PSO OPTIMALIZACE V MATLABU

M. Čapek, P. Hazdra

Katedra elektromagnetického pole, ČVUT - FEL, Technická 2, 166 27 Praha

Abstrakt

Příspěvek se věnuje implementaci PSO algoritmu v Matlabu a presentaci jednotlivých optimalizačních úloh. PSO[1] je stále relativně novou, velice rychlou a robustní techniku optimalizace, která podává ve většině testovaných případů výborné výsledky. Základní úlohu lze dále modifikovat zavedením tzv. *zdí* pro zrychlení celého procesu. Závislost výsledného řešení na vstupních parametrech, ale také efektivita celé funkce bude představena na několika příkladech. Průběh optimalizace lze efektivně modeloval pomocí několika parametrů, což ukázaly i uvedené testy. Vytvořeným programem lze řešit takřka libovolný problém, stačí pouze, aby vztah mezi vstupem a (minimalizovaným) výstupem popisovala funkce (tzv. *fitness funkce*) naprogramovatelná v Matlabu. V tomto článku jsou – s jedinou výjimkou – uvedeny pouze různě složité matematické funkce; důraz je kladen na PSO.

1 Úvod

Od roku 1995, kdy bylo PSO poprvé představeno [3], se objevila celá řada studií a výzkumů, které PSO využívají. Na základě potřeby testovat jejich závěry, ale i rozvíjet vlastní aktivity (zejm. spojené s IFS anténami, viz [6], [7], [8]), bylo rozhodnuto vyvinout vlastní algoritmus. Mezi požadavky byla prioritně rychlost algoritmu, jeho univerzálnost, možnost měnit jednotlivé parametry (iterace, počet agentů ...), dále schopnost rekurzivního volání[2], řešitelnost problémů libovolné dimenze a v neposlední řadě stabilita (zdaleka ne všechny scénáře zadané algoritmu jsou řešitelné – může jít např. o fyzikální omezení v souvislosti s optimalizací rozměrů antény).

Pro realizaci tohoto algoritmu bylo zvoleno prostředí Matlab. Matlab obsahuje velice rychlé jádro pro práci s maticemi, disponuje celou řadou knihoven a lze se tedy soustředit přímo na řešený problém, umožňuje komformní grafický i textový výstup. Velká část literatury obsahuje příklady právě z Matlabu, lze tedy porovnávat jednotlivá řešení a pro partikulární problémy využívat publikované segmenty kódu (např. [5], [2]). Velkou výhodou je také možné propojení s Comsolem, ve kterém lze využít dutinového modelu pro modální řešení antén.

Výsledná funkce, representovaná programem *PSOptimizer*, respektuje všechny výše uvedené požadavky. Podrobněji ji budou věnovány části 3 a 4.

2 PSO algoritmus

Nejprve se však krátce zmíníme o principu PSO. Optimalizace vychází z rojové inteligence pozorované např. u včelstev a napodobuje jejich chování. V některých aspektech je PSO podobná ACO[3], v jiných můžeme nalézt paralelu s GA[4]. Ve všech případech jde o samo se organizující systémy vykazující silné kolektivní chování. Zásadní rozdíl však spočívá v přístupu k členu hejna (agentovi), nad kterým jsou definovány určité operace a disponuje částí znalostí, které má celý roj. Uživatel stanoví prostor (*solution space*, dále jen s.s.), nad kterým je definována optimalizovaná funkce a v kterém hledá její minimum. V mnoho aplikacích nemá řešení mimo s.s. smysl, proto se snažíme udržet agenty v určeném prostoru (podrobněji níže).

[1] *PSO* – Particle Swarm Optimalization

[2] Návratová hodnota funkce musí obsahovat nejen optimalizované údaje, ale i informace o nastavení PSO, jednotlivých generacích, konvergenci ke globálnímu minimu atd.

[3] *ACO* – Ant Colony Optimization

[4] *GA* – Genetic Algorithm

Každý agent si pamatuje svůj dosavadní nejlepší objev (*pbest*, proměnná p_{id}^n v rovnici (1)). Nejlepší objev celého hejna (*gbest*, p_{gd}^n) je potom tím nejlepším ze všech osobních objevů. Chování jednotlivých agentů popisují následující dva vztahy. Nejprve uveďme výpočet rychlosti agenta. Rychlost je stanovena v každé iteraci pro každého agenta zvlášť a ovlivňuje směr, jímž se pohybuje. Počet složek tohoto vektoru je rovný dimenzi (index d) řešeného problému.

$$v_{id}^{n+1} = wv_{id}^n + c_1 r_1^n (p_{id}^n - \chi_{id}^n) + c_2 r_2^n (p_{gd}^n - \chi_{id}^n) \tag{1}$$

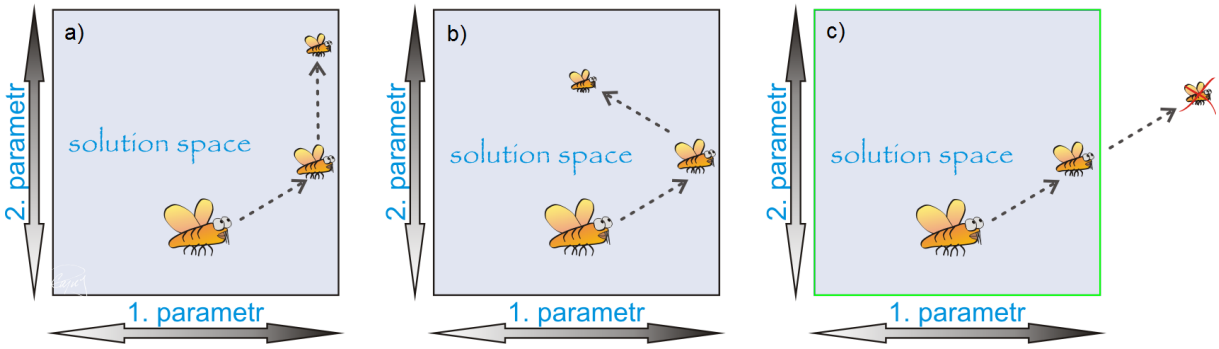
V Matlabu je potřeba vsadit tuto rovnici do *for* cyklu, čímž je zajištěno procházení celého hejna (index i označuje agenty). Zabalením do dalšího *for* cyklu umožníme iterování celého problému od $n = 1$ do $n = N$ (paralela k jednotlivým generacím u GA). Dále je nutno objastnit význam zbylých proměnných a konstant.

Koeficient w je často nazýván váhovací faktor (*weighted factor*). Může být po celou dobu optimalizace konstantní, nebo se může měnit (potom je zadávana dvojice parametrů w_{max} a w_{min}). Klesající koeficient zabraňuje nepříjemným oscilacím a zároveň stimuluje hejno ke konvergenci nad nalezeným globálním minimem. Parametry c_1 a c_2 udávají nakolik bude výsledná rychlost odvozena od osobního minima daného agenta a nakolik od společného globálního min. Jejich optimální velikost bude diskutována v části 5. Bez komentáře zůstaly již pouze hodnoty r_1^n a r_2^n. V obou případech se jedná o náhodně generovaná čísla s normálním rozložením v rozsahu (0,1), pro tyto účely je v Matlabu k dispozici funkce *rand()*.

Víme-li již, jakým směrem a jak rychle se pohybují agenti, můžeme aktualizovat jejich pozici:

$$\chi_{id}^{n+1} = \chi_{id}^n + v_{id}^{n+1} \Delta t. \tag{2}$$

Rovnice (2) odpovídá pohybové rovnicí. Nové umístění agenta tedy získáváme jako součet koordinátů v minulé iteraci pozměněné o pohyb v současné iteraci. Protože Δt může obecně nabývat libovolných hodnot, lze přiřadit $\Delta t = 1$ (jednotkový čas).



Obrázek 1: Typy zdí používané v PSO

Vztahy uvedené výše žádným způsobem neomezují pohyb agentů, mohou tedy opustit s.s. Z toho důvodu bylo navrženo několik technik, které zajistí, aby se agenti nerozběhli daleko za hranice s.s. Mezi nejcitovanější a nejužívanější patří následující:

- Omezení maximální rychlosti agenta (v_{id}^{n+1})
- Ohraničující zdi:
  1. Absorbční zeď (*Absorbing Wall*, obr. 1a)
  2. Odrazná zeď (*Reflecting Wall*, obr. 1b)
  3. **Neviditelná zeď** (*Invisible Wall*, obr. 1c)

První uvedená možnost, tedy omezení rychlosti agentů, byla využita např. v [4]. Rychlost je omezena i uvnitř s.s., což není vhodné. Lepším řešením je použití jedné ze zdí. Ty popisují způsob, jakým je naloženo s agentem, pokud překročí povolené hranice. Neovlivňují průběh optimalizace uvnitř s.s. a navíc, pokud již agent opusil prohledávaný prostor, ho efektivně nasměrují zpět. Podrobněji bude popsána pouze zeď z obr. 1c, využívaná funkcí *PSOptimizer*, ostatní jsou popsány např. v [2].

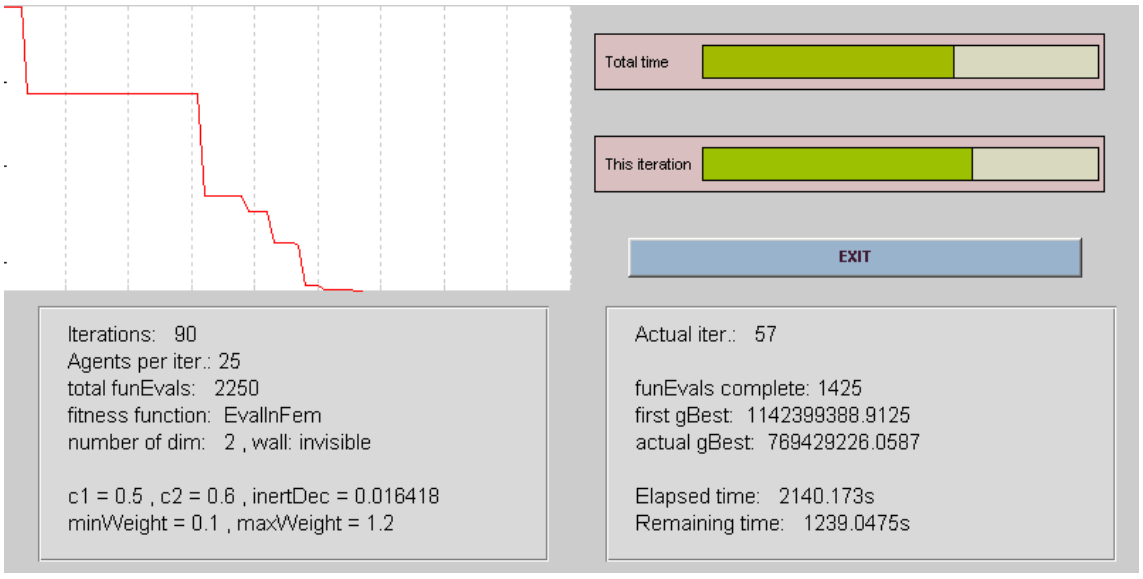
Jak můžeme vyčíst z obrázku, agentům je umožněno vyletět ze s.s., ovšem po jeho opuštění není vyhodnocována fitness funkce. To zapříčíní pozvolný návrat agentů zpět do s.s. Velkou předností je výrazná úspora výpočetního času, poněvadž jsou vyhodnocována pouze ta schémata, o která se skutečně zajímame. I z tohoto důvodu se jedná o pravděpodobně nejlepší řešení pro většinu inženýrských problémů. Na závěr zrekapitujme důležité parametry PSO optimalizace:

Parametr	
c_1	poznávací parametr (*cognitive rate*)
c_2	sociální parametr (*social rate*)
w_{min}	váhovací koeficient (na konci optimalizace)
w_{max}	váhovací koeficient (na začátku optimalizace)
Δt	časová konstanta (nejčastěji volena jednotková, tj. $\Delta t = 1$)
iterace	počet iterací (vliv velikosti iterací viz níže)
počet agentů	počet agentů nad s.s. (viz níže)
p_i^n	(současné) individiální nalezené minimum agenta ($f_{min}(x_i)$)
p_g^n	(současné) globální nalezené minimum agenta $f_{min}(x_i)$

Tabulka 1: Shrnutí parametrů

3 Implementace

Jak již bylo uvedeno, optimalizace je implementována do prostředí Matlab. Pro části programu jsme užili lokálních funkcí, fitness funkce je řešena separátně (definuje ji uživatel pro konkrétní problém). V inicializační části je ověřeno, jsou-li hodnoty správně zadány, je stanovena počáteční generace agentů a vykresleno grafické rozhraní.



Obrázek 2: GUI PSOptimizeru

GUI programu je navržen tak, aby byl jednoduchý[5], ale zároveň podrobně informoval o stávajícím stavu optimalizace. Text v levém rámu zobrazuje nastavené výchozí parametry, v pravé části potom aktuální průběh. Stupnice na ose y grafu zobrazujícího cost funkci je v (zde názornějším) logaritmickém měřítku. Tlačítko `Exit` ukončuje optimalizaci, nejdříve však po dokončení dané iterace. I v tomto případě se vrací výsledky, kterých bylo dosaženo. Vyjdeme-li ze základního tvaru volání funkce:

$$res = PSOptimizer(PsoData,'fitnessFunction',ag,it),$$

kde *ag* je počet agentů, *it* počet iterací a řetězec *fitnessFunction* obsahuje jméno mfile souboru s fitness funkcí, vytváříme potom následující pole vstupu a výstupu:

```
PsoData.data1 = []
PsoData.data2 = []
PsoData.data3 = []
PsoData.type = 'psopt'
PsoData.rank
PsoData.bound{} = []
PsoData.cond{} = []
```

```
res.data1 = []
res.data2 = []
res.data3 = []
res.done
res.score
res.type = 'optim'
res.history.populPosition = []
res.history.iter = []
res.history.value = []
res.history.psoDta = []
res.options. ...
```

Předně do funkce vstupují 3 sloty (PsoData.data1-3), ve kterých se mohou umístit optimalizovaná data. Velikost matic data1-3 je libovolná, vždy však musí být všechny tři matice definovány (alespoň jako prázdné). Počet slotů volíme tak, aby byl dostatečný pro řešení IFS antén (body základního útvaru, transformace a iterační matice). Dalším polem je PsoData.type, které je fixně zadáno string řetězcem *psopt*. Pole je potřeba při identifikaci příchozích dat uvnitř funkce. PsoData.rank je nepovinný údaj, jenž koresponduje s dimenzí optimalizace. Na závěr zde figurují pole PsoData.bound, ve kterém jsou uloženy hranice[6] definující s.s. a také PsoData.cond, indexující optimalizované parametry[7].

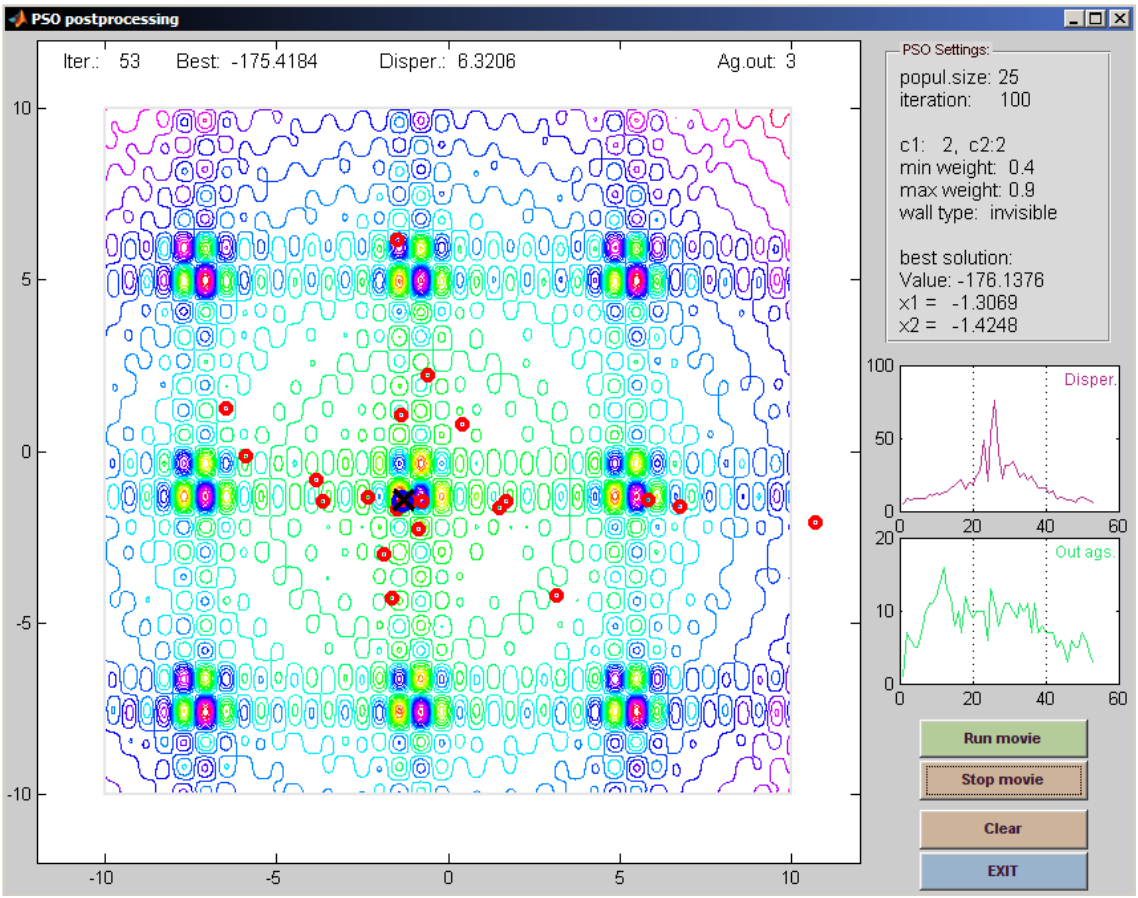
Matice PsoData.cond může mít libovolný počet řádek. Protože každá řádka označuje jednu optimalizovanou pozici v jedné z matic, lze takto svázat do jedné dimenze (PsoData.cond{dim}) několik optimalizovaných hodnot. Takové hodnoty jsou potom v rámci PSO spřaženy a jsou optimalizovány společně. Tento postup je ideální v případě, že potřebujeme mezi některými hodnotami zachovat fixní poměr (jejich rovnost). Uvedeným postupem zajistíme, že do funkce vstupují všechna uživatelská data (fitness funkce tedy může s těmito daty komplexně pracovat) a navíc ucelený údaj co a v jakých mezích se má optimalizovat.

Výstupní pole obsahuje opět sloty 1-3, patřičné hodnoty jsou ale optimalizovány. Hod-

[5]Pokud je program volán opakovaně, nezatěžuje PC.

[6]Způsob zápisu: $ProData.bound\{dimenze\} = [min\ max]$ pro danou dimenzi.

[7]Formát je následující: $PsoData.cond\{dimenze\} = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & \vdots & \vdots \end{pmatrix}$, kde a odpovídá číslu datového slotu 1-3, b ukazuje na řádku daného slotu, c potom značí pozici na řádce (číslo sloupce).



Obrázek 3: Pozice agentů v dané iteraci pro funkci Levy5, 10x10

nota fitness funkce pro tyto je uvedena v res.score. Pole done referuje o zdárném ukončení PSO ($res.done = 1$), v opačném případě je $res.done = 0$ (určeno jako návěstí pro nadřazené programy). Res.options uvádí hodnoty, za kterých byla optimalizace dokončena a konečně res.history obsahuje většinu vnitřních stavů *PSOptimizeru* v každé iteraci[8].

4 Vybrané funkce

Algoritmus byl testován následujícími funkcemi:

- Kvadratická funkce
- Rosenbrockova funkce
- Funkce Levy No.5

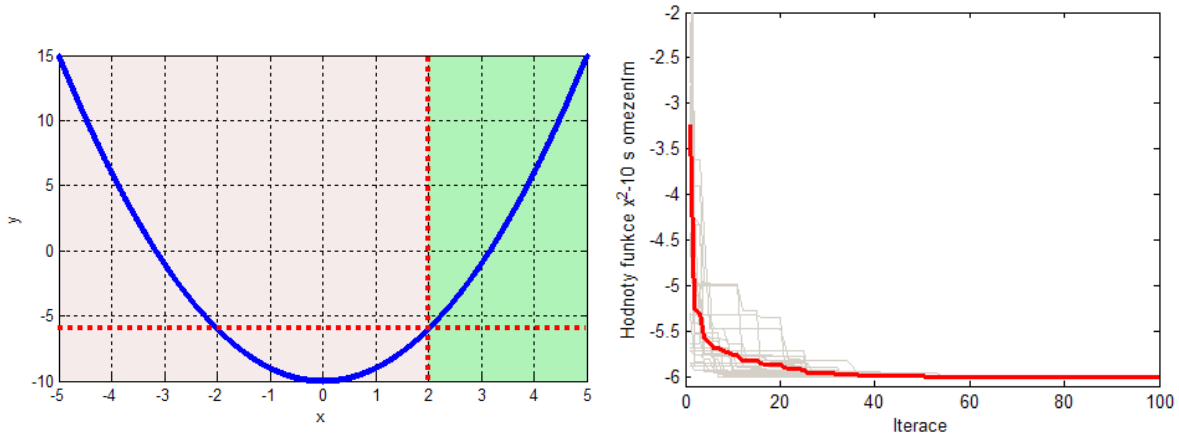
Funkce jsou rozdílně členité (krajina, kterou se pohybují agenti) a jsou dostatečně popsány v literatuře, lze tedy verifikovat získané hodnoty. Algoritmus byl využit i pro optimalizaci jiných funkcí (Levy No.3, Rastigrinova funkce...), ve všech případech PSO podalo očekávané výsledky. Metodika využitá pro hodnocení výsledků optimalizace respektuje obvyklé postupy ([1] a další).

Funkce 1 (kvadratická)

$$f_{kv}(x) = x^2 - 10 \tag{3}$$

[8]Tyto pole lze využít např. pro vykreslení COST funkce (např. obr. 4 vpravo pro kvadratickou funkci), nebo pro vykreslení pozice jednotlivých agentů v dané iteraci (screenshot 53. iterace je vidět na obr. 3).

Na této funkci demonstrujeme vliv neviditelné zdi[9]. Pokud s.s. zvolíme pouze v rozsahu $\langle 2, 5\rangle$, bude nalezené minimum rovno -6, a to i přes to, že jednotliví agenti mohou krátkodobě proniknout i blíže k 0 na ose x. Situace je znázorněna na obr. 4.



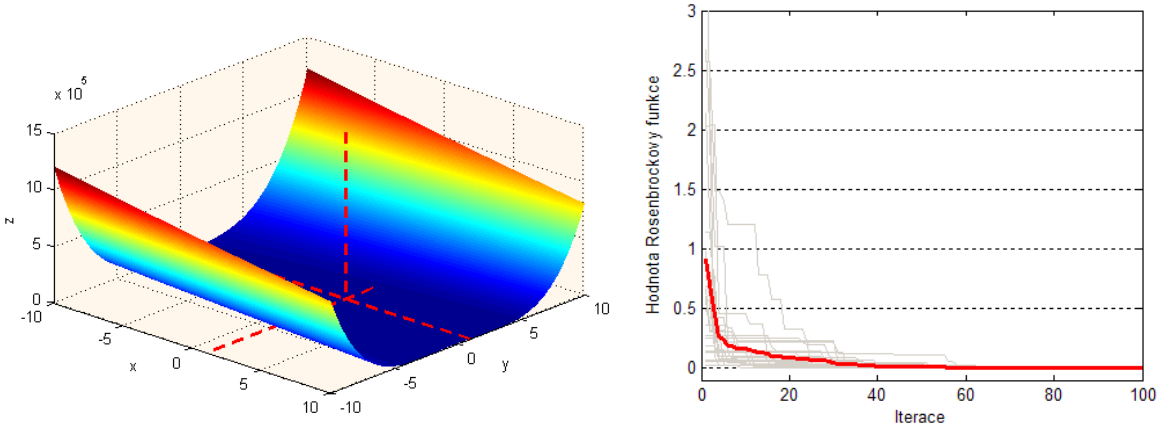
Obrázek 4: Optimalizace kvadratické funkce v rozsahu $x \in \langle 2, 5\rangle$

Funkce 2 (Rosenbrockova)

$$f(x) = \sum_{i=1}^{N} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right) \tag{4}$$

$$f_{ro}(x, y) = 100(y - x^2)^2 + (x - 1)^2 \tag{5}$$

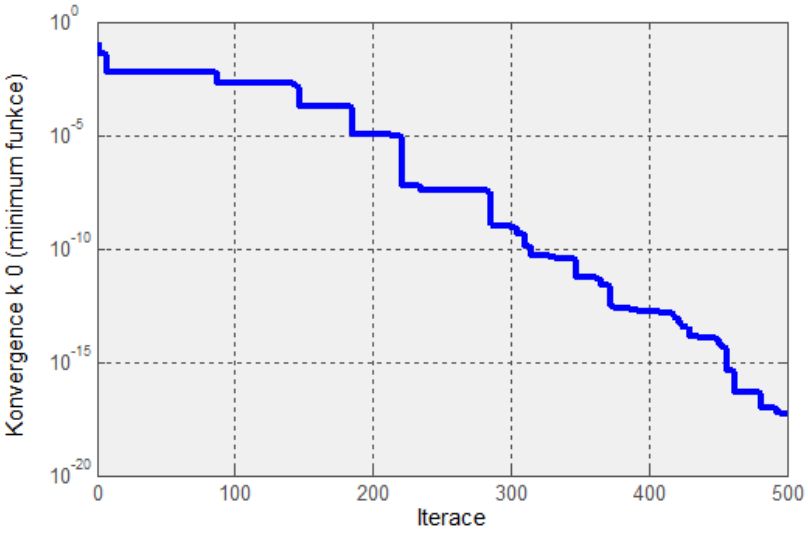
Funkce se široce uplatňuje ve většině optimalizační testů. Pro svou monotonii (obrázek 5) lze využít i gradientní metody, neboť nehrozí uvíznutí v lokálním minimu. Globální mininum můžeme nalézt na souřadnicích $x = 1$, $y = 1$ a hodnota $f(x_{min}, y_{min}) = 0$. Podle rovnice (4) je funkce definována v [2], využijeme však jednoduššího tvaru (5). Obr. 6 ukazuje, jak rychle klesá chyba,



Obrázek 5: Rosenbrockova funkce, s.s. $\in \langle -10, 10\rangle \times \langle -10, 10\rangle$ a cost funkce

s kterou je nalezeno globální minimum. Tuto rychlost lze výrazně ovlivnit nastavením parametrů w_{min} a w_{max}. Pokud váhovací koeficient s iterací klesá, klesá také příspěvek nově vypočtené rychlosti a agent je spíše udržován v současné pozici. Eliminujeme tím oscilace, při kterých agenti kmitají kolem nalezeného minima.

[9]Neviditelná zeď se uplatňuje nepřímo tím, že neumožní vyhodnotit možné nižší minimum v rozsahu $\langle -2, 2\rangle$. Vybraná pole PsoData pro tuto optimalizaci jsou rovna: *PsoData.rank* = 1; *PsoData.cond*{*1*} = [1 1 1]; *PsoData.bound*{*1*} = [2 5].

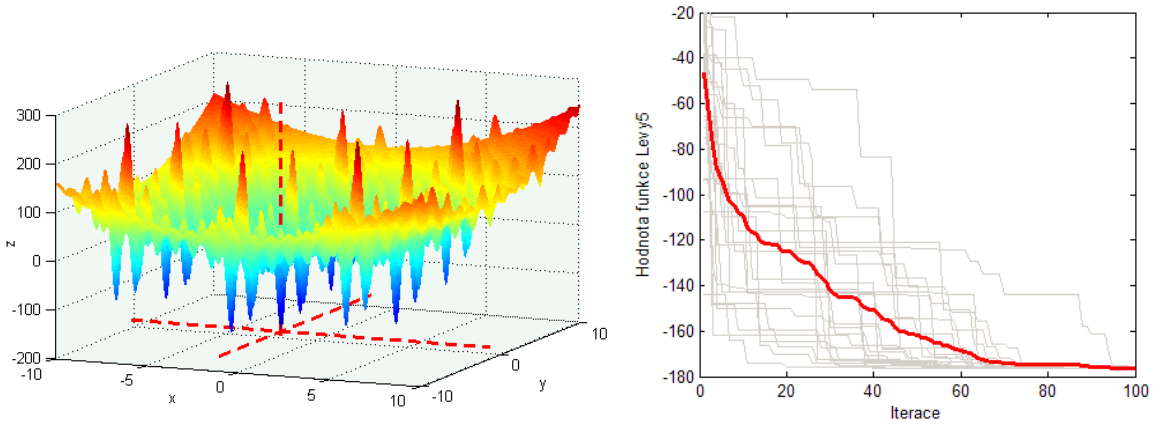


Obrázek 6: Konvergence ke skutečnému minimu Rosenbrockovy funkce

Funkce 3 (Levy No.5)

$$f_{l5}(x,y) = \sum_{i=1}^{5}\Big(i\cos\Big((i-1)x+i\Big)\Big)\sum_{j=1}^{5}\Big(j\cos\Big((j+1)y+j\Big)\Big)+(x+1.42513)^2+(y+0.80032)^2 \quad (6)$$

Na prostoru $\langle -2,2\rangle \times \langle -2,2\rangle$ můžeme nalézt 760 lokálních a jedno globální minimum (souřadnice $x = -1.3068, y = -1.4248$). Velikost s.s. je úmyslně zvětšena na $\langle -10,10\rangle \times \langle -10,10\rangle$ (jako u Rosenbrockovy f.). Tato funkce dobře testuje odolnost vůči uvíznutí agentů v lokálních extrémech.



Obrázek 7: Funkce Levy No.5, s.s. $\in \langle -10,10\rangle \times \langle -10,10\rangle$ a cost funkce

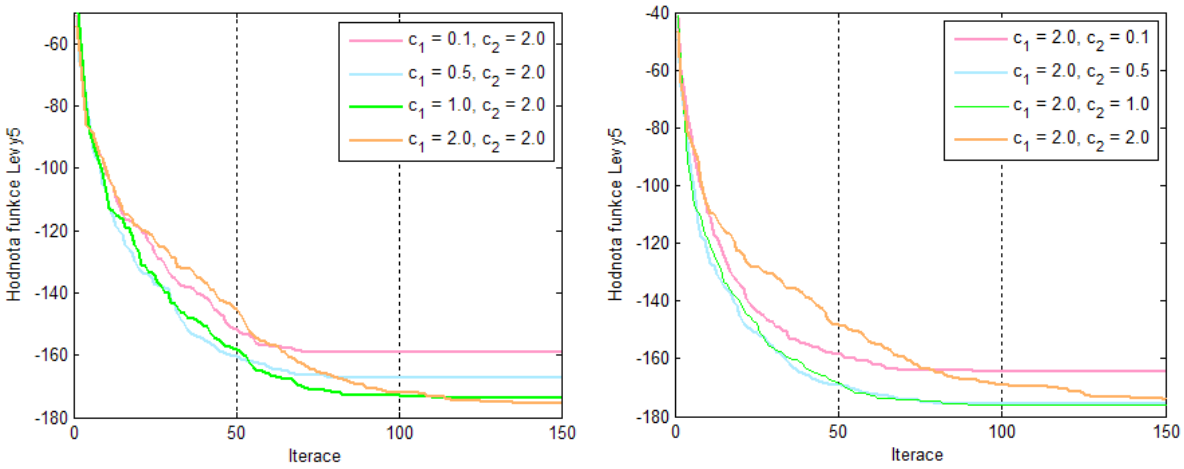
5 Optimální parametry PSO

Výsledek, stejně tak jako rychlost, s jakou je řešení nalezeno, lze významným způsobem ovlivnit vhodnou volbou parametrů z tabulky 1[10]. Jednotlivé optimalizace jsou velkou měrou závislé na náhodě a jakékoliv údaje musíme získat jako průměr velkého počtu opakování (zpravidla 50). Celá procedura se tak stává časově náročnější.

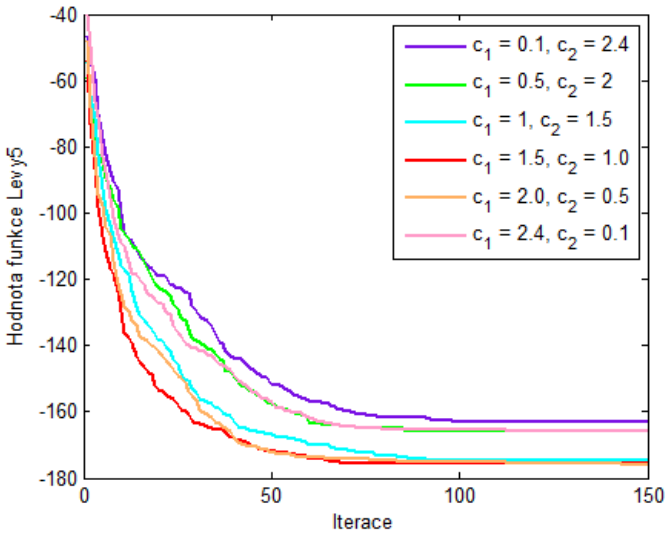
Nejprve uveďme grafické průběhy. Jejich úkolem není zobrazit nalezenou hodnotu, sledujeme zde průběh funkčních hodnot v čase. Tento typ grafů je v anglické literatuře označován jako

[10]Konkrétně se budeme zabývat počtem iterací a velikostí hejna, a také koeficienty c_1 a c_2.

cost funkce a dává dobrou představu o průběhu optimalizace. Zajímavé jsou zejména růžové a fialová křivka. Na nich vidíme, že v případě zcela nevhodně nastavených parametrů nekonverguje hejno dostatečně.



Obrázek 8: Optimalizace funkce Levy5 pro různá c_1 a c_2 (20 agentů, 150 iterací)



Obrázek 9: Optimalizace funkce Levy5 pro různá c_1 a c_2 (20 agentů, 150 iterací)

Následující tabulky demonstrují zásadní vliv parametrů c_1 a c_2 na konečný výsledek optimalizace. Úspěšnost algoritmu testujeme podmínkou, zda je nalezená hodnota menší než -176.1375 (Levy5, hodnota blízká známému globálnímu minimu, viz [1]).

Z aplikačního hlediska jsou zajímavé dva údaje. První je ten, kdy je úspěšnost poprvé rovna 100%, tedy chvíle, kdy optimalizace vždy najde správný výsledek. Tento moment lze určit pouze pro již zmapované funkce. Druhý zajímavý výsledek je ten s nejvyšší účinností v nejkratším možném čase[11]. Vidíme tedy, že úspěšnost funkce lze významným způsobem zvýšit úpravou parametrů (c_1, c_2, ale i dalšími, kterými se zde nebudeme zabývat), což popisují některé studie. Bohužel optimalizované funkce mají různý charakter, a proto jsou tyto parametry voleny zpravidla na základě doporučení v referenční literatuře. Pro úplnost uveďme i účinnost optimalizace při změně počtu agentů (tabulky 4 a 5, níže).

Počet agentů je zpravidla pevně stanoven (nejčastěji 20-25 jedinců). Zvýšit jejich počet je

[11] V přímé úměře odpovídá případu s nejmenším celkovým počtem vyhodnocení fitness funkce (v tabulce evalFun)

Funkce Levy5 (10x10 s.s.)		50x spuštěno, 20 agentů		
iterace	úspěšnost [%]	chyb	celkový čas [s]	evalFun
$c_1 = 2.0, c_2 = 2.0, w_{min} = 0.4, w_{max} = 0.9$				
50	5%	48/50	187	1000
100	78%	11/50	378	2000
150	92%	4/50	571	3000
300	**100%**	0/50	**1258**	6000
500	100%	0/50	2308	10000
$c_1 = 0.5, c_2 = 0.6, w_{min} = 0.4, w_{max} = 0.9$				
100	80%	10/50	387	2000
$c_1 = 1.0, c_2 = 1.0, w_{min} = 0.4, w_{max} = 0.9$				
100	**92%**	4/50	**387**	2000

PC: HP Compaq nx6125 (Turion64 1.86GHz, 768MB 333MHz)

Tabulka 2: *Success rate* funkce Levy5 (se změnou iterace), 20 agentů

Funkce Levy5 (10x10 s.s.)		50x spuštěno, 45 agentů		
iterace	úspěšnost [%]	chyb	celkový čas [s]	evalFun
$c_1 = 2.0, c_2 = 2.0, w_{min} = 0.4, w_{max} = 0.9$				
50	16%	42/50	370	2250
100	98%	1/50	711	4500
150	98%	1/50	1083	6750
300	**100%**	0/50	**2328**	13500
500	100%	0/50	¿4000	22500
$c_1 = 0.5, c_2 = 0.6, w_{min} = 0.4, w_{max} = 0.9$				
50	72%	14/50	362	2250
$c_1 = 1.0, c_2 = 1.0, w_{min} = 0.4, w_{max} = 0.9$				
50	**90%**	5/50	**363**	2250

PC: HP Compaq nx6125 (Turion64 1.86GHz, 768MB 333MHz)

Tabulka 3: *Success rate* funkce Levy5 (se změnou iterace), 45 agentů

Funkce Levy5 (10x10 s.s.)		50x spuštěno, 150 iterací		
agentů	úspěšnost [%]	chyb	celkový čas [s]	evalFun
$c_1 = 2.0, c_2 = 2.0, w_{min} = 0.4, w_{max} = 0.9$				
5	44%	28/50	84	750
10	64%	18/50	121	1500
20	92%	4/50	198	3000
30	**100%**	0/50	**276**	4500
45	100%	0/50	370	6750

PC: 64bit Core2Quad 9450 (2.66GHz),12MB cache, 4GB 1333MHz

Tabulka 4: *Success rate* funkce Levy5 (podle počtu agentů), 150 agentů

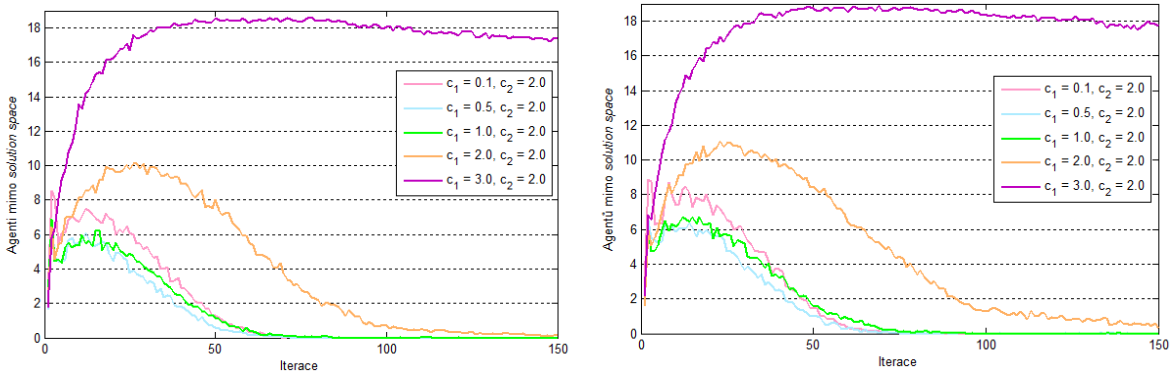
doporučeno u problémů, které jsou definovány na velmi rozlehlém s.s., anebo (zejména) pokud prohledávaný prostor obsahuje mnoho lokálních minim. Mnohem častěji je v případě problémů navýšen počet iterací, případně jsou pozměněny parametry c_1 a c_2.

Na závěr krátce okomentujme obrázky 10 a 11. Vynášíme zde dva parametry, které popisují

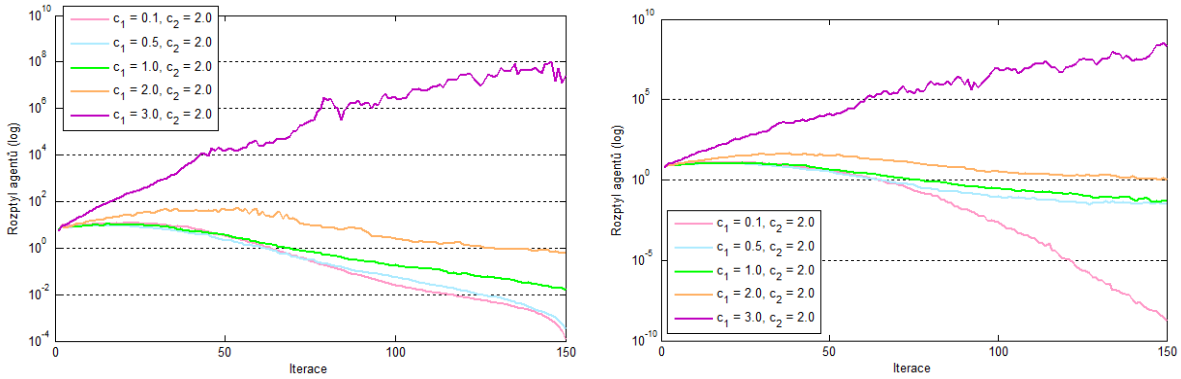
Funkce Levy5 (10x10 s.s.)		50x spuštěno, 50 iterací		
agentů	úspěšnost [%]	chyb	celkový čas [s]	evalFun
$c_1 = 2.0, c_2 = 2.0, w_{min} = 0.4, w_{max} = 0.9$				
5	0%	50/50	32	250
10	0%	50/50	42	500
20	2%	49/50	65	1000
30	6%	47/50	88	1500
45	12%	44/50	118	2250
$c_1 = 0.5, c_2 = 0.6, w_{min} = 0.4, w_{max} = 0.9$				
45	76%	12/50	160	2250
$c_1 = 1.0, c_2 = 1.0, w_{min} = 0.4, w_{max} = 0.9$				
45	**92%**	4/50	**154**	2250
$c_1 = 1.5, c_2 = 1.5, w_{min} = 0.4, w_{max} = 0.9$				
45	90%	5/50	165	2250

PC: 64bit Core2Quad 9450 (2.66GHz),12MB cache, 4GB 1333MHz

Tabulka 5: *Success rate* funkce Levy5 (podle počtu agentů), 50 iterací



Obrázek 10: Počet agentů mimo s.s. pro Rosenbrockovu a Levy5 funkci (20 agentů, 150 iterací)



Obrázek 11: Rozptyl agentů, Rosenbrockova a Levy5 funkce (20 agentů, 150 iterací)

optimalizaci nezávisle na zkoumané funkci. Počet agentů mimo s.s., stejně tak jako jejich rozptyl[12], lze pro stejně velké s.s. vzájemně porovnávat. Na první pohled je patrné, že – ač pro zcela rozdílné funkce – jsou si průběhy dosti podobné a napovídají, jakým způsobem se pohybuje hejno během optimalizace. Může být předmětem dalšího výzkumu, zda lze této závislosti využít k zefektivnění PSO.

[12]Definován: $\sigma = \frac{\sum_{i=1}^{ag}(\|p_g^n\| - \|p_i^n\|)}{ag}$, kde ag je celkový počet agentů.

6 Shrnutí

V příspěvku byla rozvedena implementace PSO algoritmu do Matlabu. Optimalizace byla popsána formálně i na konkrétních příkladech. Ukázali jsme, jakým způsobem ovlivňují jednotlivé parametry průběh optimalizace a jaké můžeme očekávat výsledky. Prostor byl věnován i popisu vstupních a výstupních polí, jejichž charakter – spolu s přeindexováním hodnot uvnitř funkce – umožňuje využívat *PSOptimizer* pro řešení celé řady problémů. Dále jsme se krátce věnovali možnosti modelovat průběh PSO, čímž by se výrazně zkrátila doba potřebná ke konvergenci do globálního minima. Přesný postup není v současné době dostatečně publikován v referenční literatuře a experimentálních dat není zatím dostatek pro formulaci vlastních závěrů.

Dokončený optimalizátor je v současné době využíván pro rozličné úlohy na katedře elektromagnetického pole, kde autoři působí. Výše uvedené poznatky jsou aplikovány zejm. v oblasti anténní techniky a teorie pole.

Reference

[1] K. E. Parsopoulos, M. N. Vrahatis: *Recent approaches to global optimization problems through Particle Swarm Optimization*, Natural Computing, pp.235–306, 2002.

[2] Jacob Robinson, Yahya Rahmat-Samii: *Particle Swarm Optimization in Electromagnetics*, IEEE Trans. on Antennas and Propagation, Vol. 52, No. 2, pp.397–407, February 2004.

[3] James Kennedy, Russell Eberhart: *Particle Swarm Optimization*, IEEE, pp.1942–1948, 1995.

[4] Ružica M. Golubović, Dragan I. Olćan: *Antenna Optimization Using Particle Swarm Optimization Algorithm*, Journal of Automatic Control, Vol. 16, pp.21–24, 2006.

[5] Sergey N. Makarov: *Antenna and EM Modeling with Matlab*, Wiley-Interscience, New York, 2002, ISBN: 0-471-21876-6.

[6] Miloslav Čapek: *Modální analýza mikropáskových patch antén*, Bakalářská práce, ČVUT-FEL, Praha, 2007.

[7] Pavel Tišnovský: *Interaktivní editor afinních transformací*. Diplomová práce, VUT. Brno, 1999.

[8] Pavel Hazdra: *Fraktálové antény*. Diplomová práce, ČVUT-FEL, Praha, 2003.

Miloslav Čapek
capekm6@fel.cvut.cz

Ing. Pavel Hazdra
hazdrap@fel.cvut.cz