Univerzitetna založba
Univerze v Mariboru

OPTIMIZACIJE V INŽENIRSTVU

Reševanje problemov z metahevrističnimi metodami v okolju MATLAB

Janez Gotlih Mirko Ficko

Univerza v Mariboru

Fakulteta za strojništvo

Optimizacije v inženirstvu

Reševanje problemov z metahevrističnimi metodami v okolju MATLAB

Avtorja

Janez Gotlih

Mirko Ficko

November 2025

Naslov
Title
Optimizacije v inženirstvu
Optimization in Engineering

Podnaslov
Subtitle
Reševanje problemov z metahevrističnimi metodami v okolju MATLAB
Solving Problems With Metaheuristic Methods in MATLAB

Avtorja
Authors
Janez Gotlih
(Univerza v Mariboru, Fakulteta za strojništvo)

Mirko Ficko
(Univerza v Mariboru, Fakulteta za strojništvo)

Recenzija
Review
Miran Brezočnik
(Univerza v Mariboru, Fakulteta za strojništvo)

Simon Klančnik
(Univerza v Mariboru, Fakulteta za strojništvo)

Jezikovni pregled
Language editing
Danica Gotlih

Tehnična urednika
Technical editors
Marina Bajić
(Univerza v Mariboru, Univerzitetna založba)

Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Oblikovanje ovitka
Cover designer
Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Grafika na ovitku
Cover graphic
Striped textile, foto: Scott Wemb, unsplash.com, 2020

Grafične priloge
Graphic material
Vsi viri so lastni, če ni navedeno drugače.
Gotlih, Ficko (avtorja), 2025

Založnik
Published by
Univerza v Mariboru
Univerzitetna založba
Slomškov trg 15, 2000 Maribor, Slovenija
https://press.um.si, zalozba@um.si

Izdajatelj
Issued by
Univerza v Mariboru
Fakulteta za strojništvo
Smetanova ulica 17, 2000 Maribor, Slovenija
https://www.fs.um.si/, fs@um.si

Izdaja
Edition
Prva izdaja

Vrsta publikacije
Publication type
E-knjiga

Dostopno na
Available at
https://press.um.si/index.php/ump/catalog/book/1076

Izdano
Published at
Maribor, Slovenija, november 2025

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

62:519.8(035)(0.034.2)

GOTLIH, Janez
Optimizacije v inženirstvu [Elektronski vir] : reševanje problemov z metahevrističnimi metodami v okolju MATLAB / avtorja Janez Gotlih, Mirko Ficko. - 1. izd. - E-knjiga. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2025

Način dostopa (URL):
https://press.um.si/index.php/ump/catalog/book/1076

ISBN 978-961-299-079-4 (PDF)
doi: 10.18690/um.fs.11.2025
COBISS.SI-ID 256635651

ISBN 978-961-299-079-4 (pdf)

DOI https://doi.org/10.18690/um.fs.11.2025

Cena
Price Brezplačni izvod

Odgovorna oseba založnika
For publisher Prof. dr. Zdravko Kačič,
rektor Univerze v Mariboru

Citiranje
Attribution Gotlih, J., Ficko, M.(2025). *Optimizacije v inženirstvu: reševanje problemov z metahevrističnimi metodami v okolju MATLAB.* Univerza v Mariboru, Univerzitetna založba. doi:10.18690/um.fs.11.2025

Univerzitetna založba
Univerze v Mariboru

Kazalo

Univerzitetna založba
Univerze v Mariboru

1 Uvod

V inženirstvu se skoraj vsak dan srečujemo z odločitvami, pri katerih je treba izbrati najboljšo možno rešitev med mnogimi alternativami. Naj bo to optimizacija stroškov, iskanje najprimernejših procesnih parametrov, izboljšanje kakovosti izdelka ali usklajevanje med hitrostjo in zanesljivostjo se za vsako od teh nalog skriva osnovni princip: kako nekaj narediti boljše. Optimizacija kot znanstvena in aplikativna disciplina nam omogoča, da take odločitve ne sprejemamo več zgolj na podlagi občutka ali preizkusov z metodo poskusov in napak, temveč sistematično, z uporabo matematičnih modelov in algoritmov.

Namen teh skript je ponuditi uvod v področje optimizacije v inženirstvu na način, ki povezuje teoretično ozadje z uporabo v praksi. Obravnavajo temeljne pojme enokriterijske in večkriterijske optimizacije, posebnosti obravnave omejitev ter implementacijo metahevrističnih algoritmov, kot sta genetski algoritem (GA) in algoritem rojev delcev (PSO). Poudarek je na razumevanju, ne zgolj na uporabi orodij, kar pomeni, da bo bralec razumel tudi ozadje, zakaj in kako algoritem deluje, ne le kateri ukaz v izbranem programskem okolju uporabiti.

V teh skriptih posebno pozornost namenjamo metahevrističnim algoritmom, saj so se izkazali kot zelo učinkoviti pri reševanju zahtevnih inženirskih optimizacijskih nalog. Za razliko od klasičnih metod, ki pogosto zahtevajo gladkost, konveksnost ali

odvodljivost funkcij, metahevristike omogočajo iskanje rešitev tudi v nelinearnih, večdimenzionalnih in diskretnih prostorih. Zaradi svoje prilagodljivosti in robustnosti se uspešno uporabljajo na številnih področjih, kot so načrtovanje proizvodnje, optimizacija virov, vodenje procesov in načrtovanje poti oz. povsod tam, kjer so razmere kompleksne, omejitve številne, cilji pa med seboj v konfliktu.

Skripta so razdeljena na več vsebinskih sklopov. Po uvodnem nagovoru se najprej posvetimo optimizacijam z enim ciljem oz. enokriterijskim optimizacijam, kjer spoznamo osnovne koncepte in matematične modele za optimizacijo brez omejitev in z njimi. Na tem temelju gradimo praktične vaje, kot so optimizacija izdelovalnih procesov, vključevanje škatlastih, enakovrednostnih in neenakovrednostnih omejitev, ter dopolnjujemo razumevanje z nalogami za samostojno delo. V nadaljevanju obravnavamo optimizacijo z več cilji oz. večkriterijsko optimizacijo in pojem Pareto optimalnosti, vključno z vajami in dodatnimi nalogami, ki od študentov zahtevajo razmislek o kompromisih med cilji.

Posebno poglavje je posvečeno algoritmu rojev delcev, kjer postopoma razvijemo njegovo matematično formulacijo, implementacijo v MATLAB-u in vaje, ki vodijo od osnovne inicializacije do vizualizacije konvergence in primerjav med parametri. Na koncu je predstavljen nabor značilnih optimizacijskih funkcij, kot so Sphere, Rastrigin, Himmelblau in Rosenbrock, ter primeri Pareto front, ki služijo kot referenčni testni primeri za večkriterijsko optimizacijo.

Pri vajah in nalogah ni bistveno, da študent najde absolutno pravilno rešitev, temveč da zna zasnovati optimizacijski problem, poiskati primerno metodo, ovrednotiti rezultat in se naučiti nekaj novega o delovanju algoritmov. Zato skripta spodbujajo aktivno delo skozi raziskovanje, primerjanje in vizualizacijo rezultatov. Vsi algoritmi so predstavljeni tako, da jih bralec lahko razume in samostojno implementira v okolju MATLAB, ki je bilo izbrano kot programska podlaga za vse primere v teh skriptih.

Vsi prikazani primeri so bili razviti in testirani v okolju MATLAB R2024a. Za izvajanje skript sta potrebna naslednja dodatka (angl. toolbox oz. Add-In):

- Optimization Toolbox,
- Global Optimization Toolbox.

Čeprav optimizacija na prvi pogled deluje kot matematično zahtevna tema, si ta skripta prizadevajo pokazati, da je mogoče ključne koncepte razumeti tudi brez naprednega matematičnega predznanja, s pomočjo analogij, vizualizacij in preprostih primerov. Bralec je vabljen, da k obravnavanim vsebinam pristopi z radovednostjo in odprtostjo. Če razumevanje na začetku še ni popolno, se s postopnim napredovanjem oblikuje jasnejša slika. In kar je najlepše, optimizacijo se da naučiti tako, da jo preprosto začnemo uporabljati.

Univerzitetna založba
Univerze v Mariboru

2 Optimizacija z enim ciljem

V inženirstvu se pogosto srečujemo s problemi, kjer je cilj izboljšati en sam kriterij, na primer minimizirati stroške, maksimirati učinkovitost ali zmanjšati porabo energije. Tovrstne probleme označujemo kot optimizacije z enim ciljem oz. enokriterijske optimizacijske probleme (angl. single-objective optimization).

Neomejene optimizacije z enim ciljem

Osnovni enokriterijski optimizacijski problem se lahko matematično zapiše kot:

$$\mathbf{min}\ \boldsymbol{f}(\boldsymbol{x})\ \mathbf{pri}\ \boldsymbol{x} \in \mathbb{R}^{\boldsymbol{m}}, \tag{2.1}$$

kjer je:

- $\boldsymbol{x} \in \mathbb{R}^{m}$: vektor spremenljivk. V inženirski praksi so to npr. parametri postopka. Velikost vektorja $\boldsymbol{x}$ definira dimenzijo problema;
- $\boldsymbol{f}(\boldsymbol{x})$**:** ciljna funkcija (npr. strošek, napaka, čas, energija).

Omejene optimizacije z enim ciljem

V praksi se optimizacijski problemi pogosto pojavljajo z različnimi omejitvami, ki jih je treba upoštevati pri njihovi zasnovi. Pri mehanski obdelavi na primer največja dovoljena hitrost vretena omejuje možen čas obdelave, čeprav bi višja rezalna hitrost teoretično lahko vodila do njegovega skrajšanja.

Matematično lahko omejeni enokriterijski optimizacijski problem zapišemo kot:

$$\min f(\boldsymbol{x}) \text{ pri } \boldsymbol{x} \in S, \tag{2.2}$$

kjer je *f* skalarna funkcija in *S* nabor omejitev.

Omejitve so lahko definirane kot:

$$S = \{\boldsymbol{x} \in \mathbb{R}^m: \boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max};\ h(\boldsymbol{x}) = 0;\ g(\boldsymbol{x}) \geq 0\},$$

kjer so:

- S: dopustna množica vseh vrednosti $\boldsymbol{x}$, ki izpolnjujejo omejitve. Če rešitev ne ustreza omejitvam, ni dopustna;

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max}; \tag{2.3}$$

- $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$: škatlaste omejitve prostora spremenljivk (angl. box constraints). Določajo domeno iskanja oz. dopustno območje vrednosti posameznih spremenljivk in so običajno definirane kot neenačbene;

$$h(\boldsymbol{x}) = 0; \tag{2.4}$$

- $h(\boldsymbol{x}) = 0$: enačbene (enakovrednostne) omejitve. Stroge omejitve, ki jih mora rešitev natančno izpolnjevati;

$$g(\boldsymbol{x}) \geq 0; \tag{2.5}$$

- $g(\boldsymbol{x}) \geq 0$: neenačbene (neenakovrednostne) omejitve. Omejitve, ki jih rešitev ne sme kršiti, vendar imajo nekaj prostosti.

2.1 Vaje s področja optimizacij z enim ciljem

V tem poglavju so predstavljene naloge, ki se osredotočajo na reševanje optimizacijskih problemov z enim ciljem (2.1). Vaje so zasnovane tako, da omogočajo razumevanje vpliva parametrov algoritmov in razvoja ustreznih modelov

za iskanje optimalnih rešitev na primerih, ki ponazarjajo realne tehnološke zahteve, kot so varna območja delovanja stroja in stabilnost sil med obdelavo. Poglavje poudarja pomen pravilnega modeliranja, uporabe ustreznih omejitev in interpretacije rezultatov optimizacije v inženirski praksi.

2.1.1 Vaja: Enodimenzionalne optimizacije z enim ciljem

Cilj te vaje je najti optimalno vrednost hitrosti podajanja v_f, ki minimizira obrabo orodja. Ker je hitrost podajanja edina neodvisna spremenljivka, pravimo, da je problem enodimenzionalen. Obraba orodja je v tem študijskem primeru modelirana s preprosto kvadratno funkcijo:

$$f(v_f) = {v_f}^2,$$

kjer je:

- $f(v_f)$: funkcija obrabe orodja v odvisnosti od podajalne hitrosti v_f in je ciljna funkcija optimizacijskega problema. Za obrabo orodja v praksi velja: nižja je boljša;
- v_f: hitrost podajanja v mm/min in je spremenljivka optimizacijskega problema.

Ciljna funkcija je konveksna kvadratna funkcija z minimumom pri $x = 0$. Vrednost minimuma ciljne funkcije pri $x = 0$ pa je prav tako $f(v_f) = 0$.

- V okolju MATLAB ustvarite nov skript z imenom *PSO_1D.m*.
- V odprt skript prilepite spodnjo kodo.
- Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.

```
%% Matlab vaja: Optimizacija hitrosti podajanja z algoritmom rojev
delcev (PSO)

%% Definicija problema
% Število spremenljivk (dimenzija problema)
steviloSpremenljivk = 1;

% Definicija ciljne funkcije (obraba orodja kot kvadrat podajalne
hitrosti)
```

```
CiljnaFunkcija = @funkcija_obrabe;

%% Zagon algoritma rojev delcev (PSO)
[v_f, y] = particleswarm(CiljnaFunkcija, steviloSpremenljivk);

%% Ciljna funkcija
function y = funkcija_obrabe(v_f)
% FUNKCIJA_OBRABE - model kvadratne odvisnosti obrabe orodja od
hitrosti podajanja

% Kvadratna funkcija obrabe
y = v_f^2;

end
```

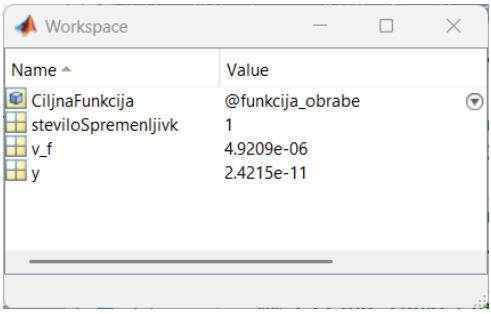
Zapisano kodo lahko razdelimo na tri logične dele, ki so v skriptu ločeni z oznako *%%*. Znak *%* označuje komentarje, ki se ne izvajajo in služijo za razlago kode, medtem ko *%%* označuje začetek novega odseka, ki ga lahko samostojno zaženemo. Takšna razdelitev olajša strukturiranje, preglednost in testiranje skripta. Prvi del običajno vsebuje definicijo problema, drugi del zajema definicijo ciljne funkcije in morebitnih omejitev, ki so lahko zapisane neposredno v skript, na njenem dnu ali v ločeni funkcijski datoteki. Tretji del predstavlja zagon optimizacijskega algoritma in prikaz rezultatov.

V odseku definicije problema določa spremenljivka *steviloSpremenljivk* dimenzijo optimizacijskega problema. Ker je ciljna funkcija odvisna od ene same spremenljivke, hitrosti podajanja (*v_f*), je dimenzija problema enaka 1. *CiljnaFunkcija* je ročica (angl. function handle) do funkcije *funkcija_obrabe*, ki predstavlja ciljno funkcijo optimizacije.

V odseku zagon algoritma funkcija *particleswarm* izvede optimizacijo z algoritmom rojev delcev (PSO) za dano ciljno funkcijo. Prvi argument funkcije *particleswarm* je ročica do ciljne funkcije *funkcija_obrabe*, drugi argument pa določa število optimizacijskih spremenljivk. Funkcija *particleswarm* vrne dva rezultata: *v_f* je optimalna vrednost hitrosti podajanja, ki minimizira obrabo orodja, *y* pa je pripadajoča minimalna vrednost ciljne funkcije (tj. najmanjša obraba).

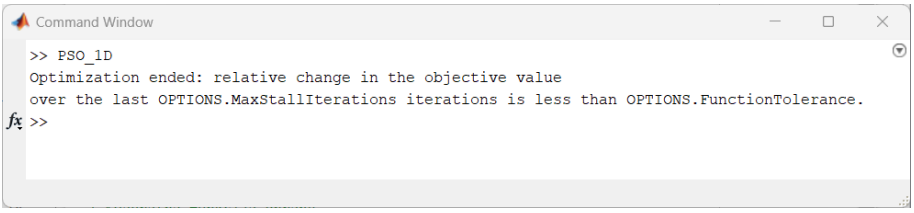
V odseku ciljna funkcija je definirana funkcija *funkcija_obrabe*, ki modelira obrabo orodja kot kvadratno odvisnost od hitrosti podajanja.

Po zagonu skripta se v delovnem prostoru (Workspace) pojavijo štiri spremenljivke (Slika 1). Spremenljivki *CiljnaFunkcija* in *steviloSpremenljivk* sta del definicije problema in se samo prepišeta v delovni prostor. Rezultat optimizacije je shranjen v spremenljivki *v_f*, katere vrednost je 4,9209e-06, kar je praktično enako vrednosti nič. Spremenljivka *y* vsebuje vrednost ciljne funkcije pri tem optimumu, v tem primeru 2,4215e-11, kar je kvadrat optimalne hitrosti in prav tako zelo blizu nič. Tak rezultat potrjuje pravilno delovanje algoritma, saj vemo, da je minimum kvadratne funkcije obrabe $y = 0$ dosežen pri podajalni hitrosti $v_f = 0$.



Slika 1: Izpis rezultatov optimizacije v delovnem prostoru (Workspace)

Slika 2 prikazuje izpis v ukaznem oknu (Command Window), ki sporoča, da je bila optimizacija zaključena, ker se vrednost ciljne funkcije v zadnjih iteracijah ni več bistveno spreminjala. To pomeni, da je algoritem dosegel stabilno rešitev in ustavil postopek optimizacije na podlagi ustavitvenega kriterija *FunctionTolerance*. Tak zaključek pomeni, da je bila konvergenca dosežena pred porabo maksimalnega števila iteracij.



Slika 2: Izpis rezultatov v ukaznem oknu (Command Window)

2.1.2 Vaja: Večdimenzionalne optimizacije z enim ciljem

Poleg hitrosti podajanja pa na obrabo orodja vpliva še več dejavnikov, zato moramo tudi optimizacijski problem znati razširiti na več spremenljivk. V našem primeru optimizacije obrabe orodja lahko k hitrosti podajanja dodamo še rezalno hitrost, tako da obravnavamo problem z dvema spremenljivkama:

- v_f: hitrost podajanja (mm/min),
- v_c: rezalna hitrost (m/min).

Zaradi preglednosti bomo tudi v tem primeru kot ciljno funkcijo uporabili preprosto kvadratno funkcijo. Če torej v model obrabe vključimo še rezalno hitrost, dobi ciljna funkcija obliko dvodimenzionalne kvadratne funkcije:

$$f(\boldsymbol{x}) = {v_f}^2 + {v_c}^2.$$

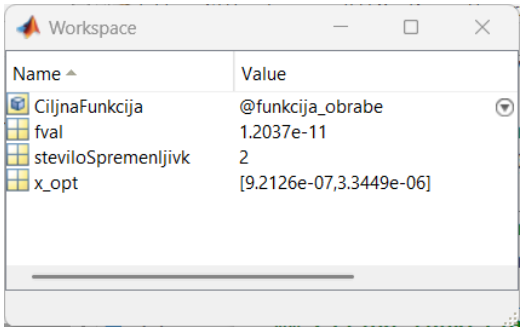
Minimum ciljne funkcije $f(x_1, x_2) = 0$ je pri $x_1 = v_f = 0$ in $x_2 = v_c = 0$. Prostor spremenljivk $\boldsymbol{x}$ je v tem primeru dvodimenzionalni vektor, saj imamo pri optimizacijskem problemu dve neodvisni spremenljivki v_f in v_c. Ciljni prostor je enodimenzionalen, saj imamo samo en cilj oz. kriterij tj. obraba orodja.

- Ustvarite nov skript z imenom *PSO_2D.m*.
- V odprt skript prilepite spodnjo kodo.
- Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.

```
%% Matlab vaja: Optimizacija podajalne in rezalne hitrosti s PSO

%% Definicija problema
% Število spremenljivk (dimenzija problema)
steviloSpremenljivk = 2;

% Definicija ciljne funkcije (obraba orodja kot kvadrat voste)
CiljnaFunkcija = @funkcija_obrabe;

%% Zagon algoritma rojev delcev (PSO)
[x_opt, fval] = particleswarm(CiljnaFunkcija, steviloSpremenljivk);

%% Ciljna funkcija
```

```
function y = funkcija_obrabe(x)

% Model kvadratne odvisnosti obrabe orodja od hitrosti podajanja in
rezalne hitrosti
y = x(1)^2+x(2)^2;

end
```

Razlika med prejšnjo in zdajšnjo vajo je v tem, da iz enodimenzionalnega problema preidemo v dvodimenzionalni optimizacijski problem s spremenljivkama *x(1)* in *x(2)*, pri čemer se spremeni tudi ciljna funkcija, ki zdaj upošteva oba vpliva.



Slika 3: Rezultati optimizacije podajalne in rezalne hitrosti s PSO

Slika 3 prikazuje delovni prostor po zaključeni optimizaciji z algoritmom rojev delcev. Optimalna vrednost spremenljivk *x_opt* predstavlja hitrost podajanja in hitrost rezanja, ki skupaj minimizirata obrabo orodja. Vrednost ciljne funkcije *fval* pri teh parametrih je zelo blizu nič (1,2037e-11), kar potrjuje uspešno iskanje minimuma kvadratne funkcije. Da rezultat ni povsem enak nič, ampak le zelo blizu, je posledica postopnega približevanja optimumu (konvergence) ter ustavitvenega kriterija algoritma, ki določa, kdaj se iskanje zaključi.

2.1.3 Vaja: Optimizacija s škatlastimi omejitvami

Pri realnih inženirskih optimizacijskih problemih spremenljivke pogosto ne smejo presegati določenih fizičnih ali tehnoloških mej. Tovrstne omejitve imenujemo škatlaste (angl. box constraints), saj omejujejo iskanje rešitve na pravokotni oz. škatlasti podprostor vektorja spremenljivk (**2.3**).

Za dvodimenzionalni primer obdelave intervale omejitev določa proizvajalec orodja ali tehnološki postopek. V našem primeru izberemo:

- $50 \leq v_f \leq 150$ [mm/min],
- $100 \leq v_c \leq 300$ [m/min].

Spodnji meji ($\boldsymbol{x}_{\min}$) za hitrost podajanja in rezanja preprečujeta neučinkovito ali nemogočo obdelavo, ki bi jo povzročile premajhne vrednosti. Zgornji meji ($\boldsymbol{x}_{\max}$) pa lahko preprečujeta pregretje, lom ali druge poškodbe orodja, nestabilnosti obdelave, lahko pa sta tudi pogojeni z zmogljivostjo stroja.

Zaradi vpeljave škatlastih omejitev prilagodimo ciljno funkcijo:

$$f(\boldsymbol{x}) = \left(v_f - 100\right)^2 + (v_c - 200)^2.$$

S to spremembo ciljne funkcije smo modelirali realno situacijo, v kateri optimum ni več v matematični ničli, temveč v tehnično priporočeni točki. Najmanjša obraba orodja bo dosežena pri hitrosti podajanja $v_f = 100$ mm/min in rezalni hitrosti $v_c = 200$ m/min, torej v notranjosti dovoljenega območja, ne na njegovih mejah.

- Ustvarite nov skript z imenom *GA_skatlaste_omejitve.m*.
- V odprt skript prilepite spodnjo kodo.
- Kodo zaženite s klikom na gumb Run ali s pritiskom tipke F5.

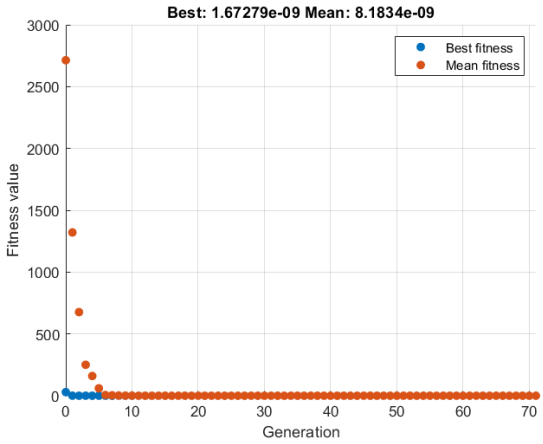
```
%% Matlab vaja: Optimizacija podajalne in rezalne hitrosti s
škatlastimi omejitvami

%% Definicija problema
% Število spremenljivk (dimenzija problema)
steviloSpremenljivk = 2;

% škatlaste omejitve prostora spremenljivk

lb = [50 150];
ub = [100 300];

% Definicija ciljne funkcije (obraba orodja kot kvadrat razlike)
CiljnaFunkcija = @funkcija_obrabe;
```

```
% možnosti GA
options = optimoptions(@ga, 'PlotFcn', {@gaplotbestf}, 'Display',
'iter');

%% Zagon genetskega algoritma (GA)
[x_opt, fval] = ga(CiljnaFunkcija, steviloSpremenljivk, [], [], [],
[], lb, ub, [], options);

%% Ciljna funkcija

function y = funkcija_obrabe(x)

% FUNKCIJA_OBRABE - model kvadratne odvisnosti obrabe orodja od
hitrosti podajanja in rezalne hitrosti
y = (x(1)-100)^2+(x(2)-200)^2;

end
```

Problem je v tej vaji razširjen z uvedbo škatlastih omejitev, kar pomeni, da sta za vsako izmed dveh spremenljivk določeni spodnja in zgornja meja. Druga ključna sprememba pri tej vaji je uporaba drugega optimizacijskega algoritma. Namesto algoritma rojev delcev je uporabljen genetski algoritem, ki je posebej primeren za optimizacijske probleme z omejitvami. Dodatno so v strukturi *options* vključene možnosti za sprotno vizualizacijo poteka optimizacije in izpis rezultatov po posameznih iteracijah, kar omogoča boljši vpogled v konvergenco algoritma.

Pomembno je poudariti, da kljub tej bistveni spremembi v načinu iskanja optimuma osnovna definicija optimizacijskega problema ostaja skoraj nespremenjena. Ciljna funkcija in število spremenljivk ostajata enaka, spremenijo se le argumenti funkcije *ga*, ki so prilagojeni posebnostim uporabe genetskega algoritma.

Poleg rezultatov v delovnem prostoru se nam med izvajanjem algoritma izriše graf poteka konvergence (Slika 4). Graf za naš primer prikazuje hitro konvergenco genetskega algoritma, saj se vrednosti ciljne funkcije že po nekaj generacijah približajo ničli. Razlika med najboljšo in povprečno rešitvijo pa je majhna, kar kaže na stabilno in učinkovito delovanje algoritma.



Slika 4: Potek konvergence genetskega algoritma

2.1.4 Vaja: Optimizacija z enakovrednostnimi omejitvami

V obdelovalnem procesu je pogosto tehnološka zahteva, da rezalna sila ostane na določeni ciljni vrednosti, na primer 700 N. Takšna zahteva izhaja iz potrebe po stabilnosti procesa, omejevanju mehanskih obremenitev ter zagotavljanju ponovljive obrabe orodja. V optimizacijskem modelu to zahtevo upoštevamo kot enakovrednostno omejitev (**2.4**), kar pomeni, da mora biti izračunana rezalna sila enaka vnaprej določeni ciljni vrednosti.

Zapis enakovrednostne omejitve:

$$h(\boldsymbol{x}) = F_{rez}(\boldsymbol{x}) - 700 = 0.$$

Če $h(\boldsymbol{x}) \neq 0$, je rešitev neveljavna. S tem optimizacijski algoritem prisilimo, da išče take parametre procesa, ki izpolnjujejo to tehnološko zahtevo.

Za model rezalne sile je uporabljen poenostavljen linearni izraz, ki povezuje rezalno silo z obema vhodnima spremenljivkama (hitrost podajanja in rezalna hitrost):

$$F_{rez} = 600 + 0.5v_f + 0.3v_c.$$

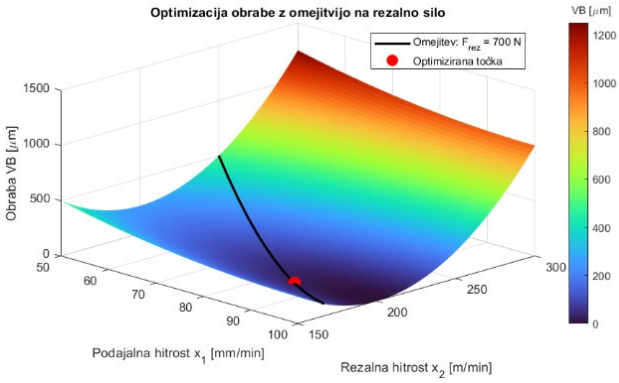
➢ Ustvarite nov skript z imenom *GA_enakovrednostne_omejitve.m*.

- V odprt skript prilepite spodnjo kodo.
- Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.

```
%% Matlab vaja: Optimizacija podajalne in rezalne hitrosti z
enakovrednostnimi omejitvami

% Število spremenljivk (dimenzija problema)
steviloSpremenljivk = 2;

% škatlaste omejitve prostora spremenljivk

lb = [50 150];
ub = [100 300];

% Definicija ciljne funkcije
CiljnaFunkcija = @funkcija_obrabe;

% Definicija omejitev
EnakovrednostneOmejitve = @omejitve;

% možnosti GA
options = optimoptions(@ga, 'PlotFcn', {@gaplotbestf}, 'Display',
'iter');

%% Zagon genetskega algoritma (GA)
[x_opt, fval] = ga(CiljnaFunkcija, steviloSpremenljivk, [], [], [],
[], lb, ub, EnakovrednostneOmejitve, options);

%% Ciljna funkcija

function y = funkcija_obrabe(x)

% model kvadratne odvisnosti obrabe orodja od hitrosti podajanja in
rezalne hitrosti
y = 0.1*((x(1)-100)^2+(x(2)-200)^2);

end

%% Enakovrednostne omejitve

function [c, ceq] = omejitve(x)
```

```
    % Model rezalne sile
    F_rez = 600 + 0.5 * x(1) + 0.3 * x(2);

    % Enakovrednostna omejitev
    ceq = F_rez - 700;

    c = [];  % brez neenakovrednostnih omejitev
end
```

V tej vaji je bil optimizacijski problem nadgrajen z uvedbo enakovrednostne omejitve, ki zahteva, da mora biti izračunana rezalna sila *F_rez* enaka 700 N. Rezalna sila je definirana kot funkcija hitrosti podajanja in rezalne hitrosti. Omejitvena funkcija je definirana ločeno in dodeljena spremenljivki *EnakovrednostneOmejitve*, kar poveča preglednost programske kode in omogoča formulacijo omejitev tudi v zunanji datoteki. Dodatno je pri ciljni funkciji uporabljena utež 0,1, ki služi skaliranju cilja, da so vrednosti obrabe orodja skladne s fizikalno pričakovanimi velikostmi in primerljive z drugimi kriteriji.



Slika 5: Površina ciljne funkcije z označeno omejitvijo na rezalno silo in optimalno točko

Slika 5 prikazuje kompromis med iskanjem optimuma in izpolnjevanjem tehnoloških omejitev. Na sliki je prikazana površina ciljne funkcije, ki predstavlja obrabo orodja v odvisnosti od podajalne hitrosti in rezalne hitrosti. Barvna lestvica ponazarja vrednost funkcije, pri čemer nižje vrednosti ustrezajo manjšim obrabam. Črna črta označuje območje, kjer je izpolnjena enakovrednostna omejitev, torej rezalna sila znaša natanko 700 N. Na tej krivulji je z rdečo piko označena optimizirana točka, ki

predstavlja kombinacijo hitrosti, pri kateri je obraba minimalna ob hkratnem upoštevanju omejitve na silo.

2.1.5 Vaja: Optimizacije z neenakovrednostnimi omejitvami

Z neenakovrednostnimi omejitvami $g(\boldsymbol{x})$ želimo preprečiti, da bi se proces obdelave izvajal v območju, kjer se pojavljajo regenerativne vibracije. Te vibracije negativno vplivajo na kakovost obdelane površine, obrabo orodja in varnost obdelovalnega postopka, saj lahko povzročijo poškodbe sistema.

Vibracijsko območje v prostoru procesnih parametrov (podajalna hitrost v_f in rezalna hitrost v_c) lahko približno opišemo s krogom s središčem v (v_{f0}, v_{c0}) in polmerom r ter omejitev zapišemo v obliki neenačbe (**2.5**):

$$r^2 - \left(v_f - v_{f0}\right)^2 - (v_c - v_{c0})^2 \leq 0.$$

Če pogoj $g(\boldsymbol{x}) \leq 0$ ni izpolnjen, to pomeni, da je rešitev znotraj območja vibracij, kjer je proces nestabilen. Takšne rešitve so iz tehnološkega vidika nesprejemljive, zato jih optimizacijski postopek zavrne. S to omejitvijo zagotovimo, da bodo dovoljene rešitve le tiste, ki ležijo zunaj vibracijskega območja.

- Ustvarite nov skript z imenom *GA_neenakovrednostne_omejitve.m*.
- V odprt skript prilepite spodnjo kodo.
- Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.

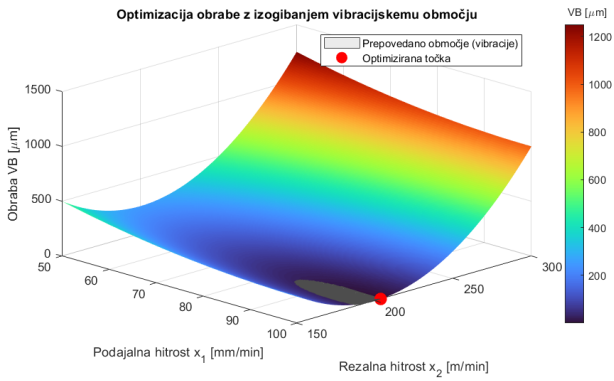
```
%% Matlab vaja: Optimizacija podajalne in rezalne hitrosti z
neenakovrednostnimi omejitvami

% Število spremenljivk (dimenzija problema)
steviloSpremenljivk = 2;

% škatlaste omejitve prostora spremenljivk

lb = [50 150];
ub = [100 300];

% Definicija ciljne funkcije (obraba orodja kot kvadrat razlike)
CiljnaFunkcija = @funkcija_obrabe;
```

```
% možnosti GA
options = optimoptions(@ga, 'PlotFcn', {@gaplotbestf}, 'Display',
'iter');

%% Zagon genetskega algoritma (GA)
[x_opt, fval] = ga(CiljnaFunkcija, steviloSpremenljivk, [], [], [],
[], lb, ub, @omejitve, options);

%% Ciljna funkcija

function y = funkcija_obrabe(x)

% model kvadratne odvisnosti obrabe orodja od hitrosti podajanja in
rezalne hitrosti
y = 0.1*(x(1)-100)^2+(x(2)-200)^2;

end

%% Nenakovrednostne omejitve

function [c, ceq] = omejitve(x)

% Središče vibracijskega območja
vf0 = 95;
vc0 = 195;

% Polmer kroga vibracijskega območja
r = 10;

% Neenačbena omejitev c <= 0 pomeni, da smo ZUNAJ kroga (varno)
krog = (x(1) - vf0)^2 + (x(2) - vc0)^2;
c = r^2 - krog;

% Ni enakovrednostnih omejitev
ceq = [];

end
```

V tej različici kode je glavna sprememba uvedba neenakovrednostne omejitve c, ki predstavlja prepovedano območje v prostoru hitrosti. Optimizacijski algoritem, mora upoštevati varnostno mejo, saj mora rešitev ležati izven kroga s središčem v

točki (95, 195) in polmerom 10. To pomeni, da algoritem išče optimum, ki je izven tega območja.

Slika 6 prikazuje kompromis med iskanjem optimalne kombinacije hitrosti in izogibanjem tehnološko neugodnim območjem. Svetlo sivo označeno območje na ciljni površini predstavlja prepovedano območje, v katerem bi lahko zaradi kombinacij hitrosti prišlo do neželenih vibracij. To območje je modelirano kot krog v prostoru hitrosti in ga optimizacijski algoritem obravnava kot neenakovrednostno omejitev, ki se ji je treba izogniti. Rdeča pika označuje optimizirano točko, ki leži izven vibracijskega območja in hkrati minimizira obrabo orodja.



Slika 6: Optimizacija obrabe z uporabo neenakovrednostih omejitev za izogibanje vibracijskemu območju

2.2 Samostojno delo s področja optimizacij z enim ciljem

V okviru tega poglavja so predstavljene samostojne naloge, katerih namen je poglabljanje razumevanja postopkov optimizacije z enim ciljem. Vsaka naloga vključuje implementacijo in uporabo izbranega optimizacijskega algoritma, kot sta genetski algoritem ali algoritem rojev delcev, za reševanje standardnih ali poenostavljenih inženirskih problemov. Obravnavani so tako realistični tehnični primeri (npr. optimizacija obdelovalnih hitrosti) kot tudi znane matematične testne funkcije, kot so Rastrigin, Himmelblau, Gomez and Levy ter Rosenbrock. Namen vaj ni le iskanje globalnega minimuma v podanem prostoru, temveč tudi vizualizacija rezultatov, primerjava uspešnosti različnih algoritmov ter razmislek o fizikalnem smislu dobljenih rešitev. Naloge spodbujajo razumevanje omejitev, občutljivosti na inicializacijo ter pomena dobro definirane ciljne funkcije in iskalnega prostora.

2.2.1 Samostojno delo: Optimizacija obdelave z genetskim algoritmom

Preoblikujte obstoječo optimizacijsko kodo iz poglavja 2.1.2, ki uporablja algoritem rojev delcev (PSO), tako da bo namesto tega uporabljala genetski algoritem (GA). Namen naloge je poiskati kombinacijo podajalne in rezalne hitrosti, ki minimizira preprosto kvadratno ciljno funkcijo obrabe orodja:

$$f(\boldsymbol{x}) = {v_f}^2 + {v_c}^2.$$

Namig: Če potrebujete pomoč glede uporabe funkcije ga(), si pomagajte z MATLAB dokumentacijo ali Help.

Zahteve naloge:

- Implementirajte ciljno funkcijo v obliki funkcije.
- Nastavite meje iskanja za hitrosti $v_f \in [50; 150]$ in $v_c \in [100; 300]$.
- Uporabite funkcijo *ga* za iskanje minimuma funkcije v prostoru dveh spremenljivk v_f in v_c.
- Izpišite optimalno rešitev in pripadajočo vrednost ciljne funkcije.
- Utemeljite, ali ima rešitev, pri kateri je $v_f = 0$ ali $v_c = 0$, smisel v kontekstu obdelovalnih procesov.

2.2.2 Samostojno delo: Optimizacija funkcije Rastrigin

V tej nalogi boste optimirali znano testno funkcijo Rastrigin z uporabo optimizacijskega algoritma rojev delcev (PSO). Cilj je poiskati globalni minimum funkcije v omejenem prostoru ter prikazati potek in rezultat optimizacije.

Zahteve naloge:

- Implementirajte Rastriginovo funkcijo (poglavje 5.2) v obliki funkcije.
- Nastavite meje iskanja na $[-5{,}12;\ 5{,}12]$ za vsako spremenljivko.
- Uporabite funkcijo *particleswarm* za iskanje minimuma funkcije za primer z dvema spremenljivkama $(n = 2)$.
- Izpišite optimalno rešitev in pripadajočo vrednost funkcije.
- (Dodatno): Prikažite 3D-graf funkcije z označeno rešitvijo in primerjajte delovanje z drugim algoritmom (npr. genetskim algoritmom).

2.2.3 Samostojno delo: Optimizacija funkcije Himmelblau

V tej nalogi boste optimirali znano testno funkcijo Himmelblau z uporabo optimizacijskega genetskega algoritma (GA). Cilj je poiskati enega izmed globalnih minimumov funkcije v omejenem prostoru ter prikazati potek in rezultat optimizacije.

Zahteve naloge:

- Implementirajte funkcijo Himmelblau (poglavje 5.3) v obliki funkcije.
- Nastavite meje iskanja na $[-5;\ 5]$ za obe spremenljivki.
- Uporabite funkcijo *ga* za iskanje minimuma funkcije.
- Izpišite optimalno rešitev in pripadajočo vrednost funkcije.
- Večkrat ponovno zaženite optimizacijo. Kaj ugotovite?
- (Dodatno): Prikažite 3D-graf funkcije z označeno rešitvijo in primerjajte delovanje z drugim algoritmom (npr. algoritem roja delcev).

2.2.4 Samostojno delo: Optimizacija funkcije Gomez and Levy

V tej nalogi boste optimirali testno funkcijo Gomez and Levy z uporabo genetskega algoritma. Cilj je poiskati globalni minimum funkcije v omejenem prostoru ter prikazati potek in rezultat optimizacije.

Zahteve naloge:

- Implementirajte funkcijo Gomez and Levy (poglavje 5.4) v obliki funkcije.
- Implementirajte neenačbno omejitev v obliki funkcije.
- Nastavite meje iskanja na $[-1;\ 0{,}75]$ za spremenljivko $\boldsymbol{x}$ in $[-1;\ 1]$ za $\boldsymbol{y}$.
- Uporabite funkcijo *ga* za iskanje minimuma funkcije.
- Izpišite optimalno rešitev in pripadajočo vrednost ciljne funkcije.
- (Dodatno): Prikažite 3D-graf funkcije z označeno rešitvijo in primerjajte delovanje z algoritmom roja delcev. Kaj ugotovite?

2.2.5 Samostojno delo: Optimizacija funkcije Rosenbrock z omejitvami

V tej nalogi boste optimirali funkcijo Rosenbrock z omejitvami z uporabo genetskega algoritma. Naloga vključuje iskanje globalnega minimuma funkcije v omejenem prostoru in pod dvema nelinearnima omejitvama.

Zahteve naloge:

- Implementirajte funkcijo Rosenbrock (poglavje 5.5) v obliki funkcije.
- Implementirajte obe neenačbni omejitvi v obliki funkcije.
- Nastavite meje iskanja na $[-1{,}5;\ 1{,}5]$ za x in $[-0{,}5;\ 2{,}5]$ za y.
- Uporabite funkcijo *ga* za iskanje minimuma funkcije.
- Izpišite optimalno rešitev in pripadajočo vrednost funkcije. Preverite, ali algoritem ob vsakem ponovnem zagonu zanesljivo najde minimum funkcije, in utemeljite, zakaj je tako.
- (Dodatno): V začetno populacijo vključite posameznika, postavljenega blizu točke globalnega minimuma, in preverite vpliv dobre inicializacije na uspešnost in hitrost optimizacije.

Univerzitetna založba
Univerze v Mariboru

3 Optimizacija z več cilji

Pri reševanju realnih problemov je pogosto treba upoštevati ne le več spremenljivk, temveč tudi več ciljev, ki so med seboj pogosto v nasprotju. Optimizacija z več cilji omogoča iskanje rešitev, ki vzpostavljajo ustrezno ravnovesje med temi cilji. Namesto ene optimalne rešitve dobimo množico kompromisnih rešitev, med katerimi lahko izbiramo glede na prednostne usmeritve ali zahteve sistema.

Optimizacija z več cilji ali večkriterijska optimizacija je v matematični obliki lahko zapisana kot:

$$\min[f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_n(\boldsymbol{x})] \text{ pri } \boldsymbol{x} \in S, \tag{3.1}$$

kjer je $n > 1$ in S nabor omejitev.

$$S = \{\boldsymbol{x} \in \mathbb{R}^m\colon \boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max};\ h(\boldsymbol{x}) = 0; g(\boldsymbol{x}) \geq 0\}.$$

Ciljni prostor (angl. objective space) je prostor, v katerega preslikamo spremenljivke $\boldsymbol{x}$ s ciljno funkcijo $\boldsymbol{f}(\boldsymbol{x})$. Množica vseh možnih vrednosti ciljev, ki so dosegljive ob upoštevanju omejitev, se imenuje ciljni prostor ali dosegljiva množica in je označena z:

$$\boldsymbol{C} = \{\boldsymbol{y} \in \mathbb{R}^n\colon \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x});\ \boldsymbol{x} \in S\}.$$

Koncept skalarne optimalnosti, značilen za enokriterijske optimizacijske probleme, ni neposredno prenosljiv v večkriterijske sisteme. Zato uvedemo pojem Pareto optimalnosti, ki temelji na primerjavi med rešitvami glede na vse ciljne funkcije hkrati.

Naj bo $\boldsymbol{x}^* \in S$ rešitev večkriterijskega problema. Rečemo, da je $\boldsymbol{x}^*$ (globalni) Pareto optimum, če ne obstaja nobena druga rešitev $\boldsymbol{x} \in S$, za katero velja:

$f_i(\mathbf{x}) \leq f_i(\boldsymbol{x}^*)$ za vse $i \in \{1, \dots, n\}$, in obstaja vsaj en $\boldsymbol{i}$ s strogo neenakostjo.

To pomeni, da ni nobene druge rešitve, ki bi bila boljša (ali enaka) v vseh ciljih in strogo boljša vsaj v enem cilju, kar opisuje naravo kompromisov med cilji.

Definicije:

Šibka optimalna rešitev Pareto $\boldsymbol{x}^* \in S$ je taka, da ne obstaja $\boldsymbol{x} \in S$, za katerega velja:

$f_i(\boldsymbol{x}) < f_i(\boldsymbol{x}^*)$ za vse $i \in \{1, \dots, n\}$.

Stroga optimalna rešitev Pareto $\boldsymbol{x}^* \in S$ (ali kar optimalna rešitev Pareto) je taka, da ne obstaja $\boldsymbol{x} \in S$, za katerega velja:

$$f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*) \text{ za vse } i \in \{1, \dots, n\}, \text{ in } f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}^*) \text{ za vsaj en } j. \tag{3.2}$$

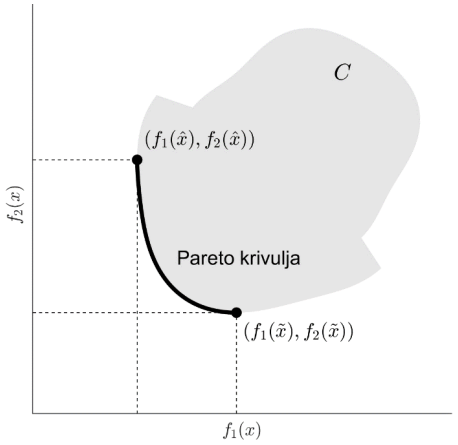
Lokalna optimalna točka Pareto je definirana podobno kot globalna, vendar z omejitvijo na okolico točke $\boldsymbol{x}^*$. Naj bo $B(\boldsymbol{x}^*, \varepsilon)$ krogla z radijem $\varepsilon > 0$. Potem je $\boldsymbol{x}^*$ lokalni Pareto optimum, če ne obstaja $\boldsymbol{x} \in S \cap B(\boldsymbol{x}^*, \varepsilon)$, za katerega velja:

$f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*)$ za vse $i \in \{1, \dots, n\}$, in $f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}^*)$ za vsaj en j.

Pareto fronta:

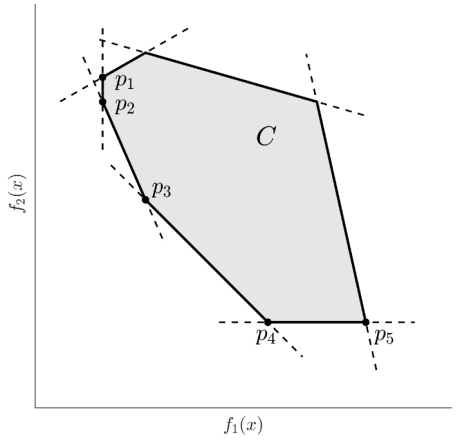
Pareto fronta, imenovana tudi Pareto krivulja ali Pareto ploskev (odvisno od števila ciljev), predstavlja množico vseh nedominiranih točk v ciljnem prostoru. To so točke $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, kjer je $\boldsymbol{x}$ Pareto optimalna rešitev.

Oblika Pareto fronte ponazarja kompromise med različnimi ciljnimi funkcijami. Slika 7 prikazuje primer Pareto krivulje za primer dvokriterijske optimizacije (večkriterijska optimizacija z dvema ciljema), kjer vse točke med $(f_2(\hat{\boldsymbol{x}}), f_1(\hat{\boldsymbol{x}}))$ in $(f_2(\tilde{\boldsymbol{x}}), f_1(\tilde{\boldsymbol{x}}))$ tvorijo Pareto fronto. Te točke so nedominirane oziroma nepodrejene, saj jih po definiciji Pareto optimalnosti ni mogoče izboljšati v enem cilju, ne da bi jih s tem poslabšali v drugem cilju (3.2).



Slika 7: Primer Pareto krivulje

Slika 8 prikazuje primer šibkega in strogega optimuma Pareto. Točki p_1 in p_5 sta šibka optimuma Pareto, točke p_2, p_3 in p_4 pa so strogi optimumi Pareto.



Slika 8: Primer šibkega in strogega optimuma Pareto

3.1 Vaje s področja optimizacij z več cilji

To poglavje vključuje naloge, namenjene razumevanju optimizacije z več cilji, kjer je cilj iskanje kompromisnih rešitev med različnimi merili optimizacije. Vaje vodijo študenta skozi postopke generiranja Pareto front in analize učinkov posameznih parametrov algoritmov na porazdelitev rešitev.

3.1.1 Vaja: Večkriterijska optimizacija procesa odrezavanja

Cilj vaje je izvesti večkriterijsko optimizacijo procesa odrezavanja, pri čemer upoštevamo dva med seboj izključujoča se cilja: minimizirati čas izdelave, saj krajši čas pomeni večjo produktivnost, in minimizirati strošek izdelave, ker nižji stroški povečujejo ekonomsko učinkovitost procesa (3.1). Cilja sta v konfliktu, saj krajši čas izdelave zahteva višje režime obdelave, kar vodi do višjih stroškov.

Odločanje temelji na dveh procesnih parametrih:

- $x_1 = v_f$, podajanje [mm/min.],
- $x_2 = v_c$, hitrost rezanja [m/min.].

Za reševanje problema uporabimo genetski algoritem za večkriterijsko optimizacijo *gamultiobj*, ki vrne množico Pareto optimalnih rešitev.

Ciljni funkciji:

Čas izdelave:

$$f_1(\boldsymbol{x}) = \frac{1}{x_1 \cdot x_2} \tag{3.3}$$

Čas je obratno sorazmeren z volumnom odrezanega materiala na enoto časa.

Strošek:

$$f_2(\boldsymbol{x}) = \frac{10}{x_1 + 0{,}01} + 0{,}3 \cdot x_2^2 \tag{3.4}$$

Strošek izdelave pada s hitrostjo podajanja (hitrejša izdelava) in narašča s hitrostjo rezanja (večja obraba orodja).

Omejitve:

Ponovno uporabimo škatlaste omejitve:

- $50 \leq v_f \leq 150$ [mm/min.],
- $100 \leq v_c \leq 300$ [m/min.].

Izbrane meje definirajo omejitve tehnološkega procesa.

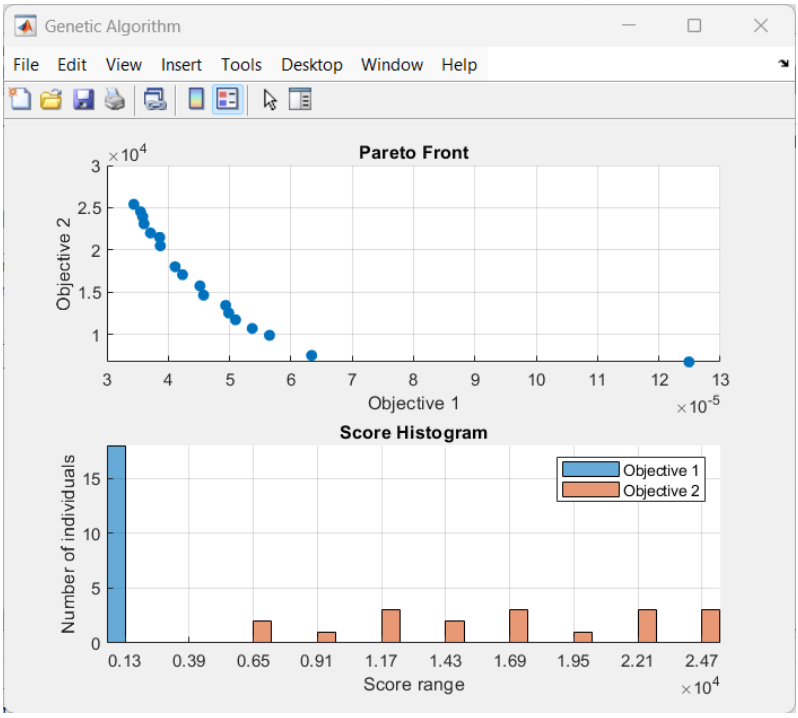
- Ustvarite nov skript z imenom *GA_veckriterijsko.m*.
- V odprt skript prilepite spodnjo kodo.
- Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.

```
%% Optimizacija z več cilji
% Število spremenljivk (dimenzija problema)
steviloSpremenljivk = 2;

% škatlaste omejitve prostora spremenljivk
lb = [50 150];
ub = [100 300];

% Definicija ciljne funkcije (čas izdelave in stroški)
CiljnaFunkcija = @cas_in_stroski;

% možnosti GA
options                                                    =
optimoptions(@gamultiobj,'PlotFcn',{@gaplotpareto,@gaplotscorediversi
ty});

% Zagon večkriterijskega genetskega algoritma (GA)
[x,fval]                                                   =
gamultiobj(CiljnaFunkcija,steviloSpremenljivk,[],[],[],[],lb,ub,[],
options);

%% Ciljna funkcija
function y = cas_in_stroski(x)
y(1) = 1/(x(1) * x(2)); % Čas izdelave
```

```
y(2) = 10/(x(1) + 0.01) + 0.3 * x(2)^2; % Strošek izdelave

end
```

V tej nalogi je optimizacijski problem razširjen na večkriterijsko optimizacijo, kjer sta hkrati upoštevana dva cilja: minimizacija časa izdelave in minimizacija stroškov. To predstavlja bistveno spremembo v primerjavi s prejšnjimi primeri, kjer je bila ciljna funkcija ena (npr. obraba orodja). Ciljna funkcija *cas_in_stroski* zdaj vrača vektor dveh vrednosti, pri čemer prvi element predstavlja čas izdelave, drugi pa strošek, kar omogoča iskanje kompromisnih rešitev.

Namesto funkcije *ga*, ki se uporablja za enokriterijsko optimizacijo, je zdaj uporabljen večkriterijski genetski algoritem *gamultiobj*, ki generira Pareto fronto optimalnih rešitev (3.2). V *options* so definirane dodane funkcije za vizualizacijo: *gaplotpareto* prikazuje razvoj Pareto fronte, *gaplotscorediversity* pa raznolikost populacije rešitev.



Slika 9: Rezultati večkriterijske optimizacije

Tudi število spremenljivk in škatlaste omejitve ostajajo enaki kot v prejšnjih nalogah, kar omogoča neposredno primerjavo med enokriterijskim in večkriterijskim pristopom. Glavna sprememba je torej v ciljni funkciji in v uporabi ustreznega algoritma, ki je prilagojen iskanju več med seboj konkurenčnih ciljev.

Slika 9 v zgornjem grafu prikazuje Pareto fronto oz. množico optimalnih rešitev. Osi predstavljata vrednosti dveh ciljev: čas izdelave (*Objective 1*) (3.3) in strošek izdelave (*Objective 2*) (3.4). Točka na levi zgoraj ima najmanjši čas, a najvišje stroške, medtem ko ima desna spodaj najnižje stroške, a daljši čas izdelave. Prikazana fronta ponazarja kompromis med obema ciljema.

V spodnjem grafu z naslovom *Score Histogram* (Slika 9) je prikazana razpršenost vrednosti posameznih ciljev v populaciji rešitev. Vidimo, da je več posameznikov doseglo enake vrednosti za čas izdelave (*Objective 1*), medtem ko so vrednosti za strošek izdelave (*Objective 2*) bolj razpršene. To pomeni, da je bila populacija bolj uspešna pri optimizaciji časa, medtem ko so stroški pokazali večjo raznolikost med rešitvami.

3.2 Samostojno delo s področja optimizacij z več cilji

V tem poglavju samostojno oblikujete in rešujete večkriterijske optimizacijske naloge na podlagi danih primerov. Cilj je poglobiti razumevanje kompromisov med cilji ter razviti sposobnost interpretacije in predstavitve Pareto optimalnih rešitev.

3.2.1 Samostojno delo: Optimizacija funkcij Binh and Korn

V tej nalogi boste izvedli večkriterijsko optimizacijo funkcij Binh and Korn (poglavje 6.1) z uporabo primernega algoritma. Gre za eno najpogosteje uporabljenih testnih funkcij v večkriterijski optimizaciji, saj vsebuje dve konfliktni ciljni funkciji in dve nelinearni omejitvi. Cilj je poiskati nabor rešitev na Pareto fronti znotraj omejenega prostora.

Zahteve naloge:

- Implementirajte obe ciljni funkciji in obe omejitvi.
- Uporabite večkriterijski optimizacijski algoritem *gamultiobj*.

- Prikažite Pareto fronto ter izpišite nabor optimalnih rešitev v prostoru spremenljivk.
- (Dodatno): Rešite večkriterijski problem z združitvijo obeh ciljnih funkcij v eno samo ciljno funkcijo z uteženo vsoto in optimizacijo izvedite z uporabo genetskega algoritma *ga*. Izvedite optimizacijo za vsaj pet različnih kombinacij uteži in za vsak primer zabeležite optimalno rešitev v prostoru ciljev. Rezultate prikažite grafično v obliki aproksimirane Pareto fronte.

3.2.2 Samostojno delo: Optimizacija funkcij Chakong and Haimes

V tej nalogi boste izvedli večkriterijsko optimizacijo funkcij Chakong and Haimes (poglavje 6.2), ki vključuje dve ciljni funkciji in dve nelinearni omejitvi. Naloga je zasnovana kot test za algoritme večkriterijske optimizacije, saj cilji in omejitve določajo zanimivo Pareto fronto.

Zahteve naloge:

- Implementirajte obe ciljni funkciji in obe omejitvi.
- Uporabite večkriterijski optimizacijski algoritem *paretosearch*.
- Prikažite Pareto fronto ter izpišite nabor optimalnih rešitev v prostoru spremenljivk.
- (Dodatno): Uporabite še algoritem *gamultiobj* ter primerjajte Pareto fronto in čas izvedbe z rezultati algoritma *paretosearch*.

3.2.3 Samostojno delo: Večkriterijska optimizacija testnih funkcij

V tej nalogi boste izvedli večkriterijsko optimizacijo funkcij Kursawe (poglavje 6.3), Deb Bimodal (poglavje 6.4), Kita (poglavje 6.5), DTLZ1 (poglavje 6.6), DTLZ7 (poglavje 6.7). Za vsako funkcijo upoštevajte pripadajoče omejitve.

Zahteve naloge:

- Implementirajte ciljne funkcije in omejitve.
- Uporabite primeren večkriterijski optimizacijski algoritem.
- Prikažite Pareto fronte ter izpišite nabore optimalnih rešitev v prostoru spremenljivk.

Univerzitetna založba
Univerze v Mariboru

4 Algoritem rojev delcev

Algoritem rojev delcev (PSO, angl. Particle Swarm Optimization) je metahevristični optimizacijski algoritem, ki temelji na skupinski inteligenci in sodelovalnem vedenju skupin živali, kot so jate ptic, jate rib ali roji žuželk. Vsak posameznik v takšni skupini, ki ga v algoritmu imenujemo delec, predstavlja eno od možnih rešitev problema. Delci se premikajo po prostoru rešitev, ocenjujejo kakovost svojih položajev in jih sproti prilagajajo na podlagi lastnih izkušenj ter izkušenj drugih.

V naravi številne vrste uspešno rešujejo kompleksne naloge s pomočjo skupinske inteligence. Ptice v jatah med selitvijo usklajeno prilagajajo smer in hitrost leta, da zmanjšajo zračni upor in povečajo energetsko učinkovitost. Ribe z zaznavanjem gibanja sosedov skupaj manevrirajo, da se izognejo plenilcem ali najdejo območja z več hrane. Mravlje s feromonskimi sledmi najdejo najkrajše poti do hrane, čebele pa si lokacije bogatih virov hrane sporočajo s plesom. Takšni sistemi temeljijo na preprostih lokalnih vedenjskih vzorcih posameznikov, a kljub temu dosegajo učinkovito globalno vedenje skupine brez centralnega nadzora.

Delovanje algoritma PSO posnema ta načela. Pri iskanju najboljše rešitve oz. pri premikanju po prostoru spremenljivk upošteva vsak delec tri osnovne vplive: vztrajnost, kognitivno komponento in socialno komponento. Vztrajnostna komponenta pomeni, da delec ohranja del svoje prejšnje hitrosti, podobno kot ptica

ali riba ohranja smer gibanja. Kognitivna komponenta predstavlja zanašanje na lastne izkušnje, saj si delec zapomni najboljšo točko, ki jo je do zdaj obiskal. Socialna komponenta pa odraža vpliv skupine, saj se delec približuje tisti točki, ki jo je kot najboljšo zaznal katerikoli drug delec v roju. Preplet teh vplivov delcem omogoča usmerjeno iskanje optimalne rešitve, pri čemer se učenje posameznika dopolnjuje z učenjem skupine.

Pri vseh populacijskih algoritmih pa je ključno tudi spoznanje, da posamezni člani populacije na začetku iskanja še niso seznanjeni s problemom. Roj sprva nima izkušenj z okolico in ne razpolaga z informacijami o tem, kje bi lahko iskali dobre rešitve. V primeru algoritma rojev delcev (PSO) to pomeni, da imajo delci sicer določen začetni položaj, vendar še nimajo znanja o kakovosti rešitev v prostoru možnih rešitev. Zato je pomembno, da se na začetku učinkovito razporedijo po iskalnem prostoru. To začetno razporeditev imenujemo začetni roj oziroma začetna populacija delcev, ki predstavlja izhodišče celotnega optimizacijskega postopka.

Razpršitev začetnega roja imenujemo inicializacija. Ena možnost inicializacije je, da vsak delec iskanje začne na naključno določeni lokaciji. V prvi ponovitvi algoritma vsak delec na svojem položaju ovrednoti kakovost svoje lokacije oz. prilagojenost na ciljno funkcijo in si lokacijo zapomni kot trenutno najboljšo. In šele nato lahko začne posodabljati svojo hitrost in položaj na podlagi treh ključnih vplivov: vztrajnosti, ki ohranja del prejšnjega gibanja, kognitivne komponente, ki ga usmerja proti njegovi najboljši zaznani točki, in socialne komponente, ki ga privlači k najboljši znani točki v roju.

V nadaljevanju bomo to vedenje opisali z matematičnimi izrazi, ki omogočajo numerično izvedbo algoritma PSO na poljubnih ciljih funkcijah, tudi takšnih s kompleksno večdimenzionalno površino in nelinearnimi omejitvami.

4.1 Matematična formulacija PSO

Algoritem rojev delcev (PSO) je zasnovan na načelih skupinskega vedenja in temelji na izmenjavi informacij med posameznimi delci v roju. Vsak delec predstavlja kandidatno rešitev in se skozi iteracije premika po prostoru rešitev z namenom izboljšanja svoje prilagojenosti glede na ciljno funkcijo. Premiki delcev temeljijo na treh osnovnih vplivih: vztrajnosti, kognitivnem vplivu oz. lastnih izkušnjah in socialnem vplivu oz. izkušnjah drugih delcev v roju.

Matematično lahko vsak delec i v trenutni iteraciji t opišemo s tremi glavnimi količinami:

- $\boldsymbol{x}_i(t)$: položaj je trenutna lokacija delca v prostoru rešitev.
- $\boldsymbol{v}_i(t)$: hitrost je vektorska količina, ki določa smer in velikost premika delca.
- $\boldsymbol{p}_i$: najboljša lastna pozicija je položaj, kjer je delec do sedaj našel najugodnejšo vrednost ciljne funkcije.

Poleg lastnosti delca pa je pomembna še najboljša globalna pozicija $\boldsymbol{g}$, ki je najboljša do sedaj najdena rešitev med vsemi delci.

Celoten postopek optimizacije z algoritmom PSO se izvede iterativno.

Koraki pri izvajanju algoritma PSO

1. **Inicializacija:** Vsak delec se naključno postavi v prostor rešitev in pridobi naključno začetno hitrost.
2. **Evalvacija**: Vsak delec oceni vrednost ciljne funkcije na svoji trenutni poziciji.
3. **Shranjevanje najboljših vrednosti:** Če je nova pozicija boljša od dosedanje najboljše, se ta shrani kot nova osebna najboljša. Če je boljša od globalne, postane nova globalna najboljša.
4. **Posodobitev hitrosti in položaja:** Na podlagi vztrajnosti, kognitivnega vpliva in socialnega vpliva se posodobita hitrost in položaj vsakega delca.
5. **Ponavljanje:** Koraki 2–4 se ponavljajo, dokler ni dosežen ustavitveni kriterij.

Natančno zaporedje izvedbe posameznih korakov lahko nazorno opišemo s psevdokodo. Psevdokodo razdelimo na dva dela: psevdokodo za začetni roj PSO in psevdokodo ponavljanja PSO.

Psevdokoda za začetni roj PSO:

1. naključna postavitev delcev v prostor rešitev
2. za vsak delec:
   2.1. za vsako dimenzijo:
      2.1.1. določi začetni položaj
      2.1.2. določi začetno hitrost

2.2. ovrednoti ciljno funkcijo
2.3. shrani osebni optimum
2.4. če je rezultat boljši od globalnega optimuma:
2.4.1. posodobi globalni optimum

Psevdokoda ponavljanja PSO:

3. dokler ni dosežen ustavitveni kriterij:
3.1. za vsak delec v roju:
3.1.1. posodobi hitrost
3.1.2. posodobi položaj
3.1.3. ovrednoti ciljno funkcijo
3.1.4. če je rezultat boljši od osebnega optimuma:
3.1.4.1. posodobi osebni optimum
3.1.5. če je rezultat boljši od globalnega optimuma:
3.1.5.1. posodobi globalni optimum

4.1.1 Inicializacija začetnega roja

V prvem koraku algoritma PSO je treba določiti začetne vrednosti za vse delce v roju. Vsak delec ima:

- $\boldsymbol{x}_i^{(0)}$: začetni položaj,
- $\boldsymbol{v}_i^{(0)}$: začetno hitrost,
- $\boldsymbol{p}_i^{(0)}$: začetni najboljši položaj.

Inicializacija vključuje naslednje korake za vsak delec $i = 1,2,\dots,n$, kjer je n število delcev:

1. Naključno določi začetni položaj $\boldsymbol{x}_i^{(0)}$ znotraj dovoljene domene spremenljivk:

$$\boldsymbol{x}_{ij}^{(0)} \sim \mathcal{U}\left(\boldsymbol{x}_j^{\min}, \boldsymbol{x}_j^{\max}\right),$$

kjer $j = 1,2,\dots,\mathcal{D}$ označuje dimenzijo problema, $\mathcal{U}$ je enakomerna porazdelitev in $\mathcal{D}$ je število spremenljivk.

2. Naključno določi začetno hitrost $\boldsymbol{v}_i^{(0)}$, npr. iz simetričnega območja:

$$\boldsymbol{v}_{ij}^{(0)} \sim \mathcal{U}\left(-\boldsymbol{v}_j^{\max}, \boldsymbol{v}_j^{\max}\right),$$

kjer je $\boldsymbol{v}_j^{\max} = \alpha\left(\boldsymbol{x}_j^{\max} - \boldsymbol{x}_j^{\min}\right)$, parameter $\alpha \in [0,1]$ pa določa začetno intenzivnost raziskovanja.

3. Določi začetno najboljšo pozicijo:

$$\boldsymbol{p}_i^{(0)} = \boldsymbol{x}_i^{(0)}.$$

Začetni najboljši položaj je v začetnem roju za vsak delec enak trenutnemu položaju.

Opomba:

Izbira začetnih hitrosti vpliva na konvergenco. Prevelike hitrosti povzročijo divergenco, premajhne pa prezgodnjo konvergenco. Namesto povsem naključne inicializacije se lahko uporabi tudi statistična disperzija (npr. latin hypercube), ki izboljša pokritost prostora.

4.1.2 Evalvacija ciljne funkcije

V fazi evalvacije vsak delec izračuna vrednost ciljne funkcije na svojem trenutnem položaju. Ta vrednost predstavlja sposobnost delca, tj. merilo kakovosti rešitve, ki jo predstavlja njegov položaj v prostoru spremenljivk.

Za funkcijo $f_i: \mathbb{R}^n \to \mathbb{R}$ to pomeni, da za vsak delec $\boldsymbol{i}$ izračunamo:

$$f_i = f(\boldsymbol{x}_i),$$

kjer je $\boldsymbol{x}_i$ trenutni položaj delca. Pri minimizacijskih problemih je nižja vrednost funkcije boljša.

4.1.3 Shranjevanje najboljših vrednosti

Po evalvaciji vsak delec primerja trenutno sposobnost z najboljšo, ki jo je do zdaj že dosegel. Če je nova vrednost boljša, delec shrani trenutni položaj kot svoj novi osebni optimum:

$$\text{če } f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i), \text{ potem } \boldsymbol{p}_i = \boldsymbol{x}_i,$$

hkrati pa se posodobi tudi osebna najboljša sposobnost:

$$f(\boldsymbol{p}_i) = f(\boldsymbol{x}_i).$$

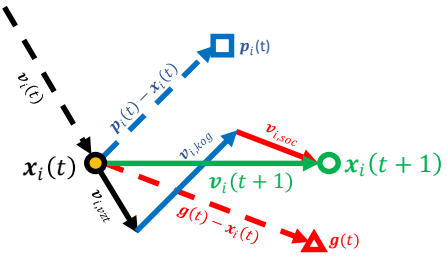
Nato se preveri, ali je katera od osebnih najboljših vrednosti boljša od globalne najboljše vrednosti v roju. Če je tako, se shrani nova globalna najboljša rešitev:

$$\text{če } f(\boldsymbol{p}_i) < f(\boldsymbol{g}), \text{ potem } \boldsymbol{g} = \boldsymbol{p}_i,$$

kjer je $\boldsymbol{g}$ položaj najboljšega delca celotnega roja do zdaj.

4.1.4 Posodobitev hitrosti in položaja

Spremembo položaja delca si lahko predstavljamo kot vektorsko vsoto treh hitrosti, ki vplivajo nanj. Slika 10 ponazarja te vplive v obliki usmerjenih puščic, ki skupaj določijo novo hitrost $\boldsymbol{v}_i(t+1)$ in novi položaj $\boldsymbol{x}_i(t+1)$ delca i.



Slika 10: Sprememba hitrosti in položaja delca kot vektorska vsota vztrajnostne, kognitivne in socialne komponente

Nova hitrost delca $\boldsymbol{v}_i(t+1)$ se določi z vsoto treh komponent po enačbi:

$$\boldsymbol{v}_i(t+1) = r \cdot \omega \cdot \boldsymbol{v}_i(t) + r_1 \cdot c_1 \cdot \big(\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\big) + r_2 \cdot c_2 \cdot \big(\boldsymbol{g}(t) - \boldsymbol{x}_i(t)\big), \tag{4.1}$$

kjer so:

- ω: koeficient vztrajnosti, ki določa težnjo delca, da ohrani trenutno smer gibanja;
- c_1, c_2: koeficienta pospeška, ki uravnavata vpliv osebne in kolektivne izkušnje;
- r, r_1, r_2: naključna števila iz enakomerne porazdelitve v intervalu [0,1], ki prispevajo k stohastični naravi algoritma.

Vsaka od komponent ima poseben pomen:

- $\boldsymbol{v}_{i,vzt}(t+1) = r \cdot \omega \cdot \boldsymbol{v}_i(t)$: vztrajnostna komponenta spodbuja gladko gibanje in preprečuje prehitro spreminjanje smeri;
- $\boldsymbol{v}_{i,kog}(t+1) = r_1 \cdot c_1 \cdot \big(\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\big)$: kognitivna komponenta usmerja delca proti točki, kjer je sam zaznal najboljši rezultat;
- $\boldsymbol{v}_{i,soc}(t+1) = r_2 \cdot c_2 \cdot \big(\boldsymbol{g}(t) - \boldsymbol{x}_i(t)\big)$: socialna komponenta privlači delec h globalno najboljši točki, ki jo je do sedaj našel roj.

Ko je hitrost določena, se položaj delca posodobi po enačbi:

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1). \tag{4.2}$$

S tem delec naredi naslednji korak v prostoru rešitev.

Opomba:

Izbira parametrov ω, c_1 in c_2 bistveno vpliva na delovanje algoritma. Visok ω spodbuja raziskovanje, nizek vodi v hitrejšo konvergenco. Razmerje med c_1 in c_2 določa, ali je posameznik bolj samozavesten ali bolj posluša skupino. V praksi se pogosto uporabljajo metode, kot je Clerc-Kennedyjeva korekcija, kjer se konstante izračunajo za zagotovitev stabilne konvergence.
Določitev parametrov ω, c_1 in c_2 po metodi Clerc-Kennedy vključuje določitev konstant Chi (X), Phi (Φ), Kapa (K):

$$X = \frac{2K}{\left|2 - \Phi - \sqrt{\Phi^2 - 4\Phi}\right|},$$
$$0 \leq \mathrm{K} \leq 1,$$

$$\Phi = \Phi_1 + \Phi_2 \geq 4,$$

pri čemer je:

$$\omega = X,$$

$$c_1 = X\Phi_1,$$

$$c_2 = X\Phi_2.$$

4.1.5 Ponavljanje algoritma PSO

Po inicializaciji, evalvaciji in shranjevanju najboljših vrednosti delcev v začetnem roju sledi ponavljajoči se del algoritma PSO, kjer delci postopoma izboljšujejo svoje rešitve. Vsaka ponovitev oz. iteracija algoritma vključuje naslednje korake:

1. Posodobitev hitrosti delca.
2. Posodobitev položaja delca z uporabo nove hitrosti.
3. Evalvacija ciljne funkcije na novi lokaciji.
4. Shranjevanje osebnih in globalnih optimumov, če so se izboljšali.

Ti koraki se ponavljajo toliko časa, dokler ni dosežen ustavitveni kriterij. Kot ustavitveni kriterij se pogosto uporablja:

- največje število iteracij,
- konvergenca (zadostna nespremenljivost rezultata),
- dovolj nizka vrednost ciljne funkcije,
- ali kombinacija več kriterijev.

Doseženo največje število iteracij

Algoritem se ustavi, ko število opravljenih iteracij t doseže vnaprej določeno mejo t_{max}:

$$t \geq t_{max},$$

kjer je:

- t: trenutna iteracija,
- t_{max}: največje dovoljeno število iteracij.

Konvergenca

Če se vrednost ciljne funkcije ali najboljša rešitev v zaporednih iteracijah spremeni za manj kot vnaprej določeni prag ε, štejemo, da je algoritem konvergiral:

$$\left|f\left(\boldsymbol{x}^{(\mathrm{t})}\right) - f\left(\boldsymbol{x}^{(\mathrm{t-k})}\right)\right| < \varepsilon \text{ ali } \left\|\boldsymbol{x}^{(\mathrm{t})} - \boldsymbol{x}^{(\mathrm{t-k})}\right\| < \varepsilon, \tag{4.3}$$

kjer je:

- $\boldsymbol{x}^{(\mathrm{t})}$: najboljša znana rešitev v iteraciji t,
- $f(\cdot)$: ciljna funkcija,
- k: št. korakov iteracije,
- ε: vnaprej določena toleranca (npr. 10^{-6}).

Dosežena dovolj nizka vrednost ciljne funkcije

Postopek optimizacije se lahko ustavi tudi, ko vrednost ciljne funkcije doseže vnaprej določeno minimalno vrednost f_{min}, ki velja za sprejemljivo rešitev:

$$f\left(\boldsymbol{x}^{(\mathrm{t})}\right) \leq f_{\mathrm{min}}.$$

4.2 Vaje s področja algoritma PSO

Vaje vodijo študenta skozi postopno sestavljanje algoritma rojev delcev (PSO), od osnovne inicializacije do modularne funkcijske oblike. Na začetku se ustvari začetni roj delcev z naključno določenimi položaji znotraj podanih mej, hitrostmi, ocenami ciljne funkcije in začetnimi osebnimi optimumi. Sledi ovrednotenje ciljne funkcije, določitev globalnega optimuma ter izvedba iterativnega postopka, kjer vsak delec posodablja svojo hitrost in položaj glede na osebne in globalne najboljše rešitve. Vizualizacija gibanja delcev in konvergence omogoča vpogled v učinkovitost optimizacije. Algoritem se nato modularizira, tako da se ciljna funkcija in algoritem

PSO preneseta v zunanji funkciji, ki omogočata fleksibilno uporabo z različnimi vhodnimi parametri in funkcijami. V zaključni fazi se funkcija PSO prikliče iz ločenega skripta, kar omogoča enostavno prilagoditev različnim optimizacijskim problemom. Takšna zasnova sledi principu modularnosti, podobno kot so definirane funkcije v knjižnici za globalno optimizacijo v okolju MATLAB.

4.2.1 Vaja: Inicializacija začetnega roja PSO

V tej vaji boste pripravili začetni roj delcev za algoritem PSO. Cilj je ustvariti 100 delcev, vsakega z naključnim položajem v prostoru spremenljivk, začetno hitrostjo nič, oceno kakovosti rešitve (ovrednotena ciljna funkcija) in shranjenim najboljšim osebnim rezultatom.

- Ustvarite nov skript z imenom *inicializacija_roja.m*.
- V odprt skript prilepite spodnjo kodo.

```
%% Inicializacija roja delcev - vaja
% V tej vaji boste dopolnili manjkajoče dele kode za inicializacijo
PSO.

clear;
clc;
close all;
```

Ta del kode izbriše vse spremenljivke v delovnem prostoru (*clear*), zapre vsa odprta grafična okna (*close all*) in počisti ukazno okno (*clc*).

- Dodajte naslednje vrstice.

```
%% Parametri algoritma
nvar = 2;               % Število spremenljivk (dimenzija problema)
vel_roj = 100;          % Velikost roja (število delcev)
```

S temi parametri določite, koliko neodvisnih spremenljivk bo imel vsak delec (*nvar*) in koliko delcev bo sodelovalo v algoritmu (*vel_roj*).

➢ Dodajte naslednje vrstice.

```
%% Omejitve prostora spremenljivk
lb = [-100 -100];       % Spodnje meje
ub = [100 100];         % Zgornje meje
```

Tukaj določite meje prostora iskanja. *lb* je spodnja meja za prvo in drugo spremenljivko delca in *ub* je zgornja meja za prvo in drugo spremenljivko delca. Vsak delec mora imeti začetni položaj znotraj teh mej.

➢ Dodajte naslednje vrstice.

```
%% Struktura posameznega delca
delec.pol = [];
delec.hitrost = [];
delec.sposobnost = [];
delec.naj_pol = [];
delec.naj_sposobnost = [];
```

Vsak delec ima svoj trenutni položaj v prostoru spremenljivk (*delec.pol*), hitrost, s katero se bo premikal (*delec.hitrost*), sposobnost oz. vrednost ciljne funkcije (*delec.sposobnost*), najboljši osebni položaj do sedaj (*delec.naj_pol*) ter sposobnost na tem najboljšem položaju (*delec.naj_sposobnost*). Spremenljivka *delec* je definirana kot struktura (angl. struct), ki se v MATLAB-u uporablja za organizirano in pregledno shranjevanje podatkov z različnimi tipi in imeni polj v eni spremenljivki.

➢ Dodajte naslednje vrstice.

```
%% Globalni optimum
naj_delec_sposobnost = inf;
naj_delec_pol = zeros(nvar,1);
```

Na začetku optimizacije še nimamo dobrih rešitev, zato globalni optimum (*naj_delec_sposobnost*) nastavimo na zelo slabo vrednost (neskončno oziroma *inf*). Ta vrednost se bo med inicializacijo posodabljala, če kateri koli delec ponudi boljšo rešitev. Za spremenljivko *naj_delec_pol*, ki predstavlja lokacijo globalnega minimuma, vnaprej rezerviramo pomnilnik (angl. preallocate) z ustvarjanjem vektorja dimenzije *nvar*, pri čemer vse vrednosti inicializiramo z ničlo.

➢ Dodajte naslednje vrstice.

```
%% Inicializacija vseh delcev
delec = repmat(delec, vel_roj, 1);
```

Funkcija *repmat* naredi kopijo strukture *delec* po vrsticah *vel_roj*-krat. S tem ustvarimo celoten roj delcev, v katerem ima vsak delec enako strukturo, kot smo jo določili zgoraj.

➢ Dodajte naslednje vrstice.

```
for i = 1 : vel_roj
    for j = 1 : nvar
        % Dopolni: naključni začetni položaj znotraj meja


        % Dopolni: začetna hitrost = 0

    end
end
```

V tem koraku smo dodali dve *for* zanki. V zunanji zanki poteka inicializacija vseh delcev v roju, pri čemer spremenljivka *i* predstavlja posamezni delec, zanka pa se izvede tolikokrat, kot je velikost roja (*vel_roj*). Če imamo na primer 100 delcev, bo ta zanka izvedena 100-krat, enkrat za vsak delec.

Znotraj zunanje zanke se nahaja še notranja zanka, ki poteka po vseh dimenzijah, ki jih ima vsak delec (*nvar*). Spremenljivka *j* predstavlja indeks posamezne dimenzije prostora rešitev. Če ima naš optimizacijski problem dve spremenljivki, bomo vsak delec obravnavali v dveh dimenzijah, npr. koordinati x in y, ki določata položaja delca.

Znotraj teh dveh zank bomo v nadaljevanju za vsak delec in vsako njegovo spremenljivko določili začetne vrednosti za položaj in hitrost. Torej bomo za vsako kombinacijo *i* in *j* izvedeli inicializacijo posamezne komponente položaja (x, y) in hitrosti (v_x, v_y).

Izziv za samostojno delo: Cilj je za vsak delec in vsako dimenzijo določiti naključni začetni položaj znotraj omejitev domene iskanja ter nastaviti začetno hitrost delca na nič. Na začetku algoritma delci še niso seznanjeni s problemom in nimajo izkušenj, na podlagi katerih bi lahko sprejeli odločitev, kam naj se premikajo, zato naj bo njihova začetna hitrost enaka 0.

- Določite naključni začetni položaj delca znotraj omejitev.
- Nastavite začetno hitrost vseh delcev na 0.

Namig: Za naključno vrednost med spodnjo in zgornjo mejo uporabite funkcijo *rand*, ki generira naključno število med 0 in 1.

Po inicializaciji delcev lahko preverimo, kako so se delci razporedili po prostoru spremenljivk. Zapišimo funkcijo, ki izriše začetne položaje vseh delcev v dveh dimenzijah. Vizualizacija nam pomaga oceniti, ali so delci ustrezno razpršeni po iskalnem prostoru ali pa so skoncentrirani le na določenem območju. Ustrezna začetna razpršenost je pomembna za uspešnost optimizacije, saj zagotavlja, da roj učinkovito preišče celoten prostor rešitev.

- Ustvarite nov skript z imenom *test_disperzije.m*.
- V odprt skript prilepite spodnjo kodo.

```
function test_disperzije(delec)
    polozaji = reshape([delec.pol], [], length(delec))';
    figure;
    scatter(polozaji(:,1), polozaji(:,2), 20, 'filled');
    xlabel('x_1');
    ylabel('x_2');
    title('Začetna razpršenost roja');
    axis equal;
    grid on;
end
```

Za klic funkcije znotraj skripta *inicializacija_roja.m* mora biti skript *test_disperzije.m* shranjen v isti mapi kot skript *inicializacija_roja.m*.

➢ V skript *inicializacija_roja.m* na koncu dodajte naslednje vrstice.

```
% Klic funkcije test_disperzije
test_disperzije(delec);
```

➢ Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.

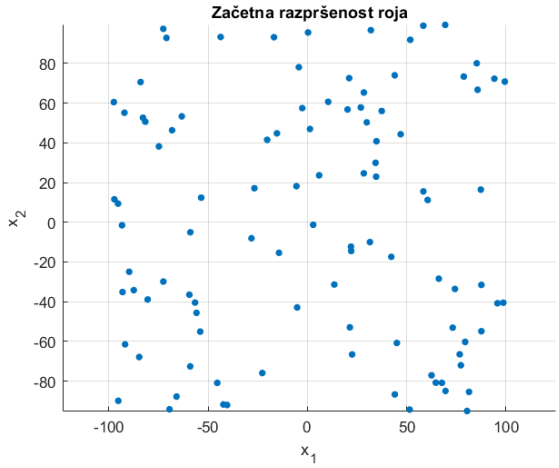
Rešitev:

```
%% Inicializacija roja delcev - vaja
% V tej vaji boste dopolnili manjkajoče dele kode za inicializacijo
PSO.

clear;
clc;
close all;

%% Parametri algoritma
nvar = 2;               % Število spremenljivk (dimenzija problema)
vel_roj = 100;          % Velikost roja (število delcev)

%% Omejitve prostora spremenljivk
lb = [-100 -100];       % Spodnje meje
ub = [100 100];         % Zgornje meje

%% Struktura posameznega delca
delec.pol = [];
delec.hitrost = [];
delec.sposobnost = [];
delec.naj_pol = [];
delec.naj_sposobnost = [];

%% Globalni optimum
naj_delec_sposobnost = inf;
naj_delec_pol = zeros(nvar,1);

%% Inicializacija vseh delcev
delec = repmat(delec, vel_roj, 1);

for i = 1 : vel_roj
    for j = 1 : nvar
```

```
        % Dopolni: naključni začetni položaj znotraj meja
        delec(i).pol(j) = lb(j) + rand * (ub(j) - lb(j));

        % Dopolni: začetna hitrost = 0
        delec(i).hitrost(j) = 0;
    end

end

test_disperzije(delec);
```

Koda ustvari roj delcev za PSO tako, da vsakemu delcu določi naključni začetni položaj znotraj podanih mej in mu nastavi začetno hitrost na nič. Globalni optimum se na začetku postavi na neskončno vrednost, njegova lokacija pa se pripravi z vnaprej dodeljenim pomnilnikom. Funkcija *test_disperzije.m* preveri razpršenost delcev v prostoru rešitev.

Slika 11 prikazuje začetno razpršenost roja delcev v dveh dimenzijah prostora rešitev (x_1 in x_2). Izris je rezultat funkcije *test_disperzije.m*, ki vizualno prikaže, ali so delci enakomerno porazdeljeni znotraj podanih meja iskanja. Vidimo, da so delci dobro razpršeni po celotnem območju, kar je ključno za učinkovito iskanje globalnega optimuma v zgodnjih fazah optimizacije.



Slika 11: Začetna razpršenost roja delcev v omejenem iskalnem prostoru

4.2.2 Vaja: Evalvacija ciljne funkcije v začetnem roju

V tej vaji boste za vsak delec izračunali vrednost ciljne funkcije. Uporabili boste dvodimenzionalno kvadratno funkcijo.

- Odprite skript *inicializacija_roja.m*.
- Pred koncem zunanje zanke zapišite ukaz za ovrednotenje dvodimenzionalne funkcije Sphere (poglavje 5.1).
- Kodo zaženite s klikom na gumb **Run** ali s pritiskom tipke **F5**.
- V delovnem prostoru (**Workspace**) poglejte, kako so se posodobile lastnosti delcev.

Namig: Ukaz za ovrednotenje ciljne funkcije zapišite v vrstico pod komentarjem »Dopolni: ovrednotenje ciljne funkcije«, kot je razvidno v spodnjem odseku kode.

```
for i = 1 : vel_roj
    for j = 1 : nvar
        % Dopolni: naključni začetni položaj znotraj meja
        delec(i).pol(j) = lb(j) + rand * (ub(j) - lb(j));
        % Dopolni: začetna hitrost = 0
        delec(i).hitrost(j) = 0;
    end
    % Dopolni: ovrednotenje ciljne funkcije


end
```

Rešitev:

```
for i = 1 : vel_roj
    for j = 1 : nvar
        % Dopolni: naključni začetni položaj znotraj meja
        delec(i).pol(j) = lb(j) + rand * (ub(j) - lb(j));
        % Dopolni: začetna hitrost = 0
        delec(i).hitrost(j) = 0;
    end
    % Dopolni: ovrednotenje ciljne funkcije
    delec(i).sposobnost = sum(delec(i).pol.^2);
end
```

V tem delu kode v polje *sposobnost* zapišemo vrednost ciljne funkcije (Sphere), ki oceni kakovost trenutne rešitve in se uporablja pri nadaljnjem iskanju globalnega optimuma.

Pomembno: Pika pred kvadratom v zapisu *pol.^2* pomeni elementno potenciranje, kar pomeni, da se vsaka komponenta vektorja posebej kvadrira. V našem primeru je *pol* dvodimenzionalni vektor z dvema elementoma, ki se najprej kvadrirata, nato pa se njuna kvadrata seštejeta z ukazom *sum*.

100x1 struct with 5 fields

Fields	pol	hitrost	sposobnost	naj_pol	naj_sposobnost
1	[62.9447,81.1584]	[0,0]	1.0549e+04	[]	[]
2	[-74.6026,82.6752]	[0,0]	1.2401e+04	[]	[]
3	[26.4718,-80.4919]	[0,0]	7.1797e+03	[]	[]

Slika 12: Struktura delec po inicializaciji položaja, hitrosti in ovrednotenju ciljne funkcije

Slika 12 prikazuje strukturo *delec*, ki vsebuje 100 delcev, vsak s petimi polji: *pol*, *hitrost*, *sposobnost*, *naj_pol* in *naj_sposobnost*. Polje *pol* vsebuje naključno določen položaj v 2D--prostoru, *hitrost* je inicializirana na [0, 0], kar pomeni, da delci na začetku mirujejo. V polju *sposobnost* je izračunana vrednost ciljne funkcije (funkcija Sphere) na osnovi trenutnega položaja. Polji *naj_pol* in *naj_sposobnost* sta prazni, saj se osebni optimumi še niso določali. Slika potrjuje, da je inicializacija roja pravilno izvedena.

4.2.3 Vaja: Shranjevanje najboljših vrednosti v začetnem roju

V tej vaji boste za vsak delec shranili njegov trenutni položaj kot najboljši osebni položaj, shranili vrednost ciljne funkcije kot osebno najboljšo sposobnost in posodobili globalni optimum, če je trenutni delec boljši od vseh prejšnjih.

- Shranite trenutni položaj delca kot njegov osebni najboljši položaj.
- Shranite trenutno sposobnost kot njegovo osebno najboljšo sposobnost.
- Shranite globalni optimum, če je trenutni delec boljši od globalnega optimuma.

Namig: Ukaza za nastavitev osebnega optimuma (*naj_pol* in *naj_sposobnost*) zapišite v vrstico pod komentarjem »Dopolni: nastavitev osebnega optimuma«. Podobno zapišite pogoj za posodobitev globalnega optimuma pod komentarjem »Dopolni:

posodobitev globalnega optimuma« znotraj *if* stavka, kot je prikazano v spodnjem odseku kode. Z *if* stavkom preverite, ali je trenutna sposobnost manjša od dosedanjega globalnega optimuma.

```
    end

    % Dopolni: evalvacija ciljne funkcije (Sphere function)
    delec(i).sposobnost = sum(delec(i).pol.^2);

    % Dopolni: nastavitev osebnega optimuma


    % Dopolni: posodobitev globalnega optimuma
    if delec(i).sposobnost < naj_delec_sposobnost


    end
```

Rešitev:

```
%% Inicializacija roja delcev - vaja
% V tej vaji boste dopolnili manjkajoče dele kode za inicializacijo
PSO.

clear;
clc;
close all;

%% Parametri algoritma
nvar = 2;                % Število spremenljivk (dimenzija problema)
vel_roj = 100;           % Velikost roja (število delcev)

%% Omejitve prostora spremenljivk
lb = [-100 -100];        % Spodnje meje
ub = [100 100];          % Zgornje meje

%% Struktura posameznega delca
delec.pol = [];
delec.hitrost = [];
```

```
delec.sposobnost = [];
delec.naj_pol = [];
delec.naj_sposobnost = [];

%% Globalni optimum
naj_delec_sposobnost = inf;
naj_delec_pol = zeros(nvar,1);

%% Inicializacija vseh delcev
delec = repmat(delec, vel_roj, 1);

for i = 1 : vel_roj
    for j = 1 : nvar
        % Dopolni: naključni začetni položaj znotraj meja
        delec(i).pol(j) = lb(j) + rand * (ub(j) - lb(j));

        % Dopolni: začetna hitrost = 0
        delec(i).hitrost(j) = 0;
    end

    % Dopolni: evalvacija ciljne funkcije (Sphere function)
    delec(i).sposobnost = sum(delec(i).pol.^2);

    % Dopolni: nastavitev osebnega optimuma
    delec(i).naj_pol = delec(i).pol;
    delec(i).naj_sposobnost = delec(i).sposobnost;

    % Dopolni: posodobitev globalnega optimuma
    if delec(i).sposobnost < naj_delec_sposobnost
        naj_delec_sposobnost = delec(i).sposobnost;
        naj_delec_pol = delec(i).pol;
    end
end
```

V delu kode, kjer nastavljamo osebni optimum, se trenutni položaj in sposobnost vsakega delca shranita kot njegov najboljši znani položaj in vrednost. V nadaljevanju se preveri, ali je trenutna sposobnost delca boljša od trenutnega globalnega optimuma. Če je, se globalni optimum ustrezno posodobi. Takšna inicializacija zagotavlja, da ima vsak delec svojo referenčno vrednost, vsem delcem v roju pa je znana najboljša začetna rešitev.

100x1 struct with 5 fields

Fields	pol	hitrost	sposobnost	naj_pol	naj_sposobnost
1	[28.8636,-24.2781]	[0,0]	1.4225e+03	[28.8636,-24....	1.4225e+03
2	[62.3161,6.5651]	[0,0]	3.9264e+03	[62.3161,6.5...	3.9264e+03
3	[-29.8546,87.8003]	[0,0]	8.6002e+03	[-29.8546,87....	8.6002e+03
4	[75.1886,10.0313]	[0,0]	5.7539e+03	[75.1886,10....	5.7539e+03

Slika 13: Struktura delec po posodobitvi osebnih optimumov delcev

Slika 13 prikazuje strukturo *delec* po posodobitvi osebnih optimumov. Vsak delec ima poleg trenutnega položaja in sposobnosti shranjeni tudi polji *naj_pol* in *naj_sposobnost*, ki predstavljata njegov osebni najboljši položaj in ustrezno vrednost.

Slika 14 prikazuje delovni prostor po posodobitvi globalnih optimumov, kjer sta med spremenljivkami ovrednotena tudi globalni najboljši položaj (*naj_delec_pol*) in najnižja dosežena vrednost ciljne funkcije (*naj_delec_sposobnost*), ki predstavljata trenutno najboljšo rešitev celotnega roja.

Workspace

Name	Value
delec	*100x1 struct*
i	100
j	2
lb	[-100,-100]
naj_delec_pol	[0.0045,-4.0156]
naj_delec_sposobnost	16.1248
nvar	2
ub	[100,100]
vel_roj	100

Slika 14: Delovni prostor z rezultati optimizacije

4.2.4 Vaja: Ponavljanje algoritma PSO

V tej vaji boste implementirali ponavljajoči del algoritma PSO, ki je ključen za iskanje optimalne rešitve skozi več iteracij. Znotraj vsake iteracije bo vsak delec posodobil svojo hitrost, izračunal nov položaj, ocenil novo vrednost ciljne funkcije, posodobil svoj osebni optimum, če je našel boljšo rešitev, in posodobil globalni optimum, če je njegova rešitev najboljša v roju.

- Skript *inicializacija_roja.m* shranite z novim imenom npr. *algoritem_PSO.m*.
- Dodajte novo spremenljivko *iterations*, ki določa število ponovitev algoritma.
- Število ponovitev algoritma nastavite na **100**.

Rešitev:

```
%% Parametri algoritma
nvar = 2;               % Število spremenljivk (dimenzija problema)
iterations = 100;       % Število ponovitev (iteracij)
vel_roj = 100;          % Velikost roja (število delcev)
```

Ker PSO deluje iterativno, v vsaki ponovitvi delci posodobijo svoje položaje in hitrosti ter ovrednotijo ciljno funkcijo. S parametrom *iterations* določimo, kolikokrat naj se ta postopek izvede.

- Na koncu kode dodajte zanko po številu iteracij.
- Znotraj zanke po iteracijah dodajte zanko po vseh delcih v roju.

Rešitev:

```
%% Zanka po vseh spremembah položaja roja

for iter = 1 : iterations

% Zanka po vseh delcih: izračun položaja in ciljne funkcije vsakega
delca
    for i = 1 : vel_roj

    end

end
```

Zanka po številu iteracij *(iter)* določa, kolikokrat se izvede posodobitev vseh delcev, notranja zanka *(i)* pa obravnava vsak delec posebej z namenom iskanja boljše rešitve.

4.2.5 Vaja: Posodobitev hitrosti in položaja

Znotraj notranje zanke po delcih implementirajte posodobitev hitrosti in položaja (4.1) in (4.2).

- V zanko vključite posodobitev hitrosti po formuli PSO.
- Za vztrajnostni koeficient uporabite vrednost $\omega = 0{,}99$.
- Za kognitivni pospešek uporabite $c_1 = 2$.
- Za socialni pospešek uporabite $c_2 = 2$.
- V zanko vključite posodobitev položaja delca.

Rešitev:

```
%% Konstante PSO
w = 0.99;      % vztrajnostni faktor
c1 = 2;        % vpliv osebnega optimuma
c2 = 2;        % vpliv globalnega optimuma
```

Te vrstice zapišemo v začetnem delu kode pri spremenljivkah algoritma.

```
for iter = 1 : iterations

    % Zanka po vseh delcih: izračun položaja in ciljne funkcije vsakega
delca
    for i = 1 : vel_roj

        for j = 1 : nvar

            delec(i).hitrost(j) = ( ...
                w*rand*delec(i).hitrost(j)                          ...
% sprememba trenutne hitrosti
                + c1*rand*(delec(i).naj_pol(j) - delec(i).pol(j)) ...
% sprememba hitrosti v smeri osebnega optimuma
                +   c2*rand*(naj_delec_pol(j)   -   delec(i).pol(j)));
% sprememba hitrosti v smeri globalnega optimuma

            delec(i).pol(j) = delec(i).pol(j) + delec(i).hitrost(j);
% sprememba položaja delca

        end

    end

end
```

V tem delu vsak delec posodobi obe komponenti svoje hitrosti glede na vztrajnost, osebni in globalni optimum, nato pa izračuna svoj nov položaj s premikom v smeri te hitrosti.

4.2.6 Vaja: Shranjevanje najboljših vrednosti

V nadaljevanju boste ovrednotili ciljno funkcijo in po potrebi posodobili najboljše vrednosti delcev in roja.

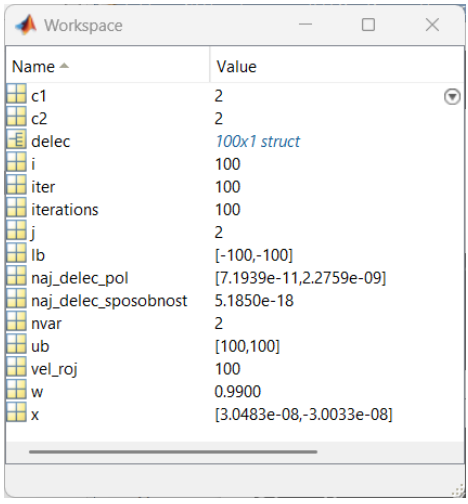
- V zanko vključite ovrednotenje ciljne funkcije.
- Vključite posodobitev osebnega optimuma, če je novi rezultat boljši.
- Vključite posodobitev globalnega optimuma, če je novi rezultat najboljši v roju.

Rešitev:

```
%% Zanka po vseh spremembah položaja roja

for iter = 1 : iterations
    % Zanka po vseh delcih: izračun položaja in ciljne funkcije vsakega
delca
    for i = 1 : vel_roj

        for j = 1 : nvar

            delec(i).hitrost(j) = ( ...
                w*rand*delec(i).hitrost(j) ...
       % sprememba trenutne hitrosti
                + c1*rand*(delec(i).naj_pol(j) - delec(i).pol(j)) ...
       % sprememba hitrosti v smeri osebnega optimuma
                +   c2*rand*(naj_delec_pol(j)   -   delec(i).pol(j)));
       % sprememba hitrosti v smeri globalnega optimuma

            delec(i).pol(j) = delec(i).pol(j) + delec(i).hitrost(j); %
sprememba položaja delca
        end

        % Ovrednotena ciljna funkcija za vsak delec
        delec(i).sposobnost = sum(delec(i).pol.^2);

        % Posodobitev najboljših lastnosti delca
```

```
        if delec(i).sposobnost < delec(i).naj_sposobnost
       % če je boljši kot je bil do zdaj

            delec(i).naj_sposobnost = delec(i).sposobnost;

            delec(i).naj_pol = delec(i).pol;
        end

        % Posodobitev lastnosti najboljšega delca
        if delec(i).sposobnost < naj_delec_sposobnost
       % če je najden boljši delec

            naj_delec_pol = delec(i).pol;

            naj_delec_sposobnost = delec(i).sposobnost;

        end
    end
end
```

V tem delu kode vsak delec po premiku ovrednoti ciljno funkcijo, nato primerja novo vrednost z dosedanjim najboljšim osebnim rezultatom ter ga po potrebi posodobi. Če pa je novi rezultat najboljši v celotnem roju, se posodobi še globalni optimum.



Slika 15: Delovni prostor po optimizaciji funkcije Sphere z najdenim globalnim optimumom

S tem smo zagradili osnovno kodo za optimizacijo z algoritmom PSO. Algoritem uspešno najde minimum ciljne funkcije in ga zapiše v spremenljivko *naj_delec_sposobnost*. Pripadajoče vrednosti argumentov funkcije se zapišejo v spremenljivko *naj_delec_pol* (Slika 15).

4.2.7 Vaja: Stekanje algoritma

V tem delu boste dodali spremenljivko za shranjevanje najboljših vrednosti skozi iteracije in jih prikazali v časovnem grafu, ki bo prikazoval stekanje oz. konvergenco algoritma (4.3).

- V vsaki iteraciji shranite sposobnost najboljšega delca.
- V vsaki iteracij shranite povprečno sposobnost roja.
- V vsaki iteracij shranite sposobnost najslabšega delca.
- Dodajte izris položajev delcev v vsaki iteraciji in izpis konvergence po koncu izvajanja.

Rešitev:

```
%% Prelokacija matrik za hitrejše procesiranje
konvergenca = zeros(iterations,1);  % konvergenca - najboljši delec
roja
kon_avg = zeros(iterations,1);      % konvergenca - povprečje roja
kon_max = zeros(iterations,1);       % konvergenca - najslabši delec
roja
Xpol = zeros(vel_roj,1);            % koordinata za izris delca
Ypol = zeros(vel_roj,1);            % koordinata za izris delca
```

V tem delu vnaprej ustvarimo matrike za shranjevanje rezultatov med izvajanjem algoritma. Na ta način rezerviramo pomnilnik, kar pohitri izvajanje kode. Vektorji *konvergenca*, *kon_avg* in *kon_max* beležijo potek optimizacije, *Xpol* in *Ypol* pa shranjujeta položaje delcev za grafični prikaz.

```
% Izris in konvergenca
Xpol(i) = delec(i).pol(1);          % koordinata 1. delca se shrani v
spremenljivko za izris
Ypol(i) = delec(i).pol(2);          % koordinata 2. delca se shrani v
spremenljivko za izris
```

V tem delu shranimo trenutno pozicijo posameznega delca v vektorja *Xpol* in *Ypol*, ki ju uporabimo za sproten izris položajev delcev v prostoru rešitev. Tako lahko vizualno spremljamo gibanje roja skozi iteracije.

```
%% Izris gibanja delcev
clf;
plot(Xpol, Ypol, '*');
axis([lb(1) ub(1) lb(2) ub(2)]);
title(['Položaji delcev – iteracija ', num2str(iter)]);
xlabel('x(1)');
ylabel('x(2)');
pause(0.01);
```

V tem delu sproti izrisujemo gibanje delcev v prostoru rešitev. *clf* izbriše prejšnji izris, *plot* nariše trenutne položaje vseh delcev, *axis* določi vidno območje, *title*, *xlabel* in *ylabel* označijo graf, *pause* pa omogoči animacijo z rahlo zakasnitvijo med posameznimi iteracijami.

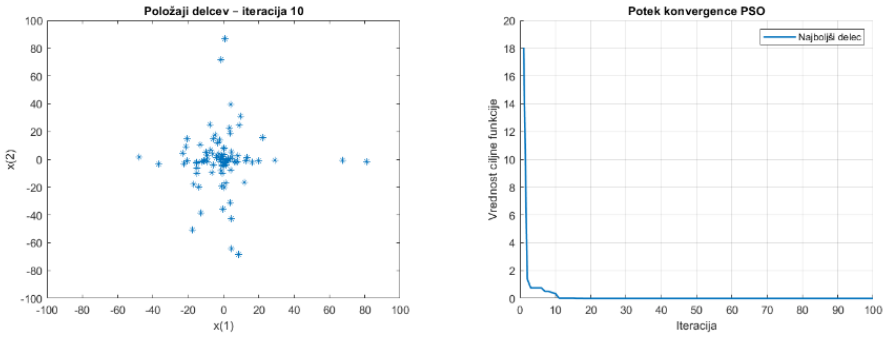
```
% Shranjevanje sposobnosti delcev
konvergenca(iter) = naj_delec_sposobnost;
kon_avg(iter) = mean([delec.sposobnost]);
kon_max(iter) = max([delec.sposobnost]);
```

V tem delu beležimo rezultate optimizacije v vsaki iteraciji: *konvergenca* shrani trenutno najboljšo vrednost ciljne funkcije, *kon_avg* povprečno sposobnost vseh delcev, *kon_max* pa najslabšo (največjo) vrednost ciljne funkcije med delci. Ti podatki bodo uporabljeni za izris poteka konvergence algoritma.

```
%% Izris konvergence
figure('Name','Konvergenca PSO');
hold on
plot(konvergenca, 'LineWidth', 1.5);
plot(kon_avg, '--');
plot(kon_max, ':');
legend('Najboljši delec','Povprečje roja','Najslabši delec');
title('Potek konvergence PSO');
xlabel('Iteracija');
ylabel('Vrednost ciljne funkcije');
grid on;
hold off
```

V tem delu izrišemo potek konvergence algoritma PSO: *figure* prikaže najboljšo, povprečno in najslabšo vrednost ciljne funkcije v vsaki iteraciji. Z ukazi *legend*, *title*, *xlabel* in *ylabel* dodamo legendo in ustrezne oznake, kar omogoča lažje razumevanje učinkovitosti in obnašanja algoritma skozi čas.

Slika 16 na levi strani prikazuje položaje delcev po deseti iteraciji algoritma PSO. Vidno je, da se je večina delcev že zbrala v okolici minimuma, kar nakazuje uspešno usmerjanje gibanja delcev v iskalnem prostoru. Desna slika prikazuje potek konvergence vrednosti ciljne funkcije najboljšega delca v vsaki iteraciji skozi vse iteracije. Krivulja hitro pada v prvih nekaj korakih, kar pomeni hitro izboljševanje rešitve, nato pa se stabilizira okoli zelo nizke vrednosti. To potrjuje, da so se nekateri delci roja uspešno približali globalnemu minimumu.



Slika 16: Razpršenost delcev po prostoru iskanja v deseti iteraciji (levo) in potek konvergence algoritma (desno)

4.2.8 Vaja: Ciljna funkcija kot zunanja funkcija

V tej vaji boste ciljno funkcijo (funkcija Sphere) zapisali kot ločeno zunanjo funkcijo v novo datoteko. To omogoča boljšo preglednost programske kode in ponovno uporabo funkcije v različnih skriptih. Datoteko boste shranili v isto mapo kot glavni skript in vanjo vključili ustrezne ukaze.

- Ustvarite novo datoteko z imenom *sphere_function.m*.
- Shranite datoteko v isto mapo kot vaš glavni skript.
- V datoteko *sphere_function.m* vnesite naslednjo funkcijo.

```
function y = sphere_function(x)

y = sum(x.^2);

end
```

Funkcija *sphere_function* definira ciljno funkcijo, ki sprejme vhodni vektor x in vrne vsoto kvadratov njegovih komponent y.

- Na začetku algoritma pri spremenljivkah PSO dodajte spodnji vrstici.

```
%% Ciljna funkcija
fun = @sphere_function;
```

S to vrstico določimo ciljno funkcijo, ki jo bo algoritem PSO minimiziral, pri čemer z uporabo ročice *fun* omogočimo, da je optimizacijski postopek univerzalen in prilagodljiv različnim funkcijam brez spreminjanja samega algoritma. @ ne pokliče funkcije *sphere_function*, ampak ustvari referenco nanjo tj. ročico, ki jo lahko pozneje uporabimo.

- Spremenite del glavne kode, kjer kličete ciljno funkcijo v inicializaciji.

```
x = delec(i).pol;
% Dopolni: evalvacija ciljne funkcije (Sphere function)
delec(i).sposobnost = fun(x);
```

S tem shranimo trenutni položaj delca v spremenljivko x, ki jo nato uporabimo kot vhodni argument pri klicu ciljne funkcije. Tako se izognemo neposrednemu dostopu do strukture *delec(i).pol* znotraj funkcije in izboljšamo berljivost ter preglednost kode. Nato ovrednotimo ciljno funkcijo za trenutni položaj delca z uporabo zunanje funkcije *sphere_function*.

- Na enak način spremenite del glavne kode, kjer kličete ciljno funkcijo v iteracijah.

```
x = delec(i).pol;

% Dopolni: evalvacija ciljne funkcije (Sphere function)
delec(i).sposobnost = fun(x);
```

S tem smo ločili algoritem od definicije funkcije, kar omogoča enostavno zamenjavo ciljne funkcije brez poseganja v kodo algoritma.

Rezultat vaje sta dve datoteki: *algoritem_PSO.m* in *sphere_function.m*. V datoteki *algoritem_PSO.m* je zapisana celotna koda algoritma PSO, vključno z definicijo problema in zagonom optimizacije. Datoteka *sphere_function.m* pa vsebuje ločeno definirano ciljno funkcijo. Takšna razdelitev omogoča boljšo preglednost in enostavno zamenjavo ciljne funkcije, saj v glavnem skriptu le spremenimo klic funkcije brez poseganja v strukturo algoritma.

4.2.9 Vaja: Algoritem PSO kot zunanja funkcija

V tej vaji boste algoritem rojev delcev (PSO), ki ste ga do zdaj razvijali znotraj enega samega skripta, preoblikovali v zunanjo funkcijo. S tem boste dosegli modularnost kode, kar omogoča enostavno ponovno uporabo in testiranje algoritma z različnimi ciljnimi funkcijami, na enak način kot to omogočajo funkcije, ki so na voljo v MATLAB-ovih knjižnicah (npr. Global Optmization Toolbox). Cilja vaje sta preoblikovati kodo PSO v zunanjo funkcijo PSO_opt in doseči, da lahko funkcijo enostavno prikličemo z različnimi parametri in ciljnimi funkcijami.

- Ustvarite novo datoteko z imenom *PSO_opt.m*.
- Shranite datoteko v isto mapo kot vaš glavni skript.
- V datoteko *PSO_opt.m* vnesite naslednjo funkcijo.

```
function [x_opt, fval] = PSO_opt(fun, nvar, lb, ub, options)

end
```

Kjer sta *x_opt*, *fval* rezultata optimizacije, *fun* je ročica do ciljne funkcije, *nvar* je število spremenljivk, *lb, ub* sta vektorja spodnjih in zgornjih mej, *options* pa je struktura z nastavitvami algoritma (npr. število delcev, iteracij ipd.).

- Znotraj funkcije implementirajte celoten algoritem PSO, kot ste ga razvili v prejšnjih vajah.
- V algoritmu omogočite strukturo za vhodne spremenljivke (*options*), ki jih želite nadzorovati zunaj funkcije.

Rešitev:

```
% struktura options za vhodne spremenljivke
iterations = options.iterations;
vel_roj = options.vel_roj;
w = options.w;
c1 = options.c1;
c2 = options.c2;
```

V tem odseku iz strukture *options* preberemo parametre, ki določajo obnašanje algoritma PSO (število iteracij, velikost roja, parametre gibanja). Tako omogočimo, da so nastavitve algoritma ločene od implementacije in jih lahko enostavno spreminjamo iz zunanjega skripta.

➢ Izbrišite naslednje vrstice.

```
%% Ciljna funkcija
fun = @sphere_function;

nvar = 2;               % Število spremenljivk (dimenzija problema)

%% Omejitve prostora spremenljivk
lb = [-100 -100];       % Spodnje meje
ub = [100 100];         % Zgornje meje
```

To so spremenljivke, ki jih zdaj pošiljamo iz zunanje skripte.

➢ Na koncu funkcije pred zadnji *end* dodajte spodnje vrstice.

```
%% Rezultat
x_opt = naj_delec_pol;
fval = naj_delec_sposobnost;
```

V tem delu funkcije določimo izhodna rezultata: optimalni položaj delca *x_opt* in pripadajočo vrednost ciljne funkcije *fval*. Ta rezultat predstavlja najboljšo rešitev, ki jo je algoritem našel po vseh iteracijah.

4.2.10 Vaja: Uporabite funkcijo PSO_opt

Cilj naloge je uporabiti algoritem rojev delcev za iskanje globalnega minimuma kvadratne funkcije z dvema spremenljivkama in vizualizirati gibanje delcev ter konvergenco ciljne funkcije.

- Ustvarite novo datoteko z imenom *zagon_PSO_opt.m*.
- Zapišite kodo, ki pokliče funkcijo *PSO_opt* z naslednjimi parametri:
  - število spremenljivk: 2
  - spodnja meja vrednosti spremenljivk: –100
  - zgornja meja vrednosti spremenljivk: +100
  - število iteracij: 100
  - število delcev v roju: 50
  - vztrajnostni koeficient (ω): 0,9
  - kognitivni koeficient (c_1): 2
  - socialni koeficient (c_2): 2

Rešitev:

```
% Definicija ciljne funkcije
fun = @sphere_function;

% Parametri problema
nvar = 2;
lb = [-100 -100];
ub = [100 100];

% Nastavitve PSO
options.iterations = 100;
options.vel_roj = 50;
options.w = 0.9;
options.c1 = 2;
options.c2 = 2;

% Zagon optimizacije
[x_opt, fval] = PSO_opt(fun, nvar, lb, ub, options);

% Izpis
fprintf('Optimalna rešitev: [%f, %f]\n', x_opt(1), x_opt(2));
fprintf('Vrednost ciljne funkcije: %f\n', fval);
```

To je primer kode *zagon_PSO_opt.m*, ki definira in zažene optimizacijo z algoritmom PSO. Najprej se določi ciljna funkcija *sphere_function* prek funkcijske ročice *fun*. Nato se določijo parametri problema (število spremenljivk in meje iskanja) ter nastavitve parametrov PSO (število iteracij, velikost roja in parametri gibanja). Funkcija *PSO_opt* izvede optimizacijo in vrne optimalno rešitev *x_opt* ter pripadajočo vrednost ciljne funkcije *fval*. Na koncu se rezultat izpiše v ukazno okno.

Prečiščena koda PSO_opt

```
function [x_opt, fval] = PSO_opt(fun, nvar, lb, ub, options)

%% Parametri algoritma
iterations = options.iterations;
vel_roj = options.vel_roj;

%% Konstante PSO
w = options.w;
c1 = options.c1;
c2 = options.c2;

%% Prelokacija matrik za hitrejše procesiranje
konvergenca = zeros(iterations,1);
kon_avg = zeros(iterations,1);
kon_max = zeros(iterations,1);
Xpol = zeros(vel_roj,1);
Ypol = zeros(vel_roj,1);

%% Struktura posameznega delca
delec.pol = [];
delec.hitrost = [];
delec.sposobnost = [];
delec.naj_pol = [];
delec.naj_sposobnost = [];

%% Globalni optimum
naj_delec_sposobnost = inf;
naj_delec_pol = zeros(nvar,1);

%% Inicializacija vseh delcev
delec = repmat(delec, vel_roj, 1);

for i = 1 : vel_roj
```

```
    for j = 1 : nvar
        % Dopolni: naključni začetni položaj znotraj meja
        delec(i).pol(j) = lb(j) + rand * (ub(j) - lb(j));

        % Dopolni: začetna hitrost = 0
        delec(i).hitrost(j) = 0;
    end

    x = delec(i).pol;

    % Dopolni: evalvacija ciljne funkcije (Sphere function)
    delec(i).sposobnost = fun(x);


    % Dopolni: nastavitev osebnega optimuma
    delec(i).naj_pol = delec(i).pol;
    delec(i).naj_sposobnost = delec(i).sposobnost;

    % Dopolni: posodobitev globalnega optimuma
    if delec(i).sposobnost < naj_delec_sposobnost
        naj_delec_sposobnost = delec(i).sposobnost;
        naj_delec_pol = delec(i).pol;
    end
end

%% Zanka po vseh spremembah položaja roja

for iter = 1 : iterations
    % Zanka po vseh delcih, izračun položaja in ciljne funkcije vsakega
delca
    for i = 1 : vel_roj

        for j = 1 : nvar

            delec(i).hitrost(j) = ( ...
                w*rand*delec(i).hitrost(j) ...
                + c1*rand*(delec(i).naj_pol(j) - delec(i).pol(j)) ...
                + c2*rand*(naj_delec_pol(j) - delec(i).pol(j)));

            delec(i).pol(j) = delec(i).pol(j) + delec(i).hitrost(j);

        end
```

```
        x = delec(i).pol;

        % Dopolni: evalvacija ciljne funkcije (Sphere function)
        delec(i).sposobnost = fun(x);

        % Posodobitev najboljših lastnosti delca
        if delec(i).sposobnost < delec(i).naj_sposobnost

            delec(i).naj_sposobnost = delec(i).sposobnost;

            delec(i).naj_pol = delec(i).pol;

        end

        % Posodobitev lastnosti najboljšega delca
        if delec(i).sposobnost < naj_delec_sposobnost

            naj_delec_pol = delec(i).pol;

            naj_delec_sposobnost = delec(i).sposobnost;

        end
        % Izris in konvergenca
        Xpol(i) = delec(i).pol(1);
        Ypol(i) = delec(i).pol(2);

    end

    %% Vizualizacija gibanja delcev
    clf;
    plot(Xpol, Ypol, '*');
    axis([lb(1) ub(1) lb(2) ub(2)]);
    title(['Položaji delcev – iteracija ', num2str(iter)]);
    xlabel('x(1)');
    ylabel('x(2)');
    pause(0.01);

    % Shranjevanje sposobnosti delcev
    konvergenca(iter) = naj_delec_sposobnost;
    kon_avg(iter) = mean([delec.sposobnost]);
    kon_max(iter) = max([delec.sposobnost]);

end
```

```
%% Izris konvergence
figure('Name','Konvergenca PSO');
hold on
plot(konvergenca, 'LineWidth', 1.5);
plot(kon_avg, '--');
plot(kon_max, ':');
legend('Najboljši delec','Povprečje roja','Najslabši delec');
title('Potek konvergence PSO');
xlabel('Iteracija');
ylabel('Vrednost ciljne funkcije');
grid on;
hold off

%% Rezultat
x_opt = naj_delec_pol;
fval = naj_delec_sposobnost;
end
```

Zapisana funkcija *PSO_opt* izvaja optimizacijo z algoritmom rojev delcev (PSO). Na začetku nastavi parametre, inicializira delce z naključnimi položaji in vrednoti njihovo sposobnost. Nato sledi glavna zanka, kjer se v vsaki iteraciji posodabljajo hitrosti, položaji, osebni in globalni optimum. Med izvajanjem se vizualizira gibanje delcev in beleži potek konvergence. Na koncu funkcija vrne najboljšo najdeno rešitev in pripadajočo vrednost ciljne funkcije.

4.3 Samostojno delo s področja algoritma PSO

Poglavje obsega samostojne naloge, namenjene poglobljenemu razumevanju delovanja algoritma PSO v različnih kontekstih. Skozi naloge raziščete vpliv parametrov na konvergenco in razpršenost, pomen začetne porazdelitve delcev ter uvedbo naprednih ustavitvenih meril. S testnimi funkcijami, kot so Rastrigin, Himmelblau in Rosenbrock, analizirate vedenje algoritma ob prisotnosti več modalnosti in omejitev. Naloge vključujejo tudi omejevanje gibanja delcev znotraj iskalnega prostora ter razširitev algoritma za obravnavo enakovrednostnih in neenakovrednostnih omejitev. Zaključna primerjalna analiza med PSO in GA omogoča oceno učinkovitosti in stabilnosti obeh pristopov. Poglavje spodbuja eksperimentiranje, analitično primerjavo rezultatov in razvijanje občutka za prilagajanje algoritma glede na problem.

4.3.1 Samostojno delo: Vpliv parametrov na konvergenco

V tej nalogi boste raziskali, kako različne nastavitve parametrov algoritma PSO vplivajo na njegovo vedenje, zlasti na hitrost konvergence, sposobnost raziskovanja prostora rešitev in razpršenost delcev. Cilji naloge so spoznati vpliv nastavitev parametrov PSO na vedenje algoritma, oceniti kompromis med raziskovanjem in konvergenco ter razumeti pomen uravnoteženega sodelovanja med delci v roju. Naloga omogoča poglobljeno razumevanje notranjih mehanizmov algoritma in podpira razvoj intuicije za njegovo učinkovito uporabo.

Zahteve naloge:

- Uporabite algoritem PSO za optimizacijo funkcije Rastrigin (poglavje 5.2).
- Izvedite več zaporednih optimizacij, pri čemer uporabite naslednje kombinacije parametrov:
  - $\omega = 0{,}9;\ c_1 = 1{,}5;\ c_2 = 1{,}5;$
  - $\omega = 0{,}4;\ c_1 = 2{,}0;\ c_2 = 2{,}0;$
  - $\omega = 0{,}7;\ c_1 = 2{,}5;\ c_2 = 0{,}5;$
  - $\omega = 0{,}6;\ c_1 = 0{,}5;\ c_2 = 2{,}5.$
- Določite parametre PSO po metodi Clerc-Kennedy za vrednosti spremenljivk:
  - $K = 1;$
  - $\Phi_1 = 2{,}05;$
  - $\Phi_2 = 2{,}05.$
- Primerjajte značilnosti obnašanja algoritma pri različnih nastavitvah in določite:
  - konvergenco algoritma,
  - obseg raziskovanja prostora rešitev,
  - usklajenost gibanja delcev.

Namig: Obseg raziskovanja prostora rešitev lahko ocenite numerično, tako da prikažite standardni odklon položajev vseh delcev v vsaki iteraciji, kar pokaže, kako razpršeni so delci po prostoru. Pri določitvi usklajenosti gibanja delcev pokažite, ali se delci gibljejo usklajeno proti istemu optimumu ali vsak zase, kar je znak slabe koordinacije. Uporabite npr. povprečno razdaljo do globalnega optimuma v vsaki iteraciji.

4.3.2 Samostojno delo: Inicializacija z znano začetno točko

V tem poglavju raziskujete, kako začetna pozicija posameznega delca vpliva na končni rezultat optimizacije. Cilj je razumeti, kako pomembna je začetna porazdelitev v prostoru rešitev, še posebej pri večmodalnih funkcijah, kot je funkcija Himmelblau

Zahteve naloge:

- Implementirajte optimizacijo funkcije Himmelblau (poglavje 5.3) z uporabo PSO.
- V začetni roj vključite delec z natančno določeno začetno pozicijo.

4.3.3 Samostojno delo: Parametrična ustavitvena merila

Pri tej nalogi se osredotočate na napredna ustavitvena merila, ki temeljijo na stabilnosti vrednosti ciljne funkcije. Cilj je razviti bolj robustne pogoje za zaustavitev, ki preprečijo nepotrebno dolge ali prehitre prekinitve optimizacije. Analizirajte trajanje optimizacije in njeno občutljivost na različna merila konvergence.

Zahteve naloge:

- Implementirajte optimizacijo funkcije Rastrigin (poglavje 5.2) z uporabo PSO.
- Razvijte merilo za ustavitev algoritma, ki temelji na spremembi sposobnosti najboljšega delca. Uporabite naslednja parametra:
  - toleranca spremembe sposobnosti $\varepsilon = 1e\text{-}6$,
  - število zaporednih iteracij *(npr. 5)*, v katerih je sprememba manjša od ε.
- Vključite uporabo funkcij *tic* in *toc* za merjenje časa izvajanja algoritma.

4.3.4 Samostojno delo: Stroga omejitev gibanja

V tem delu raziskujete, kakšen vpliv ima strogo omejevanje gibanja delcev na robove domene iskanja na učinkovitost algoritma. Cilj je razumeti, ali prisilno ohranjanje delcev znotraj dovoljene domene izboljša zanesljivost algoritma ali pa ga omejuje pri iskanju boljših rešitev.

Zahteve naloge:

- Implementirajte optimizacijo funkcije Rastrigin (poglavje 5.2) z uporabo PSO.
- Prilagodite kodo PSO tako, da delci med iteracijami ne morejo zapustiti domene iskanja (npr. s prisilnim vračanjem na mejo).
- Primerjajte delovanje s prejšnjo verzijo, kjer se je z domeno iskanja omejeval le začetni roj.

4.3.5 Samostojno delo: Enakovrednostne in neenakovrednostne omejitve

Pri tej nalogi boste za PSO omogočili optimizacijo z enakovrednostnimi in neenakovrednostnimi omejitvami. Cilji so preveriti, kako PSO ravna z omejitvami, kdaj pride do težav (npr. nihanja ob robovih) in ali lahko algoritem najde optimum, kljub zapletenemu prostoru rešitev.

Zahteve naloge:

- Implementirajte optimizacijo funkcije Rosenbrock z omejitvami (poglavje 5.5).
- Preverite, ali algoritem zanesljivo najde optimum.

Opomba:

Upoštevajte, da funkcija *particleswarm*, vgrajena v MATLAB, ne omogoča neposredne uporabe enakovrednostnih in neenakovrednostnih omejitev.

4.3.6 Samostojno delo: Primerjava z genetskim algoritmom

Primerjate robustnost in učinkovitost PSO ter GA. Cilji so analizirati, kateri algoritem je zanesljivejši glede na kakovost rešitve, čas izvajanja in stabilnost rezultatov pri večkratnem zagonu na istem problemu z omejitvami. Analizirajte, kateri algoritem je bolj zanesljiv in stabilen pri reševanju problemov z omejitvami.

Zahteve naloge:

- Rešite problem iz poglavja 4.3.5 tudi z genetskim algoritmom (GA) in uporabite enake ciljne funkcije, omejitve in začetne pogoje.

- Primerjajte naslednje kazalce:
  - najdena vrednost ciljne funkcije,
  - čas izvajanja,
  - stabilnost rezultatov (izvedite vsaj 5 ponovitev).

Univerzitetna založba
Univerze v Mariboru

5 Optimizacijske funkcije

Poglavje predstavlja izbor klasičnih testnih funkcij, ki se pogosto uporabljajo za preverjanje učinkovitosti optimizacijskih algoritmov. Med njimi so enostavna kvadratna funkcija (Sphere), večmodalni funkciji Rastrigin in Himmelblau, kompleksna Gomez and Levy z neenačbeno omejitvijo ter funkcija Rosenbrock z značilno ukrivljeno dolino in dodatnimi omejitvami. Vsaka funkcija je opisana z izrazom, globalnimi minimumi, domeno iskanja in ključnimi značilnostmi, pomembnimi za analizo vedenja optimizacijskih metod.

5.1 Funkcija Sphere

Kvadratna funkcija ali funkcija Sphere je ena najosnovnejših in najpogosteje uporabljenih testnih funkcij v optimizaciji. Predstavlja dobro izhodišče za testiranje osnovne pravilnosti, hitrosti konvergence in vpliva parametrov optimizacijskega algoritma.

Ciljna funkcija:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$$

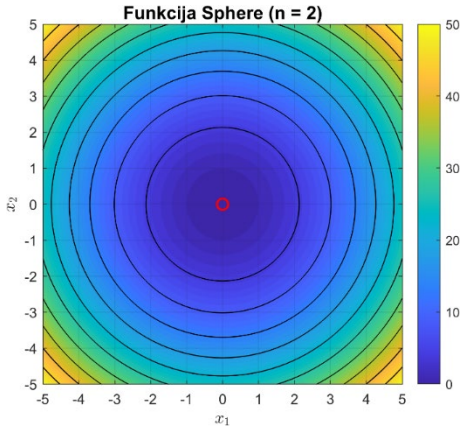
Globalni minimum:

$$f(x_1, \dots, x_n) = 0 \text{ pri } \boldsymbol{x}_n = (0, 0, \dots, 0)$$

Domena iskanja:

$$x_i \in [-\infty;\ \infty] \text{ za } i = 1, 2, \dots, n$$

Izris:



5.2 Funkcija Rastrigin

Funkcija Rastrigin je večmodalna, z mnogimi lokalnimi minimumi. Za $n = 2$ ima zelo valovito obliko z minimumom v izhodišču. Uporablja se kot testna funkcija pri globalni optimizaciji, npr. z algoritmi rojev delcev, genetskimi algoritmi ipd.

Ciljna funkcija:

$$f(\boldsymbol{x}) = An + \sum_{i=1}^{n} [x_i^2 - A\cos(2\pi x_i)],$$

kjer je običajno $A = 10$.

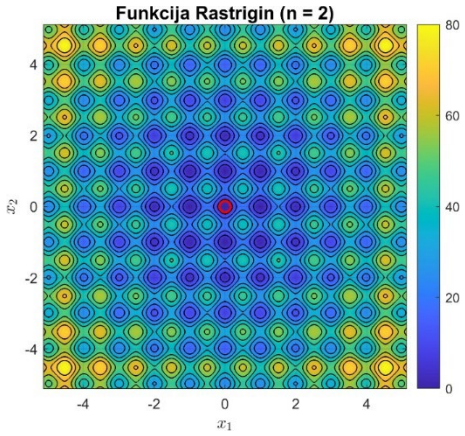
Globalni minimum:

$$f(x_1, \dots, x_n) = 0 \text{ pri } \boldsymbol{x}_n = (0, 0, \dots, 0)$$

Domena iskanja (za vsako spremenljivko):

$$x_i \in [-5{,}12;\ 5{,}12] \text{ za } i = 1, 2, \dots, n$$

Izris:



5.3 Funkcija Himmelblau

Funkcija Himmelblau je večmodalna in ima več globalnih simetrično razporejenih minimumov. Uporablja se kot testna funkcija pri preverjanju učinkovitosti optimizacijskih algoritmov.

Ciljna funkcija:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Globalni minimumi (v domeni iskanja):

$$f(3{,}0; 2{,}0) = 0$$

$$f(-2{,}805; 3{,}131) \approx 0$$

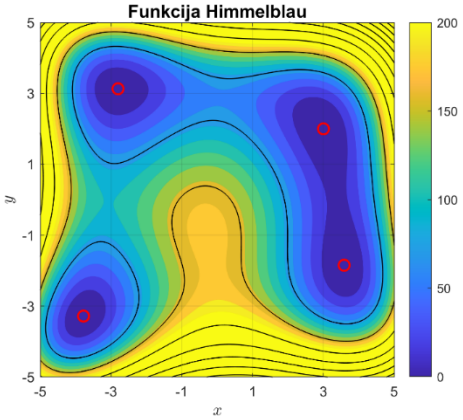
$$f(-3{,}779; -3{,}283) \approx 0$$

$$f(3{,}584; -1{,}848) \approx 0$$

Domena iskanja:

$$x, y \in [-5;\ 5]$$

Izris:



5.4 Funkcija Gomez and Levy (modified)

Funkcija Gomez and Levy je znana zaradi svoje kompleksne in valovite oblike z več lokalnimi minimumi. Uporablja se kot testni primer v optimizaciji z nelinearnimi omejitvami.

Ciljna funkcija:

$$f(x, y) = 4x^2 - 2{,}1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$

Neenačbena omejitev:

$$-\sin(4\pi x) + 2\sin^2(2\pi y) \leq 1{,}5$$

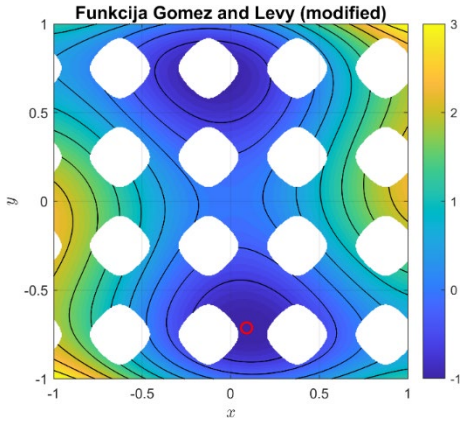
Globalni minimum:

$$f(0{,}08984201;\ -0{,}7126564) = -1{,}031628453$$

Domena iskanja:

$$x \in [-1; 0{,}75] \text{ in } y \in [-1;\ 1]$$

Izris:



5.5 Funkcija Rosenbrock z omejitvami

Funkcija Rosenbrock je znana po svoji značilni ozki in ukrivljeni dolini, ki zaradi omejitev ne vodi do globalnega minimuma. Funkcija predstavlja izziv za številne optimizacijske algoritme, saj je zelo občutljiva na začetne pogoje in zahteva natančno usmerjeno iskanje. Uporablja se kot testna funkcija pri testiranju robustnosti in učinkovitosti optimizacijskih metod.

Ciljna funkcija:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Neenačbene omejitve:

$$(x - 1)^3 - y + 1 \leq 0 \text{ in } x + y - 2 \leq 0$$

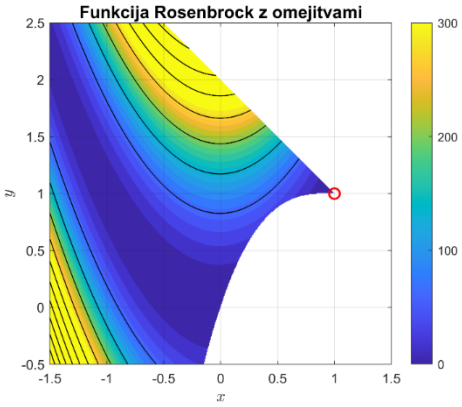
Globalni minimum:

$$f(1{,}0;\ 1{,}0) = 0$$

Domena iskanja:

$$x \in [-1{,}5;\ 1{,}5] \text{ in } y \in [-0{,}5;\ 2{,}5]$$

Izris:



Univerzitetna založba
Univerze v Mariboru

6 Primeri Pareto front

V nadaljevanju so predstavljeni primeri izgradnje nekaterih tipičnih Pareto front pod pogoji, kot jih predlagajo različni avtorji. Za vsak primer so zapisane funkcije in omejitve, ki so jih avtorji pri generaciji front uporabljali.

6.1 Pareto fronta Binh and Korn

Problem Binh and Korn je znan testni primer v večkriterijski optimizaciji, ki vključuje dva cilja in dve omejitvi. Pareto fronta te funkcije je nekonveksna, delno omejena z neenačbnimi pogoji, kar omogoča preizkušanje zmožnosti algoritmov za iskanje raznolikih in optimalnih rešitev v prisotnosti omejitev. Primerna je za analizo porazdelitve rešitev na Pareto fronti ter oceno robustnosti in konvergence večkriterijskih optimizacijskih algoritmov.

Ciljni funkciji:

$$f_1(x,y) = 4x^2 + 4y^2$$

$$f_2(x,y) = (x-5)^2 + (y-5)^2$$

Omejitve:

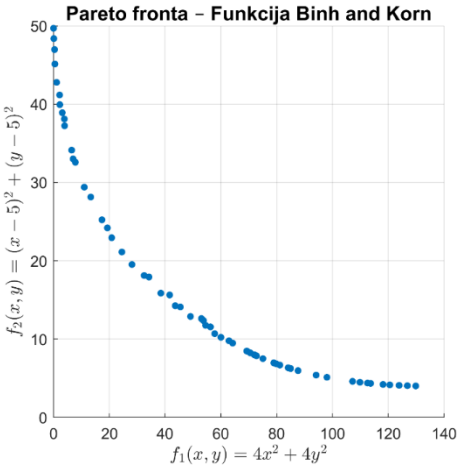
$$g_1(x, y) = (x - 5)^2 + y^2 \leq 25$$

$$g_2(x, y) = (x - 8)^2 + (y + 3)^2 \geq 7{,}7$$

Domena iskanja:

$$x \in [0;\ 5] \text{ in } y \in [0;\ 3]$$

Izris:



6.2 Pareto fronta Chakong and Haimes

Problem Chakong and Haimes je klasičen testni primer za večkriterijsko optimizacijo z dvema ciljnima funkcijama in dvema omejitvama. Pareto fronta pri tem problemu je zvezna, gladka in nekonveksna, kar omogoča preizkušanje sposobnosti algoritmov pri iskanju natančne in enakomerno porazdeljene fronte. Uporablja se za analizo uravnoteženosti med konflikti ciljev in preverjanje, kako dobro algoritem pokriva celotno Pareto fronto.

Ciljni funkciji:

$$f_1(x,y) = 2 + (x-2)^2 + (y-1)^2$$

$$f_2(x,y) = 9x - (y-1)^2$$

Omejitve:

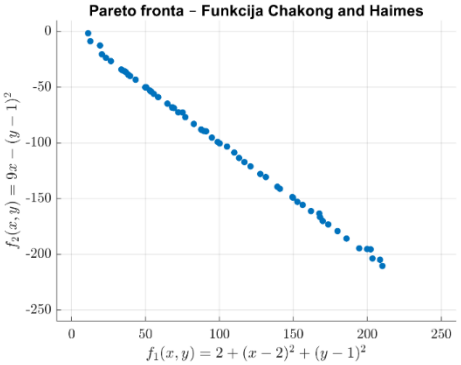
$$g_1(x,y) = x^2 + y^2 \leq 225$$

$$g_2(x,y) = x - 3y + 10 \leq 0$$

Domena iskanja:

$$x \in [-20;\ 20] \text{ in } y \in [-20;\ 20]$$

Izris:



6.3 Pareto fronta Kursawe

Problem Kursawe je testni primer za večkriterijsko optimizacijo z dvema ciljema in tremi spremenljivkami v omejenem območju. Zaradi nelinearnih in oscilatornih členov generira dvakrat ločeno nekonveksno Pareto fronto, kar predstavlja izziv za algoritme pri iskanju raznolikih in porazdeljenih rešitev. Uporablja se za testiranje algoritmov na kompleksnih, razdrobljenih frontah.

Ciljna funkcija:

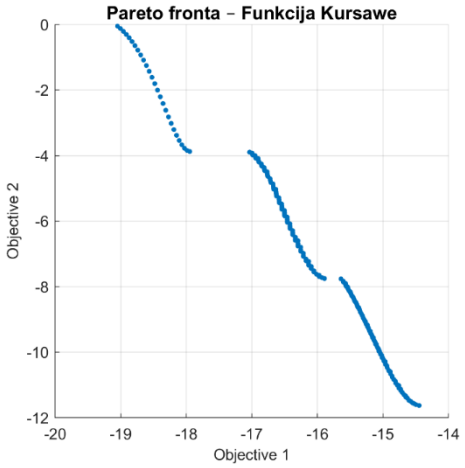
$$\text{Minimize } f_1(\boldsymbol{x}) = \sum_{i=1}^{n-1}\left(-10\exp\left(-0{,}2\sqrt{x_i^2 + x_{i+1}^2}\right)\right)$$

$$\text{Minimize } f_2(\boldsymbol{x}) = \sum_{i=1}^{n-1}(|x_i|^{0{,}8} + 5\sin(x_i)^3)$$

Domena iskanja:

$$-5 \leq x_1, x_2, x_3 \leq 5$$

Izris:



6.4 Pareto fronta Deb Bimodal

Funkcija Deb Bimodal je testni primer za večkriterijsko optimizacijo, kjer avtor z bimodalno funkcijo $g(x_2)$ ustvari zanimivo razmerje med ciljnima funkcijama. Prva funkcija je linearna, druga pa vključuje deljenje z x_1 in kompleksno nelinearno funkcijo $g(x_2)$.

Rezultat optimizacije je konveksna Pareto fronta, kar omogoča testiranje algoritmov pri obravnavi gladkih, a nelinearno povezanih ciljev. Posebej je primerna za analizo zmožnosti algoritma, da pravilno sledi konveksni strukturi fronte.

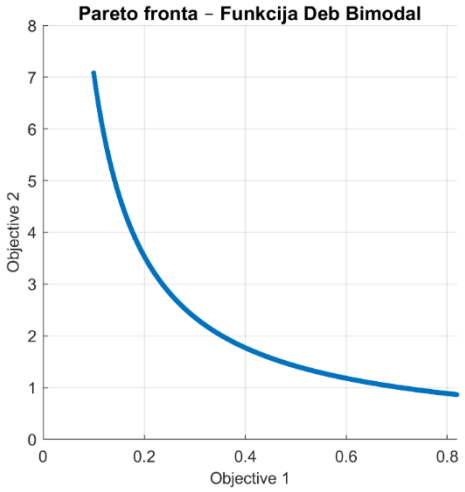
Ciljna funkcija:

$$\text{Minimize } f_1(x_1, x_2) = x_1$$

$$\text{Minimize } f_2(x_1, x_2) = \frac{g(x_2)}{x_1}$$

$$g(x_2) = 2{,}0 - \exp\left\{-\left(\frac{x_2 - 0{,}2}{0{,}004}\right)^2\right\} - 0{,}8\exp\left\{-\left(\frac{x_2 - 0{,}6}{0{,}4}\right)^2\right\}$$

Izris:



6.5 Pareto fronta Kita

Problem Kita je znan večkriterijski testni primer, ki temelji na analogiji s termodinamičnimi sistemi. Zasnovan je tako, da vključuje dva cilja, ki sta med seboj v konfliktu in nelinearno povezana, medtem ko omejitve posnemajo značilnosti termodinamičnih enačb.

Ciljni funkciji vključujeta kvadratni in linearni člen, omejitve pa definirajo nelinearno omejen prostor rešitev. Zaradi kompleksnih robnih pogojev je Pareto fronta delno odrezana in ukrivljena, kar omogoča testiranje algoritmov pri obravnavi več omejitev in različnih vrst povezav med cilji.

Ciljna funkcija:

$$\text{Maximize } f_1(x, y) = -x^2 + y$$

$$\text{Maximize } f_2(x, y) = \frac{1}{2}x + y + 1$$

Omejitve:

$$\frac{1}{6}x + y - \frac{13}{2} \leq 0$$

$$\frac{1}{2}x + y - \frac{15}{2} \leq 0$$

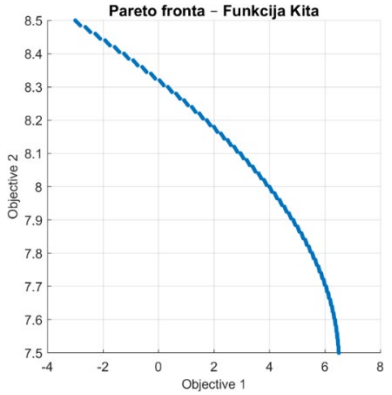
$$\frac{5}{x} + y - 30 \leq 0$$

Domena iskanja:

$$0 \leq x \leq 7$$

$$0 \leq y \leq 7$$

Izris:



6.6 Pareto fronta DTLZ1

Funkcija DTLZ1 je del znane družine testnih problemov DTLZ, namenjenih preverjanju učinkovitosti večkriterijskih optimizacijskih algoritmov, zlasti genetskih algoritmov.

Zasnovana je za poljubno število ciljev, pri čemer so ciljne funkcije simetrične in vključujejo skupno funkcijo $g(x_M)$. Vhodne spremenljivke so definirane v omejenem območju, rezultat optimizacije pa je večdimenzionalna linearna Pareto ploskev, kar omogoča jasno analizo porazdelitve in pokritosti rešitev.

DTLZ1 se pogosto uporablja za testiranje konvergence in raznolikosti algoritmov v večdimenzionalnem prostoru ciljev.

Ciljna funkcija:

$$\text{Minimize } f_1(\boldsymbol{x}) = \frac{1}{2} x_1 x_2 \dots x_{M-1}\big(1 + g(x_M)\big)$$

$$\text{Minimize } f_2(\boldsymbol{x}) = \frac{1}{2} x_1 x_2 \dots (1 - x_{M-1})\big(1 + g(x_M)\big)$$

$$\text{Minimize } f_{M-1}(\boldsymbol{x}) = \frac{1}{2} x_1 (1 - x_2)\big(1 + g(x_M)\big)$$

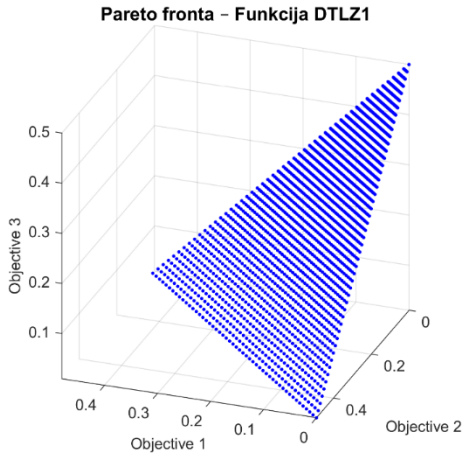
$$\text{Minimize } f_M(\boldsymbol{x}) = \frac{1}{2} (1 - x_1)\big(1 + g(x_M)\big)$$

$$g(x_M) = 100\left[|x_M| + \sum_{x_i \in x_M} (x_i - 0{,}5)^2 - \cos\big(20\pi(x_i - 0{,}5)\big)\right]$$

Domena iskanja:

$$0 \le x_i \le 1 \text{ za } i = 1,2, \dots, n$$

Izris (za $i = 3$):



6.7 Pareto fronta DTLZ7

Funkcija DTLZ7 je še en kompleksnejši testni primer iz družine DTLZ, ki ga je predlagal avtor Deb za oceno zmogljivosti večkriterijskih optimizacijskih algoritmov.

V tem primeru prve $M-1$ ciljne funkcije prevzamejo vrednosti posameznih spremenljivk, medtem ko je zadnja funkcija odvisna od vseh prejšnjih in funkcije *h*, ki vključuje sinusne oscilacije.

Rezultat optimizacije za tridimenzionalni problem so štiri nepovezane Pareto ploskve, kar predstavlja velik izziv za algoritme, zlasti pri ohranjanju raznolikosti in pokritosti nepovezanih regij. DTLZ7 je zato primeren za preverjanje raziskovanja prostora rešitev in občutljivosti algoritma na oscilacije v ciljnih funkcijah.

Ciljna funkcija:

$$\text{Minimize } f_1(x_1) = x_1$$

$$\text{Minimize } f_2(x_2) = x_2$$

$$\text{Minimize } f_{M-1}(x_{M-1}) = x_{M-1}$$

$$\text{Minimize } f_M(x) = \big(1 + g(x_M)\big)h(f_1, f_2, \dots f_{M-1}, g)$$

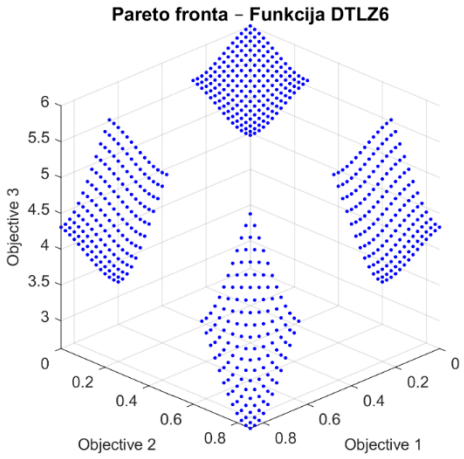
$$g(x_M) = 1 + \frac{9}{|x_M|}\sum_{x_i \in x_M} x_i$$

$$h(f_1, f_2, \dots f_{M-1}, g) = M - \sum_{i=1}^{M-1}\left[\frac{f_i}{1+g}\big(1 + \sin(3\pi f_i)\big)\right]$$

Domena iskanja:

$$0 \le x_i \le 1 \text{ za } i = 1,2, \dots, n$$

Izris (za $i = 3$):



Univerzitetna založba
Univerze v Mariboru

7 Sklep

Optimizacija v inženirstvu ni zgolj matematično orodje, temveč pomembna miselna naravnanost, ki spodbuja sistematično iskanje izboljšav v realnih procesih, napravah in sistemih. V teh skriptih smo obravnavali temeljne pristope k enokriterijskim in večkriterijskim optimizacijskim problemom ter spoznali, kako lahko z uporabo metahevrističnih populacijskih algoritmov, kot sta genetski algoritem in algoritem rojev delcev, poiščemo rešitve tudi v zahtevnih, nelinearnih in večdimenzionalnih okoljih.

Pri delu smo se poleg učenja ukazov in postopkov osredotočili predvsem na razvijanje razumevanja, kaj pomeni kakovostna rešitev v okviru danega problema. Pomemben poudarek je bil na tem, kako takšno rešitev prepoznati, kako ustrezno oblikovati optimizacijski problem glede na cilje in omejitve ter kako rezultate ustrezno interpretirati v širšem inženirskem kontekstu. S tem smo gradili ne le tehnične spretnosti, temveč tudi sposobnost analitičnega razmišljanja in presojanja, ki je ključna pri uporabi optimizacijskih pristopov v praksi.

Cilj je bil razviti občutek za smiselno uporabo optimizacijskih metod ter razumevanje njihovih zmogljivosti in omejitev, pri čemer obvladovanje vseh tehničnih podrobnosti ni nujno. Z vključevanjem raznolikih nalog, od enostavnih funkcij z enim ciljem do kompleksnih večkriterijskih kompromisov, smo želeli prikazati širok

spekter možnosti uporabe optimizacije. Hkrati smo spodbujali, da se vsak udeleženec poglobi v tiste vidike, ki so zanj najbolj pomembni, uporabni ali raziskovalno zanimivi.

Iz preučitve vsebin in reševanja vaj izhajajo naslednja znanja in spretnosti:

- formulirati enokriterijske in večkriterijske optimizacijske probleme z ustreznimi omejitvami in ciljnimi funkcijami,
- izbrati in utemeljiti primeren optimizacijski algoritem (npr. GA, PSO) glede na naravo problema,
- nastaviti meje in omejitve ter razumeti njihov vpliv na rešitev,
- izvesti in interpretirati optimizacijo v okolju MATLAB,
- vizualizirati rezultate in analizirati Pareto fronte pri večkriterijski optimizaciji,
- oceniti konvergenco algoritma in analizirati občutljivost rezultatov na parametre algoritmov.

Vabim te, da pridobljeno znanje s področja optimizacije uporabiš tudi pri drugih predmetih, projektnih nalogah ali raziskovalnem delu. Optimizacijski pristopi se pojavljajo v najrazličnejših kontekstih, od tehničnih in proizvodnih sistemov do ekonomskih, okoljskih ali organizacijskih odločitev. Povsod tam, kjer si prizadevamo za boljše, učinkovitejše ali bolj uravnotežene rešitve, ima optimizacija pomembno vlogo. Ključno pa je, da znamo prepoznati takšne situacije in znanje, ki smo ga pridobili, ustrezno uporabiti v praksi.

Univerzitetna založba
Univerze v Mariboru

Literatura

Binh, T. T., & Korn, U. (1997). *MOBES: A multiobjective evolution strategy for constrained optimization problems*. Paper presented at the The third international conference on genetic algorithms (Mendel 97).

Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation, 6*(1), 58-73.

Coello, C. A. C., Lamont, G. B., & Veldhuizen, D. A. V. (2007). *Evolutionary algorithms for solving multi-objective problems*: Springer.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*: University of Michigan.

Deb, K., Thiele, L., Laumanns, M., & Zitzler, E. (2005). Scalable Test Problems for Evolutionary Multiobjective Optimization. A. Abraham, L. Jain, & R. Goldberg (Eds.), *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications* (pp. 105-145). London: Springer London.

Eberhart, R., & Kennedy, J. (1995). *A new optimizer using particle swarm theory*. Paper presented at the MHS'95. Proceedings of the sixth international symposium on micro machine and human science.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*: Addison-Wesley.

Himmelblau, D. M. (1972). *Applied Nonlinear Programming*: McGraw-Hill.

Holland, J. H. (1975). Adaptation in natural and artificial systems. *University of Michigan Press google schola, 2*, 29-41.4

Kennedy, J., & Eberhart, R. (1995). *Particle swarm optimization*. Paper presented at the Proceedings of ICNN'95-international conference on neural networks.

Kennedy, J., & Eberhart, R. (1997). *A discrete binary version of the particle swarm algorithm*. Paper presented at the 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation.

Kursawe, F. (1990). *A variant of evolution strategies for vector optimization*. Paper presented at the International conference on parallel problem solving from nature.

MathWorks Documentation. (2025). Global Optimization Toolbox User's Guide. https://www.mathworks.com/help/gads/

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm I. Continuous parameter optimization. *Evolutionary Computation, 1*(1), 25-49.

Rastrigin, L. A. (1974). Systems of extremal control. *Nauka*.

Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function. *The computer journal, 3*(3), 175-184.

Simionescu, P. A. (2020). A collection of bivariate nonlinear optimisation test problems with graphical representations. *International Journal of Mathematical Modelling and Numerical Optimisation, 10*(4), 365-398.

Tamaki, H., Kita, H., & Kobayashi, S. (1996). *Multi-objective optimization by genetic algorithms: A review*. Paper presented at the Proceedings of IEEE international conference on evolutionary computation.

Vira, C., & Haimes, Y. Y. (1983). Multiobjective decision making: theory and methodology. *Noth-Holland Series in System Science and Engineering*, 62-109.
Wikipedia contributors. (2025). Test functions for optimization. Wikipedia. https://en.wikipedia.org/wiki/Test_functions_for_optimization
Yang, X.-S. (2010). Engineering Optimization: An Introduction with Metaheuristic Applications. Wiley.
Yarpiz. (2025). Particle Swarm Optimizationvin MATLAB [Video]. YouTube. https://www.youtube.com/watch?v=sB1n9a9yxJk

Optimizacije v inženirstvu: Reševanje problemov z metahevrističnimi metodami v okolju MATLAB

Janez Gotlih, Mirko Ficko
Univerza v Mariboru, Fakulteta za strojništvo, Maribor, Slovenija
janez.gotlih@um.si, mirko.ficko@um.si

Skripta obravnavajo temeljne pristope optimizacije v inženirstvu s poudarkom na uporabi metahevrističnih metod, kot sta genetski algoritem (GA) in algoritem rojev delcev (PSO). Namenjena so študentom in inženirjem, ki želijo razumeti tako teoretično ozadje kot praktično implementacijo optimizacijskih algoritmov v okolju MATLAB. Vključujejo poglavja o enokriterijskih in večkriterijskih optimizacijskih problemih, obravnavajo omejitve, različne ciljne funkcije ter vizualizacijo rezultatov. Vsako poglavje vsebuje strukturirane vaje in naloge za samostojno delo, ki spodbujajo razumevanje delovanja algoritmov, oblikovanje optimizacijskih modelov in interpretacijo rešitev. Poseben poudarek je na razlagi parametrov algoritmov, primerjavi konvergence ter vplivu nastavitev na vedenje optimizacije. Skripta se zaključijo s pregledom značilnih testnih funkcij in primeri Pareto front za večkriterijsko optimizacijo. Zasnovana so tako, da tudi uporabniki brez poglobljenega matematičnega znanja lahko postopoma razvijejo intuicijo za uporabo optimizacijskih pristopov v realnih inženirskih problemih.

DOI
https://doi.org/10.18690/um.fs.11.2025

ISBN
978-961-299-079-4

Ključne besede:
metahevristične metode,
genetski algoritem (GA),
algoritem rojev delcev (PSO),
eno- in večkriterijska optimizacija,
MATLAB,
inženirske aplikacije

Univerzitetna založba Univerze v Mariboru

DOI
https://doi.org/
10.18690/um.fs.11.2025

ISBN
978-961-299-079-4

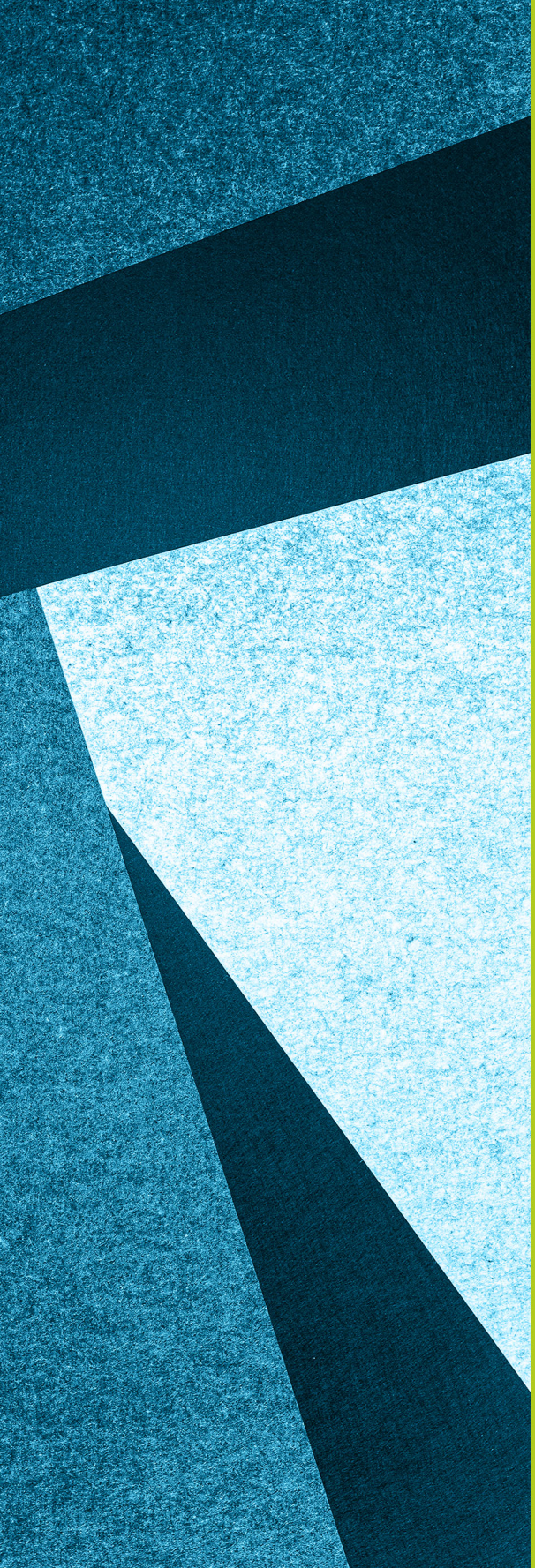
Optimization in Engineering: Solving Problems With Metaheuristic Methods in MATLAB

Janez Gotlih, Mirko Ficko
University of Maribor, Faculty of Mechanical Engineering, Maribor, Slovenia
janez.gotlih@um.si, mirko.ficko@um.si

Keywords:
metaheuristic methods,
genetic algorithm (GA),
particle swarm optimization (PSO) algorithm,
single- and multi-objective optimization,
MATLAB,
engineering applications

The book covers basic optimization approaches in engineering, focusing on the use of metaheuristic methods such as genetic algorithms (GA) and particle swarm optimization (PSO). They are intended for students and engineers who want to understand both the theoretical background and the practical implementation of optimization algorithms in the MATLAB environment. They include chapters on singleobjective and multiobjective optimization problems, discuss constraints, various objective functions, and visualization of results. Each chapter contains structured exercises and assignments for independent work that promote understanding of how algorithms work, the design of optimization models, and the interpretation of solutions. Special emphasis is placed on explaining algorithm parameters, comparing c onvergence, and the impact of settings on optimization behavior. The book concludes with an overview of typical test functions and examples of Pareto fronts for multi-objective optimization. It is designed so that even users without in-depth mathematical knowledge can gradually develop an intuition for using optimization approaches in real engineering problems.

University of Maribor Press



Univerza v Mariboru

Fakulteta za strojništvo

FS