ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

Ladění a konstrukce heuristických optimalizačních algoritmů

Jakub Lhoták
Otevřená informatika

Květen 2020
Vedoucí práce: Ing. Petr Pošík, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lhoták** Jméno: **Jakub** Osobní číslo: **474732**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Ladění a konstrukce heuristických optimalizačních algoritmů

Název bakalářské práce anglicky:

Tuning and construction of heuristic optimization algorithms

Pokyny pro vypracování:

Heuristické optimalizační algoritmy (např. Simplexová metoda Nelder-Meada) často mají nějaké interní parametry, nebo celé komponenty, které nemusejí být nastaveny optimálně vzhledem k jisté cílové třídě problémů. Cílem tohoto projektu je prozkoumat metody pro nalezení instancí daného optimalizačního algoritmu specializovaných na jisté podtřídy úloh.
1. Vyberte si určitý heuristický algoritmus a upravte jeho implementaci/parametrizaci tak, aby byla vhodná k optimalizaci.
2. Zvolte několik sad funkcí/problémů, které budou reprezentovat různé cílové třídy.
3. Navrhněte vhodnou metriku pro popis kvalitu algoritmu na dané třídě problémů.
4. Vyberte nebo navrhněte optimalizační algoritmus pro ladění/konstrukci specializované heuristiky.
5. Porovnejte nalezené specializované heuristiky s původní heuristikou a vyhodnoťte přínosy pro zvolenou třídu problémů. Posuďte, jaké efekty specializace algoritmu má pro ostatní třídy problémů.

Seznam doporučené literatury:

[1] Nelder, J.A. and Mead, R. (1965), “A simplex method for function minimization”, Comput. J., 7, pp. 308–313.
[2] Rosenbrock, H. H. An automatic method for findong the greatest or least value of a function., Comp. J., 3, pp 175-184, (1960).
[3] Fajfar, I., Puhan, J., Burmen, A. Evolving a Nelder–Mead Algorithm for Optimization with Genetic Programming. Evolutionary Computation 25(3): 351–373, 2017.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Pošík, Ph.D., Analýza a interpretace biomedicínských dat FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **03.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Petr Pošík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucímu práce panu Ing. Petrovi Pošíkovi za velké množství užitečných podnětů, rad a připomínek, které mi v průběhu mé práce na projektu poskytl.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 17. května 2020

.....

Abstrakt / Abstract

Tato práce se zabývá laděním heuristických optimalizačních algoritmů. Práce se zaměřuje speciálně na optimalizaci a ladění Nelder-Meadovy simplexové metody. Na začátku práce je čtenář seznámen s konstrukcí a možnostmi ladění Nelder-Meadovy metody. V dalších částech práce je popsán způsob ladění metody pro kvadratické funkce a Rosenbrockovu funkci. Součástí práce je také porovnání efektivity odladěné metody s původní neupravenou verzí. Ukázalo se, že odladěné verze metody dosahují při optimalizaci kvadratických funkcí i Rosenbrockovy funkce znatelně lepších výsledků než původní verze metody.

Klíčová slova: Heuristický algoritmus; Nelder-Meadova metoda; optimalizace; ladění parametrů

This thesis deals with a tuning of heuristic optimization algorithms. The main focus of this thesis is optimization and tuning of the Nelder-Mead simplex method. At the beginning the reader is acquainted with the construction of the Nelder-Mead method and its possibilities for tuning. In the next sections of the thesis there is described how the method was tuned specifically to be used for optimizing quadratic functions and the Rosenbrock function. There are also sections where both the original and optimized versions of Nelder-Mead method are compared to each other in terms of their efficiency. Newly created versions of method achieved better results when they were used for optimizing quadratic functions and the Rosenbrock function than the original version of the method.

Keywords: Heuristic algorithm; Nelder-Mead method; optimization; parameter tuning

Obsah /

Kapitola 1
Úvod

V praxi se často setkáváme s úlohami, ve kterých řešíme následující problém. Máme systém, který musíme nastavit, případně naladit tak, aby jeho fungování bylo co možná nejefektivnější. V takovém případě se bavíme o procesu zvaném optimalizace. Tento typ úloh se v některých případech dá převést do čistě matematické podoby, kde poté hledáme minimum nebo maximum nějaké matematické funkce. Této funkci se často říká účelová (také cílová) funkce a podle toho jestli hledáme minimum nebo maximum mluvíme o minimalizaci nebo maximalizaci.

Pokud označíme účelovou funkci jako $f : A \to \mathbb{R}$ a budeme ji chtít minimalizovat, pak hledáme $x_0 \in A$ takové, že $f(x_0) \leq f(x)$ pro všechna $x \in A$. Pokud bychom funkci f maximalizovali, pak bychom hledali $x_0 \in A$ takové, že $f(x_0) \geq f(x)$ pro všechna $x \in A$. Množina A se nazývá přípustná množina nebo také množina přípustných řešení. Přípustná množina bývá často podmnožinou eukleidovského prostoru $\mathbb{R}^n$. K vymezení této podmnožiny se používají takzvané omezující podmínky, se kterými se nejčastěji pracuje ve tvaru rovností a nerovností. Minimalizujeme pak například funkci f za podmínky $g(x) \geq 0$. V této práci se s optimalizací funkcí, jejichž přípustná množina by byla vymezena omezujícími podmínkami, nesetkáme a budeme pracovat pouze s přípustnou množinou $\mathbb{R}^n$. Nalezené řešení, které označíme x^{opt} se nazývá optimální řešení. Optimálních řešení může být více, nebo nemusí být žádné. Úloha nemá optimální řešení, pokud neexistuje x_0 takové, že $f(x_0) \leq f(x)$ pro všechna $x \in A$. Pokud bychom například minimalizovali funkci $f(x) = 2x$ na množině reálných čísel, pak pro každé $x \in \mathbb{R}$ platí $f(x - 1) \leq f(x)$, a tedy nemůžeme o žádném $x_0 \in \mathbb{R}$ tvrdit, že je optimálním řešením této úlohy.

K optimalizaci se používá celá řada různých technik, metod a nástrojů [1]. Hledat optimální řešení funkcí můžeme čistě pomocí nástrojů matematické analýzy. To ovšem nemusí vždy být ten nejlépe zvolený postup, neboť optimalizace pouze pomocí matematické analýzy může být velmi obtížná. Může se také stát, že nebudeme mít k dispozici matematický předpis optimalizované funkce, a tudíž optimalizace pouze pomocí nástrojů analýzy bude nemožná. Z těchto důvodů se v praxi často k optimalizaci používají různé numerické a iterační metody, které jsou přizpůsobené k řešení konkrétních typů úloh. Dobrým příkladem může být Gradientní metoda. Ta pracuje v iteracích, ve kterých hledá nové nejlepší řešení pomocí doposud nejlepšího nalezeného a znalosti gradientu v tomto bodě. Výhodou této metody je spolehlivost, neboť se dá zajistit, aby metoda vždy dokonvergovala do nějakého lokálního optima. Nevýhodou metody může být pomalá konvergence nebo nutnost existence první derivace optimalizované funkce.

Jednou z často používaných metod je heuristická Nelder-Meadova simplexová metoda. Tato metoda nepotřebuje ke svému běhu derivace optimalizovaných funkcí a vystačí si pouze s funkčními evaluacemi. Celá tato bakalářská práce se věnuje právě Nelder-Meadově metodě a cílem práce je metodu podrobně prozkoumat a pokusit se o její vylepšení a odladění pro konkrétní sady funkcí.

Nelder-Meadova metoda je zde vybrána pouze jako zástupce z celé řady heuristických optimalizačních algoritmů. Další metoda, která by mohla být zpracována podobným způsobem, je například Rosenbrockova metoda [2].

Kapitola 2
Nelder-Meadova simplexová metoda

Nelder-Meadova simplexová metoda je heuristická optimalizační metoda, kterou v roce 1965 navrhl John Nelder společně s Rogerem Meadem [3]. Používá se k nalezení lokálních minim funkcí více proměnných. Jak už jsem uvedl v úvodní kapitole, tato metoda se hodí zejména tehdy, pokud nemáme k dispozici derivace optimalizovaných funkcí.

Metoda je založena na postupném přemísťování $n+1$ bodů tvořících simplex v n-rozměrném prostoru reálných čísel $\mathbb{R}^n$. Simplex je n-rozměrným zobecněním trojúhelníku. Body simplexu jsou afinně nezávislé a n-rozměrný simplex obsahující $n+1$ bodů může být umístěný pouze v eukleidovském prostoru dimenze n a výše. Přemísťováním bodu je zde myšleno nahrazování nejhoršího bodu simplexu za nový. Nejhorší bod simplexu je takový bod simplexu, pro který platí, že jeho funkční hodnota je největší ze všech bodů simplexu. Nový bod simplexu se bude nacházet někde na polopřímce, jejíž počátečním bodem je nejhorší bod a která protíná geometrický střed zbylých n bodů simplexu. Pokud bychom si body simplexu označili x_1 až x_{n+1} tak, aby platilo $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$, pak polopřímka, na které se bude nacházet nový bod, má svoji parametrickou rovnici určenou takto:

$$p(t) = x_0 + t\,(x_{n+1} - x_0)\,;\ t \in \langle 0, \infty)$$

$$x_0 = \frac{x_1 + x_2 + ... + x_n}{n}$$

Pozice nového bodu na polopřímce je určena podle hodnot 3 vnitřních parametrů označovaných řeckými písmeny α, γ, ρ. Algoritmus iterace na základě hodnot těchto parametrů vypočte přesnou pozici pro nový bod. Jako příklad uvedu konstrukci nového bodu dvourozměrného simplexu na obrázku 2.1.

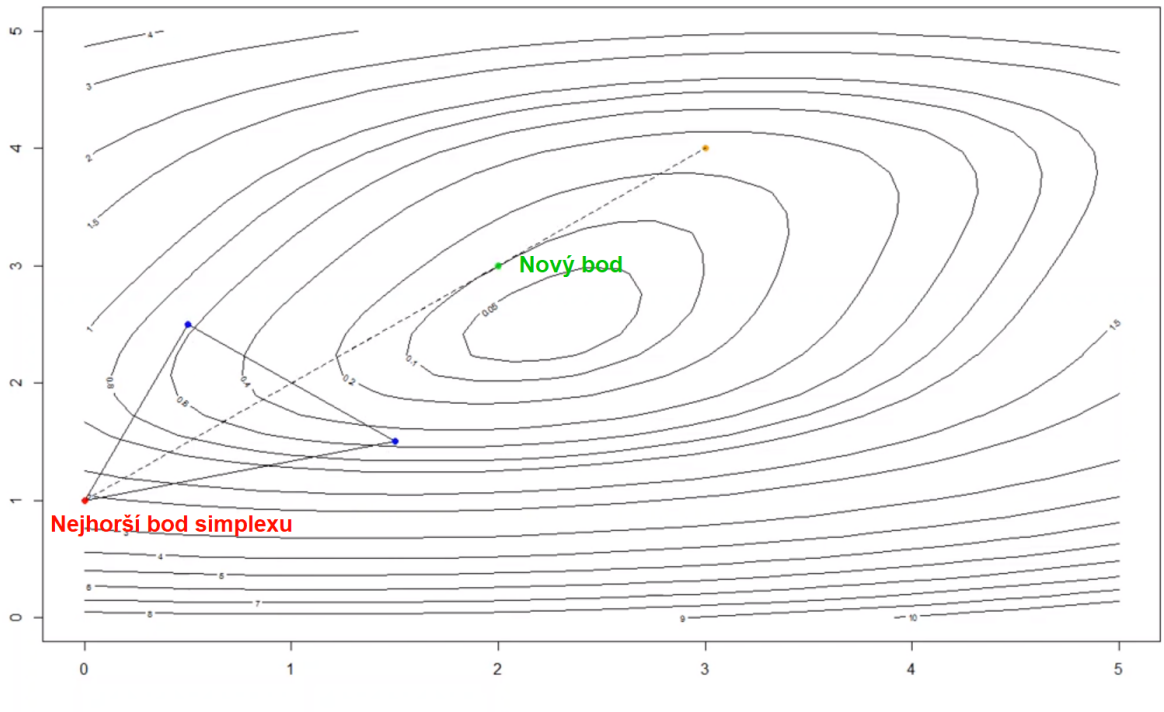
V některých případech nedochází během iterace metody k přesunutí pouze 1 bodu, ale rovnou n nejhorších bodů simplexu. V takovém případě dochází k takzvanému „scvrknutí“ simplexu, kdy se každý bod simplexu (kromě nejlepšího) přesune směrem k nejlepšímu bodu simplexu. Pokud bychom si seřadili body od nejlepšího po nejhorší, tak jak bylo uvedeno výše, mohli bychom scvrknutí bodů vyjádřit takto:

$$x_i := x_1 + \sigma\,(x_i - x_1)\,;\ i = 2, 3, ..., n+1$$

$$0 < \sigma < 1$$

Kde σ je čtvrtý parametr Nelder-Meadovy metody. Pozice scvrknutých bodů záleží pouze na hodnotě tohoto parametru.

V následujících sekcích kapitoly popíšu použitý algoritmus a některé důležité vlastnosti a doplňující informace metody, které jsou pro další kapitoly této práce velmi důležité.



Obrázek 2.1. Příklad konstrukce nového bodu dvourozměrného simplexu. Zdroj: YouTube (Nelder Mead Algorithm Animation; `https://www.youtube.com/watch?v=j2gcuRVbwR0`)

2.1 Implementace metody

Existuje několik variant implementace Nelder-Meadovy metody (například [3], [4] nebo [5]). Já jsem si pro svůj projekt zvolil tu variantu, která je uvedena na anglických stránkách Wikipedie (Nelder-Mead method) [4]. Následující část popisuje algoritmus jedné iterace Nelder-Meadovy metody. Snažíme se o nalezení minima funkce $f : \mathbb{R}^n \to \mathbb{R}$. V každé iteraci máme k dispozici $n + 1$ bodů z $\mathbb{R}^n$, které tvoří aktuální simplex. Algoritmus pracuje následovně:

1. Body aktuálního simplexu $x_1, x_2, ..., x_{n+1}$ seřadíme (označíme) tak, aby platila následující nerovnost:
$$f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$$

2. Vypočítáme nový bod x_0, který je geometrickým středem (centroid, těžiště) n nejlepších bodů simplexu, tedy všech kromě nejhoršího bodu x_{n+1}.
$$x_0 = \frac{x_1 + x_2 + ... + x_n}{n}$$

3. **Reflexe**
Spočítáme nový bod x_r.
$$x_r = x_0 + \alpha\,(x_0 - x_{n+1})$$
$$\alpha > 0$$

Pokud je tento nový bod lepší, než druhý nejhorší bod, ale není lepší než nejlepší, tedy platí $f(x_1) \leq f(x_r) \leq f(x_n)$, pak nahradíme nejhorší bod x_{n+1} bodem x_r a ukončíme iteraci. Pokud ne, pokračujeme krokem 4.

4. **Expanze**
   Pokud je bod x_r lepší, než doposud nejlepší bod, tedy $f(x_r) < f(x_1)$, tak spočítáme nový bod x_e.

$$x_e = x_0 + \gamma\,(x_r - x_0)$$

$$\gamma > 1$$

   Pokud je tento expandovaný bod lepší, než bod x_r $(f(x_e) < f(x_r))$, tak nahradíme nejhorší bod simplexu x_{n+1} bodem x_e a ukončíme iteraci. V opačném případě nahradíme x_{n+1} bodem x_r a ukončíme iteraci.

5. **Kontrakce**
   Pokud bod x_r není lepší, než doposud nejlepší bod $(f(x_r) \geq f(x_1))$, tak spočítáme nový bod x_c.

$$x_c = x_0 + \rho\,(x_{n+1} - x_0)$$

$$0 < \rho < \frac{1}{2}$$

   Pokud je tento nový bod lepší, než nejhorší bod simplexu $(f(x_c) < f(x_{n+1}))$, pak nahradíme bod x_{n+1} bodem x_c a ukončíme iteraci. V opačném případě pokračujeme krokem 6.

6. **Scvrknutí**
   Nahradíme všechny body simplexu kromě nejlepšího (x_1) za body nové podle následujícího vzorce:

$$x_i := x_1 + \sigma\,(x_i - x_1)$$

$$0 < \sigma < 1$$

   Poté ukončíme iteraci.

Body v 1. kroku ve skutečnosti nemusíme řadit. Stačí, když vybereme z aktuálního simplexu nejlepší, druhý nejhorší a nejhorší bod. Pokud bychom měli body simplexu seřazené, odpovídá tato trojice následujícím bodům: x_1, x_n, x_{n+1}. Vynecháním řazení ušetříme trochu výpočetního výkonu.

Parametry α, γ, ρ, σ určují způsob a rychlost konvergence algoritmu. Autoři metody doporučují nastavit parametry na tyto hodnoty: $\alpha = 1$, $\gamma = 2$, $\rho = \frac{1}{2}$, $\sigma = \frac{1}{2}$.

Metodu jsem implementoval v programovacím jazyce Python. Implementoval jsem ji tak, aby bylo možné před zavoláním nastavit hodnoty parametrů α, γ, ρ a σ. Toto mi později dovolilo ladit tyto parametry a zkoušet různá nastavení metody.

2.2 Počáteční simplex

Body počátečního simplexu určují jak bude hledání dalších bodů probíhat. Proto je důležité zvolit počáteční body vhodně vzhledem k řešenému problému. Jedním ze způsobů jak zvolit body simplexu je vybrat si nějaký počáteční bod x_1 (například přibližné řešení), ke kterému přidáme body $x_2, ..., x_{n+1}$. Tyto body vzniknou posunutím bodu x_1 o konstantní hodnotu c vždy v jednom směru souřadnicových os. Pokud si například

zvolíme bod x_1 jako počátek ($x_1 = [0, 0, ..., 0]$), pak souřadnice dalších bodů simplexu jsou následující:

$$x_2 = [c, 0, ..., 0]$$
$$x_3 = [0, c, ..., 0]$$
$$...$$
$$x_{n+1} = [0, 0, ..., c]$$

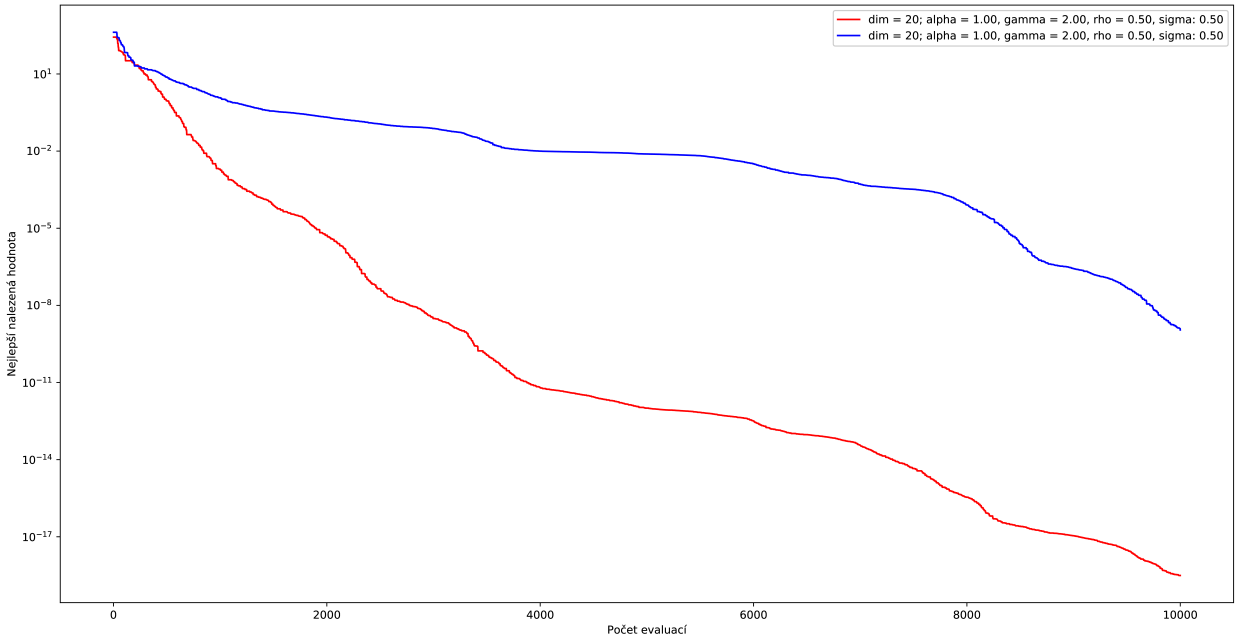
Hodnotu parametru c je vhodné zvolit tak, aby měl počáteční simplex nějakou rozumnou velikost. Jedním ze způsobů, který jsem v projektu často používal, je zvolit hodnotu c tak, aby počáteční simplex pokrýval tu část prostoru, o které si myslíme, že obsahuje hledané minimum. Pokud například víme, že se optimální řešení nachází někde v prostoru $\langle 0, 5\rangle^n$, pak můžeme nastavit počáteční bod jako počátek a hodnotu c zvolit 5.

Tento způsob generování počátečního simplexu se mi velmi osvědčil a rozhodl jsem se simplexy generovat vždy tímto způsobem. Původně jsem zkoušel body generovat náhodně v mezích prohledávaného prostoru. Ovšem výsledky, ke kterým jsem se pomocí Nelder-Meadovy metody s takto náhodně vygenerovaným simplexem dostal, nebyly zdaleka tak přesné a konvergovaly mnohem pomaleji. Na obrázku 2.2 lze dobře vidět rozdíl v rychlosti konvergence Nelder-Meadovy metody při jejím použití k nalezení lokálního (v tomto případě i globálního) minima náhodně vygenerované 20-dimenzionální kvadratické funkce. Funkční hodnota této funkce v bodě, které je hledaným řešením této úlohy je 0. Metoda byla jednou zavolána s počátečním simplexem vygenerovaným způsobem, který jsem popsal na začátku této sekce (červená křivka) a podruhé s počátečním simplexem vygenerovaným náhodně (modrá křivka). Oba běhy metody měly povoleno 10000 funkčních evaluací. Hned na první pohled si lze všimnout, že Nelder-Meadova metoda s náhodně vygenerovaným počátečním simplexem se zdaleka nedostala k hodnotě minima optimalizované funkce tak blízko, jako když tuto metodu zavoláme se simplexem vygenerovaným podle výše zmíněného postupu. Rozdíl v nalezených minimech je opravdu velký (v tomto případě i několik řádů). Vidíme tedy, že volba počátečního simplexu při optimalizaci reálných funkcí pomocí Nelder-Meadovy simplexové metody může velmi ovlivnit průběh metody.

2.3 Tvar simplexu

Počáteční simplex je jediný simplex v celém průběhu metody, u kterého máme nějakou možnost volby jeho tvaru a velikosti. V každé iteraci metody se v simplexu upraví pozice 1 nebo n bodů a změní se jeho tvar, velikost nebo orientace. Souřadnice bodů v simplexu po k iteracích jsou předem pevně určené. Nemůže se tedy stát, že by Nelder-Meadova metoda po stejném počtu iterací se stejným počátečním simplexem dokonvergovala k jinému simplexu. Toto samozřejmě platí pouze pokud jsou parametry α, γ, ρ a σ nastaveny v obou bězích metody stejně a minimalizuje se ta samá funkce.

Občas se při běhu metody stává, že tvar simplexu se postupně změní do podoby, ve které se simplex stává téměř nepoužitelným pro další iterace metody (simplex takzvaně zdegeneruje). Může se například stát, že se simplex úplně zploští do nižší dimenze. K tomu dojde ve chvíli, kdy nově vzniklý bod leží v afinním obalu zbylých n bodů simplexu. K tomuto extrému pravděpodobně v praxi nedochází. Může se ale stát, že nově vzniklý bod bude ležet v těsné blízkosti afinního obalu zbylých n bodů. Takový simplex stále bude n-rozměrný, ovšem jeho zploštělý tvar může způsobovat problémy s konvergencí a celkově zpomalovat průběh metody.

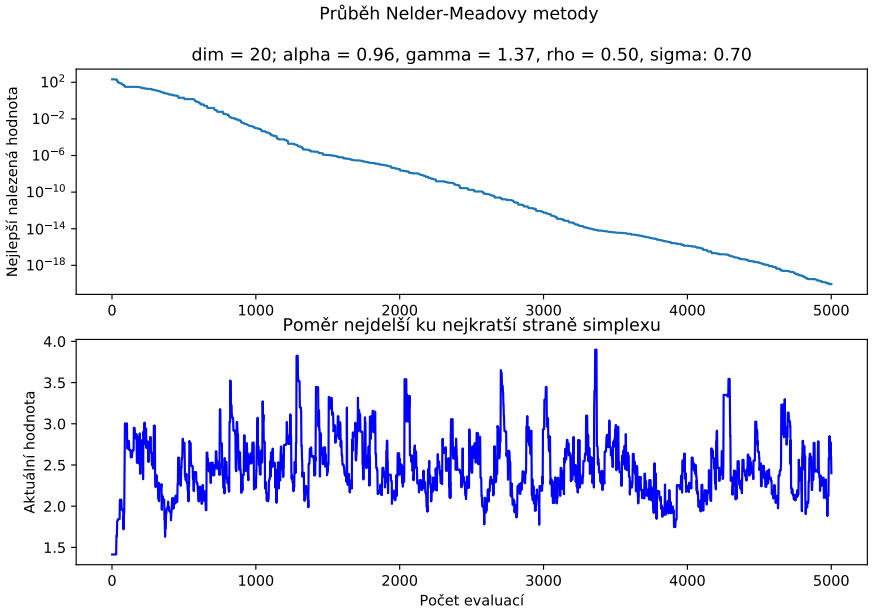


Obrázek 2.2. Průběh optimalizace kvadratické funkce pomocí Nelder-Meadovy metody - rozdílné počáteční simplexy

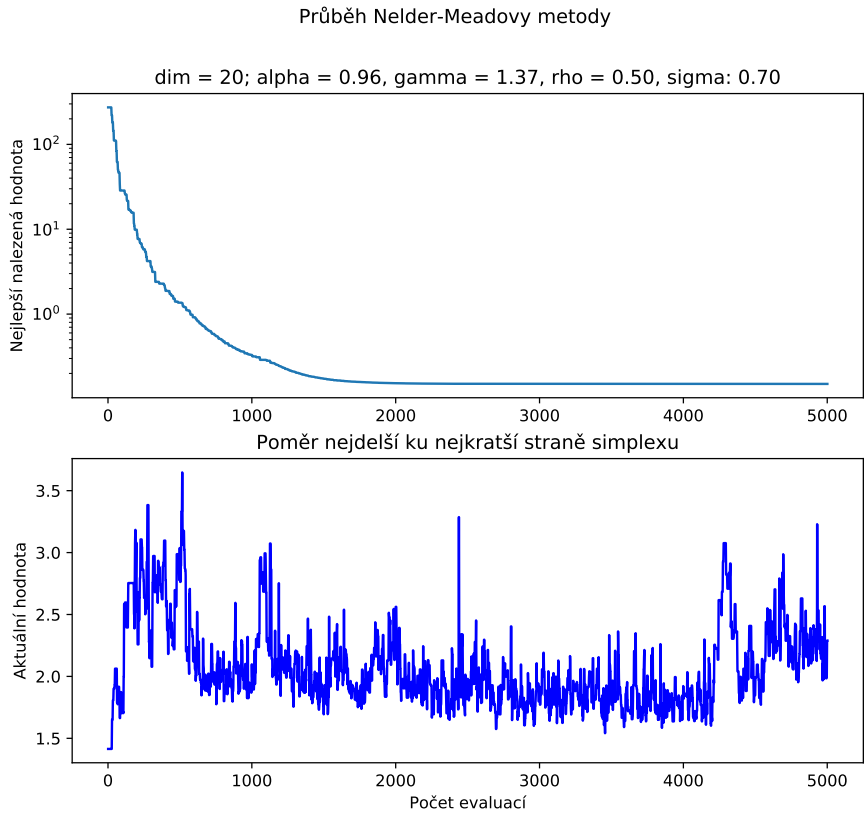
Abychom byli schopni detekovat podezřelé tvary simplexů, potřebujeme nějaký pozorovatelný údaj simplexu, který v průběhu optimalizace mění svou hodnotu, a na kterém půjde pozorovat, kdy simplex formuje výše zmíněné nevhodné tvary. V mém projektu jsem zaznamenával 2 takovéto údaje. Jedním z nich byl poměr délek hran simplexu a druhým poměr vlastních čísel kovarianční matice bodů simplexu. Následující 2 podsekce podrobně rozebírají tyto diagnostické nástroje.

2.3.1 Poměr délek hran simplexu

Motivace k tomu, abych tento údaj ze simplexu zjišťoval byla následující. Pokud v průběhu optimalizace dojde k tomu, že aktuální simplex bude v nějakém směru extrémně protáhlý, pak tento jednorozměrný údaj dokáže takovýto špatný tvar simplexu detekovat a mohu ho poté upravit do přijatelnější podoby. Tento údaj, který označím r_1, spočítám jako poměr nejdelší hrany simplexu ku nejkratší. Tento nástroj ovšem pro můj projekt nebyl příliš užitečný. Ukázalo se totiž, že rychlost a způsob konvergence Nelder-Meadovy metody mají minimální (nebo spíše žádný) vliv na hodnotu údaje r_1. Obrázky 2.3 a 2.4 zachytávají průběhy Nelder-Meadovy metody při hledání minim 2 náhodných 20-dimenzionálních kvadratických funkcí (hodnota hledaného minima obou funkcí je 0) spolu s údajem r_1, který se počítá po každé funkční evaluaci. První obrázek zachytává plynulý průběh, kdy metoda dokonvergovala velmi blízko k hledanému minimu. Druhý obrázek zachytává průběh, ve kterém došlo k extrémnímu zpomalení konvergence metody, pravděpodobně vlivem nevhodného tvaru simplexu. Pokud se podíváme na grafy poměrů délek hran simplexů u obou průběhů, tak zjistíme, že se hodnoty v grafech r_1 údajů pohybují pořád kolem stejných hodnot a nedochází k žádným významnějším změnám ani výkyvům.



Obrázek 2.3. Průběh Nelder-Meadovy metody spolu s poměrem délek hran simplexu - bezproblémová konvergence směrem k minimu ($f(x^{\text{opt}}) = 0$)



Obrázek 2.4. Průběh Nelder-Meadovy metody spolu s poměrem délek hran simplexu - zpomalená konvergence směrem k minimu ($f(x^{\text{opt}}) = 0$)

Ukázalo se, že hodnota r_1 nemá s konvergencí Nelder-Meadovy metody žádnou souvislost a dále jsem s tímto nástrojem v mém projektu nepracoval.

2.3.2 Poměr vlastních čísel kovarianční matice bodů simplexu

Tento údaj, který budu značit r_2, mi přinesl mnohem užitečnější pohled na tvar simplexu, než údaj r_1 z minulé podsekce. Chtěl jsem vytvořit nějaký nástroj, kterým bych mohl detekovat zploštění simplexu. Jako řešení se nabízelo využít vlastností vlastních čísel kovariančních matic [6–7].

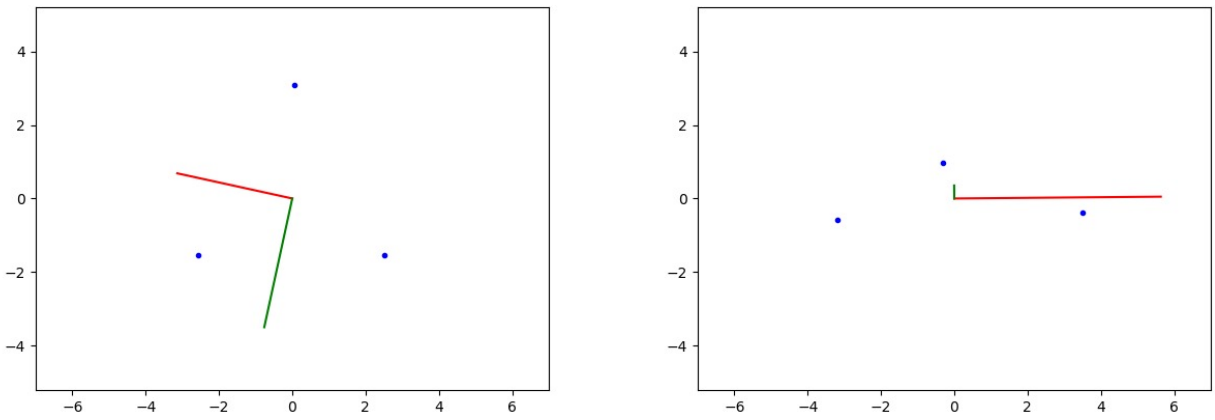
Nejdříve body vycentrujeme na počátek (odečteme od každého bodu centroid simplexu) a poté je zapíšeme do řádků pod sebe a vytvoříme tak matici, kterou označíme $\mathbf{X}$.

$$\mathbf{X} = \begin{bmatrix} x_{1_1} & x_{1_2} & \dots & x_{1_n} \\ x_{2_1} & x_{1_2} & \dots & x_{1_n} \\ \dots & \dots & \dots & \dots \\ x_{n+1_1} & x_{n+1_2} & \dots & x_{n+1_n} \end{bmatrix}$$

Kde body $x_1, ..., x_{n+1}$ jsou již vycentrované body simplexu a x_{i_j} je j-tá souřadnice i-tého bodu. Kovarianční matici spočítáme podle následujícího vzorce:

$$\mathbf{C} = \frac{\mathbf{X}^T\mathbf{X}}{n-1}$$

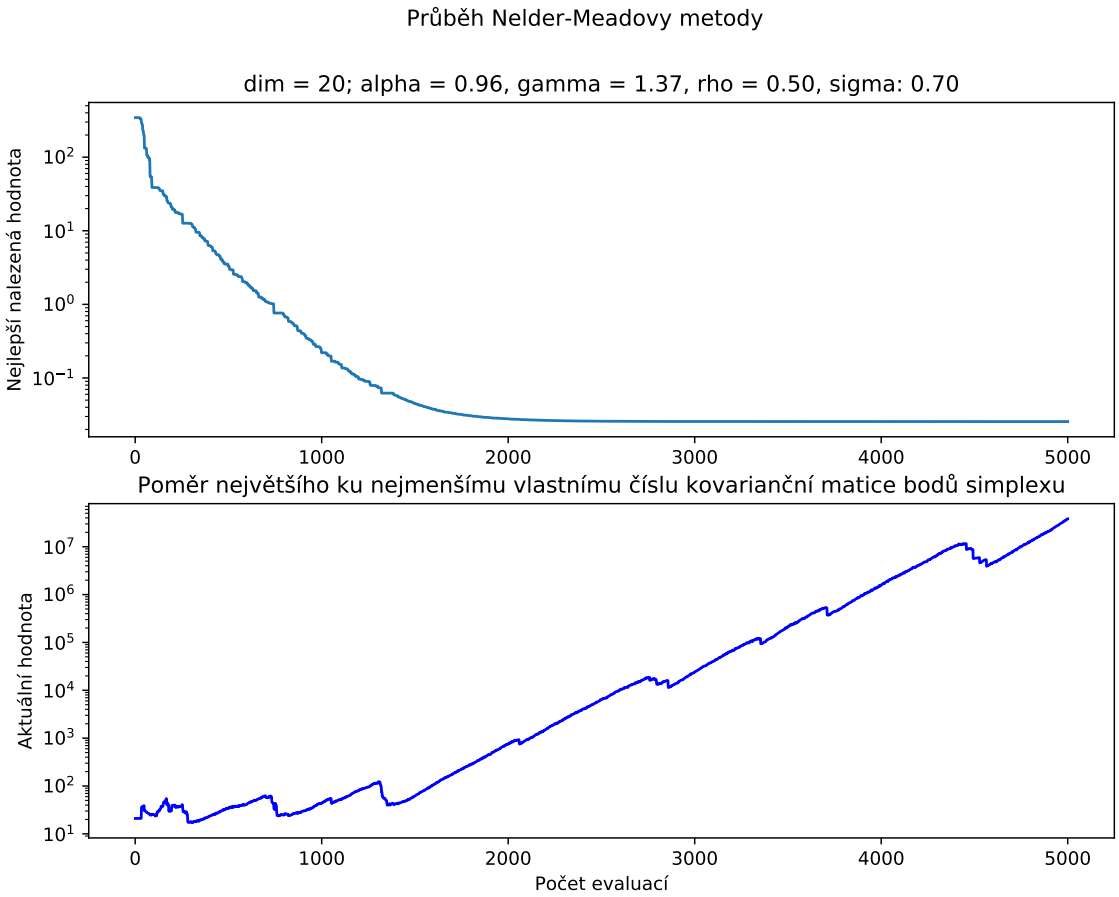
Kde $\mathbf{X}^T$ je transponovaná matice $\mathbf{X}$. Pak už jenom provedeme rozklad kovarianční matice $\mathbf{C}$ na matici vlastních vektorů a matici příslušných vlastních čísel. Pokud si nyní uděláme poměr největšího vlastního čísla matice $\mathbf{C}$ ku nejmenšímu, dostaneme hodnotu r_2. Hodnota údaje r_2 vyjadřuje jakýsi poměr mezi největší a nejmenší mírou rozptýlení bodů v navzájem ortogonálních směrech. Největší z vlastních čísel odpovídá vlastnímu vektoru, který určuje směr největšího rozptýlení bodů. Podobně vektor, kterému přísluší nejmenší z vlastních čísel matice $\mathbf{C}$, určuje směr nejmenšího rozptýlení bodů. Pokud dojde v průběhu optimalizace ke zploštění simplexu v nějakém směru, projeví se to na hodnotě nejmenšího vlastního čísla a tudíž i na hodnotě r_2. Obrázek 2.5 porovnává 2 tvary simplexů spolu s vlastními vektory odpovídajících kovariančních matic. Vlastní vektory jsou prodlouženy tak, aby poměr jejich vzdáleností odpovídal poměru hodnot příslušných vlastních čísel. Simplex v levém grafu je v pořádku a lze si všimnout, že zobrazené vlastní vektory jsou přibližně stejně dlouhé. Simplex v pravém grafu je zploštěný a proto se délky jednotlivých vektorů velmi liší.



Obrázek 2.5. Vlastní vektory kovariančních matic bodů simplexů

Údaj r_2 jsem ve svém projektu zaznamenával u průběhů metody častokrát. Ukázalo se totiž, že u optimalizace kvadratických funkcí jsou zpomalení v konvergenci metody způsobena převážně zploštěním simplexu.

Obrázek 2.6 zobrazuje graf průběhu metody spolu s grafem r_2 údaje, který byl zaznamenáván po každé funkční evaluaci. Optimalizuje se 20-dimenzionální kvadratická funkce, která má ve svém minimu hodnotu 0. V grafu průběhu si lze všimnout, že došlo ke zpomalení konvergence metody a metoda se tak ani po 5000 funkčních evaluacích nedostala k hledanému minimu příliš blízko. Hodnota r_2 údaje od přibližně 1500. funkční evaluace začíná prudce stoupat. Tento nárůst naznačuje, že jakmile se simplex dostane do zploštěného tvaru, tak každou další iterací metody se tvar simplexu ještě více deformuje do ploché podoby a zpomaluje se tak průběh metody.



Obrázek 2.6. Průběh Nelder-Meadovy metody spolu s poměrem vlastních čísel kovarianční matice bodů simplexu

2.4 Ukončení metody

K ukončení metody lze použít několik různých strategií [4–5, 8]. Jednou z nich je stanovení si pevného počtu iterací a po dosažení této hranice vrátit nejlepší nalezené řešení. Nevýhodou této strategie je, že předem nevíme kolik je třeba provést funkčních evaluací, a také nemáme nijak zaručeno že nalezené řešení bude dostatečně optimální.

Další způsob ukončení průběhu metody je sledování velikosti simplexu. Tato strategie spočívá v tom, že si předem stanovíme nějakou dostatečně malou velikost d a jakmile

bude platit, že nejdelší hrana simplexu bude kratší než d, tak ukončíme metodu. Tento způsob je dobré používat v kombinaci se strategií pevného počtu iterací. Může se totiž stát, že se simplex dostane do takového tvaru, že se délka nejdelší strany simplexu nikdy nedostane pod předem stanovenou mez a způsobí to nekonečné zacyklení metody.

Asi nejpoužívanější strategií k ukončování metody je porovnávání nejlepší a nejhorší funkční hodnoty účelové funkce v bodech simplexu. Tento způsob spočívá v tom, že si zvolíme nějakou mez ε a metodu ukončíme, jakmile bude platit tato nerovnost:

$$|f(x_{n+1}) - f(x_1)| < \varepsilon$$

Bod x_1 je nejlepší bod simplexu, x_{n+1} je nejhorší. Tuto strategii je ze stejných důvodů jako předchozí strategii vhodné používat v kombinaci s maximálním počtem iterací metody.

Poslední zde zmíněný způsob jak ukončovat metodu je zvolit si předem maximální počet funkčních evaluací. Poté co je tento limit překročen se metoda ukončí. Oproti způsobu s pevným počtem iterací předem víme, kolik funkčních evaluací budeme v rámci optimalizace potřebovat. Tento způsob ukončování metody jsem si ve svém projektu zvolil jako hlavní strategii a to převážně z důvodů ladění - velmi jednoduše se porovnává efektivita jednotlivých běhů metody.

Kapitola 3
Ladění Nelder-Meadovy metody

Konečně se dostáváme k hlavnímu tématu této práce. Tím je ladění Nelder-Meadovy simplexové metody tak, aby její průběh na vybraných funkcích byl co nejefektivnější. Ladit a vylepšovat metodu lze několika způsoby. Jedním z nich je pokus o nalezení co nejlepších hodnot parametrů α, γ, ρ, σ - takových, s kterými metoda dosahuje nejlepších výsledků.

Původně jsem chtěl k hledání parametrů využít nějaký evoluční algoritmus. V průběhu práce jsem ovšem od této myšlenky ustoupil a začal jsem přemýšlet nad optimalizačním přístupem jak hledat optimální parametry metody. Dospěl jsem k závěru, že k hledání těchto parametru mohu použít onu samotnou Nelder-Meadovu metodu.

3.1 Ladění parametrů pomocí Nelder-Meadovy metody

Nelder-Meadova simplexová metoda slouží k hledání lokálních minim funkcí více proměnných. Pokud si vytvoříme nějakou vyhodnocovací funkci $\mathcal{F}$, která přijímá 4 hledané parametry α, γ, ρ, σ a která vrací 1 číslo udávající kvalitu (nebo spíše míru kvality) metody, pak můžeme tuto funkci optimalizovat právě pomocí Nelder-Meadovy metody a získat tak optimální hodnoty parametrů.

Narážíme zde ovšem na 2 problémy. Tím prvním je, že nemáme k dispozici žádné přesné vyčíslení kvality metody. Můžeme pouze zkonstruovat vyhodnocovací funkce, které přiřazují konkrétní čtveřici parametrů jedno číslo na základě nějakého pozorovatelného údaje (například počet potřebných evaluací nebo odchylka od hledaného minima). Druhým problémem je definiční obor parametrů α, γ, ρ, σ. Nelder-Meadova metoda se dá použít pouze k optimalizaci funkcí, jejichž definiční obor je celé $\mathbb{R}^n$. Parametry α, γ, ρ, σ ovšem mají svá omezení, a jelikož je definičním oborem funkce $\mathcal{F}$ pouze část prostoru $\mathbb{R}^4$, je potřebné při každém volání funkce parametry vhodně transformovat.

3.1.1 Transformace prostoru do definičního oboru parametrů Nelder-Meadovy metody

Pokud chceme použít metodu k nalezení optimálních hodnot parametrů, musíme nejprve vytvořit zobrazení $\mathcal{T}$, které každému prvku z množiny $\mathbb{R}^4$ přiřadí 4-rozměrný vektor, jehož jednotlivé složky x'_1, x'_2, x'_3, x'_4 respektují omezení, která se vztahují na jednotlivé parametry Nelder-Meadovy metody. Zobrazení $\mathcal{T}$ musí být spojité, jinak by metoda nepracovala správně.

Já jsem zobrazení $\mathcal{T}$ vytvořil následovně. Pro parametr α musí platit $\alpha > 0$. k tomu abych spojitě přetransformoval první argument x_1 jsem použil exponenciální funkci e^x.

$$x'_1 = e^{x_1}$$

Hodnota parametru γ musí být větší než 1. Opět jsem k transformaci použil exponenciální funkci e^x, tentokrát s konstantním posunutím $+1$.

$$x'_2 = e^{x_2} + 1$$

Parametr ρ musí ležet v intervalu $\langle 0, \frac{1}{2} \rangle$. Třetí argument x_3 jsem transformoval pomocí logistické funkce (neboli sigmoidy).

$$x'_3 = \frac{1}{2} S(x_3)$$

Kde $S(x) = \frac{1}{1+e^{-x}}$ je sigmoida. Poslední argument zobrazení $\mathcal{T}$, tedy x_4, jsem opět transformoval pomocí sigmody, neboť parametr σ musí ležet v intervalu $\langle 0, 1 \rangle$.

$$x'_4 = S(x_4)$$

Výsledné zobrazení $\mathcal{T}$ vypadá takto:

$$\mathcal{T}(x_1, x_2, x_3, x_4) = \left(e^{x_1}, e^{x_2} + 1, \frac{1}{2} S(x_3), S(x_4) \right), \ (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$$

3.2 Benchmarkovací platforma COCO

Ve chvíli kdy budeme mít k dispozici Nelder-Meadovu metodu s konkrétním nastavením a budeme chtít zjistit, jak moc je metoda s tímto nastavením efektivní při použití k optimalizaci širšího spektra funkcí, bude se nám hodit nějaký systém, který dokáže metodu spolehlivě otestovat za nás. Přesně k těmto účelům se dá využít platforma COCO.

COCO (COmparing Continuous Optimisers) [9] je benchmarkovací platforma vytvořená za účelem automatizace opakujících se úloh spojených s benchmarkováním numerických optimalizačních algoritmů. Součástí balíku COCO je 24 různých vícerozměrných reálných funkcí, které se snažíme pomocí zvolených algoritmů a metod (dále budu tyto konkrétní instance metod označovat jako solvery) optimalizovat. Výsledky a průběh optimalizace jsou zaznamenávány spolu s dalšími informacemi do souborů, ze kterých je později možné vygenerovat různé užitečné grafy.

Některé funkce, které jsou součástí balíku COCO, jsou multimodální (mají více lokálních minim) a proto není vhodné tyto funkce optimalizovat pomocí základní neupravené Nelder-Meadovy metody. Z tohoto důvodu jsem neporovnával efektivitu Nelder-Mead solverů na základě výsledného souhrnu vytvořeného z průběhů optimalizace všech 24 funkcí, ale porovnával jsem pouze průběhy optimalizace u vybraných funkcí.

Kapitola 4
Ladění metody pro kvadratické funkce

V této kapitole čtenáři vysvětlím, jak jsem postupoval v mé práci při ladění Nelder-Meadovy metody za účelem dosažení co nejlepších výsledků při optimalizaci kvadratických funkcí více proměnných. Součástí kapitoly je shrnutí nejlepších nalezených solverů i porovnání jejich efektivity na jiných než kvadratických funkcích.

4.1 Inspirace

Předtím, než jsem začal sám pracovat na ladění Nelder-Meadovy metody, přečetl jsem si článek *Evolving a Nelder-Mead Algorithm for Optimization with Genetic Programming* [10] od Slovinských autorů Iztoka Fajfara, Janeze Puhana a Árpáda Bűrmena. Ti se v této práci snažili o vytvoření algoritmu k optimalizaci funkcí více proměnných, který vznikne evolucí z původní Nelder-Meadovy metody pomocí genetického programovaní. Byli úspěšní a opravdu se jim podařilo nalézt optimalizační solver, který na některých typech funkcí fungoval lépe, než původní Nelder-Meadova metoda.

Aby mohli autoři článku použít algoritmy genetického programování, potřebovali sestrojit vyhodnocovací účelovou funkci, která jim nějakým způsobem ohodnotí míru kvality jednotlivých optimalizačních solverů tak, aby je bylo možné mezi sebou porovnávat. Autoři potřebovali jednoduchou, ale zároveň dostatečně obecnou funkci. Proto si jako vyhodnocovací funkci zvolili průměrnou hodnotu z nalezených minim 10 náhodně posunutých 10-dimenzionálních kvadratických funkcí ve tvaru

$$f(x_1, x_2, ..., x_{10}) = \sum_{i=1}^{10} (x_i - d_i)^2$$

Kde d_i je náhodně zvolené posunutí i-tého členu. Hledaná optimální hodnota této účelové funkce je 0 a platí, že čím nižší hodnoty solver dosáhne, tím lépe.

Tento článek mě inspiroval natolik, že jsem se rozhodl, že začnu ladit Nelder-Meadovu metodu právě na kvadratických funkcích podobného tvaru.

4.2 Optimalizace parametrů

Vyhodnocovací účelovou funkci jsem vytvořil následujícím způsobem. Vstupem jsou 4 reálná čísla, která se pomocí zobrazení $\mathcal{T}$ (viz 3.1.1), transformují do jednotlivých parametrů α, γ, ρ, σ a vytvoří se solver Nelder-Meadovy metody s těmito parametry. Poté se provede 15 optimalizačních testů, kdy se optimalizují náhodně vygenerované 20-dimenzionální kvadratické funkce. Funkce jsou ve tvaru

$$f(x_1, x_2, ..., x_{20}) = \sum_{i=1}^{20} a_i(x_i - d_i)^2$$

Kde $d_i \in \langle -5, 5 \rangle$ je náhodně zvolený faktor posunutí i-tého členu a $a_i \in \langle 0.5, 3.5 \rangle$ je náhodně zvolený koeficient, který určuje strmost stoupání/klesání ve směru i-té osy

souřadnic. Každý z těchto 15 běhů metody má k dispozici 5000 funkčních evaluací. Na konci se vybere medián z nejlepších dosažených funkčních hodnot a vrátí se jako výsledek vyhodnocovací funkce. Původně jsem používal průměr z nalezených hodnot, ale nakonec jsem přešel k mediánu z toho důvodu, že pokud došlo u jedné z 15 optimalizovaných funkcí k mírnému vychýlení, tak se to silně projevilo na celkovém průměru a docházelo tak častěji ke zkreslení výsledné hodnoty.

Vyhodnocovací funkci jsem poté optimalizoval pomocí základní Nelder-Meadovy metody s parametry nastavenými na doporučené výchozí hodnoty ($\alpha = 1$, $\gamma = 2$, $\rho = \frac{1}{2}$, $\sigma = \frac{1}{2}$). Metoda měla k dispozici 1000 evaluací vyhodnocovací funkce.

Metodu jsem takto volal celkem pětkrát, pokaždé s jiným počátečním simplexem. Počáteční simplexy jsem konstruoval stejným způsobem jako jsem uvedl v sekci 2.2 na straně 5. Počáteční bod x_1 jsem volil v počátku a pouze jsem měnil hodnotu parametru c na hodnoty 1, 2, 3, 4 a 5.

Dostal jsem tedy celkem 5 různých 4-rozměrných vektorů, které po transformaci pomocí zobrazení $\mathcal{T}$ reprezentovaly jednotlivá nastavení parametrů Nelder-Meadovy metody.

$$1)\ \alpha = 0.971,\ \gamma = 1.131,\ \rho = 0.483,\ \sigma = 0.595$$

$$2)\ \alpha = 0.958,\ \gamma = 1.380,\ \rho = 0.499,\ \sigma = 0.552$$

$$3)\ \alpha = 0.956,\ \gamma = 1.373,\ \rho = 0.499,\ \sigma = 0.698$$

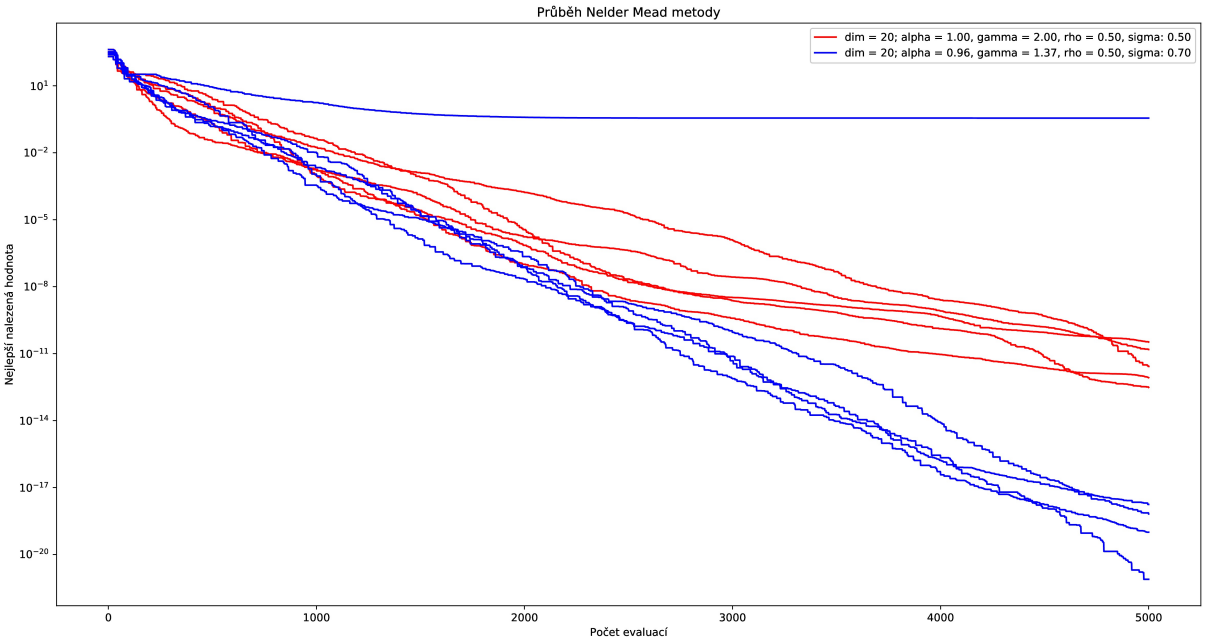
$$4)\ \alpha = 0.957,\ \gamma = 1.298,\ \rho = 0.498,\ \sigma = 0.973$$

$$5)\ \alpha = 0.962,\ \gamma = 1.325,\ \rho = 0.499,\ \sigma = 0.960$$

Hodnoty jsou zaokrouhlené na 3 desetinná místa. Z hodnot parametrů si lze všimnout, že i když byl počáteční simplex rozdílný, tak přesto Nelder-Meadova metoda zkonvergovala (kromě parametru σ) k velmi podobným hodnotám.

Nelder-Meadova metoda, která má své parametry nastavené na jednu z nalezených čtveřic parametrů, dosahuje při optimalizaci kvadratických funkcí vyšších dimenzí lepších výsledků, než metoda s výchozím nastavením. Nejlepších výsledků ze všech 5 čtveřic parametrů dosahuje třetí čtveřice, tedy $\alpha = 0.956$, $\gamma = 1.373$, $\rho = 0.499$, $\sigma = 0.698$. Následující rozbor srovnává výchozí nastavení s tímto nově získaným nastavením.

Aby byl dobře vidět postupný průběh optimalizace, zapisoval jsem si nejnižší dosaženou funkční hodnotu při každé funkční evaluaci během celého procesu optimalizování. Tímto způsobem jsem potom mohl lehce vizualizovat průběhy metody při optimalizaci různých funkcí. Graf na obrázku 4.1 zobrazuje průběhy 2 solverů při optimalizaci 5 náhodně vygenerovaných 20-dimenzionálních kvadratických funkcí. Červené křivky zobrazují průběhy Nelder-Meadovy metody s výchozími hodnotami parametrů a modré zobrazují průběhy metody s nově získaným nastavením. Oba solvery měly k dispozici 5000 evaluací na každou z optimalizovaných funkcí. Lze si všimnout, že většina modrých křivek dosáhla mnohem nižšího minima (rozdíl i 8 řádů), než jakého dosáhly křivky červené. Na graf se lze podívat i z jiného úhlu pohledu. Pokud bychom například hledali optimální řešení optimalizované funkce s povolenou tolerancí 10^{-10}, pak bychom potřebovali přibližně 4000 až 5000 funkčních evaluací, pokud bychom používali základní verzi Nelder-Meadovy metody. Pokud bychom použili metodu s vnitřními parametry nastavenými na nově nalezené hodnoty, pak by stačilo přibližně 2500 funkčních evaluací.



Obrázek 4.1. Průběhy optimalizace kvadratických funkcí pomocí Nelder-Meadovy metody s rozdílnými parametry

4.3 Reinicializace simplexu

Při optimalizaci kvadratických funkcí pomocí nově vzniklého solveru docházelo často extrémnímu zpomalení konvergence metody, nebo dokonce k úplnému zastavení metody, kdy další iterace už nenacházely nové lepší řešení. Na obrázku 4.1 je tento jev velmi dobře vidět. Jedna z modrých křivek není schopna konvergovat k nižším hodnotám. I po 5000 funkčních evaluací je nejlepší nalezená hodnota přibližně 10^{-1}. K tomuto jevu docházelo u nových solverů velmi často a bylo třeba přijít s nějakým řešením.

Ukázalo se, že pokud si u průběhu metody s novým nastavením necháme při každé funkční evaluaci zobrazit poměr vlastních čísel kovarianční matice bodů simplexu (2.3.2), tak lze vypozorovat, za jakých okolností dochází ke zpomalení konvergence metody. Pokud hodnota údaje r_2 (poměr vlastních čísel) přesáhne určitou mez, simplex se zploští a další iterace už nenalézají žádné lepší hodnoty (viz opět obrázek 2.6).

Jako řešení se nabízí reinicializovat simplex pokaždé, když se hodnota r_2 dostane nad určitou, předem zvolenou hodnotu. Simplex jsem se rozhodl reinicializovat podobně, jako když vytvářím počáteční simplex. Nejlepší bod (tedy bod s nejnižší funkční hodnotou) z původního simplexu zachovám a zvolím ho jako počáteční bod x_1 nového simplexu. Parametr c, který potřebuji k vygenerování nového simplexu, zvolím jako průměrnou hodnotu ze vzdáleností jednotlivých bodů původního simplexu od bodu x_1.

$$c = \frac{1}{n} \sum_{i=2}^{n+1} d(x_1, x_i)$$

Kde $d(A, B)$ je vzdálenost bodu A od bodu B. Nově vzniklý simplex ještě otočím kolem bodu x_1 pomocí náhodně vygenerované rotační ortogonální matice $\mathcal{R}$ (odečtu od všech

bodů simplexu bod x_1, provedu lineární transformaci pomocí $\mathcal{R}$ a opět k bodům přičtu x_1). Tento krok s náhodným otočením jsem do reinicializace zahrnul z toho důvodu, že se mi zdálo užitečné měnit v průběhu optimalizace orientaci simplexu.

Upravil jsem Nelder-Meadovu metodu tak, že jsem ji přidal pátý parametr R a přidal jsem na začátek iterace reinicializační podmínku. Pokud je poměr vlastních čísel r_2 větší než R, tak se provede reinicializace simplexu a pokračuje se dál v algoritmu. Výpočet hodnoty r_2 je výpočetně náročný a proto tuto reinicializační podmínku kontroluji každých 10 iterací. Na výsledný průběh metody to má minimální vliv, ale metoda se tím znatelně zrychlí.

Zkoušel jsem nastavovat parametr R na různé hodnoty z intervalu $\langle 20, 2000 \rangle$. Zjistil jsem, že nejlepších výsledků při optimalizaci kvadratických funkcí metoda dosahuje, pokud se parametr R nastaví na nižší hodnoty (50 až 200). To ale znamená, že k reinicializaci simplexu dochází velmi často. Vysvětluji si to tím, že lépe tvarovaný simplex může dělat efektivnější kroky a tudíž metoda konverguje rychleji. Častá reinicializace simplexu velmi zlepšila efektivitu metody i u verze s výchozími hodnotami parametrů.

4.4 Optimalizace metody spolu s reinicializačním parametrem R

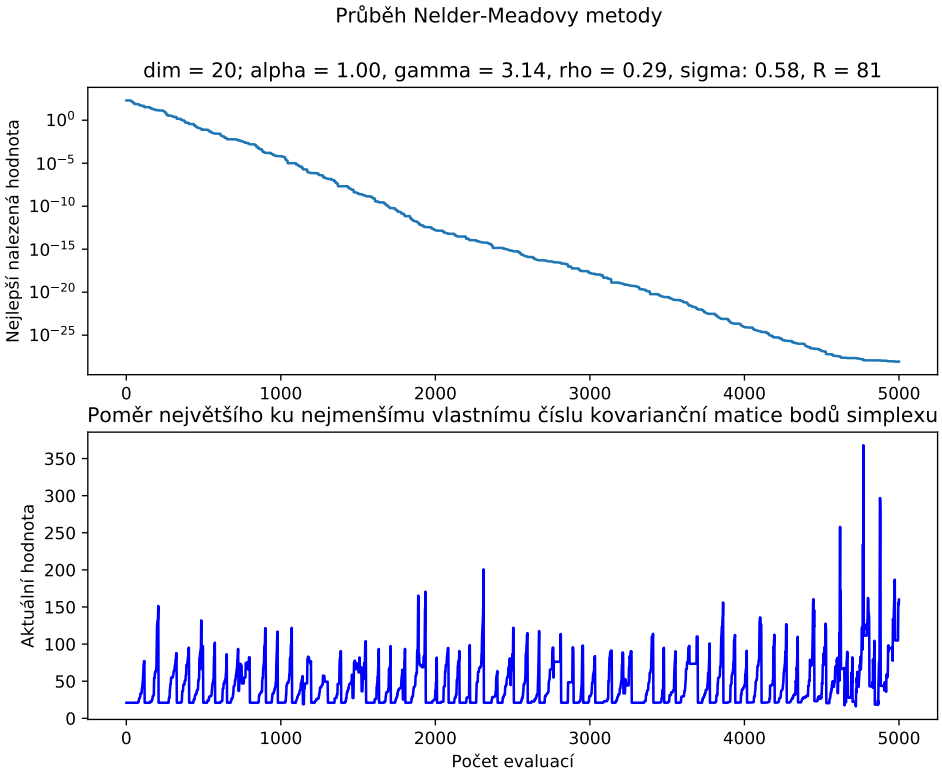
Nabízí se možnost optimalizovat parametr R spolu s ostatními parametry Nelder-Meadovy metody. Pokud si upravíme vyhodnocovací účelovou funkci tak, aby přijímala 5 reálných čísel s tím, že poslední argument funkce odpovídá parametru R, můžeme optimalizovat takto vzniklou funkci podobně, jako jsem uvedl v sekci 4.2. Pátý argument účelové funkce je vhodné transformovat funkcí $f(x) = e^x + 20$, protože jsem experimentálně zjistil, že hodnoty R menší než 20 nemají příliš smysl. Když k reinicializaci dochází příliš často, tak metoda neprodukuje žádné smysluplné výsledky.

Provedl jsem celkem 3 optimalizační běhy základní Nelder-Meadovy metody nad výše zmíněnou upravenou účelovou funkcí. Pokaždé s jiným počátečním simplexem. Metoda vždy měla k dispozici 1000 funkčních evaluací účelové funkce. Účelová funkce vracela medián z 15 nalezených minim náhodně vygenerovaných 20-dimenzionální kvadratických funkcí. Solvery (metody s novým nastavením parametrů), které hledaly minima těchto kvadratických funkcí, měly k dispozici 5000 evaluací pro každou z 15 funkcí.

Dále jsem provedl 3 identické běhy, s tím rozdílem, že jsem zafixoval parametr α pevně na hodnotu 1 (optimalizoval jsem tedy 4-rozměrnou účelovou funkci). Různými experimenty a testy jsem totiž zjistil, že u solverů, které mají parametr α nastavený na 1, téměř nedochází k zaseknutí v konvergenci a tyto solvery dosahují průměrně lepších výsledků. Z 6 nalezených solverů dosahoval nejlepších výsledků při optimalizaci kvadratických funkcí následující:

$$\alpha = 1,\ \gamma = 3.13,\ \rho = 0.28,\ \sigma = 0.57,\ R = 81.85$$

Graf na obrázku 4.2 zobrazuje průběh Nelder-Meadovy metody, jejíž vnitřní parametry a reinicializační parametr R jsou nastaveny na nově nalezené hodnoty. Hledá se zde minimum náhodně vygenerované 20-dimenzionální kvadratické funkce. Obrázek také zobrazuje vývoj hodnoty poměru největšího ku nejmenšímu vlastnímu číslu kovarianční matice bodů simplexu – r_2. Z grafu r_2 lze lehce vypozorovat, kdy dochází v průběhu metody k reinicializaci simplexu. Ke konci průběhu metody se hodnota poměru vlastních čísel velmi odchýlí od průměrných hodnot v grafu. To je způsobeno tím, že ve svém programu používám 64 bitové floaty pro vyjadřování reálných čísel. Během optimalizace



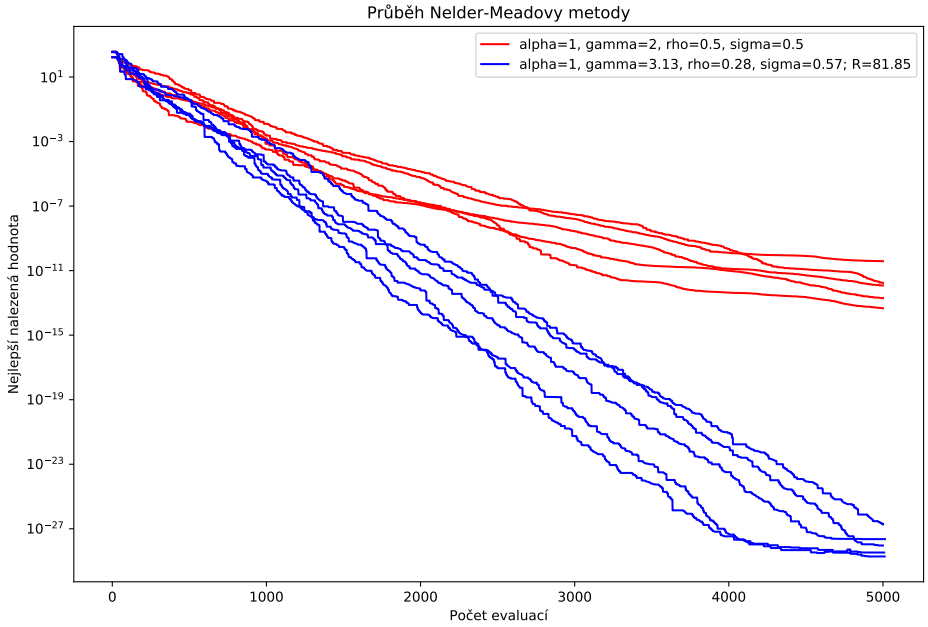
Obrázek 4.2. Průběh optimalizace kvadratické funkce pomocí Nelder-Meadovy metody s novými parametry a reinicializační technikou

se může stát, že se simplex zmenší a vzdálenost jednotlivých bodů od sebe bude velmi malá ($< 10^{-15}$). To způsobí zaokrouhlovací chyby v aritmetických výpočtech s používanými 64 bitovými floaty. Hodnota poměru vlastních čísel r_2 tedy přestane mít jakoukoli vypovídající hodnotu a reinicializace simplexu se stane nemožnou.

Obrázek 4.3 porovnává průběhy optimalizace 5 různých kvadratických funkcí. Červené křivky zobrazují průběhy Nelder-Meadovy metody s výchozím nastavením bez reinicializace simplexů a modré zobrazují průběhy metody s parametry nastavenými na $\alpha = 1,\ \gamma = 3.13,\ \rho = 0.28,\ \sigma = 0.57$. Reinicializačním parametrem R je nastavený na hodnotu 81.85. Rozdíl mezi dosaženými výsledky je opravdu znatelný. Reinicializační technika spolu s vhodně zvolenými parametry velmi zvyšují efektivitu Nelder-Meadovy metody, tedy alespoň při optimalizaci kvadratických funkcí vyšších dimenzí.

4.5 Periodická reinicializace simplexu

Nelder-Meadova metoda s použitím reinicializační techniky dosahuje lepších výsledků, než základní neupravená verze metody. Jediný problém této techniky spočívá v tom, že celý princip reinicializace závisí pouze na hodnotě poměru vlastních čísel kovarianční matice bodů simplexu. To může být hned ze dvou důvodů nepříjemná překážka při použití metody. První důvod je nevyhnutelné zpomalení metody. Spočítání údaje r_2 je časově náročné, protože je třeba hledat vlastní čísla matice, která může mít i větší rozměry (záleží na dimenzi optimalizované funkce). Jako řešení se nabízí provádět tuto kontrolu ne každou iteraci, ale tak jak jsem to používal já v 4.3, například jednou za 10 iterací. Tento způsob testování simplexu velmi zrychlí průběh metody a pokles v efek-



Obrázek 4.3. Porovnání průběhů optimalizace kvadratických funkcí pomocí Nelder-Meadovy metody s rozdílným nastavením

tivitě metody není tolik znatelný. Druhý důvod, proč reinicializování simplexu podle hodnoty r_2 může být problém, je, že různé funkce mohou vyžadovat jinak tvarované simplexy. Simplex, který by byl pro jednoduché kvadratické funkce příliš zploštělý (tedy došlo by k reinicializaci) by mohl být ideální simplex pro nějaký složitější typ funkcí.

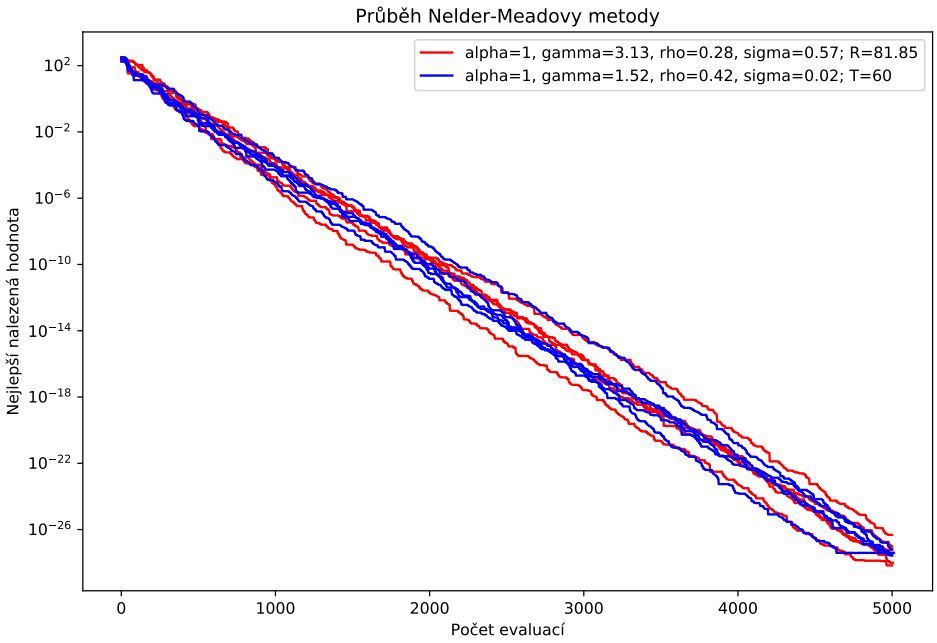
V článku *BBOB: Nelder-Mead with Resize and Halfruns* [11] se v jedné sekci píše o tom, jak autoři vybavili klasickou Nelder-Meadovu metodu funkcí, která jednou za 1000 iterací reinicializovala simplex téměř stejným způsobem jako jsem to dělal já v 4.3. Autoři v článku tvrdí, že takto upravená metoda dosahovala znatelně lepších výsledků při optimalizace funkcí s vyšším počtem proměnných, než metoda bez reinicializací.

Já jsem se rozhodl metodu upravit tak, že jsem ji přidal parametr T, který určuje, po kolika iteracích má dojít k reinicializaci simplexu. Reinicializace probíhá stejným způsobem jako jsem uvedl v 4.3. Takto upravená metoda je připravena k optimalizaci, podobně jako Nelder-Meadova metoda s parametrem R. Účelová funkce, kterou budu optimalizovat, vypadá podobně jako účelová funkce v 4.4 s tím rozdílem, že poslední argument transformuji pomocí funkce $f(x) = \lfloor e^x + 5 \rfloor$ (pouze přirozená čísla větší nebo rovno 5). Reinicializace častěji než jednou za 5 iterací mi přijde zbytečně častá. Účelovou funkci opět optimalizuji pomocí základní Nelder-Meadovy metody s výchozími hodnotami vnitřních parametrů.

Podobně jako při optimalizaci metody s parametrem R jsem provedl i teď celkem 6 optimalizačních běhů – 3 s parametrem α nastaveným na hodnotu 1 a 3 s libovolnou hodnotou α. Pokaždé jsem začínal s jinak velikým počátečním simplexem. Nejlepších výsledků dosahoval následující solver:

$$\alpha = 1,\ \gamma = 1.52,\ \rho = 0.42,\ \sigma = 0.02,\ T = 60$$

Graf na obrázku 4.4 porovnává efektivitu tohoto nově nalezeného solveru (modré křivky) se solverem nalezeným v sekci 4.4 (červené křivky). Optimalizovalo se celkem



Obrázek 4.4. Porovnání průběhů optimalizace kvadratických funkcí pomocí Nelder-Meadovy metody s rozdílným nastavením

5 různých kvadratických 20-dimenzionálních funkcí. Lze si všimnout, že solvery dosahují podobných výsledků. Můžeme tedy bez ztráty efektivity nahradit reinicializaci založenou na hodnotě údaje r_2 za reinicializaci periodickou.

4.6 Optimalizovaní metody pro další dimenze

Do teď jsem ladil metodu pouze na kvadratických funkcích dimenze 20. Pro úplnost jsem provedl podobné optimalizační pokusy jako v předešlých sekcích pro následující dimenze: 2, 5 a 10. Přizpůsobil jsem počet evaluací povolených k optimalizaci tak, aby se solvery při optimalizaci dostaly k podobným hodnotám jako při optimalizaci 20-dimenzionálních funkcí. Počet povolených evaluací pro dimenze 2, 5 a 10 je 100, 350 a 1000.

Provedl jsem všechny způsoby ladění jako při ladění metody pro 20-dimenzionální funkce. Zahrnul jsem i ladění metody s parametrem R a parametrem T. Abych zpřehlednil formát prezentace dosažených výsledků a nalezených solverů, rozhodl jsem se že budu data zobrazovat v jednoduchých tabulkách. Tabulka 4.1 zobrazuje nejlepší nalezené solvery specializované na kvadratické funkce pro jednotlivé dimenze.

Tabulka 4.1. Nejlepší nastavení parametrů pro konkrétní dimenzi.

Solver	Dimenze	α	γ	ρ	σ	T
1.	2	1	2.01	0.27	0.14	170
2.	5	0.95	2.34	0.14	0.56	13
3.	10	1	2.11	0.04	0.88	27
4.	20	1	1.52	0.42	0.02	60

Nejlepší solver pro každou dimenzi byl vždy jeden z těch, který využíval periodickou reinicializaci simplexu. Zajímavé je, že nejlepší nalezený solver pro 2-dimenzionální

funkce má parametr T nastavený na hodnotu 170, přičemž pro tuto dimenzi měl povoleno pouze 100 funkčních evaluací, tedy k reinicializaci nikdy nedošlo. První solver se tedy dá chápat jako metoda bez reinicializace, pouze se změněnými parametry. Hodnoty parametrů jednotlivých solverů jsou velmi odlišné a nedá se tvrdit, že by optimalizace metody napříč různými dimenzemi vedla k podobným solverům. Jediný společnou vlastnost, kterou jsem z hodnot parametrů nalezených solverů dokázal vypozorovat, je, že Nelder-Meadova metoda s parametrem α nastaveným na hodnotu 1, nebo velmi blízko 1, dosahuje znatelně lepších výsledků, než metoda s parametrem α nastaveným volněji.

Tabulka 4.2 prezentuje jednoduchou statistiku nad výsledky, kterých dosáhly jednotlivé solvery z tabulky 4.1 při obsáhlém testu, při kterém se optimalizovalo 1000 náhodně vygenerovaných kvadratických funkcí příslušných dimenzí. Pro srovnání je v tabulce 4.3 zobrazena stejná statistika, s tím rozdílem, že se k optimalizaci funkcí používala základní Nelder-Meadova metoda s výchozím nastavením parametrů.

Tabulka 4.2. Jednoduchá statistika nad 1000 nejnižšími nalezenými funkčními hodnotami kvadratických funkcí optimalizovaných pomocí solverů z tabulky 4.1.

Dimenze	medián	průměr	minimum	maximum
2	8.0567^{-15}	3.9428^{-11}	1.4166^{-20}	1.1845^{-08}
5	9.5278^{-14}	1.2651^{-10}	1.0343^{-19}	1.8364^{-08}
10	1.0526^{-16}	1.0117^{-12}	5.4688^{-27}	6.4818^{-10}
20	1.5217^{-28}	1.1048^{-25}	1.3151^{-29}	7.2194^{-23}

Tabulka 4.3. Jednoduchá statistika nad 1000 nejnižšími nalezenými funkčními hodnotami kvadratických funkcí optimalizovaných pomocí základní verze Nelder-Meadovy metody.

Dimenze	medián	průměr	minimum	maximum
2	8.9932^{-12}	3.3371^{-11}	3.1594^{-14}	4.8531^{-09}
5	1.1083^{-12}	3.2531^{-11}	7.8690^{-16}	5.8569^{-09}
10	1.7958^{-11}	1.8556^{-09}	4.5547^{-16}	8.9220^{-07}
20	2.2609^{-12}	4.1351^{-10}	2.9678^{-18}	2.7934^{-07}

Z tabulek si lze všimnout, že velkého zlepšení oproti základní verzi metody dosahují nově nalezené solvery zejména při optimalizaci funkcí vyšších dimenzí (dimenze 10 a více). Pro funkce nižších dimenzí dokonce platí, že původní Nelder-Meadova metoda dosahuje v průměrné hodnotě a maximu lepších výsledků, než nové solvery.

Zkoušel jsem také použít čtvrtý solver z tabulky 4.1 k optimalizaci funkcí zbylých dimenzí. Použil jsem stejný způsob testování jako předtím, tedy jednoduchou statistiku nad nejnižšími nalezenými funkčními hodnotami 1000 optimalizovaných funkcí příslušných dimenzí. Ukázalo se, že tento solver nedosahuje lepších výsledků, než ostatní nalezené solvery pro zbylé dimenze. To ale vůbec není překvapivé. Každý solver z tabulky 4.1 byl laděný pro optimalizaci kvadratických funkcí konkrétní dimenze. Není tedy divu, že solver laděný speciálně pro optimalizaci 20-dimenzionálních funkcí dosahuje při optimalizaci 10-dimenzionálních funkcí horších výsledků, než solver laděný pro 10-dimenzionální funkce.

Poslední solver, který jsem testoval 1000 vygenerovanými funkcemi, byla Nelder-Meadova metoda s výchozím nastavením parametrů a reinicializačním parametrem T nastaveným na 100. Ta při optimalizaci 20-dimenzionálních funkcí nedosahovala tak přesných výsledků jako čtvrtý solver, který byl na 20-dimenzionální funkce specializovaný, ale při optimalizaci funkcí zbylých dimenzí dosahovala solidních výsledků a zároveň zachovává některé užitečné vlastnosti původní neupravené metody při optimalizaci jiných než kvadratických funkcí (viz následující sekce).

4.7 Benchmarkování solverů pomocí platformy COCO

Podařilo se mi nalézt takové hodnoty vnitřních parametrů Nelder-Meadovy metody, se kterými metoda v kombinaci s periodickou reinicializací simplexu dosahuje znatelně lepších výsledků při optimalizaci kvadratických funkcí dimenzí 10 a výše. Nemám ale vůbec žádné informace o tom, jak se solverům daří při optimalizaci i jiných než kvadratických funkcí.

K tomu, abych porovnal efektivitu nových solverů s původní verzí metody, použiji benchmarkovací platformu COCO (viz sekce 3.2). Platformu využívám tak, že jsem si vytvořil jednoduchý skript, kterému dodám konkrétní solver a poté zavolám připravenou funkci, která už sama otestuje efektivitu dodaného solveru na všech 24 testovacích funkcích, které jsou součástí sady COCO. Průběh optimalizace se ukládá do lokálních souborů, ze kterých pak lze velmi jednoduchým způsobem vygenerovat velké množství různých grafů popisujících efektivitu solveru při optimalizaci konkrétního typu funkcí.

Pomocí COCO benchmarku jsem otestoval následující solvery:

1. Nelder-Meadova metoda s výchozím nastavením vnitřních parametrů

$$\alpha = 1,\ \gamma = 2,\ \rho = \frac{1}{2},\ \sigma = \frac{1}{2}$$

2. Nelder-Meadova metoda s výchozím nastavením vnitřních parametrů a reinicializační technikou aplikovanou jednou za 100 iterací metody.

$$\alpha = 1,\ \gamma = 2,\ \rho = \frac{1}{2},\ \sigma = \frac{1}{2},\ T = 100$$

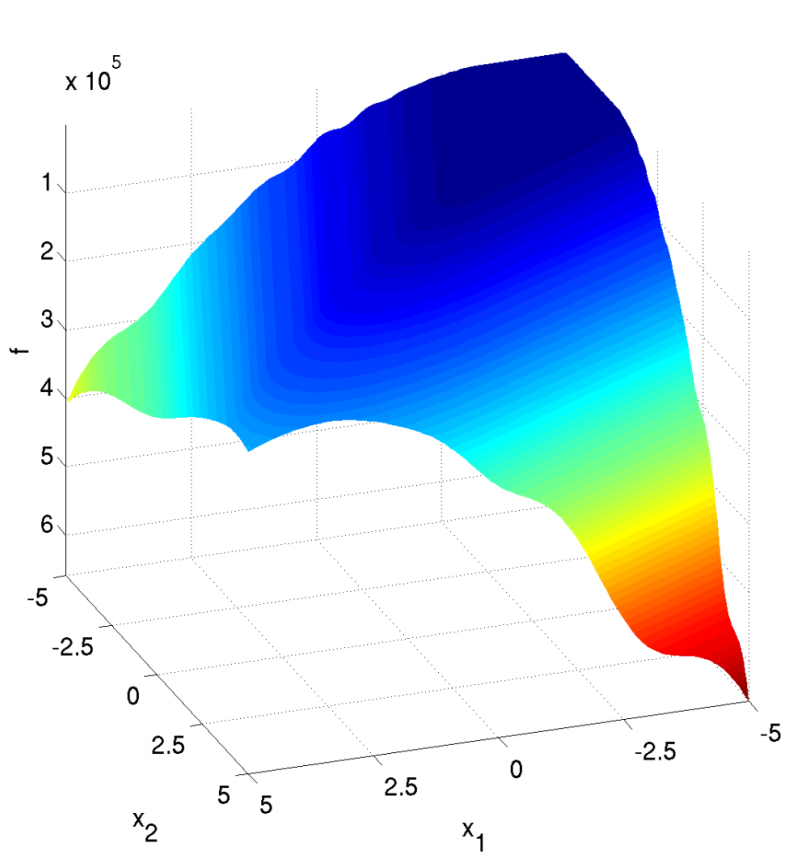
3. Nalezený solver, který dosahoval nejlepších výsledků při optimalizaci kvadratických funkcí dimenze 20

$$\alpha = 1,\ \gamma = 1.52,\ \rho = 0.42,\ \sigma = 0.02,\ T = 60$$

Všechny 3 solvery dosahovaly při optimalizaci funkcí podobných výsledků. Nejlépe si vedl pravděpodobně druhý solver, který kombinoval originální verzi metody s reinicializační technikou. Některé testovací funkce jsou multimodální, nebo nevhodné pro optimalizaci pomocí Nelder-Meadovy metody, a proto jsem porovnával efektivitu solverů pouze u vybraných funkcí.

První funkce, u které byly vyloženě znatelné rozdíly v průběhu a efektivitě optimalizace jednotlivými solvery, je funkce, která je v COCO sadě označována jako *Attractive Sector Function*. Jedná se o unimodální funkci s velmi asymetrickým tvarem. Graf její 2-rozměrné varianty můžete vidět na obrázku 4.5. Vrstevnice této funkce jsou zobrazeny na obrázku 4.6. Matematický předpis této funkce lze najít v dokumentaci jednotlivých funkcí sady COCO.

Funkce Attractive Sector je zajímavá zejména z toho důvodu, že optimalizace vícerozměrné verze této funkce pomocí základní metody s výchozími hodnotami parametrů nedosahuje uspokojivých výsledků, přestože další 2 testované solvery, které využívaly reinicializační techniku, dosahovaly na této funkci velmi dobrých výsledků. Obrázek 4.7 zobrazuje do jaké míry se defaultní Nelder-Meadova metoda přiblížila k hledanému minimu. Vodorovná osa reprezentuje počet provedených funkčních evaluací a svislá osa reprezentuje jaký podíl z celkového počtu jednotlivých cílů přesnosti (tolerance 100 až 10^{-8}) solver dosáhnul. V grafu je celkem 5 křivek. Každá z nich reprezentuje výsledky pro jinou dimenzi. Křížky, které se nacházejí na jednotlivých křivkách, zobrazují, ke



Obrázek 4.5. Graf 2-dimenzionální Attractive Sector Function

které nejlepší hodnotě se solver s pevně daným počtem evaluací dostal. Zbylá část křivek je predikce dalšího vývoje. Z grafu lze vidět, že defaultní Nelder-Meadova metoda si nedokázala poradit s Attractive Sector funkcí s vyšším počtem proměnných.

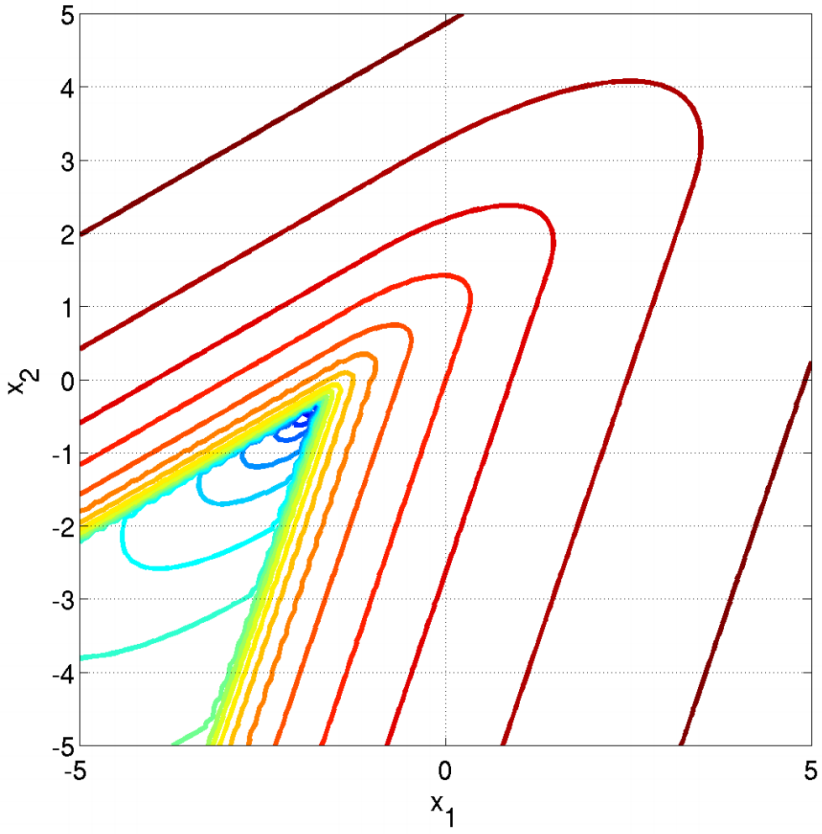
Graf na obrázku obrázku 4.8 zobrazuje efektivitu druhého solveru (defaultní metoda s reinicializací) při optimalizaci Attractive Sector funkci. Solver dokázal najít hledané minimum funkce s dostatečnou přesností pro všechny testované dimenze. Téměř identických výsledků při optimalizaci této funkce dosahoval i třetí testovaný solver. Z prezentovaných grafů efektivity lze usoudit, že periodická reinicializace velmi zlepšuje průběh a efektivitu Nelder-Meadovy metody při optimalizaci tohoto typu funkcí.

Periodická reinicilizace ovšem nepřináší pouze zlepšení metody. V jednom konkretním případě došlo i k velmi zásadnímu zhoršení oproti základní verzi metody. Při optimalizaci vícerozměrné verze Rosenbrockovy funkce (černé a červené křivky) dosahuje zakládní verze metody, tedy první z testovaných solverů nejlepších výsledků. Častá reinicializace je zde pravděpodobně nevhodně zvolené řešení, z důvodů nezvyklého tvaru Rosenbrockovy funkce. Grafy na obrázcích 4.9 a 4.10 zobrazují efektivitu použití prvního a druhého testovaného solveru k optimalizaci Rosenbrockovy funkce.

4.8 Shrnutí dosažených výsledků

Při ladění Nelder-Meadovy metody na kvadratických funkcích jsem dospěl k několika zajímavým řešením a závěrům.

Jednotlivé vnitřní parametry lze naladit na takové hodnoty, aby metoda konvergovala k minimu kvadratických funkcí rychleji než s původními hodnotami parametrů. Zlepšo-



Obrázek 4.6. Vrstevnice 2-dimenzionální Attractive Sector Function

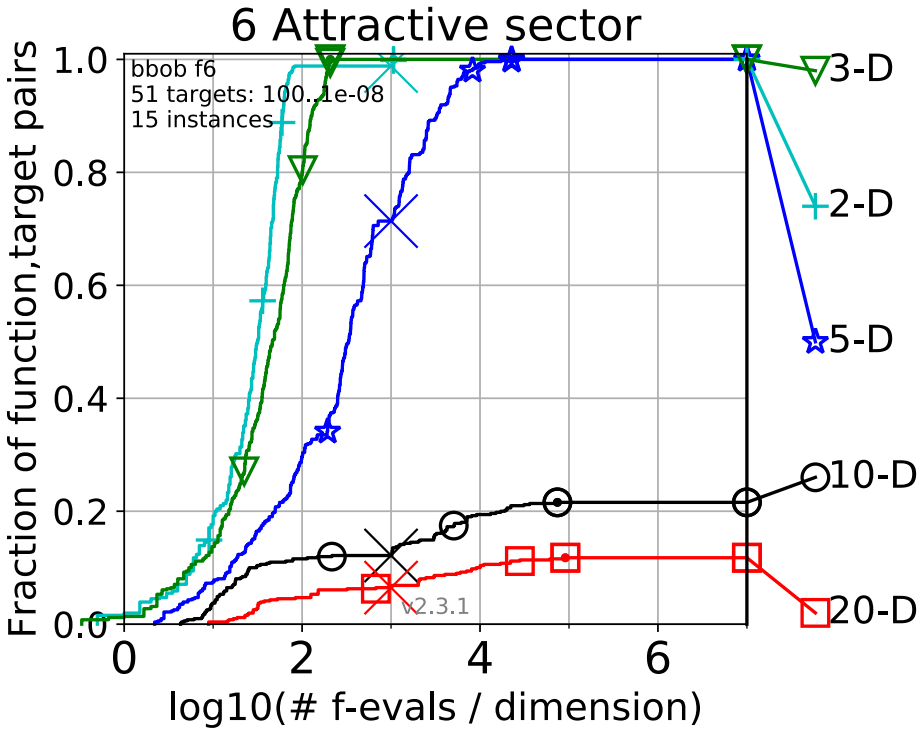
vání efektivity tímto způsobem dosahujeme především při optimalizaci funkcí dimenze 10 a více.

Pokud pouze upravíme hodnoty parametrů α, γ, ρ a σ, tak s velkou pravděpodobností narazíme na problém zplošťování simplexu a bude během vykonávání metody častěji docházet k problémům s konvergencí. To se dá v případě kvadratických funkcích vyřešit reinicializací simplexu ve chvíli, kdy k problému dochází. Častá reinicializace simplexu velmi znatelně zlepšuje průběh Nelder-Meadovy metody při optimalizaci kvadratických funkcí. Následující 2 nastavení vnitřních parametrů Nelder-Meadovy metody dosahují při optimalizaci kvadratických funkcí nejlepších výsledků:

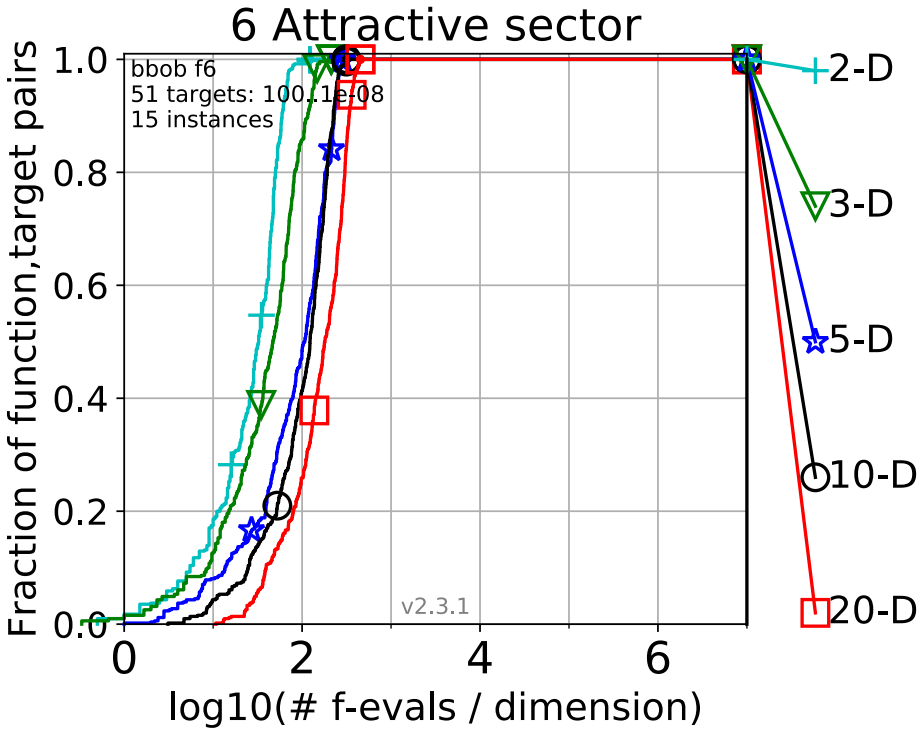
$$\alpha = 1,\ \gamma = 1.52,\ \rho = 0.42,\ \sigma = 0.02,\ T = 60$$

$$\alpha = 1,\ \gamma = 2,\ \rho = \frac{1}{2},\ \sigma = \frac{1}{2},\ T = 100$$

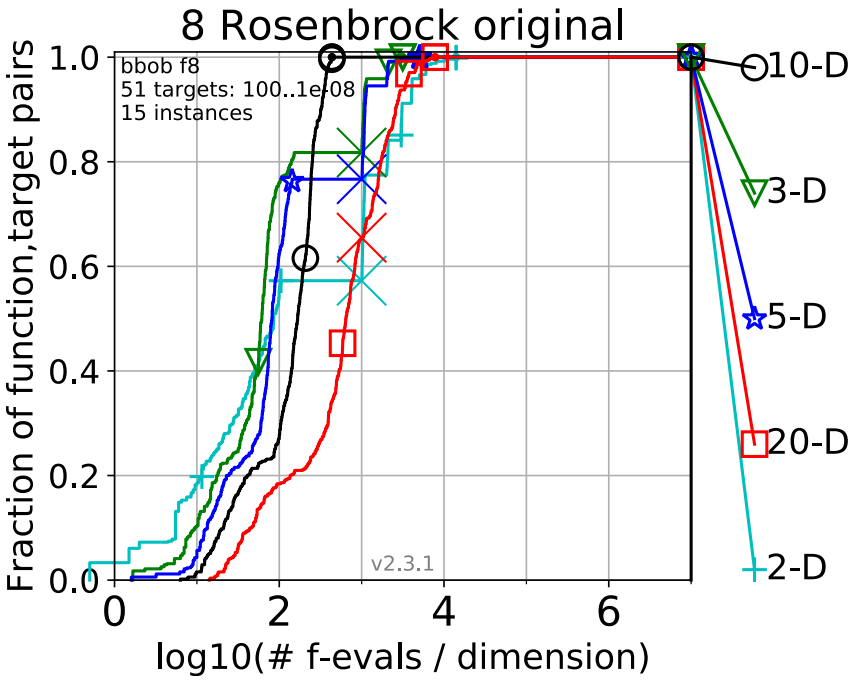
Kde parametr T určuje po kolika iteracích základního algoritmu metody má dojít k reinicializaci simplexu (způsobem popsaným v sekci 4.3). První z uvedených solverů je nejefektivnější úprava metody pro kvadratické funkce. Pro optimalizaci jiných než kvadratických funkcí už zmíněný solver není tak vhodný. Druhý solver je zakládní Nelder-Meadova metoda, u které se každých 100 iterací aplikuje reinicializace simplexu. Dosahuje podobných výsledků na kvadratických funkcích jako první solver, ale jak se ukázalo na benchmarkovací platformě COCO, je tato verze metody užitečnější při optimalizaci širšího spektra účelových funkcí.



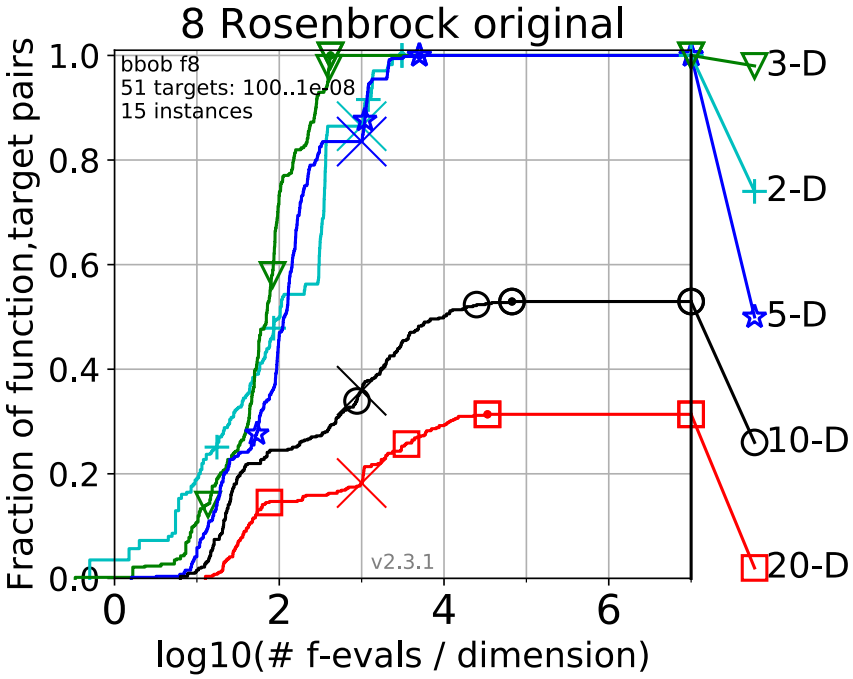
Obrázek 4.7. Efektivita základní Nelder-Meadovy metody při optimalizaci Attractive Sector Function



Obrázek 4.8. Efektivita základní Nelder-Meadovy metody s periodickou reinicializací při optimalizaci Attractive Sector Function



Obrázek 4.9. Efektivita základní Nelder-Meadovy metody při optimalizaci Rosenbrockovy funkce



Obrázek 4.10. Efektivita základní Nelder-Meadovy metody s periodickou reinicializací při optimalizaci Rosenbrockovy funkce

Kapitola 5
Ladění metody pro Rosenbrockovu funkci

V minulé kapitole jsem prozkoumal možnosti ladění Nelder-Meadovy metody na kvadratických funkcích. Solvery, ke kterým jsem se v průběhu ladění dopracoval, fungovaly na kvadratických funkcích velmi dobře a nacházely přesnější řešení, než metoda s původními hodnotami vnitřních parametrů. Ke zlepšení ovšem nedocházelo při optimalizaci všech testovaných funkcí. Jedna z funkcí, u které jsem si byl jistý tím, že k žádnému zlepšení oproti původní verzi Nelder-Meadovy metody nedochází, je vícerozměrná Rosenbrockova funkce. V této kapitole se pokusím o ladění Nelder-Meadovy metody speciálně pro tuto funkci.

5.1 Rosenbrockova funkce

Rosenbrockova funkce je známá matematická nekonvexní funkce, kterou v roce 1960 představil Howard H. Rosenbrock [12]. Tato funkce se často používá k testování optimalizačních algoritmů. Má tvar protáhlého parabolického údolí. Nalézt toto údolí je relativně jednoduché, ovšem najít přesnou lokaci globálního minima, které se v tomto údolí nalézá, je poněkud složitější. Protože má Rosenbrockova funkce takovýto specifický tvar, říká se jí často Rosenbrockovo údolí (*Rosenbrock's valley*) nebo také Rosenbrockova banánová funkce (*Rosenbrock's banana function*).

Rosenbrockova funkce je definována takto:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

Globální minimum se nachází v $(x, y) = (a, a^2)$ a funkční hodnota v tomto bodě je 0. Parametry a a b se nejčastěji nastavují na hodnoty $a = 1$ a $b = 100$. Graf na obrázku 5.1 zobrazuje Rosenbrockovu funkci s parametry nastavenými na hodnoty $a = 1$ a $b = 100$.

5.1.1 Zobecnění funkce pro vyšší dimenze

Rosenbrockova funkce lze také zobecnit i pro vyšší dimenze. V praxi se používají 2 způsoby zobecnění. Já jsem ve svých optimalizačních testech používal tu variantu, která je použita v platformě COCO. Stejně jako v COCO jsem i já tuto funkci upravil tak, aby její globální minimum nebylo vždy na stejném místě. Vždy když vygeneruji instanci této funkce, její globální minimum je posunuté nějakým vektorem.

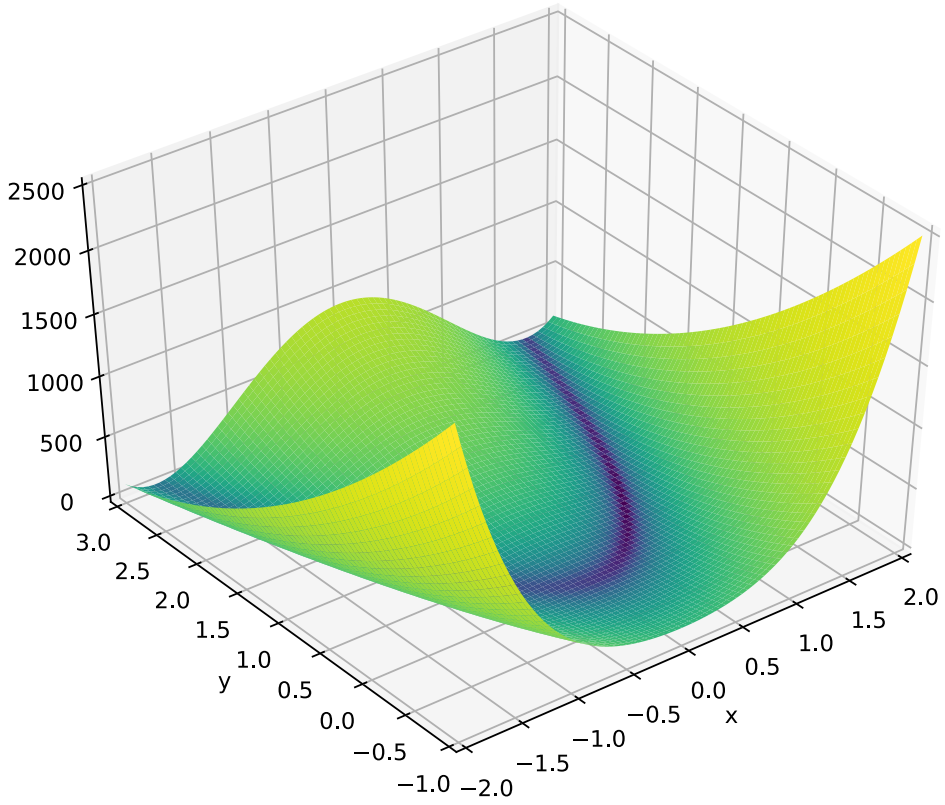
Předpis mnou použité verze Rosenbrockovy funkce je následující:

$$f(x) = \sum_{i=1}^{n-1} 100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2$$

Kde n je dimenze funkce, a vektor z je definován takto:

$$z = \max\left(1, \frac{\sqrt{n}}{8}\right)(x - x^{\text{opt}}) + (1, 1, ..., 1)$$

Vektor x^{opt} je hledané řešení. Při generování instance Rosenbrockovy funkce ho generuji náhodně tak, že jednotlivé složky vektoru ${x_i}^{\text{opt}}$, $i = 1, 2, 3, ..., n$ leží v předem zvoleném intervalu (zpravidla $\langle -5, 5 \rangle$).



Obrázek 5.1. Graf Rosenbrockovy funkce s parametry $a = 1$, $b = 100$.

5.2 Nalezení nejlepšího solveru

V této sekci čtenáři objasním, jakým způsobem jsem hledal nejlepší solver Nelder-Meadovy metody pro optimalizaci Rosenbrockovy funkce. Zaměřil jsem se pouze na 20-dimenzionální verzi této funkce a to zejména z toho důvodu, že pro nižší dimenze metoda fungovala dostatečně efektivně i s ne úplně optimálním nastavením (viz obrázek 4.10).

Solvery jsem se opět rozhodl hledat stejnými technikami jako při ladění metody pro kvadratické funkce. Nejdříve jsem se snažil najít solver pouhou optimalizací vnitřních parametrů pomocí neupravené Nelder-Meadovy metody. Účelovou funkci jsem zvolil jako medián z 10 nalezených minim náhodně posunutých Rosenbrockových funkcí. Nalezl jsem takto celkem 6 různých čtveřic parametrů. 3 s parametrem α nastaveným pevně na hodnotu 1 a 3 s parametrem α volným. Každý z těchto optimalizačních běhů jsem volal s 1000 evaluacemi účelové funkce a pokaždé s jinak velkým počátečním simplexem. Solvery, ke kterým jsem se optimalizací parametrů dopracoval, byly velmi rozdílné (velké rozdíly v hodnotách parametrů) a pouze 2 z nich dosahovaly při optimalizaci Rosenbrockovy funkce stejných nebo lepších výsledků, než Nelder-Meadova metoda s výchozími hodnotami parametrů. Dále jsem provedl podobných 6 optimalizačních pokusů s tím rozdílem, že jsem k optimalizovaným parametrům α, γ, ρ, σ přidal také parametr T určující frekvenci periodické reinicializace.

Ze všech získaných solverů si nejlépe při optimalizaci 20-dimenzionální Rosenbrockovy funkce vedl následující:

$$\alpha = 1,\ \gamma = 1.3739,\ \rho = 0.499,\ \sigma = 0.0485,\ T = 1316$$

Oproti solverům, které byly efektivní k optimalizaci kvadratických funkcí si lze všimnout, že hodnota parametru T tohoto nového solveru je velmi vysoká. To dává smysl, protože technika časté reinicializace simplexu při optimalizaci vícedimenzionální Rosenbrockovy funkce byla velmi neefektivní (viz 4.10).

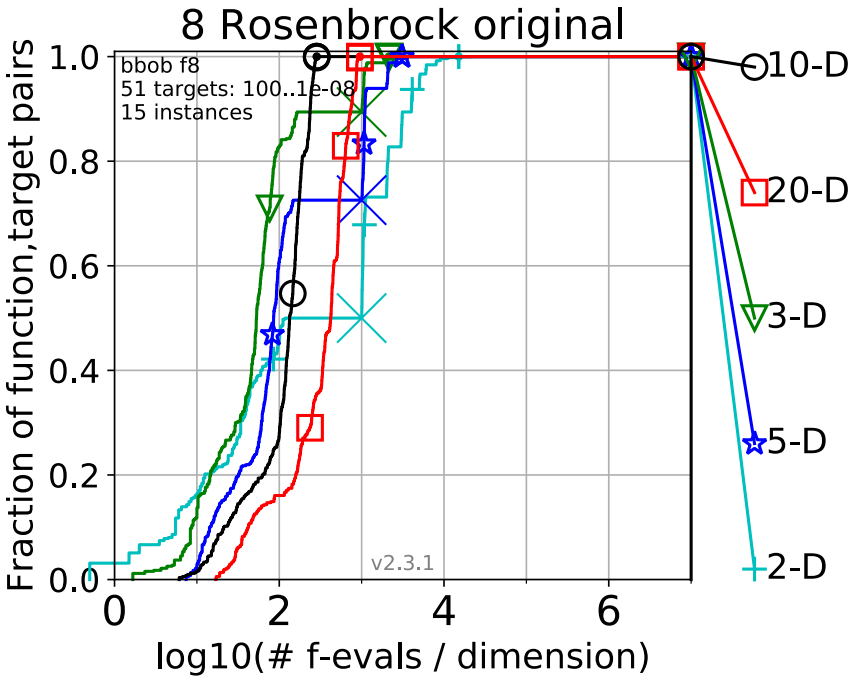
5.3 Porovnání s výchozí verzí Nelder-Meadovy metody

Provedl jsem několik obsáhlých testů, při kterých jsem zkoumal efektivitu nově nalezeného solveru v porovnání s originální Nelder-Meadovou metodou. Tabulka 5.1 porovnává efektivitu původní neupravené verze metody s nově nalezeným solverem při optimalizaci 1000 náhodně vygenerovaných Rosenbrockovo funkcí. Tabulka zobrazuje jednoduchou statistiku nad dosaženými výsledky. Pro každou z optimalizovaných funkcí měly solvery k dispozici 10000 funkčních evaluací. Je vidět, že nově nalezený solver dosahuje mnohem lepších výsledků než původní neupravená metoda.

Tabulka 5.1. Jednoduchá statistika nad 1000 nejnižšími nalezenými funkčními hodnotami náhodně posunutých Rosenbrockových funkcí optimalizovaných pomocí solveru z 5.2.

Solver	medián	průměr	minimum	maximum
Originál	1.8358^{-13}	5.7778^{-4}	2.4546^{-20}	3.9904^{-1}
Nový	1.9806^{-24}	9.1402^{-22}	1.0509^{-26}	8.1521^{-19}

Pro úplnost jsem otestoval nový solver i na platformě COCO. Tam si solver vedl velmi dobře nejen při optimalizaci Rosenbrockovy funkce. Ukázalo se, že v porovnání s ostatními solvery, které jsem testoval pomocí platformy COCO (viz 4.7), dosahuje tato nová Nelder-Meadova metoda nejlepších výsledků při optimalizaci většiny funkcí dimenze 10 a výše. Graf na obrázku 5.2 zobrazuje efektivitu nového solveru při optimalizaci Rosenbrockovy funkce. Oproti grafům 4.9 a 4.10 si lze všimnout zlepšení efektivity optimalizace funkcí dimenze 10 a 20 (černé a červené křivky).



Obrázek 5.2. Efektivita nově nalezeného solveru při optimalizaci Rosenbrockovy funkce v benchmarkovací platformě COCO.

Kapitola 6
Závěr

Z heuristických optimalizačních algoritmů jsem si k ladění vybral Nelder-Meadovu simplexovou metodu. Tu jsem implementoval tak, aby bylo možné optimalizovat její vnitřní parametry. Jako testovací funkce jsem používal náhodně vygenerované kvadratické funkce a poté i náhodně posunutou Rosenbrockovu funkci. Kvalitu a efektivitu metody při použití k optimalizaci konkrétního typu funkcí jsem určoval podle nejnižší nalezené hodnoty při pevně daném počtu povolených funkčních evaluací.

Nelder-Meadovu metodu jsem ladil optimalizací jejích vnitřních parametrů. Tuto optimalizaci jsem prováděl pomocí základní neupravené verze Nelder-Meadovy metody.

Metodu jsem také upravil tím způsobem, že jsem jí přidal funkci, která ji umožňuje vytvořit si zcela nový simplex během procesu optimalizování. Tato malá úprava velmi zvýšila efektivitu metody při optimalizaci většiny testovaných funkcí vyšších dimenzí.

Nalezené, optimalizované a upravené verze Nelder-Meadovy metody jsem poté porovnal s původní verzí metody. Nové verze metody dosahovaly znatelně lepších výsledků než původní neupravená verze jak při optimalizaci kvadratických funkcí, tak i v případě optimalizace Rosenbrockovy funkce. Odladěné verze metody jsem také zkoušel otestovat pomocí benchmarkovací platformy COCO, kde jsem se pokoušel optimalizovat různé jiné funkce. Ukázalo se, že nové verze metody byly k optimalizaci některých funkcí, které jsou součástí testovací sady COCO, vhodnější než originální Nelder-Meadova metoda.

Literatura

[1] Zdeněk Dostál a Petr Beremlijski. *Metody optimalizace*. 2015. http://mi21.vsb.cz/modul/metody-optimalizace.

[2] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*. 1960, 3 (3), 175-184. DOI 10.1093/comjnl/3.3.175.

[3] John A. Nelder a Ronald Mead. A Simplex Method for Function Minimization. *Comput. J.*. 1965, 7 308-313.

[4] Wikipedia contributors. *Nelder–Mead method — Wikipedia, The Free Encyclopedia*. 2019. https://en.wikipedia.org/wiki/Nelder-Mead_method. Online; accessed 16-May-2020.

[5] S. Singer a J. Nelder. Nelder-Mead algorithm. *Scholarpedia*. 2009, 4 (7), 2928. DOI 10.4249/scholarpedia.2928. Online; accessed 16-May-2020.

[6] Vincent Spruyt. *A geometric interpretation of the covariance matrix*. 2014. https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/. Online; accessed 16-May-2020.

[7] Zichen Wang. *PCA and SVD explained with numpy*. 2019. https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8. Online; accessed 16-May-2020.

[8] Pavel Koška. *Hybrid Nelder-Mead a rojové optimalizace*. 2013. http://hdl.handle.net/10467/21123.

[9] *COmparing Continuous Optimisers: COCO*. 2009. https://coco.gforge.inria.fr. Online; accessed 16-May-2020.

[10] Iztok Fajfar, Janez Puhan a Árpád Bűrmen. *Evolving a Nelder-Mead Algorithm for Optimization with Genetic Programming*. 2017. https://doi.org/10.1162/EVCO_a_00174.

[11] Benjamin Doerr, Mahmoud Fouz, Martin Schmidt a Magnus Wahlström. BBOB: Nelder-Mead with resize and halfruns. *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (Companion Material), ACM, 2239-2246 (2009)*. 2009, DOI 10.1145/1570256.1570312.

[12] Wikipedia contributors. *Rosenbrock function — Wikipedia, The Free Encyclopedia*. 2020. https://en.wikipedia.org/w/index.php?title=Rosenbrock_function&oldid=951682111. Online; accessed 11-May-2020.

Příloha A
Manuál k přiloženému programu

Přiložený program je skript psaný v programovacím jazyce Python. Jedná se o jednoduchou demonstraci systému, který jsem si vytvořil k testování a ladění Nelder-Meadovy metody.

V programu je možné porovnat efektivitu metody s různým nastavením svých vnitřních parametrů na náhodně vygenerované kvadratické nebo Rosenbrockově funkci. Celý skript je včetně svých výstupů psaný v angličtině a data se do něj zadávají přímou změnou konstant ve zdrojovém kódu vstupního skriptu.

A.1 Požadovaný software

Ke správnému běhu skriptu je třeba mít nainstalovaný Python 3 s následujícími knihovnami:

- NumPy
- Matplotlib
- SciPy

Se skriptem jsem pracoval v Pythonu 3.7.6, ale věřím, že i ostatní verze Pythonu 3 si s tímto programem poradí.

A.2 Použití skriptu

Ve zdrojovém kódu vstupního skriptu `main.py` se nachází několik konstant, pomocí kterých lze jednoduše zvolit, co má skript vykonat.

- FUNCTION – zvolí funkci k optimalizaci (0 – kvadratická, 1 – Rosenbrockova)
- DIM – dimenze optimalizované funkce
- EVALUATIONS – počet povolených funkčních evaluací
- SHOW_GRAPH – skript zobrazí graf průběhu optimalizace (`True` / `False`)
- BOUNDS – mez, která určuje, kde se může nacházet x^{opt}
- SOLVER_SETTINGS – čtveřice vnitřních parametrů (α, γ, ρ, σ); lze přidat i reinicializační parametr T jako pátý parametr

Zavoláním příkazu `python main.py` pomocí příkazového řádku ve složce, kde se skript nachází, se skript spustí a vykoná. Vygeneruje se zvolená funkce a provede se optimalizace pomocí zvolené verze Nelder-Meadovy metody. Na výstupu bude v textové podobě vypsáno, jaké optimální řešení solver nalezl a jaká je jeho funkční hodnota.

Pokud je ve skriptu konstanta SHOW_GRAPH nastavena na hodnotu `True`, tak se kromě textové podoby výstupu zobrazí i graf průběhu optimalizace.