MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jakub Bílý

CLM-metoda pro hledání globálního optima

Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: doc. RNDr. Petr Lachout, CSc.
Studijní program: Obecná matematika

Praha 2025

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Rád bych zde poděkoval svému vedoucímu bakalářské práce doc. RNDr. Petrovi Lachoutovi, CSc. za cenné rady a připomínky, kterými přispěl k vypracování této práce.

Název práce: CLM-metoda pro hledání globálního optima

Autor: Jakub Bílý

Katedra: Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: doc. RNDr. Petr Lachout, CSc., Katedra pravděpodobnosti a matematické statistiky

Abstrakt: Problém globální optimalizace se objevuje v mnoha oblastech matematiky, ekonomie i jiných vědách. Tato práce podává přehled různých přístupů k jeho řešení, včetně podrobného popisu konktrétních algoritmů. Značení z různých zdrojů je ujednoceno, důkazy správnosti těchto přístupů jsou předvedeny. Výhody a nevýhody různých postupů jsou diskutovány, včetně možných vylepšení. Zvýšená pozornost je věnována algoritmu Coupled local minimizers (CLM). Algoritmy jsou následně porovnány na různých testovacích funkcích a získané výsledky jsou představeny.

Klíčová slova: globální optimalizace, Coupled local minimizers, CLM

Title: CLM-method for global optimization

Author: Jakub Bílý

Department: Department of Probability and Mathematical Statistics

Supervisor: doc. RNDr. Petr Lachout, CSc., Department of Probability and Mathematical Statistics

Abstract: Global optimization problem appears in many domains of mathematics, economics and other sciences. This thesis presents various approaches for solving it, including detailed descriptions of particular algorithms. Notation from different sources is unified, proofs of correctness of these approaches are shown. Advantages and disadvantages of different approaches are discussed, with possible upgrades. Increased attention is given to Coupled local minimizers (CLM) algorithm. The algorithms are subsequently tested on different functions and obtained results are presented.

Keywords: global optimization, Coupled local minimizers, CLM

Obsah

1 Lokální a globální optimalizace **6**
1.1 Základní definice . . . 6
1.2 Lokální a globální optimalizace . . . 6
1.3 Příklad . . . 7

2 Přehled algoritmů globální optimalizace **9**
2.1 Rozdělení algoritmů . . . 9
2.2 Pokrývací metody . . . 9
2.2.1 Pijavského-Shubertův algoritmus (P-S algoritmus) . . . 9
2.2.2 Brentův algoritmus . . . 12
2.3 Metody náhodného průzkumu . . . 17
2.3.1 Algoritmy lokální optimalizace . . . 17
2.3.2 Náhodný vzorek, singlestart, multistart . . . 19
2.3.3 Adaptabilní algoritmy . . . 19
2.3.3.1 Algoritmy založené na normálním rozdělení . . . 20
2.3.3.2 Algoritmy se součtovou hustotou . . . 21
2.4 Metody obecného sestupu . . . 24
2.4.1 Metody trajektorií . . . 24
2.4.1.1 Algoritmus přechodu přes sedlové body . . . 24
2.4.2 Penalizační metody . . . 25
2.4.2.1 Tunelovací algoritmus . . . 26
2.5 Seskupovací/klastrovací algoritmy . . . 29
2.5.1 Klastrovací metody . . . 29
2.5.2 Törnův algoritmus . . . 31
2.5.3 Becker-Lagův algoritmus . . . 32
2.6 Nepřímé metody . . . 34

3 Coupled local minimizers **35**
3.1 Pomocné funkce a algoritmy . . . 35
3.2 CLM algoritmus . . . 38

4 Testování algoritmů **41**
4.1 Polynomy v jedné proměnné . . . 41
4.1.1 Testování parametrů pro CLM: . . . 42
4.1.2 Porovnání algoritmů: . . . 42
4.2 Griewankova funkce v různých dimenzích . . . 43
4.2.1 Porovnání algoritmů: . . . 46
4.3 Testování algoritmů - závěr . . . 47

Závěr **49**

Seznam literatury **50**

Seznam obrázků **51**

Seznam tabulek **52**

1 Lokální a globální optimalizace

Úlohy lokální i globální optimalizace spočívají v hledání lokálních a globálních extrémů funkce za daných podmínek a (obvykle) za nějakých předpokladů o vyšetřované funkci. V této kapitole budou představeny základní definice potřebné k popsání optimalizačního problému a dále budou představeny a porovnány lokální a globální optimalizační úloha. Na příkladu pak bude ilustrována úloha globální optimalizace.

1.1 Základní definice

Zde si nejprve připomeneme definice lokálního a globálního minima a lokálního a globálního maxima.

Definice 1 *Ať $B \subset \mathbb{R}^n, f : B \to \mathbb{R}$ je funkce.* ***Globálním minimem*** *funkce f na B rozumíme $x \in B$ takové, že pro všechna $y \in B$ platí $f(x) \leq f(y)$.*

Definice 2 *Ať $B \subset \mathbb{R}^n, f : B \to \mathbb{R}$ je funkce.* ***Ostrým globálním minimem*** *funkce f na B rozumíme $x \in B$ takové, že pro všechna $y \in B; y \neq x$ platí $f(x) < f(y)$.*

Definice 3 *Ať $B \subset \mathbb{R}^n, f : B \to \mathbb{R}$ je funkce.* ***Lokálním minimem*** *funkce f na B rozumíme $x \in B$ takové, že existuje $\delta > 0$ takové, že pro všechna $y \in B(x; \delta) \cap B$ platí $f(x) \leq f(y)$.*

Definice 4 *Ať $B \subset \mathbb{R}^n, f : B \to \mathbb{R}$ je funkce.* ***Ostrým lokálním minimem*** *funkce f na B rozumíme $x \in B$ takové, že existuje $\delta > 0$ takové, že pro všechna $y \in B(x; \delta) \cap B; y \neq x$ platí $f(x) < f(y)$.*

Globální a lokální maximum se definují obdobně, stejně pro globální ostré a lokální ostré maximum.

Poznámka V optimalizačních úlohách obvykle hledáme minimum funkce f. V případě, že je třeba nalézt maximum funkce f, lze využít skutečnosti, že funkce f nabývá maxima ve stejném bodě, kde funkce $-f$ nabývá minima.

Ve zbytku práce budeme proto uvažovat úlohu hledání minima.

1.2 Lokální a globální optimalizace

Obecně je optimalizační úloha obvykle zadána následovně: Ať $B \subset \mathbb{R}^n$. Nalezněte minimum funkce $f : B \to \mathbb{R}$ za podmínek:
$g_k(x) \leq 0; k \in K; K \subset \mathbb{N}_0$, K je konečná.
$h_l(x) = 0; l \in L; L \subset \mathbb{N}_0$, L je konečná.
Funkce f se nazývá *účelová funkce*, množina

$$D = \{x \in B | g_k(x) \leq 0, h_l(x) = 0, \forall k \in K, \forall l \in L\}$$

se nazývá *množina přípustných řešení*. Minimum funkce $f(x)$ na D se nazývá *optimální řešení*.

Všimněme si, že v zadání optimalizační úlohy nebylo rozlišeno, zda hledáme globální, či lokální minimum. Právě tento fakt rozděluje úlohy lokální a globální optimalizace. Pokud hledáme (jakékoli) lokální minimum, pak řešíme úlohu lokální optimalizace. Když hledáme globální minimum, jde o úlohu globální optimalizace.

Lokální optimalizace obvykle představuje méně obtížný problém k řešení. Algoritmy lokální optimalizace obvykle pracují rychle a garantují nalezení správného řešení. Globální optimalizace vytváří obtížnější problém. Algoritmy jsou výpočetně náročnější a nalezení správného řešení je garantováno pouze při splnění některých předpokladů pro f. Některé globální algoritmy využívají proces lokální optimalizace spuštěný ve více bodech najednou k nalezení všech lokálních minim a z nich následně vybírají globální minimum.

Poznámka Za určitých předpokladů o funkci f mohou úlohy globální a lokální optimalizace splývat do jedné. Jako příklad použijeme funkci f, která je na D konvexní a D je konvexní množina. Potom každé (obvykle jedno) lokální minimum f na D je i globální minimum.

1.3 Příklad

Následující příklad nám představí, jak může vypadat problém globální optimalizace v praxi a ukáže některé vlastnosti problémů globální optimalizace.

Příklad Letecká společnost plánuje novou leteckou trasu. Trasa částečně povede dosud nezmapovaným územím, kde nejdou k dispozici topografická data. Společnost potřebuje určit nadmořskou výšku nejvyššího bodu tohoto území, aby mohla stanovit minimální letovou hladinu pro svá letadla tak, aby nedošlo k havárii. Společnost má k dispozici satelit vybavený technologií LIDAR, který může změřit nadmořskou výšku libovolného bodu kdekoli na Zemi.

Matematická úloha se poté dá formulovat následovně: Uvažujeme systém souřadnic $x = (x_1; x_2)$, např. geografické souřadnice. Následně označme D množinu přípustných řešení (tedy dané nezmapované území). Označme
$f(x) = 10\ 000 -$ nadmořská výška v bodě x v metrech, tím získáme odpovídající úlohu s hledáním minima.

Nyní bychom chtěli najít souřadnice globálního minima $f(x)$ na D a funkční hodnotu v tomto bodě.

$$x_{min} = \arg\min_{x \in D} f(x)$$
$$y_{min} = \min_{x \in D} f(x)$$

V praxi se může stát, že funkce f má více globálních minim se stejnou funkční hodnotou, tedy přesnější zápis je:

$$X_{min} = \{x \in D | f(x) = y_{min}\}$$

Mezi další významné informace mohou patřit souřadnice a funkční hodnoty lokálních minim, nebo významné množiny maximálních přípustných hodnot (level sets):

$$L_d = \{x \in D | f(x) \leq d\}$$

Jak lze pak takový problém řešit? O povaze terénu víme velmi málo, uvažujme např. útes, kde funkce $f(x)$ nemusí být ani spojitá. Zdá se logické zvolit množinu bodů, které budou rovnoměrně rozděleny po celé množině D a v těchto bodech provést měření nadmořské výšky. Nejjednodušší možností je použít pravoúhlou mřížku, která pokrývá množinu D a provést měření v průsečících linií této mřížky. Tím získáme *globální* informaci o topografii D.

V oblastech s velkými rozdíly mezi fukčními hodnotami se nabízí v dalším kroku „zahustit "mřížku provedením dalších měření. Stejný postup je vhodné opakovat v okolí bodů s nízkou funkční hodnotou. Tím získáváme dodatečné *lokální* informace o těchto oblastech. Naopak je nepravděpodobné, že v okolí bodu s vysokou funkční hodnotou se bude nacházet globální minimum, tedy zde není vhodné provádět další měření.

Provedení měření samozřejmě není zadarmo, z čehož plyne, že není vhodné používat velký počet měření. Na druhou stranu s rostoucím hustotou mřížky a tím rostoucím počtem měření roste přesnost výsledků. I přesto nelze očekávat, že osamocený strom rostoucí na vrcholu hory bude algoritmem objeven.

Příklad ukazuje některé typické vlastnosti problému globální optimalizace. Jediná metoda, jak získat informace o f je získat její funkční hodnotu v x. Další informace o $f(x)$, jako hodnoty derivací nemusí být k dispozici. Pokud však tyto informace (hladkost funkce, lipschitzovskost, nebo omezenost druhé derivace) k dispozici jsou, tak lze využít efektivnější algoritmy. Hrubá dolní hranice pro f může být známá (všimněte si, že ve formulaci matematické úlohy očekáváme, že nezmapovaný terén nebude vyšší než 10 000 m). Někdy může být užitečné určit kromě globálního minima i všechna lokální minima.

Strategie řešení spočívá ve dvou fázích. V globální fázi prohledáváme celou množinu přípustných řešení D. V této fázi je třeba považovat všechny oblasti D za stejně důležité. Jakmile jsou získány prvotní informace o f, je možné nějaké oblasti D považovat za více důležité. V těchto oblastech je potom provedena lokální fáze. V lokální fázi pak dojde ke zlepšení výsledků v malé oblasti D.

2 Přehled algoritmů globální optimalizace

V této kapitole je představena jedna z možností rozdělení algoritmů globální optimalizace. Dále jsou v každé kategorii ukázány příklady těchto algoritmů. Jsou diskutovány předpoklady, pravděpodobnost nalezení správného řešení, rychlost konvergence a další vlastnosti těchto algoritmů.

2.1 Rozdělení algoritmů

Pro účely této práce budeme používat rozdělení algoritmů dle [Törn a Žilinskas 1989, s. 19], (v závorce anglický ekvivalent českého názvu):

1. Metody s garantovanou přesností (methods with guaranteed accuracy)
   (a) Pokrývací metody (covering methods)
2. Přímé metody (direct methods)
   (a) Metody náhodného průzkumu (random search methods)
   (b) Metody obecného sestupu (generalized descent methods)
   (c) Seskupovací/klastrovací metody (clustering methods)
3. Nepřímé metody (indirect methods)
   (a) Metody aproximující maximální přípustné hodnoty (methods approximating level sets)
   (b) Metody aproximující účelovou funkci (methods approximating objective function)

2.2 Pokrývací metody

Pro praktické problémy je obvyklé, že rychlost růstu účelové funkce je omezená. Ať $y_k, k \in \mathbb{N}$ je zatím nejnižší odhad hodnoty f v globálním minimu. Potom omezenost růstu f zaručuje, že ke každému bodu $x_i; i \in \mathbb{N}$, ve kterém je funkční hodnota $f(x) > y_k$, existuje okolí U_i, na kterém se globální minimum nenachází. Pokrývací metody využívají tato okolí k postupnému vylučování podmnožin D, ve kterých se globální minimum nenachází. Vzniká tak síť bodů $x_i; i \in \{1; 2; \ldots; k\}, k \in \mathbb{N}$ a pokrytí $D \subset \bigcup_{i \in \mathbb{N}} U_i$, proto se tyto metody nazývají pokrývací.

2.2.1 Pijavského-Shubertův algoritmus (P-S algoritmus)

Pro pochopení P-S algoritmu si připomeneme definici lipschitzovské funkce:

Definice 5 *Ať (X; μ); (Y; ν) jsou metrické prostory. Funkce $f : X \to Y$ se nazývá* ***lipschitzovská****, pokud $\exists K \geq 0$, že $\forall x_1; x_2 \in X$ platí $\nu(f(x_1); f(x_2)) \leq K\mu(x_1; x_2)$.*

Poznámka V této práci uvažujeme pouze reálné funkce. μ je eukleidovská metrika, tedy absolutní hodnota rozdílu (pro jednorozměrný případ), nebo norma rozdílu vektorů (pro vícerozměrný případ). ν je jednorozměrná eukleidovská metrika.

Pro potřeby P-S algoritmu použitého na $f : \mathbb{R} \to \mathbb{R}$ je vhodnější se na lipschitzovskost dívat alternativním pohledem: Zvolme pevné $x \in \mathbb{R}$. Díky lipschitzovskosti f můžeme pro $\forall y \in \mathbb{R}$ určit hodnoty, kterých může $f(y)$ nabývat. Tyto hodnoty jsou omezeny zhora a zdola dvojicí přímek. V grafu tyto přímky procházejí bodem $[x; f(x)]$ a mají směrnice K, resp. -K. Dohromady tak dělí graf na 4 kužely, přičemž vždy 2 tvoří oblast, kterou funkce f nemůže procházet (Obr. 2.1 nešrafovaná oblast) a 2 tvoří oblast, kde se f nachází (Obr. 2.1 šikmé šrafy).

Obrázek 2.1 Lipshitzovský kužel pro $f(x) = sin(x) - \frac{x}{4}; K = \frac{5}{4}, x = 7$.

Poznámka Tato vizualizace funguje i pro $f : \mathbb{R}^n \to \mathbb{R}$, zde pak získáme 2 kužely, ve kterých se f nemůže nacházet a zbytek (už ne 2 kužely), kde ano.

P-S algoritmus využívá v každé iteraci lipschitzovskost účelové funkce $f : [a; b] \to \mathbb{R}$ k vyloučení intervalů, ve kterých f nemůže nabývat globálního minima. Výsledkem po k iteracích je pak malý interval (nebo intervaly), ve kterých globální minimum zaručeně leží.

Předpokládejme, že účelová funkce f je lipschitzovská s konstantou K a máme najít její minimum na intervalu [a;b] $\subset \mathbb{R}$. P-S algoritmus pak provádíme následovně:

1. Globální fáze

   1.1 Volba počátečních bodů

2. Lokální fáze

   2.1 Vyloučení intervalů, kde minimum neleží

   2.2 Volba dalšího bodu

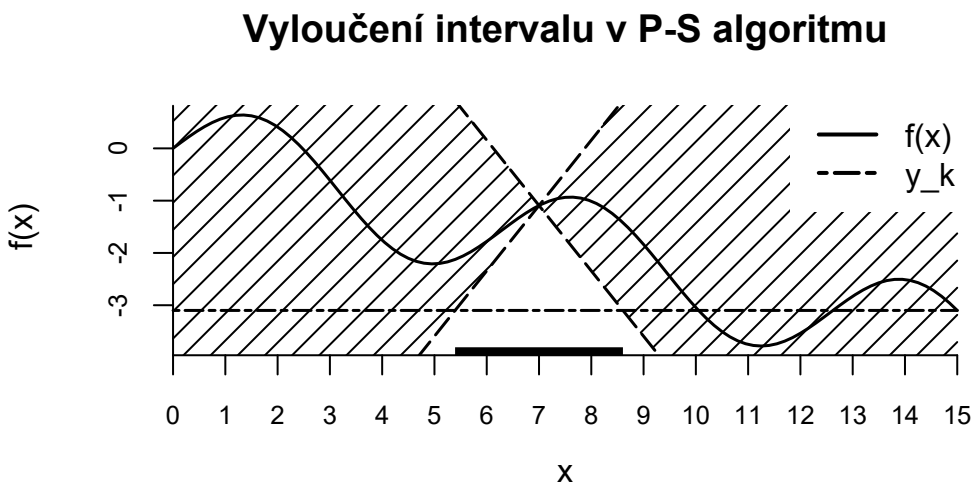
   2.3 Kontrola, zda nebylo nalezeno řešení

Globální fáze

Volba počátečních bodů: Pro nastartování algoritmu je třeba zvolit $k \in \mathbb{N}$ pozorování f. Označme tyto body $x_i;\ i \in \{1; 2; \ldots; k\}$ Z těchto pozorování získáme zatím nejlepší odhad hodnoty globálního minima $y_k = \min_{1 \leq i \leq k} f(x_i)$. Na počtu prvotních bodů příliš nezáleží, ale pro rychlý postup do lokální fáze je nejvýhodnější volit co nejméně. Pokud ale zvolíme jen jeden bod, vzniká pak problém s implementací v části Volba dalšího bodu. Jednou z vhodných možností je proto $(x_1; x_2; x_3)^T = (a; \frac{a+b}{2}, b)^T$.

Lokální fáze

Vyloučení intervalů, kde minimum neleží: Využijeme lipschitzovskosti f k vyloučení intervalů, kde minimum neleží. Pro $\forall\, x_i$ totiž můžeme vyloučit množinu $\{x | f(x_i) - K|x - x_i| > y_k\}$, kde nám lipschitzovskost f spolu s již známým odhadem hodnoty minima vylučuje existenci globálního minima (Obr. 2.2 zvýrazněná čás osy x).



Obrázek 2.2 Vyloučení množiny kolem $x_2 = 7; f(x) = sin(x) - \frac{x}{4}; K = \frac{5}{4}$

Dohromady tedy můžeme vyloučit množinu

$$A_k = \{x | \max_{1 \leq i \leq k} (f(x_i) - K|x - x_i|) > y_k\} \tag{2.1}$$

Volba dalšího bodu: Pro postup do další iterace volíme další bod x_{k+1}. Ten může být zvolen libovolně, ale jako intuitivní se zdá následující volba. Za x_{k+1} zvolíme bod, kde nám lipschitzovskost f spolu se známými hodnotami v x_i dává nejnižší přípustnou hodnotu $f(x)$:

$$x_{k+1} = \arg \min_{x \in [a;b]} \max\{f(x_i) - K|x - x_i|; 1 \leq i \leq k\} \tag{2.2}$$

Kontrola, zda nebylo nalezeno řešení: V této fázi provedeme kontrolu, zda nebylo nalezeno řešení s danou přesností ϵ. To lze provést různými způsoby, nastíníme následující dva:

Pomocí délky A_k: Nalezeneme celkovou délku A_k (A_k se obvykle rozpadá na více intervalů). Pokud platí $délka(A_k) > (b-a)-\epsilon$, tak jsme dosáhli stanovené přesnosti a můžeme algoritmus ukončit. Tento přístup lze upravit i pro vícerozměrný případ s použitím objemu A_k vůči objemu D. Při složitější struktuře D ale může být složité její objem určit. Navíc bude mít tento přístup problém s funkcí f, která je konstantní kolem svého globálního minima. Proto může být výhodné použít druhou metodu.

Pomocí rozdílu $f(x_k)$ a $f(x_{k+1})$: Řekneme, že řešení bylo nalezeno, když „zlepšení“hodnoty odhadu globálního minima je již dostatečně malé, tedy pokud platí $f(x_{k+1}) < f(x_k)$ a $f(x_k) - f(x_{k+1}) < \epsilon$. Tento přístup ale může algoritmus ukončit předčasně, pokud při volbě x_{k+1} zvolí bod s funkční hodnotou blízkou $f(x_k)$. Jako vhodná modifikace se tedy jeví kontrolovat více kroků zpět, než jen x_{k+1} a x_k.

Poznámka Ukončení algorimu nám garantuje pouze nalezení hodnoty globálního minima s danou přesností. Jeho poloha na $[a; b]$ ale může být stále nejistá. Jako příklad uvažujme funkci $f(x) = sin(x) - \delta\, x$ pro velmi malé δ na [0;15]. Pro velká ϵ zde algoritmus nechá dvě přípustné oblasti (okolí $\frac{3\pi}{4}$ a okolí $\frac{11\pi}{4}$). Pokud toto nastane, je možné použít algorimy lokální optimalizace na prohledání těchto přípustných oblastí a určení globálního minima jako nejmenšího z lokálních minim v těchto oblastech. Pokud zbyde jen jedna souvislá přípustná oblast, je možné za globální minimum označit x_{k+1}.

P-S algoritmus lze snadno modifikovat i pro vícerozměrný případ, absolutní hodnota v 2.1 a 2.2 je pak nahrazena vícerozměrnou eukleidovskou metrikou. Nicméně, způsob volby x_{k+1} popsaný v 2.2 se při rostoucí dimenzi problému stává obtížně řešitelný. Obvykle jsou proto používány jiné metody volby x_{k+1}.

P-S algoritmus u většiny funkcí garantuje nalezení globálního minima. Jako protipříklad lze použít lomené funkce, kde P-S algoritmus najde vícero lokálních minim, mezi nimi i globální. Nevýhodou je i pomalá konvergence (ukázáno např. v [Törn a Žilinskas 1989, s. 175]. Navíc hodnota K není obvykle známá. Je teda třeba používat algoritmy pro odhad K, které zvyšují pravděpodobnost chyby a výpočetní náročnost.

2.2.2 Brentův algoritmus

Brentův algoritmus je obdobou P-S algoritmu pro funkce s omezenou druhou derivací. Místo lipshitzovských kuželů zde přípustné hodnoty f omezují paraboly. Ať pro $f : [a; b] \to \mathbb{R}$ platí $\forall\, x\, |f''(x)| \leq M; M > 0$. Algoritmus pak postupuje následovně:

1. Globální fáze
   1.1 Volba počátečních bodů
2. Lokální fáze
   2.1 Kontrukce parabol a vyloučení intervalů, kde minimum neleží
   2.2 Volba dalšího bodu
   2.3 Kontrola, zda nebylo nalezeno řešení

Globální fáze

Volba počátečních bodů: Volba počátečních bodů je obdobná jako u P-S algoritmu. Označme tyto body $x_i; i \in \{1; 2; \ldots; k\}$ tak, že platí $x_1 < x_2 < \cdots < x_k$. y_k je opět zatím nejlepší odhad hodnoty lokálního minima.

Lokální fáze

Kontrukce parabol a vyloučení intervalů, kde minimum neleží: Pro každou dvojici po sobě jdoucích bodů x_i a x_{i+1}, zkonstruujeme parabolu P(x) takovou, že $P(x_i) = f(x_i); P(x_{i+1}) = f(x_{i+1}), P''(x) = M$ pro $\forall x \in (x_i; x_{i+1})$.

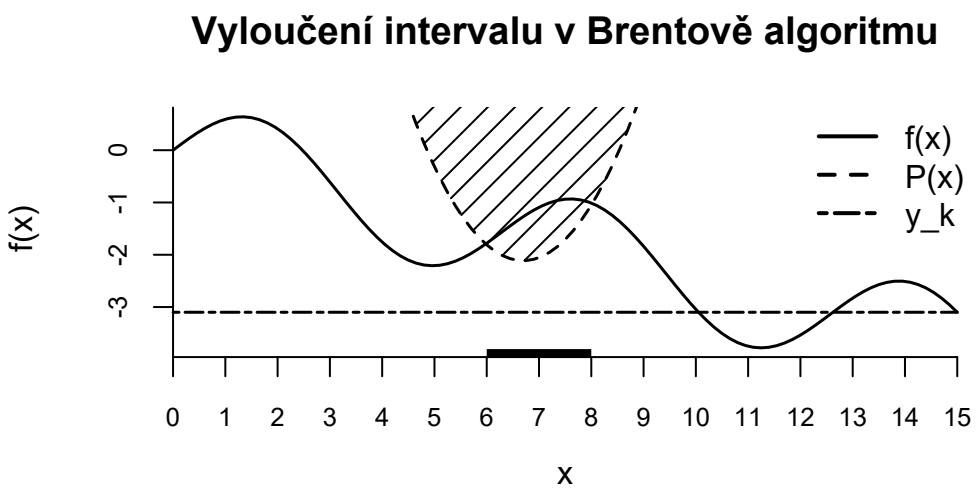
Ať $P(x) = a_1x^2 + a_2x + a_3$, potom platí $M = P''(x) = 2a_1$, tedy $a_1 = \frac{1}{2}M$. Ze zbylých podmínek získáme následující rovnice:

$$f(x_i) = P(x_i) = \frac{1}{2}Mx_i^2 + a_2x_i + a_3$$
$$f(x_{i+1}) = P(x_{i+1}) = \frac{1}{2}Mx_{i+1}^2 + a_2x_{i+1} + a_3$$

Řešením těchto rovnic získáme:

$$a_2 = \frac{f(x_{i+1}) - f(x_i) - \frac{1}{2}M(x_{i+1}^2 - x_i^2)}{x_{i+1} - x_i}$$
$$a_3 = f(x_i) - \frac{1}{2}Mx_i^2 - \frac{f(x_{i+1}) - f(x_i) - \frac{1}{2}M(x_{i+1}^2 - x_i^2)}{x_{i+1} - x_i}x_i$$

Platí, že P(x) je dolní mezí pro hodnoty f(x) na $(x_i; x_{i+1})$. Důkaz je předveden na konci kapitoly. Označme $p_i = \arg\min_{x \in [x_i; x_{i+1}]} P(x)$. p_k je tedy minimum paraboly P(x). Pokud platí $P(p_i) > y_k$, pak interval $(x_i; x_{i+1})$ neobsahuje globální minimum. Pokud ale $P(p_i) \leq y_k$, pak $[x_i; x_{i+1}]$ může globální minimum obsahovat a přecházíme do dalšího kroku.



Obrázek 2.3 Vyloučení intervalu v Brentově algoritmu pro $x_1 = 6; x_2 = 8; f(x) = sin(x) - \frac{x}{4}; K = \frac{5}{4}$.

Volba dalšího bodu: Předpokládejme, že v předchozím kroku jsme na intervalu $(x_i; x_{i+1})$ nevyloučili existenci globálního minima. Je tedy třeba interval dále rozdělit na dvě části a vyšetřit obě zvlášť. Metoda rozdělení intervalu se může různit, nejjednodušším řešením je $x_{k+1} = \frac{x_i + x_{i+1}}{2}$. Jiná intuitivní možnost je $x_{k+1} = p_i$. Všimněme si, že zatím jsme prováděli hledání pouze na malém intervalu $(x_i; x_{i+1})$, ne na celém $[a; b]$. Průzkum celého $[a; b]$ se dá zajistit vhodným střídáním intervalů, které zrovna prohledáváme po zvolení x_{k+1}. Jednou možností je volit vždy nejdelší interval, který ještě nebyl vyloučen. Druhou, složitější možností je volit interval, ve kterém je $P(p_k)$ nejnižší, tedy obdobně, jako u P-S algoritmu. Pro provedení tohoto přístupu však musíme po globální fázi provést kontrukci paraboly P(x) pro všechny intervaly $(x_i; x_{i+1}); i \in \{1; 2; \ldots; k\}$ a $(a; x_1); (x_k; b)$.

Kontrola, zda nebylo nalezeno řešení: Brentův algoritmus ukončujeme stejným způsobem, jako P-S algoritmus, tedy když délka množiny, kde se globální minimum nemůže nacházet přesáhne délku $[a; b] - \epsilon$, nebo jsou funkční hodnoty dostatečně blízko: $f(x_{k-1}) - f(x_k) < \epsilon$. Nalezená řešení se poté stejně jako u P-S algoritmu mohou nacházet ve více různých intervalech.

Brentův algoritmus stejně jako P-S garantuje nalezení globálního minima. Podobně jako P-S algoritmus má problém s konstantními částmi funkcí. Při praktickém použití s odhady K a M, kde je K a M odhadováno pomocí stejného počtu bodů, je Brentův algorimus rychlejší než P-S algoritmus, ale pravděpodobnost nalezení správného řešení je nižší viz [Törn a Žilinskas 1989, s. 175].

Brentův algoritmus využívá předpoklad, že P(x) je dolní mezí pro hodnoty f(x) na $(x_i; x_{i+1})$. To nyní dokážeme. K tomu použijeme následující lemma (všimneme si rozvolnění podmínky na druhou derivaci):

Lemma 1 *Ať pro $f : [a;b] \to \mathbb{R}$ platí $f \in C^2(a;b)$ a $M \geq 0$. Potom jsou následující ekvivalentní:*

(i) $\forall x \in (a;b); f''(x) \leq M$

(ii) $\forall x; y \in (a;b); x < y; f'(y) - f'(x) \leq M(y-x)$

(iii) $\forall u \in (a;b)$ *a* $\forall h > 0$ *dostatečně malé;* $f(u+h) + f(u-h) - 2f(u) \leq Mh^2$

Důkaz. Funkce f je $C^2(a;b)$ a [a;b] je omezený interval, tedy můžeme používat Lagrangeovu větu o střední hodnotě a integrovat druhou, první i nultou derivaci.

$(ii) \Rightarrow (i)$ Z vlastností limity platí

$$f''(x) = \lim_{y \to x} \frac{f'(y) - f'(x)}{(y-x)} \leq M$$

$(i) \Rightarrow (ii)$ Použijeme spor, ať (ii) neplatí, tedy
$\exists x; y \in (a;b); x < y : \frac{f'(y)-f'(x)}{(y-x)} > M$. Z Lagrangeovy věty platí $\exists\, c \in (x;y)$: $\frac{f'(y)-f'(x)}{(y-x)} = f''(c)$. Z toho získáváme

$$M < \frac{f'(y) - f'(x)}{(y-x)} = f''(c) \leq M$$

Tím získáváme spor.

$(i) \Rightarrow (iii)$ Zvolíme $u \in (a;b)$ a najdeme h takové, aby $a < u-h < u < u+h < b$. Upravíme levou stranu (iii):

$$\begin{aligned}
&f(u+h) - f(u) - (f(u) - f(u-h)) = \int_u^{u+h} f'(x)\,dx - \int_{u-h}^{u} f'(x)\,dx \\
&= \int_u^{u+h} f'(x) - f'(u)\,dx + \int_{u-h}^{u} f'(u) - f'(x)\,dx + \int_u^{u+h} f'(u)\,dx - \int_{u-h}^{u} f'(u)\,dx \\
&= \int_u^{u+h} \int_u^x f''(y)\,dy\,dx + \int_{u-h}^{u} \int_x^u f''(y)\,dy\,dx \\
&\overset{(i)}{\leq} \int_u^{u+h} \int_u^x M\,dy\,dx + \int_{u-h}^{u} \int_x^u M\,dy\,dx \\
&= M \int_u^{u+h} x - u\,dx + M \int_{u-h}^{u} u - x\,dx \\
&= \frac{M}{2}(h^2 - (-h^2)) = Mh^2
\end{aligned}$$

Ve druhém řádku jsme do rovnice přidali konstantu $f'(u)$ do obou integrálů. Poslední dva integrály v druhém řádku se navzájem vyruší.

$(iii) \Rightarrow (i)$ Rozvineme $f(u+h)$ a $f(u-h)$ do Taylorova polynomu v $f(u)$ s Lagrangeovo tvarem zbytku:

$$f(u+h) = f(u) + \frac{f'(u)}{1!}(u+h-u) + \frac{f''(c_h)}{2!}(u+h-u)^2; c_h \in (u; u+h)$$
$$f(u-h) = f(u) + \frac{f'(u)}{1!}(u-h-u) + \frac{f''(d_h)}{2!}(u-h-u)^2; d_h \in (u-h; u)$$

Dosadíme toto do (iii) a získáme:

$$\begin{aligned} Mh^2 \geq & f(u+h) + f(u-h) - 2f(u) = \frac{f''(c_h)}{2}h^2 + \frac{f''(d_h)}{2}h^2 \\ M \geq & \frac{f''(c_h) + f''(d_h)}{2} \end{aligned} \tag{2.3}$$

$c_h \in (u; u+h)$ a $d_h \in (u-h; u)$ tedy $\lim\limits_{h\to 0} c_h = u$, $\lim\limits_{h\to 0} d_h = u$. $f \in C^2(a;b)$, z vlastností limity a 2.3 pak platí:

$$M \geq \lim_{h\to 0} \frac{f''(c_h) + f''(d_h)}{2} = f''(u)$$

□

Lemma nám dává více ekvivalentních podmínek, za kterých je možné Brentův algoritmus použít. Vzorec (iii) nyní použijeme k důkazu, že P(x) je dolní mezí pro hodnoty f(x) na $(x_i; x_{i+1})$. Můžeme také rozvolnit podmínku pro druhou derivaci.

Tvrzení 2 *Ať pro $f : [a;b] \to \mathbb{R}$ platí $\forall\, x \in (a;b) :\ f''(x) \leq M; M > 0$. Položme:*

$$P(x) = \frac{(b-x)f(a) + (x-a)f(b)}{b-a} - \frac{1}{2}M(x-a)(b-x). \tag{2.4}$$

P(x) je stejný polynom jako P(x) definovaný na začátku této kapitoly. Potom platí $P(x) \leq f(x)\, \forall x \in [x_i; x_{i+1}] \subset [a;b]$.

Důkaz. Přeznačíme $(x_i; x_{i+1}) = (c;d)$. Položme $g(x) = f(x) - P(x)$, potom platí $g(c) = g(d) = 0$; $g''(x) \leq 0\, \forall\, x \in (c;d)$. Chceme tedy ukázat, že $\min\{g(x) | x \in [c;d]\} = y \geq 0$. Pro spor předpokládejme opak, tedy $y < 0$. Označme $u = \sup\{x \in [c;d] | g(x) = y\}$. Tedy u je bod, kde g nabývá minima a zároveň je z těchto bodů „nejvíce vpravo“ na [c;d].

$g(u) = y < 0 = g(c) = g(d)$, tedy $u \neq c; u \neq d$. g je navíc spojitá a tedy pro dostatečně malé h platí $c < u - h < u < u + h < d$. Protože f nabývá v u globálního minima, pak $g(u-h) \geq g(u)$. A protože u je „nejvíce vpravo“ z bodů, kde je nabýváno globálního minima, tak $g(u+h) > g(u)$. Tedy platí:

$$\frac{g(u+h) - g(u) + g(u-h) - g(u)}{h^2} > 0$$

Z předchozího lemmatu tedy $g''(x) > 0\, \forall x \in (c;d)$ Tím získáváme spor s vlastnostmi g.

□

2.3 Metody náhodného průzkumu

Metody náhodného průzkumu vycházejí z metod určených pro lokální optimalizaci. Prvním krokem obvykle bývá pokrytí D pomocí množiny bodů $\{x_k | k \in \{1; 2; \ldots; m\}; m \in \mathbb{N}\}$. Následně je provedena lokální optimalizace. Přitom předpokládáme, že pokud je pokrytí D body dostatečně husté, tak musíme najít všechna lokální minima, tedy i globální minimum.

2.3.1 Algoritmy lokální optimalizace

Pro potřeby základních algoritmů náhodného průzkumu si nejprve ukážeme dvojici metod lokální optimalizace - Newtonův algoritmus a algoritmus gradientního/nejprudšího sestupu.

Newtonův algoritmus

Mějme funkci $f : D \to \mathbb{R}; D \in \mathbb{R}^n$ a předpokládejme, že f má spojité parciální derivace druhého řádu na D. Dále předpokládejme, že f je konvexní na D.

Newtonův algoritmus spočívá v aproximaci funkce f v bodě x_k pomocí prvních 3 členů Taylorova polynomu. Algoritmus se poté přesune do bodu x_{k+1}, který je určen jako globální minimum této aproximace. Postup Newtonova algoritmu lze tedy popsat následovně:

1. Volba počátečního bodu
2. Konstrukce Taylorova polynomu
3. Nalezení minima Taylorova polynomu a x_{k+1}
4. Kontrola, zda nebylo nalezeno řešení

Volba počátečního bodu: Zvolíme x_0 jako libovolný bod z D $\subset \mathbb{R}^n$.

Konstrukce Taylorova polynomu: Spočteme první 3 členy Taylorova polynomu v bodě x_k:

$$f(x) \doteq f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) \tag{2.5}$$

Poznámka První a druhé derivace často nemáme k dispozici. V takovém případě je lze nahradit numerickou derivací:

$$\frac{\partial f(x)}{\partial x_i} \doteq \frac{f(x_1; \ldots; x_i + \delta; \ldots; x_n) - f(x_1; \ldots; x_i - \delta; \ldots; x_n)}{2\delta}$$

Obdobně lze z první derivace získat i druhou derivaci.

Nalezení minima Taylorova polynomu a x_{k+1}: Nalezneme x_{k+1} jako minimum Taylorova polynomu, tedy:

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \cdot \nabla f(x_k) \tag{2.6}$$

Kontrola, zda nebylo nalezeno řešení: Kontrolu, zda nebylo nalezeno řešení lze provést porovnáním funkčních hodnot posledních kroků, stejně jako u pokrývacích algoritmů. Druhou možností je porovnání délky posledního kroku, tedy $\|x_{k+1} - x_k\| < \epsilon$. Pokud řešení nalezeno nebylo, tak se vracíme ke konstrukci Taylorova polynomu s x_{k+1}.

Newtonův algoritmus má pro naše potřeby jednu nevýhodu. Jakožto algoritmus lokální optimalizace předpokládá, že funkce f je na D konvexní. Pokud f na D konvexní není, pak Newtonův algoritmus nemusí konvergovat, nebo konverguje k sedlovému bodu, či lokálnímu maximu [Nocedal a Wright 1999, s. 51]. Proto ukážeme ještě druhou metodu lokální optimalizace - metodu gradientního/nejprudšího sestupu.

Algoritmus gradientního/nejprudšího sestupu

Mějme funkci $f : D \to \mathbb{R}; D \in \mathbb{R}^n$, předpokládejme, že f má spojitý gradient ve všech bodech na D.

Tento algoritmus v každém kroku určí směr, kde funkční hodnota f na okolí x_k klesá nejrychleji. Následně se přesune v tomto směru.

1. Volba počátečního bodu
2. Určení směru nejprudšího sestupu
3. Určení délky kroku a provedení kroku
4. Kontrola, zda nebylo nalezeno řešení

Volba počátečního bodu: Zvolíme x_0 jako libovolný bod D $\subset \mathbb{R}^n$.

Určení směru nejprudšího sestupu: Mějme bod x_k, potom směr, ve kterém funkce f nejrychleji klesá je určen jako $s_k = -\nabla f(x_k)$

Určení délky kroku a provedení kroku: Provedeme krok ve směru s_k o délce γ_k, neboli $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$. Určení γ_k může být různé, ukážeme jednu z nich:

Položíme γ_k jako řešení následující pomocné úlohy:
$\gamma_k = \arg\min_{\gamma \geq 0} f(x_k) - \gamma \nabla f(x_k)$. Není nutné tuto úlohu vyřešit přesně, ale je důležité, aby získané γ_k bylo dostatečně malé, aby algoritmus „nepřeskočil“ k jinému lokálnímu minimu. Je tedy vhodné předem určit nějakou vhodnou horní hranici pro γ.

Kontrola, zda nebylo nalezeno řešení: Kontrolu, zda nebylo nalezeno řešení lze opět provést porovnáním funkčních hodnot posledních kroků, nebo porovnáním délky posledního kroku, tedy $\|x_{k+1} - x_k\| < \epsilon$. Pokud řešení nalezeno nebylo, tak provedeme další krok s x_{k+1}.

Výhododou gradientního algoritmu oproti Newtonovu algoritmu je garance, že najde lokální minimum pokaždé, nezávisle na x_0 (pokud platí určité předpoklady o f). Nevýhodou je pak pomalejší konvergence [Nocedal a Wright 1999, s. 51].

Nabízí se použít kombinaci těchto dvou algoritmů. Zvolíme x_0 a budeme postupovat pomocí gradientního algoritmu. Jakmile zjistíme, že je f v nějakém x_k

konvexní, tak přepneme na Newtonův algoritmus. To, zda je f v x_k konvexní ověříme díky následujícím ekvivalencím:

$$\begin{aligned} f\ je\ konvexní\ v\ x_k &\Leftarrow H_f(x_k)\ je\ pozitivně\ definitní \\ &\Leftarrow\ všechna\ vlastní\ čísla\ H_f(x_k)\ jsou\ kladná \end{aligned}$$

Kde $H_f(x_k)$ je matice druhých derivací f v x_k. Vlastní čísla lze určit např. QR algoritmem (více v [Sucháček 2017, s. 36]). Pro problémy ve vysoké dimenzi může být určování konvexity výpočetně náročné. Je tedy možné konvexitu ověřovat vždy jen v každém l-tém kroku, nebo vůbec a použít jen gradientní algoritmus.

V této práci budeme tento společný algoritmus nazývat Newton-gradientní algoritmus.

2.3.2 Náhodný vzorek, singlestart, multistart

Nejjednodušími metodami náhodného průzkumu jsou náhodný vzorek (random sample), singlestart a multistart. Všechny začínají zvolením $m \in \mathbb{N}$ a bodů $x_i; i \in \{1; 2; \ldots; m\}$, které jsou rozloženy tak, že rovnoměrně pokrývají množinu D.

Náhodný vzorek za globální minimum prohlásí bod s nejnižší funkční hodnotou, žádná další optimalizace není prováděna. Singlestart tuto metodu vylepšuje tím, že z tohoto nejlepšího bodu provede lokální optimalizaci (např. pomocí Newton-gradientního algoritmu). Tím nalezne lokální minimum příslušné tomuto bodu a prohlásí jej za řešení. Multistart provádí lokální optimalizaci pro všechny body $x_i; i \in \{1; 2; \ldots; m\}$ a následně vybere nejnižší dosažené lokální minimum.

Všechny tři metody při rostoucím m konvergují ke globálnímu minimu v pravděpodobnosti. Nejsou však příliš efektivní, náhodný vzorek nevyužívá dat z vyhodnocení funkčních hodnot z předchozích bodů. To způsobuje, že náhodný vzorek považuje všechny oblasti D za stejně důležité. Singlestart použitý s příliš malým m může pro provedení lokální optimalizace zvolit nevhodný bod x_i a skončit v lokálním minimu. Lokální fáze multistartu může skončit ve stejném lokálním minimu pro různé počáteční body. Díky tomu se zvyšuje výpočetní náročnost, kdy musí proběhnout několik lokálních optimalizací bez zlepšení výsledku, který byl dosažen při prvním nalezení lokálního minima.

I přes jejich zjevné nevýhody jsou tyto algoritmy hojně využívány pro jejich snadnou implementaci a robustnost. Dále často slouží jako základ komplexnějších metod, nebo pro porovnání efektivity pokročilejších algoritmů.

2.3.3 Adaptabilní algoritmy

Jednou z nevýhod předchozích tří metod je volba zkoumaných bodů před proběhnutím algoritmu, bez možnosti použít nalezené hodnoty $f(x_i)$ pro volbu dalších bodů v okolí lokálních minim. Tento problém řeší další skupina algoritmů, zvaná adaptabilní algoritmy. Ty mohou být rozděleny na parametrické a strukturální. Strukturální metody spočívají v změnách používaného algoritmu během jeho běhu. Je tak možné kombinovat výhody různých algoritmů v různých fázích optimalizace. Strukturálními metodami se tato práce dále nezabývá.

Základem parametrických metod je parametrická funkce, která určuje, jak volíme x_i. V každém kroku jsou pak parametry této funkce upravovány tak, aby další body byly rozloženy hustěji v okolí lokálních minim. Obvyklá volba spočítá ve zvolení nějakého pravděpodobnostního rozdělení s hustotou $g(x; p_k)$, kde p_k je vektor parametrů. V k-tém kroku pak volíme m_k bodů z rozdělení určeného $g(x; p_k)$. Po provedení jejich vyhodnocení určíme p_{k+1} a přejdeme do dalšího kroku.

2.3.3.1 Algoritmy založené na normálním rozdělení

Tyto algoritmy používají ke generování nových bodů vícerozměrné normální rozdělení $N_n(\mu_k; \Sigma_k)$. μ_k je nastaveno jako zatím nejlepší odhad minima f. Volba Σ_k je více komplikovaná. Nejjednodušší je Σ_k zvolit jako diagonální matici s σ_k^2 na diagonále. Složitější konstrukce Σ_k jsou také možné. Například je možné zohlednit i jiné body s hodnotou blízkou y_k a manipulovat Σ_k takovým způsobem, aby se více bodů generovalo mezi μ_k a těmito body.

σ_k^2 pak lze snižovat, pokud v k-tém kroku došlo k nalezení lepšího odhadu minima f a zvyšovat, pokud k nalezení lepšího odhadu nedošlo. Tento postup má své logické opodstatnění: Pokud se k minimu blížíme, tak chceme generovat více bodů v jeho okolí, abychom mohli tento odhad upřesnit. Pokud se však algorimus dostane
jeho hodnotu, místo toho, aby se přesunul jinam. Díky vlivu náhody na volbu bodů ve všech krocích, ale nakonec lokální minimum opustí a najde globální minimum. Lze vidět, že snižování σ_k^2 při každém úspěšném kroku má své nevýhody. Za určitých podmínek proto může být výhodné zvolit i jiný postup, kde se σ_k^2 zvyšuje i když v k-tém kroku došlo k nalezení lepšího odhadu minima f. Lze vidět, že možností nastavení $\mu_k; \Sigma_k$ a dalších parametrů je mnoho. Ukážeme tedy jednu z nich:

1. Globální fáze
   - 1.1 Volba počátečních bodů
2. Lokální fáze
   - 2.1 Vyhodnocení bodů a volba $\mu_k; \Sigma_k$
   - 2.2 Kontrola, zda nebylo nalezeno řešení
   - 2.3 Vygenerování nových bodů

Globální fáze

Volba počátečních bodů: Zvolme m bodů $x_{0;i}; i \in \{1; 2; \ldots; m\}$ rovnoměrně rozdělených na množině D. $\mu_0 = \arg\min_{i=1;2;\ldots;m}\{f(x_{0;i})\}$. Σ_0 zvolíme jako diagonální matici s σ_0^2 na diagonále. Volba σ_0^2 je pak závislá na velikosti D. Je třeba zvolit σ_0^2 dostatečně velké, aby rozdělení v prvních krocích nebylo příliš koncentrované do jednoho bodu. Ale pokud je σ_0^2 příliš velké, pak při generování bodů v dalších krocích většina bodů skončí mimo D a bude třeba je generovat znovu. Využijeme tedy statistického pravidla 3σ a položíme $\sigma_0^2 = (\frac{diam(D)}{3})^2$.

Lokální fáze

Vyhodnocení bodů a volba $\mu_k; \sigma_k$: Provedeme vyhodnocení f ve všech bodech $x_{k;i}$ a položíme $y_k = \min_{i=1;2;\ldots;m} f(x_{k;i})$. Dále $\mu_{k+1} = \arg\min\{f(\mu_k); f(\overline{\mu_k})\}$, kde $\overline{\mu_k} = \arg\min_{i=1;2;\ldots;m}\{f(x_{k;i})\}$. Zvolme $l \in \mathbb{N}$. Pokud v posledních l krocích nedošlo ke dostatečnému zlepšení odhadu, tedy platí $y_k > y_{k-l} - \epsilon$, tak položíme $\sigma_{k+1} = \sigma_0$. Pokud k dostatečnému zlepšení odhadu došlo, tak položíme $\sigma_{k+1} = a * \sigma_k$, kde $a \in (0; 1)$.

Kontrola, zda nebylo nalezeno řešení: Algoritmus zastavíme, pokud v posledních $2l$ krocích nedošlo k výraznému zlepšení výsledku, tedy $y_k > y_{k-2l} - \epsilon$. Globálním minimem poté nazveme μ_{k+1} s hodnotou y_k.

Vygenerování nových bodů: Vygenerujeme m nových bodů $x_{k+1;i}$ z rozdělení $N_n(\mu_{k+1}; \sigma_{k+1})$. Pokud se některý bod $x_{k+1;i}$ vygeneruje mimo množinu D, tak jej ignorujeme a vygenerujeme místo něj další.

Lze vidět, že fungování algoritmu závisí na volbě velkého množství parametrů (m, σ_0, a, l) a dalších rozhodnutí (Když se nový bod vygeneruje mimo D, generuji místo něj nový?, Pokud se nenajde lepší bod než v minulém kroku, používám $\mu_{k+1} = \mu_k$? apod.). Pro jeho implementaci bude obtížné nastavit všechny parametry tak, aby algoritmus fungoval efektivně, pokud o f nemáme další informace. V takovém případě je vhodné nastavit algorimus opatrněji (velké m, l, a) za cenu vyšší výpočetní náročnosti. Při opatrné implementaci pak algoritmus pracuje dobře i pro velmi nepravidelné funkce. Pokud však máme k dispozici informace o funkci f dopředu, tak je možné parametry nastavit způsobem, který zaručí rychlejší konvergenci. Díky vlivu náhody na běh algoritmu platí, že pravděpodobnost, že algoritmus globální minimum nenajde, není nulová.

Proto tento algoritmus může být dobře uplatněn, pokud řešíme podobné úlohy opakovaně a stačí nám najít optimální řešení ve většině případů.

2.3.3.2 Algoritmy se součtovou hustotou

Jinou možností, jak vytvořit pravděpodobnostní rozdělení pro generování bodů do dalšího kroku algoritmu, je přiřadit ke každému bodu vlastní hustotu a váhu, která se odvíjí od dosažené funkční hodnoty. Volba hustoty příslušné bodu x_i může být různá, lze například zvolit hustotu příslušnou $N_n(x_{k;i}; \Sigma_{k;i})$. Vážený součet těchto hustot pak vytvoří celkovou hustotu, ze které generujeme body do dalšího kroku. Uvedeme opět jeden z možných algoritmů:

1. Globální fáze
   - 1.1 Volba počátečních bodů
2. Lokální fáze
   - 2.1 Vyhodnocení bodů a výpočet nové hustoty
   - 2.2 Kontrola, zda nebylo nalezeno řešení
   - 2.3 Vygenerování nových bodů

Globální fáze

Volba počátečních bodů: Zvolme m bodů $x_{0;i}; i \in \{1; 2; \ldots; m\}$ rovnoměrně rozdělených na množině D.

Lokální fáze

Vyhodnocení bodů a výpočet nové hustoty: Spočteme $y_k = \min_{i=1;2;\ldots;m}\{f(x_{k;i})\}$ a určíme t_k jako index bodu, ve kterém f nabývá y_k. Ať $\alpha \in (0;1)$. Určíme váhy $c_{k;i}$ pro indexy $i \neq t_k$:

$$c_{k;i} = (1-\alpha)\frac{(f(x_{k;i}) - y_k)^{-1}}{\sum_{i=1;i\neq t_k}^{m}(f(x_{k;i}) - y_k)^{-1}}$$

Kde α je váha, kterou přiřazujeme bodu $x_{k;t_k}$, tedy $\alpha = c_{k;t_k}$.
Dále vytvoříme hustoty pro body $x_{k;i}$:

$$g_{k;i}(x) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}}e^{-\frac{1}{2}(x-x_{k;i})^T\Sigma^{-1}(x-x_{k;i})}$$

$g_{k;i}$ je pak hustota odpovídající $N_n(x_{k;i};\Sigma)$. Σ volíme jako diagonální matici s σ^2 na diagonále. σ^2 můžeme volit stejně jako v předchozím algoritmu: $\sigma^2 = (\frac{diam(D)}{3})^2$. Tato volba se ale zdá zbytečně obecná, protože se snaží rozložit body po celém D. Pro tento algoritmus ale stačí, aby $g_{k;i}$ rozložila body jen v okolí $x_{k;i}$. Tedy můžeme využít jinou volbu Σ:

Volíme $\Sigma_{k;i}$ jako diagonální s $\sigma^2_{k;i}$ na diagonále, $\sigma^2_{k;i} = (\frac{\frac{1}{2}K_{k;i}}{3})^2$, kde $K_{k;i} = \min\{\|x_{k;j} - x_{k;i}\| | j = 1;2;\ldots;j-1;j+1;\ldots;m\}$. Tedy každý bod ke společné hustotě přispívá hlavně ve svém okolí. Díky tomu se méně bodů generuje mimo D a více bodů se generuje v blízkosti zatím nejlepších $x_{k;i}$.

Společnou hustotu následně určíme jako:

$$g_k(x) = \sum_{i=1}^{m} c_{k;i}g_{k;i}(x)$$

Toto opravdu je hustota, protože:

$$\int_\Omega \sum_{i=1}^{m} c_{k;i}g_{k;i}(x)dx = \sum_{i=1}^{m} c_{k;i}\int_\Omega g_{k;i}(x)dx = \sum_{i=1}^{m} c_{k;i} =$$
$$= \alpha + (1-\alpha)\sum_{i=1;i\neq t_k}^{m}\frac{(f(x_{k;i}) - y_k)^{-1}}{\sum_{i=1;i\neq t_k}^{m}(f(x_{k;i}) - y_k)^{-1}} = \alpha + (1-\alpha) = 1$$

Kontrola, zda nebylo nalezeno řešení: Algoritmus opět ukončujeme při již malých změnách v y_k. Zvolme $l \in \mathbb{N}$, pak algoritmus ukončíme pokud $y_k > y_{k-l} - \epsilon$. Jako globální minimum označíme $x_{k;t_k}$.

Vygenerování nových bodů: Vytvoříme m nových bodů $x_{k+1;i}$ z rozdělení určeného $g_k(x)$. Bod $x_{k+1;i}$, který se vygeneruje mimo D ignorujeme a vygenerujeme jej znovu.

Stejně jako algoritmus založený na normálním rozdělení je i součtová hustota závislá na mnoha parametrech (m, l, α). Pokud předem nemáme žádnou informaci o f, musíme algoritmus nastavovat opatrně (velké m, l, malé α).

Pro volbu $\alpha < \frac{1}{2}$ může nastat situace, kdy váha odpovídající bodu $x_{k;t_k}$ bude menší než váha jiného bodu s funkční hodnotou blízkou y_k. Problém také nastává, pokud je funkce na nějaké množině konstantní. Potom se dva různé body mohou vygenerovat v této množině a vznikne problém při volbě t_k a $c_{k;i}$. Řešením je

modifikovat algoritmus tak, že nejprve vypočteme všechny $(f(x_{k;i})-y_k)^{-1}$ a až poté volíme α. Pokud je y_k dosaženo ve více bodech, tak lze „rozdělit“α mezi tyto body.

I pro algoritmy se součtovou hodnotou platí, že jsou robustní a je možné je snadno modifikovat podle různých informací o f a D. Nalezení globálního minima sice není garantováno, ale platí, že posloupnost řešení $x_{k;t_k}$ k němu konverguje v pravděpodobnosti. Stejně jako u algoritmu s normálním rozdělením může být tento algoritmus dobře uplatněn, pokud řešíme podobné úlohy opakovaně a stačí nám najít optimální řešení ve většině případů.

2.4 Metody obecného sestupu

Metody obecného sestupu jsou, obdobně jako metody náhodného průzkumu, založené na algoritmech určených pro lokální optimalizaci. Jejich základem je modifikace algoritmu lokální optimalizace tak, aby neuvízl v lokálním minimu. Toho lze dosáhnout více způsoby, dva z nich budou v této kapitole představeny.

2.4.1 Metody trajektorií

Metody trajetorií upravují Newton-gradientní algoritmus tak, aby po nalezení lokálního minima „vystoupal“po grafu funkce a překročil do oblasti, odkud zase „klesá“do dalšího lokálního minima. Po nalezení všech lokálních minim na D je určeno globální minimum.

2.4.1.1 Algoritmus přechodu přes sedlové body

Tento algoritmus po nalezení lokálního minima f stoupá do sedlového bodu, tím projde a následně zase přejde k lokální optimalizaci a najde další lokální minimum.

1. Globální fáze

   1.1 Zvolení startovacích bodů algoritmu

2. Lokální fáze

   2.1 Sestup do lokálního minima

   2.2 Kontrola, zda nedochází k zacyklení

   2.3 Kontrola, zda nebylo nalezeno řešení

   2.4 Výstup do sedlového bodu

   2.5 Průchod sedlovým bodem

Globální fáze

Zvolení startovacích bodů algoritmu: Protože se algorimus může zacyklit a postupovat opakovaně přes stejných několik lokálních minim, je třeba jej startovat postupně z více bodů. Vytvoříme mřížku překrývající množinu D. Průsečíky této přímky označíme jako $x_0^i; i \in \{1; 2; \ldots; m\}$. Zvolíme x_0^1 jako bod pro první spuštění algoritmu.

Lokální fáze

Sestup do lokálního minima: Buďme v bodě $x_{k_i}^i$, odtud chceme dojít do lokálního minima, použijeme tedy Newton-gradientní algoritmus. Zapamatujeme si nalezené lokální minimum jako $\chi_{k_i}^i$ a jeho hodnotu jako $y_{k_i}^i$.

Kontrola, zda nedochází k zacyklení: Algoritmus se může po návštěvách několika lokálních minim vrátit do minima, které už bylo předtím nalezeno. Případně také může být opětovně spuštěn z bodu, odkud bude klesat do minima, co již bylo nalezeno. Abychom tak bránili zacyklení algoritmu, tak zkontrolujeme, zda $\exists s; t \; s \leq k_i; t \leq i : \chi_s^t = \chi_k^i$ (kromě $s = k_i$ a $t = i$). Pokud takové s a t najdeme, tak algoritmus znovu spustíme z x_0^{i+1}. V praxi se χ_s^t „netrefí“přesně do $\chi_{k_i}^i$. Je tedy lepší ověřovat $\|\chi_s^t - \chi_{k_i}^i\| < \epsilon$.

Kontrola, zda nebylo nalezeno řešení: Pokud v předchozím kroku bylo zjištěno zacyklení a už nezbývá nový bod x_0^{i+1}, odkud by šlo algoritmus spustit (tedy $i = m$), tak algoritmus ukončíme. Za globální minimum pak prohlásíme χ_c^d, takové, že $y_c^d \leq y_s^t \; \forall s; t \; s \leq k_i; t \leq i$.

Výstup do sedlového bodu: Zvolíme $\epsilon > 0$ velmi malé. Sedlový bod leží ve směru vlastního vektoru v_{max}^1 Hessovy matice v bodě $\chi_{k_i}^i$, který odpovídá nejvyššímu vlastnímu číslu λ_{max}^1. λ_{max}^1 spolu s v_{max}^1 lze získat např. pomocí mocninné metody (viz. [Sucháček 2017, s. 12]). Následně provedeme krok ve směru v_{max}^1 o ϵ.

Zde opět určíme nový vlastní vektor v_{max}^2 odpovídající nejvyššímu vlastnímu číslu λ_{max}^2. Takto postupujeme dokud Hessova matice není semi-definitní. Odtud již můžeme použít Newtonovu metodu, tím se dostaneme do sedlového bodu $s_{k_i}^i$.

Průchod sedlovým bodem: Nalezený sedlový bod $s_{k_i}^i$ překonáme krokem ve směru $s_{k_i}^i - \chi_{k_i}^i$ o ϵ. Tím získáme bod $x_{k_i+1}^i = s_{k_i}^i + \epsilon(s_{k_i}^i - \chi_{k_i}^i)$. Následně přejdeme opět k sestupu do nového lokálního minima.

Z popisu algoritmu je zřejmé, že je velmi výpočetně náročný. Dochází totiž k častému vyhodnocování funkce f a to jak při sestupu (Newton-gradientní metoda), tak při výstupu (opakované výpočty Hessovy matice a mocninná metoda). Dále může dojít k opětovnému určení jednoho stejného lokálního minima pro každý bod x_0^i.

Pokud je v globální fázi mřížka zvolena nedostatečně hustě, algoritmus může vynechat některá lokální minima a tedy i globální minimum. Stejně tak volba ϵ je závislá na konkrétním problému. Příliš nízké ϵ zvyšuje výpočetní náročnost, příliš vysoké zase může vést k „přeskočení“ lokálního minima.

Předpokládejme, že dokážeme zkontruovat mřížku takovou, že vzdálenost mezi dvěma sousedními průsečíky je h a navíc pro každé lokální minimum $\chi_{k_i}^i$ funkce f platí: $\exists! \, x_0^i : x_0^i \in B(\chi_{k_i}^i; \frac{h}{2})$. Potom se volba $\epsilon = \frac{h}{16}$ zdá jako vhodná, neboť můžeme předpokládat, že sedlový bod se bude nacházet poblíž poloviny vzdálenosti mezi dvojicí $\chi_{k_i}^i; \chi_{k_i+1}^i$ a potřebujeme alespoň několik kroků výstupu k jeho přesnému určení.

Výše popsané nevýhody algoritmu přechodu přes sedlové body ukazují, že není příliš efektivním algoritmem. Pokud použijeme vhodné předpoklady o f, tak sice nalezne správné řešení, ale mnohem pomaleji než např. algoritmy náhodného průzkumu. Tedy pro praktické využití není vhodný.

2.4.2 Penalizační metody

Jednou z nevýhod používání metod lokální optimalizace pro globální problém je opakované určování stejného lokálního minima (jak již bylo diskutováno u multistartu). Penalizační metody zabraňují určení stejného lokálního minima přiřazením penalizační funkce ke každému nalezenému lokálnímu minimu. Následně je minimalizována pomocná funkce sestávající z účelové a penalizační funkce. Tím získáme nový bod, který (při správně zvolené pomocné a penalizační funkci) je v okolí zatím nenalezeného lokálního minima. To opět nalezneme pomocí lokální optimalizace a algoritmus pokračuje do dalšího kroku. Jako ukázku použijeme tunelovací algoritmus:

2.4.2.1 Tunelovací algoritmus

Tunelovací algoritmus je jednoduchým penalizačním algoritmem. Pro jednoduchost jej popíšeme v jednodimenzionálním případě.

1. Globální fáze

   1.1 Nalezení prvního lokálního minima

2. Lokální fáze

   2.1 Vytvoření a minimalizace tunelovací funkce

   2.2 Kontrola, zda bylo nalezeno lepší lokální minimum

   2.3 Kontrola, zda nebylo nalezeno řešení

Globální fáze

Nalezení prvního lokálního minima: Zvolíme nějaký bod $x \in D$. Provedeme lokální optimalizaci (např. Newtonovým algoritmem) a nalezeme lokální minimum x_1.

Lokální fáze

Vytvoření a minimalizace tunelovací funkce: Předpokládejme, že jsme v k-tém kroku a máme lokální minima $x_1; x_2; \ldots; x_k$. $\alpha = (\alpha_i; i \in \{1; 2; \ldots; k-1\})$ Zvolíme $\alpha_k = 1$. Definujeme tunelovací funkci:

$$T_\alpha^k(x) = \frac{f(x) - f(x_k)}{\prod_{j=1}^{k}((x - x_j)^2)^{\alpha_j}}$$

$\frac{1}{\prod_{j=1}^{k}((x-x_j)^2)^{\alpha_j}}$ zde funguje jako penalizační funkce. $T_\alpha^k(x)$ následně minimalizujeme. Minimalizaci provedeme zvolením bodu v okolí x_k a použitím algoritmu lokální optimalizace (např. Newtonova). Označme nalezené lokální minimum $\overline{x_{k+1}}$.

Kontrola, zda bylo nalezeno lepší lokální minimum: Pokud platí $f(\overline{x_{k+1}}) < f(x_k)$ pak se $\overline{x_{k+1}}$ nachází v okolí zatím neznámého lokálního minima f. Tedy provedeme lokální optimalizaci f započatou v $\overline{x_{k+1}}$ a získáme bod x_{k+1} a pokračujeme do další iterace algoritmu.

Pokud platí $f(\overline{x_{k+1}}) \geq f(x_k)$, tak není jasné, zda se $\overline{x_{k+1}}$ nachází v okolí lepšího lokálního minima. Můžeme dále postupovat dvěma způsoby: Volba nového bodu, odkud minimalizujeme $T_\alpha^k(x)$, nebo zvýšení α_k.

Volba nového bodu je v jednodimezionálním případě snadná: $\overline{x_{k+1}}$ se nachází mezi stejnou dvojicí $x_i; x_{i+1}$ $i \in \{1; 2; \ldots; k-1\}$ (zde předpokládáme, že jsou x_i seřazeny) jako bod, odkud byla spuštěna minimalizace $T_\alpha^k(x)$. To platí protože $\lim_{x \to x_i} T_\alpha^k(x) = \infty$; $\forall i \in \{1; 2; \ldots; k-1\}$, tedy algoritmus pro lokální optimalizaci nedokáže „přeskočit“mezi jinou dvojici $x_j; x_{j+1}$; $j \in \{1; 2; \ldots; k-1\}$. Pro pokračování algoritmu tedy stačí zvolit bod mezi jinou dvojicí $x_j; x_{j+1}$; $j \in \{1; 2; \ldots; k-1\}$, kde ještě nebyla funkce $T_\alpha^k(x)$ minimalizována.

Pokud už nelze zvolit další bod pro hledání minima $T_\alpha^k(x)$, pak musíme přistoupit k zvýšení α_k. Tím snížíme vliv penalizační funkce na $T_\alpha^k(x)$, zatímco vlastnost $\lim_{x \to x_i} T_\alpha^k(x) = \infty$; $\forall i \in \{1; 2; \ldots; k-1\}$ zůstane zachována. Potom opakujeme postup hledání lokálního minima s novou $T_\alpha^k(x)$. O kolik bude α_k zvýšeno je závislé na konkrétním problému. Postup zdvojnásobit α_k se zdá praktický pro většinu problémů.

Kontrola, zda nebylo nalezeno řešení: Pro kontrolu, zda bylo nalezeno řešení lze použít standartní postup s kontrolou $f(x_{k-1}) - f(x_k) < \epsilon$, případně se dívat více kroků zpět. Za globální minimum je pak prohlášeno x_k. Může ale nastat případ, že všechna lokální minima f na D byla nalezena a $f(x_{k-1}) - f(x_k) \geq \epsilon$. Pak může dojít k zacyklení algoritmu. Je tedy vhodné stanovit horní limit pro zvyšování α_k. Protože máme v každém kroku konečný počet možností volby bodů, odkud minimalizujeme $T_\alpha^k(x)$, tak nám tato podmínka zaručuje skončení algoritmu.

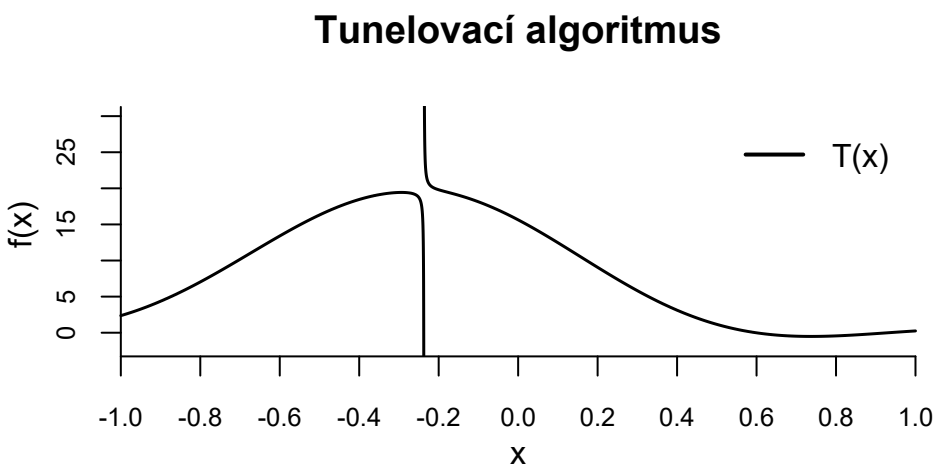
Pro vícedimenzionální případ je třeba provést několik modifikací, $T_\alpha^k(x)$ je třeba upravit takto:

$$T_\alpha^k(x) = \frac{f(x) - f(x_k)}{\prod_{j=1}^{k}((x - x_j)^T(x - x_j))^{\alpha_j}}$$

Pokud $f(\overline{x_{k+1}}) \geq f(x_k)$ a chceme zvolit nový bod pro lokální optimalizaci $T_\alpha^k(x)$, pak postup z jednodimenzionálního případu nejde použít. Nabízí se použít mřížku pokrývající D a postupně použít všechny body na průsečících tého mřížky. Další možnost je volit nový bod náhodně na D, zde je nutné zvolit nějaký maximální počet volených bodů, jinak může dojít k zacyklení algoritmu.

Pokud není lokální minimum x_k určeno zcela přesně, což je ve výpočetním softwaru obvyklé, tak na jedné straně od tohoto minima bude $T_\alpha^k(x)$ konvergovat do $-\infty$ namísto zamýšleného ∞. To nám vytváří problém, neboť minimalizace $T_\alpha^k(x)$ může skončit opět ve stejném minimu, nebo zkolabovat.

Jako příklad uvažujme $D = [-1; 1]; f(x) = sin(2\pi x) - \frac{x}{2}; \alpha = 1$. Předpokládejme, že jsme nalezli $x_1 \cong -0{,}237$, což je blízko skutečného lokálního minima. Potom funkce $T_\alpha^k(x)$ má následující tvar:



Obrázek 2.4 Tunelovací algoritmus s nepřesným lokálním minimem pro $f(x) = sin(x) - \frac{x}{2}$

Oprava tohoto problému není jednoduchá, můžeme zkusit dát čitatele $T_\alpha^k(x)$ do abs. hodnoty. To musíme provést jen na okolí x_k, neboť jinak při optimalizaci $T_\alpha^k(x)$ nenajdeme bod s menší funční hodnotou než x_k. Když použijeme pouze okolí,

pak zase $T_\alpha^k(x)$ může být nespojitá. Lepší možností pak je při optimalizaci $T_\alpha^k(x)$ kontrolovat vzdálenost od x_k a ukončit ji, pokud se blížíme zpět do x_k.

Při správně zvolených parametrech a používaném algoritmu pro lokální minimalizaci dokáže tunelovací algoritmus nalézt globální minimum. Efektivita záleží především na zvoleném algoritmu lokální optimalizace.

Nevýhodou je velký počet bodů, ve kterých je vyhodnocována f během lokální optimalizace. Pokud se pohybujeme v části D daleko od 0, pak je graf $T_\alpha^k(x)$ je obvykle skoro plochý. Newton-gradientní algorimus pak postupuje velmi malými kroky a zvyšují se numerické chyby.

Funkce f navíc může mít mezi x_i a x_{i+1} vícero lokálních minim, z jichž jen jedno má nižší funkční hodnotu než $f(x_k)$. Algoritmus pak může přejít mezi jiné x_j a x_{j+1} a toto možné globální minimum vynechat.

Tedy správné nastavení algoritmu s nedostatečnými informacemi o f je velmi obtížné. Pokud tyto informace existují, pak algoritmus nalezne globální minimum. Ale jiný algoritmus může být efektivnější při znalosti těchto informací o f.

2.5 Seskupovací/klastrovací algoritmy

Klastrovací algoritmy jsou skupinou algoritmů, které využívají nějakou klastrovací metodu. Mějme body $x_i \in D; i \in \{1; 2; \ldots; m\}; m \in \mathbb{N}$. Hlavním účelem klastrovací metody je identifikovat skupiny/klastry bodů, které se nacházejí v okolí stejného lokálního minima. Pokud se tento krok podaří, pak spustíme v každém klastru lokální optimalizaci, která určí odpovídající lokální minimum. Globální minimum pak snadno získáme jako nejmenší z lokálních minim.

Výhodou oproti např. multistartu je pak snížená výpočetní náročnost, protože lokální optimalizace není prováděna pro každý bod, ale jen jednou pro každé lokální minimum.

Poznámka Doposud jsme termíny algoritmus a metoda zaměňovali. V této kapitole budeme algoritmy globální optimalizace nazývat klastrovací algoritmy. Metody klastrování bodů, které jsou potřeba pro klastrovací algoritmy, budeme nazývat klastrovací metody.

2.5.1 Klastrovací metody

Začneme stručným úvodem do klastrovacích metod. Klastrovací metody lze obecně popsat následujícím postupem:

1. Volba počátečních bodů
2. Seskupení bodů kolem lokálních minim
3. Identifikace klastrů

Volba počátečních bodů: Zvolíme body $x_i \in D; i \in \{1; 2; \ldots; m\}; m \in \mathbb{N}$. To může být provedeno jak s pomocí mřížky, tak i náhodným vzorkem.

Seskupení bodů kolem lokálních minim: V dalším kroku je třeba nechat body seskupit poblíž lokálních minim. Jednou možností je provést v každém x_i několik kroků lokální optimalizace (např. Newton-gradientním algoritmem). Jinou je zvolit $0 < \alpha < 1$ a vyřadit $m \cdot \alpha$ bodů s nejvyšší $f(x_i)$. Tedy zůstanou jen body s nízkou hodnotou f.

Identifikace klastrů: Postupy jak identifikovat klastry se různí. Lze je však rozdělit do dvou kategorií - hierarchické (hierarchical) a rozdělovací (partitional).

Hierarchické metody se dají dále dělit na aglomerativní (agglomerative) a dělící (divisive). Aglomerativní začínají s klastry obsahujícími po jednom bodu, které jsou postupně spojovány. Dělící metody začínají s jedním klastrem obsahujícím všechny body, ten je následně rozdělován na menší klastry. Hierarchické metody se obvykle v optimalizaci nepoužívají, proto nebudou dále popisovány.

Rozdělovací metody zvolí počáteční bod (seed point) a k němu přidávají body, které jsou dostatečně blízko. Když už nejde další přidat, tak je ze zatím nepřidaných bodů zvolen nový počáteční bod. Dvě takové metody budou představeny.

Törnova klastrovací metoda: Pro potřeby Törnova algoritmu ukážeme Törnovu klastrovací metodu ([Törn 1977]):

Mějme body $x_i \in D \subset \mathbb{R}^n; i \in \{1; 2; \ldots; m\}; m \in \mathbb{N}$, které chceme klastrovat. Jako počáteční bod zvolíme bod x_p, kde má f nejnižší hodnotu ze všech bodů, které zatím nejsou v žádném klastru. Kolem x_p následně vytváříme zvětšující se koule $B(x_p; r_j)$ a přidáváme všechny body x_i, které jsou v těchto koulích a nejsou zatím v žádném klastru. Tvorbu koulí ukončíme, pokud je hustota bodů v $B(x_p; r_j) \backslash B(x_p; r_{j-1})$ menší, než hustota bodů v referenčním regionu. Hustotou bodů v množině A zde myslíme ρ_A definované jako:

$$\rho_A = \frac{\sum_{i=1}^{m} \mathbb{1}[x_i \in A]}{\lambda^n(A)}$$

Referenční region by měl obsahovat všechny body x_i. Jeho konstrukce může být složitá, proto počítáme rovnou jeho objem:

$$V = \prod_{i=1}^{m} 4\sqrt{\lambda_i}$$

Kde λ_i jsou vlastní čísla kovarianční matice $C = \{c_{ij}\}_{i;j=1}^{m}$, pro kterou platí

$$c_{ij} = \frac{1}{m-1}(\sum_{l=1}^{m} x_{il}x_{jl} - \frac{1}{m}\sum_{l=1}^{m} x_{il} \sum_{l=1}^{m} x_{jl})$$

Vlastní čísla můžeme získat např. QR algoritmem (popsáno např. v [Sucháček 2017]). Hustota bodů v referenčním regionu je potom $\rho = \frac{m}{V}$.
Objem $B(x_i; r_j) \backslash B(x_i; r_{j-1})$ je:

$$V_j = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)}(r_j - r_{j-1})^n$$

Hustotu bodů v $B(x_i; r_j) \backslash B(x_i; r_{j-1})$ poté získáme jako

$$\rho_j = \frac{\sum_{i=1}^{m} \mathbb{1}[r_{j-1} \leq \|x_i - x_p\| < r_j]}{V_j}$$

Pokud platí $\rho_j < \rho$, tak do klastru přidáme všechny body, které jsou v $B(x_p; r_{j-1})$ a zatím nejsou v žádném klastru. Potom přejdeme k hledání nového počátečního bodu.

Becker-Lagova klastrovací metoda: Pro potřeby Becker-Lagova algoritmu představíme ještě jednu klastrovací metodu.

Mějme body $x_i; i \in \{1; 2; \ldots; m\}; m \in \mathbb{N}$ a ať jsou tyto body seřazené: $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_m)$. Spočteme r_1 jako průměrnou vzdálenost bodu a jeho nejbližšího souseda a r_2 jako průměrnou vzdálenost mezi všemi body:

$$r_1 = \frac{1}{m}\sum_{j=1}^{m} \min\{\|x_i - x_j\| \mid i \in \{1;2;\ldots;m\}; i \neq j\}$$

$$r_2 = \frac{1}{\binom{m}{2}}\sum_{j=1}^{m}\sum_{i=1}^{j} \|x_i - x_j\|$$

Zvolme x_1 jako počáteční bod prvního klastru. Potom iterujeme následující postup:

Mějme $c_0^k = x_j$ jako počáteční bod k-tého klastru. Hledáme všechny body x_i, které zatím nejsou v žádném klastru a leží v $\overline{B(c_0^k; r_1)}$, neboli $\|x_i - c_0^k\| \leq r_1$. Pokud takové body nalezneme, tak položíme c_1^k jako jejich průměr. Tento postup opakujeme, dokud $c_l^k = c_{l-1}^k$, tedy už nejsou žádné nové body, které by se nacházely v $\overline{B(c_l^k; r_1)}$. Potom k-tý klastr obsahuje všechny body z poslední iterace, neboli body $x_i \in \overline{B(c_l^k; r_1)}$, které zároveň nejsou v žádném jiném klastru.

Následně se pokusíme najít bod x_i, takový, že x_i leží mimo $B(c_l^k; r_2)$, tedy $\|x_i - c_l^k\| \geq r_2$ a zároveň má z takových bodů nejnižší index (tedy i funkční hodnotu, neboť body x_i jsou seřazené). Pokud takový najdeme, tak ho prohlásíme za počáteční bod nového klastru a přejdeme do další iterace.

Pokud žádné takové x_i není, tak ukončíme algoritmus. Může se ale stát, že zbydou nějaké body x_i, které nejsou přiřazeny k žádnému klastru. Ty poté přiřadíme ke klastru, jehož středem je c_l^k nejbližší k x_i.

2.5.2 Törnův algoritmus

1. Globální fáze

    1.1 Volba počátečních bodů

2. Lokální fáze

    2.1 Seskupení bodů kolem lokálních minim

    2.2 Identifikace klastrů

    2.3 Kontrola zda nebylo nalezeno řešení

    2.4 Redukce počtu bodů v klastrech

Globální fáze

Volba počátečních bodů: Počáteční body jsou voleny náhodně, zvolíme m bodů.

Lokální fáze

Seskupení bodů kolem lokálních minim: Seskupení bodů je provedeno provedením p kroků lokální optimalizace.

Identifikace klastrů: Algoritmus používá Törnovu klastrovací techniku. Položíme x_i^k; $i \in \{1;2;\ldots;I(k)\}; k \in \{1;2;\ldots;K\}$ jako i-tý bod v k-tém klastru.

Kontrola zda nebylo nalezeno řešení: Pokud již v každém klastru zbývá jen jeden bod, nebo se od předchozího kroku nezměnil počet klastrů, tak ukončujeme algoritmus. Z každého klastru pak vybereme x_i^k s nejmenší funkční hodnotou a provedeme lokální optimalizaci. Za globální minimum pak vybereme nejnižší získané lokální minimum.

Redukce počtu bodů v klastrech: V každém klastru seřadíme body dle funkční hodnoty a každý $(N+1)$-ní bod $N \in \{0; 1; \dots\}$ ponecháme, zbylé vyřadíme. Přejdeme k seskupování bodů kolem lokálních minim.

Efektivita Törnova algoritmu záleží na použité metodě lokální optimalizace. Oproti multistartu je provedeno méně kroků lokální optimalizace v závislosti na N. Kombinace parametrů p a N nám dává možnost kontrolovat výpočetní náročnost a pravděpodobnost nalezení správného řešení.

Rychlost vyřazování bodů v Törnově algoritmu je velmi vysoká. Zvláště v prvních iteracích je možné, že bude vyřazen bod, který je v okolí globálního minima. Můžeme tedy modifikovat algoritmus tak, abychom v prvních iteracích vyřazovali body pomalu a v pozdějších rychleji.

2.5.3 Becker-Lagův algoritmus

Becker-Lagův algoritmus využívá Becker-Lagovu klastrovací techniku, aby nalezl množiny podezřelé z globáního minima. Následně je na těchto množinách spuštěn znovu k upřesnění jeho hodnoty. Budeme značit p = hloubka rekurze algoritmu.

1. Globální fáze
   - 1.1 Volba počátečních bodů
   - 1.2 Kontrola, zda nebylo nalezeno řešení
   - 1.3 Redukce počtu počátečních bodů
2. Lokální fáze
   - 2.1 Identifikace a seřazení klastrů
   - 2.2 Sestavení podmnožin D

Globální fáze

Volba počátečních bodů: Počáteční body jsou voleny náhodně, zvolíme m bodů.

Kontrola, zda nebylo nalezeno řešení: Položíme $y^p_{min} = \min\{f(x_i) | \, i \in \{1; 2; \dots; m\}$ a $x^p_{min} = \arg\min\{f(x_i) | \, i \in \{1; 2; \dots; m\}$. Pokud platí $y^{p-1}_{min} - y^p_{min} < \epsilon$, tak algoritmus v tomto rekurzivním kroku ukončíme a za minimum v tomto kroku prohlásíme x^p_{min}.

Redukce počtu počátečních bodů: Zvolíme $N < m$ a ponecháme N bodů s nejnižší funkční hodnotou.

Lokální fáze

Identifikace a seřazení klastrů: Zbylé body rozdělíme pomocí Becker-Lagovy klastrovací techniky. Získáme tak k klastrů. Seřadíme klastry dle nejnižší hodnoty $f(x_i)$ nacházející se v každém klastru.

Sestavení podmnožin D: Sestavíme podmnožiny $A_1; A_2; \ldots; A_k \subset D$, takové, aby A_i obsahovala všechny body z i-tého klastru. A_i můžeme zkonstruovat jako $A_i = B(c_i, r_i + \delta_i)$ kde c_i je střed i-tého klastru,
$r_i = \max\{\|x_j^i - c_i\| \mid x_j^i \, je \, bod \, v \, i - tém \, klastru\}$ a $\delta_i \geq 0$ je konstanta.

Následně na každé A_i spustíme algoritmus znovu. Tím získáme $x_{min;\,i}^{p+1}$, z nich poté vybereme $x_{min}^p = \arg\min\{f(x_{min;\,i}^{p+1}) | i \in \{1; 2; \ldots; k\}$ a to prohlásíme za řešení.

Poznámka Pokud zvolíme $\delta_i = 0$; tak bude jeden z bodů x_j^i na hranici A_i. Potom může být globální minimum sice nedaleko x_j^i, ale mimo A_i. Tedy ho algoritmus může vynechat. Pro δ_i příliš velké se mohou množiny A_i navzájem překrývat, potom algoritmus bude zbytečně vyhodnocovat stejnou oblast vícekrát.

Poznámka Pro zlepšení výpočetní náročnosti lze rekurzi algoritmu spouštět pouze v $A_1; A_2; \ldots; A_l; l < k$, tedy v l „nejlepších“klastrech.

Díky tomu, že Becker-Lagův algoritmus nevyužívá žádný algoritmus lokální optimalizace, tak je velmi robustní. Volba δ_i, l a N dává možnost vyvažovat výpočetní náročnost a pravděpodobnost nalezení špatného řešení.

Pokud algoritmus v prvním rekurzivním kroku vynechá okolí globálního minima, tak začne hledat v jiných oblastech D a s velkou pravděpodobností se již nevrátí (pokud nebylo zvoleno δ_i velké). Tedy první kroky algoritmu jsou důležitější než poslední. Nabízí setedy nastavit algoritmus striktně pro první kroky rekurze (vysoké m, N, δ_i, $l = k$) a volněji pro vyšší p.

2.6 Nepřímé metody

Nepřímé metody místo přímého hledání globálního minima vytvářejí zjednodušený model účelové funkce f. Následně je nalezeno globální minimum tohoto modelu a prohlášeno za globální minimum f.

Pro tyto metody je typická obtížná implemetace, proto nejsou v praxi obvykle využívány. Ze stejného důvodu budou v této práci pouze stučně nastíněny.

Metody aproximující maximální přípustné hodnoty

Jedním postupem je aproximovat množiny maximálních přípustných hodnot $L_d = \{x \in D | f(x) \leq d\}$. A následně najít bod, ke kterému se tyto množiny zmenšují, když snižujeme d. Snažit se L_d modelovat přímo z f se ukázalo jako nepraktické. Proto je f nejprve v m bodech x_i vyhodnocena a následně proložena jednodušší funkcí (obvykle lineární kombinace ortogonálních polynomů). U této funkce je pak určení L_d jednodušší.

Metody aproximující účelovou funkci

Tyto metody pracují se statistickým modelem účelové funkce. Obvyklou volbou je modelovat účelovou funkci náhodným procesem (např. Wienerovým). To provedeme tak, že vyhodnotíme funkci v m bodech x_i a následně předpokládáme, že trajektorie procesu procházejí ve všech bodech x_i odpovídající funkční hodnotou $f(x_i)$. Následně hledáme body s vysokou pravděpodobností, že v nich f má nízkou funkční hodnotu. Tyto body pak přidáváme do modelu.

3 Coupled local minimizers

Zde je představena pokročilá metoda globální optimalizace: Coupled local minimizers (CLM). V této metodě je nejprve zavedena rozšířená Lagrangeova funkce, kombinující funkční hodnoty ve všech bodech vzorku a informaci o jejich vzájemné vzdálenosti. Následně je tato funkce minimalizována pomocí modifikované Newtonovy metody. S rostoucím počtem provedených iterací CLM pak všechny body ve vzorku konvergují do globálního minima.

K popisu CLM algoritmu je potřeba nejprve definovat rozšířenou Lagrangeovu funkci a princip hledání jejího minima - Newtonův algoritmus s věrohodnými oblastmi (Trust region Newton algorithm).

3.1 Pomocné funkce a algoritmy

Obecná rozšířená Lagrangeova funkce

Definice 6 *Uvažujme minimalizaci účelové funkce $f(x) : \mathbb{R}^n \to \mathbb{R}$ za podmínek $h_i(x) = 0; i \in \{1; 2; \ldots; I\}; I \in \mathbb{N}$. Potom* ***rozšířenou Lagrangeovu funkci*** *definujeme jako:*

$$L_A(x; \lambda) = f(x) + \sum_{i=1}^{I} \lambda_i h_i(x) + \frac{\gamma}{2} \sum_{i=1}^{I} h_i^2(x)$$

Zde $\lambda = (\lambda_1; \lambda_2; \ldots; \lambda_I)$ je vektor Lagrangeových multiplikátorů a γ je penalizační parametr.

V k-tém kroku optimalizace chceme $L_A(x; \lambda^k)$ minimalizovat v x a získat tak x_k. Následně postoupíme do dalšího kroku nalezením nových λ_i^k:

$$\lambda_i^{k+1} = \lambda_i^k + \gamma h_i(x_k) \tag{3.1}$$

Pokud $\lambda_i^{k+1} = \lambda_i^k$, pak byly nalezeny optimální Lagrangeovy multiplikátory a tedy i minimum $L_A(x; \lambda^k)$.

Rozšířená Lagrangeova funkce pro CLM

Pro CLM využijeme upravenou verzi rozšířené Lagrangeovy funkce. Uvažujeme $x = (x_1; x_2; \ldots; x_J)^T$, kde x_j jsou body na D. Tyto body doplníme o $x_{J+1} = x_1$. Za účelovou funkci zvolíme průměr funkčních hodnot $f(x_j)$.
Dále položíme $0 = h_j(x) = x_j - x_{j+1}$. Potom rozšířená Lagrangeova funkce má následující tvar:

$$L_A(x; \lambda) = \frac{\eta}{J} \sum_{j=1}^{J} f(x_j) + \sum_{j=1}^{J} (\lambda_j)^\top (x_j - x_{j+1}) + \frac{\gamma}{2} \sum_{j=1}^{J} \|x_j - x_{j+1}\|^2$$

Kde η je váha pro účelovou funkci a γ je penalizační parametr. Správná volba těchto konstant má vliv na správnost a rychlost konvergence algoritmu. V CLM algoritmu pak pomocí Newtonovy metody (doplněné o věrohodné oblasti) hledáme minimum $L_A(x; \lambda)$.

Newtonův algoritmus s věrohodnými oblastmi (Trust region Newton algorithm) pro CLM

Poznámka Pro Newtonův algoritmus s věrohodnými oblastmi budeme bod, ze kterého v k-tém kroku provádíme optimalizaci, značit y_k místo x_k. Díky tomu půjde tyto body snadněji odlišit od bodů používaných v CLM algoritmu, které budeme snačit standartně x_k.

Jednoduchý Newtonovův algoritmus je založený na aproximaci minimalizované funkce pomocí Taylorova polynomu druhého řádu v bodě y_k. Z toho vycházejí následující dvě nevýhody: Přesnost této aproximace klesá s rostoucí vzdáleností od y_k a pokud $\nabla^2 f(y_k)$ není pozitivně definitní, tak v dalším kroku nemusí dojít ke zlepšení výsledku, neboli $f(y_{k+1}) > f(y_k)$ (z [Nocedal a Wright 1999, s. 135]).

Tyto nevýhody opravuje pokročilá varianta Newtonova algoritmu, zvaná Newtonův algoritmus s věrohodnými oblastmi (Trust region Newton algorithm, dále TRNA).

Využijeme Steihaugovy realizace TRNA (viz [Nocedal a Wright 1999, s. 75]). Hlavním rozdílem oproti standartnímu Newtonovu algoritmu je omezení maximální délky kroku, který algoritmus může mezi iteracemi udělat. Navíc v každém kroku probíhá kontrola, že nedochází ke zhoršení výsledku oproti předchozímu kroku.

Předpokádejme, že máme funkci $f(y) : \mathbb{R}^n \to \mathbb{R}$, známe její první a druhé derivace. Dále máme bod y_k, odkud chceme udělat krok do y_{k+1}. Tento krok omezíme na věrohodnou oblast danou $\Delta_k > 0$, neboli $\|y_{k+1} - y_k\| \leq \Delta_k$. Volba Δ_k bude diskutována později v (3.3). V každém kroku potom řešíme pomocnou optimalizační úlohu:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f(y_k) + (\nabla f(y_k))^\top p + \frac{1}{2} p^T (\nabla^2 f(y_k)) p \qquad z.p. \|p\| \leq \Delta_k \tag{3.2}$$

Pomocnou úlohu řešíme pomocí Steihaugovy realizace TRNA:

1. Volba počátečního směru
2. Kontrola konvexity ve směru d_j
3. Kontrola zda řešení není mimo věrohodnou oblast
4. Kontrola zda řešení není již dostatečně přesné

Volba počátečního směru: Algoritmus začneme zvolením počátečních parametrů. Zvolme $\epsilon > 0$, a položme $p_0 = 0; r_0 = \nabla f(y_k); d_0 = -r_0$. ϵ je přesnost, se kterou chceme najít řešení. d_0 odpovídá směru nejprudšího sestupu.

Kontrola konvexity ve směru d_j: Zkontrolujeme, zda je funkce „konvexní ve směru d_j“, neboli zda platí $d_j^\top (\nabla^2 f(y_k)) d_j > 0$. Pokud toto neplatí, tak řešíme lineární optimalizační úlohu:

$$\min_{\tau \in \mathbb{R}} m_k(p_j + \tau d_j) \qquad z.p. \|p_j + \tau d_j\| = \Delta_k$$

Tato úloha má jen dvě přípustná řešení a to průsečíky přímky procházející přes p_j ve směru d_j s kružnicí se středem v 0 a poloměrem Δ_k. Ty lze najít např. pomocí dělení intervalů $[-2\Delta_k; 0]$ a $[0; 2\Delta_k]$. Získáme dvojici řešení

$\overline{p_1} = p_j + \tau_1 d_j; \overline{p_2} = p_2 + \tau_2 d_j$. Za řešení úlohy pak prohlásíme $\overline{p} = \arg\min\{m_k(\overline{p_1}); m_k(\overline{p_2})\}$ a ukončíme algoritmus.

Pokud $d_j^\top(\nabla^2 f(y_k))d_j > 0$ platí, pak položíme:

$$\alpha_j = \frac{r_j^\top r_j}{d_j^\top(\nabla^2 f(y_k))d_j}$$
$$p_{j+1} = p_j + \alpha_j d_j$$

Kontrola zda řešení není mimo věrohodnou oblast: Pokud platí $\|p_{j+1}\| > \Delta_k$, neboli další krok je mimo věrohodnou oblast, tak musíme řešení „vrátit "zpět do věrohodné oblasti. Tedy najdeme $\tau \geq 0$ takové, že $\|p_j + \tau d_j\| = \Delta_k$. Opět lze použít dělení intervalů. Řešení $\overline{p} = p_j + \tau d_j$ pak prohlásíme za řešení úlohy a ukončíme algoritmus.

Pokud $\|p_{j+1}\| < \Delta_k$, pak

$$r_{j+1} = r_j + \alpha_j(\nabla^2 f(y_k))d_j$$

Kontrola zda řešení není již dostatečně přesné: Pokud platí $\|r_{j+1}\| < \epsilon\|r_0\|$, tak algoritmus ukončíme a za řešení úlohy prohlásíme $\overline{p} = p_{j+1}$. Pokud ne, tak položíme:

$$\beta_{j+1} = \frac{r_{j+1}^\top r_{j+1}}{r_j^\top r_j}$$
$$d_{j+1} = r_{j+1} + \beta_{j+1} d_j$$

A zahájíme novou iteraci algoritmu od kontroly konvexity d_{j+1}.

Steihaugův postup navrhne vektor posunu $\overline{p}$, resp. nový bod $y_{k+1} = y_k + \overline{p}$ kam bychom se měli přesunout. Nyní můžeme ukázat celý TRNA algoritmus, ten postupuje následovně:

1. Volba počátečního bodu a věrohodnostní oblasti
2. Nalezení řešení pomocné úlohy na věrohodnostní oblasti
3. Kontrola zda bylo nalezeno řešení a určení nové věrohodnostní oblasti

Volba počátečního bodu a věrohodnostní oblasti: Počáteční bod y_0 může být zadaný, nebo ho můžeme zvolit libovolně na D. Zvolíme $\overline{\Delta}$, neboli horní hranici pro poloměr věrohodné oblasti. Vhodná volba $\overline{\Delta}$ je závislá na povaze problému, pokud o funkci f máme dodatečné informace, jako např. předpokládanou odchylku m_k od f, tak lze $\overline{\Delta}$ volit lépe, než pro obecný případ. V obecném případě můžeme volit např. $\overline{\Delta} = \frac{1}{4}diam(D)$. Pokud nechceme algoritmus omezovat v maximální velikosti věrohodné oblasti, tak zvolíme $\overline{\Delta} = diam(D)$. Pro spuštění algoritmu zvolíme $\Delta_0 = \overline{\Delta}$.

Nalezení řešení pomocné úlohy na věrohodnostní oblasti: Mějme bod y_k, vyřešíme pomocnou úlohu 3.2 pomocí Steihaugovy realizace TRNA, získáme vektor posunu $\overline{p_k}$.

Kontrola zda bylo nalezeno řešení a určení nové věrohodnostní oblasti: Položíme

$$\rho_k = \frac{f(y_k) - f(y_k + \overline{p_k})}{m_k(0) - m_k(\overline{p_k})} \tag{3.3}$$

Čitatel tohoto zlomku nazýváme skutečná redukce, jmenovatel předpokládaná redukce. Na základě hodnoty ρ_k pak měníme velikost Δ_k:

Pokud $\rho_k < \frac{1}{4}$, pak položíme $\Delta_{k+1} = \frac{1}{4}\|\overline{p_k}\|$. Pokud $\rho_k > \frac{3}{4}$ a zároveň $\|\overline{p_k}\| = \Delta_k$, pak položíme $\Delta_{k+1} = \min\{2\Delta_k; \overline{\Delta}\}$ Pokud nenastane ani jedna z předchozích, pak $\Delta_{k+1} = \Delta_k$.

Dále musíme rozhodnout, zda krok $y_{k+1} = y_k + \overline{p_k}$ přijmeme, nebo ne. Protože $\overline{p_k}$ je výsledkem minimalizace m_k na oblasti obsahující 0, tak předpokládaná redukce je vždy nezáporná. Pokud platí $\rho_k < 0$, tak pomocná úloha nenašla $\overline{p_k}$ takové, aby došlo k zlepšení výsledku do dalšího kroku a tedy necháváme $y_{k+1} = y_k$. Pokud ke zlepšení výsledku došlo, pak $y_{k+1} = y_k + \overline{p_k}$.

Volby Δ_{k+1} a y_{k+1} pak lze shrnout následovně: Pokud je zlepšení výsledku příliš malé (nebo nedošlo ke zlepšení), pak zmenšujeme věrohodnou oblast. Pokud přitom nedojde ke zlepšení ($\rho_k < 0$), tak nepřijmeme k-tý krok a začínáme znovu v y_k. Pokud je naopak zlepšení velké a algoritmus došel až na kraj věrohodné oblasti, tak mu v dalším kroku věrohodnu oblast rozšíříme a přesuneme se do $y_{k+1} = y_k + \overline{p_k}$. V ostatních případech necháme věrohodnou oblast o stejné velikosti a přesouváme se do $y_{k+1} = y_k + \overline{p_k}$.

Poznámka Může se stát, že výsledkem řešení pomocné úlohy bude $\overline{p_k}$ takové, že $y_k + \overline{p_k} \notin D$. V takovém případě odmítneme krok a zmenšíme věrohodnou oblast, neboli $y_{k+1} = y_k$ a $\Delta_{k+1} = \frac{1}{4}\|\overline{p_k}\|$

Standartní postup ukončení algoritmu, pokud $f(y_k) - f(y_{k+1}) < \epsilon$ zde nelze použít, protože za běhu algoritmu dochází k odmítání některých kroků, tedy $y_k = y_{k+1}$. Lepším postupem je kontrolovat velikost navrhnutého posunutí do dalšího kroku, neboli zda $\overline{p_k} < \epsilon$.

3.2 CLM algoritmus

V předchozí části jsme definovali všechny potřebné funkce a algoritmy. Můžeme přejít k popisu CLM algoritmu:

1. Globální fáze
   - 1.1 Volba počátečních bodů a L. multiplikátorů
2. Lokální fáze
   - 2.1 Minimalizace $L_A(x; \lambda_k)$
   - 2.2 Nalezení λ_{k+1}
   - 2.3 Kontrola, zda nebylo nalezeno řešení

Globální fáze

Volba počátečních bodů a L. multiplikátorů: Počáteční body mohou být voleny náhodně, nebo s pomocí mřížky. Zvolíme $J \in \mathbb{N}$ bodů x_j, položíme $x_0 = (x_{0;1}; x_{0;2}; \ldots; x_{0;J})^\top$. Dále zvolíme počáteční Lagrangeovy multiplikátory $\lambda_0 = (\lambda_{0;1}; \lambda_{0;2}; \ldots; \lambda_{0;J})^\top$. Můžeme položit $\lambda_0 = (0; 0; \ldots; 0)^\top$, nebo volit $\lambda_{0;j}$ náhodně.

Lokální fáze

Minimalizace $\boldsymbol{L}_A(x; \lambda_k)$: Předpokládejme, že jsme v n-tém kroku algoritmu a máme $x_n = (x_{n;1}; x_{n;2}; \ldots; x_{n;J})^\top$, kde $x_{n;j}$ jsou body na D. Ty doplníme o $x_{n;0} = x_{n;J}$ a $x_{n;J+1} = x_{n;1}$. Mějme Lagrangeovy multiplikátory $\lambda_n = (\lambda_{n;1}; \lambda_{n;2}; \ldots; \lambda_{n;J})^\top$. Ty doplníme o $\lambda_{n;0} = \lambda_{n;J}$.

Použijeme TRNA na $L_A(x; \lambda_k)$, s počátečním bodem $y_0 = x_n$. Polynom $m_k(p)$ v k-tém kroku TRNA konstruujeme následovně:

$$m_k(p) = L_A(y_k; \lambda_k) + (\nabla L_A(y_k; \lambda_k))^T p + \frac{1}{2} p^T (\nabla^2 L_A(y_k; \lambda_k)) p$$

První a druhé derivace $\nabla L_A(y_k; \lambda_k) = (\nabla_{y_{k;j}} L_A(y_k; \lambda_k))_{j=1}^J$ a $\nabla^2 L_A(y_k; \lambda_k) = (\nabla^2_{y_{k;j}; y_{k;i}} L_A(y_k; \lambda_k))_{j;i=1}^J$ získáme jako:

$$\begin{aligned}
\nabla_{y_{n;j}} L_A(y_n; \lambda_k) &= \frac{\eta}{J} \nabla_{y_{n;j}} f(y_n) + \lambda_{k;j} - \lambda_{k;j-1} + \\
&\quad + \gamma(y_{n;j} - y_{n;j+1}) - \gamma(y_{n;j-1} - y_{n;j}) \ \forall j \in \{1; 2; \ldots; J\} \\
\nabla^2_{y_{n;j}; y_{n;j}} L_A(y_n; \lambda_k) &= \frac{\eta}{J} \nabla^2_{y_{n;j}; y_{n;j}} f(y_n) + 2\gamma \qquad \forall j \in \{1; 2; \ldots; J\} \\
\nabla^2_{y_{n;j}; y_{n;j-1}} L_A(y_n; \lambda_k) &= -\gamma \qquad \forall j \in \{2; 3; \ldots; J\} \\
\nabla^2_{y_{n;j}; y_{n;j+1}} L_A(y_n; \lambda_k) &= -\gamma \qquad \forall j \in \{1; 2; \ldots; J-1\} \\
\nabla^2_{y_{n;j}; y_{n;i}} L_A(y_n; \lambda_k) &= 0 \qquad \forall i; j : |j - i| \geq 2
\end{aligned}$$

TRNA algoritmus začneme volbou $y_0 = x_k$. Výsledek TRNA algoritmu označíme x_{k+1}.

Nalezení λ_{k+1}: Nalezneme nové Lagrangeovy multiplikátory λ_{k+1}:

$$\lambda_{k+1;j} = \lambda_{k;j} + \gamma h_j(x_{k+1})$$

Kontrola, zda nebylo nalezeno řešení: Zastavení algoritmu lze provést několika způsoby: Použijeme způsob využívající změny $L_A(x_k; \lambda_k)$. Pokud platí $L_A(x_k; \lambda_k) - L_A(x_{k+1}; \lambda_{k+1}) < \epsilon$ pak algoritmus zastavíme a za řešení prohlásíme x_{k+1} s odpovídajícími Lagrangeovými multiplikátory λ_{k+1}.

Při správném nastavení parametrů η a γ CLM algoritmus konverguje do globálního minima ([A. Teughels a Suykens 2003, s. 5]). Vhodná volba těchto parametrů závisí na daném problému. Zvolíme-li vysokou hodnotu pro γ, pak zvyšujeme rychlost konvergence algoritmu, ale také pravděpodobnost, že algoritmus skončí v lokálním minimu, které není globálním minimem. Vliv η na běh algoritmu je více složitý, při změně jeho hodnoty vůči γ může dojít ke zrychlení i zpomalení konvergence nebo ke konvergenci k neoptimálnímu řešení (viz. [A. Teughels a Suykens 2003, s. 5]).

Využití Newtonovy medody dává CLM algoritmu rychlou konvergenci. TRNA modifikace potom zajišťuje, že v každém kroku dochází ke zlepšení výsledku. Na

druhou stranu je třeba znát druhé derivace minimalizované funkce. Implementace algoritmu ve výpočetním softwaru je poměrně složitá.

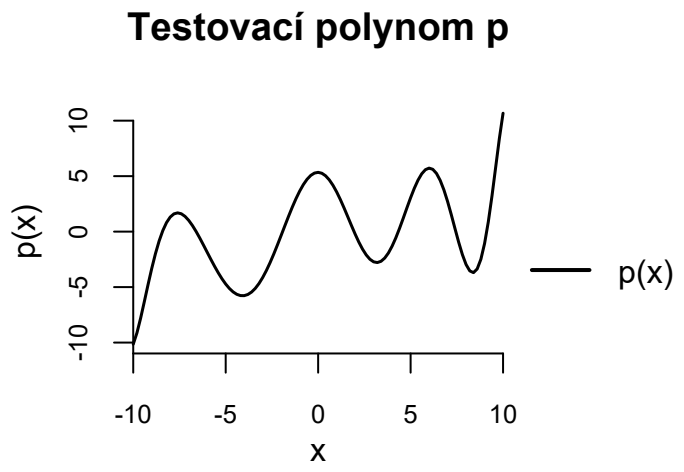
4 Testování algoritmů

4.1 Polynomy v jedné proměnné

V této kapitole jsou vybrané algoritmy globální optimalizace testovány na náhodně generovaných polynomech v jedné proměnné. Algoritmy jsou porovnávány dle dosažené úspěšnosti nalezení řešení a času běhu algoritmu. Algoritmus je považován za úspěšný, pokud se získaný výsledek liší od skutečného globálního minima o nejvýše 0,1.

Testovací funkce

Náhodné polynomy jsou generovány následujícím způsobem:
K číslům $x_i \in \{-13; -11; \ldots 11; 13\}$ jsou přiřazeny náhodné funkční hodnoty z intervalu $(-5; 5)$. Následně jsou tyto body proloženy Lagrangeovým polynomem. Tím získáváme polynom s několika lokálními minimy.



Obrázek 4.1 Příklad testovacího polynomu

Poznámka Protože Lagrangeovy polynomy na krajích intervalu, ze kterého jsou vytvářeny často oscilují, tak minimalizaci provádíme pouze na intervalu $(-10; 10)$. Testování dále ukázalo, že CLM algoritmus má problém nalézt globální minimum, které se nachází na hranici intervalu (-10, nebo 10), proto jsou takové polynomy vyřazeny. V praktické aplikaci se tato nevýhoda dá opravit porovnáním výsledku CLM s hodnotami v krajních bodech.

Testované algoritmy a jejich nastavení

Algoritmy byly testovány v prostředí R. Byly porovnávány následující algoritmy:
P-S algoritmus (P-S): Parametr K byl získán odhadem pomocí 21 bodů rovnoměrně rozložených po $(-10; 10)$.

Náhodný vzorek (RS): Náhodný vzorek využíval 1000 bodů rovnoměrně rozložených po $(-10; 10)$.

TRNA-multistart (TRNA-MS): Multistart byl spuštěn ve 20 bodech rovnoměrně rozložených na $(-10; 10)$. Jako algoritmus lokální optimalizace byla použita Steinhaugova realizace TRNA.

Součtová hustota (souč. hust.): Algoritmus využívající součtovou hustotu využívá 200 bodů a parametr $\alpha = 0.5$.

Törnův algoritmus (Törn): Törnův algoritmus využívá 3 kroky line-search Newtonova algoritmu (zajišťuje funkční hodnotu $\leq$ než v předchozím kroku). Algoritmus je spuštěch ve 20 náhodně vybraných bodech na $(-10; 10)$.

CLM (CLM): Různé volby η, γ a J byly testovány.

4.1.1 Testování parametrů pro CLM:

Pro vhodné nastavení CLM algoritmu je třeba určit správné parametry $J; \eta; \gamma$. Autorovi práce není známý žádný postup ani intuice, jak tuto volbu provést. Proto byly testovány různé kombinace těchto parametrů na 100 náhodných polynomech. Vybrané výsledky jsou představeny v následujících tabulkách (pole v tabulce je tvaru: úspěšnost v % | průměrná doba běhu v s). Případy s 40% a vyšší úspěšností jsou zvýrazněny.

Poznámka V případě, že doba běhu algoritmu překročila 10 s, byl algoritmus považován za neúspěšný a doba jeho běhu byla nastavena na 10 s. Případy, kde tato situace nastala alespoň jednou, jsou v tabulce označeny hvězdičkou.

Výsledky testů CLM pro J = 11 a různé $\eta; \gamma$						
$\eta \backslash \gamma$	0,1	0,2	0,5	0,7	1	1,5
0,5	31 | 0,16	35 | 0,07	16 | 0,38	7* | 0,28*	9 | 0,57	1* | 1,47*
1	41 | 0,08	26 | 0,10	20 | 0,09	21 | 0,13	15* | 0,32*	4* | 0,69*
2	29* | 0,30*	27 | 0,10	27 | 0,06	24 | 0,08	26 | 0,15	20 | 0,25
5	37 | 0,24	41* | 0,42*	37 | 0,08	24 | 0,06	40 | 0,10	23 | 0,18
10	28* | 0,58*	51 | 0,25	40 | 0,13	34* | 0,28*	32* | 0,24*	27 | 0,07
15	28* | 0,64*	56 | 0,35	45 | 0,37	41* | 0,41*	46 | 0,09	27 | 0,07
20	25* | 0,98*	35* | 0,58*	44* | 0,51*	49* | 0,47*	41 | 0,14	32* | 0,30*
25	24* | 1,51*	34* | 0,59*	43* | 0,54*	43 | 0,19	49* | 0,45*	39* | 0,26*
30	27* | 1,59*	33* | 0,59*	47 | 0,31	56* | 0,54*	43 | 0,30	42* | 0,43*

Tabulka 4.1 Výsledky testů CLM pro J = 11 a různé $\eta; \gamma$

Tabulka 4.2 Výsledky testů CLM pro J = 8 a různé $\eta; \gamma$

Vidíme, že žádná z testovaných kombinací parametrů $J; \eta; \gamma$ nezaručuje nalezení globálního minima na každé testovací funkci. Pro porovnání s jinými algoritmy zvolíme nejlepší kombinaci parametrů z hlediska úspěšnosti nalezení minima.

4.1.2 Porovnání algoritmů:

CLM algoritmus je porovnán s jinými algoritmy globální optimalizace. Porovnání je provedeno na 1000 náhodných polynomech. Dvě volby parametrů pro CLM

Výsledky testů CLM pro J = 8 a různé $\eta; \gamma$						
$\eta \backslash \gamma$	0,1	0,2	0,5	0,7	1	1,5
0,5	20 | 0,03	25 | 0,07	16 | 0,11	4* | 0,29*	2 | 0,14	0* | 0,44*
1	37 | 0,05	28 | 0,04	25* | 0,16*	14 | 0,05	6 | 0,16	9 | 0,09
2	39* | 0,19*	35 | 0,05	37 | 0,03	21 | 0,05	24 | 0,05	18 | 0,06
5	39 | 0,23	36 | 0,11	32 | 0,05	30 | 0,05	27 | 0,40	28 | 0,50
10	35* | 0,56*	41 | 0,24	36 | 0,19	38 | 0,05	21 | 0,05	24 | 0,04
15	40* | 0,99*	38 | 0,37	52 | 0,17	44 | 0,09	40* | 0,18*	28 | 0,05
20	25* | 0,90*	32* | 0,63*	46 | 0,17	38 | 0,13	37* | 0,18*	31 | 0,05
25	22* | 1,68*	47* | 0,61*	49* | 0,31*	39 | 0,12	42* | 0,20*	40* | 0,17*
30	21* | 1,59*	32* | 0,64*	40 | 0,27	50 | 0,16	45 | 0,10	34* | 0,21*

jsou použity:
CLM_1 s parametry $(J; \eta; \gamma) = (8; 10; 0{,}3)$
CLM_2 s parametry $(J; \eta; \gamma) = (11; 15; 0{,}6)$

Porovnání algoritmů na náhodných polynomech							
	P-S	RS	TRNA-MS	souč. hust.	Törn	CLM_1	CLM_2
úspěšnost (%)	99,8	100	100	99,4	96,6	46,2	44,4
prům. doba běhu ($s \cdot 10^{-2}$)	2,74	0,0067	0,91	0,17	0,99	16,14	34,67

Tabulka 4.3 Porovnání vybraných algoritmů na náhodných polynomech v jedné proměnné

Kromě CLM je úspěšnost algoritmů rovna nebo velmi blízko 100 %. U algoritmů které 100 % nedosahují lze tento výsledek vysvětlit vlivem náhody na volbu parametrů (P-S), nebo při běhu algoritmu (součtová hustota, Törn).
Pro obě nastavení CLM vykazuje výrazně horší úspěšnost a (často až o několik řádů) delší dobu běhu. Jednotné nastavení $J; \eta; \gamma$ pro všechny testované funkce se ukázalo jako nevhodný postup. Pokud by existovala metoda, jak pro danou testovanou funkci parametry určit (podobně jako např. odhad K pro P-S algoritmus), bylo by možné výsledky CLM zlepšit. Jak již bylo řečeno, žádná taková metoda není autorovi známa.
Volba používané testovací funkce - polynom jedné proměnné více pomáhá algoritmům používajícím hrubou sílu (RS, součtová hustota) oproti složitějším algoritmům (CLM, TRNA-MS). Je pravděpodobné, že při rostoucí dimenzi problému se výsledky více nakloní směrem ke složitějším algoritmům. Více v další kapitole.

4.2 Griewankova funkce v různých dimenzích

V této kapitole jsou vybrané algoritmy globální optimalizace testovány na Griewankově funkci. Tato funkce je definována pro libovolný počet proměnných, testy

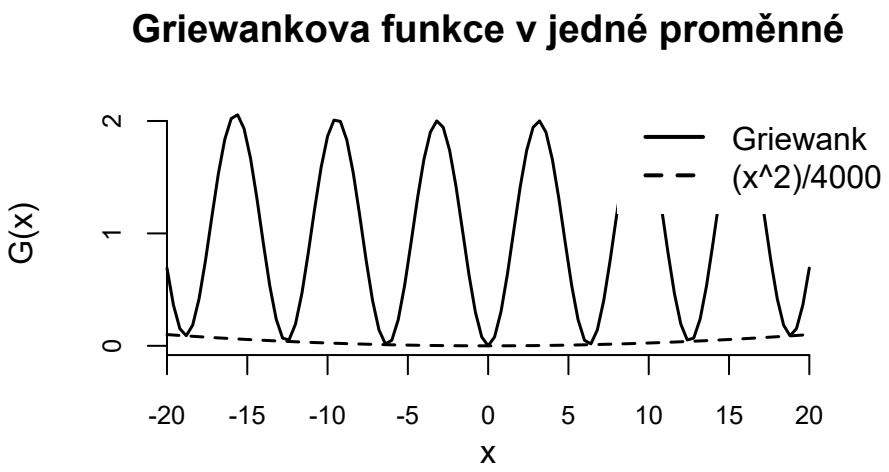
jsou prováděny pro 3, 5 a 7 proměnných. Algoritmy jsou porovnávány dle dosažené úspěšnosti nalezení řešení a času běhu algoritmu. Algoritmus je považován za úspěšný, pokud se získaný výsledek liší od skutečného globálního minima o nejvýše 0,1 (v odpovídající eukleidovské normě).

Testovací funkce

Griewankova funkce v proměnných $x_1; x_2; \ldots; x_n$ $n \in \mathbb{N}$ je definována jako:

$$G(x_1; x_2; \ldots; x_n) = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right)$$

Pro libovolné n má Griewankova funkce globální minimum v bodě $(0; 0; \ldots; 0)$ a mnoho lokálních minim. (viz obrázek pro $n = 1$). Derivace jsou snadno odvoditelné ze vzorce. Testování bylo prováděno na množině $[-20, 20]^n$.



Obrázek 4.2 Griewankova funkce jedné proměnné na okolí 0

Volba bodů, ze kterých spouštíme algoritmus

Body, ze kterých jsou algoritmy spouštěny volíme následovně: Nejprve zvolíme $k \in \mathbb{N}$, následně spočítáme $r = \frac{b-a}{k}$ a vytvoříme souřadnice z_j, tak, že $z_1 = a + \frac{r}{2}$; $z_2 = z_1 + r; \ldots; z_k = b - \frac{r}{2}$. Ze všech možných n-tic těchto souřadnic pak získáme body $x_i; i \in \{1; 2; \ldots; k^n\}$. Tyto body jsou pro stejná $k; n; a; b$ pevně dané, tedy opakovaným spouštěním algoritmu bychom dospěli ke stejnému výsledku. Přidáme tedy vliv náhody: Ke každé souřadnici každého bodu přičteme c_i, které bylo zvoleno náhodně na intervalu $(-\frac{r}{2}; \frac{r}{2})$.

Testované algoritmy a jejich nastavení

Poznámka k bylo voleno tak, aby se doba běhu algoritmu pohybovala řádově v desítkách sekund, ne vždy však byla taková volba možná (např. Törnův algoritmus v dimenzi 7), pak bylo použito menší k.

Algoritmy byly testovány v prostředí R. Byly porovnávány následující algoritmy:

Náhodný vzorek (RS): Byly použity následující hodnoty k:

k pro Náhodný vzorek:			
dimenze	3	5	7
k	200	22	9

Tabulka 4.4 Volba k pro Náhodný vzorek v různých dimenzích

TRNA-multistart (TRNA-MS): Jako algoritmus lokální optimalizace byla opět použita Steinhaugova realizace TRNA. k bylo voleno následovně:

k pro TRNA-multistart:			
dimenze	3	5	7
k	10	3	2

Tabulka 4.5 Volba k pro TRNA-multistart v různých dimenzích

Törnův algoritmus (Törn): Törnův algoritmus využívá 3 kroky line-search Newtonova algoritmu (zajišťuje funkční hodnotu $\leq$ než v předchozím kroku). k bylo voleno následovně:

k pro Törnův algoritmus:			
dimenze	3	5	7
k	10	4	2

Tabulka 4.6 Volba k pro Törnův algoritmus v různých dimenzích

CLM (CLM): Pro určení η, γ byl proveden test různých kombinací podobně jako při testování na náhodných polynomech. Každá kombinace byla testována pomocí 10-ti pokusů. Následně byla použita kombinace η, γ s nejvyšší úspěšností a nejnižším časem.

Nastavení parametrů CLM:			
dimenze	3	5	7
k	5	2	2
η	10	15	10
γ	1	0,5	0,5

Tabulka 4.7 Volba $k; \eta$, γ pro CLM v různých dimenzích

4.2.1 Porovnání algoritmů:

Poznámka V případě, že doba běhu algoritmu překročila 120 s, byl algoritmus považován za neúspěšný a doba jeho běhu byla nastavena na 120 s. Případy, kde tato situace nastala alespoň 10x, jsou v tabulce označeny hvězdičkou.

Porovnání algoritmů v dimenzi 3				
	RS	TRNA-MS	Törn	*CLM*
úspěšnost (%)	40	76*	87	34*
prům. doba běhu (s)	41,27	67,04*	21,78	61,97*

Tabulka 4.8 Porovnání algoritmů na Griewankově funkci ve třech proměnných

Porovnání algoritmů v dimenzi 5				
	RS	TRNA-MS	Törn	*CLM*
úspěšnost (%)	0	26	83	60
prům. doba běhu (s)	67,80	36,96	35,51	21.28

Tabulka 4.9 Porovnání algoritmů na Griewankově funkci v pěti proměnných

Porovnání algoritmů v dimenzi 7				
	RS	TRNA-MS	Törn	*CLM*
úspěšnost (%)	0	30	79	7*
prům. doba běhu (s)	42,59	17,89	0,77	113,56*

Tabulka 4.10 Porovnání algoritmů na Griewankově funkci v sedmi proměnných

Poznámka Pro $n = 7$ je malá úspěšnost a vysoká doba běhu CLM způsobena častým nedodržením limitu 120 s. Nicméně není možné zvolit menší k, neboť pro $k = 1$ by se vygeneroval jediný bod. Pro přesnější testy ve vysokých dimenzích by bylo vhodné vyměnit způsob generování x_i za nějaký, který je schopen vytvořit N bodů co nejrovnoměrněji rozložených po $[-20; 20]^n$ místo k^n bodů (přesně) rovnoměrně rozložených po $[-20; 20]^n$.

Vidíme, že s rostoucí dimenzí problému se úspěšnost algoritmů (kromě CLM) snižuje. U CLM lze nahodilost výsledků vysvětlit horší volbou η, γ při testování pro $n = 3$, než pro $n = 5$. Náhodný vzorek pro $n \geq 5$ již nebyl schopen najít správné řešení, neboť tvorba dostatečně hustého pokrytí by zabrala více času, než bylo k dispozici.
Törnův algoritmus se ukazuje jako výhodnější přístup než CLM. Je však zajímavé, že CLM má v některých případech větší úspěšnost než TRNA-MS (navíc např. pro $n = 5$ používá TRNA-MS více bodů a stejně je méně úspěšný).
Mohlo by tak být z následujícího důvodu: TRNA-MS najde správné globální minimum, ale není k němu dost blízko, aby byl test považován za úspěšný. To se může stát, neboť k ukončení algoritmu byl použit rozdíl funkčníh hodnot, ne délka posunu. Naopak pokud v CLM jsou všechny body již blízko k sobě, tak v $L_A(\mathbf{x})$ převládne první člen $\frac{\eta}{J} \sum_{j=1}^{J} f(x_j)$. V testu je η nastaveno na 10 nebo 15 a k ukončení CLM je také používán rozdíl funkčních hodnot. Díky tomu je poloha globálního minima určena mnohem přesněji. Navíc jsou na konci všechny x_j zprůměrovány, čímž se poloha ještě zpřesní.
Důvodem proč více bodů nezaručuje úspěch může být snižující se počet dalších lokálních minim Griewankovy funkce při zvyšování dimenze. Griewankova funkce v n proměnných má lokální minima v bodech, kde $cos(\frac{x_i}{\sqrt{i}}) = 1 \ \forall i \in \{1; \ldots; n\}$. Pro zvyšující se i ale $cos(\frac{x_i}{\sqrt{i}}) = 1$ dále a dále od sebe (pro $i \geq 11$ je na $[-20; 20]$ řešením pouze $x_i = 0$). Může tedy nastat situace, kde rozdělení intervalu na více bodů $[-20; 20]^n$ již nepřináší zlepšení úspěšnosti algoritmu.

4.3 Testování algoritmů - závěr

V této kapitole jsme testovali různé algoritmy globální optimalizace na dvojici testovaných funkcí. V prvním testu (náhodné polynomy jedné proměnné) vyšel CLM jako nejhorší z testovaných algoritmů. V druhém testu (Griewankova funkce) dokázal porazit algoritmus používající hrubou sílu (náhodný vzorek) a v některých případech i TRNA-multistart.
Nicméně do tohoto výsledku není započítán čas strávený odhadováním η a γ. Pokud by tento čas měl TRNA-multistart k dispozici, pak by mohl dosáhnout lepších výsledků než CLM. Törnův algoritmus pak CLM překonává při všech testech. Proto se CLM nezdá jako zlepšení oproti zavedeným metodám. Shrneme hlavní důvody:

- Špatný koncept algoritmu
- Neexistující postup určení $\eta; \gamma$
- Výpočetní náročnost $L_A(\mathbf{x})$
- Složitá implementace

Špatný koncept algoritmu: Koncept CLM byl představen v kapitole 3, hlavní ideou je kombinace konvergence každého bodu do lokálního minima a zároveň konvergence bodů k tomu s nejnižší funkční hodnotou. Tato koncepce má ale několik chyb:

Pokud jeden bod „přeskočí“ do okolí jiného, pak je zbytečné dále vyhodnocovat oba, neboť oba konvergují k stejnému lokálnímu minimu. Lepším postupem je tedy tento bod smazat (jako v Törnově algoritmu), nebo nechat v původní pozici a pokračovat s průzkumem zde (jako v TRNA-multistart).
Pokud má účelová funkce více minim s velmi blízkými funkčními hodnotami, tak může snadno převládnout vliv konvergence bodů k sobě a body se seskupí ve špatném lokálním minimu.

Neexistující postup určení $\eta; \gamma$: Při testování polynomů jedné proměnné bylo zmíněno, že není znám postup na určení správných parametrů $\eta; \gamma$. Tyto parametry však mají klíčový vliv na pravděpodobnost nalezení správného řešení. Jednotná volba během testování polynomů se ukázala jako nevhodná. Navíc nelze zvolit nějaké „bezpečné“ $\eta; \gamma$ (viz. [A. Teughels a Suykens 2003, s. 5]), které by zvyšovaly úspěšnost za cenu doby běhu algoritmu. U jiných algoritmů je toto často možné (např. vyšší K pro P-S algoritmus). Stejně tak není garantováno zvýšení úspěšnosti s rostoucím počtem použitých bodů.

Výpočetní náročnost $L_A(\mathbf{x})$: Rozšířená Lagrangeova funkce je složitější na výpočet než samotná účelová funkce f. Pro f, která už sama může být výpočetně náročná je pak CLM pomalejší než jiné algoritmy.

Složitá implementace: Z představených algoritmů je CLM nejsložitější na implementaci kvůli složité kontrukci $L_A(\mathbf{x})$ a použití několika pomocných algoritmů. Tvar $L_A(\mathbf{x})$ není snadno představitelný i pro jednoduché f a málo bodů, díky tomu je složité hledat chyby v implementaci.
Na základě těchto argumentů a výsledku testování nelze doporučit CLM jako vhodný algoritmus pro řešení problému globální optimalizace. TRNA-multistart a Törnův algoritmus se jeví jako výhodnější ve většině situací, navíc jsou mnohem více předvídatelné ve svém chování.

Závěr

V práci byly představeny různé přístupy k problému globální optimalizace, včetně konkrétních algoritmů. Výhody a nevýhody algoritmů byly diskutovány. Důkazy, heuristiky a argumenty pro fungování algoritmů jsou předvedeny. Některá možná zlepšení algoritmů jsou představena.
Algoritmus CLM je podrobně přednesen, včetně pomocných algoritmů, a rozšířené Lagrangeovy funkce.
Dále byly vybrané algoritmy testovány na dvojici účelových funkcí - náhodně volených polynomech v jedné proměnné a Griewankově funkci pro různé dimenze. Algoritmy byly porovnány dle dosažené úspěšnosti a doby běhu.

Seznam literatury

Törn, Aimo a Antanas Žilinskas (1989). *Lecture Notes in Computer Science*. First Edition. Springer-Verlag. ISBN: 3-540-50871-6.

Nocedal, Jorge a Stephen J. Wright (1999). *Numerical Optimization*. First Edition. Springer-Verlag. ISBN: 0-387-98793-2.

Sucháček, Jan (2017). *Numerický výpočet vlastních čísel matice*.

Törn, Aimo (1977). „Cluster Analysis Using Seed Points and Density-Determined Hyperspheres as an Aid to Global Optimization“. In: *IEEE Transactions on Systems, Man, and Cybernetics* 7, s. 610–616.

A. Teughels, G. de Roeck a J.A.K. Suykens (2003). „Coupled Local Minimizers: a new global optimization method“. In: *Proceedings of 11th IFIP Working Conference on Reliability and Optimization of Structural Systems*.

Seznam obrázků

2.1 Lipshitzovský kužel pro $f(x) = sin(x) - \frac{x}{4}; K = \frac{5}{4}, x = 7.$ 10
2.2 Vyloučení množiny kolem $x_2 = 7; f(x) = sin(x) - \frac{x}{4}; K = \frac{5}{4}$ 11
2.3 Vyloučení intervalu v Brentově algoritmu pro $x_1 = 6; x_2 = 8; f(x) = sin(x) - \frac{x}{4}; K = \frac{5}{4}$. 14
2.4 Tunelovací algoritmus s nepřesným lokálním minimem pro $f(x) = sin(x) - \frac{x}{2}$ 27

4.1 Příklad testovacího polynomu 41
4.2 Griewankova funkce jedné proměnné na okolí 0 44

Seznam tabulek

4.1 Výsledky testů CLM pro J = 11 a různé $\eta; \gamma$ 42
4.2 Výsledky testů CLM pro J = 8 a různé $\eta; \gamma$ 42
4.3 Porovnání vybraných algoritmů na náhodných polynomech v jedné proměnné 43
4.4 Volba k pro Náhodný vzorek v různých dimenzích 45
4.5 Volba k pro TRNA-multistart v různých dimenzích 45
4.6 Volba k pro Törnův algoritmus v různých dimenzích 45
4.7 Volba $k; \eta$, γ pro CLM v různých dimenzích 46
4.8 Porovnání algoritmů na Griewankově funkci ve třech proměnných 46
4.9 Porovnání algoritmů na Griewankově funkci v pěti proměnných . 46
4.10 Porovnání algoritmů na Griewankově funkci v sedmi proměnných 46