Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Simon Beltram

Evolucijska optimizacija geometrije profila letalskega krila

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Marko Robnik-Šikonja
Somentor: doc. dr. Gregor Papa

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Profil letalskega krila pomembno vpliva na lastnosti letala in njegove letalne značilnosti. Ker so letalne lastnosti, ki jih želimo doseči, večkrat med seboj nasprotujoče in so v praksi posledica številnih faktorjev, jih pogosto poskušamo načrtovati s pomočjo evolucijskih optimizacijskih algoritmov. Predstavite problem optimizacije profila letalskega krila v obliki, ki bo primerna za evolucijske algoritme. Pristop preverite na problemu načrtovanja profila letalskega krila za prostoleteče modele letal kategorije F1A. Analizirajte in medsebojno primerjajte delovanje različnih optimizacijskih algoritmov. Izdelajte prototip rešitve, ki omogoča optimizacijo za različne kriterije in vizualizacijo rešitev.

Zahvaljujem se mentorju, izr. prof. dr. Marku Robniku-Šikonji, za vso pomoč pri izdelavi diplomske naloge ter somentorju, doc. dr. Gregorju Papi, za strokovno pomoč v času diplome in delovne prakse na Institutu "Jožef Stefan". Nenazadnje se zahvaljujem svoji družini, ki mi je omogočila študij in me na tej poti podpirala.
Hvala.

Kazalo

Povzetek

Naslov: Evolucijska optimizacija geometrije profila letalskega krila

V diplomski nalogi je obravnavana hevristična optimizacija geometrije profila krila prostoletečih modelov letal kategorije F1A. Preveriti želimo ali je mogoče s pomočjo hevristične optimizacije generirati profil, ki bi bil boljši od tistih, ki so trenutno v uporabi. V ta namen je bila razvita programska rešitev, ki ima implementirane optimizacijske algoritme diferencialna evolucija (DE), metoda roja delcev (PSO) ter metoda ognjemeta (FA). Profili, ki so trenutno v uporabi, se imenujejo profili LDA (low-drag airfoil) in so bili razviti na podlagi znanja ter izkušenj. Profili LDA lahko dosegajo visoko hitrost v fazi pridobivanja višine, a nekoliko slabše prosto letijo v primerjavi s predhodniki. S pomočjo hevrističnih optimizacijskih metod poskušamo pridobiti tak profil, ki bi lahko obe nalogi opravljal dovolj dobro.
Vsi trije algoritmi so bili implementirani v programskem jeziku java, s pomočjo programske knjižnice JMetal, ki jo je bilo potrebno ustrezno prilagoditi. Največja sprememba je način predstavitve osebkov. V našem primeru je profil letalskega krila predstavljen s parom Bézierjevih krivulj, ena za zgornjo in druga za spodnjo polovico krila. Tak način predstavitve nam da veliko kontrole in svobode za ceno morebitnih nepravilnih oblik (zaradi naključnosti). Zato je potrebno pri generiranju profilov paziti tudi na konstrukcijske omejitve (pretanki ali predebeli profili, itd.). Profile ocenjujemo s pomočjo Xfoil [3] programa, ki preračuna karakteristike profila. Izkazalo se je, da upoštevanje le enega kriterija kakovosti ni dovolj, zato bi bilo smiselno upoštevati še druge parametre in implementirati večkriterijski različici algoritmov.

Ključne besede: hevristična optimizacija, diferencialna evolucija, optimizacija z metodo ognjemeta, optimizacija z rojem delcev, profil letalskega krila.

Abstract

Title: Evolutionary optimization of aerofoil profile

This thesis deals with the heuristic optimization of an F1A free flight airfoil. The goal is to generate an airfoil that is better than the ones currently in use. We implemented three optimization algorithms, differential evolution (DE), Firework algorithm (FA) and Particle swarm optimization (PSO). Currently used airfoils, called LDA (Low-drag airfoil) are able to obtain high speed in the phase of climbing but in the free flight phase they fly poorly compared to their predecessors. With the help of optimization methods we aim to find an airfoil that performs both tasks well.
All heuristic algorithms were developed in java programming language using a customized JMetal library. Each airfoil is presented with two Bézier curves, one for the upper and one for the lower half of the foil. This kind of presentation gives us more freedom and control, but it comes at a price - the software may generate irregular shapes due to randomness. That is why it is important to pay attention to structural constraints (i.e. airfoil thickness). Airfoils are evaluated using Xfoil [3] which calculates the characteristics of an airfoil. Our experiments show that L/D ratio is not a good indicator of airfoil quality and one has to take into account other parameters as well. A multi-objective criteria function would therefore be advisable.

Keywords: heuristic optimization, differential evolution, firework algorithm, particle swarm optimization, airfoil.

Poglavje 1

Uvod

Letalsko krilo je najpomembnejša nosilna površina letala. Ta ustvarja silo, ki nasprotuje sili težnosti in omogoča, da letalo leti. Običajno je, da krilu posvečamo največ pozornosti, kajti od oblike in lastnosti krila so odvisne tudi značilnosti celotnega letala. Pri oblikovanju krila se najprej osredotočimo na dvodimenzionalno obliko preseka krila oz. profila, šele nato dodamo tretjo dimenzijo in upoštevamo dejavnike vzdolž krila. Preden začnemo oblikovati profil je potrebno razmisliti, v kakšnih razmerah bo letalo letelo in katere segmente leta želimo izpopolniti (npr. planiranje, vzpenjanje, najvišjo hitrost, itd.), definirati moramo cilje, ki jih želimo doseči. Nato začrtamo začetno obliko profila in jo testiramo. Profil lahko testiramo v vetrovniku ali, v našem primeru, s pomočjo računalniške simulacije. V kolikor nismo zadovoljni z rezultati, obliko popravimo in ponovno testiramo. Postopek ponavljamo dokler ne pridemo do oblike, ki zadosti zadanim ciljem. Tak postopek je dolgotrajen in drag, poleg tega potrebujemo ekipo strokovnjakov s področja letalstva oz. aerodinamike, ki imajo izkušnje z oblikovanjem letalskih profilov. Lahko pa tudi iz zbirke obstoječih profilov izberemo takega ki najbolj ustreza našim potrebam. Ampak to je daleč od optimalne rešitve.

V raziskavi se bom posvetil krilom prostoletečih modelov letal kategorije F1A. Razlog za to so dolgoletne izkušnje s tekmovalnim modelarstvom. Vprašanje, na katerega skušamo odgovoriti je: ali je mogoče s hevrističnimi optimiza-

cijskimi metodami razviti profil, ki bi bil konkurenčen profilom razvitim na podlagi znanja in izkušenj.

Za krila F1A modelov je značilna nizka hitrost letenja ter zmožnost doseganja visoke hitrosti v vzpenjanju. Idealen profil bi bil torej sposoben dobro leteti pri nizkih hitrostih in razviti visoke hitrosti v fazi vzpenjanja, kar sta nasprotujoča si cilja, zato bomo s pomočjo optimizacijskih metod poskusili dobiti dobro kompromisno rešitev. V ta namen je bila izdelana programska rešitev, ki vsebuje hevristično optimizacijo.

Hevristična optimizacija je metoda reševanja težkih optimizacijskih problemov. Vsi trije obravnavani algoritmi upravljajo s populacijo možnih rešitev, ki so na začetku enakomerno razpršeni po preiskovalnem prostoru. Tekom optimizacije se rešitve skoncentrirajo okoli vrhov kriterijske funkcije. Hevristični optimizacijski algoritmi ne zagotavljajo, da bo najdena rešitev optimalna. Iskanje usmerjamo s pomočjo operatorjev kot so križanje, mutacija in selekcija. Pogosta težava pri optimiranju so lokalni optimumi, ki otežujejo iskanje optimalne rešitve. Tej težavi se izognemo z dobro nastavljenimi parametri optimizacijskega algoritma. Primerne nastavitve določimo s sistematičnim poskušanjem. Oba algoritma sta stohastična, kar pomeni, da je populacija na začetku naključno generirana in z vsakim zagonom dobimo drugačne rešitve. Zato je potrebno večkrat zagnati algoritem z istimi parametri, da lahko določimo povprečno uspešnost.

Izbira predstavitve osebkov in podatkovnih struktur je pri naši optimizaciji ključnega pomena. Od tega je odvisna računska zahtevnost algoritma. V našem primeru moramo geometrijo profila matematično opisati oz. parametrizirati. Profil predstavimo s parom Bézierjevih krivulj, eno za zgornjo in drugo za spodnjo polovico profila. Bézierjevo krivuljo določa množica kontrolnih točk, ki so osebki optimizacijskega algoritma. Kontrolne točke so shranjene v enodimenzionalnem polju tako, da je na i-tem mestu shranjena x koordinata in na $i+1$ mestu y koordinata. Tak način predstavitve nam da več kontrole in svobode kot metoda PARSEC [9] za ceno morebitnih nepravilnih oblik (zaradi naključnosti), zato je potrebno pri generiranju profilov paziti

tudi na konstrukcijske omejitve (pretanki ali predebeli profili, itd.). Profile ocenjujemo s pomočjo Xfoil programa, ki preračuna karakteristike profila.
Pomemben dejavnik pri optimizaciji je izbira kriterijske funkcije. Ta ocenjuje kakovost rešitve. V primeru optimizacije geometrije profila je kriterijska funkcija razmerje L/D (ang.: lift-to-drag ratio) oz. razmerje med vzgonom in uporom. Ta nam pove, kakšne so zmogljivosti profila. Razmerje L/D se izračuna med koeficientom vzgona in koeficientom upora, ki ju določi program Xfoil.
V 2. poglavju je podrobneje opisana kategorija prostoletečih modelov letal F1A. Preden pričnemo z optimizacijo profilov moramo spoznati osnovne principe letenja, zato v 3. poglavju predstavimo aerodinamiko. V 4. poglavju je opisano področje optimizacije. V 5. poglavju predstavimo zasnovo naše programske rešitve. V 6. poglavju pa analiziramo uporabljene algoritme in nekatere profile. V zadnjem poglavju povzamemo narejeno delo in predstavimo ideje za izboljšave.

Poglavje 2

Prostoleteči modeli letal F1A

V kategorijo prostoletečih jadralnih modelov letal spadajo modeli, ki ne potrebujejo zunanjega upravljanja v fazi prostega letenja (npr. radijsko vodenje) in nimajo nikakršnega pogona. Edina oblika kontrole v fazi pred prostim letenjem je s pomočjo 50 metrov dolge vlečne vrvice. Modeli, ki nimajo pogona, izkoriščajo dviganje toplega zraka t.i. termiko. Za letenje F1A modelov sta potrebna dva, pomočnik in tekmovalec. Pomočnik na začetku pridrži model nad glavo, tekmovalec pa stoji 50 metrov stran z vrvico v roki, ki je zapeta za vlečno kljuko na modelu. S kratkim zaletom tekmovalec model spravi v zrak in ga s pomočjo vrvice upravlja. To fazo imenujemo kroženje, v njej tekmovalec išče področje, kjer je največja termična aktivnost. Termične stebre zaznava z opazovanjem obnašanja modela. Potem ko tekmovalec najde primeren kraj, nastopi faza odklopa. V tej fazi tekmovalec spusti model nekoliko nižje in s šprintom pospeši model, ki lahko doseže hitrost 35 m/s. Ko je model skoraj nad glavo tekmovalec s hitrim zamahom roke navzdol odpne vlečno kljuko in model se s pomočjo mikrokrmilnika postavi vertikalno. Z visoko hitrostjo se model prične vertikalno vzpenjati. Na ta način lahko pridobimo do 50 metrov dodatne višine. Potem mikrokrmilnik, s primernim pomikom stabilizacijskih površin, poskrbi, da model iz vertikalnega preide ponovno v horizontalni položaj. Od tu dalje prosto leti brez kontrole in je prepuščen zračnim tokovom. Za doseganje tako visokih hitrosti mora biti

Slika 2.1: Model kategorije F1A z zapeto vlečno vrvico v fazi kroženja (Foto: R. Koglot).

celoten model dovolj trden in imeti primerno obliko profila. Zato so sodobna krila in trupi zgrajeni iz karbonskih vlaken. Po določenem času nastopi še faza determalizacije. V tej fazi prisilimo model v pristanek. Z dvigom repa model izgubi hitrost in prične kontrolirano (v spirali) padati proti tlom. V kolikor model med letenjem ne uide iz termičnega stebra, lahko v treh minutah doseže višino tudi do 200 metrov in lahko pristane več kilometrov od mesta odklopa. Zato so modeli opremljeni z radijskimi oddajniki in GPS napravami. V trupu modela se nahaja mikrokrmilniško vezje oz. timer, servomotorji in vlečna kljuka, na katero zapnemo vlečno vrvico. Timer premika stabilizacijske površine ob vnaprej določenih časovnih intervalih ali ob aktivaciji mikrostikal (preko vlečne kljuke). V rednem delu tekmovanja se opravi sedem turnusov oz. letov. Cilj tekmovalca v posameznem turnusu je odleteti maksimalni čas, ki je običajno 3 minute, a se lahko spreminja glede na razmere na tekmovališču. Če ima po sedmih turnusih več tekmovalcev enak rezultat, se izvede t.i. fly-off, ko se poveča maksimalni čas. Fly-off se običajno izvede nekoliko pozneje v popoldnevu in tako postane težje doseči maksimalen čas, saj je takrat manj termične aktivnosti [7].

Razpetina kril varira med 2.20 m in 2.50 m. Za njih je značilna izredna sta-

bilnost v prostem letu pri nizkih hitrostih in zmožnost doseganja visokih hitrosti.
Omejitve po FAI pravilniku [6]:

- največja dovoljena površina nosilnih površin: 34 dm^2,
- dolžina vlečne vrvice: 50 m (obtežena na 5 kg),
- najmanjša dovoljena teža: 410 g,
- nosilne površine se ne smejo premikati (vrteti ali mahati),
- lahko se spreminja samo ukrivljenost profila in vpadni kot krila.

Poglavje 3

Aerodinamika

Da bi lahko optimizirali geometrijo profilov, moramo najprej razumeti, zakaj sploh letala letijo.V tem poglavju se bomo posvetili osnovam aerodinamike.

3.1 Sestavni deli letala

Ena izmed nosilnih površin na letalu so krila. Krila omogočajo letalu, da leti, saj proizvajajo silo, ki nasprotuje gravitacijski sili, t.i. silo vzgona (usmerjena je pravokotno navzgor na smer letenja). Ta nastaja ob premikanju letala skozi zrak. Hkrati nastaja tudi sila upora, ki je usmerjena nasprotno smeri letenja. Da bi letalo uspešno letelo, se mora letalo premikati z dovolj veliko hitrostjo, da bo sila vzgona večja od gravitacijske sile (pri čim manjši sili upora). Poleg krila letalo uporablja t.i. stabilizacijske površine. Te opravljajo nalogo stabilizacije in krmiljenja. Krilo se običajno nahaja na prednjem delu letala oz. pred težiščem. To pomeni, da ima zaradi vzgona skoncentriranega na prednjem delu letala (t.i. nos), letalo težnjo dvigovanja prednjega dela letala. Da bi letalo doseglo stabilnost po prečni osi (vodoravni let) mora sila vzgona višinskega krmila (rep) postavljenega na zadnjem koncu letala, delovati navzdol (v nasprotni smeri kot krilo). Poleg repa imamo tudi smerni oz. vertikalni stabilizator, ki obrača letalo po vertikalni osi. Letalo ima možnost tudi nagibanja oz. prevračanja po vzdolžni osi s pomočjo zakrilc. Slednje se

nahajajo pri zadnjem robu krila.

3.2 Vzgon

Vzgon je sila, ki nastane, ko kapljevina (v našem primeru zrak) teče mimo telesa. Sila je pravokotna na tok. Vzgon na krilu nastane zaradi razlike v tlaku na zgornji in spodnji strani krila. Na zgornji strani krila je manjši tlak (celo podtlak) kot na spodnji strani, kar pomeni, da je rezultanta sil usmerjena navzgor. Količina vzgona je odvisna od:

- oblike profila,
- hitrosti in gostote kapljevine (v našem primeru zraka),
- dolžine profila ali površine krila,
- vpadnega kota profila na tok zraka.

Teorij, ki razlagajo, zakaj pride do razlike v tlaku med zgornjo in spodnjo ploskvijo, je več. Bralec jih najde v [15].

3.3 Laminarni in turbulentni tok

Ugotovili smo, kako nastane sila, ki drži letalo v zraku, nismo pa še spoznali kaj se dogaja s tokom ob stiku s površino profila. Preden tok doseže profil je laminaren (če ne upoštevamo turbulence v ozračju). To pomeni, da tok teče enakomerno v "plasteh", brez motenj. Ko se tok približa površini profila, del toka, ki je najbljižji površini, upočasni zaradi trenja. Bližje kot smo površini profila, manjša je hitrost toka. Izkaže se, da je hitrost toka na površini profila enaka nič, kar je posledica viskoznosti kapljevine. To območje imenujemo mejna plast in deluje kot zračni upor, saj (aerodinamično gledano) poveča efektivno debelino profila.

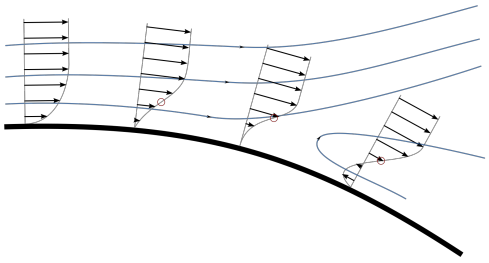
- Laminarni tok: Laminarna mejna plast proizvaja najmanj upora, zato si želimo, da bi tak tok obstajal po celotni dolžini profila. Za laminarno

mejno plast je značilno, da hitrost plasti enakomerno pada, bolj kot se približujemo površini profila. Hkrati velja opomniti, da laminarni tok zaradi majhne energije ni stabilen in slej ko prej postane turbulenten.

- Turbulentni tok: Ta nastane, ko se mejna plast loči od površine in nastanejo vrtinci. Do ločitve pride, ko se plast toka, ki je najbližja površini profila, začne pomikati v nasprotno smer. To se zgodi, ko tok pride iz območja nizkega v območje visokega statičnega tlaka, kar povzroči zmanjšanje hitrosti toka. Na profilu se to zgodi na neki razdalji za najvišjo točko ukrivljenosti. Slika 3.1 prikazuje štiri profile hitrosti pri tranziciji iz laminarnega v turbulentni tok. Puščice prikazujejo smer premikanja zraka, njihova dolžina pa hitrost. Iz profilov hitrosti je razvidno, da hitrost pada bližje kot smo površini krila. Pri zadnjem profilu hitrosti se smer premikanja zraka obrne, kar povzroči odcepitev od površine, povečanje debeline mejne plasti in kasneje prehod v turbulentni tok.

Sedaj lahko definiramo, kakšen je idealen profil:

- ima dovolj velik koeficient vzgona,
- ima čim nižji koeficient upora,
- ima čim manjšo mejno plast,
- ima mejno plast, ki poteka od začetka do konca krila.



Slika 3.1: Profili hitrosti mejne plasti [19].

Poglavje 4

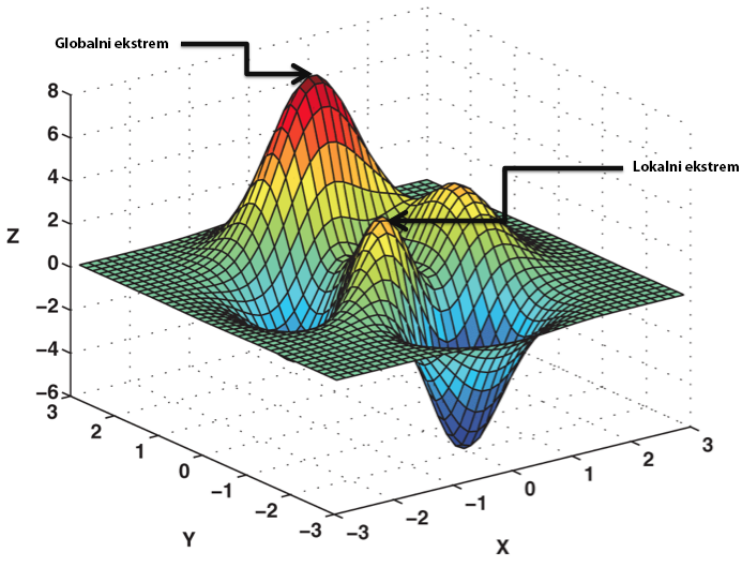
Hevristična optimizacija

Optimizacija je postopek, ki skuša najti maksimalno ali minimalno vrednost neke kriterijske funkcije (znotraj danega območja). Hevristično optimizacijo običajno uporabljamo v primeru, ko imamo problem, ki ga je praktično nemogoče točno rešiti v doglednem času in se moramo zadovoljiti s približkom prave rešitve. Optimizacijo definirajmo za primer minimizacije:

Minimiziraj: $f_0(x)$
ob upoštevanju: $g_i(x) \quad i = 1, ...n$

- $x \in R^n$(vektor) vrednosti spremenljivk
- $f_0(x)$ kriterijska funkcija, ki jo minimiziramo
- $g_1(x), ..., g_n(x)$ omejitve

Spremenljivka x lahko predstavlja npr. stanje vrednostnih papirjev, opravilo znotraj urnika, premik kontrolne površine pri letalu, načrt oz. obliko (naprave, vezja, ...). Kriterijska funkcija je lahko npr. dobiček, oddaljenost od željene vrednosti, poraba goriva, kombinacija cene, teže, moči itd.
Definicijsko območje kriterijske funkcije je običajno dodatno omejeno z množico omejitev. Veljavno območje imenujemo preiskovalni prostor. V preiskovalnem prostoru se nahaja množica dopustnih rešitev, ki izpolnjujejo omejitve. Tipično je ta prostor prevelik, da bi ga lahko izčrpno preiskali. Optimalna



Slika 4.1: Preiskovalni prostor z globalnim in lokalnimi ekstremi [10].

rešitev je tista dopustna rešitev, ki minimizira oz. maksimizira dano kriterijsko funkcijo. Pri postopku optimizacije skušamo najti globalni ekstrem funkcije. Pri tem nas lahko ovirajo številni lokalni ekstremi, ki postopek optimizacije zapeljejo v napačno smer (slika 4.1). Iz tega sledi, da hevristični optimizacijski algoritmi ne zagotavljajo optimalnih rešitev. Kriterijska funkcija je lahko tudi sestavljena iz več komponent.

$$\min F(x) = min(f_1(x), f_2(x), ..., f_n(x))$$

Tako optimiranje imenujemo večkriterijska optimizacija. Pogosto pri večkriterijski optimizaciji nastopajo nasprotujoči si cilji. To pomeni, da ne moremo izboljšati posamezne komponente, ne da bi poslabšali vsaj eno od ostalih komponent. V takem primeru moramo najti kompromis. Večkriterijska optimizacija vrne nabor t.i. pareto-optimalnih rešitev. Rešitev je pareto optimalna, če pareto napredek ni mogoč. Pareto napredek je definiran kot sprememba, ki izboljša vsaj eno rešitev, ne da bi poslabšala ostale. Končno rešitev izbere strokovnjak iz domene glede na nek dodatni kriterij (izkušnje, preference, itd.).

4.1 Diferencialna evolucija

Diferencialna evolucija je stohastični preiskovalni algoritem, ki je bil razvit za preiskovanje v zveznem prostoru. Razvila sta ga Rainer Storn in Ken Price leta 1995 [13]. Algoritem DE odlikujejo enostavnost implementacije, odlične performance v primerjavi z ostalimi sorodnimi metodami, majhno število kontrolnih parametrov ter nizka prostorska kompleksnost, kar omogoča delovanje tudi pri visoko dimenzionalnih problemih.
Pri algoritmu DE na kromosome gledamo kot na D-dimenzionalne vektorje realnih števil, kjer ima vrednost vsakega gena določeno spodnjo in zgornjo mejo. Za razliko od ostalih evolucijskih metod algoritem DE uporablja mutacijo kot glavno orodje za generiranje potomcev. Ta deluje na principu utežene razlike osebkov. Križanje opravlja delo običajne mutacije, ki spreminja posamezne naključno določene gene.

Inicializacija

Najprej moramo generirati naključno začetno populacijo. Začetna populacija bo vsebovala NP D-dimenzionalnih kromosomov. Število generacije označimo z $G = 0, 1, ..., G_{max}$. Notacija i-tega vektorja [4]

$$\overrightarrow{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, ..., x_{D,i,G}]$$

Vrednost vsakega gena je omejena. Običajno so te vrednosti povezane z vrednostmi, ki so naravno omejene (npr. dolžina in teža ne morata biti negativni). Spodnjo mejo genov lahko zapišemo kot:

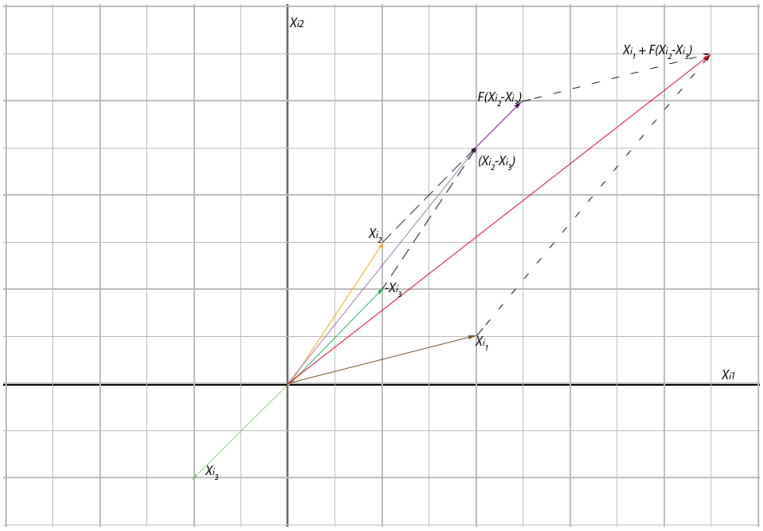
$$\overrightarrow{X}_{min} = \{x_{1,min}, x_{2,min}, x_{3,min}, \ldots, \ x_{D,min}\}.$$

Podobno lahko zapišemo še zgornjo mejo:

$$\overrightarrow{X}_{max} = \{x_{1,max}, x_{2,max}, x_{3,max}, \ldots, \ x_{D,max}\}.$$

Definiramo postopek generiranja naključne populacije (število generacije je 0) [4].

$$x_{j,i,0} = x_{j,min} + rand_{i,j}[0, 1] * (x_{j,max} - x_{j,min}),$$



Slika 4.2: Grafični prikaz mutacije.

kjer $rand_{i,j}[0,1]$ vrne naključno število med 0 in 1 iz enakomerne porazdelitve.

Mutacija

Operator mutacije generira preizkusni vektor u_i za vsak vektor v populaciji tako, da mutira ciljni vektor x_{i_1} z uteženo razliko. Preizkusni vektor bo kasneje uporabljen pri postopku križanja. Za vsak ciljni vektor x_{i_1}, iz populacije naključno izberemo dva vektorja x_{i_2} in x_{i_3}. Preizkusni vektor dobimo po formuli [4]

$$u_i = x_{i_1} + F * (x_{i_2} - x_{i_3})$$

kjer $i \neq i_1 \neq i_2 \neq i_3$ in $F \in [0, \infty]$ je utež, ki uravnava razliko med vektorjema x_{i_2} in x_{i_3}. Slika 4.2 prikazuje postopek mutacije ciljnega vektorja.

Križanje

Operator križanja vrši diskretno rekombinacijo med preizkusnim vektorjem u_i in staršem x_i. Potomca označimo z x'_i. Križanje je definirano s pravilom [4]:

$$x'_{ij} = \begin{cases} u_{ij} & \text{če } j \in \mathcal{J}, \\ x_{ij} & \text{sicer} \end{cases}$$

kjer se x_{ij} nanaša na j-ti element vektorja x_i in je $\mathcal{J}$ množica mest (kazalcev), kjer bomo vršili križanje. Potomca uvrstimo v novo populacijo, če je boljši od starša, v nasprotnem primeru ohranimo starša.

4.2 Inteligenca roja

Inteligenca roja (ang.: swarm inteligence - SI) se zgleduje po naravi, kjer se določene živali združijo v jate oz. roje. Posamezna ptica bo prej prišla do hrane, če sodeluje v skupnem iskanju z drugimi pticami, kjer vsaka sporoča okoliškim pticam svoje najdbe. Kljub temu, da so posamezniki znotraj roja preprosti, je socialna interakcija lahko kompleksna. Bistvo podobnih algoritmov SI je modeliranje enostavnega obnašanja osebkov ter njihove interakcije z okoljem in sosednjimi osebki. Tako dobimo kompleksne vedenjske vzorce, s katerimi lahko rešimo različne probleme [4]. Za razliko od evolucijskih algoritmov, kjer se informacije iz generacije v generacijo prenašajo preko genetskega zapisa, se tukaj informacije prenašajo s pomočjo komunikacije. Osebki komunicirajo med seboj, da bi dosegli skupni cilj.

4.2.1 Metoda roja delcev

Metoda roja delcev (ang.: Particle swarm optimization - PSO) je tipični predstavnik inteligence roja. Algoritem PSO vzdržuje populacijo delcev, kjer vsak delec predstavlja možno rešitev. Delec je vektor v večdimenzionalnem prostoru in ta se premika s pomočjo vektorja hitrosti, ki se spreminja v vsaki iteraciji. Spodnji izraz definira položaj i-tega delca v naslednjem časovnem kraku [4]:

$$x_i(t+1) = x_i(t) + v_i(t+1),$$

kjer je $x_i(t)$ trenutni položaj i-tega delca in je $v_i(t+1)$ hitrost istega delca v naslednjem časovnem koraku. Potek optimizacije nadzoruje smer in velikost hitrosti delcev. Hitrost je sestavljena iz dveh komponent. Prva, kognitivna komponenta, predstavlja izkušnje posameznega delca, ki kaže na najboljše obiskano mesto do tega trenutka. Druga komponenta je t.i. socialna komponenta, ki kaže na najboljše najdeno mesto delcev v okolici. Da ne bi prišlo do prevelike spremembe smeri, vsak delec teži k prej obiskani lokaciji. To si lahko predstavljamo kot vztrajnost delca.
Glede na to, kako se posamezen delec povezuje z okoliškimi delci, razlikujemo dve različici PSO algoritma.

Globalni PSO

Pri globalni različici algoritma vsi delci komunicirajo med seboj. Informacija, ki jo posamezen delec dobi od celotnega roja, je globalno najboljša lokacija do sedaj. Običajno se okolico za vsak delec določi glede na vrstni red v podatkovni strukturi, saj je veliko hitreje kot določitev gruče glede na lokacijo v preiskovalnem prostoru. Za določitev okolice oz. gruč glede na lokacijo je potrebno izračunati medsebojno Evklidsko razdaljo med vsemi pari delcev. Hitrost za i-ti delec dobimo s spodnjim izrazom.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

$v_{ij}(t+1)$ je j-ta vrednost i-tega vektorja hitrosti v naslednjem časovnem koraku. Izraz je razdeljen na tri člene. Prvi člen $v_{ij}(t)$ predstavlja trenutno vrednost hitrosti, drugi kognitivno in tretji socialno komponento. Pri kognitivni komponenti $y_{ij}(t)$ je najboljša lokacija i-tega delca do sedaj, pri socialni $\hat{y}_j(t)$ pa globano najboljša lokacija celotnega roja najdena do sedaj. Konstanti c_1 in c_2 določata vpliv posamezne komponente, $r_{1j}(t)$ in $r_{2j}(t)$ pa sta naključni vrednosti med 0 in 1, ki dodajata naključni šum in se spreminjata z vsako iteracijo.

Lokalni PSO

Pri lokalni različici PSO algoritma se delec povezuje z nekaj sosednjimi delci tako, da tvori socialno mrežo v obliki sklenjenega kroga. V gručah si delci delijo znanje o bližnji okolici preiskovalnega prostora. Gruče se lahko med seboj prekrivajo, kar omogoča prenos informacije med gručami in posledično konvergenco delcev v eno točko. Enačba za izračun hitrosti delca ostane enaka, le $\hat{y}_j(t)$ sedaj predstavlja najboljšo najdeno lokacijo do sedaj s strani sosednjih delcev, ki so del okolice.

4.3 Metoda ognjemeta

Metoda ognjemeta (ang.: Fireworks algorithm - FA) simulira obnašanje isker pri eksploziji ognjemeta. Pri algoritmu FA se na začetku naključno določi začetne lokacije ognjemetov. Nato te točke eksplodiramo in iskre zapolnijo preiskovalni prostor okoli ognjemeta. Število isker in amplituda eksplozije se določa glede na strategijo. Eksplozija je glavno orodje preiskovanja, zato je pomembno, da je dobro načrtovana. Po evalvaciji isker in ognjemetov sledi selekcija. V novo generacijo najprej pošljemo najboljši osebek iz trenutne populacije (elitizem). Preostale osebke izberemo glede na njihovo medsebojno oddaljenost. Postopek eksplodiranja ognjemetov ponavljamo dokler ne najdemo optimalne rešiteve.

Eksplozija ognjemetov

Eksplozija ognjemetov je glavni mehanizem preiskovanja in generiranja novih osebkov. Najprej je potrebno določiti dva parametra. Prvi je število isker, ki jih eksplozija generira [16].

$$S_i = S * \frac{Y_{max} - f(x_i) + \epsilon}{\sum_{i=1}^{N}(Y_{max} - f(x_i)) + \epsilon}.$$

S_i predstavlja število isker za i-ti ognjemet ($1 \leq i \leq N$). Konstanta S uravnava končno število isker. Če je naš cilj najti minimum funkcije, Y_{max} predstavlja najslabšo vrednost kriterijske funkcije, Y_{min} pa najboljšo; $f(x_i)$ predstavlja vrednost kriterijske funkcije osebka x_i. Vrednost ϵ preprečuje, da bi imenovalec postal nič, običajno ga nastavimo na majhno vrednost. Nato je potrebno določiti amplitudo isker. To določimo z izrazom [16]:

$$A_i = A * \frac{f(x_i) - Y_{min} + \epsilon}{\sum_{i=1}^{N}(f(x_i) - Y_{min}) + \epsilon}$$

A_i predstavlja amplitudo i-te eksplozije, konstanta A pa uravnava končno amplitudo. V primeru minimizacije Y_{min} predstavlja najboljšo vrednost kriterijske funkcije in, tako kot v prejšjni formuli, ϵ preprečuje, da imenovalec postane 0. Po eksploziji ognjemeta se lahko zgodi, da nekatere iskre uidejo iz preiskovalnega prostora. V takem primeru opravimo preslikavo, ki iskre preslika nazaj v preiskovalni prostor. To storimo z izrazom [16]:

$$x_i = x_{min} + |x_i| \bmod (x_{max} - x_{min}),$$

kjer x_i predstavlja i-to iskro, x_{max} in x_{min} pa predstavljata zgornjo in spodnjo mejo preiskovalnega prostora.

Mutacija

Pogleg eksplozije uporabimo tudi poseben operator, ki doda nekaj naključnega šuma, da ohranimo diverziteto. Šum dodamo tako, da nekatere iskre naključno zamaknemo. Naključna števila generiramo po Gaussovi porazdelitvi. Obstoječe iskre zamaknemo po formuli [16]:

$$x_k^j = x_k^j * g,$$

kjer x_k^j predstavlja j-to koordinato k-te iskre. Parameter g je naključna spremenljivka iz Gaussove porazdelitve s povprečjem in standardnim odklonom 1.

$$g = \mathrm{N}(1,1)$$

4.3.1 Selekcija

Algoritem FA uporablja koncept elitizma. To pomeni, da najboljši osebek v trenutni populaciji preživi v naslednjo generacijo. Ostalih $N-1$ osebkov izbiramo na podlagi medsebojne razdalje. Splošna oblika formule za razdaljo je definirana kot [16]:

$$R(x_i) = \sum_{j \in K} d(x_i, x_j),$$

kjer sta x_i in x_j različna osebka ($i \neq j$), K je množica vseh osebkov in $d(x_i, x_j)$ predstavlja razdaljo. V poštev pridejo številne mere razdalje. Na primer: evklidska, manhattanska in kosinusna razdalja. Katero izberemo, je odvisno od narave problema. Algoritem FA običajno uporablja evklidsko razdaljo za d-dimenzionalne osebke.

$$d(x_i, x_j) = |f(x_i) - f(x_j)| = \sqrt{(x_{i_1} - x_{j_1})^2 + (x_{i_2} + x_{j_2})^2 + \ldots + (x_{i_d} + x_{j_d})^2}$$

Vsak osebek opremimo še z verjetnostjo izbire. Osebki, ki so bolj oddaljeni, imajo večjo verjetnost izbire. Na ta način preprečimo prezgodnjo konvergenco, saj se izogibamo osebkov iz gostih gruč. Za izračun verjetnosti uporabimo izbiranje z ruletnim kolesom [16].

$$p(x_i) = \frac{R(x_i)}{\sum_{j \in K} R(x_i)},$$

kjer je

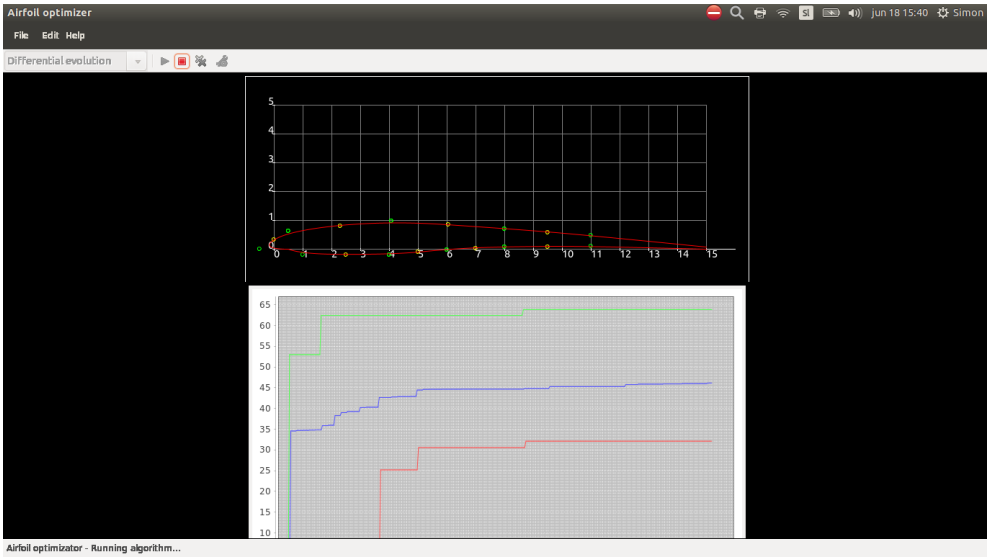
$$R(x_i) = \sum_{j \in K} d(x_i, x_j) = \sum_{j \in K} |x_i - x_j|$$

Poglavje 5

Zasnova programske rešitve

Naša programska rešitev za optimizacijo profilov letalskih kril je napisana v programskem jeziku Java. Airfoil Optimizer uporablja algortime diferencialne evolucije [13], metodo roja delcev [4] ter metoda ognjemeta [16]. Med razvojem je bila uporabljena objektno orientirana javanska knjižnica JMetal [11], ki ponuja vrsto implementacij enokriterijskih in večkriterijskih optimizacijskih algoritmov, ustreznih operatorjev in podpornih razredov. Algoritma DE in PSO iz knjižnice je bilo potrebno prilagoditi za potrebe optimizacije profilov. Algoritem FA ni del JMetal knjižnice, zato je ta bil implementiran samostojno. Programska rešitev je bila napisana po arhitekturnem vzorcu MVC (Model-View-Controller), ki loči uporabniški vmesnik (prezentacijo) od procesiranja oz. logike. Poleg tega je bil uporabljen tudi vzorec opazovalca (Observer design pattern). Pri MVC se model spreminja ob uporabnikovi interakciji z uporabniškim vmesnikom. V našem primeru se model spreminja tudi med izvajanjem optimizacijskih algoritmov (brez uporabnikove interakcije). Pri vzorcu opazovalca model dobi seznam razredov, ki predstavljajo dele uporabniškega vmesnika in, ob spremembi modelà, kličejo funkcijo, ki skrbi za ponoven izris uporabniškega vmesnika. Programsko rešitev je moč zaganjati iz ukazne vrstice brez prikazovanja uporabniškega vmesnika. S pomočjo argumentov iz ukazne vrstice lahko to funkcionalnost izkoristimo za avtomatizirano zaganjanje oz. testiranje programske rešitve.

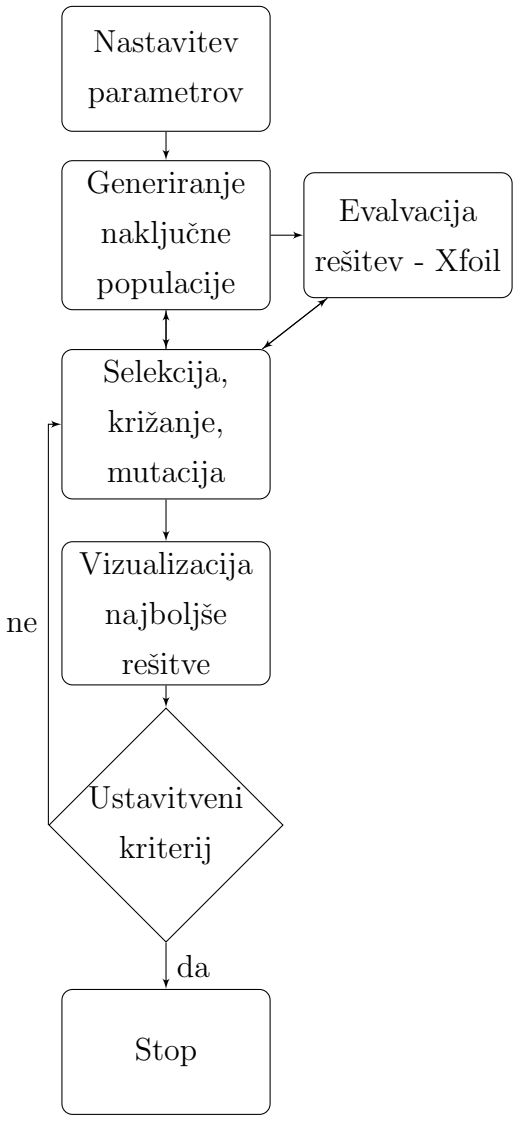
Na sliki 5.1 je prikazan uporabniški vmesnik programa Airfoil optimizer.



Slika 5.1: Uporabniški vmesnik.

Preden zaženemo optimizacijski algoritem je potrebno nastaviti parametre algoritma kot so: velikost populacije, verjetnost križanja ter mutacije. Primerne vrednosti parametrov je potrebno določiti s predhodnim testiranjem. V prvem koraku programska rešitev generira populacijo rešitev z naključnimi vrednostmi spremenljivk. Rešitev je del začetne populacije, v kolikor ima vrednost ocenitvene funkcije oz. razmerje L/D vsaj 40. S pomočjo spodnje meje odstranimo dele preiskovalnega prostora, ki niso obetavni. Sedaj nastopi glavni del algoritma. Algoritem DE omogoča več načinov izvajanja selekcije, križanja in mutacije. Uporabljen je bil način DE/rand/1/bin, saj velja za najuspešnejšega. Rand pomeni, da so kandidati za križanje izbrani naključno, število 1 pomeni, da križanje generira samo enega potomca in bin predstavlja način mutacije. Starša mutiramo tako, da na naključno določenih mestih zamenjamo vrednost spremenljivke z vrednostjo spremenljivke potomca. Za algoritem PSO smo uporabili globalno različico algoritma, algoritem FA pa uporablja princip elitizma, kjer je najboljši osebek vedno del nove populacije. Nato sledi evalvacija vseh potomcev s pomočjo XFoil programa. Trenutno

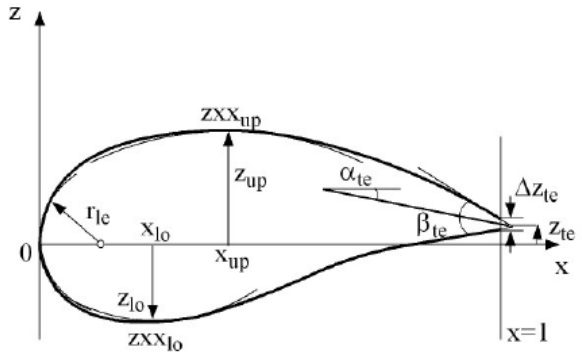
najboljšo rešitev tudi prikažemo. Na koncu sledi še sestava nove populacije. V populacijo nove generacije gredo vsi tisti potomci, ki so boljši od svojih staršev. V kolikor je potomec slabši od starša slednji preživi. To omogoča, da povprečna vrednost ocenitvene funkcije raste oz. ostane enaka. Postopek ponavljamo dokler se povprečna vrednost ocenitvene funkcije ne spreminja več. Na sliki 5.2 je prikazan postopek optimizacije.



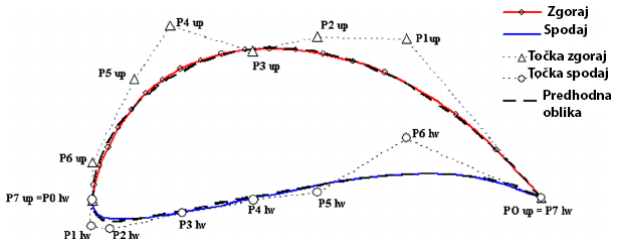
Slika 5.2: Diagram poteka programske rešitve.

5.1 Predstavitev profila

Eden najpomembnejših korakov pri implementaciji optimizacijskih algoritmov je določitev primerne predstavitve osebkov in primerne podatkovne strukture. Želimo imeti čim manj spremenljivk, a hkrati mora predstavitev omogočati generiranje široke palete profilov. Najbolj pogosta načina parametrizacije sta PARSEC in Bézierjeve krivulje. Prva metoda določa 11 geometričnih karakteristik profila, ki enolično določajo obliko profila. To so: polmer prednjega roba, debelina zgornje in spodnje polovice, lokacija največje ukrivljenosti za spodnjo in zgornjo polovico, ukrivljenost spodnje in zgornje polovice profila, lokacija zadnjega roba, velikost vrzeli pri zadnjem robu, smer zadnjega roba in velikost klina pri zadnjem robu (slika 5.3). V drugem primeru pa lahko uporabimo dve Bézierjevi krivulji kot prikazuje slika 5.4. Ena za zgornjo, druga za spodnjo polovico. Dobra lastnost PARSEC metode je, da so generirani profili vedno veljavne oblike, ampak ne omogočajo veliko fleksibilnosti. Po drugi strani Bézierjeve krivulje omogočajo veliko fleksibilnosti in kontrolo po poljubno velikih segmentih, ampak lahko se zgodi, da algoritem generira tako slab profil, da Xfoil program ne uspe preračunati kvalitete.



Slika 5.3: PARSEC parametrizacija [9].



Slika 5.4: Primer predstavitve profila z Bézierjevimi krivuljami [14].

Kljub temu smo uporabili dve Bézierjevi krivulji, ki imata po 7 kontrolnih točk. Vsaka se lahko pomika po obeh oseh znotraj določenega območja. Samo začetna in zadnja sta fiksni, da se dolžina profila ne spreminja (vsi profili so nastavljeni na dolžino 150 mm). Vsaka točka je shranjena v enodimenzionalnem polju, pri čemer je na m-tem mestu x in na $m+1$ mestu y komponenta točke. Skupno našo podatkovno strukturo sestavlja 28 komponent, vsaka ima svojo zgornjo in spodnjo mejo.

5.1.1 Bézierjeve krivulje

Odkril jih je Paul de Casteljau (po njem se imenuje de Casteljau algoritem), Pierre Bézier jih je uporabil pri oblikovanju avtomobilov v tovarni Renault. Danes jih uporabljamo v računalniški grafiki in tipografiji.
Bézierjeva krivulja je parametrična krivulja za risanje gladkih krivulj. Sestavljena je iz več Bersteinovih polinomov [18]. Krivulja je definirana z množico kontrolnih točk $(P_0, ...P_n)$ [17]. Bersteinovi polinomi so definirani kot:

$$b_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i, i = 0, \ldots, n.$$

Po zgornji definiciji zapišimo Bézierjeve krivulje:

- prve stopnje
$$B(t) = (1-t)P_0 + tP_1, 0 \leq t \leq 1,$$
- druge stopnje
$$B(t) = (1-t)^2P_0 + 2(1-t)tP_1 + t^2P_2, 0 \leq t \leq 1,$$

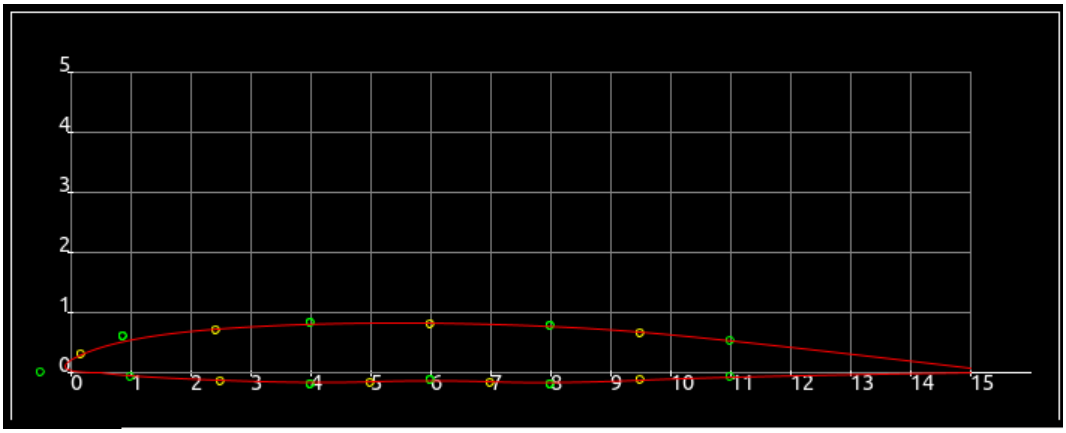
- tretje stopnje

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3, 0 < t < 1.$$

Splošna oblika Bézierjeve krivulje stopnje n, kjer P_i predstavlja i-to kontrolno točko je:

$$B(t) = \sum_{i=0}^{n} b_{i,n}(t) P_i.$$

5.2 Diskretizacija profila



Slika 5.5: Vizualizacija profila.

V poglavju o predstavitvi osebkov smo ugotovili, da je profil predstavljen kot sosledje koordinat kontrolnih točk, ki definirajo Bézierjevo krivuljo. Če želimo rešitev oceniti s pomočjo Xfoil programa, mu moramo posredovati koordinate točk na krivulji. Torej moramo krivuljo diskretizirati. V našem primeru imamo opravka z Bézierjevo krivuljo šeste stopnje. Da bi se izognili računanju s polinomi višjih stopenj, lahko krivuljo poenostavimo z linearno kombinacijo kvadratnih Bézierjevih krivulj. Tako dobimo šest kvadratičnih Bézerjevih krivulj, ki so lažje izračunljive, saj imamo opravka s polinomi druge stopnje. V spodnjem algoritmu so točke predstavljene kot objekti tipa

Point, kjer hranimo x in y koordinate. Krivulja je razdeljena na intervale dolžine 0,05, x in y komponento točke pa dobimo z uporabo zgoraj navedene formule za Bézierjevo krivuljo druge stopnje. Programska rešitev omogoča tudi vizualizacijo najboljšega profila v populaciji. Izvirno krivuljo smo primorani razdeliti na več krivulj nižje stopnje, ker v programskem jeziku java ne obstaja metoda, ki bi lahko izrisovala Bézierjeve krivulje višjih stopenj. Java ima implementirane metode za risanje Bézierjeve krivulje druge in tretje stopnje. Krivuljo razdelimo tako, da izračunamo sredinsko točko med drugo in tretjo kontrolno točko. Sedaj imamo Bézierjovo krivuljo druge stopnje, ki poteka med začetno, drugo in sredinsko točko. Nato sredinska točko postane nova začetna točka in ponovno izračunamo sredinsko točko med naslednjima dvema točkama. S pomočjo sredinske točke poskrbimo, da je spoj krivulj gladek, krivulja je zvezna. Prednost take delitve je tudi večja kontrola krivulje po segmentih.

Input: inputArray - seznam objektov tipa Point

Output: outputArray - seznam točk na krivulji

1: s ← array[0] - začetna točka
2: $outputArray \leftarrow s$
3: len ← array.length - dolžina seznama
4: **for** $i = 1$ to $len - 2$ **do**
5: Pridobi prvo in drugo kontrolno točko
6: $cp_1 \leftarrow$ inputArray[i]
7: $cp_2 \leftarrow$ inputArray[$i + 1$]
8: Izračunaj sredinsko točko med cp1 in cp2
9: midPoint.x ← (cp_2.x $-cp_1$.x)/2
10: midPoint.y ← (cp_2.y $-cp_1$.y)/2
11: $outputArray \leftarrow cp_1$
12: $outputArray \leftarrow midPoint$
13: Izračunamo točke na krivulji
14: **for** $t = 0.0$ to 1.0 **do**
15: $newPoint.x = (1-t)^2 s + 2(1-t)tcp_1 + t^2 MidPoint.x$
16: $newPoint.y = (1-t)^2 s + 2(1-t)tcp_1 + t^2 MidPoint.y$
17: $outputArray \leftarrow newPoint$
18: $t \leftarrow t + 0.05$
19: **end for**
20: Sredinska točka je sedaj nova začetna točka
21: $s \leftarrow$ midPoint
22: **end for**
23: Še zadnja krivulja
24: $cp_1 \leftarrow$ array[len-2]
25: $cp_2 \leftarrow$ array[len-1]
26: **for** $t = 0$ to 1 **do**
27: $newPoint.x \leftarrow (1-t)^2 s + 2(1-t)tcp_1.x + t^2 cp_2.x$
28: $newPoint.y \leftarrow (1-t)^2 s + 2(1-t)tcp_1.y + t^2 cp_2.y$
29: $outputArray \leftarrow newPoint$
30: $t \leftarrow t + 0.05$
31: **end for**
32: **return** outputArray

Slika 5.6: Postopek diskretizacije profila.

Ukaz	Pomen
PLOP	Meni za grafiko
G	Ne prikazuj graf. vmesnika
	Nova vrstica - pojdi nazaj
LOAD /airfoil.dat	Naloži profil
testAirfoil	ime profila
PANE	Zgladij profil
OPER	Oper meni
VISC 1.0e5	Računamo za viskozni tok, gostota kapljevine
Re 20408.685	Nastavimo Reynoldsovo število
M 0.0058	Hitrost toka v Mach-ih
PACC	Ukaz za shranjevanje rezultatov
/polar.txt	Ime datoteke
	Nova vrstica - pojdi nazaj
ALFA 2.5	Nastavimo upadni kot
!	Če ne konvergira naredimo še eno iteracijo
PACC	Konec shranjevanja
QUIT	Zakjučimo

Tabela 5.1: Seznam ukazov v vhodni datoteki.

5.3 Ocenjevanje rešitev

Optimizacijski algoritem v vsaki iteraciji preveri kakovost rešitev trenutne populacije. Ocena se poda na osnovi t.i. ocenitvene funkcije. Ta nam pove, kako daleč smo od popolne rešitve. Cilj optimizacijskega algoritma je maksimizacija te funkcije. Izračun karakteristik profila je dokaj zapleteno opravilo, zato je to delo prepuščeno odprtokodnemu programu Xfoil. Xfoil je interaktivni konzolni program za oblikovanje in analizo profilov. Xfoil omogoča izvajanje s pomočjo vhodne datoteke z ukazi. To je prikladno, saj lahko izvajanje avtomatiziramo. Za avtomatizacijo procesa evalvacije je potrebno sestaviti vhodno datoteko (tabela 5.1), ki ima nanizane Xfoil ukaze ter datoteko z x in y koordinatami. Nato v skriptnem jeziku lupine operacijskega sistema napišemo kratko skripto, ki zažene Xfoil program in na vhod poda datoteko z ukazi. Slednjo lahko brez težav programsko zaženemo.

Xfoil preračunane vrednosti shrani v izhodno datoteko. Iz nje preberemo vrednosti koeficienta vzgona in upora. Ti dve vrednosti lahko združimo v razmerje L/D (lift-to-drag ratio). To je naša ocenitvena funkcija in naš cilj je maksimizirati to vrednost.

Koeficient vzgona C_l za presek krila temelji na dvodimenzionalnem zračnem toku po krilu neskončne dolžine (zanemarimo dejavnike vzdolž krila končne dolžine). Definiran je kot razmerje med silo vzgona l (na enoto dolžine razpona) ter produktom dinamičnega pritiska in dolžine profila c. Dinamični pritisk je definiran kot produkt med gostoto kapljevine ρ in kvadratom hitrosti v [21].

$$C_l = \frac{l}{\frac{1}{2}\rho v^2 c}$$

Koeficient upora je definiran kot razmerje med silo upora in produktom med dinamičnim pritiskom in površino. V našem primeru A predstavlja površino prednjega dela profila [20].

$$C_d = \frac{F_d}{A\frac{1}{2}\rho v^2}$$

Oba koeficienta lahko zapišemo kot razmerje L/D:

$$L/D = \frac{C_l}{C_d}.$$

Programu Xfoil je potrebno določiti parametre, ki jih uporablja pri preračunavanju karakteristik profila. To so Reynoldsovo in Machovo število ter vpadni kot profila. Parametri so bili nastavljeni tako, da odražajo pogoje in režim letenja letalskih prostoletečih modelov kategorije F1A. Za njih je značilna nizka hitrost letenja (Mach: 0,0058), višina letenja pa okoli 85 m (povprečno), kar privede do nizkega Reynoldsovega števila (Re: 46000). Vpadni kot je nastavljen na 2,50 stopinj.

Poglavje 6

Rezultati

6.1 Analiza algoritmov

V tem razdelku so predstavljeni rezultati testiranj implementiranih algoritmov. Za vsak algoritem smo najprej poiskali najboljšo kombinacijo parametrov, sledi statistična analiza razmerja L/D (zaustavitveni pogoj je konvergenca) ter števila potrebnih iteracij. Pridobljene podatke uporabimo pri testiranju konvergence razmerja L/D ter koeficienta vzgona in upora.

Stohastični algoritmi imajo vrsto parametrov, ki jih je potrebno pravilno nastaviti, da dobimo dobre rezultate. Pravilna nastavitev parametrov je odvisna od problema in zahteva kar nekaj sistematičnega preizkušanja. Za potrebe preizkušanja je bilo potrebno programsko rešitev dodelati tako, da bi lahko avtomatizirano preizkušali nastavitve parametrov. To različico programa lahko zaganjamo v tekstovnem načinu iz ukazne vrstice in s pomočjo argumentov določimo parametre algoritma. Napisali smo še skripto v jeziku ukazne lupine, ki sistematično zaganja aplikacijo z različnimi argumenti. Vsi trije algoritmi so stohastični, kar pomeni da z vsakim zagonom dobimo drugačne rešitve. Zato je potrebno narediti več zagonov z istimi parametri, da lahko določimo uspešnost nastavitev.

Diferencialna evolucija

V Tabeli 6.1 je prikazano razmerje L/D za vse možne kombinacije vrednosti parametrov pri velikosti populacije 10. Najboljša kombinacija parametrov je par vrednosti 0,4 in 0,8. Ta vrednost parametrov je bila uporabljena pri statistični analizi razmerja L/D, pri kateri smo uporabili konvergenco kot zaustavitveni pogoj. Rezultati so vidni v tabeli 6.2. V tabeli 6.3 so predstavljeni rezultati analize potrebnega števila iteracij za popolno konvergenco. Algoritem konvergira, ko se povprečna vrednost kriterijske funkcije ne spreminja več. Slika 6.1 prikazuje konvergenco kriterijske funkcije algoritma v odvisnosti od števila iteracij za 20 zagonov (slika 6.1) in en sam zagon (slika 6.2). Sliki 6.3 in 6.4 prikazujeta konvergenco keoficienta vzgona in koeficienta upora.

Verjetnost križanja	Verjetnost mutacije	razmerje L/D
0,2	0,2	59.47
	0,4	57.64
	0,6	58.58
	0,8	62.44
	0,2	59.88
0,4	0,4	60.77
	0,6	62.70
	0,8	64.66
	0,2	58.96
0,6	0,4	62.13
	0,6	62.27
	0,8	58.58
	0,2	62.67
0,8	0,4	63.76
	0,6	63.94
	0,8	62.18

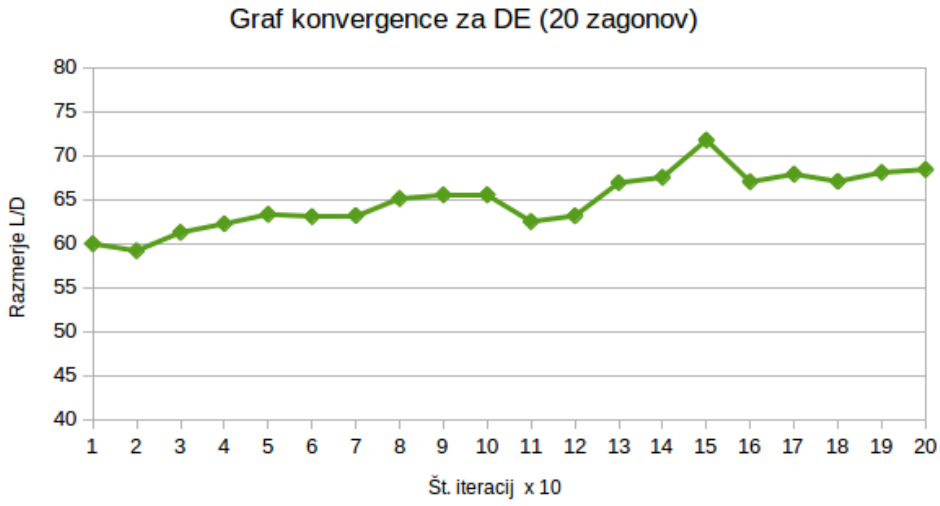
Tabela 6.1: Testiranje parametrov za algoritem DE.

Minimalna vrednost	53,57
Mediana	61,58
Maksimalna vrednost	74,51
Povprečje	61,12
Deviacija	5,88

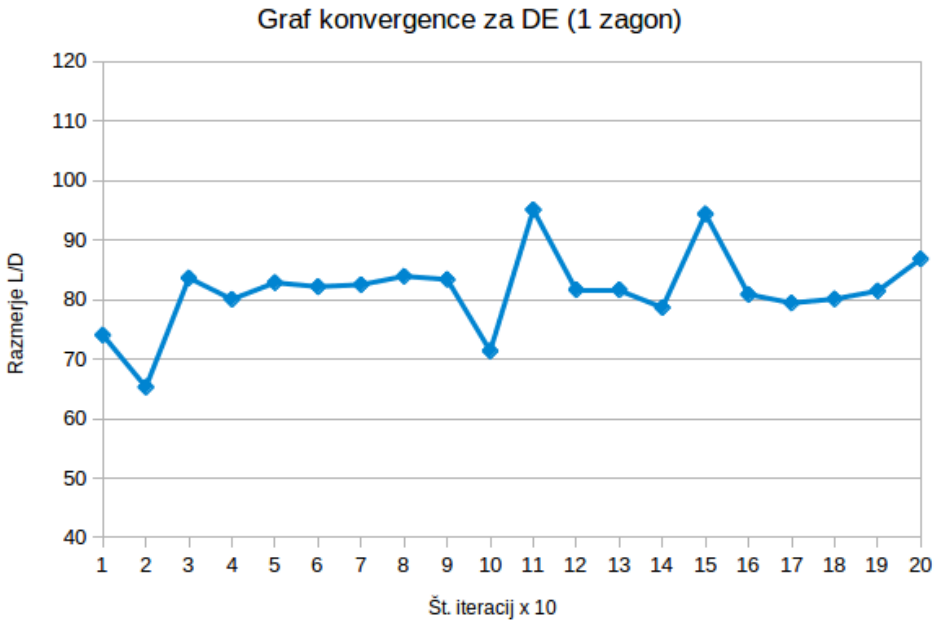
Tabela 6.2: Razmerje L/D po 20 zagonih za algoritem DE.

Minimalna vrednost	57
Mediana	134
Maksimalna vrednost	190
Povprečje	125
Deviacija	41

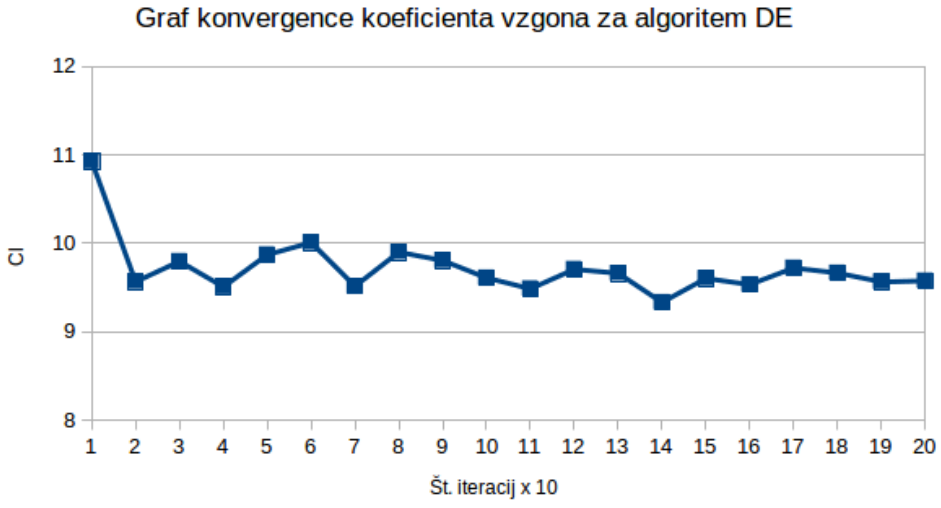
Tabela 6.3: Število potrebnih iteracij za algoritem DE.



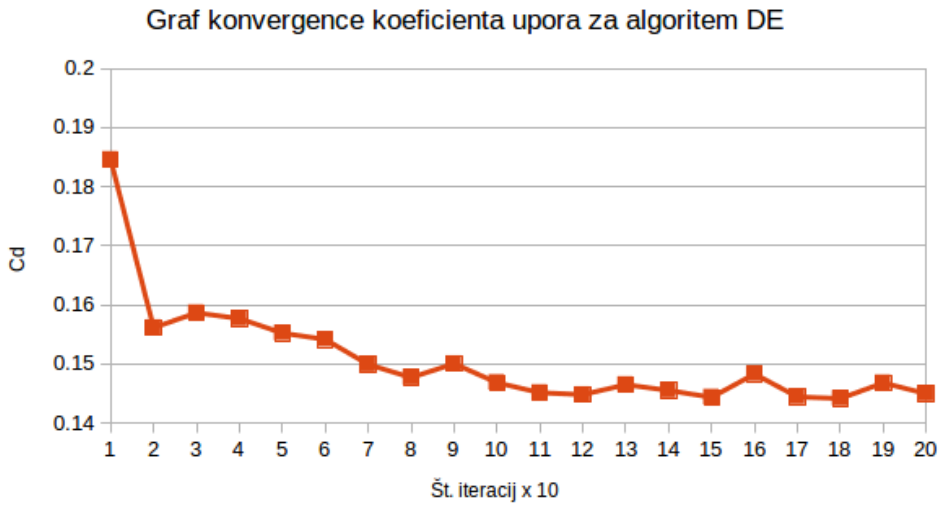
Slika 6.1: Graf kriterijske funkcije v odvisnosti od števila iteracij (povprečje 20 zagonov).



Slika 6.2: Graf kriterijske funkcije v odvisnosti od števila iteracij za en zagon.



Slika 6.3: Graf koeficienta vzgona v odvisnosti od števila iteracij (20 zagonov).



Slika 6.4: Graf koeficienta upora v odvisnosti od števila iteracij (20 zagonov).

Metoda roja delcev

V tabeli 6.4 so prikazane vrednosti kriterijske funkcije za kombinacijo vrednsoti parametrov. Pri nadaljnjem testiranju je bila uporabljena kombinacija vrednosti 0,9 (kognitivna komponenta) in 0,3 (socialna komponenta) pri velikosti populacije 10. V tabelah 6.5 in 6.6 pa statistični podatki za kriterijsko funkcijo in število potrebnih iteracij. Sledijo grafi, ki prikazujejo konvergenco algoritma (slika 6.5 in slika 6.6) ter koeficienta vzgona (slika 6.7) in upora (slika 6.8).

Kognitivna komponenta	Socialna komponenta	Razmerje L/D
0,1	0,1	70,63
	0,3	69,85
	0,5	66,84
	0,7	59,88
	0,9	57,82
	0,1	73,01
0,3	0,3	69,26
	0,5	64,38
	0,7	60,36
	0,9	62,33
	0,1	78,19
0,5	0,3	69,84
	0,5	71,33
	0,7	60,15
	0,9	58,14
	0,1	74,15
0,7	0,3	64,89
	0,5	64,65
	0,7	61,39
	0,9	57,44
	0,1	69,11
0,9	0,3	77,70
	0,5	67,12
	0,7	62,50
	0,9	60,32

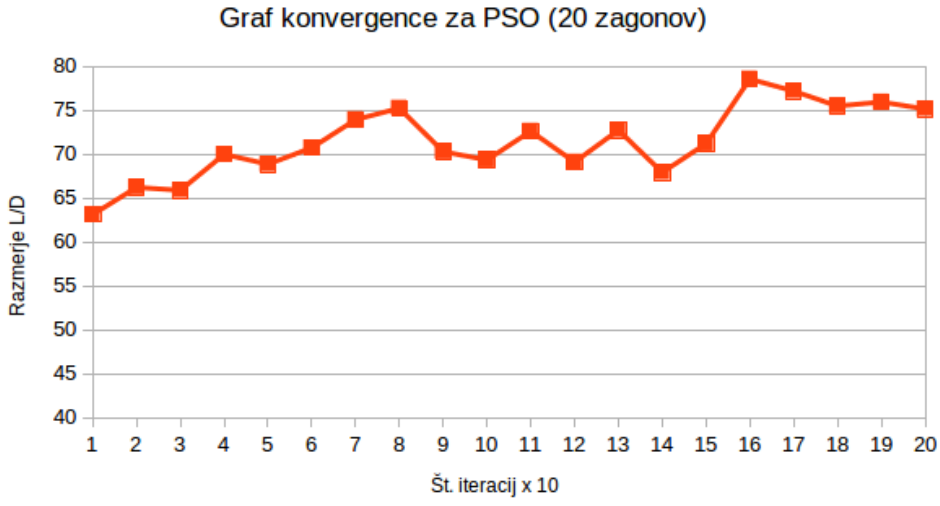
Tabela 6.4: Testiranje parametrov za algoritem PSO.

Minimalna vrednost	62,64
Mediana	69,98
Maksimalna vrednost	89,92
Povprečje	71,81
Deviacija	7,65

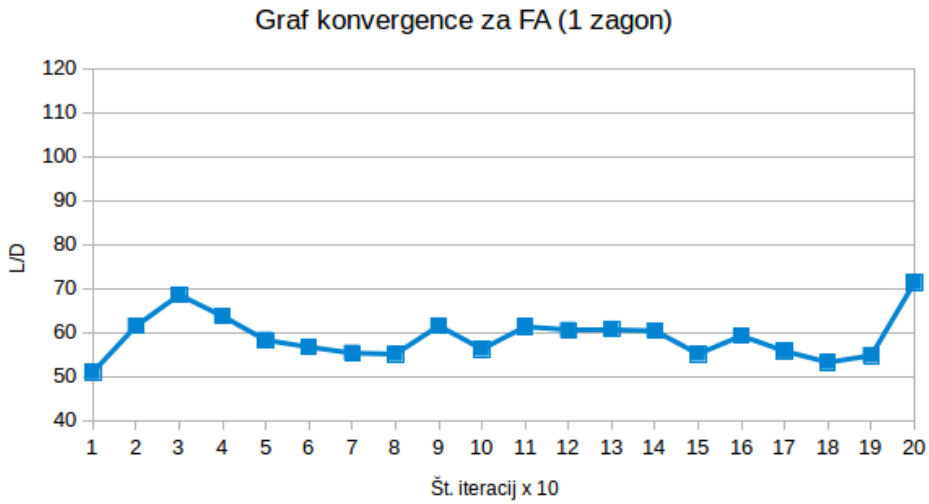
Tabela 6.5: Razmerje L/D po 20 zagonih za algoritem PSO.

Minimalna vrednost	240
Mediana	134
Maksimalna vrednost	315
Povprečje	273
Deviacija	17

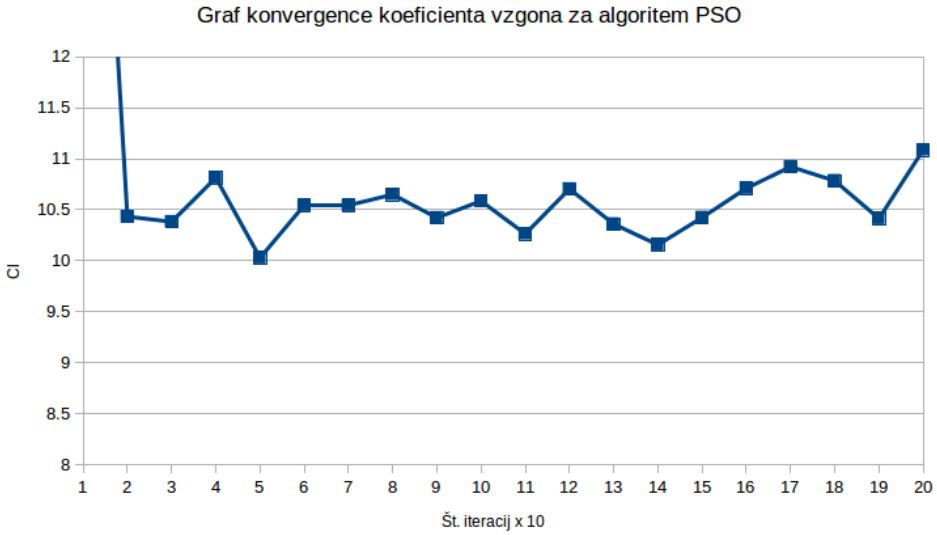
Tabela 6.6: Število potrebnih iteracij za algoritem PSO.



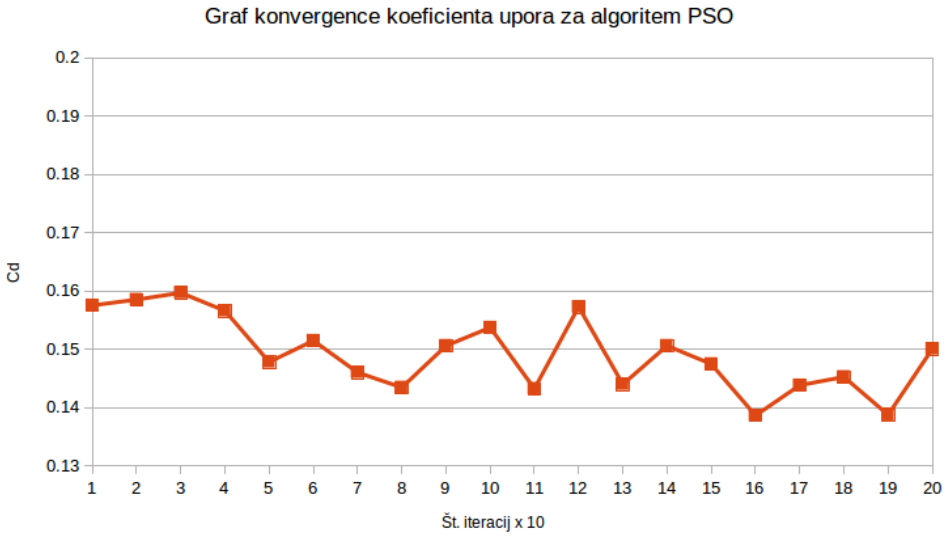
Slika 6.5: Graf kriterijske funkcije v odvisnosti od števila iteracij (povprečje 20 zagonov).



Slika 6.6: Graf kriterijske funkcije v odvisnosti od števila iteracij za en zagon.



Slika 6.7: Graf koeficienta vzgona v odvisnosti od števila iteracij (povprečje 20 zagonov).



Slika 6.8: Graf koeficienta upora v odvisnosti od števila iteracij (povprečje 20 zagonov).

Metoda ognjemeta

Pri metodi ognjemeta sta bila testirana parametra število ognjemetov (velikost populacije) in amplituda eksplozije. Število isker, ki se generirajo pri eksploziji smo omejili med 2 in 15. V tabeli 6.7 je prikazano testiranje parametrov. V tabelah 6.8 in 6.9 so prikazani statistični podatki glede razmerja L/D in števila potrebnih iteracij. Sledijo grafi konvergence kriterijske funkcije (slika 6.9 in slika 6.10) ter koeficienta vzgona (slika 6.11) in upora (slika 6.12).

Število ognjemetov	Amplituda eksplozije	Razmerje L/D
5	1	49.5424508725
	2	47.904257868
	3	47.779510608
15	1	48.9376710264
	2	49.3750937152
	3	48.3373585113
20	1	48.1516416974
	2	44.7974921378
	3	44.7753264824

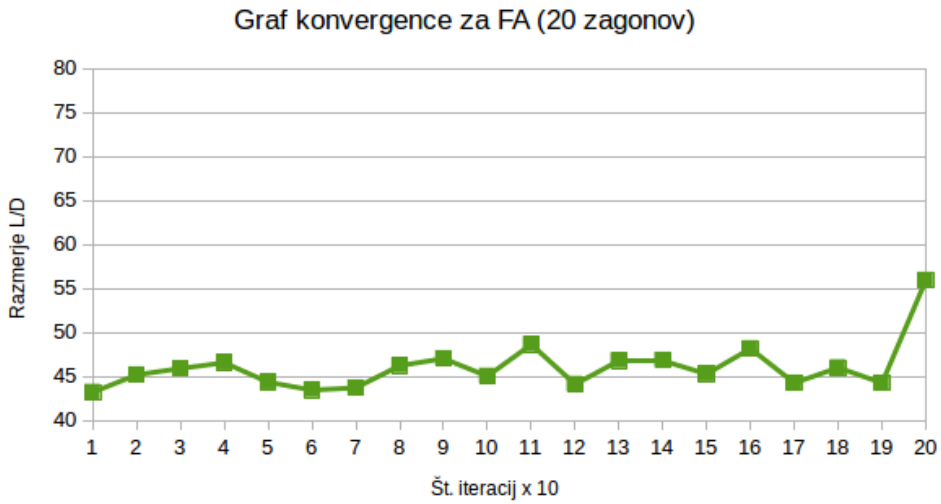
Tabela 6.7: Testiranje parametrov za algoritem FA.

Minimalna vrednost	35,37
Mediana	69,21
Maksimalna vrednost	95,74
Povprečje	66.92
Deviacija	14.98

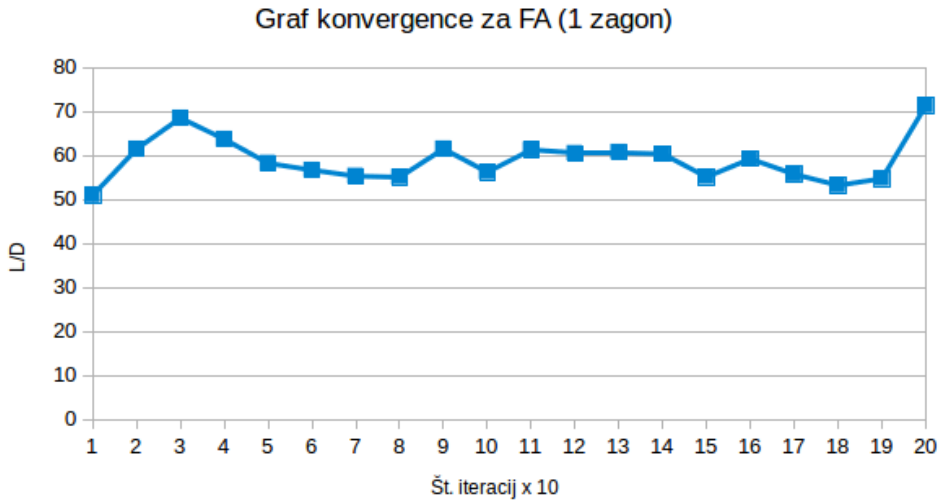
Tabela 6.8: Razmerje L/D po 20 zagonih za algoritem FA.

Minimalna vrednost	157
Mediana	229,5
Maksimalna vrednost	632
Povprečje	262,28
Deviacija	111,16

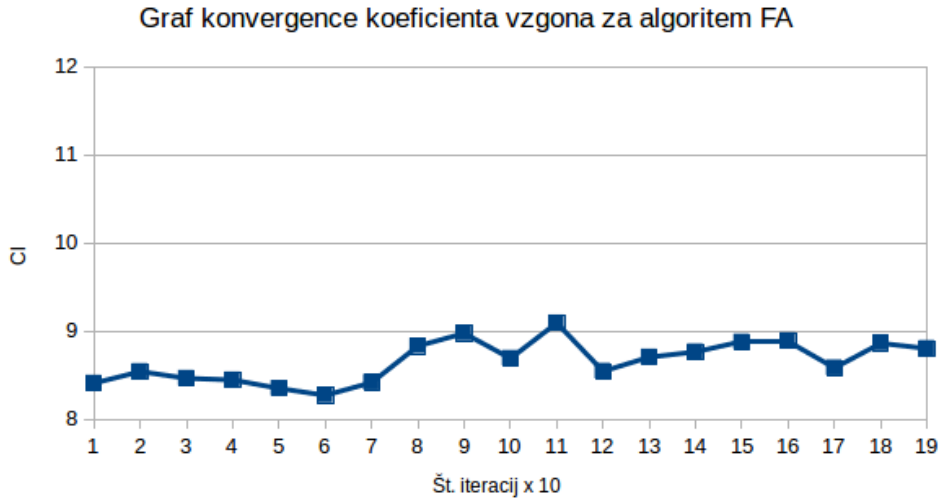
Tabela 6.9: Število potrebnih iteracij za algoritem FA.



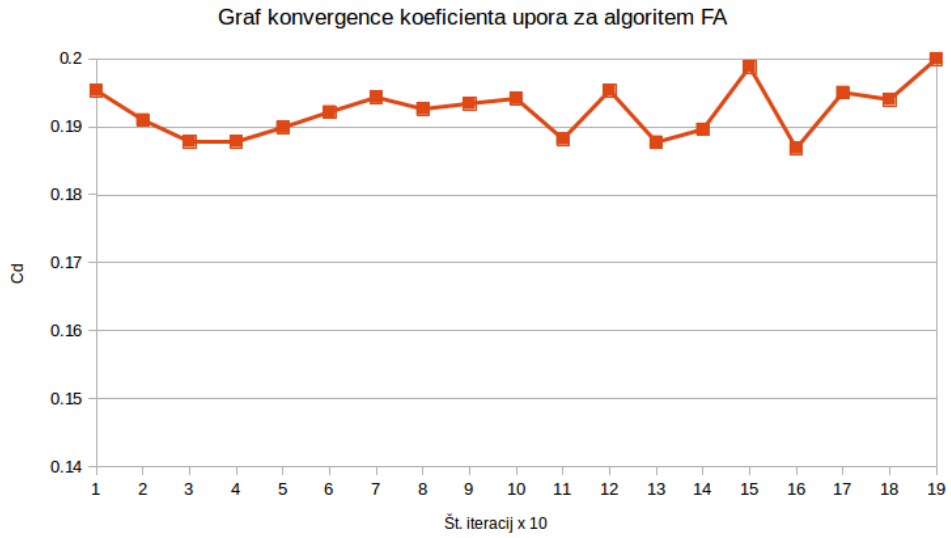
Slika 6.9: Graf kriterijske funkcije v odvisnosti od števila iteracij (povprečje 20 zagonov).



Slika 6.10: Graf kriterijske funkcije v odvisnosti od števila iteracij za en zagon.

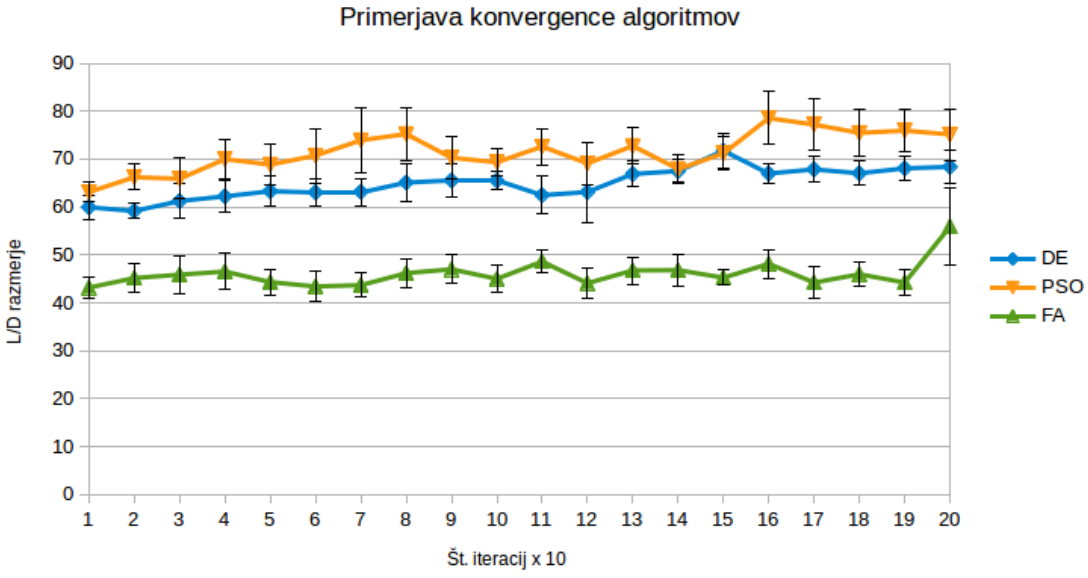


Slika 6.11: Graf koeficienta vzgona v odvisnosti od števila iteracij (povprečje 20 zagonov).

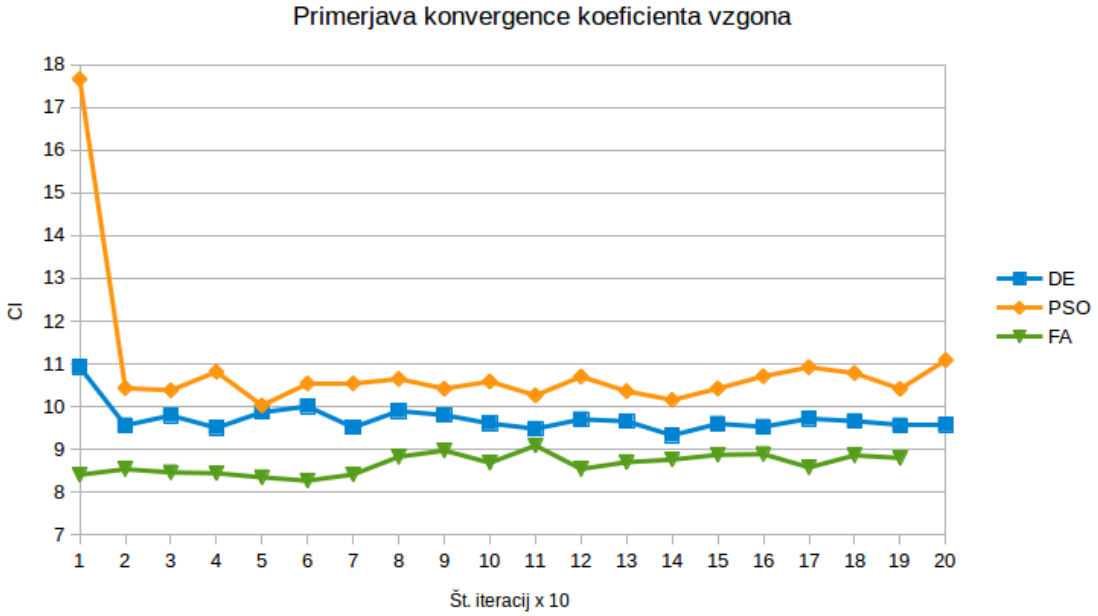


Slika 6.12: Graf koeficienta upora v odvisnosti od števila iteracij (povprečje 20 zagonov).

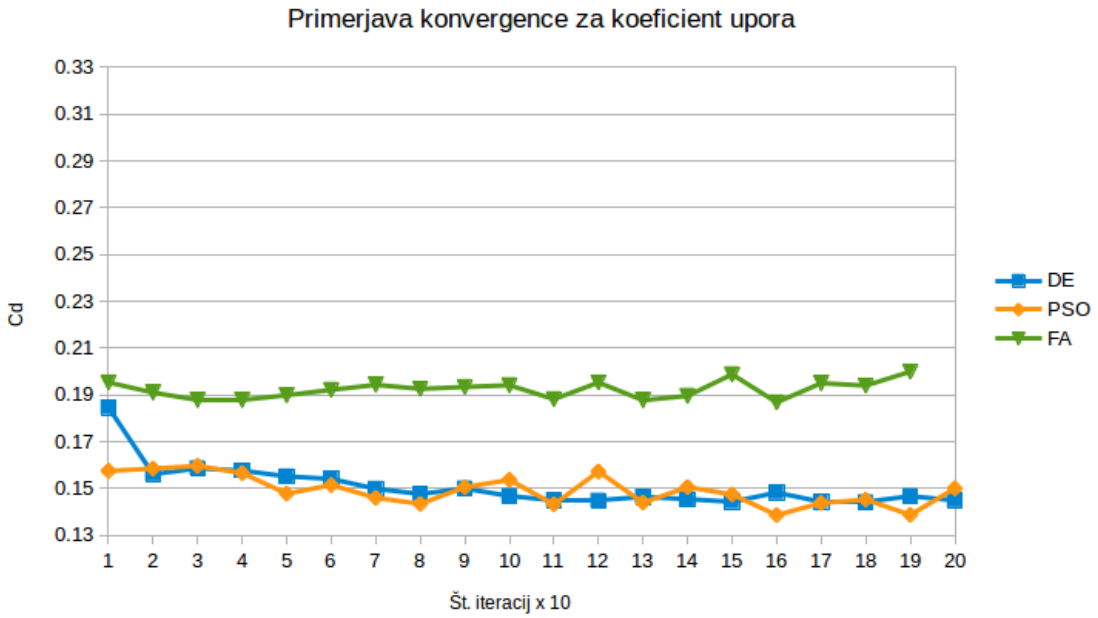
Na slikah 6.13, 6.14, 6.15 primerjamo konvergenco kriterijske funkcije, koeficienta vzgona in koeficienta upora za vse tri algoritme. Iz slike 6.13 je vidno, da se je algoritem FA zanesljivo najslabše odrezal. Vzrok za slabše performanse gre pripisati dejstvu, da algoritem generira profile z daleč največjim koeficientom upora (slika 6.15). Visoke vrednosti koeficienta upora (in velika odstopanja na sliki 6.11 in 6.12) nastanejo, ker je algoritem FA večkrat generiral profile, ki so povsem neprimerni. Poleg tega algoritem FA generira profile z najmanjšim koeficientom vzgona (slika 6.14). Algoritem PSO konvergira k večjim vrednostim kriterijske funkcije kot algoritem DE, čeprav generirata profile s primerljiv koeficientom upora. Slika 6.13 prikazuje odstopanje kriterijske funkcije. S 95% gotovostjo lahko trdimo, da se prava vrednost nahaja v danem intervalu. Zaradi velikih odklonov v vrednosti kriterijske funkcije težko rečemo, da je algoritem PSO bistveno boljši od algoritma DE. V nekaterih točkah se odstopanja algoritma PSO prekrivajo z odstopanji algoritma DE. To pomeni, da je v teh točkah algoritem PSO zgolj po naključju boljši. Teoretično sta si algoritma podobna.



Slika 6.13: Primerjava konvergence algoritmov za razmerje L/D.



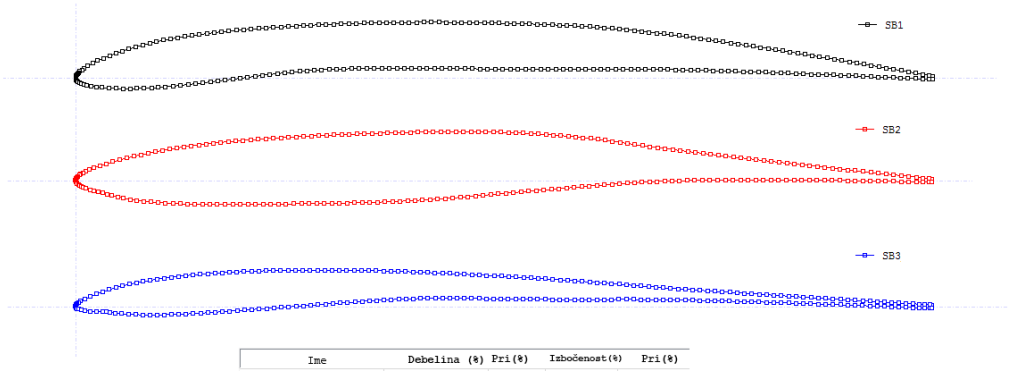
Slika 6.14: Primerjava konvergence koeficienta vzgona.



Slika 6.15: Primerjava konvergence koeficienta upora.

6.2 Analiza profilov

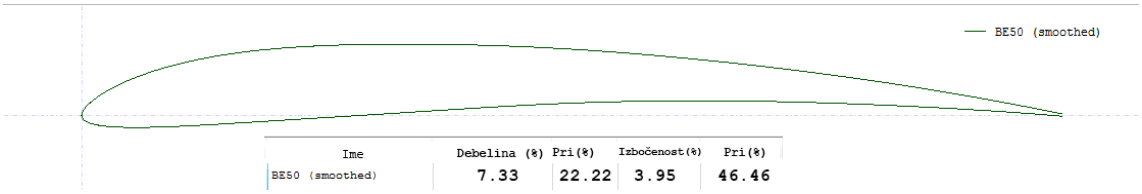
Da bi lahko odgovorili na vprašanje, ki smo si ga na začetku zastavili, moramo podrobneje analizirati še dobljene profile. Iz nabora generiranih profilov so bili izbrani tisti profili, ki se po obliki najbolj približujejo trenutnim LDA profilom. Izbrani so bili trije profili, ki so poimenovani SB1, SB2 in SB3. SB2 in SB3 sta bila generirana z algoritmom DE, medtem ko SB1 pa z algoritmom PSO. Algoritem FA ni uspel generirati oblike profila, ki bi vsaj na videz bila primerna za let. Na sliki 6.16 so prikazani vsi trije izbrani profili s podatki o debelini, točki največje debeline, izbočenosti in točko največje izbočenosti. Izbrane profile bomo primerjali z LDA profilom "Verbitsky BE50 (smoothed)" (v nadaljevanju: BE50), ki trenutno velja za enega najuspešnejih profilov (slika 6.17).



Ime	Debelina (%)	Pri(%)	Izbočenost(%)	Pri(%)
SB1	5.68	15.15	3.77	88.38
SB2	8.03	35.35	2.16	66.57
SB3	4.55	16.16	2.49	88.38

Slika 6.16: Trije najobetavnejši profili.

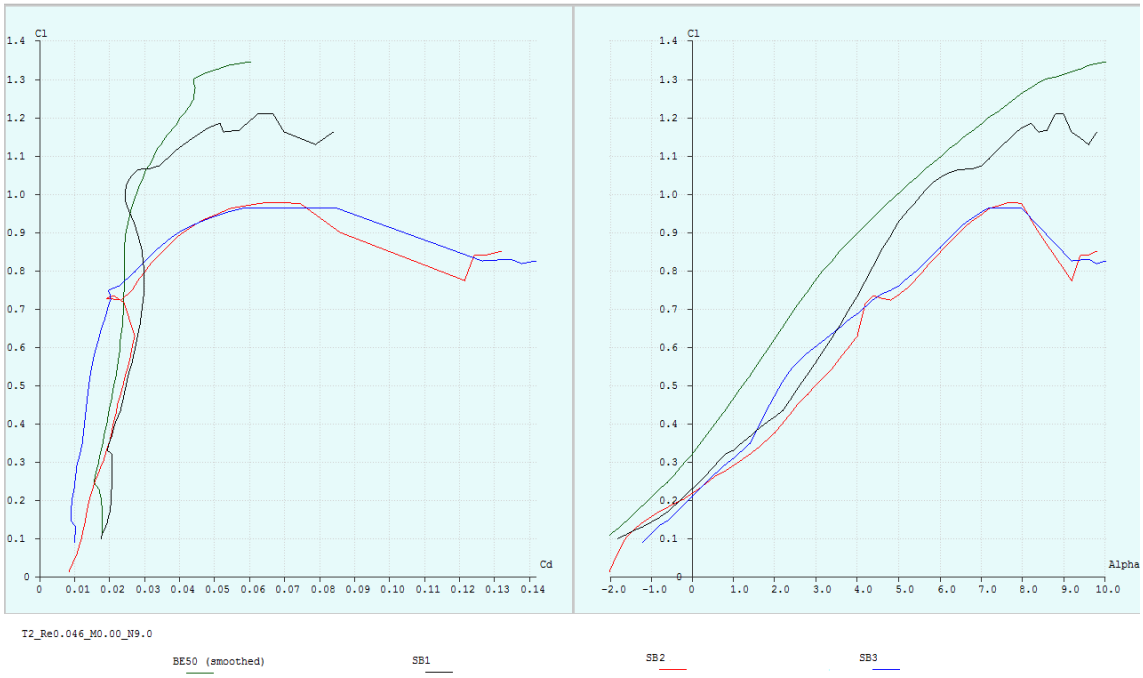
Podrobno analizo profilov opravimo s programom XFLR5 [1], ki v ozadju uporablja konzolni program Xfoil. XFLR5 ima, za razliko od Xfoil, uporabniku prijazen uporabniški vmesnik, kar omogoča hitrejše delo, poleg tega omogoča še analizo in vizualizacijo celotnih kril v 3D prostoru. Analizirali bomo dva aspekta letenja modelov kategorije F1A: planiranje (vodoravni let) in vzpenjanje (vertikalni let).



Slika 6.17: Trenutno najuspešnejši LDA profil Verbitsky BE50 (smoothed).

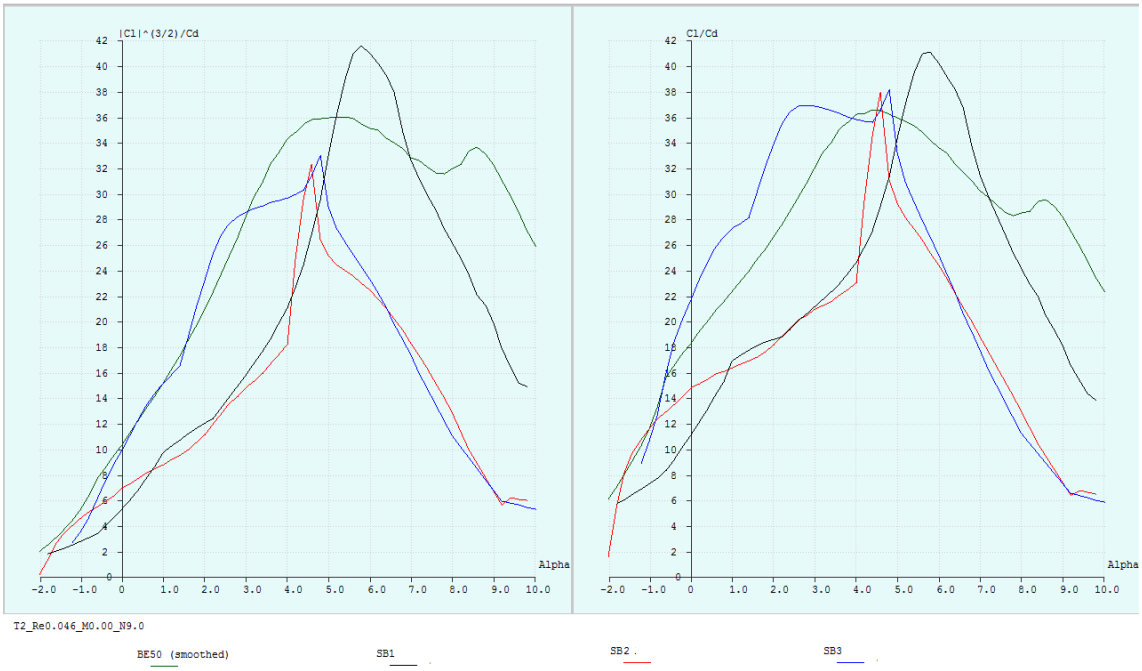
Planiranje

V programu XFLR5 je potrebno primerno nastaviti parametre, ki predstavljajo planiranje oz. vodoravni let modela. V programu nastavimo testiranje tipa 2 (fiksni vzgon), saj je v vodoravnem letu sila vzgona enaka teži modela in je zato konstantna. Spreminjata se hitrost in vpadni kot krila. Reynoldsovo število nastavimo na tipično vrednost za modele kategorije F1A, tj. 46000. Na sliki 6.18 (levo) je prikazan graf koeficienta vzgona v odvisnosti od koeficienta upora pri različnih vpadnih kotih profila. Ta nam pove, koliko upora proizvede profil pri danem koeficientu vzgona. Želimo si, da je graf pomaknjen čimbolj v levo. Profili so bili analizirani pri vpadnih kotih z intervala med -2 in 10 stopinj s korakom 0.2 stopinj. Iz grafa profila SB1 so vidne "stopničke", ki predstavljajo prehode iz laminarnega v turbuletni tok. Pri profilu BE50 je prisotna samo ena taka "stopnička". Če narišemo tangente, ki izhajajo v koordinatnem izhodišču, dobimo vrednost koeficienta vzgona pri najmanjši možni vrednosti koeficienta upora oz. razmerje L/D. SB1 je v tem pogledu boljši. Na grafu koeficienta vzgona v odvisnosti od vpadnega kota (desno) lahko odčitamo pod kolikšnim vpadnim kotom mora profil leteti, da doseže prej dobljeni koeficient vzgona. Iz tega grafa lahko tudi vidimo, kolikšen je maksimalni koeficient vzgona, preden se modelu poruši vzgon (nenadno padanje koeficienta vzgona) in pri kolikšnem vpadnem kotu se to zgodi. V tem pogledu je boljši BE50.
Najpomembnejši graf je na sliki 6.19. Prikazan je graf $\frac{|Cl|^{3/2}}{Cd}$ v odvisnosti od vpadnega kota. V kategoriji F1A je pomemben čas letenja. Da bi model letel kar največ časa, mora leteti s hitrostjo, pri kateri je propadanje modela



Slika 6.18: Graf koeficienta vzgona v odvisnoti od koeficienta upora (levo) in graf koeficienta vzgona v odvisnosti od vpadnega kota profila (desno).

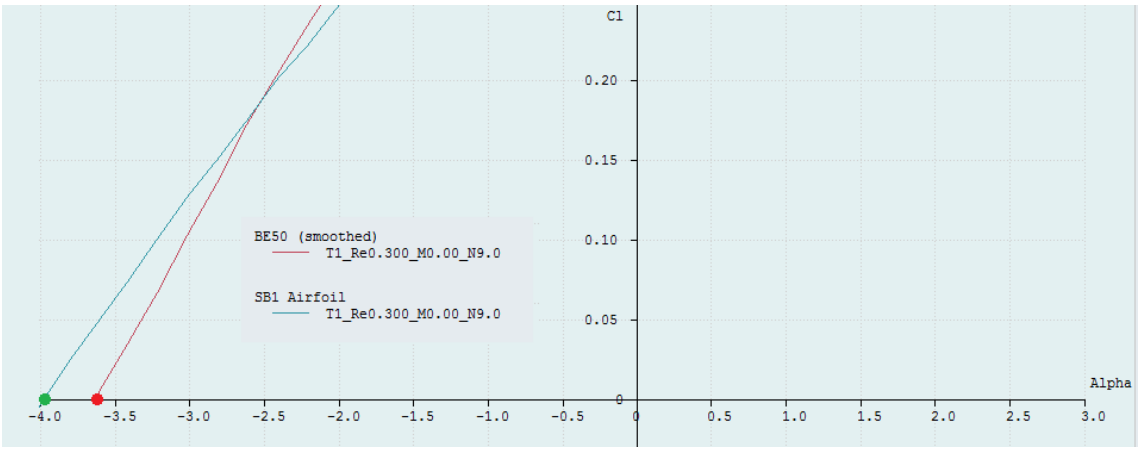
najmanjše (ang.: minimum-sink speed). Hitrost, pri kateri je propadanje najmanjše, se nahaja v točki, kjer je vrednost $\frac{|Cl|^{3/2}}{Cd}$ največja. To je t.i. power-factor. Želimo si, da je vrh grafa čim širši, kar pomeni, da model lahko leti s široko paleto vpadnih kotov, preden se hitrost padanja drastično poveča. Iz grafa je razvidno, da ima SB1 precej višji vrh kot BE50, ampak zaradi ozkega vrha bi model ob vsaki spremebi vpadnega kota letel z veliko večjim propadanjem. BE50 je kljub večjemu propadanju stabilnejši (širok vrh), kar pomeni, da bi bil primeren za letenje v turbulentnih in termičnih razmerah. SB1 bi lahko letel samo v zelo mirnih razmerah. Desni graf na sliki 6.19 pa prikazuje stabilnost modela v fazi kroženja (ko je model pripet na vrvico). Širši vrh grafa pomeni, da je model bolj vodljiv.



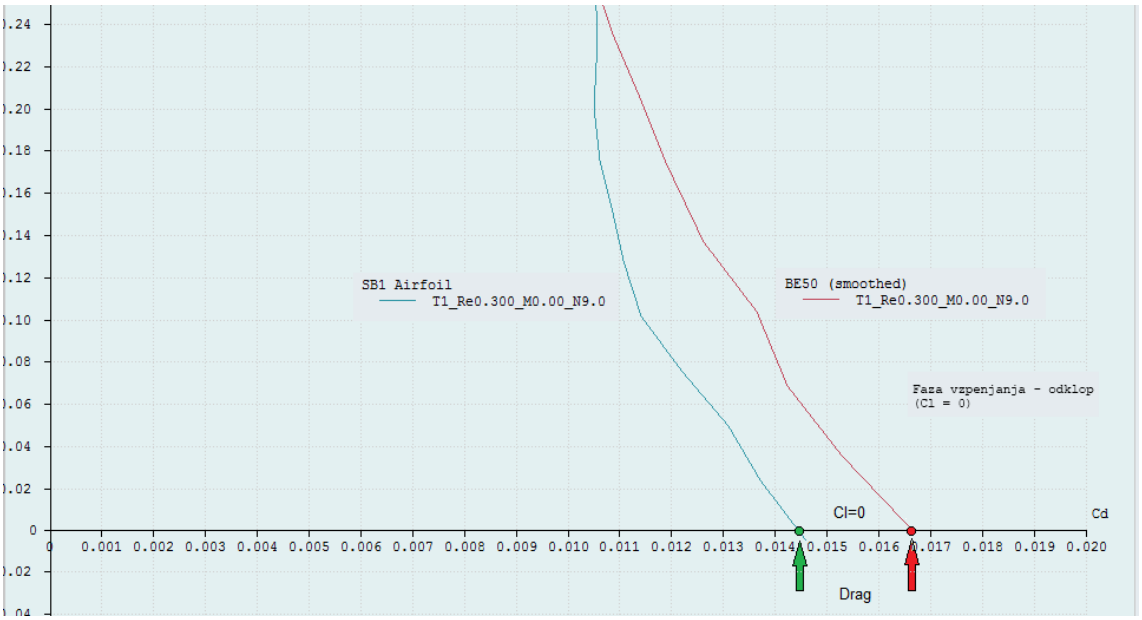
Slika 6.19: Graf $\frac{|Cl|^{3/2}}{Cd}$ v odvisnosti od vpadnega kota profila (levo) in graf razmerja L/D v odvisnoti od vpadnega kota profila (desno).

Vzpenjanje

Takoj po odklopu modela se prične faza vzpenjanja, kjer model leti vertikalno z visoko hitrostjo (povprečno 35 m/s). Tokrat moramo opraviti test tipa 1 (fiksna hitrost). Za to fazo je značilno Reynoldsovo število okoli 300000, pri koeficientu vzgona 0. Koeficient vzgona mora biti 0 zato, da model potuje s čim manj upora. Nični koeficient upora dosežemo s pravilno nastavitvijo vpadnega kota krila. Slika 6.20 prikazuje vrednost vpadnega kota pri $C_l = 0$. Tukaj se bomo posvetili samo profilu SB1, saj se najbolj približuje profilu BE50. Profila bomo primerjali glede na količino upora, ki ga generirata. Iz slike 6.21 lahko vidimo, da profil SB1 generira manj upora kot BE50 (pri vpadnem kotu, kjer je koeficient vzgona enak 0). C_d pri SB1 je 0,0144, pri BE50 pa 0,0166. To pomeni, da bo v fazi vzpenjanja pridobil več višine.
Izmed analiziranih profilov se je SB1 najbolj približal uveljavljenemu profilu



Slika 6.20: Vpadni kot profila pri koeficientu vzgona 0.



Slika 6.21: Koeficient upora pri Cl = 0 (vzpenjanje) za profila SB1 in BE50.

"Verbitsky BE50 (smoothed)". Z nekaj ročnimi popravki oblike je možno izboljšati letalne lastnosti profila SB1, ki bi teoretično prekašale profil BE50. Edina pomankljivost je višina profila SB1, saj je zelo težko zgraditi tako tanko krilo, ki bi bilo dovolj robustno (problem velikosti vzdolžnega nosilca).

Poglavje 7

Zaključek

V okviru naloge smo na problemu optimizacije profila letalskega krila spoznali različne algoritme ter njihove prednosti in slabosti. V ta namen je bil razvit program Airfoil optimizer, ki ima implementirane optimizacijske algoritme diferencialna evolucija, metodo roja delcev in metodo ognjemeta. Za izračunavanje lastnosti profilov je bil uporabljen konzolni program Xfoil, za podrobnejšo ročno analizo pa program XFLR5 [1]. Program se lahko izvaja v grafičnem ali v tekstovnem načinu iz ukazne vrstice. V grafičnem načinu sta bila implementirana vizualizacija profila in črtni graf za spremljanje napredka optimizacije. Za slednjega je bila uporabljena knjižnica JFreeChart [5]. Tekstovni način izvajanja je namenjen testiranju parametrov oziroma serijskemu izvajanju programa. Možnost serijskega izvajanja izkoristimo za iskanje najboljše kombinacije parametrov algoritma.

Trenutno se vrednosti parametrov določijo vnaprej in se tekom izvajanja ne spreminjajo. Testiranju parametrov se lahko popolnoma izognemo, če algoritem sam ustrezno popravlja vrednost parametrov tekom izvajanja. V delu [12] je predstavljen algoritem PLES (ang.: Parameter-less evolutionary search), kjer ni potrebno nastaviti nobenega parametra, saj se ti iz generacije v generacijo popravljajo glede na potek optimizacije. Potrebno je samo določiti nekatere meta-parametre (npr.: število parametrov, natančnost itd.), s katerimi se avtomatsko določi velikost začetne populacije.

Po testiranju programske rešitve je na površje priplavalo nekaj pomanjkljivosti. Izkaže se, da upoštevanje razmerja vzgona in upora ni dovolj dober pokazatelj kakovosti geometrije profila, saj razmerje L/D favorizira nižje profile, ki imajo nižji koeficient upora in tako profili postanejo pretanki za praktično uporabo. Zaenkrat je še preuranjeno reči, ali je možno s hevrističnimi optimizacijskimi metodami priti do profila, ki bi bil boljši od trenutno uporabljenih profilov za modele F1A kategorije. Veliko sistematičnega testiranja je še potrebnega za določitev primernih omejitev posameznih kontrolnih točk Bézirjeve krivulje. Spoznali smo, da bi bilo bolje implementirati večkriterijske različice algoritmov, kjer bi optimirali vzgon, upor, točko prehoda iz laminarnega v turbulentni tok in velikost mejne plasti ter upoštevali konstrukcijske omejitve.

Literatura

[1] André Deperrois. XFLR5. http://www.xflr5.com/xflr5.htm. [Dostop 4.1.2017].

[2] Bernabé Dorronsoro. Genetic Algorithms. http://neo.lcc.uma.es/cEA-web/GA.htm, 2004. [Dostop 12.9.2016].

[3] Mark Drela. Xfoil. http://web.mit.edu/drela/Public/web/xfoil/. [Dostop 12.9.2016].

[4] Andries P. Engelbrecht. *Computational Intelligence - An Introduction*. John Wiley and Sons Ltd, 2007.

[5] David Gilbert. JFreeChart. http://www.jfree.org/jfreechart/. [Dostop 12.9.2016].

[6] Fédération Aéronautique Internationale. *FAI Sporting Code*.

[7] Fédération Aéronautique Internationale. Free flight. http://www.fai.org/ciam-our-sport/f1-free-flight. [Dostop 12.9.2016].

[8] Igor Kononenko and Marko Robnik Šikonja. *Inteligentni sistemi*. Založba FE in FRI, 2010.

[9] Jeremy Maginot, Marin Guenov, Paolo Fantini, and Mattia Padulo. A method for assisting the study of Pareto solutions in multi-objective optimization. In *7th AIAA ATIO Conf*, 2007.

[10] Wilson Mongwe. Understanding the EM Algorithm. http://www.wilsonmongwe.co.za/understanding-the-em-algorithm/, 2015. [Dostop 12.9.2016].

[11] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. Javanska knjižnica JMetal. http://jmetal.sourceforge.net/. [Dostop 12.9.2016].

[12] Gregor Papa. Parameter-less algorithm for evolutionary-based optimization. *Computational Optimization and Applications*, 56(1):209–229, 2013.

[13] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer, 2005.

[14] Krishnil Ravinesh Ram, Sunil Lal, and M.Rafiuddin Ahmed. Low Reynolds number airfoil optimization for wind turbine applications using genetic algorithm. *Journal of Renewable and Sustainable Energy 5*, 2013.

[15] Theodore A. Talay. *Introduction to the aerodynamics of flight*. Nasa, 1975.

[16] Ying Tan and Yuanchun Zhu. Fireworks algorithm for optimization. In *International Conference in Swarm Intelligence*, pages 355–364. Springer, 2010.

[17] Eric W. Weisstein. Bernstein polynomia. http://mathworld.wolfram.com/BernsteinPolynomial.html. [Dostop 12.9.2016].

[18] Eric W. Weisstein. Bézier curve. http://mathworld.wolfram.com/BezierCurve.html. [Dostop 12.9.2016].

[19] Wikipedia. Boundary layer — Wikipedia and the free encyclopedia. https://en.wikipedia.org/wiki/Flow_separation, 2016. [Dostop 12.9.2016].

[20] Wikipedia. Drag coeficient — Wikipedia and the free encyclopedia. `https://en.wikipedia.org/wiki/Drag_coefficient`, 2016. [Dostop 12.9.2016].

[21] Wikipedia. Lift coeficient — Wikipedia and the free encyclopedia. `https://en.wikipedia.org/wiki/Lift_coefficient`, 2016. [Dostop 12.9.2016].