TECHNICKÁ UNIVERITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 3906T001 – Mechatronika

Implementace optimalizačního algoritmu diferenciální evoluce

Implementation of the optimization algorithm differential evolution

Diplomová práce

Autor: **Bc. František Makyta**

Vedoucí práce: Ing. Pavel Herajn

Konzultant: Ing. Radek Srb

V Liberci 21. 5. 2010

TECHNICKÁ UNIVERITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Ústav mechatroniky a technické informatiky

Akademický rok: 2009/ 2010

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. František MAKYTA**

Studijní program: **N2612 Elektrotechnika a informatika**

Studijní obor: **Mechatronika**

Název tématu: **Implementace optimalizačního algoritmu diferenciální evoluce**

ZÁSADY PRO VYPRACOVÁNÍ

1. Seznamte se s principem algoritmu diferenciální evoluce.
2. V prostředí Matlab vytvořte funkce umožňující využití daného algoritmu.
3. V prostředí Matlab vytvořte grafické rozhraní pro jednoduché použití naprogramovaných funkcí pro optimalizaci.
4. Pomocí vhodných testovacích funkcí porovnejte výsledky daného algoritmu se standardními optimalizačními algoritmy.
5. Zhodnoťte dosažené výsledky a vypracujte písemnou zprávu.

Rozsah grafických prací:	**dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 40 – 60 stran**
Forma zpracování diplomové práce:	**tištěná/ elektronická**

Seznam odborné literatury:

[1] Godfrey C. Onwubolu, B. V. Babu: New Optimization Techniques in Engineering (Studies in Fuzziness and Soft Computing). Springer, 1. edition, 2004, ISBN-13: 978-3540201670

[2] Kenneth V. Price, Rainer M. Storn, Jouni A. Lampinen: Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series). Springer, 1. edition, 2005, ISBN-13: 978-3540209508

[3] Uday K. Chakraborty: Advances in Differential Evolution (Studies in Computational Intelligence). Springer, 1. edition, 2008, ISBN-13: 978-3540688273

Vedoucí diplomové práce:	**Ing. Pavel Herajn** Ústav mechatroniky a technické informatiky
Konzultant diplomové práce:	**Ing. Radek Srb** Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce:	**16. října 2009**
Termín odevzdání diplomové práce:	**21. května 2010**

L. S.

prof. Ing. Václav Kopecký, CSc.
děkan

doc. Ing. Petr Tůma, CSc.
Vedoucí ústavu

V Liberci 16. října 2009

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském § 60 – školní dílo.

Beru na vědomí, že technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 21. 5. 2010

Podpis

Abstrakt

Tato práce se zabývá optimalizačním algoritmem diferenciální evoluce a jeho implementací pro řešení technických problémů. Podrobně je analyzována struktura a mechanismus algoritmu v procesu hledání extrému funkce více proměnných. Funkčnost algoritmu je ověřena na testovacích funkcích, dosažené výsledky jsou porovnány s výsledky řešení metody největšího spádu a simplexové metody. Výsledkem této práce je grafické prostředí vytvoření v programu Matlab.

Klíčová slova: optimalizace parametrů funkce, diferenciální evoluce.

Abstract

This work deals with the optimization differential evolution algorithm and its implementation for solving technical problems. In detail is analyzed the structure and mechanism of the algorithm in the process of finding an extreme of functions of several variables. The functionality of the algorithm is verified on test functions, the results are compared with results of the solution method of steepest descent and simplex method. The result of this work is to create a graphical user interface in Matlab.

Keywords: function parameters optimization, differential evolution.

Obsah

Seznam použitých symbolů

bin	binomické křížení
$f(\vec{x})$	funkce více proměnných, ve které hledáme optimální parametry
D	počet parametrů funkce
$\vec{d}_k$	směrový vektor gradientní metody největšího spádu
exp	exponenciální křížení
G	pořadí generace
$\vec{g}$	těžiště simplexu
$g_i(\vec{x})$	funkční omezující podmínky
i	index jedince v populaci
j	index parametru jedince
j_{rand}	náhodně generovaný index parametru jedince
k	index omezujících fiunkcí
jk	jedno-bodové křížení
NG	počet generačních cyklů
NP	počet jedinců populace
P^0	počáteční populace
$\vec{x}$	vektor řešení
x_j^L, x_j^H	dolní a horní hodnota intervalu j-té proměnné
x(1)	příklad zadávání proměnné do programu
$\vec{y}$	bod vytvořený reflexí simplexu
$\vec{u}$	zkušební vektor
$\vec{v}$	šumový vektor
r	náhodně generovaný index jedince populace
z_a, z_b, z_c, z_d	počet ozubení kol
λ_k	krok gradientní metody největšího spádu

1 Úvod

Problematika optimalizace jako taková byla dlouhou dobu řešena klasickým matematickým aparátem, který je založen na infinitezimálním počtu, variačních či numerických metodách. Tento klasický výpočetní aparát dovoluje nalézt optimální řešení pro jednodušší problémy, ale pro složitější problémy většinou nalézá řešení suboptimální. Klasické metody jsou dobře aplikovatelné pro různé typy úloh, ale se vzrůstající obtížností a komplexností problému se obvykle přechází od analytických metod k numerickým, které navíc vyžadují delší čas k řešení, ale i účast zkušenějšího řešitele.

Existuje skupina algoritmů nového typu, které mají několik zvláštností, jež je činí široce použitelnými. Tyto algoritmy mají základ ve filosofii, jsou to tzv. evoluční algoritmy. Jejich výhodou je, že z principu své přirozenosti se zaměřují na hledání globálních extrémů a že při ukončení poskytují hned několik řešení. Jejich nevýhodou je, že pracují částečně s náhodou, a proto výsledek nelze dopředu přesně předvídat, z toho plyne i skutečnost, že se matematické důkazy stanovují velmi obtížně, proto se vychází ze zkušeností s těmito algoritmy, které ukazují jejich životaschopnost a použitelnost.

V této práci se budeme zabývat algoritmem diferenciální evoluce. Porovnáme výsledky algoritmu diferenciální evoluce s jinými optimalizačními algoritmy, jako je simplexová metoda a gradientní metoda největšího spádu. V prostředí Matlabu vytvoříme programové funkce pro využití algoritmu a nakonec i grafické prostředí, které usnadní využití algoritmu.

2 Vymezení pojmů v úloze optimalizace

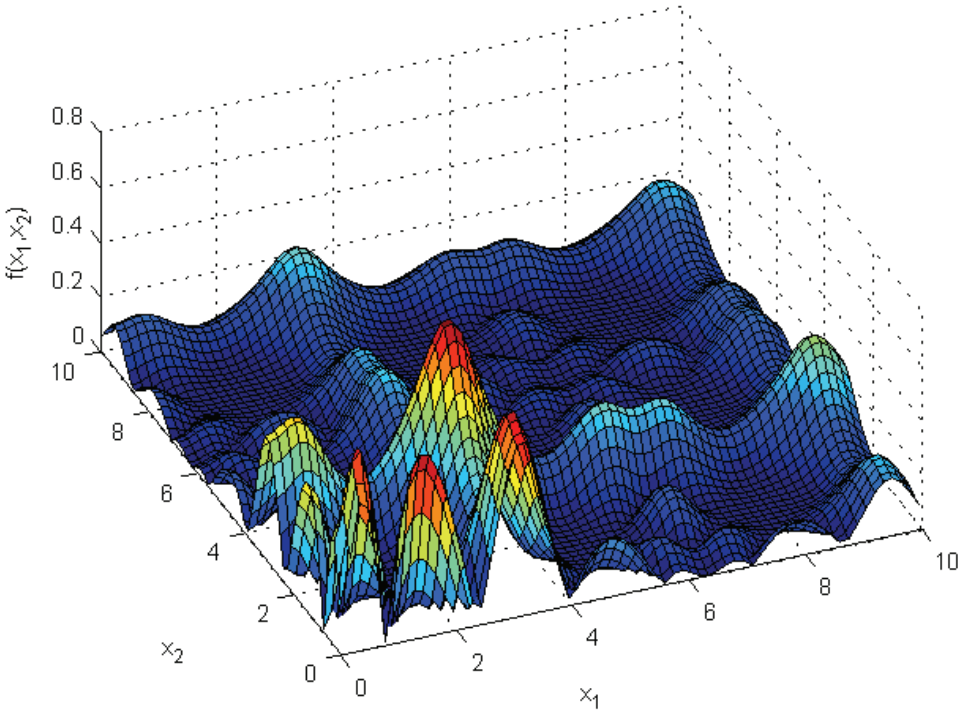
Optimalizační úlohy se hodně používají v oblasti ekonomie, které lze shrnout pod název problémy optimalizace výrobních programů. Některé terminologie se tak používají i v oblasti techniky.

2.1 Účelová funkce

Účelovou funkcí rozumíme funkci, jejíž optimalizace (nalezení minima či maxima) vede k nalezení optimálních hodnot jejich argumentů. Funkce se také nazývá jako účelová nebo cenová funkce, její označení bude dále $f(\vec{x})$, kde vektor $\vec{x} = (x_1, x_2, ..., x_n)$ představuje hledanou sadu parametrů. V tomto textu se zaměříme na hledání minima funkce, pokud bychom chtěli hledat maximum funkce, pak stačí příslušnou funkci vynásobit číslem mínus jedna. Pokud hledáme extrémy funkce bez omezujících podmínek, pak hledáme tzv. volné extrémy, pokud budou definovány omezující podmínky, pak hledáme vázané extrémy.

2.2 Geometrická interpretace účelové funkce

Každá funkce představuje geometrický problém, ve kterém hledáme pozici maxima nebo minima na ploše ležící v n+1 – rozměrném prostoru, tato plocha se také nazývá hyperplocha, nebo prostor možných řešení daného problému. Počet dimenzí n je dán počtem nezávisle proměnných účelové funkce. Např. pokud účelová funkce obsahuje dvě nezávisle proměnné (dva parametry), pak hledáme extrém na dvourozměrné ploše v třírozměrném prostoru, kde třetí dimenze je dána návratovou hodnotou funkce. Uvedený příklad můžeme graficky znázornit na obrázku 5.2.1, kde x_1, x_2 představují parametry funkce $f(x_1, x_2)$, svislá osa určuje její funkční hodnotu.



Obrázek 2.1: Dvourozměrná plocha ve třírozměrném prostoru

2.3 Oblasti použití evolučních algoritmů

V technice je zajímavou oblastí tzv. evoluční hardware [4]. Evolučním způsobem jsou dnes navrhovány nejen programy, ale i elektronické obvody, mechanické systémy, apod., jednoduše vše, co může být reprezentováno pomocí symbolů, tedy potom i v počítači.

Pomocí evolučního návrhu obvodů lze nalézt taková řešení, které expert v daném oboru není vůbec schopen vymyslet. Existují analogové a číslicové obvody nebo antény vytvořené pomocí evolučních technik. Evoluční návrh obvodů je často v reálném fyzickém prostředí s různými typy nepříznivých podmínek, jako je např. elektromagnetické záření, apod. Tímto způsobem může být nalezeno řešení, které je adaptováno pro daný obvod, prostor a čas. Nevýhodou evolučního návrhu je neschopnost navrhovat složité obvody, které lze konvenčními metodami dobře řešit.

Evoluční techniky se používají také pro hledání funkcí, které aproximují data, tzv. metodou symbolické regrese [4] (pomocí tzv. gramatické evoluce).

3 Klasifikace optimalizačních algoritmů

Optimalizační algoritmy hledají minimum účelové funkce numerickou kombinací argumentů. Podle principů jejich činnosti, podle jejich složitosti atp., se dají algoritmy rozdělit do jistých kategorií. Názory na klasifikaci optimalizačních algoritmů se liší, podle toho, z jakých principů vychází, nebo podle problému, pro které jsou určeny, ale obecně je lze rozdělit do těchto kategorií [4]:

- **Enumerativní algoritmy** – algoritmy tohoto typu provedou výpočet všech možných kombinací daného problému. Tento postup je vhodný pro řešení problémů, u kterých jsou argumenty účelové funkce diskrétního charakteru a přitom je jich relativně málo, protože velké množství argumentů prodlužuje dobu výpočtu.

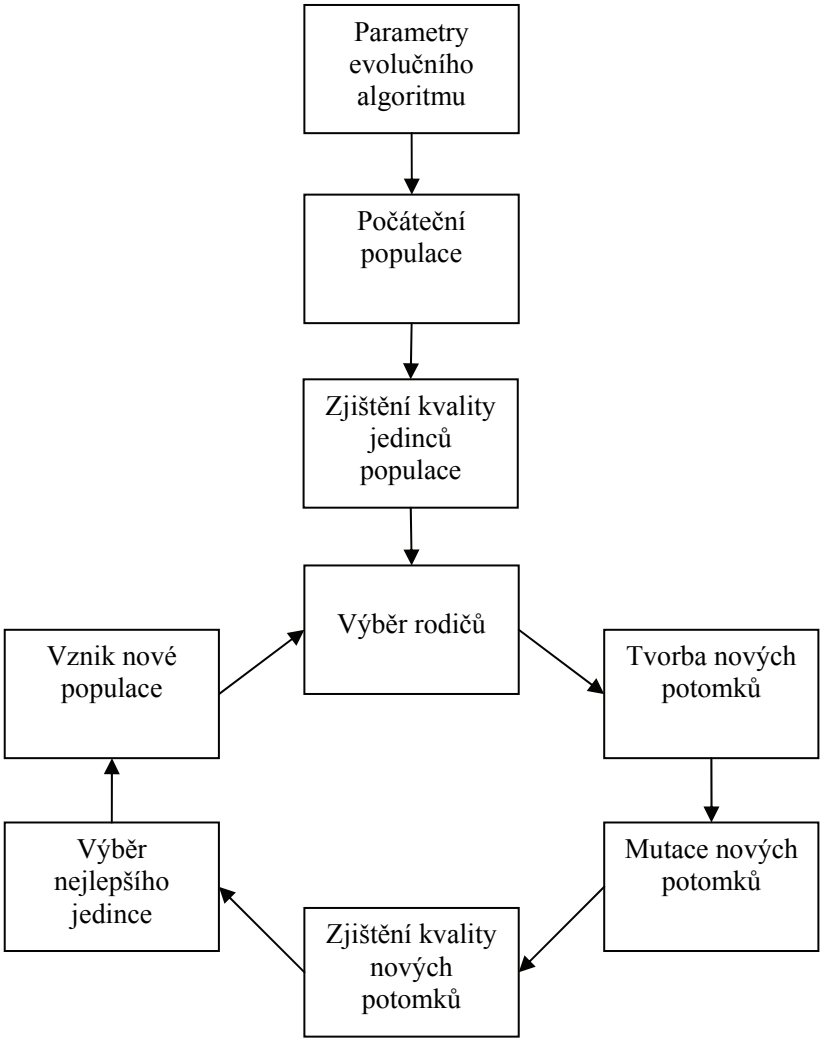
- **Deterministické algoritmy** – algoritmy tohoto typu jsou postaveny na metodách klasické matematiky. Pro svou činnost potřebují omezující předpoklady, aby podávali efektivní výsledky. Tyto předpoklady jsou např.:
  - je znám analytický tvar funkce
  - známe gradient
  - problém je lineární
  - problém je konvexní
  - prostor možných řešení je souvislý, apod.

- **Stochastické algoritmy** – skupina těchto algoritmů je založena na využití náhody. V principu se hledají argumenty účelové funkce čistě náhodně, proto jsou tyto algoritmy obvykle pomalé a vhodné jen pro malé prostory hledaných řešení. Slouží spíše pro hrubý odhad parametrů.

- **Smíšené algoritmy** – tyto algoritmy představují kombinaci deterministických a stochastických metod, které při spolupráci dosahují dobrých výsledků. Často najdou jedno nebo více řešení typu globálního extrému a to nezávisle na počátečních podmínkách. Ke své činnosti nepotřebují znát analytický popis problému. Řešení najdou za relativně malého počtu ohodnocení účelové funkce.

4 Evoluční výpočetní techniky

Evoluční výpočetní techniky jsou numerické algoritmy, které vycházejí z principů Darwinovy a Mendelovy teorie evoluce[1]. Ideou této teorie je předávání rodičovského genomu novým potomkům a pak uvolnění životního prostoru těmto potomkům. Přesněji podle této evoluční teorie se jednotlivé druhy vyvíjejí tak, že rodiče plodí potomky, kteří při vzniku podléhají mutaci. Cyklicky v tzv. generacích vymírají rodiče a nevhodní potomci, a tím uvolňují místo lepším potomkům, kteří se stávají novými rodiči. Obrázek 3.1 schématicky znázorňuje evoluční principy, které jsou zjednodušeně přeneseny do výpočetních metod.



Obrázek 4.1: Schéma evolučního výpočetního postupu [4]

[1] Na principy evolučního vývoje přišli už starověcí myslitelé, jako např. Anaximandros z Milétu, který v díle „O přírodě“ anticipuje moderní vývojovou teorii.

Podle obrázku 4.1 podrobněji rozvedeme principy evolučního algoritmu, který postupuje následujícími body:

1) **Vymezení parametrů evolučního algoritmu**

   Algoritmus musí mít vymezeny parametry, které řídí běh programu, takové jako jsou ukončovací kritéria. Musí být stanovena účelová funkce, která představuje matematický model řešeného problému a v analogii s evolučními principy je ekvivalentem životního prostoru jedinců.

2) **Zjištění kvality jedinců populace**

   Ke každému jedinci populace se stanoví hodnota účelové funkce.

3) **Tvorba počáteční populace**

   Počáteční populace je tvořena NP jedinci s D parametry. Každý jedinec populace představuje náhodně vygenerované jedno řešení účelové funkce, celá populace je tak vyjádřena maticí $NP \times D$.

4) **Výběr rodičů**

   Rodiče jsou vybírány především na základě kvalitativního ohodnocení účelovou funkcí podle bodu 3, případně podle dalších kritérií.

5) **Tvorba nových potomků**

   Potomci jsou tvořeny kombinací parametrů rodičů tzv. procesem křížení.

6) **Mutace nových potomků**

   Nově vytvoření jedinci jsou pozměněni mutací.

7) **Zjištění kvality nových potomků**

   Kvalita každého nového potomka je zjišťována pomocí účelové funkce, stejně jako v bodě 2 a 7.

8) **Výběr nejlepšího jedince**

Z nově vzniklých jedinců, se podle hodnoty účelové funkce, nebo dalších kritérií, vybere nejlepší jedinec.

9) **Vznik nové populace**

Nová populace nastupuje na místo staré, stará populace je vymazána.

5 Teorie složitosti prohledávaného prostoru

Úlohy, které vedou k výpočtu extrémů funkcí více proměnných, se často potýkají s komplikacemi, které mají několik zdrojů [4]:

- Prostor všech řešení je příliš veliký, aby klasické výpočetní metody nalezly řešení za relativně krátký čas.
- Řešený problém je natolik složitý, že je nahrazen matematickým modelem, který aproximuje realitu. Optimální řešení sice souhlasí s modelem, ale s realitou nikoliv.
- Účelová funkce se mění v čase, nebo je zatížena šumem.
- Řešení podléhá takovým omezením, které způsobují řešený problém velmi složitým.

Příklad obtížného hledání extrému je uveden na obrázku 2.1. Tato funkce se používá jako testovací funkce pro různé typy evolučních algoritmů. Při řešení takového problému si musíme uvědomit, že výpočet, který běží v počítači, tedy i optimalizace takové funkce, je realizována číslicově. Kdyby se výpočet prováděl na spojité rovině, pak by bylo nutné vypočítat hodnotu funkce v nekonečně mnoha bodech. Vzhledem k číslicovému zpracování se ono nekonečno redukuje na konečnou množinu hodnot. Pokud budeme uvažovat, že počítač má přesnost na 15 desetinných míst, pak každý parametr funkce nabývá až 10^{16} různých hodnot, v obecném případě pro n parametrů, pak bude funkce obsahovat 10^{16n} hodnot. Např. pro dva parametry to znamená 10^{32} funkčních hodnot. Pro představu o velikosti tohoto čísla si představme, že např. pro jedno ohodnocení funkce je zapotřebí doba 10^{-15} s, pak celé ohodnocení by trvalo 10^{17} s, což je asi 3,17 miliard let, to znamená, že bychom na výpočet čekali asi stejnou dobu, jako existuje náš vesmír.

7 Formulace problému

V úloze optimalizace mohou evoluční algoritmy řešit problémy s reálnými, celočíselnými či diskrétními argumenty. Tyto problémy mohou být formulovány následovně:

nalézt takové

$$\vec{x} = (x_1, x_2, ..., x_n),$$

které minimalizuje funkci

$$f(\vec{x}) : D \rightarrow R \qquad D \subset R^n,\ \vec{x} \in D$$

vzhledem k funkčním omezením

$$g_k(\vec{x}) \leq 0 \qquad k = 1, 2, \ldots, m \tag{7.1}$$

a k omezením argumentů

$$x_j^{(L)} \leq x_j \leq x_j^{(H)} \qquad j = 1, 2, \ldots, D \tag{7.2}$$

V této práci se budeme zabývat případem hledání řešení funkce více proměnných $f(\vec{x})$ v oboru reálných čísel. Funkčním omezením (7.1) rozumíme omezující podmínky, které jsou kladeny na parametry funkce. Znamená to tedy nalézt takové řešení, které minimalizuje danou funkci a přitom jsou splněny omezující podmínky (7.1) a rozsahy argumentů (7.2). Podrobněji jsou omezující podmínky formulovány v kapitole 8.10.

8 Evoluční algoritmus diferenciální evoluce

Evoluční algoritmus diferenciální evoluce vyvinuli Ken Price a Rainer Storm a poprvé ho zveřejnili v roce 1995 [2]. Schéma tohoto algoritmu se podobá genetickým algoritmům, se kterými má několik společných rysů, jako je například tvorba potomků v generacích, křížení apod.

Podle obrázku 4.1 rozpracujeme jednotlivé části, které je nutné zadat pro chod algoritmu, jsou to zejména parametry algoritmu, vymezení populace, mutace, křížení, atd.

8.1 Stanovení parametrů

Stejně jako u ostatních algoritmů, je činnost algoritmu diferenciální evoluce ovlivněna jejími parametry, jejich popis a doporučené hodnoty jsou následující:

- $F \in <0,2>$ je tzv. mutační konstanta, její nastavení ovlivňuje tvorbu nových řešení. Při volbě $F = 1$ dochází k generování dvojic stejných řešení, které může zastavit vývoj optimalizačního procesu.

- $CR \in <0,1>$ je konstanta křížení, která také ovlivňuje tvorbu nových řešení, při nastavení $CR = 1$ opět může dojít k zastavení optimalizace, protože novým kandidátem řešení se stává šumový vektor $\vec{v}_j$, který je blíže popsán v kapitole 8.3. Z tohoto důvodu je doporučeno nastavovat konstantu $CR < 1$.

- D – počet parametrů funkce (počet nezávisle proměnných).

- $NP \in <2D, 100D>$ je počet jedinců populace. Doporučená velikost populace (počet jedinců v populaci) je od dvojnásobku až do deseti násobku počtu parametrů D účelové funkce (minimální $NP = 4$). Větší hodnota populace redukuje riziko předčasné konvergence do suboptimálního řešení.

- $NG > 0$ je počet generací, při kterých populace dospěje do hledaného globálního extrému. Velikost tohoto parametru vždy závisí na řešeném problému.

- Rozsahy pro každý parametr funkce: $x_{i,j}^{(L)} \le x_{i,j} \le x_{i,j}^{(H)}$, kde $j = 1, 2, \ldots, D$, $i = 1, 2, \ldots, NP$, parametry $x_{i,j}^{(L)}$, $x_{i,j}^{(H)}$ vyjadřují dolní a horní mez j-tého parametru jedince (jedinec je vektor s D parametry). Dále nebudeme používat uzávorkování horních indexů.

8.2 Inicializace populace

Diferenciální evoluce pracuje s populacemi, tak jako ostatní evoluční algoritmy. Populace představuje matici $NP \times D$, kde NP je počet jedinců (řádků matice) s D parametry (sloupce matice). Každý jedinec populace je řešením daného problému, přitom kvalita každého jedince je vyjádřena hodnotou účelové funkce, která říká, jak je jedinec vhodný pro další vývoj populace.

Evoluční algoritmus cyklicky nahrazuje starou populaci novou, podle přesně definovaných pravidel. V diferenciální evoluci je počáteční populace vytvořena podle vztahu 8.2.1, který generuje jedince populace podle nastavených mezí každého j-tého parametru x_j účelové funkce:

$$P_{i,j}^0 = x_{i,j}^0 = \text{rand}[0,1] \cdot (x_{i,j}^H - x_{i,j}^L) + x_{i,j}^L \qquad (8.1)$$

$$x_{i,j}^{(L)} \le x_{i,j} \le x_{i,j}^{(H)}$$

$$i = 1, 2, \ldots, NP \qquad j = 1, 2, \ldots, D$$

$x_{i,j}$ představuje j-tý parametr i-tého jedince v počáteční populaci P^0, pokud např. řešíme problém zadaný funkcí osmi proměnných se sto jedinci populace, pak každý jedinec je vektor, který obsahuje 8 parametrů. Vztah 8.1 zajišťuje, že všechny parametry jedince budou generovány náhodně a v povolených mezích hranic parametrů.

8.3 Mutace

Operace mutace probíhá tak, že se zvolí náhodně tři jedinci z populace (vektory), dva se od sebe odečtou a vynásobí mutační konstantou F, vznikne tak váhový diferenční vektor, ke kterému se přičte třetí jedinec, vznikne tzv. šumový vektor $\vec{v}_j$, tento postup je zapsán vztahem 8.2.

$$\vec{v}_j = \vec{x}^G_{r3,j} + F \cdot (\vec{x}^G_{r1,j} - \vec{x}^G_{r2,j}) \tag{8.2}$$

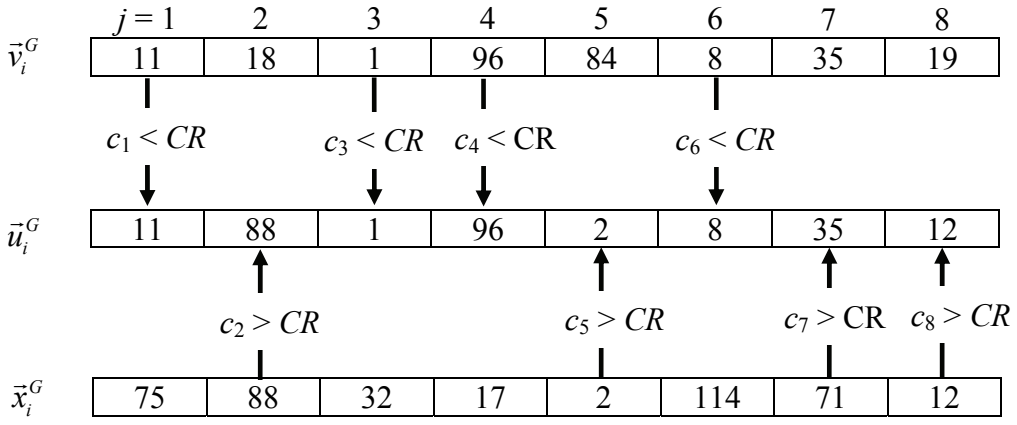
kde $r_1 \neq r_2 \neq r_3$ jsou náhodně vygenerované indexy z intervalu <1, *NP*>, *G* představuje pořadí generace, např. *G* = 17.

8.4 Křížení

Genetické algoritmy vytváří nejprve potomka křížením a poté dochází k jeho mutaci. V diferenciální evoluci dochází k procesu křížení po mutaci, tím způsobem, že se ze zatím nepoužitého čtvrtého jedince vytvoří nový jedinec tzv. zkušební vektor $\vec{u}_j$. Zkušební vektor je vytvořen pomocí konstanty křížení *CR* tak, že se vybírají sobě odpovídající parametry z šumového vektoru $\vec{v}_i^G$ a čtvrtého jedince $\vec{x}_i^G$ (první s prvním, druhý s druhým, …), pro každou takto vybranou dvojici je vygenerováno náhodné číslo $c_i \in <0;1>$ s rovnoměrným rozdělením. Pokud je toto číslo menší než konstanta křížení *CR*, pak se do zkušebního vektoru přesune parametr z šumového vektoru, v opačném případě parametr ze čtvrtého jedince. Tento mechanismus křížení je naznačen vztahem 8.3 a znázorněn na obrázku 8.1.

$$u^G_{i,j} = \begin{cases} x^G_{r3,j} + F \cdot (x^G_{r1,j} - x^G_{r2,j}) & \text{jestliže } \text{rand}_j[0;1] \leq CR \\ x^G_{i,j} & \text{jinak} \end{cases} \tag{8.3}$$

$$r_1, r_2, r_3, i \in <1, NP> \qquad r_1 \neq r_2 \neq r_3 \neq i \qquad j \in <1, D>$$



Obrázek 8.1: Proces křížení pro jedince s osmi parametry

8.5 Pseudokód diferenciální evoluce

Ucelený pohled na funkci algoritmu podává obrázek 8.2, který demonstruje vznik nové generace. Výpočetní algoritmus můžeme zapsat následujícím pseudokódem [1]:

Vymezení parametrů

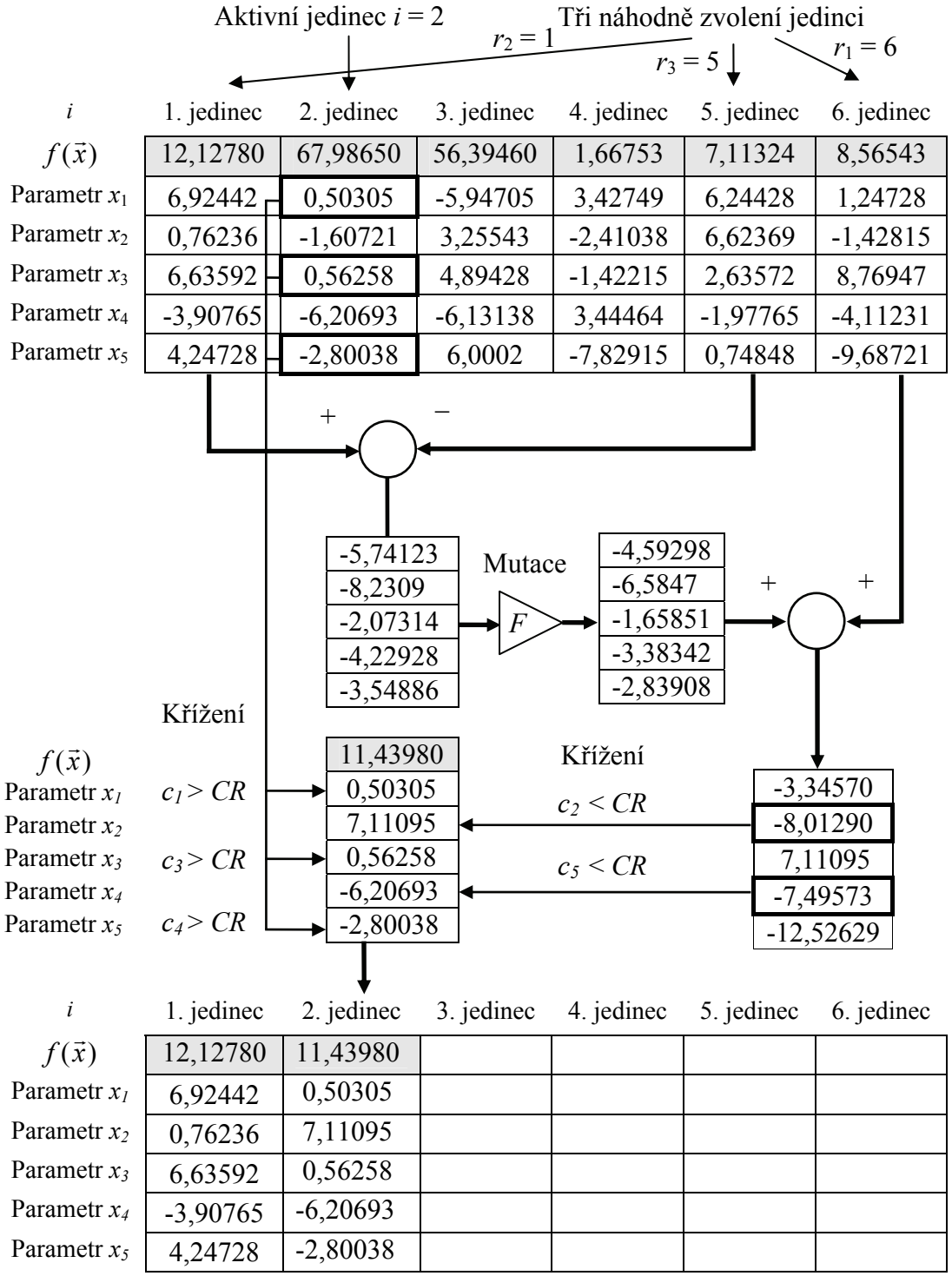
$$\begin{cases} D, NG, NP \geq 4, F \in <0,1+>, CR \in <0;1> \\ i = 1, 2, ..., NP \\ j = 1, 2, ..., D \\ rand[0,1]_j \in <0,1> \\ \text{Stanovení hranic parametrů} \quad x_{i,j}^L, x_{i,j}^H \end{cases}$$

Tvorba počáteční populace

$$\begin{cases} \text{While } i \leq NP \\ \quad \begin{cases} \text{While j} \leq D \\ \quad\quad x_{i,j}^{G=0} = rand[0,1]_j \cdot (x_{i,j}^H - x_{i,j}^L) + x_{i,j}^L \\ \quad\quad j = j + 1 \\ \quad \text{end while} \end{cases} \\ \quad i = i + 1 \\ \text{end while} \end{cases}$$

$$\begin{cases} \text{While } G < NG \\ \quad \begin{cases} \text{While } i \leq NP \\ \quad \text{Mutace a křížení} \\ \quad\quad r_1, r_2, r_3 \in \{1,2,...,D\}, \text{ náhodně vybrané indexy, mimo } r_1 \neq r_2 \neq r_3 \neq i \\ \quad\quad \begin{cases} \text{While } j \leq D \\ \quad\quad\quad u_{i,j}^G = \begin{cases} x_{r_3,j}^G + F \cdot (x_{r_1,j}^G - x_{r_2,j}^G) & \text{jestliže } (rand[0,1]_j < \text{CR}) \\ x_{i,j}^{(G)} & \text{jinak} \end{cases} \\ \quad\quad\quad j = j + 1 \\ \quad\quad \text{end while} \end{cases} \\ \quad\quad \text{Výběr jedince} \\ \quad\quad \vec{x}_i^{G+1} = \begin{cases} \vec{u}_i^G & \text{jestliže } f(\vec{u}_i^G) \leq f(\vec{x}_i^G) \\ \vec{x}_i^G & \text{jinak} \end{cases} \\ \quad\quad i = i + 1 \\ \quad \text{end while} \end{cases} \\ \quad G = G + 1 \\ \text{end while} \end{cases}$$

Parametry diferenciální evoluce			
Dimenze *D*	Velikost populace *NP*	Mutační *F*	Konstanta křížení *CR*
5	6	0.8	0,7



Obrázek 8.2: Princip diferenciální evoluce – vznik jedné populace [4]

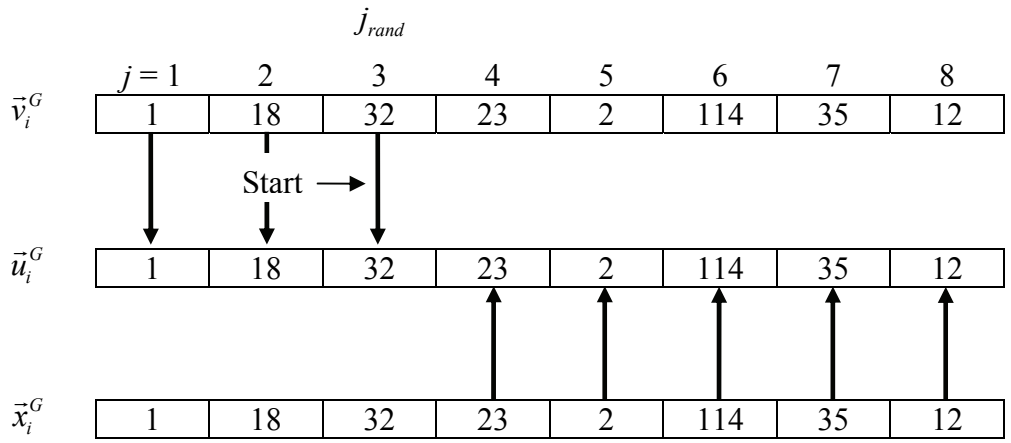
8.6 Varianty diferenciální evoluce

V předešlém textu byla uvedena tzv. klasická diferenciální evoluce, která v procesu křížení používá k naplnění zkušebního vektoru $\vec{u}_i$ postup, který je uveden v kapitole 8.4 vztahem 8.3. V takovém případě má každý parametr z šumového vektoru $\vec{v}_i$ a zdrojového vektoru $\vec{x}_i$ stejnou pravděpodobnost, že se dostane do zkušebního vektoru. Tyto parametry jsou zděděny z šumového vektoru a ze zdrojového vektoru. Existují další varianty křížení, které umisťují parametry ze zdrojového a šumového vektoru, podle pravděpodobnostním rozdělení. Některé postupy křížení jsou následující [3]:

- jednobodové křížení,
- exponenciální křížení,
- binomické křížení aj.

8.6.1 Jedno-bodové křížení

Jednou metodou, jak přiřadit parametry do zkušebního vektoru $\vec{u}_i$, je použití jedno-bodového křížení. V této metodě křížení se náhodně vybere index j_{rand} z rozsahu parametrů D. Do zkušebního vektoru se nejprve zkopíruje parametr ležící na tomto indexu v šumovém vektoru $\vec{v}_i$ spolu s ostatními parametry nalevo. Pravá strana zkušebního vektoru je naplněna zbývajícími parametry ze zdrojového vektoru $\vec{x}_i$, tento postup ilustruje obrázek 8.3.



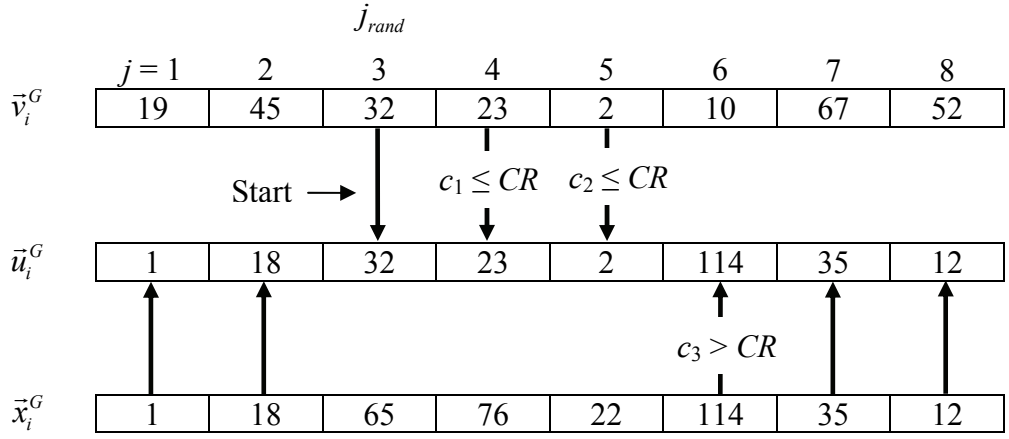
Obrázek 8.3: Příklad jedno-bodového křížení

Stejným způsobem lze vytvořit další zkušební vektor přehozením šumového vektoru za zdrojový. Podobně jako jedno-bodové křížení lze vytvořit i dvou-bodové a vícebodové křížení.

8.6.2 Exponenciální křížení

Tento způsob plnění zkušebního vektoru začíná náhodným generováním indexu j_{rand}, z této pozice se provede kopie parametru z šumového vektoru $\vec{v}_i$ do zkušebního vektoru $\vec{u}_i$. Kopírování parametrů pokračuje od tohoto indexu zvýšeného o jedničku a to tak dlouho, dokud platí, že náhodně vygenerované číslo z rovnoměrného rozdělení c_j = rand[0, 1] < *CR*, jinak se zkušební vektor naplní parametry ze zdrojového vektoru $\vec{x}_i$.

Na obrázku 8.4 je naznačen příklad exponenciálního křížení pro osm parametrů (*D* = 8). Nejprve je náhodně generován index např. j_{rand} = 3, ze šumového vektoru se zkopíruje parametr na tomto indexu do zkušebního vektoru, kopírování pokračuje, dokud jsou náhodně generovaná čísla $c_j \leq CR$. V příkladě předpokládáme, že to jsou čísla $c_1 \leq CR$, $c_2 \leq CR$ pro j = 4, 5, proto jsou parametry na těchto indexech kopírovány do zkušebního vektoru. Od indexu j = 6 platí, že číslo $c_3 > CR$, proto jsou ostatní hodnoty zkušebního vektoru naplněny z vektoru zdrojového, zde se jedná o parametry na indexech j = 1, 2, 6, 7, 8.



Obrázek 8.4: Příklad exponenciálního křížení

Možná realizace exponenciálního křížení je naznačena pseudokódem:

Exponenciální křížení

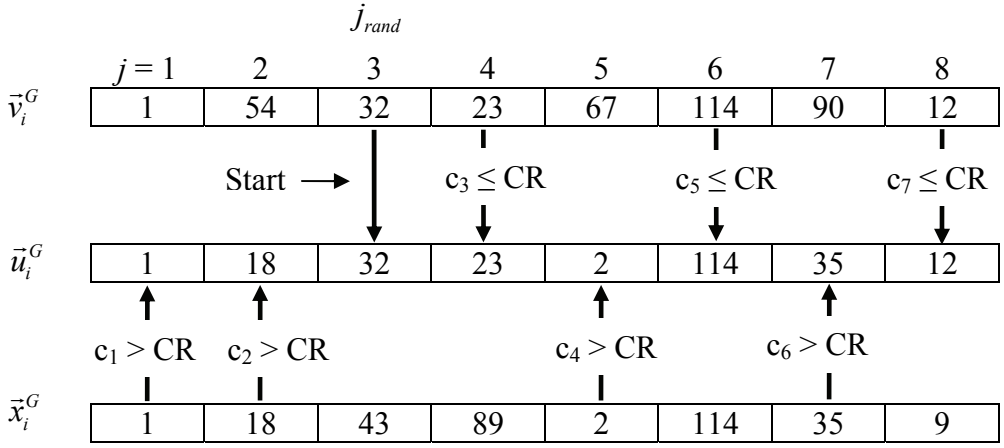
$$
\begin{cases}
\begin{cases} j_{rand} \in <1, D> \\ j = j_{rand} \\ u_{i,j} = v_{i,j} \\ j = j \text{ modulo } D + 1 \end{cases} & \text{zbytek po celočíselném dělení + 1} \\
\begin{cases} \text{While } (\text{rand}[0,1] < CR) \,\&\, (j \neq j_{rand}) \\ \quad u_{i,j} = v_{i,j} \\ \quad j = j \text{ modulo } D + 1 \\ \text{end while} \end{cases} & \\
\begin{cases} \text{While } (j \neq j_{rand}) \\ \quad u_{i,j} = x_{i,j} \\ \quad j = j \text{ modulo } D + 1 \\ \text{end while} \end{cases} & \text{zkopírovat zbývající parametry ze zdrojového vektoru do zkušebního}
\end{cases}
$$

8.6.3 Binomické křížení

Binomické křížení začíná náhodným generováním indexu j_{rand}, parametr šumového vektoru $\vec{v}_i$ na této pozici se zkopíruje na stejnou pozici do zkušebního vektoru $\vec{u}_i$. Podle náhodně generovaného čísla c_j = rand[0, 1] z rovnoměrného rozdělení se naplní zbývající prvky zkušebního vektoru. Jestliže číslo $c_j \leq CR$, pak se parametr z šumového vektoru přesune do zkušebního vektoru, v ostatních případech pro $c_j > CR$ se přesunou parametry ze zdrojového vektoru $\vec{x}_i$. Binomické křížení je naznačeno na obrázku 8.5 pro jedince s osmi parametry (D = 8). Způsob naplnění zkušebního vektoru můžeme zapsat následujícím pseudo-kódem:

Binomické křížení

$$
\begin{cases}
\text{While } j \leq D \\
\quad u_{i,j}^{G} = \begin{cases} v_{i,j}^{G} & \text{jestliže } (\text{rand}[0,1]_j < \text{CR}) \vee j = j_{rand} \\ x_{i,j}^{G} & \text{jinak} \end{cases} \\
\quad j = j + 1 \\
\text{end while}
\end{cases}
$$



Obrázek 8.5: Příklad binomického křížení

8.6.4 Výpočet šumového vektoru

Doposud jsme k sestavení šumového vektoru $\vec{v}$ použili tři náhodně zvolené jedince s populace s indexy r_1, r_2, r_3 podle vztahu 8.2 a to zvlášť pro každého i-tého jedince v populaci, přitom platilo, že indexy odkazující na jedince jsou $r_1 \neq r_2 \neq r_3 \neq i$. Experimentálně bylo ověřeno, že na pořadí indexů r_1, r_2, r_3 nezáleží, protože jejich výběr je náhodný.

Šumový vektor lze vypočítat z několika dalších jedinců, lze také použít např. nejlepšího jednice $\vec{x}_{best,j}^G$ v populaci. Obecný zápis variant diferenciální evoluce (DE) je ve tvaru DE/*x*/*y*/*z*, kde *x* představuje zdrojový vektor, *y* počet diferenčních vektorů a *z* metodu křížení. Uvedeme zde některé možné varianty výpočtu šumového vektoru bez dosazení za písmeno *z*, které je v klasické diferenciální evoluci vynecháno. Pokud je použito binomické, exponenciální nebo jednobodové křížení, pak je za *z* dosazena zkratka bin, exp resp. jk. Uvedeme pět variant, ze kterých můžeme vytvořit až variant dvacet tak, že za písmeno *z* dosadíme zkratku bin, exp, jk nebo ho jednoduše vynecháme, např. DE/rand/1/bin.

Varianta 1: DE/rand/1/*z*

$$\vec{v} = \vec{x}_{r1,j}^G + F \cdot (\vec{x}_{r2,j}^G - \vec{x}_{r3,j}^G) \tag{8.4}$$

Varianta 2: DE/best/1/*z*

$$\vec{v} = \vec{x}_{best,j}^G + F \cdot (\vec{x}_{r2,j}^G - \vec{x}_{r3,j}^G) \tag{8.5}$$

Varianta 3: DE/rand/2/*z*

$$\vec{v} = \vec{x}^{G}_{r5,j} + F \cdot (\vec{x}^{G}_{r1,j} + \vec{x}^{G}_{r2,j} - \vec{x}^{G}_{r3,j} - \vec{x}^{G}_{r4,j}) \tag{8.6}$$

Varianta 4: DE/best/2/*z*

$$\vec{v} = \vec{x}^{G}_{best,j} + F \cdot (\vec{x}^{G}_{r1,j} + \vec{x}^{G}_{r2,j} - \vec{x}^{G}_{r3,j} - \vec{x}^{G}_{r4,j}) \tag{8.7}$$

Varianta 5: DE/local-to-best/1/*z*

$$\vec{v} = \vec{x}^{G}_{i,j} + F \cdot (\vec{x}^{G}_{best,j} - \vec{x}^{G}_{i,j}) + F \cdot (\vec{x}^{G}_{r1,j} - \vec{x}^{G}_{r2,j}) \tag{8.8}$$

8.7 Ukončovací kritéria

V každé generaci se vytvoří stanovený počet jedinců podle parametru *NP*. Celkový počet generací je dán parametrem *NG*, tzn., že pokud proběhne stanovený počet generací, tak se algoritmus ukončí a populace by přitom měla dospět do globálního extrému. Je zřejmé, že nastavením malého počtu generací může evoluce skončit dříve, než nalezne globální extrém. Naopak nastavením velkého počtu generací se globální extrém nalezne, ale algoritmus se ukončí až po provedení stanoveného počtu generací, ale tím se prodlužuje celková doba výpočtu. Z těchto důvodů je vhodné zavést další ukončovací kritéria, která umožní podat výsledky po kratší době. V nejhorším případě nebudou tyto kritéria splněny a algoritmus se jednoduše ukončí vyčerpáním stanoveného počtu generací. Podle [3] lze ukončovací kritéria rozdělit do třech tříd, z toho je první třída zaměřena na hodnoty účelové funkce, druhá na vzdálenosti přesunu populace a třetí na vyhodnocení střední hodnoty.

1) **Ukončovací kritéria odvozená od odchylek funkční hodnoty**

   Tento způsob je založen na vyhodnocení odchylek hodnot funkce za jistý čas, který je dán počtem několika generací. Tyto odchylky mohou být určovány třemi způsoby:

   a) Uchovat odchylku nejlepší hodnoty funkce po dobu několika generací. Pokud po stanoveném počtu generací poklesne odchylka funkce pod uživatelem stanovenou úroveň, pak optimalizace bude ukončena.

b) Stanovit střední hodnotu z odchylek funkčních hodnot všech jedinců z populace z několika generací. Ukončit optimalizaci v případě, že tato střední hodnota poklesne pod uživatelem stanovenou úroveň po několika generacích.

c) Vyhodnocovat počet postoupení zkušebního vektoru do další populace. Pokud tento počet klesne pod stanovenou mez za několik generací, pak ukončit optimalizaci.

2) Ukončovací kritéria odvozená od polohy

Na začátku je populace vygenerována náhodně v prohledávaném prostoru a v průběhu optimalizace populace konverguje do jednoho bodu. Přemístění jednotlivých parametrů jedince mohou být použita k odvození následujících dvou ukončujících kritérií:

a) Jestliže střední hodnota přesunu parametrů jednotlivých členů populace v prohledávaném prostoru parametrů poklesne za několik generací pod uživatelem stanovenou úroveň, pak se optimalizace ukončí.

b) Posuzovat lze také střední hodnotu posunu funkčních hodnot od všech jedinců. Jestliže tato hodnota poklesne za několik generací pod uživatelem stanovenou úroveň, pak se optimalizace ukončí.

3) Ukončovací kritéria odvozená z rozdělení funkčních hodnot

K odvození ukončovacích kritérií lze použít i rozdělení hodnot funkce každého jednice od pozice nejlepší hodnoty. Možné varianty jsou např.:

a) Jestliže je vzdálenost hodnoty funkce jakéhokoliv členu populace od nejlepší hodnoty menší než nastavená mez, pak se proces optimalizace ukončí.

b) Jako ukončovací kritérium je možné použít směrodatnou odchylku vypočtenou z hodnot funkce všech jedinců. Pokud tato hodnota klesne pod stanovenou mez, pak se proces optimalizace ukončí.

8.8 Stagnace

Stagnace je jev, při kterém se zastaví vývoj hodnoty účelové funkce směrem k nižším hodnotám ještě před dosažení globálního extrému. K tomuto jevu dochází bez zjevných příčin, protože populace zůstává stále diverzibilní, ale optimalizační proces dále nepostupuje. Obecně je známo, že evoluční algoritmy dochází k předčasné konvergenci, jestliže:

- Populace dospěla do lokálního extrému.
- Populace není diverzibilní (různorodá).
- Proces optimalizace probíhá pomalu nebo vůbec.

Ovšem za určitých podmínek přesto dochází ke stagnaci, i když populace nedospěla do lokálního extrému, je diverzibilní a optimalizační proces stále probíhá. Příkladem může být probíhající optimalizace v rovině.

8.9 Omezující podmínky

Důvody omezení plynou z reálného světa, kde některá řešení nemusí být fyzikálně realizovatelná nebo mohou být nevýhodná (např. záporná tloušťka stěny nádoby atp.).

8.9.1 Omezení kladená na argumenty účelové funkce

V průběhu optimalizačního procesu se hodnoty jednotlivých argumentů účelové funkce mohou dostat mimo povolené hranice, tak jak je naznačeno na obrázku 8.6. Návrat jedince do povolených hranic můžeme zajistit např. těmito postupy:

Obrázek 8.6: Překročení povolených hranic jednotlivých parametrů jedince

1) Vygenerovat nového jedince

– pokud *j*-tý parametr *i*-tého jedince překročí horní hranici $x_{i,j}^{H}$ nebo dolní hranici $x_{i,j}^{L}$, pak se vygeneruje nová hodnota parametru jedince podle vztahu:

$$x_{i,j}^{G+1} = \begin{cases} rand[0,1] \cdot (x_{i,j}^{H} - x_{i,j}^{L}) + x_{i,j}^{L} & \text{jestli} \quad x_{i,j}^{G+1} < x_{i,j}^{L} \vee x_{i,j}^{G+1} > x_{i,j}^{H} \\ x_{i,j}^{G+1} \end{cases} \tag{8.9}$$

kde

i = 1, 2, ..., D

j = 1, 2, ..., NP

G = 1, 2, ..., G_{max}

2) Zastavení jedince na hranici

– pokud parametr jedince překročí dolní hranici, pak se hodnota parametru nastaví na dolní hranici $x_{i,j}^{L}$, u překročení horní hranice se provede nastavení na horní hranici $x_{i,j}^{H}$, tak jak ukazuje vztah:

$$x_{i,j}^{G+1} = \begin{cases} \begin{cases} x_{i,j}^{L} & \text{jestli} \quad x_{i,j}^{G+1} < x_{i,j}^{L} \\ \vee \\ x_{i,j}^{H} & \text{jestli} \quad x_{i,j}^{G+1} > x_{i,j}^{H} \end{cases} \\ x_{i,j}^{G+1} \quad \text{jinak} \end{cases} \tag{8.10}$$

3) Zrcadlení jedince okolo hranice

– pokud parametr jedince překročí dolní hranici, pak se parametr zrcadlí okolo dolní hranice $x_{i,j}^{L}$ do povolené oblasti, u překročení horní hranice se zrcadlí okolo hranice $x_{i,j}^{H}$, tak jak ukazuje vztah:

$$x_{i,j}^{G+1} = \begin{cases} \begin{cases} 2x_{i,j}^{L} - x_{i,j}^{G+1} & \text{jestli} \quad x_{i,j}^{G+1} < x_{i,j}^{L} \\ \vee \\ 2x_{i,j}^{H} - x_{i,j}^{G+1} & \text{jestli} \quad x_{i,j}^{G+1} > x_{i,j}^{H} \end{cases} \\ x_{i,j}^{G+1} \quad \text{jinak} \end{cases} \tag{8.11}$$

8.9.2 Penalizace funkcí

Penalizace funkcí je postup, kterým se záměrně upraví hodnota účelové funkce ve vybraných oblastech argumentů, které z nejrůznějších důvodů nevyhovují. V případě, že jsou tyto oblasti jednoznačně zakázány, pak se tato penalizace nazývá jako *hard-constraints*, zde se ve skutečnosti jedná o omezení hodnot argumentů účelové funkce. V takovém to případě se hodnota účelové funkce nemění, ale jedinec, který by se v této oblasti nacházel, by byl zrušen a nahrazen jedincem ležícím v povolené oblasti. Další možností jsou tzv. *soft-constraints*. Tato omezení nezruší jedince ležícího v zakázané oblasti, ale představuje jeho znevýhodnění změnou hodnoty účelové funkce.

Na prostor možných řešení můžeme také nahlížet jako na životní prostor jedinců populace, penalizace má efekt znepříjemnění pobytu jedinců v penalizovaných oblastech. Geometricky lze penalizovanou oblast interpretovat jako lokální deformaci hyperplochy opačným směrem k extrému, který není hledán. Pokud se v důsledku evolučního procesu dostane jedinec do penalizované oblasti, pak ji rychle opustí, nebo nepostoupí do další populace. Jeden z několika přístupů penalizace uvádí vztah (8.12):

$$f(\vec{x}) = [f(\vec{x}) + a] \cdot \prod_{i=1}^{n} c_i^{b_i} \tag{8.12}$$

kde

$$c_i = \begin{cases} 1 + s_i \cdot g_i(\vec{x}), \text{ jestli } \; g_i(\vec{x}) > 0 \\ 1 \text{ jinak} \end{cases}$$

$$s_i \geq 1$$

$$b_i \geq 1$$

$$\min[f(\vec{x})] + a > 0$$

kde konstanta a zajišťuje, že účelová funkce nebude nabývat jen nezáporných hodnot, a je volena velmi vysoká. Konstanty s_i jsou použity k transformaci funkce do vhodného měřítka a exponent b_i tvaruje prohledávanou plochu, optimalizace pracuje uspokojivě i pro nastavení $s = 1$, $b = 1$.

Mnohonásobné *hard-constrains* rozdělí prohledávaný prostor do izolovaných ostrůvků, které mohou vytvářet lokální extrémy na jejich hranicích, z těchto důvodů je penalizační metoda *soft-constrains* považována za výhodnější.

8.9.3 Přímá aplikace omezujících podmínek

Penalizace funkcí vyžaduje správné nastavení konstant *a, b_i, s_i,* ovšem jejich odhad nemusí být správný, dokonce nalezené řešení nemusí odpovídat skutečnému minimu účelové funkce, protože se procesem penalizace upravuje prostor funkčních hodnot. Z těchto důvodů byly navrženy metody přímého zpracování omezujících podmínek, které nepožadují žádné nastavení konstant.

Uvažujeme, omezují podmínky ve tvaru (8.11), podobně jako v kapitole 7:

$$g_k(\vec{x}) \leq 0 \qquad k = 1, 2, \ldots, m \tag{8.13}$$

Každou omezující podmínku ve tvaru $g_k(\vec{x}) = b$ můžeme převést na tvar $g_k(\vec{x}) - b \leq 0$, kde b je konstanta. Cílem je najít takové řešení $\vec{x} = (x_1, x_2, ..., x_n)$, které splňuje podmínky (8.13) a minimalizuje zadanou funkci $f(\vec{x})$. V případě, že je zadána pouze jedna omezující podmínka, pak je hledání řešení $\vec{x}$ relativně jednoduché, protože kontrolujeme pouze splnění jedné podmínky a přitom sledujeme, zda hodnota funkce konverguje k nižším hodnotám.

Podle [1] lze volit jedince, který postoupí do nové generace na základě vyhodnocení šumového vektoru $\vec{u}_i^G$:

- $\vec{u}_i^G$ vyhovuje všem omezením a má menší nebo rovnu hodnotu funkce než $\vec{x}_i^G$, nebo
- $\vec{u}_i^G$ je vyhovující a $\vec{x}_i^G$ není, nebo
- $\vec{u}_i^G$ a $\vec{x}_i^G$ jsou oba vyhovující, ale $\vec{u}_i^G$ lépe splňuje omezující podmínku než $\vec{x}_i^G$

Volba jedince pro další generaci je zapsána pseudokódem [2]:

$$\vec{x}_i^{G+1} = \begin{cases} \vec{u}_i^G \quad \text{jestli} \begin{cases} \begin{cases} \forall k \in \{1,...,m\} : g_k(\vec{u}_i^G) \le 0 \ \wedge \ g_k(\vec{x}_i^G) \le 0 \\ \wedge \\ f(\vec{u}_i^G) \le f(\vec{x}_i^G) \end{cases} \\ \vee \\ \begin{cases} \forall k \in \{1,...,m\} : g_k(\vec{u}_i^G) \le 0 \\ \wedge \\ \exists k \in \{1,...,m\} : g_k(\vec{x}_i^G) > 0 \end{cases} \\ \vee \\ \begin{cases} \exists k \in \{1,...,m\} : g_k(\vec{u}_i^G) > 0 \\ \wedge \\ \forall k \in \{1,...,m\} : \max[g_k(\vec{u}_i^G), 0] \le \max[g_k(\vec{x}_i^G), 0] \end{cases} \end{cases} \\ \vec{x}_i^G \quad \text{jinde} \end{cases}$$

Pokud chceme zadat podmínku ve tvaru $g_k(\vec{x}) = 0$, tak to budeme řešet pomocí dvou nerovností, lépe však s malou odchylkou, protože kvůli numerickému výpočtu nemusí nikdy dojít k nalezení dané rovnosti:

$g_k(\vec{x}) = 0$ přepíšeme na $g_k(\vec{x}) - a \le 0$ a $-g_k(\vec{x}) + b \le 0$, přitom $a > b$, přesnost hledané rovnosti je $e = a - b$.

9 Vybrané numerické metody optimalizace

Pro hledání extrémů funkce více proměnných lze použít např. simplexovou metodu nebo některou z gradientních metod, my se zde jako zástupce gradientních metod zvolíme metodu největšího spádu.

9.1 Simplexová metoda

Podle [7] je simplexová metoda příklad algoritmu, který hledá minimum funkce více proměnných ve spojité oblasti a přitom nevyužívá derivace funkce. Základní myšlenka této metody je následující:

Pro účelovou funkci

$$f(\vec{x}) : D \to R \qquad D \subset R^n$$

hledáme globální minimum v souvislé oblasti $D = \prod_{i=1}^{n} \langle a_i, b_i \rangle$, přitom účelovou funkci $f(\vec{x})$ umíme vyčíslit s požadovanou přesností v každém bodě $\vec{x} \in D$, tato funkce nemusí být ani diferencovatelná.

Základním pojmem algoritmu je tzv. simplex *S*, který představuje množinu *n* +1 nekomplanárních bodů v *D*.

$$S = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_{n+1}\}$$

Jinými slovy v *n* dimenzionálním prostoru vytvoříme *n* +1 bodů, které tvoří simplex.
V simplexu nalezneme body s nejvyšší a nejnižší funkční hodnotou:

$$\vec{x}^H = \arg\max_{x \in S} f(\vec{x})$$

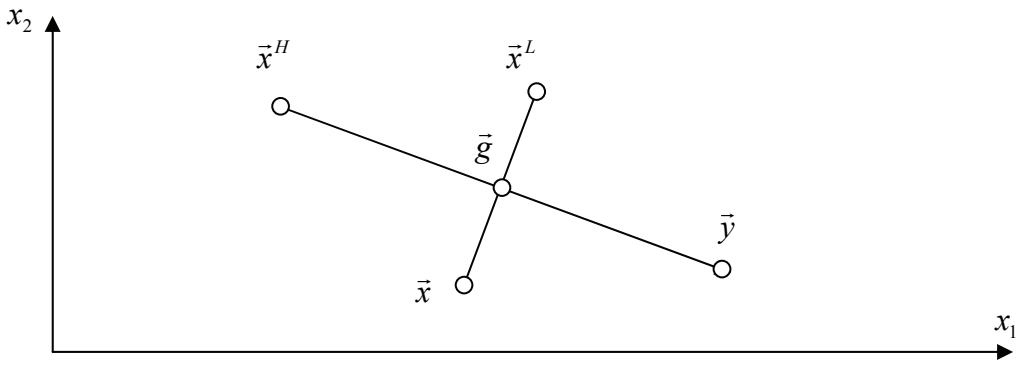
$$\vec{x}^L = \arg\max_{x \in S} f(\vec{x})$$

Spočítáme těžiště, tzn. průměr *n* bodů, které zůstanou v simplexu po odstranění bodu s nejvyšší funkční hodnotou $\vec{x}^H$, podle vztahu 9.1:

$$\vec{g} = \frac{1}{n}\left(\sum_{i=1}^{n} \vec{x} - \vec{x}^H\right) \tag{9.1}$$

Nový bod $\vec{y}$ hledáme pomocí simplexu, který nahradí doposud nejhorší bod $\vec{x}^H$, v případě, že $f(\vec{y}) < f(\vec{x}^H)$. Bod $\vec{y}$ získáme tzv. reflexí, tzn. překlopení bodu $\vec{x}^H$ přes těžiště $\vec{g}$ podle vztahu 9.2, graficky je metoda reflexe zobrazena na obrázku 9.1.

$$\vec{y} = \vec{g} + (\vec{g} - \vec{x}^H) = 2\vec{g} - \vec{x}^H \tag{9.2}$$

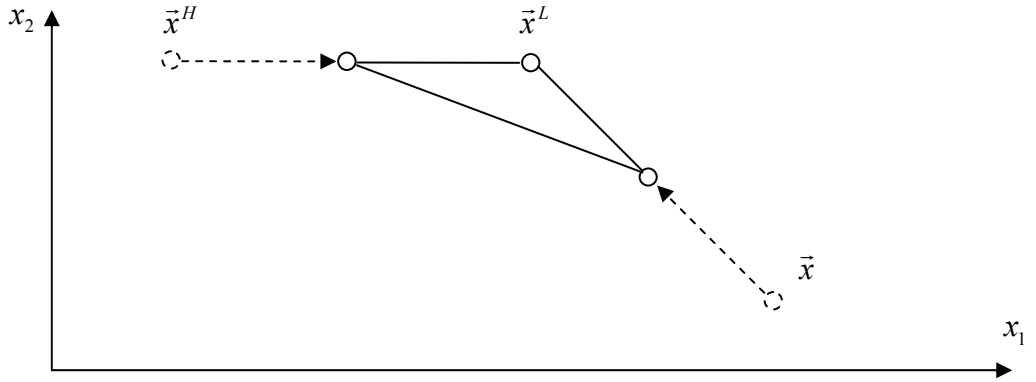


Obrázek 9.1: Konstrukce bodu $\vec{y}$ pomocí reflexe ve 2D

Jestliže nově získaný bod $\vec{y}$ nesplňuje podmínku $f(\vec{y}) < f(\vec{x}^H)$, tak se použije redukce simplexu, tj. jeho zmenšení k bodu $\vec{x}^L$, tak že jeho vzdálenosti budou poloviční. Formálně tuto operaci lze zapsat vztahem 9.3:

$$\vec{x} \leftarrow \frac{1}{2}(\vec{x} + \vec{x}^L) \qquad \vec{x} \in S,\ \vec{x} \neq S \tag{9.3}$$

Redukce simplexu je uvedena na obrázku 9.2, ve kterém se body původního simplexu posunou k bodu $\vec{x}^L$.



Obrázek 9.2: Redukce simplexu

Nejjednodušší varianta simplexové metody může být zapsána pseudokódem:

1. Generovat simplex *S*, tzn. *n* +1 bodů náhodně v *D*

2. Proces optimalizace

While $i \le i_{\max}$

$\vec{x}^H = \arg\max_{x \in S} f(\vec{x})$

$\vec{x}^L = \arg\min_{x \in S} f(\vec{x})$

reflexe(S):

$\vec{g} = \frac{1}{n}(\sum_{i=1}^{n} \vec{x} - \vec{x}^H)$

$\vec{y} = \vec{g} + (\vec{g} - \vec{x}^H) = 2\vec{g} - \vec{x}^H$

jestli $f(\vec{y}) < f(\vec{x}^H)$ tak $\vec{x}^H = \vec{y}$

$\vec{x} = \frac{1}{2}(\vec{x} + \vec{x}^L),\ \vec{x} \neq \vec{x}^L$ jinak

$i = i + 1$

end while

9.2 Metoda největšího spádu

Stejně jako v předchozím případě hledáme minimum funkce více proměnných $f(\vec{x})$. Na rozdíl od simplexové metody tato metoda používá derivaci účelové funkce.

Z počátečního bodu $\vec{x}_k$ se dostaneme blíž k minimu volbou spádového vektoru $\vec{d}_k$, ve kterém funkce $f(\vec{x})$ klesá, pak vybereme bod

$$\vec{x}_{k+1} = \vec{x}_k + \lambda_k \vec{d}_k \tag{9.4}$$

ve kterém $f(\vec{x}_{k+1}) < f(\vec{x}_k)$. Parametr λ_k se nazývá délka kroku, který můžeme zvolit, nebo ho získat minimalizací funkce $\varphi(\lambda) = f(\vec{x}_k + \lambda \vec{d}_k)$ pro $\lambda > 0$:

$$\lambda_k \approx \min_{\lambda > 0} \varphi(\lambda) \tag{9.5}$$

Je známo, že funkce $f(\vec{x})$ klesá ve směru záporného gradientu, proto volíme jako spádový vektor

$$\vec{d}_k = -\nabla f(\vec{x}) \tag{9.6}$$

Existují i další modifikace této gradientní metody, které se zaměřují na výpočet délky ktoku λ_k v každé iteraci.

Gradientní metody[2] lze použít i v případech, kdy nejsou známy derivace. Používají se numerické odhady derivací, které mohou být symetrické a nebo nesymetrické, podle vztahů:

Symetrický odhad

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, x_2, ..., x_i + h, x_{i+1}, ..., x_n) - f(x_1, x_2, ..., x_i - h, x_{i+1}, ..., x_n)}{2h} \tag{9.7}$$

Levý nesymetrický odhad

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, x_2, ..., x_i + h, x_{i+1}, ..., x_n) - f(x_1, x_2, ..., x_n)}{h} \tag{9.8}$$

Pravý nesymetrický odhad

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, x_2, ..., x_n) - f(x_1, x_2, ..., x_i + h, x_{i+1}, ..., x_n)}{h} \tag{9.9}$$

Vyšší derivace se počítají z předchozích derivací.

Poznámka: symetrickou diferenci použijeme k odhadu derivací.

[2] Pro hledání extrémů funkcí více proměnných existují i další gradientní metody, jako je metoda konjugovaných gradientů, metoda paralelních tečen atp.

10 Testování algoritmu diferenciální evoluce

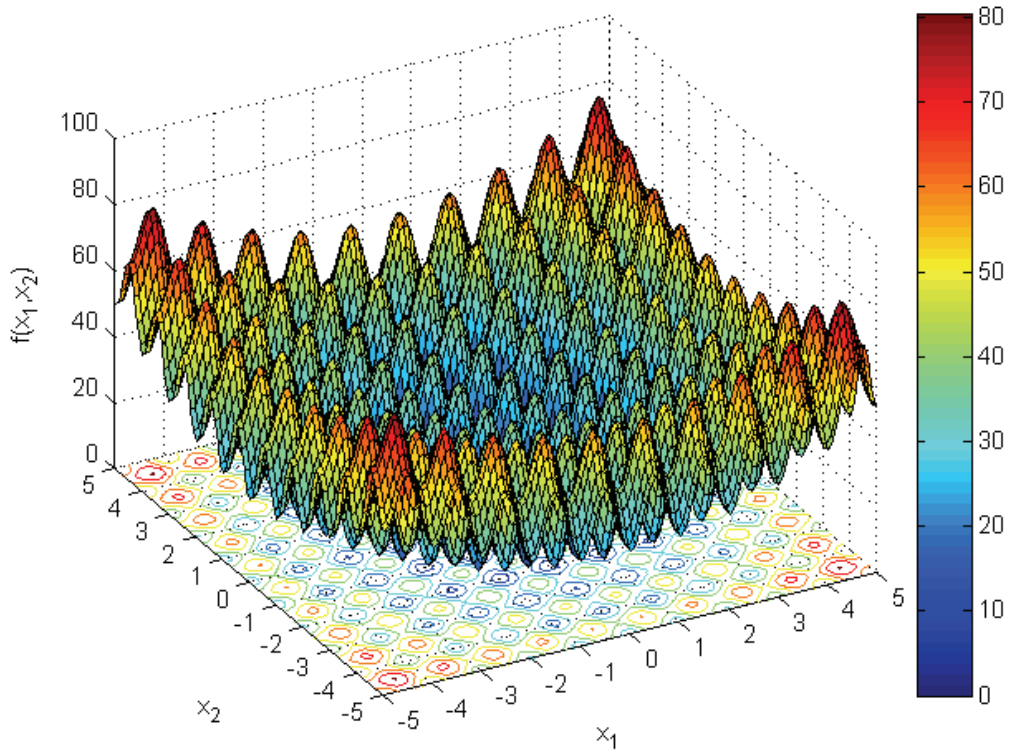
Optimalizační algoritmy jsou testovány dvěma postupy. Jedním postupem je porovnání dosažených výsledků testovaného algoritmu s výsledky, které byly zjištěny jinými algoritmy. Druhým postupem je použití nelineárních testovacích funkcí, u kterých je známé správné řešení, tj. globální minimum.

Algoritmus diferenciální evoluce budeme testovat na dvou nelineárních funkcích, ve kterých známe hodnoty globálních minim. Dosažené výsledky budeme také porovnávat s výsledky simplexové metody a gradientní metody největšího spádu.

10.1 První testovací funkce – Rastriginova funkce

Rastriginova funkce je dána vztahem (10.1). Pro funkci dvou proměnných, tj. pro $D = 2$ je uveden graf na obrázku (10.1).

$$f_1(x_1,...,x_D) = 10D + \left\{ \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i)] \right\} \tag{10.1}$$



Obrázek 10.1: Rastriginova funkce

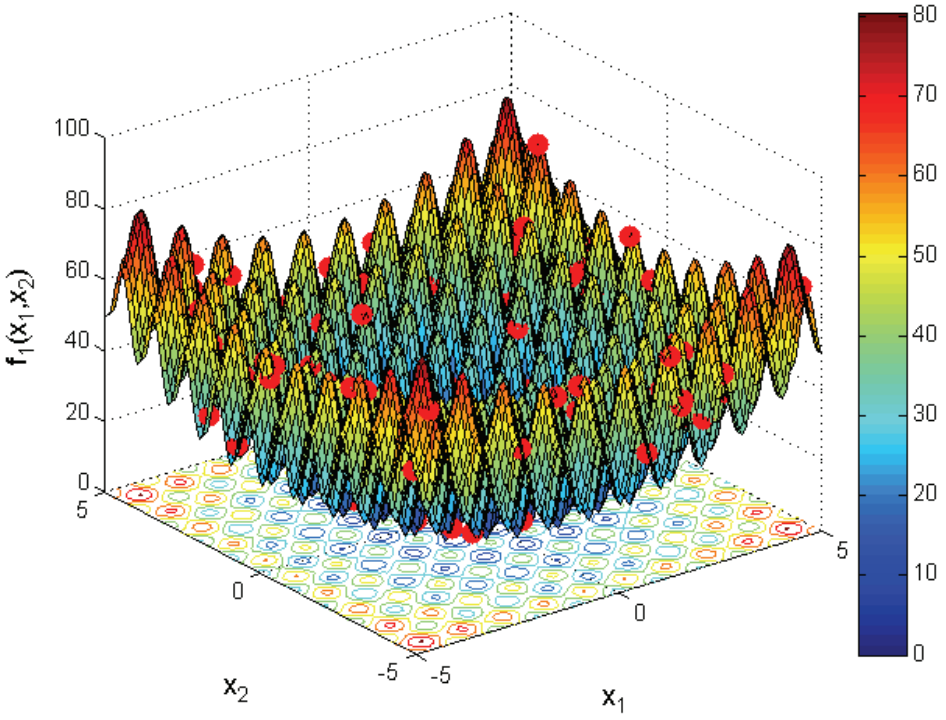
Známá hodnota globálního minima je $\vec{x}^* = (0;0;...;0)$, $f(\vec{x}^*) = 0$. Nalezení minima této funkce je považováno za složitou úlohu globální optimalizace.

Řešení pomocí algoritmu diferenciální evoluce

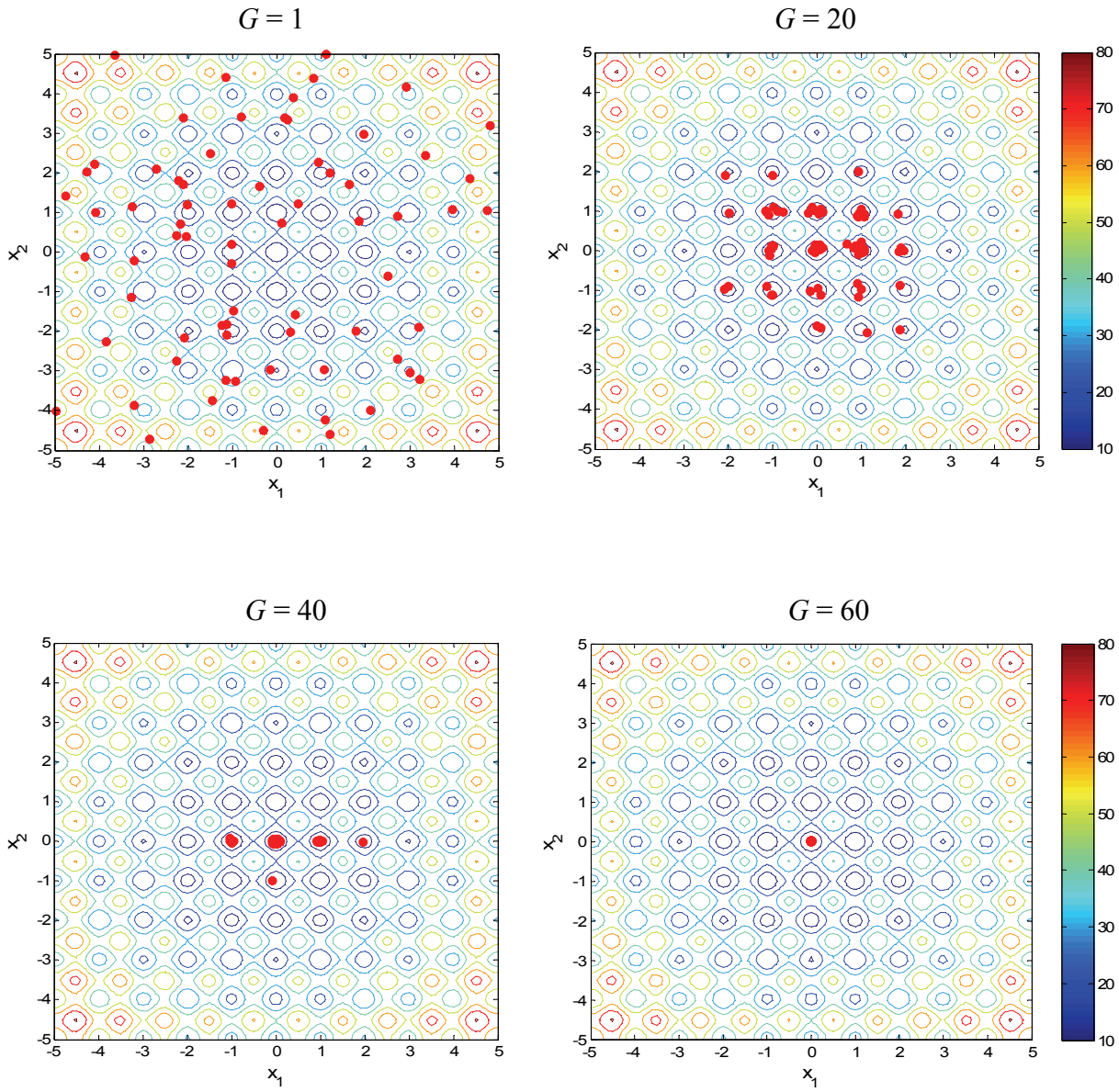
Obrázek 10.2 ukazuje rozmístění počáteční populace na ploše Rastriginovy funkce. Algoritmus byl spuštěn desetkrát a v každé generaci byla zjištěna nejmenší hodnota funkce. Obrázek 10.4 deklaruje vývoj hodnoty funkce k nulové hodnotě v deseti historiích h_1 až h_{10}. Především z něj vidíme, že vývoj nejlepší hodnoty postupuje vždy jinak, protože s každým novým spuštěním úlohy je vygenerována různá populace. Celkový vývoj v generacích ilustruje obrázek (10.3).

Parametry diferenciální evoluce:

- počet jedinců populace $NP = 70$
- maximální počet generací $NG = 60$ ($G = 0, 1, 2, \ldots, NG$)
- křížící konstanta $CR = 0{,}4$
- mutační konstanta $F = 0{,}8$
- omezení parametrů $-5 \le x_j \le 5$, $j = 1, 2$
- varianta DE/rand/1/bin

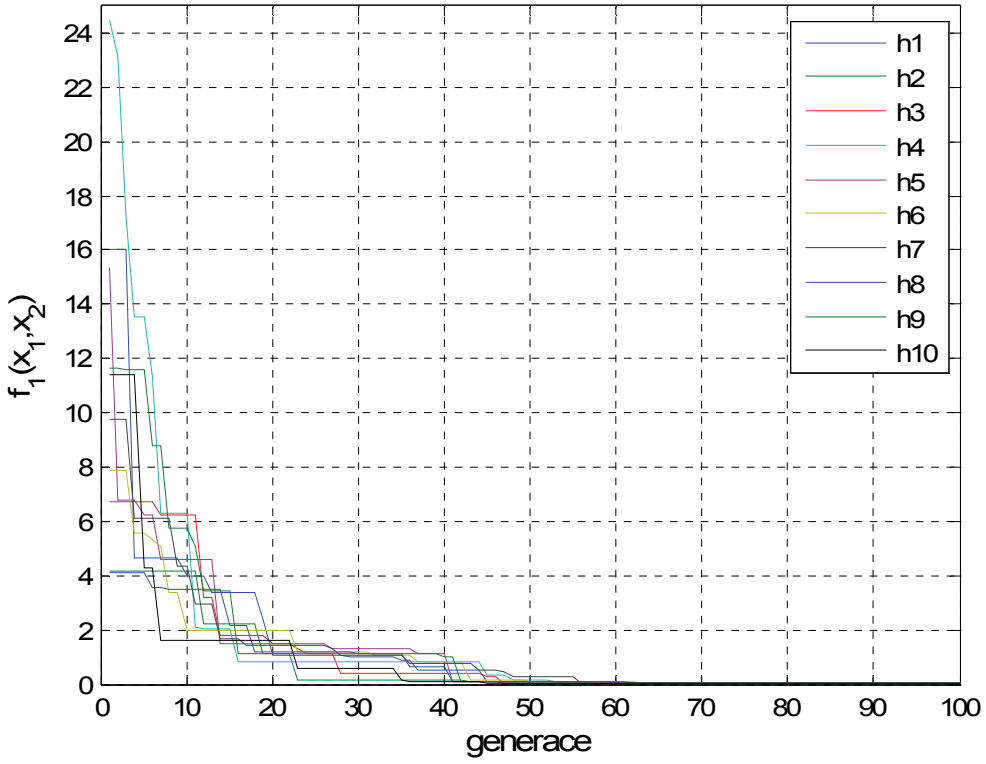


Obrázek 10.2: Počáteční populace ($G = 0$)



Obrázek 10.3: Vývoj populace v generacích G

Nalezené souřadnice globálního minima jsou $\vec{x}^* = [x_1\ x_2] =$ **[-1,2725; -1,4187]**$\cdot 10^{-4}$ s hodnotou funkce f(-1,2725; -1,4187) = 7,2056 $\cdot$ 10^{-6}.



Obrázek 10.4: Historie vývoje nejlepší hodnoty funkce

Řešení pomocí simplexové metody

Simplexová metoda je implementována v matlabu funkcí fminserch. Vstupním parametrem je účelová funkce, počáteční přiblížení a maximální počet iterací. Nalezené řešení bylo vždy blízko bodu odhadu (obrázek 10.5) $\vec{x}^* = [x_1\ x_2] = [2;\ 2]$.

Nalezené řešení: $\vec{x}^* = [x_1\ x_2] = [\mathbf{1{,}9899;\ \ 1{,}9899}]$.

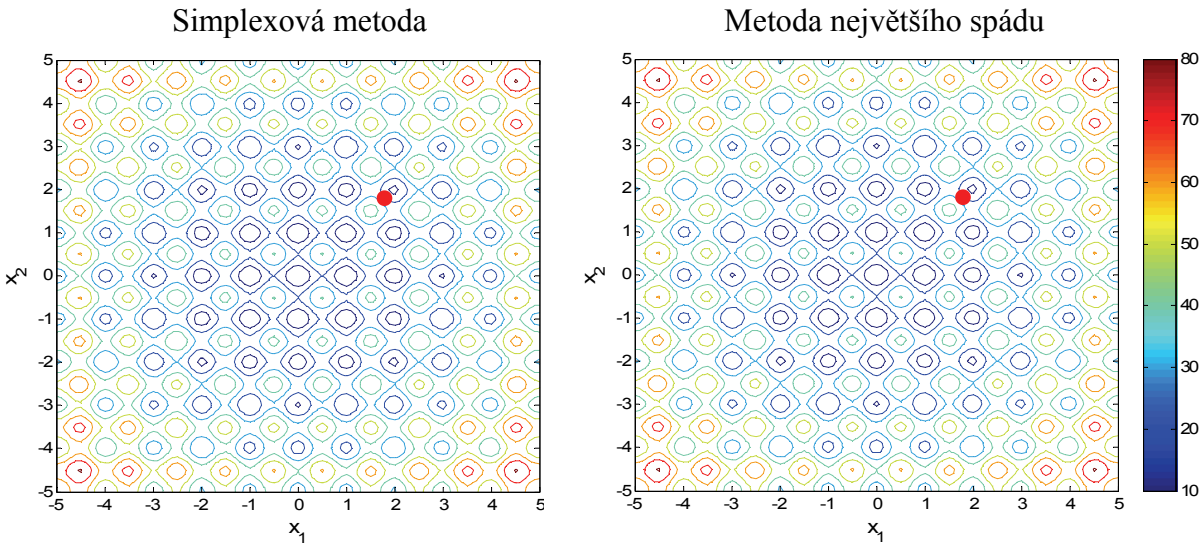
Hodnota funkce $f(1{,}9899;\ \ 1{,}9899) = -3{,}9289$.

Řešení pomocí metody největšího spádu

Metoda největšího spádu byla v Matlabu realizována funkcí, která používá symetrický odhad derivace a konstantní krok, počáteční odhad $\vec{x} = [2;\ 2]$. Funkce byla několikrát spuštěna pro různý počet iterací se stejnými výsledky (obrázek 10.5).

Nelezené řešení: $\vec{x}^* = [x_1\ x_2] = [\mathbf{1{,}7901;\ \ 1{,}7901}]$.

Hodnota funkce $f(1{,}7901;\ \ 1{,}7901) = -3{,}4836$.

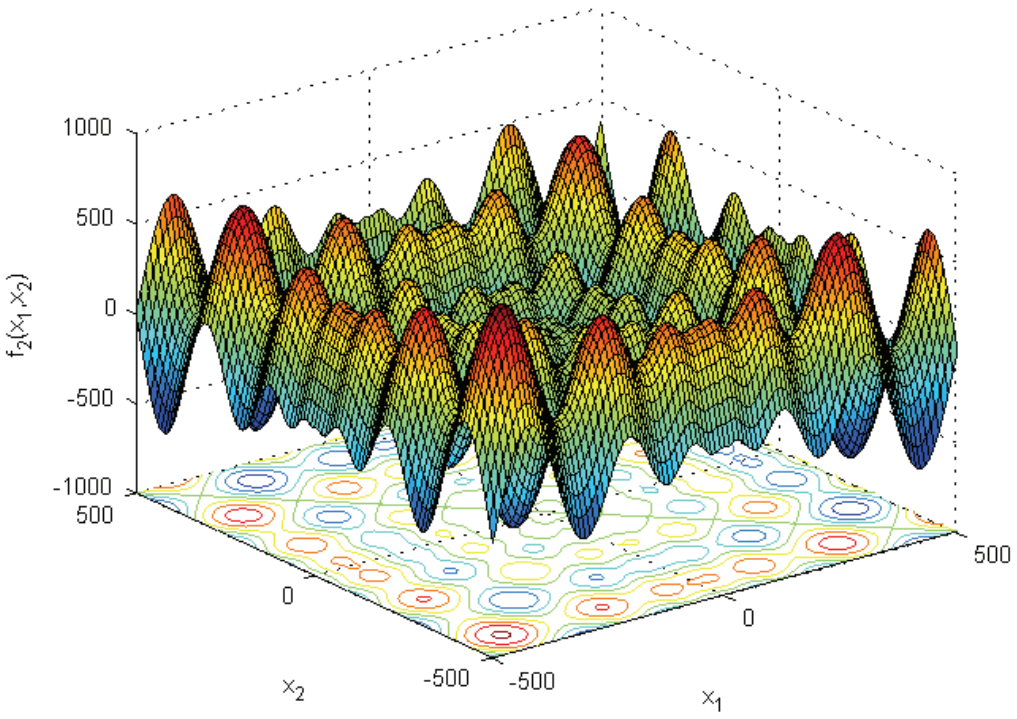


Obrázek 10.5: Nalezená řešení simplexovou metodou a metodou největšího spádu

10.2 Druhá testovací funkce – Schwefelova funkce

Schwefelova funkce je pro *D* parametrů je dána vztahem (10.2), její geometrická interpretace pro dva parametry je uvedena na obrázku (10.6).

$$f_2(x_1,...,x_D) = \sum_{i=1}^{D}[-x_i \sin(\sqrt{|x_i|})] \tag{10.2}$$

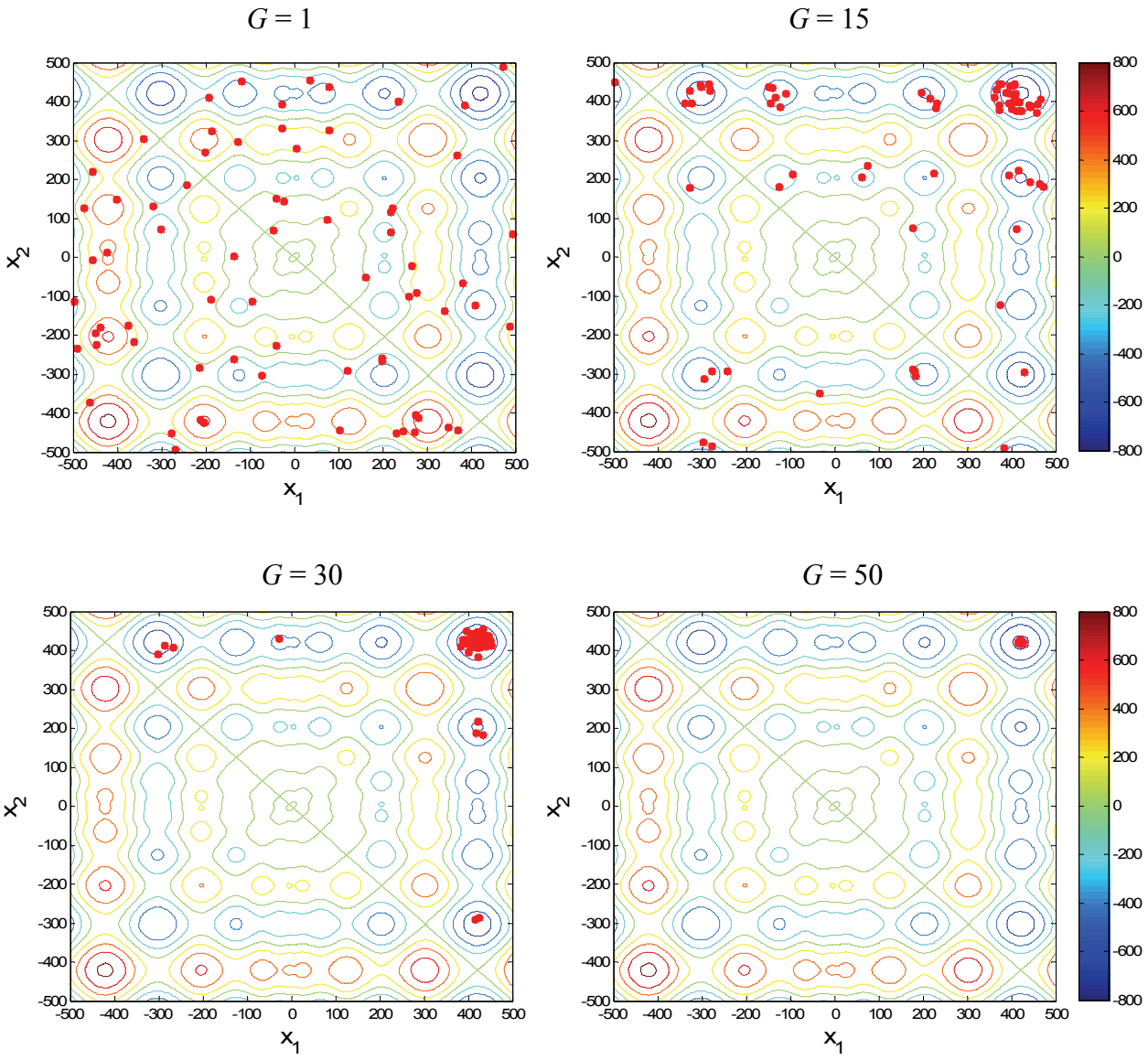


Obrázek 10.6: Schwefelova funkce

Známá hodnota globálního minima $\vec{x}^* = (420{,}97; 420{,}97; ...; 420{,}97)$ je pro rozsah parametrů $x_i \in <-500;\ 500>$, $f(\vec{x}^*) = -418D$.

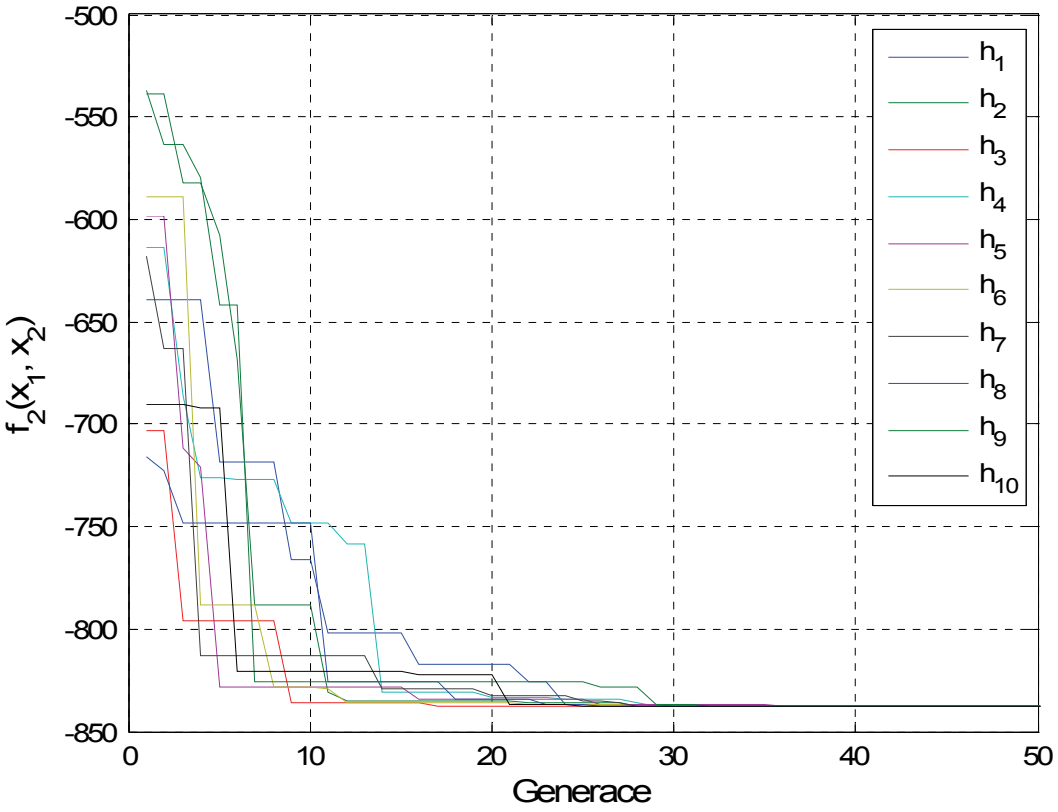
Řešení pomocí algoritmu diferenciální evoluce

Parametry diferenciální evoluce jsou stejné jako v předchozím případě, jen se změnily intervaly parametrů $-500 \le x_j \le 500$, $j = 1, 2$, počet generací $NG = 50$. Vývoj v několika generacích deklaruje obrázek 10.7. Pro deset různých spuštění úlohy se stejnými parametry je vývoj funkční hodnoty nejlepšího řešení uveden na obrázku 10.8.



Obrázek 10.7: Vývoj populace v generacích G

Nalezené souřadnice globálního minima jsou $\vec{x}^* = [x_1\ x_2]$ = [**420,8929; 420,7889**] s hodnotou funkce f(420,8929; 420,7889) = -837,9610.



Obrázek 10.8: Historie vývoje nejlepší hodnoty funkce

Řešení pomocí simplexové metody

Nalezené řešení: $\vec{x}^* = [x_1, x_2]$ = [**203,8143; -124,8294**].

Počáteční odhad $[x_1, x_2]$ = [2; 2]. Funkce byla několikrát spuštěna pro různý počet iterací s podobnými výsledky (obrázek 10.9).

Nelezené řešení: $\vec{x}^* = [x_1\ x_2]$ = [**5,2392; 5,2392**].

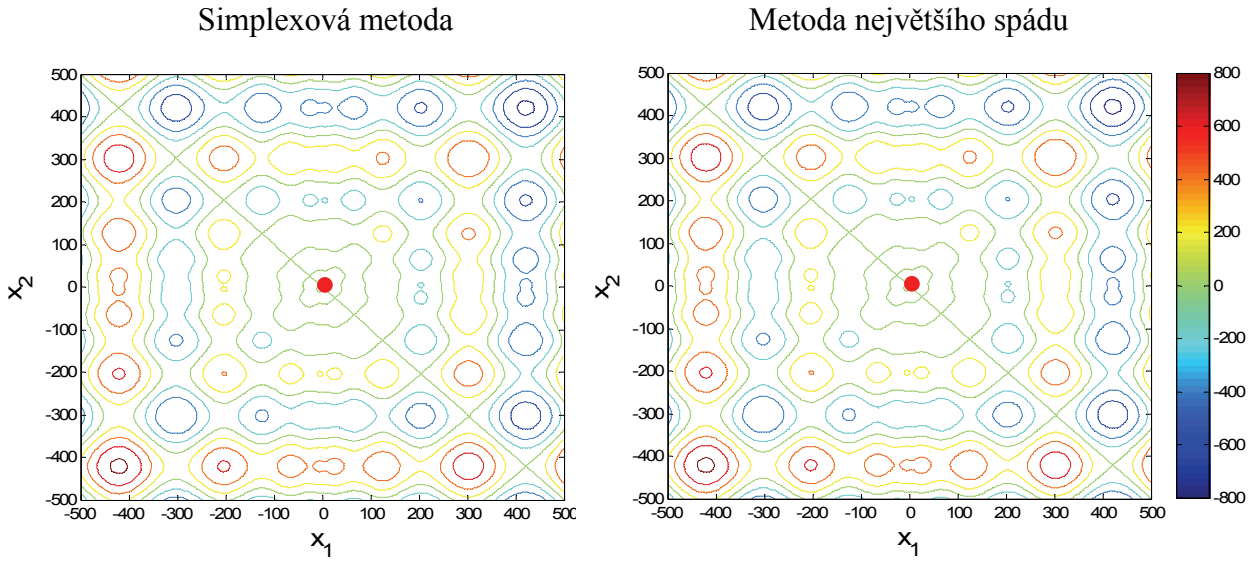
Hodnota funkce f(5,2392; 5,2392) = -7,8906.

Řešení pomocí metody největšího spádu

Počáteční odhad $[x_1, x_2] = [2; 2]$. Funkce byla několikrát spuštěna pro různý počet iterací s podobnými výsledky (obrázek 10.9).

Nelezené řešení: $\vec{x}^* = [x_1\ x_2] = [\mathbf{5; 5}]$.

Hodnota funkce $f(5;\ 5) = -7{,}8675$.



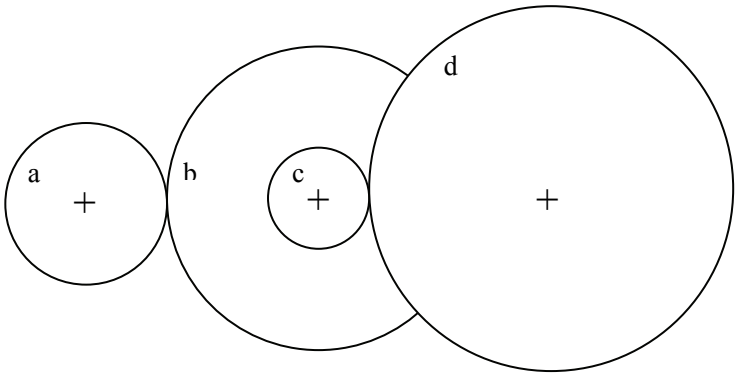
Obrázek 10.9: Nalezená řešení simplexovou metodou a metodou největšího spádu

11 Technická aplikace – návrh převodu ozubených kol

Tento příklad podle [5] je zaměřen na optimalizaci poměru převodu čtyř ozubených kol, toto rozmístění ozubených kol je uvedeno na obrázku 11.1. Převodový poměr *I* je definován jako poměr úhlové rychlosti hnacího členu k úhlové rychlosti hnaného členu:

$$I = \frac{\omega_o}{\omega_i} = \frac{z_a}{z_b}\frac{z_c}{z_d} \tag{11.1}$$

kde ω_o a ω_i jsou jednotlivé úhlové rychlosti a *z* je počet zubů každého kola *a*, *b*, *c*, *d*. Cílem je nalézt počet zubů na čtyřech kolech, aby byl splněn požadovaný převodový poměr $I_p = \frac{1}{6{,}931} = 0{,}1443$. Minimální počet zubů je stanoven na 12 a maximální na 60.



Obrázek 11.1: Rozmístění ozubených kol

Úlohu hledání počtu zubů jednotlivých kol můžeme formulovat takto:

nalézt

$\vec{x} = (x_1, x_2, x_3, x_4)$, $x \in \{12;13;14;...;60\}$

aby minimalizovaly funkci

$$f(\vec{x}) = \left(I_p - \frac{x_1 x_2}{x_3 x_4} \right)^2$$

s ohledem na omezení

$12 \le x_i \le 60$, $i = 1,2,3,4$

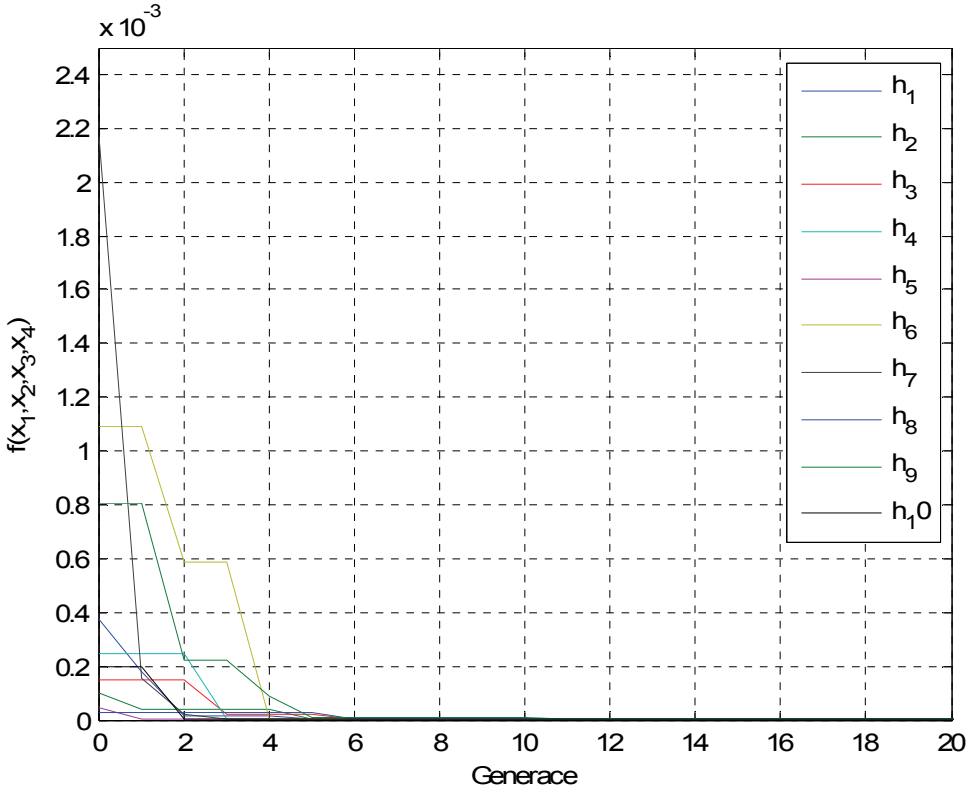
Účelová funkce je definována jako druhá mocnina rozdílu mezi požadovaným převodovým poměrem I_p a aktuálně počítaným poměrem I. Hodnoty parametrů jsou omezeny na celočíselné hodnoty v intervalu od 12 do 60.

Řešení pomocí algoritmu diferenciální evoluce

Parametry diferenciální evoluce:

- počet jedinců populace $NP = 70$
- počet generací $NG = 20$
- křížící konstanta $CR = 0{,}4$
- mutační konstanta $F = 0{,}8$
- omezení parametrů $12 \le x_j \le 60$, $j = 1, 2$
- varianta DE/rand/1/bin

Algoritmus byl spuštěn desetkrát a v každé generaci byla zjištěna nejmenší hodnota funkce. Obrázek 11.2 deklaruje vývoj nejlepší hodnoty funkce k nulové hodnotě v deseti historiích h_1 až h_{10}.



Obrázek 11.2: Vývoj nejlepší hodnoty funkce v deseti spuštěních úlohy

Řešení pomocí simplexové metody

Pomocí funkce fminserch se hledalo řešení účelové funkce s počátečním odhadem řešení [x_1 x_2 x_3 x_4] = [20; 20; 20; 20].

Řešení hledané pomocí gradientní metody největšího spádu

U metody největšího spádu byl zadán stejný počáteční odhad jako u simplexové metody.

V následující tabulce je uvedeno řešení, které bylo nalezeno v oboru reálných čísel, tyto čísla byla matematicky zaokrouhlena a teprve potom vyčíslena hodnota účelové funkce $f(\vec{x})$ a převodového poměru *I*.

Tabulka 11.1: Nalezené řešení pomocí třech metod

Položka	Diferenciální evoluce	Simplexová metoda	Metoda největšího spádu
x_1	24	23	11
x_2	15	21	26
x_3	48	19	26
x_4	51	17	26
$f(\vec{x})$	$7{,}7256 \cdot 10^{-6}$	$3{,}4762 \cdot 10^{-6}$	0,3777
I	0,1471	0,1424	0,4231

Algoritmus diferenciální evoluce byl celkem desetkrát spuštěn a generoval stále nová řešení, zatímco simplexová metoda s metodou největšího spádu pouze jedno řešení pro stejný počáteční odhad, pro jiný počáteční odhad pak obě metody generovaly jiné řešení. Z tabulky 11.1 bylo nalezeno vyhovující řešení simlexovou metodou a diferenciální evolucí.

12 Programové funkce pro využití algoritmu diferenciální evoluce

V předešlém textu byl algoritmus testován na třech úlohách s použitím vytvořených funkcí v Matlabu. Úplný popis použitých algoritmů s pseudokódy byl uveden v kapitole 8, zde zbývá jen uvést názvy vytvořených funkcí a způsob jejich použití. V podstatě byly vytvořeny dvě funkce, které přímo řeší problematiku optimalizace. Jedna z nich se zabývá generováním počáteční populace, další pak jejím šlechtěním v jednotlivých generacích. Záměrně je tvorba počáteční populace vyjmuta z funkce přímo se zabývající šlechtěním populace, protože existují i další možnosti rozmístění populace podle uživatelem stanovených kritérií.

12.1 Programová funkce pro tvorbu počáteční populace

Počáteční populace představuje nultou generaci, která je vytvořena podle vztahu (8.1). Je startovacím bodem pro další vývoj populace, obecně při řešení problému neznáme pozici globálního minima, proto se musíme pokusit vytvořit populaci po celé ploše prohledávaného prostoru, tento vztah provede rozmístění členů populace s rovnoměrným rozdělením. To zajišťuje funkce s názvem pocatecniGenerace, která má vstupní parametr typ *struktura*, funkce vrací také *strukturu*, použití je následující:

pole2 = pocatecniGenerace(pole1)

S*truktura* je datový typ, který dokáže seskupit data různého formátu. Vytvoření položek *struktury* a jejich plnění lze provést několika způsoby, z toho zde uvedeme tu nejjednodušší metodu, tj. vytvoření a zároveň její inicializace přes tečkovou notaci, např. pole.NP = 100. Vytvořili jsme tak proměnnou pole typu *struktura* s jednou položkou NP, jejíž hodnota je 100. Přístup k položce je opět přes tečku, např. přiřazení hodnoty NP do proměnné provedeme zápisem u = pole.NP. Ostatní přiřazení je podle následujícího kódu:

```
pole.NP = 70;
pole.D = 2;
pole.fce = '10*2+x(1)^2+x(2)^2-10*(cos(2*pi*x(1))+cos(2*pi*x(2)))';
pole.interval = [-5 5];
```

```
pole.VTR = presnost; % presnost max. hodnota fce-min. hodnota fce<VTR
pole = pocatecniGenerace(pole);
```

Takto použitá funkce `pole = pocatecniGenerace(pole)` přepíše *strukturu* pole a doplní ji o další položky, které funkce vrací, jako je např. nejlepší hodnota funkce, historie nejlepší hodnoty a další položky. Výhodou tohoto přístupu je, že si vystačíme s jednou proměnnou typu struktura, která uchovává všechny parametry úlohy. Podrobnější popis je uveden v samotné funkci. Funkci lze zadat pouze jeden interval, který se převezme pro všechny ostatní parametry účelové funkce, jinak lze samozřejmě zadat do vektoru příslušný počet intervalů. Např. pro funkci s dvěma parametry $D = 2$, s požadavkem na rozsah pro první parametr <-1; 1> a pro druhý <-2; 2>, zadáme interval takto:

```
pole.interval = [-1 1 -2 2];
```

12.2 Programová funkce pro evoluční cyklus

Optimalizační proces se odehrává v jednotlivých generacích, k tomuto účelu byla vytvořena funkce, které se předává stejná struktura s názvem pole, volání funkce je pole = DE(pole). Do struktury pole musíme přidat (pole už z předchozího případu obsahuje některé položky) celkem mnoho parametrů, které si žádá algoritmus diferenciální evoluce. Tyto parametry jsou následující:

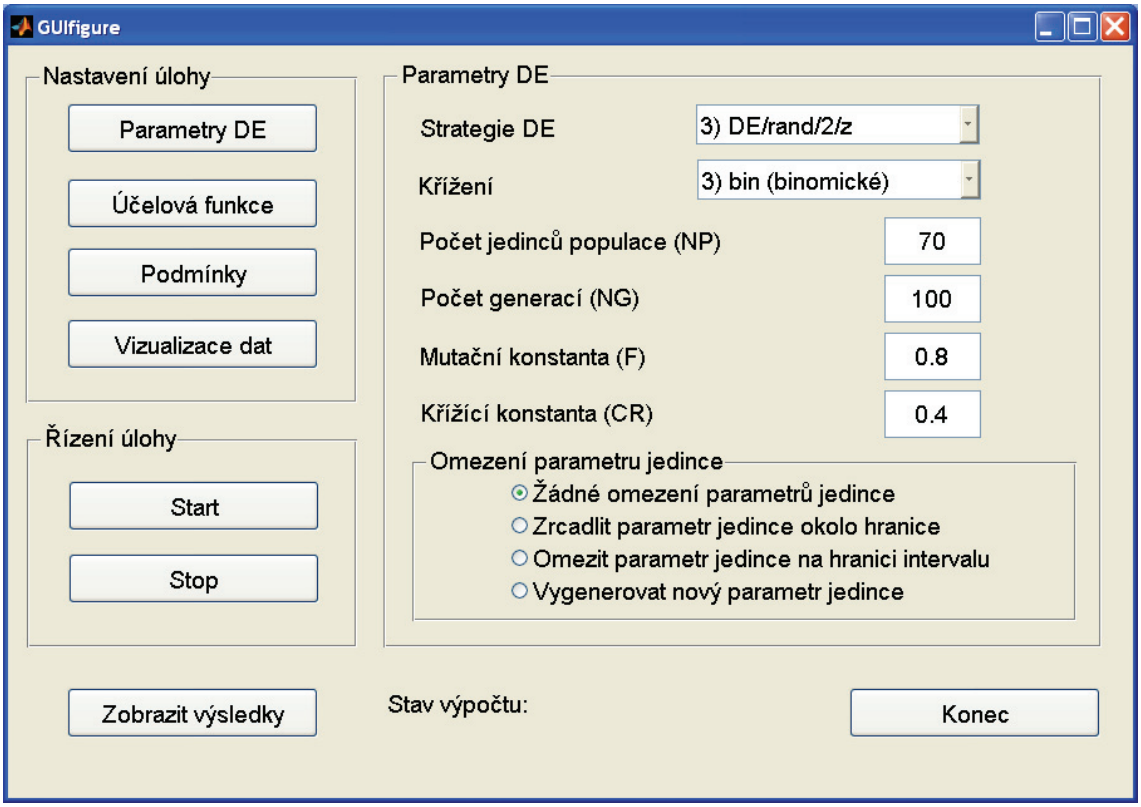
```
pole.CR = 0.4;          % krizici konstanta
pole.F = 0.8;           % mutacni konstanta
pole.strategie = 3;     % volba varianty diferenciální evoluce

pole.presnost = 1e-10;  % rozdil mezi nejlepším a nejhorším jedincem

pole.podminky = 0;      % 1 uvazovani funkčních omezeni, 0 neuvazovani
pole.bnd_constr = 1;    % volba omezeni parametru, 0 neuvazovani
pole.itermax = 100;     % počet generaci
pole.vypocetX = 0;      % 1 uvazovani funkčních omezeni, 0 neuvazovani
```

Pro použití algoritmu diferenciální evoluce byl vytvořen textový soubor (M-File) se jménem test.m (viz. příloha). Tento soubor obsahuje bližší informace o nastavení parametrů.

13 Grafické uživatelské prostředí

Pro použití vytvořených funkcí bylo vytvořeno uživatelské prostředí v Matlabu pomocí tzv. systému Handle Graphics. Systém Handle Graphic je nástroj, který dovoluje efektivně pracovat s grafickými objekty [7].

Grafické rozhraní obsahuje v levé části tlačítka pro volbu nastavení a řízení úlohy, v pravé části je vždy zobrazena zvolená oblast, zde např. na obrázku 13.1 je zvoleno tlačítko Parametry DE, v pravé části jsou možnosti nastavení pro tuto volbu. Všechna nastavení jsou v souladu s uvedenou tématikou a zavedeným značením.



Obrázek 13.1: Grafické uživatelské rozhraní

Zásady pro zadávání hodnot:

Účelová funkce se zadává standardně se všemi operátory, které jsou v Matlabu povoleny. Proměnné se zadávají ve formátu $x(1)$, $x(2)$, atd. Např. rotační paraboloid zadáme ve tvaru $x(1)$^2+$x(2)$^2. Podobným způsobem zadáváme omezující podmínky g_i, ovšem ty musí být zadány jako vektor s oddělujícím středníkem, např. dvě omezující

podmínky zadáme ve tvaru [*x*(1)^2+*x*(2)^2-5; *x*(1)^2+*x*(2)^2+4]. Interval pro proměnné pak postačí zadat pouze čísla, např. chceme zadat dva intervaly pro dvě proměnné $x_1 \in <-1;1>, x_2 \in <-2;2>$, tak do pole *intervaly argumentů funkce* zadáme čísla -1, 1, 2, -2, platí, že stačí zadat jeden interval, který se převezme pro ostatní parametry.

Parametr přesnost je ukončovacím kritériem pro algoritmus diferenciální evoluce. Kontroluje se rozdíl funkčních hodnot nejhoršího a nejlepšího jedince populaci v každé generaci. Ostatní nastavení vyplývá z předešlého textu. Uživatelské prostředí spustíme zápisem start do okna Command Windows v Matlabu.

14 Závěr

Evoluční algoritmus diferenciální evoluce nalézá globální extrémy složitých nelineárních funkcí, které klasické numerické metody nalézají jen obtížně nebo vůbec. Evoluční strategie přenesená do výpočetních metod, založená na vývoji populace v jednotlivých generacích, podává dobré výsledky. Metody, které využívají jednoho jedince, často „uvíznou“ v lokálním extrému, tak jak se v této práci potvrdilo u gradientní metody největšího spádu a metody simplexu.

V testech byla volena různá hodnota konstanty mutace a křížení pro shodný počet generací. Výsledky potvrdili očekávání v tom smyslu, že mutační konstanta představuje krok optimalizace a že křížení má velký vliv na tvorbu nových řešení. Větší hodnoty konstanty mutace a křížení zajišťovali rychlou konvergenci populace do globálního extrému s méně přesným řešením. Přesnějších výsledků bylo dosaženo především malou volbou konstanty mutace, ale potřebný počet generací byl vyšší.

Test provedený na rotačním paraboloidu jako na funkci s jedním globálním extrémem ukázal, že řešení bylo nalezeno metodou největšího spádu i metodou simplexu za mnohem kratší dobu, než algoritmem diferenciální evoluce. Musíme zde konstatovat, že neexistuje obecný algoritmus, který by vyřešil obecný problém a přitom byl časově optimální, proto pro daný problém by se měla volit správná výpočetní metoda. Společným jmenovatelem všech optimalizačních algoritmů je nalézt správné řešení za přijatelný čas.

Seznam použité literatury

[1] Godfrey C. Onwubolu, B. V. Babu. *New Optimization Techniques in Engineering (Studies in Fuzziness and Soft Computing)*. Springer, 1. edition, 2004, ISBN-13: 978-3540201670

[2] Kenneth V. Price, Rainer M. Storn, Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer, 1. edition, 2005, ISBN-13: 978-3540209508

[3] Uday K. Chakraborty. *Advances in Differential Evolution (Studies in Computational Intelligence)*. Springer, 1. edition, 2008, ISBN-13: 978-3540688273

[4] Zelinka, I.: *Evoluční výpočetní techniky - principy a aplikace*. 1. české vydání. Praha: BEN, 2008. 536 s. ISBN 978-80-7300-218-3.

[5] Zelinka, I.: *Umělá inteligence v problémech globální optimalizace*. Praha : BEN, 2002. 192 s. ISBN 80-7300-069-5.

[6] Zaplatílek, K.: Doňar, B.: *Matlab - tvorba uživatelských aplikací*. Praha : BEN, 2004. 215 s. ISBN 80-7300-133-0.

[7] Tvrdík, J. [online]. *Evoluční algoritmy*,2004 [cit. 21. 05 2010]. Dostupné z WWW: http://prf.osu.cz/doktorske_studium/dokumenty/Evolutionary_Algorithms.pdf>.

Seznam obrázků

Příloha – obsah CD

V přiloženém CD jsou složky:

- TextDP – tato složka obsahuje text diplomové práce ve formátu pdf.

- Test – složka obsahuje soubory pocatecniGenerace.m, test.m a DE.m. Soubor test.m demonstruje použití algoritmu diferenciální evoluce.

- GUI – tato složka obsahuje soubory start.m, pocatecniGenerace.m, DE.m, vykreslitFci.m, vykreslitKontury.m, GUIfigure.m, GUIfigure.fig.

- Metody – složka metody obsahuje soubory fmins.m a NejvetsiSpad.m