathbf{g}(\mathbf{x}) \leq \mathbf{0},$$
$$\mathbf{h}(\mathbf{x}) = \mathbf{0},$$

gdje su:
$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$ projektne varijable,
$f : \mathbb{R}^n \to \mathbb{R}$ funkcija cilja,
$\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^p$ ograničenja tipa nejednakosti,
$\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$ ograničenja tipa jednakosti.

Optimalno rješenje $\mathbf{x}^*$ je vektor projektnih varijabli koji, od svih vektora iz projektnog prostora koji zadovoljavaju ograničenja, ima najmanju vrijednost funckije cilja f, [10]. Vrijedi sljedeće svojstvo

$$\max_{\mathbf{x}} f(\mathbf{x}) \Longleftrightarrow \min_{\mathbf{x}} -f(\mathbf{x}). \tag{6.1}$$

6.1. Primjena hiperdualnih brojeva u optimizaciji kemijskog procesa

Rješavanje nelinearnih optimizacijskih problema bez ograničenja varijabli se u MATLAB-u ostvaruje korištenjem ugrađene funkcije `'fminunc'` [12], [11]. Ta se funkcija poziva kako bi se našla optimalna vrijednost parametara za koje je funkcija cilja minimalna uz to da iteracija kreće od inicijalne vrijednosti varijabli koje korisnik unaprijed zadaje. Funkcija `'fminunc'` kao ulazne parametre ima funkciju cilja, zatim početne vrijednosti varijabli po kojima se traži minimum funkcije cilja i kao treći argument mogu se unijeti željene postavke na algoritam traženja minimuma preko strukture `'options'`, a izlaz iz programa je optimalna vrijednost parametara:

```
1 function u = control(u0)
2
3 global tau N
4
5 options1=optimset('Display','iter','TolX',0.0001,'TolFun',0.0001);
```

```
u_temp=fminunc('cost_function_2',u0,options1);

%options=optimset('GradObj','on','Display','iter','MaxIter',1);
options=optimset('Algorithm','trust-region','Display','iter','TolX
    ',...
0.0001,'TolFun',0.0001,'MaxIter',2);
u = fminunc('cost_function',u_temp,options);
```

Ugrađeni solver za nelinearno programiranje `'fminunc'` koristi kvazi-Newtonovu metodu u kojoj se matrica Hessijan aproksimira Broyden-Fletcher-Goldfarb-Shanno metodom koja za evaluaciju koristi prvu derivaciju funkcije cilja po parametrima koja se računa metodom konačnih diferencija. Nadodajmo kako se unutar strukture `'options'` može opskrbiti minimizacijski algoritam s Jacobijanom radi ubrzavanja algoritma i tada je u strukturi `'options'` potrebno postaviti algoritam na `'trust-region'` pri čemu je Jacobijan drugi izlazni argument funckije cilja.

```
function [J,grad] = cost_function(x)

global N tau

xout = my_euler2(x, tau, N-1);

J = -xout(1,N-1) + xout(2,N-1) + xout(3,N-1);
grad =HD_Jacobian_Call(@derivacija,x);
```

Kod 6.1: Funkcija cilja u optimizacijskom problemu

U kodu 6.1 `'my_euler2'` je m-funckija za numeričko rješavanje sustava diferencijalnih jednadžbi s početnim uvjetima i zadanom upravljačkom varijablom, a `'derivacija'` je m-funkcija identična `'cost_function'` ali bez Jacobijana kao drugog izlaza. Zbog načina rada klase za računanje Jacobijana gdje se funckija cilja poziva ponovno za svaki element u upravljačkoj varijabli, te zbog većeg broja operacija unutar računala za svaku matematičku operaciju nad hiperdualnim brojevima nego u slučaju rada s ugrađenim objektima za brojeve, sada je potrebno podosta vremena za izvršavanje programa, ali se osigurava točnost do strojne preciznosti. Iz koda 6.1 može se vidjeti da je funckija cilja ovisna o količinama reaktanata u terminalnom vremenu, a želi se maksimizirati koncentracija prve tvari dok se istovremeno želi minimizirati koncentracija ostale dvije. Kao drugi izlazni argument funckije izračunali smo varijablu

'grad' koja sadrži numeričke vrijednosti derivacije funkcije cilja po ulaznoj varijabli, to jest po upravljačkoj varijabli koja je u ovoj funkciji nazvana 'x'. Derivacije funkcije po svakom elementu unutar upravljačke varijable računaju se prema algoritmu 7, a točan poziv funkcije unutar MATLAB-a je prikazan u kodu ispod:

```
function J = derivacija(x)

  global N tau

  xout = my_euler(x, tau, N-1);

  J = -xout(1) + xout(2) + xout(3);
```

Kod 6.2: Računanje Jacobijana HD klasom

Diferencijalne jednadžbe koje se rješavaju unutar funkcije 'my_euler2' po prirodi su nelinearne obične diferencijalne jednadžbe prvog reda i koristi se Eulerova metoda za rješavanje. Kod ispod prikazuje diferencijalne jednadžbe koje opisuju sustav:

```
  dx(1) = k1r*cont - k3r*x1;
  dx(2) = 2*k2r*cont^2;
  dx(3)= k3r*x1;
```

Kod 6.3: Jednadžbe diferencija

Zbog vremena izvršavanja dosjetljivo je prvo izračunati optimalnu upravljačku varijablu koristeći ugrađeni rješavač i to bez gradijenta kao drugog izlaza funkcije cilja i zatim koristiti tu vrijednost kao početnu u iteraciji sa opskrbljivanjem gradijenta koji se računa metodom hiperdualnih brojeva. Rezultati koji se dobiju koristeći osnovni rješavač su prikazani ispod:

```
                                                        First-order
Iteration  Func-count       f(x)        Step-size       optimality
0        1001                0                          0.0138
1        3003        -0.461559              10            0.01
2        4004        -0.967133               1        4.99e-07

Local minimum found.

Optimization completed because the size of the gradient is less than
the selected value of the optimality tolerance.

```

```
<stopping criteria details>
```

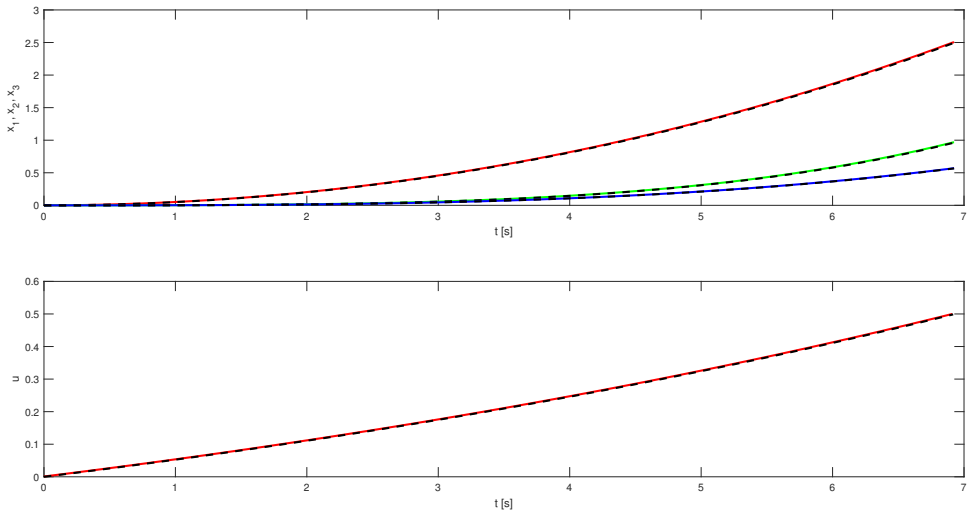
Kod 6.4: Optimizacija s ugrađenim algoritmom

Međutim koristeći klasu hiperdualnih brojeva za računanje gradijenta funkcije cilja uz istu početnu točku iteracije nakon prva dva koraka dobivaju se ipak bolji rezultati s obzirom na kriterij `'First order optimality'`, a rezultati su prikazani u kodu ispod:

```
                                                    First-order
Iteration  Func-count    f(x)          Step-size      optimality
0            1           0                            0.0138
1            3           -0.461559     10             0.01
2            4           -0.967133     1              7.81e-18
Local minimum found.

Optimization completed because the size of the gradient is less than
the selected value of the optimality tolerance.

<stopping criteria details>
```

Kod 6.5: Rezultati optimizacije s računanjem gradijenta preko HD klase

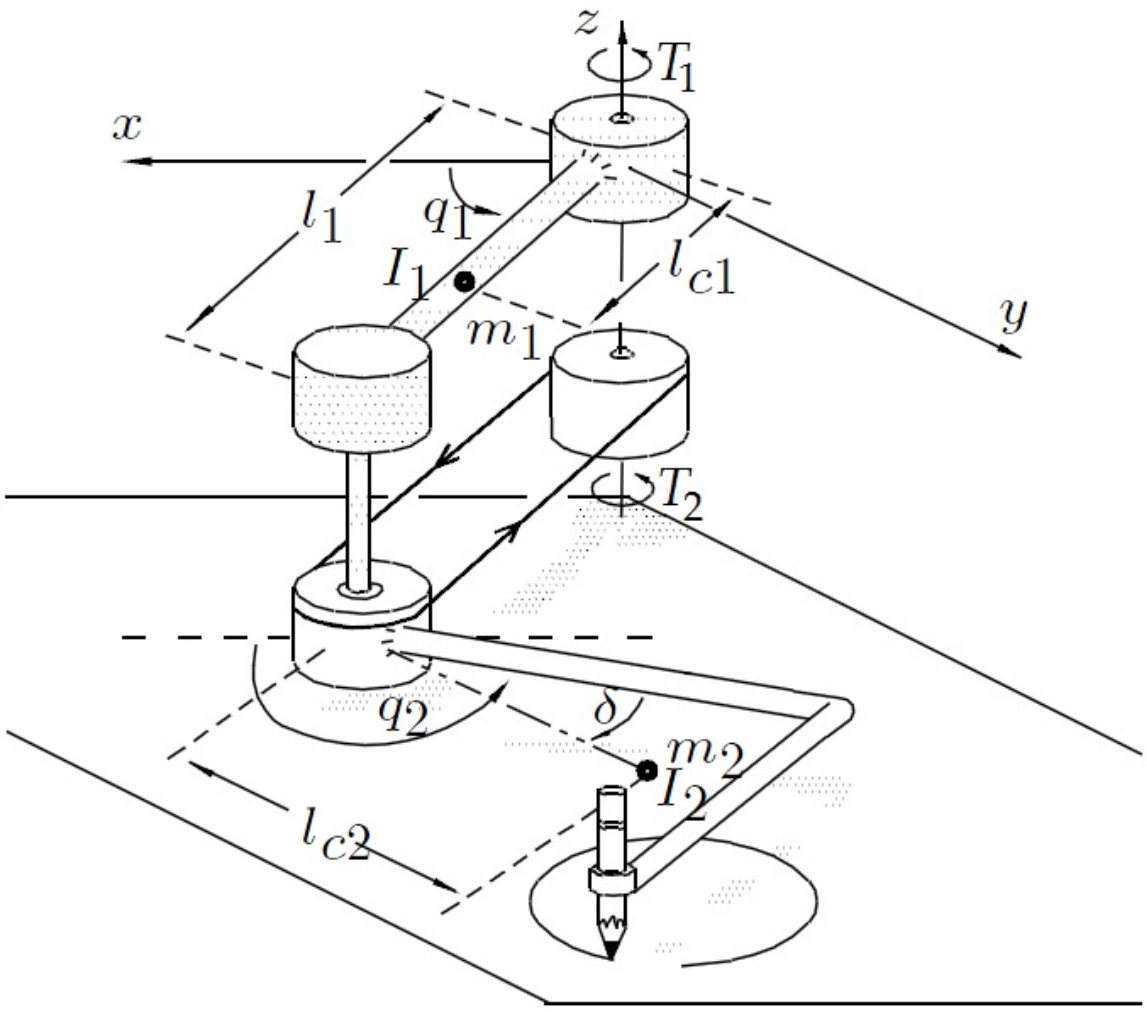
Provođenjem optimizacije uz računanje Jacobijana klasom HD brojeva, i usporedbom analitičkog rješenja dobiveni su rezultati prikazani na slici 6.1:



Slika 6.1: Usporedba analitičkog s rezultatima optimizacije u MATLAB-u

6.2. Primjena hiperdualnih brojeva u optimalnom upravljanju 2 DOF robota

Primjena hiperdualnih brojeva u optimalnom upravljanju robota s dva stupnja slobode gibanja se ovdje također odnosi na računanje derivacija funkcije cilja po članovima iz upravljačke varijable. Iz [2] uzimamo primjer robota prikazanog na slici 6.2. Gibanje je ograničeno na horizontalnu plohu definiranu osima $x - y$, gdje je l_1



Slika 6.2: 2 DOF SCARA robot, [2]

duljina prvog linka robota, a m_1 i m_2 označuju mase prvog odnosno drugog linka. Na slici su također naznačeni i položaji centara mase oba linka, dok se pozicija centra

mase drugog linka nalazi fizički izvan njega zbog geometrije te ovisi o konstantnom kutu δ. Nadalje, udaljenosti od rotirajućih osi do centara mase pojedinog linka označene su varijablama l_{c1} i l_{c2} za prvi, odnosno za drugi link. Momenti inercije s obzirom na osi paralelne s osi-z i koje prolaze kroz centre masa su označene kao I_1, I_2 za prvi idrugi link, respektivno. Zakreti kuta linkova označeni su kao q_1 i q_2 te se mjere između svakog linka zasebno i osi paralelne s osi-x te se uzimaju u smjeru obrnutom od zakreta kazaljke na satu. Gibanje linka dva omogućeno je preko remenice koja je spojena s rotirajućim aktuatorom lociranim ispod baze samog robota.

Matematički model koji opisuje dinamiku robota izvodi se postavljajući Euler-Lagrange jednadžbu kao u [2], a konačni oblici diferencijalnih jednadžbi su

$$\begin{aligned} \theta_1\ddot{q_1} + (\theta_3 C_{21} + \theta_4 S_{21})\ddot{q_2} - \theta_3 S_{21}\dot{q_2}^2 + \theta_4 C_{21}\dot{q_2}^2 = T_1, \\ (\theta_3 C_{21} + \theta_4 S_{21})\ddot{q_2} + \theta_2\ddot{q_2} + \theta_3 S_{21}\dot{q_1}^2 - \theta_4 C_{21}\dot{q_1}^2 = T_2, \end{aligned} \tag{6.2}$$

Gdje je $C_{21} = \cos(q_2 - q_1)$, $S_{21} = \sin q_2 - q_1$, $\theta_1 = m_1 l_{c1}^2 + m_2 l_1^2 + I_1$, $\theta_2 = m_2 l_{c2}^2 + I_2$, $\theta_3 = m_2 l_1 l_{c2}\cos(\delta)$ te $\theta_4 = m_2 l_1 l_{c2}\sin(\delta)$. Numeričke vrijednosti navedenih parametara su: $l_1 = 1$ m, $l_{c1} = l_{c2} = 0.5$ m, $m_1 = 3$ kg, $m_2 = 4$ kg, $I_1 = 1$ kgm^2, $I_2 = 2$ kgm^2, $\delta = 0$.

Povoljniji oblik pisanja jednadžbi (6.2) je u kompaktnoj, matričnoj formi

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{u}, \tag{6.3}$$

gdje je $\mathbf{q} = [q_1 q_2]^\intercal$ vektor kuteva zakreta linkova, a $\mathbf{u} = [T_1 T_2]^\intercal$ je vektor upravljačkih varijabli. Uz to, imamo

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} \theta_1 & \theta_3 C_{21} + \theta_3 S_{21} \\ * & \theta_2 \end{bmatrix}, \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} 0 & (\theta_4 C_{21} - \theta_3 S_{21})\dot{q_2} \\ (\theta_3 S_{21} - \theta_4 C_{21})\dot{q_1} & 0 \end{bmatrix}. \tag{6.4}$$

Konačno, uvođenjem vektora stanja $x = [\mathbf{q}^\intercal \dot{\mathbf{q}}^\intercal]^\intercal$, prostor stanja se može napisati kao

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{M}(\mathbf{q})^{-1}(\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}) \end{bmatrix}. \tag{6.5}$$

U nastavku se razmatraju se dva problema upravljanja.

6.2.1. Transformacija robota iz inicijalnog u željeno stanje

Iz početnog stanja robota

$$x_1(0) = \frac{\pi}{4} \text{ rad}, x_2(0) = \frac{\pi}{4} \text{ rad}, x_3(0) = 0 \text{ rad/s}, x_4(0) = 0 \text{ rad/s}, \tag{6.6}$$

u konačno stanje

$$x_{1f} = \frac{\pi}{2} \text{ rad}, x_{2f} = \frac{\pi}{2} \text{ rad}, x_{3f} = 0 \text{ rad/s}, x_{4f} = 0 \text{ rad/s}. \tag{6.7}$$

Uz ograničenja nad upravljačkom varijablom u obliku nejednakosti

$$-u_{max} \leq u_1 \leq u_{max}, -u_{max} \leq u_2 \leq u_{max}, u_{max} = 10 \text{ Nm}. \tag{6.8}$$

Funkcija cilja zadana je kao

$$J = \sum_{k=1}^{q} K_{b,k} b_k^2(\mathbf{x}(t_f)) \tag{6.9}$$

gdje

$$b_k = x_{kf} - x_k(t_f), \ k = 1, 2, 3, 4. \tag{6.10}$$

Koeficijenti unutar funkcije cilja $K_{b,k}$ će osigurati ograničenja na upravljačku varijablu za više vrijednosti.

Način rješavanja ovog problema koristeći klasu hiperdualnih brojeva za traženje rješenja boljeg od onog dobivenog ugrađenim rješavačem s postavljenim opcijama na *default* je sličan poput onog u problemu optimiranja kemijskog procesa. Prvo se traži vrijednost upravljačke varijable po ugrađenom algoritmu koristeći također ugrađeni `'sqp'` algoritam koji je poseban po tome što u svakoj iteraciji se zadovoljavaju uvjeti postavljeni na ograničavanje upravljačke varijable. Nadalje, upravljačka varijabla dobivena ovim postupkom sada je početna točka u algoritmu koji koristi klasu hiperdualnih brojeva za računanje Jacobijana u svakom koraku iteracije. Točan način pozivanja ovih dviju rješavača prikazan je u kodu ispod:

```
function uizl = control(u)

global N

umin = -10*ones(2, N);
umax = 10*ones(2, N);
options1 = optimset('Display', 'iter', 'Algorithm', 'sqp');
```

```
uizl_temp = fmincon('cost_function_base', u, [], [], [], [], umin,
    umax, [], options1);
options = optimset('GradObj','on','Display', 'iter', 'Algorithm', 'sqp
    ','TolFun',5e-7,'MaxIter',3);
uizl = fmincon('cost_function', uizl_temp, [], [], [], [], umin, umax,
     [], options);
```

Kod 6.6: Funkcija za dobivanje optimalne upravljačke varijable

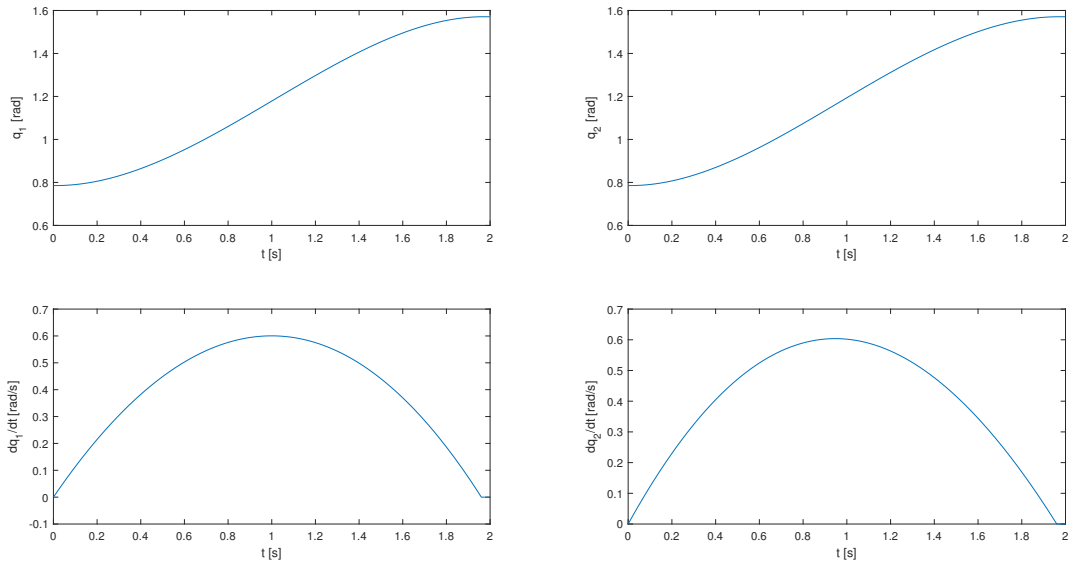
Funkcija `'cost_function_base'` u kodu 6.6 zapravo je m-datoteka u kojoj se rješava diferencijalna jednadžba sustava zadanog jednadžbama (6.2) uz početne uvjete. Zatim se ta dobivena vrijednost koristi kao početna točka u iteraciji gdje se Jacobijan funckije cilja računa koristeći HD brojeve. U kodu ispod prikazani su rezultati iteracija provedeni u MATLAB-u:

```
Iter  Func-count  Fval            Norm of      First-order
                                  step         optimality
33      6834       2.391658e-09    1.548e-02    5.289e-07
Optimization completed because the objective function is non-
    decreasing in feasible directions.
0           1    2.391658e-09     0.000e+00    5.226e-07
1           3    2.380166e-09     3.408e-06    5.176e-07
fmincon stopped because the size of the current step is less than
the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.
```

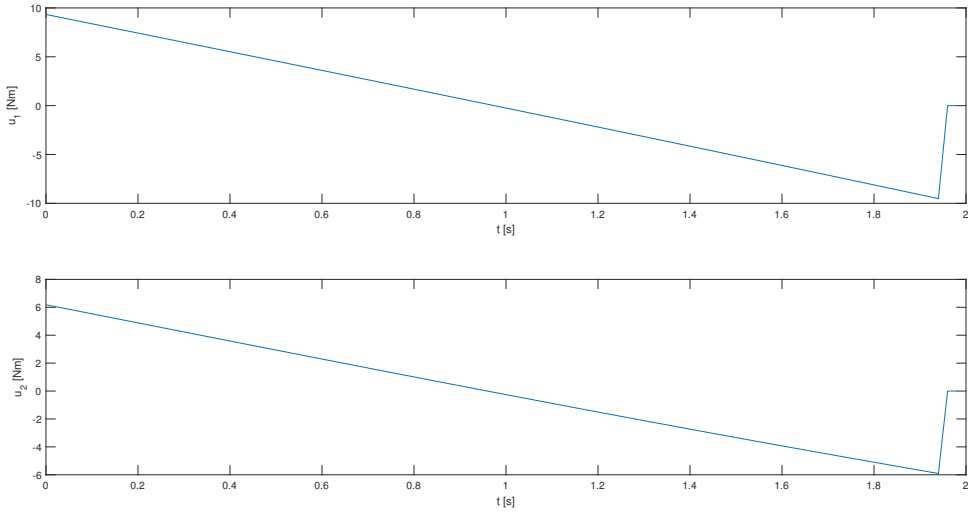
Kod 6.7: Rezultati iteracija u MATLAB-u

Poruka programa prikazana u kodu 6.7 govori da je mjera za veličinu koraka korištenog u traženju iduće točke u iteraciji dosegla veličinu manju od zadane (*default*) vrijednosti algoritma. Prema tome dobiveno rješenje je zadovoljavajuće za postavljene tolerancije korisnika. Rješenje koristeći klasu HDbrojeva za računanje Jacobijana prema tome daje bolje rješenje od iteracije s zadanim postavkama jer je i vrijednost funckije cilja minimalnija. Rezultati rješavanja diferencijalne jednadže koristeći upravljačku varijablu dobivenu preko `'fmincon'` funkcije su prikazani na slikama koje slijede.

Na slici 6.3 vidi se da varijable stanja dosežu traženu vrijednost te da se brzine vraćaju na vrijednost nula, prema tome su rezultati zadovoljavajući. Ostaje prikazati još vrijednosti upravljačke varijable:



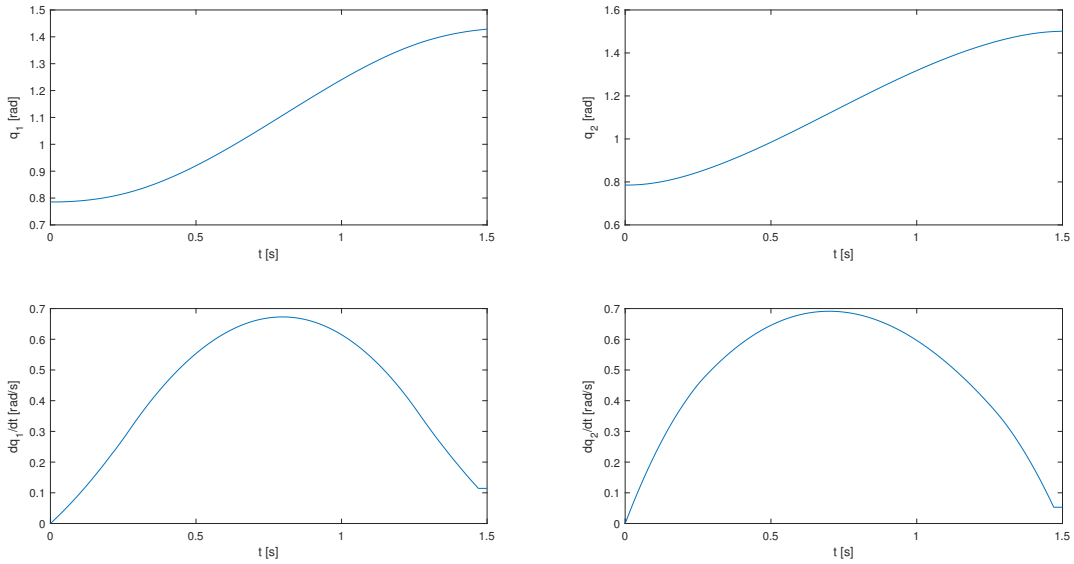
Slika 6.3: Varijable stanja SCARA robota za slučaj 1



Slika 6.4: Varijable stanja SCARA robota za slučaj 1

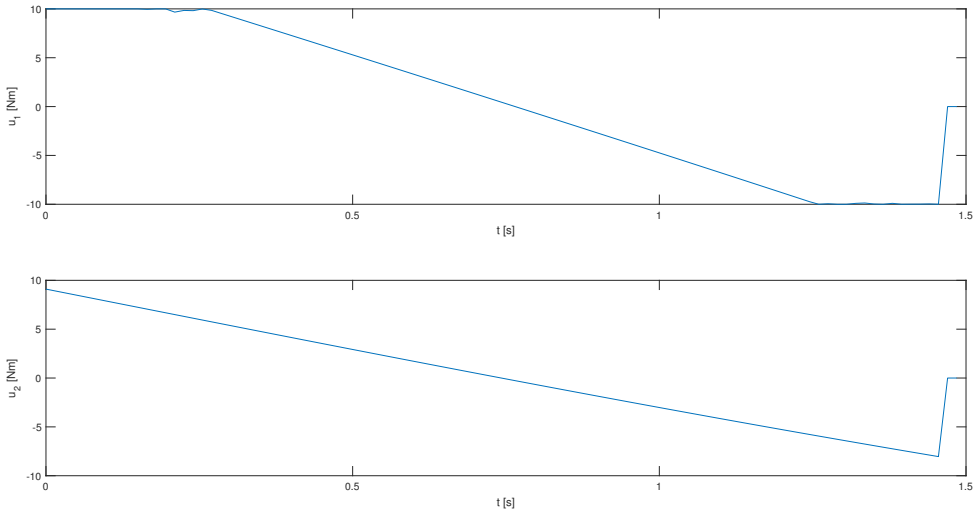
Na slici 6.4 vidi se da i upravljačke varijable ostaju unutar zadanih ograničenja, a mogla bi se uvesti u saturaciju tako što bi smanjili vrijeme izvršavanja procesa na ispod 2 sekunde.

Za vrijeme izvršavanja transformacije robota iz početnog u isto željeno stanje definirano jednadžbama (6.7) postavljeno na $t_f = 1.5$ s uz istu funckiju cilja i ograničenja na upravljačku varijablu imamo rezultate prikazane na slijedećim slikama.



Slika 6.5: Varijable stanja SCARA robota za slučaj 1 za vrijeme $t_f = 1.5$ s

Na slici 6.5 iz grafova pozicija q_1 i q_2 vidljivo je da je zahtjev za vrijeme izvršavanja od 1.5 s ipak prezahtjevno jer robot ne stigne izvršiti transofmaciju, također ni brzine nisu postigle zahtijevanu vrijednost od nula. Pretpostavka je da su upravljačke varijable ušle u saturaciju jer inače bi ostalo prostora za bolje performanse.



Slika 6.6: Upravljačke varijable SCARA robota za slučaj 1 za vrijeme $t_f = 1.5$ s

Na slici 6.6 vidljivo je da upravljačka varijable u_1 ulazi u saturaciju za razliku od u_2, a to je zato što je na link 1spojen i link 2 pa podnosi veće opterećenje. Rezultati optimizacije pokazuju da iteracije u kojima se Jacobijan računa metodom HD brojeva brže konvergiraju od ugrađenih postavki optimizacijskog algoritma jer i norma koraka pada brže te je prikazano u kodu ispod:

```
Iter   Func-count   Fval            Norm of      First-order
                                      step         optimality
99     20100        4.213728e-02    1.638e-01    5.368e-04
fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunctionEvaluations = 20000 (the default value).
0      1            4.213728e-02    0.000e+00    5.369e-04
1      3            4.213097e-02    2.571e-03    5.366e-04
2      5            4.210026e-02    1.238e-02    5.354e-04
3      7            4.200551e-02    3.925e-02    5.316e-04
fmincon stopped because it exceeded the iteration limit,
```

```
11 options.MaxIterations = 3 (the selected value).
```

Kod 6.8: Rezultati iteracija u MATLAB-u

6.2.2. Transformacija robota iz inicijalnog u željeno stanje uz praćenje referentne trajektorije

U ovom slučaju početno stanje robota je

$$x_1(0) = 0 \text{ rad}, x_2(0) = \frac{\pi}{4} \text{ rad}, x_3(0) = 0 \text{ rad/s}, x_4(0) = 0 \text{ rad/s}, \tag{6.11}$$

dok su ograničenja upravljačke varijable

$$-u_{max} \leq u_1 \leq u_{max}, -u_{max} \leq u_2 \leq u_{max}, u_{max} = 10 \text{ Nm}. \tag{6.12}$$

Funkcija cilja zadana je u obliku

$$J = \chi(t_f), \tag{6.13}$$

gdje je

$$\begin{aligned}
\dot{\chi} &= (X_r - X_d)^2 + (Y_r - Y_d)^2, \\
X_r &= l_1 \cos(x_1) + l_2 \cos(x_1 + x_2), \\
Y_r &= l_1 \sin(x_1) + l_2 \sin(x_1 + x_2), \\
X_d &= 2\cos(\omega t), \\
Y_d &= 2\sin(\omega t),\ \omega = \frac{\pi}{2t_f}.
\end{aligned}$$

Koordinate X_r i Y_r označuju poziciju vrha drugog linka s obzirom na koordinatni sustav vezan uz postolje robota definiran kao Kartezijski koordinatni sustav s dvije dimenzije čije je ishodište u prvom zglobu prvog linka. Varijable X_d i Y_d su zadane koordinate vrha drugog linka.

U ovom slučaju opet se traži početna točka iteracije algoritma u kojem se horisti klasa HD preko iteracije s ugrađenim algoritmom i postavkama na *default*.

```
1 function uizl = control(u)
2
3 global N
4
```

```
umin = -10*ones(2, N);
umax = 10*ones(2, N);
options1 = optimset('Display', 'iter', 'Algorithm', 'sqp');
uizl_temp = fmincon('cost_function_base', u, [], [], [], [], umin,
    umax, [], options1);
options = optimset('GradObj','on','Display', 'iter', 'Algorithm', 'sqp
    ','TolFun',5e-7,'MaxIter',4);
uizl = fmincon('cost_function', uizl_temp, [], [], [], [], umin, umax,
     [], options);
```

Kod 6.9: Traženje optimalne upravljačke varijable sa slučaj 2

U kodu 6.9 funckija `'cost_function_base'` je zasebna m-funkcija u kojoj se računa funckija cilja:

```
function J = cost_function_base(u)

  global N tau
  xout = my_euler_old(u, tau, N);
  J= xout(5,N-1);
```

Kod 6.10: Funckija cilja u slučaju 2

gdje je `'my_euler_old'` zasebna m-funckija u kojoj se rješava diferencijalna jednadžba sustava za početne uvjete:

```
function x = my_euler_stari(u, h, n_i)

  % Euler method for ODEs solution

  global x0 n

  x(1:n, 1) = x0;

  for j = 1:n_i
  f = ODE_scara(x, u, j);
  x(:, j+1) = x(:, j) + h*f;
end
```

Kod 6.11: Rješavanje diferencijalne jednadžbe sustava

Nadalje, u kodu 6.9 funckija `'cost_function'` je m-funkcija u kojoj je uz funckiju cilja drugi izlazni argument Jacobijan koji se računa koristeći klasu HD brojeva.

```
function [J,grad] = cost_function(u)

  global N tau

  xout = my_euler_stari(u, tau, N);
  J= xout(5,N-1);
  grad =HD_Jacobian_Call(@derivacija,u);
```

Kod 6.12: Funckija cilja s Jacobijanom

Ovdje je potrebno naglasiti da u funckiji `'derivacija'` rješenje sustava diferencijalnih jednadžbi računa drugačije od onog u prikazanog u funkciji 6.11, jer je potrebno prilagoditi rezultate međuoperacija klasi HD brojeva koja u ovoj implementaciji ne može raditi sa vektorskim produktima matematičkih funckija ili operatora.

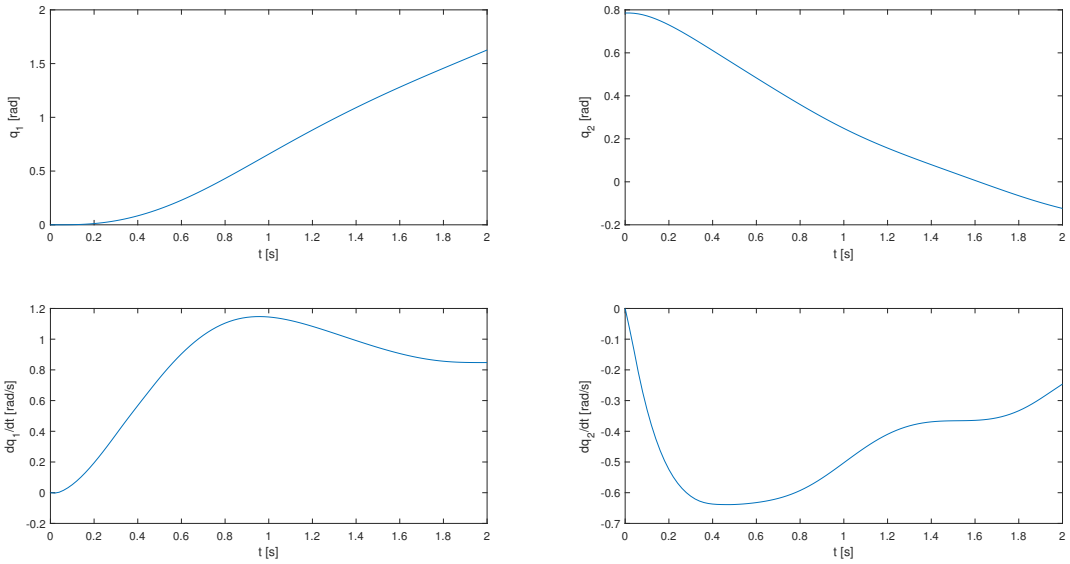
Rješenje koje dobivamo u naredbenom prozoru MATLAB-a izvođenjem koda prikazanog u 6.9 je sljedeće:

```
Iter   Func-count   Fval            Norm of      First-order
                                    step         optimality
99     20100        9.405245e-02    3.089e-01    1.809e-04
fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunctionEvaluations = 20000 (the default value).
0      1            9.405245e-02    0.000e+00    1.736e-04
1      3            9.405232e-02    4.112e-04    1.083e-04
2      5            9.405161e-02    2.009e-03    1.086e-04
3      7            9.404820e-02    9.988e-03    1.104e-04
4      9            9.403316e-02    4.854e-02    1.192e-04
fmincon stopped because it exceeded the iteration limit,
options.MaxIterations = 3 (the selected value).
```

Kod 6.13: Naredbeni prozor u MATLA-u za slučaj 2

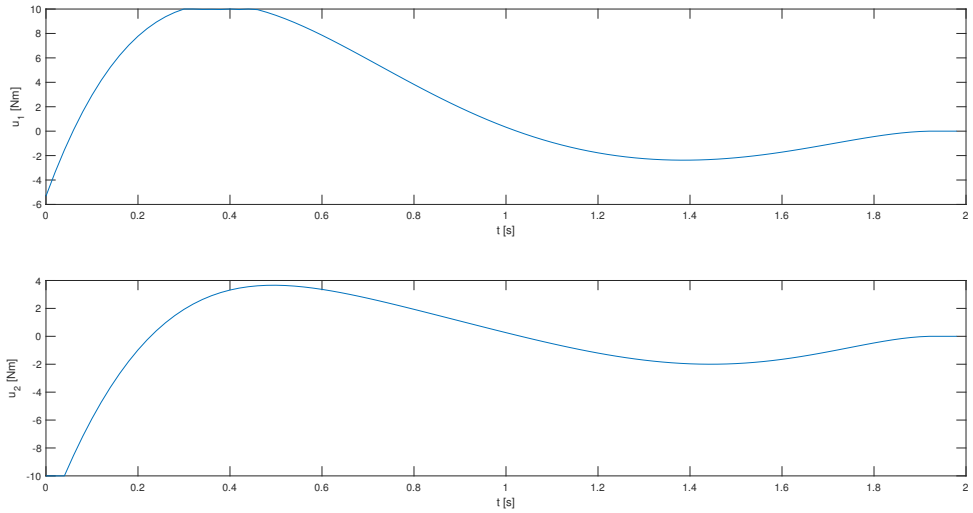
Opet je vidljivo kako iteracije u kojima se koristi HD kod za računanje matrice Jacobijan brže konvergiraju optimalnom rješenju te da norma koraka postiže manje vrijednosti za nekoliko redova veličine.

Varijable stanja sustava nakon izvođenja simulacija sa optimalnom vrijednošću upravljačke varijable su prikazani na slici 6.7.



Slika 6.7: Varijable stanja SCARA robota za slučaj 2 za vrijeme $t_f = 2$ s.

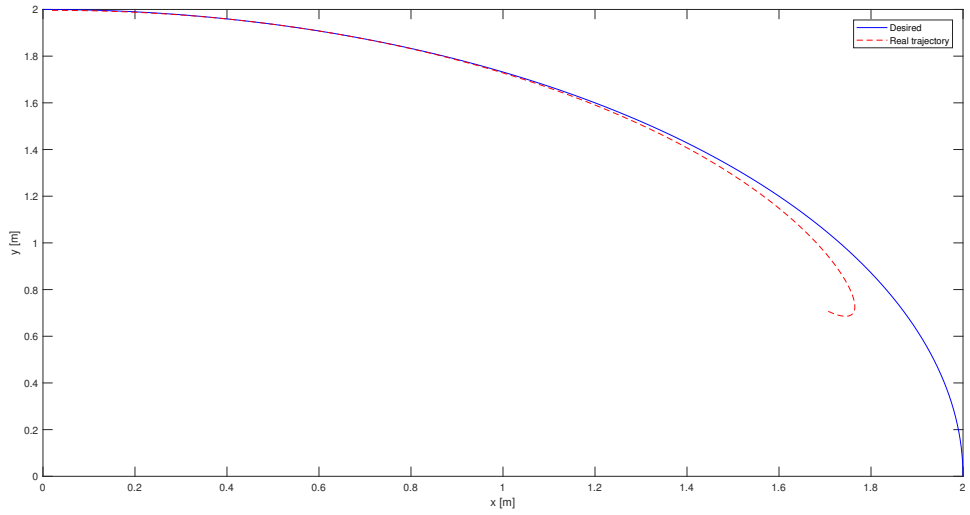
Nadalje, upravljačke varijable su prikazane na slijedećim grafovima.



Slika 6.8: Upravljačke varijable SCARA robota za slučaj 2.

Na slici 6.8 vidljivo je kako je samo prva upravljačka varijabla ušla u područje saturacije zbog toga što dijelom pogoni i drugi robotski link.

Praćenje referentne trajektorije prikazano je na slici 6.9.



Slika 6.9: Praćenje referentne trajektorije za slučaj 2.

Greška praćenja na slici 6.9 se povećava kako vrijeme simulacije se odmiče.

7 Zaključak

U ovom radu predstavljena je metoda računanja prve i druge derivacije funkcije više varijabli bazirane na aritmetici hiperdualnih brojeva s primjenama u MATLAB-u. Krenulo se od uvođenja najprije jednostavnog računalnog programa koji automatski računa derivaciju funckije što je poslužilo kao motivacija za uvođenje univerzalnog algoritma čija je zadaća računanje derivacija složenih viševarijabilnih funkcija. Objektno programiranje u MATLAB-u dozvoljava da se algoritam baziran na HD brojevima realizira kreiranjem klase objekata i definiranjem metoda nad atributima sadržanim u objekatima i time je osigurano da operacije nad objektima klase strogo slijede osnovne jednadžbe algoritma. Koristeći niz jednostavnijih algoritama približili smo čitatelju mehaniku rada klase i pojasnili bitne korake u postupku računanja vrijednosti derivacija te upozorili na ograničenja i pojasnili moguće nejasnoće u radu klase. Jedno poglavlje rada u potpunosti je posvećeno klasifikaciji svih tehnika objektog programiranja potrebnih za izradu klase u MATLAB-u, a detaljnim obrazloženjima uz primjere samog koda i čitatelji koji nemaju iskustva s programiranjem imaju mogućnost upoznati se sa osnovnim principima i tehnikama objektnog programiranja. Za potvrdu točnosti algoritma u trećem poglavlju rada provedena je usporedba s metodom centralne razlike i koračno kompleksnom metodom na primjerima računanja prve i druge derivacije viševarijabilne funckije čime je i numeričkim eksperimentom potvrđena točnost definicije 1.

U ostalim poglavljima rada provedeni su numerički eksperimenti na primjerima u kojima točnost vrijednosti derivacija dolazi do izražaja. Prvo, kod Newtonove metode traženja nultočke poznato je da kovergencija ovisi o vrijednostima derivacija te da je

uz dovoljnu točnost onda potrebno izvršiti manje iteracija. Na kraju, u dva optimizacijska problema koristili smo klasu HD brojeva u MATLAB-ovim ugrađenim solverima `fmincon` i `fminunc` za računanje Jacobijana funkcije cilja te su rezultati prikazali bolju točnost u odnosu na ugrađene metode računanja gradijenta.

A Prvi prilog

A.1. Kodovi za računanje derivacija iz poglavlja 4

Kod za računanje Jacobijana prema algoritmu 2:

```
function dfdx = Jacobian_finite_diff(x)


global n m tt

%dfdx=zeros(3,1);
for i=1:3
  ss=zeros(3,1);
  ss(i)=1;
    f1=eq_sys(x-tt*ss, 2);
    f2=eq_sys(x+tt*ss,2);
    dfdx(i)=(f1-f2)/(2*tt);
    dfdx(i)=(f2-f1)/(2*tt);
end
```

Kod za računanje Jacobijana koristeći kompleksnu varijablu prema algoritmu 3:

```
function dfdx = Complex_step_jacobian(x)
global n m tt

x1 = x;
for j=1:3

    x1(j)=complex(x(j),tt);
    f1=(x1(1)*x1(2)*sin(x1(3)) + exp(x1(1)*x1(2)))/x1(3);
```

```
    dfdx(j)=imag(f1)/tt;
    x1 = x;
end
```

Kod za računanje Jacobijana koristeći metodu centralne razlike prema algoritmu 4

```
function hfd_x = hessian_fd(x)

% Hessian calculation using finite-difference method

global n m tt

hfd_x = zeros(n,m);

for k = 1:m
    f = eq_sys(x,2);
    for l = 1:n
        ss = zeros(3,1);
        ss(l) = 1;
        ff1 = eq_sys(x-2*tt*ss, 2);
        ff2 = eq_sys(x-tt*ss, 2);
        ff3 = eq_sys(x+tt*ss, 2);
        ff4 = eq_sys(x+2*tt*ss, 2);
        hfd_x(l,l,k) = 1/(12*tt^2)*(-ff1(k) + 16*ff2(k) - 30*f(k) +
   16*ff3(k) - ff4(k));
    end
    lam=1;
    l=n-1;
    while l>0
        for p = 1:l
            img_vec=zeros(n,1);
            img_vec(p:p+lam,1)=1;
            ff1 = eq_sys(x-2*tt*img_vec, 2);
            ff2 = eq_sys(x-tt*img_vec, 2);
            ff3 = eq_sys(x+tt*img_vec, 2);
            ff4 = eq_sys(x+2*tt*img_vec, 2);
            hfd_x(p,p+lam,k)=(1/(12*tt^2)*(-ff1(k) + 16*ff2(k) - 30*f(
   k) + 16*ff3(k) - ff4(k)) - sum(sum(hfd_x(p:p+lam,p:p+lam,k))))/2;
            hfd_x(p+lam,p,k)=hfd_x(p,p+lam,k);
        end
        l=l-1;
```

```
        lam=lam+1;
    end
end
```

Kod za računanje Jacobijana koristeći koračno kompleksnu metodu prema algoritmu 5:

```
function Hc_x = hessian_complex(x)

% Hessian calculation using complex variable method

global n m tt

Hc_x = zeros(n,m);

for k = 1:m
    for l = 1:n
        ss = zeros(3,1);
        ss(l) = 1;
        xx1 = x + (tt*1i^(1/2))*ss;   %complex(x,tt*ss);
        xx2 = x + (tt*1i^(5/2))*ss;
        ff1 = eq_sys(xx1,2);
        ff2 = eq_sys(xx2,2);
        Hc_x(l,l,k) = 1/tt^2*imag(ff1(k) + ff2(k));
    end
    lam=1;
    l=n-1;
    while l>0
        for p = 1:l
            img_vec=zeros(n,1);
            img_vec(p:p+lam,1)=1;
            ff1 = eq_sys(x + (tt*1i^(1/2))*img_vec,2);
            ff2 = eq_sys(x + (tt*1i^(5/2))*img_vec,2);
            Hc_x(p,p+lam,k)=(1/tt^2*imag(ff1(k) + ff2(k)) - sum(sum(
   Hc_x(p:p+lam,p:p+lam,k))))/2;
            Hc_x(p+lam,p,k)=Hc_x(p,p+lam,k);
        end
        l=l-1;
        lam=lam+1;
    end
end
```

Glavni kod za poziv funkcija i prikaz rezultata:

```
close all
clear all
clc

global n m hh tt

n = 3;
m = 1;

syms x1 x2 x3
x = [x1;x2;x3];

Jacobijan_sym=jacobian(eq_sys(x,1),x);
Jacobijan_val=double(subs(Jacobijan_sym,{x1,x2,x3},{1,1.5,0.5}));

%Ja_x = jacobian(jacobian_analytic(x),x);
%Ja_x = double(subs(Ja_x, {x1,x2,x3}, {1.0,1.5,0.5}));


x_hd = [1.0;1.5;0.5];
Jacobian_HD=HD_Jacobian_Call(@eq_sys,x_hd,2);
%Hessian_HD=HD_Hessian_Call(@eq_sys,x_hd,2);
%error calculation by method of HD numbers
%err_hd = (norm(Hessian_HD - Ja_x)/norm(Ja_x));
err_hd=(norm(Jacobian_HD - Jacobijan_val)/norm(Jacobijan_val));


for k = 1:1:50

    hh = 10^(-k);
    tt = hh;


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Hessian calculation using finite-difference method
    %x_fd = [1.0;1.5;0.5];
    %Jfd_x = hessian_fd(x_fd);
```

```
39
40 % Hessian calculation using complex variable method
41     %Jc_x = hessian_complex(x_c);
42
43     %Jacobian calculation using complex step
44     x_c = [1.0;1.5;0.5];
45     Jacobijan_complex=Complex_step_jacobian(x_c);
46
47     Jacobian_fd=Jacobian_finite_diff(x_c);
48
49     %x_f_c = [1.0;1.5;0.5];
50     %Jc_x_forward = moja_forward_fd(x_f_c);
51
52
53     Kk(k) = k;
54
55
56     eror1(k)=(norm(Jacobijan_complex-Jacobijan_val)/norm(Jacobijan_val
       ));
57     eror2(k)=(norm(Jacobian_fd-Jacobijan_val)/norm(Jacobijan_val));
58     eror_hd(k)=err_hd;
59     %err1(k) = (norm(Jfd_x - Ja_x)/norm(Ja_x));
60     %err2(k) = (norm(Jc_x - Ja_x)/norm(Ja_x));
61     %err3(k) = (norm(Jc_x_forward - Ja_x)/norm(Ja_x));
62     %err_hd_vec(k)=err_hd;
63
64 end
65 figure;
66 semilogy(Kk,eror_hd,'b','Linewidth',2);
67 hold on
68 semilogy(Kk, eror1, 'g', 'LineWidth', 2);
69 semilogy(Kk, eror2, 'r', 'LineWidth', 2);
70 hold off;
71 legend('Hyperdual numbers','Complex-step method', 'Central difference
       method')
72 xlabel('k (Step size h=10^{-k})'), ylabel('Normalized error')
73 %semilogy(Kk,err_hd_vec,Kk, err1, Kk, err2,'r', 'LineWidth',2)
74 %figure;
75 %semilogy(Kk, err_hd_vec, 'b', 'LineWidth', 2);
```

```
%hold on;
%semilogy(Kk, err1, 'g', 'LineWidth', 2);
%semilogy(Kk, err2, 'r', 'LineWidth', 2);
%semilogy(Kk, err3, '--', 'LineWidth', 2);

%hold off;
%legend('Hyperdual numbers','Finite Difference', 'Complex variable
    method')
%xlabel('k (Step size h=10^{-k})'), ylabel('Normalized error')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A.2. Kodovi kod korištenja klase HD brojeva u Newtonovoj metodi

Kod za izvođenje algoritma 6:

```
function [X, N] = multinewton(funcx, x0, tol)
% Funkcija za rjesavanje sustava nelinearnih jednadzbi primjenom
% Newtonove metode
% Poziv funkcije: [X, N] = multinewton(funcx, x0, tol) gdje su
% X - izlaz --> rjesenja sustava jednadzbi
% N - izlaz --> broj iteracija Newtonove metode
% funcx - ulaz --> M-funkcija u kojoj su definirane jednadzbe sustava
% x0 - ulaz --> pocetne iteracije
% tol - ulaz --> tolerancija pogreske

N = 0;
X = x0;
err = 1;

while err > tol
    Ja_x = [HD_Jacobian_Call(@Jonj_mult_new_fija_1,X);HD_Jacobian_Call
    (...
        @Jonj_mul_new_fija_2,X);HD_Jacobian_Call(@Jonj_mult_new_fija_3
    ,X)];
    f = feval(funcx, X);
    XX = X - Ja_x\f;
    err = norm(XX-X);
```

```
    N = N + 1;
    X = XX;
end
```

Glavni kod za rješavanje problema pronalaženja ravnotežnog stanja sustava iz 5.6:

```
close all
clear all
clc

global n m

% dimenzije problema
n = 2;
m = 1;

% parametri sustava
grav = 9.81;
k1 = 130;
l1 = 0.6;
k2 = 120;
l2 = 0.6;
d = 1;
M = 0.25;

% pocetne aproksimacije
x10 = -0.3;
x20 = -0.8;
% x10 = 1.5;
% x20 = 0.7;
% x10 = 1;
% x20 = 0;

% inicijalizacija Newton-ove metode
X = [x10;x20];
err = 1;
Xvals = X;

i=1;

while err>=1e-6
```

```
36
37      %syms x1 x2
38      %x = [x1;x2];
39      %Ja_x = jacobian_analytic(x);
40      %Ja_x = double(subs(Ja_x, {x1,x2}, {X(1),X(2)}));
41      Ja_x = HD_Jacobian_Call(@potencijal,X);
42
43      %Ha_x = jacobian(jacobian_analytic(x),x);
44      %Ha_x = double(subs(Ha_x, {x1,x2}, {X(1),X(2)}));
45      Ha_x=HD_Hessian_Call(@potencijal,X);
46
47      XX = X - Ha_x\Ja_x(:); % racunanje tocaka sljedece iteracije
48      Xvals = [Xvals XX]; % formiranje polja tocaka po iteracijama
49      err = norm(Ja_x); % inf. norma gradijenta
50      er(i)=err;
51      i=1+i;
52      X = XX;
53
54  end
55
56  Xvals
57
58  x0 = [x10; x20]; % pocetne aproksimacije
59  options = optimset('Display','iter');
60  ff = @(x)M*grav*x(2) + k1/2*(l1 - sqrt(x(1)^2+x(2)^2))^2 + k2/2*(l2 -
        ...
61       sqrt((d-x(1))^2+x(2)^2))^2;
62  [x,fval,exitflag,output,grad,hessian] = fminunc(ff, x0, options)  ...
63      % poziv MATLAB-ove funkcije
64
65  [X, Y] = meshgrid(-0.5:0.01:1.5, -1.0:0.01:0.8);
66  Z = M*grav*Y + 0.5*k1*(l1 - sqrt(X.^2+Y.^2)).^2 + 0.5*k2*(l2 - ...
67      sqrt((d-X).^2+Y.^2)).^2;
68
69  figure(1)
70  contour(X,Y,Z,60), hold on
71  plot(Xvals(1,length(Xvals)),Xvals(2,length(Xvals)),'k*', x10,x20,'ko
        ',...
72       'LineWidth', 2)
```

```
xlabel('x'), ylabel('y')

[XX, YY] = meshgrid(-0.5:0.05:1.5, -1.0:0.05:0.8);
ZZ = M*grav*YY + 0.5*k1*(l1 - sqrt(XX.^2+YY.^2)).^2 + 0.5*k2*(l2 - ...
    sqrt((d-XX).^2+YY.^2)).^2;

figure(2)
surf(XX, YY, ZZ)
```

A.3. Kodovi za rješavanje optimizacijskih problema u MATLAB-u

A.3.1. Primjer reaktanta

Računanje diferencijalnih jednadžbi reaktanta prilagođenih aritmetici HD brojeva:

```
function x_final = my_euler(u, h, n_i)

% Euler method for ODEs solution

global k1r k2r k3r t0 T N tau t_vec x0 n


x1=x0(1);
x2=x0(2);
x3=x0(3);

%% Euler %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:n_i

    dx1=k1r*u(j)-k3r*x1;
    dx2=2*k2r*u(j).^2;
    dx3=k3r*x1;

    x1novi=x1+h*dx1;
    x2novi=x2+h*dx2;
    x3novi=x3+h*dx3;

```

```
    x1=x1novi;
    x2=x2novi;
    x3=x3novi;

end

xfin=[x1;x2];
x_final=[xfin;x3];
```

A.3.2. SCARA robot s 2-DOF

Glavni kod za izvođenje rješavanja optimizacijskog problema SCARA robota u oba slučaja:

```
close all
clear all
clc

global x0 n l1 lc1 lc2 m1 m2 I1 I2 N tau omega


l1 = 1;
lc1 = 0.5;
lc2 = 0.5;
m1 = 3;
m2 = 4;
I1 = 1;
I2 = 2;
%promijeniti na 4 u prvom slucaju
n = 5; % red sustava
nu = 2; % broj upravljackih varijabli

t0 = 0; % pocetno vrijeme
T = 2; % konacno vrijeme
N = 100; % broj vremenskih intervala
tau = (T-t0)/N; % korak integracije
t_vec = t0:tau:T; % vremenski interval
omega= pi/(2*T);
tu = t0:tau:T-tau;

```

```
Xd=2*cos(omega*t_vec);
Yd=2*sin(omega*t_vec);

x0 = [0 pi/4 0 0 0]; % pocetna stanja

u0 = zeros(nu, N); % pocetne upravljacke varijable

uopt = control(u0);

xout = my_euler_stari(uopt, tau, N);
%nakon sto dobijemo rjesenje sustava moguce je i plotati trajektoriju
   vrha
%drugog linka
xr=l1*cos(xout(1,:))+l1*cos(xout(1,:)+xout(2,:));
yr=l1*sin(xout(1,:))+l1*sin(xout(1,:)+xout(2,:));
figure(1)
subplot(221)
plot(t_vec(1:end), xout(1, :))
xlabel('t [s]'), ylabel('q_1 [rad]')

subplot(222)
plot(t_vec(1:end), xout(2, :))
xlabel('t [s]'), ylabel('q_2 [rad]')

subplot(223)
plot(t_vec(1:end), xout(3, :))
xlabel('t [s]'), ylabel('dq_1/dt [rad/s]')

subplot(224)
plot(t_vec(1:end), xout(4, :))
xlabel('t [s]'), ylabel('dq_2/dt [rad/s]')


figure(2)
subplot(211)
plot(tu, uopt(1, :))
xlabel('t [s]'), ylabel('u_1 [Nm]')

subplot(212)
```

```
plot(tu, uopt(2, :))
xlabel('t [s]'), ylabel('u_2 [Nm]')


figure(3)
plot(Xd,Yd,'b',xr,yr,'--r')
legend('Desired','Real trajectory')
xlabel('x [m]'), ylabel('y [m]')
```

Rješavanje diferencijalnih jednadžbi prilagođenih aritmetici HD brojeva:

```
function x = my_euler(u, tau, n_i)

global x0 l1 lc1 lc2 m1 m2 I1 I2 omega

x1=x0(1);
x2=x0(2);
x3=x0(3);
x4=x0(4);
x5=x0(5);

%clanovi matrica koji ne ovise o koraku
M11 = m1*lc1^2 + m2*l1^2 + I1;
M22 = m2*lc2^2 + I2;

C11 = 0;
C22=0;

%% Euler %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 1:n_i-1

    u1=u(1,j);
    u2=u(2,j);
    %racunanje clanova matrica koji ovise o koraku
    M12 = m2*l1*lc2*cos(x2-x1);
    M21 = M12;

    C12 = -m2*l1*lc2*sin(x2-x1)*x4;
    C21 = m2*l1*lc2*sin(x2-x1)*x3;

%%
```

```
31
32      K=(M11*M22-M12*M21).^(-1);
33      m_new11=K*M22;
34      m_new12=K*-M12;
35      m_new21=K*-M21;
36      m_new22=K*M11;
37      %%%%%%%%%%%%%%%%%%%%
38  %%
39
40      %nakon racunanja matrica mozemo izracunati vrijednosti promjena
41      %varijabli stanja dx:
42      dx1=x3;
43      dx2=x4;
44      %Racunanje elemenata potrebnih za petu varijablu stanja:
45      Xr=l1*cos(x1)+l1*cos(x1+x2);
46      Yr=l1*sin(x1)+l1*sin(x1+x2);
47      Xd=2*cos(omega*(j)*tau);
48      Yd=2*sin(omega*(j)*tau);
49      %%%%%
50
51      mc11=m_new12*C21;
52      mc12=m_new11*C12;
53      mc21=m_new22*C21;
54      mc22=m_new21*C12;
55
56      dx3=m_new11.*u1+m_new12.*u2-(mc11*x3+mc12*x4);
57      dx4=m_new21.*u1+m_new22.*u2-(mc21*x3+mc22*x4);
58      dx5=(Xr-Xd).^2+(Yr-Yd).^2;
59
60      x1novi=x1+tau*dx1;
61      x2novi=x2+tau*dx2;
62      x3novi=x3+tau*dx3;
63      x4novi=x4+tau*dx4;
64      x5novi=x5+tau*dx5;
65      x1=x1novi;
66      x2=x2novi;
67      x3=x3novi;
68      x4=x4novi;
69      x5=x5novi;
```

```
70
71 end
72 x=[x1;x2];
73 x=[x;x3];
74 x=[x;x4];
75 x=[x;x5];
```

Literatura

[1] I. Aganović and K. Veselić. *Matematički modeli i metode*. Osijek: Sveučilište Josip Juraj Strossmayer u Osijeku, 2014.

[2] R. Kelly, V. Santibá nez, and A. Loría. *Control of Robot Manipulators in Joint Space*. Springer-Verlag, 2005.

[3] M. P. Neuenhofen. Review of theory and implementation of hyper-dual numbers for first and second order automatic differentiation. https://arxiv.org/abs/1801.03614, 2018.

[4] M. P. Neuenhofen. Note on usage and theory of hyper-dual numbers for first and second order automatic differentiation. https://arxiv.org/abs/1801.03614v1, 2018.

[5] J. A. Fike and J. Alonso. The development of hyper-dual numbers for exact second-derivative calculations. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 17, 2012.

[6] R. D. Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM Review*, 52(3):545–563, 2010.

[7] J. A. Fike. *Multi-objective optimization using hyper-dual numbers*. Phd thesis, Stanford university, 2013.

[8] K.-L. Lai and J.L. Crassidis. Extensions of the first and second complex-step derivative approximations. *Journal of Computational and Applied Mathematics*, 219(1):276–293, 2008.

[9] V. Milić. Računalna matematika, vježbe. https://titan.fsb.hr/~vmilic/RacMat/Vjezbe_mehatron/vj_03/Koracna_kompleksna_metoda.pdf. Pristupljeno: 2024-01-19.

[10] V. Milić. Računalna Matematika, predavanja. https://titan.fsb.hr/~vmilic/RacMat/Predavanja_21/Predavanje_07_Optimizacija/Predavanje_07_Optimizacija.pdfl. Pristupljeno: 2023-08-26.

[11] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer Nature, 2006.

[12] S. Gros and M. Diehl. *Numerical Optimal Control, Lecture notes*. University of Freiburg, 2022.