Particle Swarm optimization with Time variable acceleration coefficients

Autori: Boris Rajić, Katarina Šćekić

1. Uvod

Cilj našeg projekta bio je da implementiramo i istražimo **optimizacioni algoritam PSO (Particle Swarm Optimization)** i njegove varijante.
Ideja potiče iz ponašanja jata ptica i roja pčela, u pitanju su čestice koje se kreću kroz prostor, dele informacije o dobrim lokacijama i zajedno pokušavaju da pronađu globalni minimum funkcije.

Projekat je razvijan u više faza:

1. osnovni PSO algoritam,
2. PSO sa vremenski promenljivim koeficijentima (TVAC),
3. PSO sa različitim tipovima čestica — heterogeni tim,
4. testiranje i poređenje timova.

Rad je rađen u Jupyter Notebook okruženju na Pythonu.

2. Test funkcija i osnovna ideja "pustinje"

Na početku smo želeli funkciju koja predstavlja „teren" po kome se čestice kreću.
Koristili smo **Schwefel funkciju** jer ima mnogo lokalnih minimuma i jasno pokazuje ponašanje optimizacije, dok pritom liči na dine u pustinji, što je tematski odgovaralo.

Zatim smo prikazali njen izgled u 3D grafiku da vidimo „teren" optimizacije.

🗺️ Mapa pustinje (Schwefel funkcija)

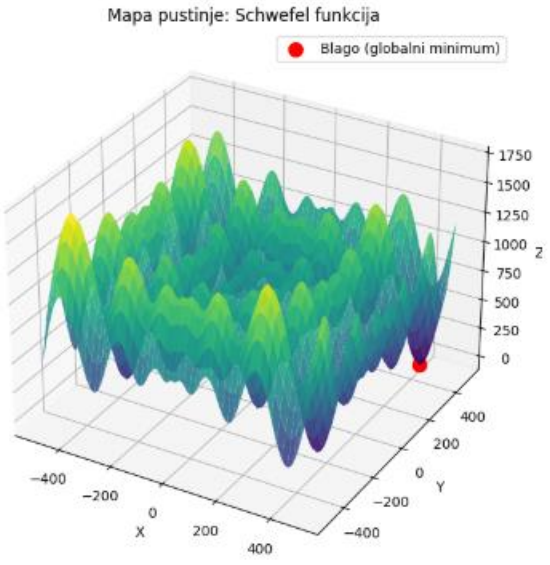
Schwefel funkcija se često koristi za testiranje optimizacionih algoritama jer sadrži **mnoštvo lokalnih minimuma** i samo **jedan globalni minimum** u tački `(420.9687, 420.9687)` sa vrednošću `f(x, y) = 0`.

U kontekstu igre:

- visoke vrednosti `Z` predstavljaju **brda i prepreke**,
- niske vrednosti predstavljaju **dolinu** u kojoj se krije blago.

U sledećoj ćeliji se prikazuje 3D vizuelizacija terena koji čestice treba da istraže.

In [34]:
```
# Vizuelizacija "pustinje" pomoću Schwefel funkcije

def schwefel_2d(x, y):
    """
    Schwefel funkcija 2D:
    f(x, y) = 418.9829*2 - x*sin(sqrt(|x|)) - y*sin(sqrt(|y|))
    Globalni minimum: (420.9687, 420.9687), vrednost = 0
    """
    return 418.9829*2 - x*np.sin(np.sqrt(np.abs(x))) - y*np.sin(np.sqrt(np.abs(y)))

# Kreiranje mreže
x = np.linspace(-500, 500, 400)
y = np.linspace(-500, 500, 400)
X, Y = np.meshgrid(x, y)
Z = schwefel_2d(X, Y)

# 3D prikaz funkcije
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none', alpha=0.9)

# Obeležavanje globalnog minimuma
ax.scatter(420.9687, 420.9687, 0, color='red', s=100, label='Blago (globalni minimum)')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Mapa pustinje: Schwefel funkcija')
ax.legend()
plt.show()
```



3. Osnovni PSO algoritam

U sledećoj fazi definisali smo osnovnu PSO logiku.
Svaka čestica ima:

- **poziciju** (gde se trenutno nalazi),
- **brzinu** (pravac i jačina kretanja),

- **lični najbolji rezultat**,
- **globalni najbolji rezultat** (poznat celom roju).

U svakoj iteraciji čestice razmatraju tri koeficijenta: inerciju, kognitivni(c1) I socijalni (c2).

In [35]:
```
class Particle_default:
    def __init__(self, bounds, f):
        self.lower_bounds = bounds[:, 0]
        self.upper_bounds = bounds[:, 1]
        self.position = np.random.uniform(self.lower_bounds, self.upper_bounds)
        self.velocity = np.random.uniform(-1, 1, size=len(bounds))
        self.f = f

        # Evaluacija funkcije
        self.value = self.f(self.position[0], self.position[1])
        self.personal_best_position = self.position.copy()
        self.personal_best_value = self.value

    def update_velocity(self, global_best_position, w=0.7, c1=1.5, c2=1.5):
        r1 = np.random.rand(len(self.position))
        r2 = np.random.rand(len(self.position))
        cognitive = c1 * r1 * (self.personal_best_position - self.position)
        social = c2 * r2 * (global_best_position - self.position)
        self.velocity = w * self.velocity + cognitive + social

    def move(self):
        self.position = np.clip(self.position + self.velocity, self.lower_bounds, self.upper_bounds)
        self.value = self.f(self.position[0], self.position[1])
        if self.value < self.personal_best_value:
            self.personal_best_position = self.position.copy()
            self.personal_best_value = self.value
```

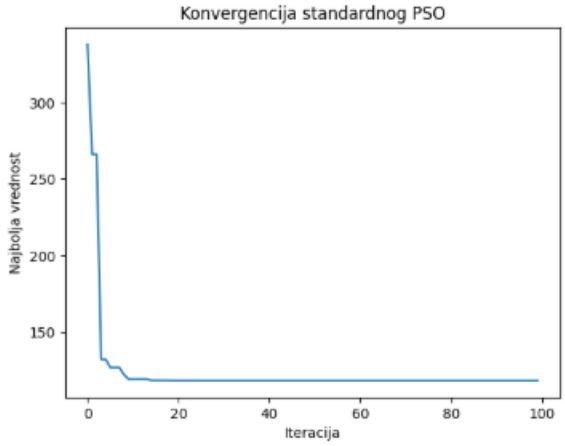
In [36]:
```
def pso(f, bounds, swarm_size=30, num_iters=100, w=0.7, c1=1.5, c2=1.5):
    swarm = [Particle_default(bounds, f) for _ in range(swarm_size)]
    global_best_position = swarm[0].personal_best_position.copy()
    global_best_value = swarm[0].personal_best_value

    history = []

    for _ in range(num_iters):
        for p in swarm:
            p.update_velocity(global_best_position, w, c1, c2)
            p.move()

            if p.personal_best_value < global_best_value:
                global_best_position = p.personal_best_position.copy()
                global_best_value = p.personal_best_value

        history.append(global_best_value)

    return global_best_position, global_best_value, history
```

Nakon svake iteracije pratili smo vrednost funkcije i prikazivali **konvergenciju**, odnosno kako roj sve više prilazi minimumu.

In [37]:
```python
def convergence_speed(history):
    """Računa prosečan pad funkcije po iteraciji."""
    if len(history) < 2:
        return 0.0

    # Uzimamo samo deo gde se realno dešava konvergencija (druga polovina)
    start_val = np.mean(history[:len(history)//2])
    end_val = np.mean(history[-len(history)//4:])

    avg_speed = (start_val - end_val) / (len(history)//2)
    return avg_speed
```

In [71]:
```python
import matplotlib.pyplot as plt

# Definisanje granica (npr. 2D Schwefel)
bounds = np.array([[-500, 500], [-500, 500]])

# Pokretanje standardnog PSO
best_pos, best_val, history = pso(schwefel_2d, bounds, swarm_size=30, num_iters=100)

print("Najbolja pozicija:", best_pos)
print("Vrednost na toj poziciji:", best_val)
speed_pso = convergence_speed(history)
print(f"Brzina konvergencije (PSO): {speed_pso:.4f}")

# Grafikon konvergencije
plt.plot(history)
plt.xlabel("Iteracija")
plt.ylabel("Najbolja vrednost")
plt.title("Konvergencija standardnog PSO")
plt.show()
```

```
Najbolja pozicija: [-302.52491113  420.96872404]
Vrednost na toj poziciji: 118.43836006970918
Brzina konvergencije (PSO): 0.2309
```



Zaključak: PSO se približava minimumu, ali može da „zapne“ u lokalnom minimumu jer sve čestice brzo slede isto rešenje.

4. PSO sa TVAC (Time-Varying Acceleration Coefficients)

Da bismo poboljšali ponašanje, uveli smo **TVAC** koeficijente koji se **menjaju tokom iteracija**.
Na početku daju veću težinu ličnom istraživanju (čestice više lutaju), a kasnije veću težinu grupnom ponašanju (čestice se okupljaju).

Napomena: Ovaj pristup je opisan u:
Y. Shi and R. Eberhart, "A modified particle swarm optimizer," 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), Anchorage, AK, USA, 1998, pp. 69-73, doi: 10.1109/ICEC.1998.699146. keywords:

{Particle swarm optimization;Nonlinear equations;Computational modeling;Evolutionary computation;Genetic programming;Genetic algorithms;Genetic mutations;Educational institutions;Birds;Collaboration},

PSO with TVAC

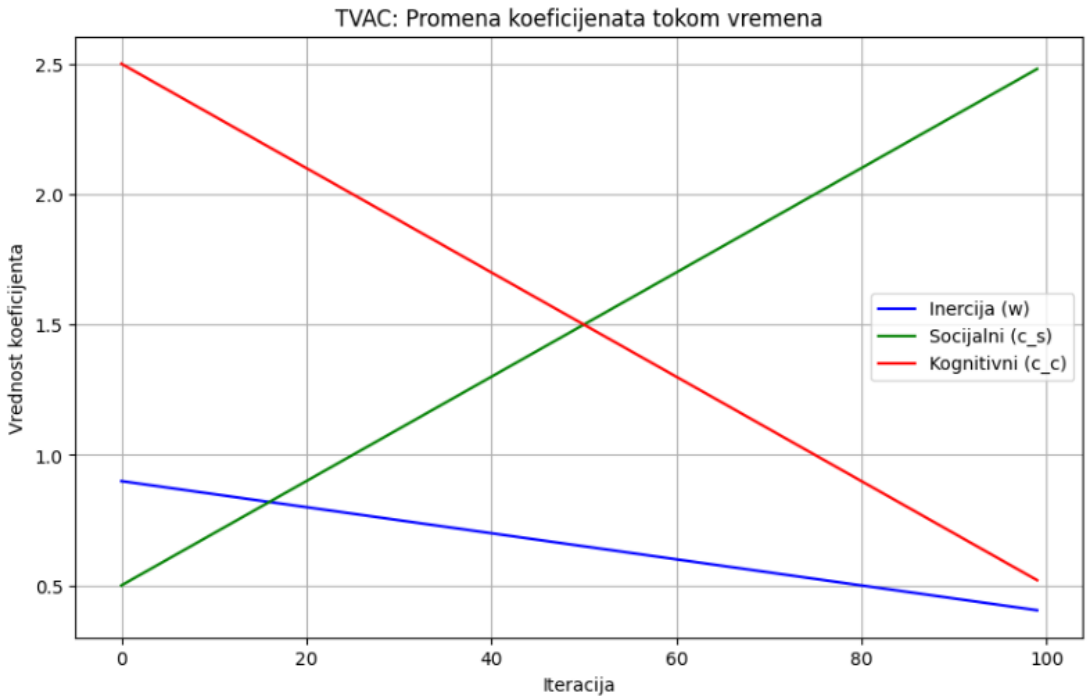
Napomena o TVAC modelu:

Koeficijenti inercije, socijalnog i kognitivnog dela se menjaju tokom iteracija linearnom funkcijom, što je standardni model poznat kao *Time Varying Acceleration Coefficients* (TVAC).

Ovaj pristup je opisan u:
Y. Shi and R. Eberhart, "A modified particle swarm optimizer," 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), Anchorage, AK, USA, 1998, pp. 69-73, doi: 10.1109/ICEC.1998.699146. keywords: {Particle swarm optimization;Nonlinear equations;Computational modeling;Evolutionary computation;Genetic programming;Genetic algorithms;Genetic mutations;Educational institutions;Birds;Collaboration},

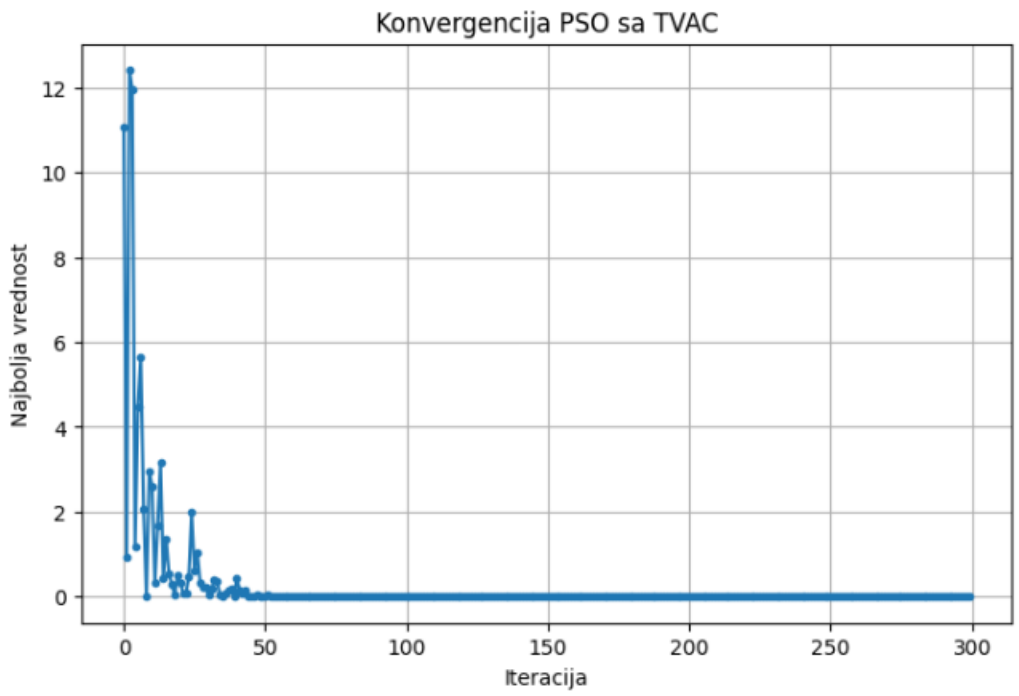
In [39]:
```
def tvac_coefficients(iteration, max_iter,
                      c_inertia_start=0.9, c_inertia_end=0.4,
                      c_social_start=2.5, c_social_end=0.5,
                      c_cognitive_start=0.5, c_cognitive_end=2.5):
    """Vraca trenutne vrednosti c_inertia, c_social i c_cognitive"""
    w = c_inertia_start - (c_inertia_start - c_inertia_end) * iteration / max_iter
    c_s = c_social_start - (c_social_start - c_social_end) * iteration / max_iter
    c_c = c_cognitive_start + (c_cognitive_end - c_cognitive_start) * iteration / max_iter
    return w, c_s, c_c
```

Prikazali smo i istoriju promena koeficijenata:



Rezultat: TVAC verzija postiže stabilniju i bržu konvergenciju, ređe zapinje u lokalne minimume.

```
Najbolja pozicija: [420.96874627 420.96874666]
Vrednost na toj poziciji: 2.5455132345086895e-05
Brzina konvergencije (TVAC PSO): 0.0099
```



5. PSO sa heterogenim timovima

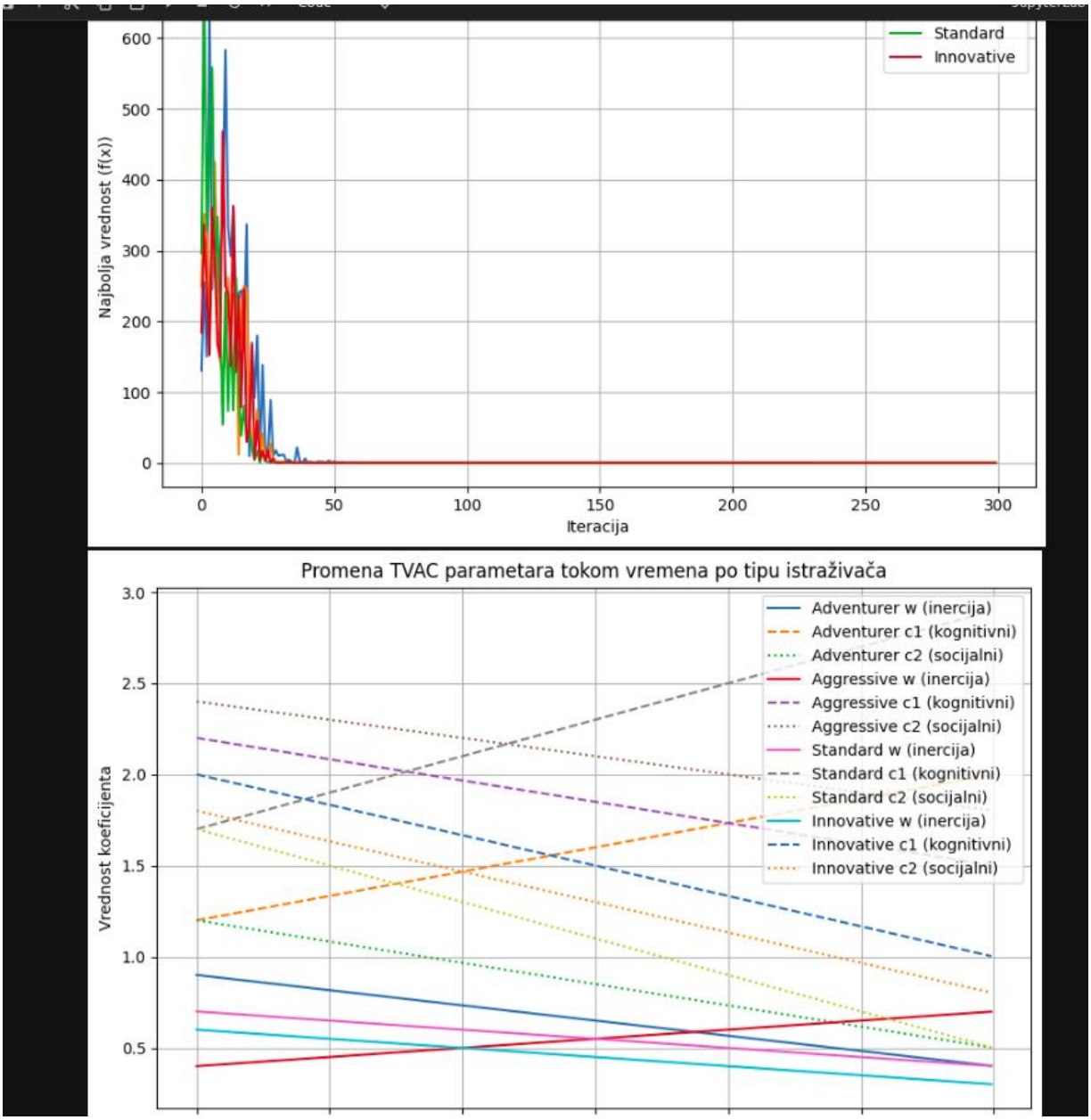
Da bismo uneli više raznovrsnosti, podelili smo roj na **četiri tipa čestica (potklase od Explorer koji je ekvivalent klase Particle u TVAC pristupu)**:

- **Adventurer** – istraživači, kreću se agresivnije i sa više slučajnosti,
- **Aggressive** – brzo jure globalni optimum,
- **Standard** – ponašaju se kao klasičan PSO (Shi & Eberhart),
- **Innovative** – pokušavaju nova rešenja kad napredak uspori.

```python
# TIPOVI EXPLORERA
class Adventurer(Explorer):
    def __init__(self, dim, bounds):
        super().__init__(dim, bounds, w=0.9, c1=1.2, c2=1.2)

class Aggressive(Explorer):
    def __init__(self, dim, bounds):
        super().__init__(dim, bounds, w=0.4, c1=2.2, c2=2.4)

class Standard(Explorer):
    def __init__(self, dim, bounds):
        super().__init__(dim, bounds, w=0.7, c1=1.7, c2=1.7)

class Innovative(Explorer):
    def __init__(self, dim, bounds):
        super().__init__(dim, bounds, w=0.6, c1=2.0, c2=1.8)
    def update_position(self, bounds):
        super().update_position(bounds)
        if np.random.rand() < 0.1:
            self.position += np.random.normal(0, 0.1, len(self.position))
            self.position = np.clip(self.position, bounds[:,0], bounds[:,1])
```

In [*]:
```python
# TVAC po tipu
def tvac_coefficients_by_type(iteration, max_iter):
    coeffs = {}
    coeffs['Adventurer'] = (0.9 - 0.5*iteration/max_iter,
                            1.2 + 0.8*iteration/max_iter,
                            1.2 - 0.7*iteration/max_iter)
    coeffs['Aggressive'] = (0.4 + 0.3*iteration/max_iter,
                            2.2 - 0.7*iteration/max_iter,
                            2.4 - 0.6*iteration/max_iter)
    coeffs['Standard'] = (0.7 - 0.3*iteration/max_iter,
                          1.7 + 1.2*iteration/max_iter,
                          1.7 - 1.2*iteration/max_iter)
    coeffs['Innovative'] = (0.6 - 0.3*iteration/max_iter,
                            2.0 - 1.0*iteration/max_iter,
                            1.8 - 1.0*iteration/max_iter)
    return coeffs
```

Ova ideja omogućava da se tim bolje prilagodi različitim delovima prostora – neki traže nova mesta, dok drugi usavršavaju postojeća rešenja.



Time smo formirali novi tip PSO-TVAC-a, heterogeni tip.

```python
# HETEROGENI PSO
def pso_heterogeneous(num_iters, swarm, bounds, f):
    types = ['Adventurer','Aggressive','Standard','Innovative']
    history_positions_by_type = {t: [] for t in types}
    history_coeffs_by_type = {t: {'w':[], 'c1':[], 'c2':[]} for t in types}

    # Inicijalni global best
    global_best_value = float('inf')
    global_best_position = None
    for p in swarm:
        val = f(p.position[0], p.position[1])
        if val < global_best_value:
            global_best_value = val
            global_best_position = p.position.copy()
        p.best_value = val
        p.best_position = p.position.copy()

    for it in range(num_iters):
        # TVAC parametri po tipu
        coeffs = tvac_coefficients_by_type(it, num_iters)
        for p in swarm:
            t = p.__class__.__name__
            p.w, p.c1, p.c2 = coeffs[t]
            p.update_velocity(global_best_position)
            p.update_position(bounds)
            val = f(p.position[0], p.position[1])

            # Lični best
            if val < p.best_value:
                p.best_value = val
                p.best_position = p.position.copy()
            # Globalni best
            if val < global_best_value:
                global_best_value = val
                global_best_position = p.position.copy()

        # Čuvanje istorije
        for t in types:
            positions_of_type = [p.position.copy() for p in swarm if p.__class__.__name__ == t]
            history_positions_by_type[t].append(positions_of_type)
            ws = np.mean([p.w for p in swarm if p.__class__.__name__ == t])
            c1s = np.mean([p.c1 for p in swarm if p.__class__.__name__ == t])
            c2s = np.mean([p.c2 for p in swarm if p.__class__.__name__ == t])
            history_coeffs_by_type[t]['w'].append(ws)
            history_coeffs_by_type[t]['c1'].append(c1s)
            history_coeffs_by_type[t]['c2'].append(c2s)

    return global_best_position, global_best_value, history_positions_by_type, history_coeffs_by_type
```

```python
# --- Parametri za test ---
NUM_ADVENTURER = 10
NUM_AGGRESSIVE = 10
NUM_STANDARD = 10
NUM_INNOVATIVE = 10
dim = 2
bounds = np.array([[-500, 500], [-500, 500]])
swarm = ([Adventurer(dim, bounds) for _ in range(NUM_ADVENTURER)] +
         [Aggressive(dim, bounds) for _ in range(NUM_AGGRESSIVE)] +
         [Standard(dim, bounds) for _ in range(NUM_STANDARD)] +
         [Innovative(dim, bounds) for _ in range(NUM_INNOVATIVE)])
num_iters = 300

# Pokretanje
global_best_pos, global_best_val, history_positions, history_coeffs = pso_heterogeneous(
    num_iters, swarm, bounds, schwefel_2d
)

print("Globalni najbolji:", global_best_pos)
print("Vrednost:", global_best_val)
```

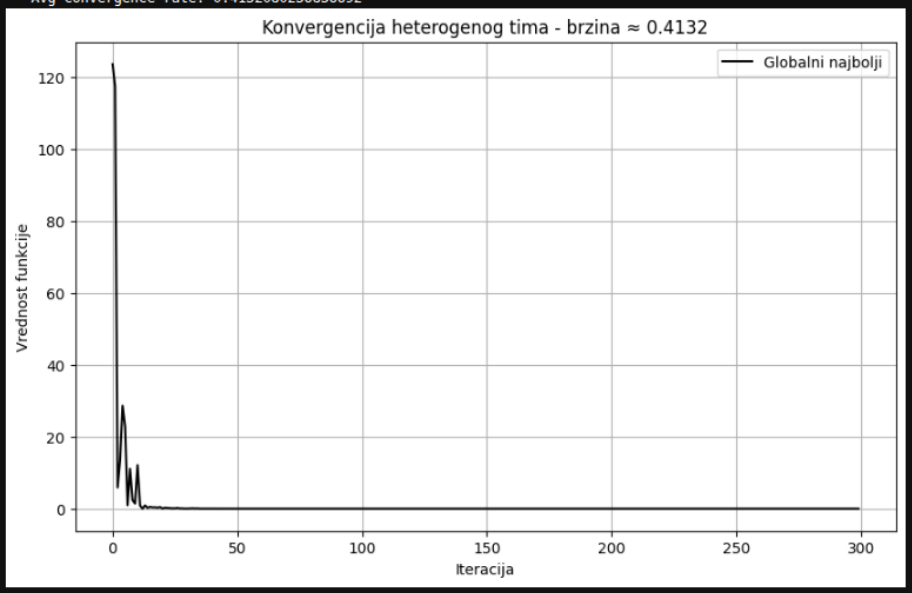
Video snimci konvergencija heterogenih kao I ranijih homogenih rojeva mogu se naci u jupyter sveskama.

7. Zaključak I analiza rezultata:

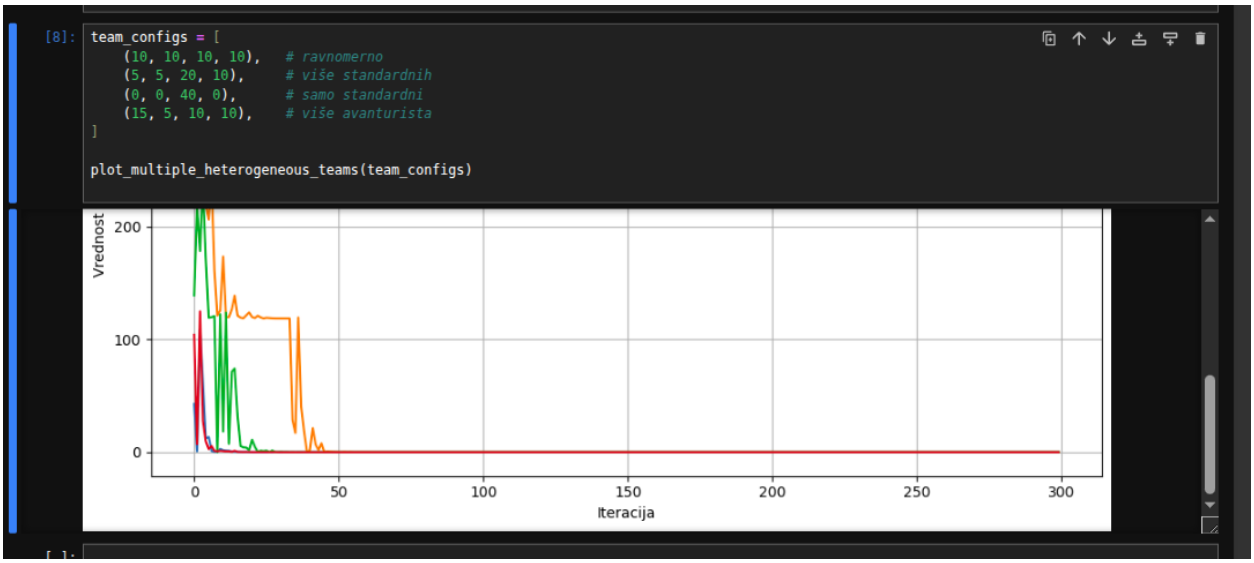
Pokrenuli smo više konfiguracija timova (različiti brojevi svake vrste čestica) i uporedili rezultate.
Timovi sa uravnoteženim brojem standardnih i avanturističkih čestica pokazali su najbolje rezultate.
Samo standardne čestice su bile stabilne, ali sporije; samo agresivne ili avanturističke su često previše „skačuće“.
Kombinovani timovi imaju bolju ravnotežu između istraživanja i iskorišćavanja.

[6]:
```
run_heterogeneous_team(2, 2, 16, 2);
```

```
Team A=2, Ag=2, S=16, I=2
  Globalni najbolji: [420.96874599 420.96874678]
  Vrednost: 2.5455132345086895e-05
  Avg convergence rate: 0.4132080236836692
```



[8]:
```
team_configs = [
    (10, 10, 10, 10),   # ravnomerno
    (5, 5, 20, 10),     # više standardnih
    (0, 0, 40, 0),      # samo standardni
    (15, 5, 10, 10),    # više avanturista
]

plot_multiple_heterogeneous_teams(team_configs)
```



8. Dalji rad

Moguća poboljšanja:

- Višedimenzionalni test-funkcijski prostori (npr. dim > 2).
- Dinamička adaptacija tipova čestica tokom simulacije.
- Primena neuronskih mreza kako bi se pronasle optimalne kombinacije timova.
- Primena na realne probleme (npr. rutiranje, raspoređivanje, podešavanje hiperparametara).