UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

METODY OPTIMALIZACE BEZ VÝPOČTU DERIVACÍ

Tomáš Hladík

Bakalářská práce

2023

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš Hladík**
Osobní číslo:	**I19182**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Řízení procesů**
Téma práce:	**Metody optimalizace bez výpočtu derivací**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování

Cílem práce je v prostředí MATLAB implementovat min. 3 v principu odlišné algoritmy minimalizace funkcí více parametrů bez výpočtu derivací a objektivně porovnat jejich efektivitu vzájemně a s funkcí *fminsearch* MATLABu. Pro porovnání bude využito několika testovacích problémů uvedených v literatuře. V případě, že algoritmy budou záviset na volitelných parametrech, bude snaha získat závěry o vhodné volbě těchto parametrů. Teoretická část bude obsahovat výklad obecných souvislostí a popis algoritmů využitých v praktické části. Praktická část bude obsahovat příslušné programy a přehledné zpracování experimentálních výsledků.

Rozsah pracovní zprávy: **cca 40 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

CVEJN, J. *Algorithms of optimization* [online]. Pardubice: Univerzita Pardubice, FEI, 2021-. Elektronický studijní materiál k předmětu Algoritmy optimalizace.
PRESS, H., TEUKOLSKY, S.A., VETTERLING, W. T., FLANNERY, B. P. *Numerical Recipes. The Art of Scientific Programming. Third Edition.* New York: Cambridge University Press, 2007. ISBN 0-521-88068-8.
TVRDÍK, J. *Evoluční algoritmy*. Ostrava: Ostravská univerzita, 2004.

Vedoucí bakalářské práce: **doc. Ing. Jan Cvejn, Ph.D.**
Katedra řízení procesů

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Daniel Honc, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 9. ledna 2023

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 20.8.2023

Tomáš Hladík

Poděkování

Úvodem bych rád poděkoval za poskytnutý čas a materiály vedoucímu práce panu doc. Ing. Janu Cvejnovi, Ph.D. Dále jsem vděčný za předané vědomosti od všech učitelů v průběhu bakalářského studia. Nemohu nezmínit obrovskou podporu rodiny a přátel, bez které bych studium na vysoké škole nezvládl.

V Pardubicích dne 20.9.2022

Tomáš Hladík

ANOTACE

Práce je zaměřena na porovnání metod optimalizace bez výpočtu derivací. Konkrétně je zaměřena na porovnání metody flexibilního simplexu, který je implementován v prostředí MATLAB se třemi alternativními algoritmy optimalizace bez výpočtu derivací. Pro porovnání je využito několik testovacích problémů převzatých z literatury.

KLÍČOVÁ SLOVA

metody optimalizace, hledání minima, metody Monte Carlo, diferenciální evoluce

TITLE

DERIVATIVE FREE OPTIMIZATION METHODS

ANNOTATION

The work is focused on a comparison of optimization methods without calculation of derivatives. Specifically, it focused on a comparison of the flexible simplex method implemented in the MATLAB environment with three alternative optimization algorithms without calculation of derivatives. Several test problems taken from the literature are used for the comparison.

KEYWORDS

optimization methods, minimum search, Monte Carlo methods, differential evolution

OBSAH

SEZNAM SYMBOLŮ VELIČIN A FUNKCÍ

f	testovaná funkce
x_0	počáteční bod
n	počet proměnných
N	počet populace
i	počet iterací
d	délka úsečky
k	směrový vektor
U	počet úseček
σ	velikost oblasti
f_{min}	minimum testované funkce
lowerBound	dolní mez hledaného intervalu
upperBound	horní mez hledaného intervalu

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

ÚVOD

Optimalizace funkcí je důležitým problémem v mnoha oblastech, jako jsou průmyslové inženýrství, ekonomie a mnoho dalších. Cílem optimalizace je najít hodnotu nebo množinu hodnot vstupních parametrů funkce, které maximalizují nebo minimalizují její výstupní hodnotu. Klasickou metodou pro řešení optimalizačních problémů je výpočet derivací a hledání extrémů na základě jejich hodnot. Nicméně, v některých případech derivace funkce není k dispozici. V takových případech mohou být použity metody optimalizace bez výpočtu derivací, které se zaměřují na hledání extrémů pomocí pouze výpočtu hodnot funkce v různých bodech. Tato práce popisuje a porovnává několik takových metod a ukazuje, jak mohou být úspěšně použity k řešení optimalizačních problémů.

1 OBLASTI POUŽITÍ OPTIMALIZACE

Optimalizace je klíčová technika, která se používá v mnoha různých oborech, od vědy a techniky po ekonomiku a logistiku. Zde je pět největších oblastí, kde se optimalizace v praxi využívá:

- Průmyslová výroba a logistika: Průmyslové podniky a logistické společnosti využívají optimalizaci pro zlepšení efektivity a snížení nákladů. Mohou například optimalizovat výrobní procesy, aby minimalizovaly spotřebu energie nebo materiálů. V oblasti logistiky se optimalizace používá pro plánování tras dopravy tak, aby byly co nejefektivnější a nejkratší.
- Finanční sektor: Banky a investiční společnosti využívají optimalizaci pro řízení portfolií a rizik. Mohou například hledat optimální kombinaci investic, která maximalizuje očekávaný výnos a zároveň minimalizuje riziko. Optimalizace se také používá pro stanovení cen derivátů a jiných finančních produktů.
- Věda a technika: Vědci a inženýři používají optimalizaci pro řešení různých problémů, od návrhu experimentů po optimalizaci konstrukcí. Například v oblasti strojového učení se optimalizace používá pro nalezení nejlepších parametrů modelu. V oblasti civilního inženýrství se optimalizace může použít pro návrh budov a mostů, které jsou co nejodolnější a nejlevnější.
- Energetika a životní prostředí: Energetické společnosti a environmentální inženýři využívají optimalizaci pro zlepšení efektivity a snížení dopadů na životní prostředí. Mohou například optimalizovat rozmístění větrných turbín nebo solárních panelů pro maximalizaci produkce energie. Optimalizace také může pomoci při návrhu strategií pro snížení emisí skleníkových plynů.

Ať už je to plánování dopravy, správa investičního portfolia, návrh technické infrastruktury, optimalizace výrobních procesů nebo zlepšování zdravotnické péče, optimalizace hraje klíčovou roli v mnoha různých oblastech. Použití těchto technik může vést k výrazným zlepšením v efektivitě a produktivitě, což může mít velký dopad na společnost jako celek.

2 OPTIMALIZAČNÍ PROBLÉMY

Optimalizace je oblastí, která se zabývá nalezením nejlepšího řešení pro daný problém. Cílem optimalizace je minimalizovat nebo maximalizovat hodnotu účelové funkce při splnění určitých omezení. To může zahrnovat hledání nejlepšího designu, nejefektivnějšího procesu nebo optimalizaci různých rozhodovacích problémů.

V tradičních metodách optimalizace se často využívá derivací, které poskytují informace o rychlosti změny účelové funkce. Derivace jsou velmi užitečné, pokud jsou k dispozici a lze je efektivně vypočítat. Avšak v některých případech nejsou derivace funkce k dispozici. Třeba v případech, kdy není funkce diferencovatelná. To může výrazně omezit použitelnost tradičních metod optimalizace založených na derivacích.

Motivace pro vývoj metod optimalizace bez výpočtu derivací vychází z potřeby vyhnout se těmto omezením. Optimalizace bez výpočtu derivací se zaměřuje na vývoj algoritmů a technik, které umožní efektivní hledání optimálního řešení, aniž by vyžadovaly výpočet derivací. Tento přístup umožňuje širší aplikace v praktických problémech a rozšiřuje možnosti optimalizace i pro situace, kdy derivace nejsou k dispozici nebo jsou obtížně vypočitatelné.

3 METODY OPTIMALIZACE BEZ VÝPOČTU DERIVACÍ

Derivace hrají klíčovou roli v tradičních metodách optimalizace. Tyto metody často využívají gradient, což je vektor obsahující parciální derivace cílové funkce vzhledem k jednotlivým proměnným. Gradient ukazuje směr největšího růstu.

Hessián je matice obsahující parciální derivace druhého řádu cílové funkce. Tato informace je důležitá pro určení, zda se jedná o lokální minimum, maximum nebo sedlový bod. Problémem je, že výpočet Hessiánu může být složitý a časově náročný, zejména pokud cílová funkce závisí na velkém počtu proměnných.

Omezení spojená s výpočtem derivací vedla k rozvoji alternativních přístupů. Metody optimalizace bez výpočtu derivací se snaží najít optimální řešení pouze pomocí hodnot funkce samotné, aniž by vyžadovaly výpočet derivací. Tímto způsobem lze výrazně rozšířit možnosti a aplikace optimalizace na problémy, které by jinak byly obtížné nebo nemožné řešit tradičními metodami založenými na výpočtu derivací.

Potřeba optimalizace bez výpočtu derivací vychází z několika faktorů a specifických situací, ve kterých tradiční metody založené na derivacích dosahují svých limitů. Zde jsou některé důvody, proč je potřeba využívat metody optimalizace bez výpočtu derivací:

- Nedostupnost derivací: Existují problémy, u kterých je obtížné nebo nemožné získat analytický tvar derivací cílové funkce. To se může vyskytovat při optimalizaci složitějších systémů, jako jsou simulační modely, nebo u funkcí s nespojitostmi, skoky nebo singularitami.
- Výpočetní náročnost: Výpočet derivací může být časově náročný, zejména pro složité a velké systémy s mnoha proměnnými. Tradiční metody optimalizace založené na derivacích vyžadují výpočet gradientu, Hessiánu nebo jiných derivací, což může být velice nepraktické a neefektivní.
- Robustnost a průzkum prostoru řešení: Metody optimalizace založené na výpočtu derivací jsou náchylné k uvíznutí v lokálních extrémech a nemají schopnost prozkoumat celý prostor řešení. Metody bez výpočtu derivací, jako jsou genetické algoritmy, rojové algoritmy nebo simulované ochlazování, jsou schopné vyhledávat globální extrémy a překonávat lokální extrémy.
- Flexibilita a univerzálnost: Metody optimalizace bez výpočtu derivací jsou obecnější a flexibilnější než metody založené na derivacích. Tyto metody mohou být použity pro

širokou škálu problémů, včetně problémů s diskrétními proměnnými, nediferencovatelnými funkcemi, kombinatorickými problémy nebo problémy s omezeními.

Přestože metody optimalizace bez výpočtu derivací nabízejí mnoho výhod, je důležité si být vědom také některých omezení a výzev spojených s použitím metod optimalizace bez výpočtu derivací:

- Větší počet vyhodnocení cílové funkce: Metody bez výpočtu derivací obvykle vyžadují vyšší počet vyhodnocení cílové funkce než tradiční metody založené na výpočtu derivací. To může být náročné pro problémy s vysokým počtem proměnných nebo složitými funkcemi, které vyžadují mnoho iterací pro dosažení optimálního řešení.
- Potřeba vhodně volených parametrů: Metody optimalizace bez výpočtu derivací často zahrnují různé parametry, jako je velikost populace, pravděpodobnost mutace nebo teplotní schéma. Volba vhodných parametrů může vyžadovat experimentování a ladění, což může komplikovat využití.
- Lokální vs. globální optimalizace: I když metody bez výpočtu derivací jsou obecně robustnější v hledání globálního extrému než metody založené na derivacích, mohou mít stále potíže s nalezením globálního extrému v některých složitých problémech. Je tedy důležité provést dostatečné porovnání a vyhodnocení různých metod, aby se získalo nejlepší řešení.
- Přes tato omezení jsou metody optimalizace bez výpočtu derivací silným nástrojem pro řešení různých optimalizačních problémů, zejména pak těch s nediferencovatelnými nebo diskrétními funkcemi. Jejich využití přináší nové možnosti pro efektivní a robustní optimalizaci s širokou škálou aplikací.

4 TYPY ALGORITMŮ OPTIMALIZACE BEZ VÝPOČTU DERIVACÍ

Metody optimalizace bez výpočtu derivací zahrnují širokou škálu technik a algoritmů, které se zaměřují na hledání optimálního řešení bez nutnosti výpočtu derivací cílové funkce. Následuje přehled základních metod. Tři konkrétní algoritmy jsou popsány podrobně v kap. 5 - 7.

4.1 Simulované žíhání

Simulované žíhání je heuristická optimalizační metoda, která je inspirována fyzikálním procesem žíhání, kde pevná látka je zahřívána a poté pomalu ochlazována s cílem snížit její energetický potenciál.

Metoda Simulovaného žíhání (Simulated Annealing, SA) je zejména užitečná pro optimalizaci funkcí s velkým množstvím lokálních extremů, kde je riziko uvíznutí v lokálním minimu nebo maximu. Při SA se nejprve náhodně vybere počáteční řešení. V každém kroku se náhodně generuje nové řešení v okolí stávajícího řešení. Pokud je nové řešení lepší (má nižší hodnotu cílové funkce), je přijato. Pokud je horší, je přijato s pravděpodobností, která závisí na rozdílu mezi hodnotami cílové funkce a na aktuální teplotě. Tato teplota postupně klesá v průběhu času podle předem daného schématu chlazení.

Tato metoda byla použita v širokém spektru aplikací, včetně optimalizace rozvrhů a problému obchodního cestujícího. SA je však značně pomalý algoritmus a může vyžadovat velké množství výpočetního času, zejména pro rozsáhlé problémy. Existují různé techniky pro zrychlení SA, včetně paralelního simulovaného žíhání a rychlého simulovaného žíhání.

4.2 Metoda flexibilního simplexu

Metoda Flexibilního simplexu je jednou z efektivních metod pro optimalizaci funkce. Je již napsána, jako funkce v prostředí MATLAB. Metoda flexibilního simplexu je modifikovaná verze klasické metody simplexu, která umožňuje flexibilnější pohyb v prostoru řešení.

Metoda Flexibilního simplexu využívá geometrického přístupu k řešení optimalizačních problémů. Je založena na konceptu jednoduchého geometrického útvaru nazývaného simplex, který je obecně polygonem v prostoru n-rozměrných proměnných. V každém kroku metody je simplex přizpůsobován na základě hodnoty cílové funkce v jeho vrcholech.

Flexibilní simplex se od klasického simplexu liší tím, že se může deformovat podle tvaru účelové funkce. To je dosaženo přidáním dodatečných pravidel pro způsob, jakým se simplex upravuje. Tyto pravidla se zakládají na analýze aktuálního stavu simplexu a hodnot účelové funkce. Metoda flexibilního simplexu tak poskytuje vyšší míru flexibility při pohybu v prostoru řešení, což může vést k rychlejší konvergenci k optimálnímu řešení. (Nemhauser, G. L., Wolsey L. A., 1988)

4.3 Náhodné prohledávání

Náhodné prohledávání (anglicky random search) je základní algoritmický přístup pro optimalizaci funkce. Tento přístup je zvláště užitečný v situacích, kdy je funkce nelineární, nedierencovatelná nebo vysokodimenzionální.

Algoritmus náhodného prohledávání spočívá v generování náhodných bodů v daném prostoru a hodnocení funkčních hodnot těchto bodů. Pokud je nově generovaný bod lepší než aktuálně nejlepší známý bod, aktualizuje se nejlepší známý bod. Tento proces se opakuje až do dosažení určitého kritéria ukončení, jako je například maximální počet iterací, nebo dokud se nezlepší kvalita řešení o určitý předem stanovený úsek (Tvrdík, 2010).

Náhodné prohledávání je zvláště účinné pro funkce s mnoha lokálními extrémy, kde gradientní metody často uvíznou v lokálních extrémech a nedosáhnou globálního extrému. Navíc, náhodné prohledávání je zvláště užitečné, když není k dispozici žádná další informace o funkci.

Nicméně, náhodné prohledávání má také některé nevýhody. Především, pro velmi velké prostory může být tato metoda velmi neefektivní, protože pravděpodobnost nalezení extrému je velmi malá. Další nevýhodou je, že náhodné prohledávání nevyužívá žádné informace z předchozích iterací, což může vést k opakovanému prohledávání stejných oblastí prostoru.

Přestože je náhodné prohledávání jednoduchým a základním přístupem k optimalizaci, je stále široce používané a slouží jako základ pro mnoho dalších sofistikovanějších metod, jako jsou evoluční algoritmy a optimalizace pomocí roje částic (Tvrdík, 2010).

4.4 Genetické algoritmy

Genetické algoritmy (GA) jsou heuristické optimalizační techniky založené na principu evoluce a přírodního výběru. Představil je americký vědec John Henry Holland v roce 1975 (Holland, 1975). Tato metoda je široce používána pro hledání optimálních řešení v komplexních problémech.

GA fungují na základě iterativního procesu, který se skládá z několika kroků. Nejprve je vytvořena počáteční populace potenciálních řešení. Každé řešení je reprezentováno jako chromozom, který je obvykle bitový řetězec nebo pole čísel. Kvalita každého řešení je ohodnocena pomocí hodnoty účelové funkce. Poté následuje proces výběru, křížení a mutace. V procesu výběru jsou preferována lepší řešení. Křížení kombinuje dvě řešení a produkuje nová řešení, která zdědí části obou rodičů. Mutace náhodně mění malou část řešení, aby zajišťovala diverzitu v populaci. Tento proces se opakuje několikrát až do splnění určitého kritéria zastavení, jako je maximální počet generací nebo dosažení požadované hodnoty účelové funkce. Je důležité poznamenat, že výběr správných parametrů GA, jako je velikost populace, pravděpodobnost křížení a mutace, může výrazně ovlivnit výsledky optimalizace. Někdy může být vhodné použít adaptivní genetické algoritmy, které mohou dynamicky upravovat tyto parametry během procesu evoluce.

4.5 Diferenciální evoluce

Diferenciální evoluce (DE - Differential Evolution) v principu vychází z genetických algoritmů, ale na rozdíl od nich je vhodná pro spojitou optimalizaci (tj. když parametry jsou spojité), zatímco genetické algoritmy jsou vhodné spíše pro diskrétní optimalizaci. Byla poprvé představena v roce 1995 Stornem a Pricem.

Základem DE je jednoduchý evoluční algoritmus, který používá mechanismy mutace, křížení a výběru, ale implementované v prostoru spojitých parametrů. Mutace v DE je založena na rozdílech mezi jedinci populace, odtud také název algoritmu. Díky tomu je DE schopna robustně a efektivně prozkoumávat prostor řešení. DE pracuje s populací vektorů, které reprezentují možná řešení optimalizačního problému. V každé generaci se jednotliví členové populace mění a vyvíjejí. To vede k vytvoření tzv. mutantního vektoru. Křížení v DE je proces, kde každý prvek mutantního vektoru má určitou pravděpodobnost, že bude nahrazen prvkem z původního vektoru. Výběr v DE je založen na principu přežití nejlepšího. Pokud má nový vektor nižší hodnotu účelové funkce než původní, nahradí starý vektor v populaci (Price et al. 2006).

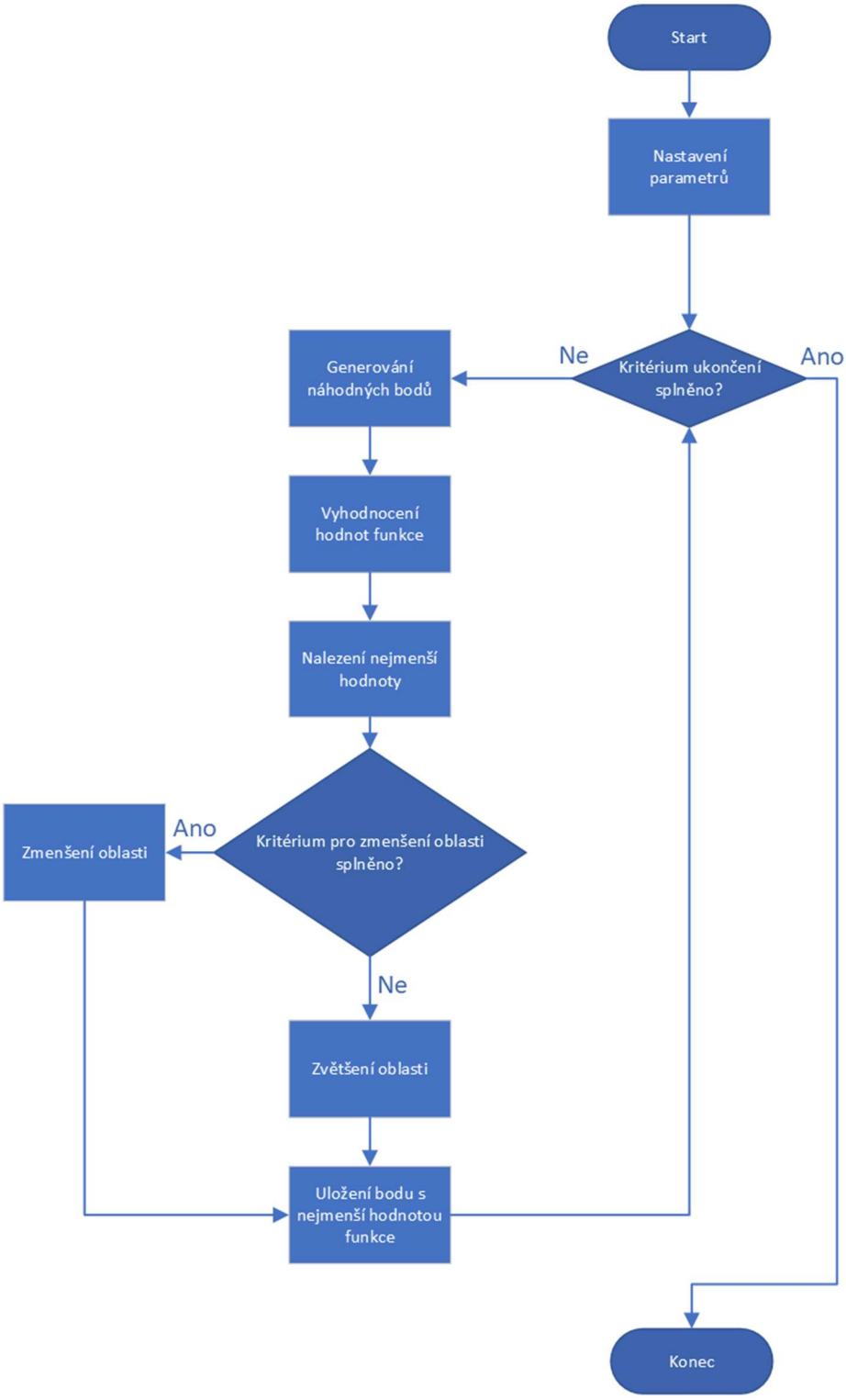
Diferenciální evoluce je robustní, jednoduchá na implementaci a má málo nastavitelných parametrů, což z ní dělá velmi atraktivní metodu pro řešení mnoha druhů optimalizačních problémů.

5 PRINCIP METODY GENEROVÁNÍ NÁHODNÝCH BODŮ

Jednotlivé kroky jsou pro lepší představu znázorněný vývojovým diagramem na Obr. 5.1.

1. Nastavení parametrů**:** Nejprve se nastaví nezbytné parametry a uloží do proměnných. Počet iterací, počet náhodných bodů vygenerovaných v jedné iteraci, testovací funkce, počáteční bod hledání, počet proměnných v testovací funkci a velikost oblasti, ve které se generují náhodné body. Případně se nastaví i parametr přesnosti výsledku.
2. V případě, že počet iterací zatím nedosáhl nastavené hodnoty, nebo hodnota funkce není dostatečně přesná, opakuj následující kroky:
    a. Generování náhodných bodů**:** Vygeneruje se N bodů s rozdělením $N(\mathbf{x_0}, \sigma^2)$, kde $\mathbf{x_0}$ je počáteční bod hledání a σ odpovídá rozsahu prohledávané oblasti.
    b. Vyhodnocení hodnot funkce**:** Vypočítá se hodnota účelové funkce pro každý náhodný bod.
    c. Nalezení nejmenší hodnoty**:** Vyhledá se bod, pro který je hodnota účelové funkce nejmenší
    d. Kritérium pro zmenšení a zvětšení oblasti**:** Pokud je N/5 bodů s menší, nebo větší hodnotou účelové funkce než hodnota účelové funkce původního počátečního bodu, provede se následující:
        - Zmenšení oblasti: Oblast ve které se generují body se násobí zvolenou hodnotou menší než 1.
    e. Zvětšení oblasti: Oblast ve které se generují body se násobí zvolenou hodnotou větší než 1.
        - Uložení bodu s nejmenší hodnotou funkce**:** Bod s nejmenší hodnotou účelové funkce se ukládá, jako nový bod $\mathbf{x_0}$.

Pokud kritérium ukončení hlavního cyklu bylo splněno, tak se algoritmus ukončí.



Obrázek 5.1 – Vývojový diagram – Generování náhodných bodů

6 PRINCIP METODY GENEROVÁNÍ NÁHODNÝCH ÚSEČEK

Princip metody generování úseček je následující:

1. Nastavení parametrů**:** Nejprve se nastaví nezbytné parametry a uloží do proměnných. Počet iterací, testovací funkce, počáteční bod hledání, počet proměnných v testovací funkci a vzdálenost, na které se generují body. Případně se nastaví i parametr přesnosti výsledku.
2. V případě, že počet iterací zatím nedosáhl nastavené hodnoty, nebo hodnota funkce není dostatečně přesná, opakuj následující kroky:
    a. Generování úsečky**:** Vygenerují se body úsečky U podle vztahu:

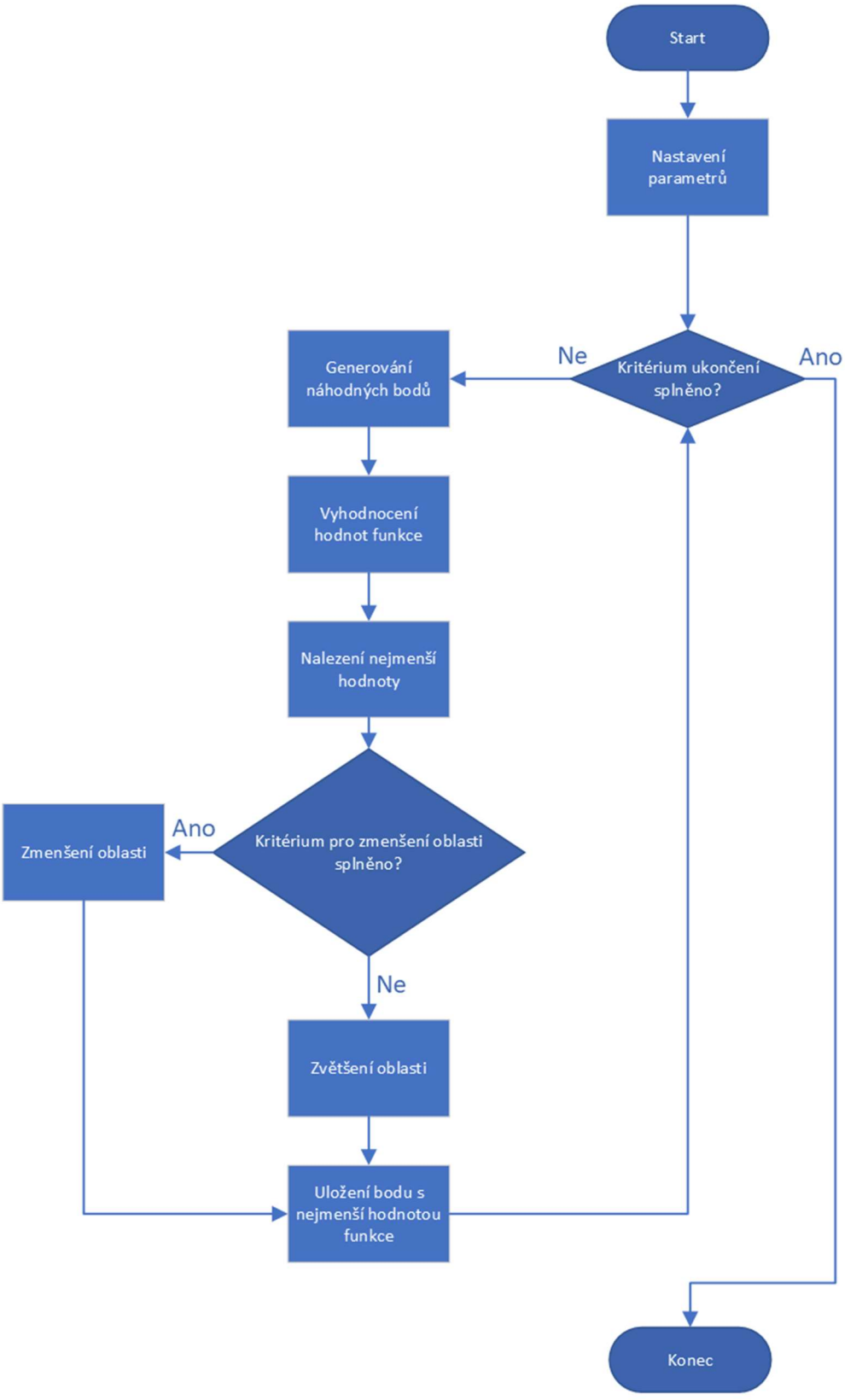
$$\boldsymbol{U} = (\boldsymbol{x_0} + t \times \boldsymbol{s} \times d)$$

kde $\mathbf{x_0}$ je počáteční bod hledání, t = -1, 0, 1, 2 je vektor s hodnotami pozic bodů, $\boldsymbol{s}$ je směrový vektor s náhodnou hodnotou a d je vzdálenost mezi body.

    b. Vyhodnocení hodnot funkce**:** Vypočítá se hodnota účelové funkce pro body ležící na úsečce.
    c. Nalezení nejmenší hodnoty**:** Vyhledá se bod, pro který je hodnota účelové funkce nejmenší
    d. Kritérium pro zmenšení vzdálenosti: Pokud se n-krát nenalezne hodnota menší, vzdálenost se násobí vhodnou hodnotou menší než 1.
    e. Uložení bodu s nejmenší hodnotou funkce: Bod s nejmenší hodnotou účelové funkce se ukládá, jako nový počáteční bod hledání.

Pokud kritérium ukončení hlavního cyklu bylo splněno, tak se algoritmus ukončí.

Jednotlivé kroky jsou pro lepší představu znázorněný na vývojovém diagramu níže na Obr. 6.1.



Obrázek 6.1 – Vývojový diagram – Generování náhodných úseček

7 PRINCIP DIFERENCIÁLNÍ EVOLUCE

Princip metody diferenciální evoluce je následující:

1. Nastavení parametrů: Nejprve se nastaví nezbytné parametry a uloží do proměnných. Počet iterací, testovací funkce, spodní a horní mez pro vygenerování první populace bodů, počet proměnných v testovací funkci. Případně se nastaví i parametr přesnosti výsledku.
2. Inicializace populace: Vytvoří se matice **X** o velikosti N × n a naplní se hodnotami bodů pomocí vztahu:

$$\boldsymbol{X} = \boldsymbol{a} + (\boldsymbol{b} - \boldsymbol{a}) \otimes \boldsymbol{r}$$

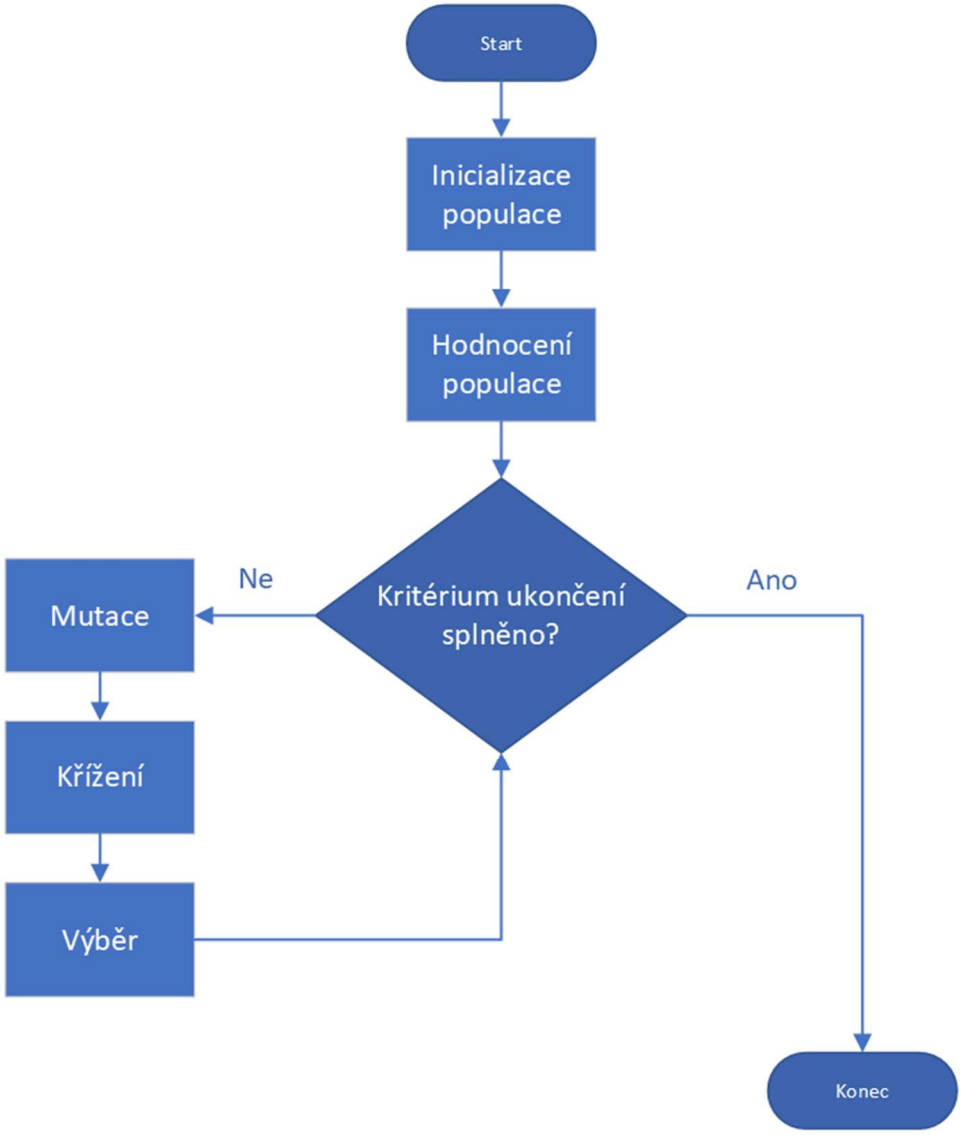
   kde r je matice o velikosti $N \times n$ s náhodnými hodnotami a vektory **a, b** jsou horní a spodní hranice hledání. Znak ⊗ označuje násobení vektorů po složkách, např. **c=a*b** znamená $c_j=a_j \, . \, b_j$ pro všechny složky vektorů **a, b** a **c**, které jsou stejné velikosti.
3. Hodnocení populace: Pro každé bod v populaci se vyhodnotí hodnota účelové funkce.
4. Pokud není splněn maximální počet iterací nebo dostatečně nízká hodnota účelové funkce, opakuj následující kroky:
   a. Mutace: Pro každé řešení v populaci vytvoř mutované řešení pomocí kombinace tří jiných náhodně vybraných řešení $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$ podle vztahu:

$$\boldsymbol{u} = \boldsymbol{r_1} + F(\boldsymbol{r_2} - \boldsymbol{r_3})$$

      kde $F > 0$ je zvolený parametr mutace.
   b. Křížení: Pro každé mutované řešení se kombinují jeho složky s původním řešením, čímž se vytvoří potenciálně nové řešení.
   c. Výběr: Pro každý pár původních a potenciálně nových řešení se porovnávají jejich hodnoty účelové funkce a ponechá se v populaci to s hodnotou menší.

   Pokud kritérium ukončení hlavního cyklu bylo splněno, algoritmus se ukončí.

Jednotlivé kroky jsou pro lepší představu znázorněný na vývojovém diagramu níže na Obr. 7.1.



Obrázek 7.1 – Vývojový diagram – Diferenciální evoluce

8 TESTOVACÍ FUNKCE

Testovací funkce jsou matematické funkce navržené speciálně pro testování a porovnávání výkonu algoritmů pro optimalizaci. Tyto funkce mají obvykle určité vlastnosti, které umožňují analyzovat chování algoritmů a jejich schopnost různé problémy řešit.

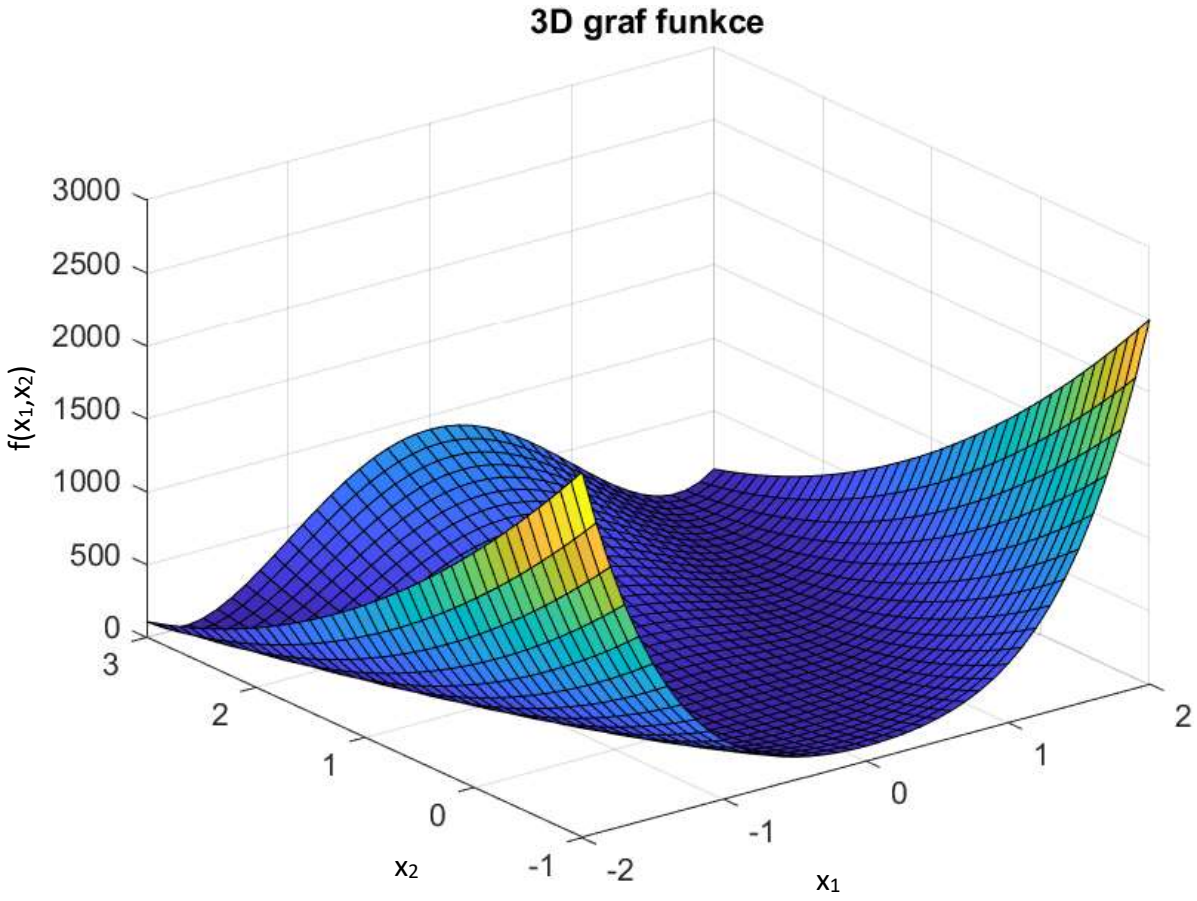
Pro porovnání vlastností optimalizačních metod bylo použito několik testovacích problémů z článku (MORÉ, Jorge J., Burton S. GARBOW a Kenneth E. HILLSTROM, 1981).

8.1 Rosenbrockova funkce

Tato testovací funkce, známá jako Rosenbrockovo banánové údolí, je nelineární optimalizační problém, který je často používán k hodnocení výkonnosti optimalizačních algoritmů. Funkce byla poprvé představena Howardem H. Rosenbrockem v roce 1960.

Funkce je velice populární v oblasti optimalizace, protože její tvar a vlastnosti poskytují dobrý test pro optimalizační algoritmy. Má nesymetrický tvar a úzké údolí (viz. Obrázek 8.1 pro n=2) s globálním minimem, což činí její optimalizaci náročnou. Počáteční bod pro hledání minima funkce je možné zvolit jako: $\mathbf{x_0}$ = (0, 0, …, 0). Minimum funkce je 0 a bod ve kterém se minimum nachází je $\mathbf{x_{min}}$ = (1, 1, …, 1).

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \tag{8.1}$$



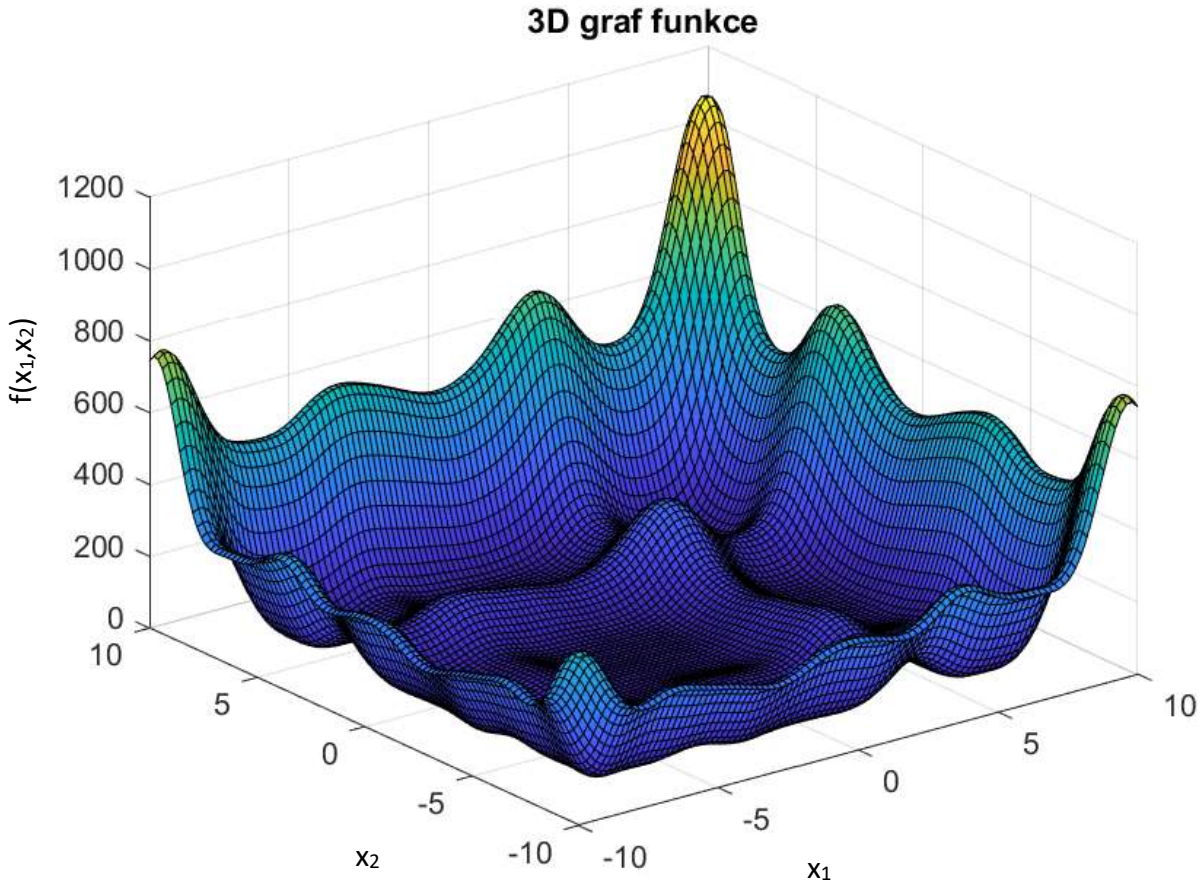
Obrázek 8.1 – Graf Rosenbrockovy funkce pro n=2

8.2 Trigonometrická funkce

Trigonometrické funkce se využívají k popisu vztahů v pravoúhlých trojúhelnících a pro modelování periodických jevů. Tato trigonometrická funkce má minimum funkce 0 v bodě $\mathbf{x_{min}} = (0, 0, \ldots, 0)$.

Počáteční bod pro hledání minima funkce je doporučeno volit následovně: $\mathbf{x_0} = (1/n, 1/n, \ldots, 1/n)$.

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \left[n - \left(\sum_{j=1}^{n} \cos x_j \right) + i\left(1 - \cos x_j\right) - \sin x_j \right]^2 \tag{8.2}$$



Obrázek 8.2 – Graf trigonometrické funkce pro n=2

8.3 Dvoubodový okrajový problém

Dvoubodový okrajový problém je problém řešení diferenciální rovnice, kde jsou zadány dvě okrajové podmínky, tedy podmínky, které jsou zadány na obou koncích intervalu, na kterém se rovnice řeší. Funkce dvoubodového okrajového problému vyjadřuje chybu řešení rovnice v daném počtu bodů, při splnění okrajových podmínek. Tato funkce má minimum v bodě $\mathbf{x}_{\mathbf{min}} = (0, 0, \ldots, 0)$. Počáteční bod pro hledání minima funkce použijeme: $\mathbf{x}_0 = (1, 1, \ldots, 1)$. Zvolená diferenciální rovnice je

$$f''(\boldsymbol{x}) = \frac{1}{2}(y + t + 1) \tag{8.3}$$

kde y(0) = y(1) = 0.

9 PROSTŘEDÍ MATLAB

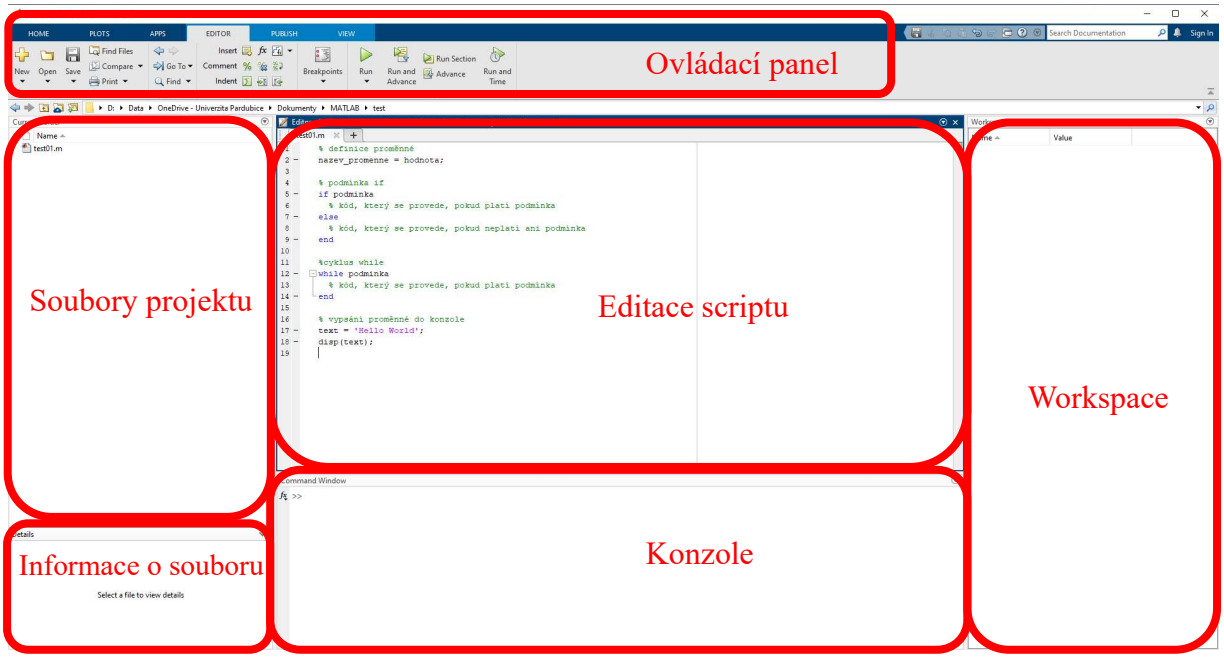
MATLAB je vysokoúrovňový programovací jazyk a prostředí určené pro numerické výpočty, analýzu dat, vizualizaci a vývoj algoritmů. Byl vyvinut firmou MathWorks a je široce využíván v akademickém prostředí, výzkumu a průmyslu.

MATLAB je obzvláště silný v manipulaci s maticemi a vektorovými operacemi, což ho činí vhodným nástrojem pro řešení matematických a technických problémů. Jeho syntaxe je podobná klasickým programovacím jazykům, což usnadňuje uživatelům přechod z jiných programovacích prostředí.

Existuje mnoho oblastí, ve kterých se porostředí MATLAB používá:

- Matematické modelování a simulace: MATLAB umožňuje vytváření matematických modelů a jejich simulaci. To je užitečné v oblastech jako je fyzika, matematika, inženýrství a ekonomie.
- Analýza a vizualizace dat: MATLAB poskytuje širokou škálu funkcí pro analýzu a vizualizaci dat. Umožňuje importovat, zpracovávat a analyzovat data z různých zdrojů. Lze ho využít pro statistickou analýzu, tvorbu grafů, vyhlazování dat, rozpoznávání vzorů a mnoho dalšího.
- Algoritmy a programování: MATLAB je vhodný pro vývoj algoritmů a implementaci numerických metod. Lze ho použít k implementaci a testování algoritmů pro zpracování obrazu, signálů, optimalizaci a strojové učení.
- Základní ovládání MATLABu zahrnuje práci v příkazovém okně nebo ve skriptech, kde lze psát a spouštět příkazy. MATLAB obsahuje rozsáhlou knihovnu funkcí a toolboxů, které se používají k řešení konkrétních problémů.

Zde je náhled prostředí MATLAB a popis základních prvků:



Obrázek 9.1 – Prostředí MATLAB

9.1 Funkce fminsearch

Funkce fminsearch je v MATLABu implementovaná metoda flexibilního simplexu pro hledání minima nelineární funkce.

Syntaxe funkce fminsearch je patrná z obr. 9.2.

```
[x, fval] = fminsearch(fun, x0)
[x, fval] = fminsearch(fun, x0, options)
```

Obrázek 9.2 – Syntaxe fminsearch

Parametr fun je odkaz na uživatelem definovanou funkci, která je minimalizována. Tato funkce musí přijímat jediný vstupní argument, a to vektor x, který představuje aktuální bod v prohledávaném prostoru. Funkce fun vrací hodnotu účelové funkce v daném bodě.

Parametr x0 je vektor, který představuje počáteční bod, ze kterého začne fminsearch vyhledávat minimum. Parametr options, který umožňuje nastavit parametry optimalizace, je popsán dále v kap. 9.2

Po spuštění fminsearch algoritmus postupně iteruje v prohledávaném prostoru a hledá minimum účelové funkce. Během každé iterace se snaží upravit aktuální bod x tak, aby snížil hodnotu účelové funkce.

Po dosažení konvergence, tj. když algoritmus dosáhne dostatečně blízko k potenciálnímu minimu, fminsearch vrátí nalezený bod x a hodnotu účelové funkce fval v tomto bodě.

Funkce fminsearch je vhodná pro relativně malé rozměry prohledávaného prostoru a pro účelové funkce, které neobsahují příliš mnoho lokálních minim. V případě složitějších úloh nebo funkcí s výraznými nelinearitami může být vhodné použít jiné pokročilejší metody, které nabízí Optimization Toolbox v MATLABu, jako například funkce fmincon pro problémy s omezeními.

9.2 Parametr options

Parametr options funkce fminsearch umožňuje specifikovat různá nastavení algoritmu pro hledání minima. Tento parametr je volitelný a umožňuje uživateli upravit chování fminsearch podle svých potřeb. Zde je detailní popis některých základních možností a hodnot parametru options:

options.Display: Tento parametr ovlivňuje zobrazení informací během běhu algoritmu. Hodnoty jsou:

'final' (výchozí hodnota): Zobrazuje pouze výsledky po ukončení algoritmu.

'iter': Zobrazuje informace po každé iteraci algoritmu.

'off': Nezobrazuje žádné informace.

options.MaxIter: Tento parametr určuje maximální počet iterací, které fminsearch provede, než ukončí hledání minima. Výchozí hodnota je 400.

options.MaxFunEvals: Určuje maximální počet vyhodnocení účelové funkce, které fminsearch provede, než ukončí hledání minima. Výchozí hodnota je 100*numberOfVariables, kde numberOfVariables je počet proměnných.

options.TolX: Tento parametr určuje toleranci pro ukončení algoritmu na základě změny hodnoty proměnných. Pokud je absolutní změna hodnoty všech proměnných menší než TolX, algoritmus je zastaven. Výchozí hodnota je $1e^{-4}$.

options.TolFun: Určuje toleranci pro ukončení algoritmu na základě změny hodnoty účelové funkce. Pokud je absolutní změna hodnoty účelové funkce menší než TolFun, algoritmus je zastaven. Výchozí hodnota je $1e^{-4}$.

options.PlotFcns: Parametr umožňuje uživateli zadat funkce, které budou volány po každé iteraci algoritmu. Tyto funkce mohou sloužit k zobrazení postupu hledání minima. Například options.PlotFcns = @optimplotfval zobrazí vývoj hodnoty účelové funkce v průběhu hledání minima.

Pro nastavení hodnot parametru options je třeba vytvořit strukturu, která obsahuje příslušné pole a hodnoty. Příklad je na obr. 9.3.

```
options = optimset('Display', 'iter', 'MaxIter', 1000);
```

Obrázek 9.3 – Syntaxe options

Příkaz na obr. 9.3 vytvoří strukturu options a nastaví zobrazení informací při každé iteraci algoritmu a maximální počet iterací na 1000.

PRAKTICKÁ ČÁST

V této části práce jsou prezentovány skripty s komentářem napsané v prostředí MATLAB pro optimalizaci pomocí algoritmů metody náhodného generování bodů, metody náhodného generování úseček a Diferenciální evoluce, které jsou popsané v kap. 5-7. Tyto metody jsou porovnány s funkcí fminsearch, která je k dispozici v porostředí MATLAB.

Pro porovnání výkonnosti algoritmů je využita Rosenbrockova funkce, trigonometrická funkce a dvoubodový okrajový problém, které jsou popsané v kapitole 8. Testovací problém je v jednotlivých skriptech nastaven konstantou f (1 až 3).

Kapitola 10 obsahuje jednotlivé algoritmy vytvořené v MATLABu s komentářem pro lepší pochopení a v kapitole 11 jsou tyto algoritmy porovnány pomocí grafu a poté podle počtu vyhodnocení účelové funkce pro zvolenou přesnost.

Výpočty byly provedeny na přenosném počítači s procesorem i7 6. generace od výrobce Intel s operační pamětí 8GB a na počítači je nainstalován 64bitový operační systém Windows 10 od firmy Microsoft.

10 PROGRAMOVÁ REALIZACE OPTIMIZAČNÍCH METOD

V této kapitole je podrobně vysvětlen kód jednotlivých algoritmů optimalizačních metod, které jsou součástí praktické části této práce.

10.1 Kód metody generování náhodných bodů

Nejprve je potřeba definovat a nastavit hodnoty proměnným, které jsou nutné pro správný chod scriptu - viz obr. 10.1.

```
% Nastavení parametrů
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f = 3; % typ funkce
            %1 Rosenborckova funkce
            %2 Trigonometrická funkce
            %3 Dvoubodový okrajový problém
x0 = [1,1,1]; % počáteční bod hledání
n = length(x0); % počet proměnných
N = 200; % počet náhodných bodů
i = 100000; % Nastavení počtu iterací
presnost = 1e-4; %1e-4
S = 0.5; % velikost oblasti hledání
info = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Obrázek 10.1 – Definice proměnných

Poté je definována globální proměnná `callCount`, která se využívá pro počítání volání funkce, ve které je hledáno minimum. Tato proměnná je důležitá pro pozdější porovnávání metod. Proměnná `area` vytvoří vektor, který bude pomáhat vytvářet oblast ve které se boudou generovat body. Proměnná oldMinF je nastavená na vysokou hodnotu, protože je využívána pro pozdější porovnávání nejlepšího nalezeného minima funkce - viz obr. 10.2.

```
global callCount
callCount = 0;
area = ones(1, n) * S;
```

Obrázek 10.2 – Definice globální proměnné a oblasti

Následně přichází hlavní smyčka scriptu, která se vykonává dokud proměnná i s informací o počtu maximálních iterací je větší než 0, nebo bude splněna podmínka s přesností výsledku minimalizace účelové funkce. Od této proměnné se na konci hlavní smyčky odčítá pokaždé -1. V hlavní smyčce se prvně připraví pole X s náhodnými hodnotami o velikosti N

(počet bodů) a n (počet proměnných funkce). Následně se do pole X uloží i počáteční bod x0 - viz obr. 10.3.

```
while i >= iNow && minF > presnost
    % Generování náhodných bodů
    X = randn(N,n);

    X = X.* area + x0; % posun do počátečního bodu hledání
    X(1, :) = x0;
```

Obrázek 10.3 – Hlavní cyklus a definice pole X

V této části scriptu se definuje pole F, které slouží pro ukládání jednotlivých hodnot funkce pro vygenerované body a nastaví své hodnoty na hodnotu „0“. Následně se opakuje cyklus for dokud pole F nebude mít všechny hodnoty funkce pro dané body. Funkce switch přijímá hodnotu proměnné f a určuje pro jakou testovací funkci je hodnota počítána. Poté už se jen do pole F ukládá daná hodnota funkce na určitou pozici - viz obr. 10.4.

```
F = zeros(N,1);
for j = 1:N
    switch f
        case 1
            F(j) = BP_rosen(X(j,:));
        case 2
            F(j) = BP_trigon(X(j,:));
        case 3
            F(j) = BP_dvoubodovy_okrajovy_problem(X(j,:));
    end
end
```

Obrázek 10.4 – funkce switch

Funkce min() hledá nejmenší hodnotu v poli F a ukládá jí do proměnné minF a proměnná minIndex ukládá pozici na kterém se minimální hodnota nachází. Proměnná x0 ukládá bod ve kterém se nejlepší hodnota funkce nachází - viz obr. 10.5.

```
[minF, minIndex] = min(F);
x0 = X(minIndex,:);
```

Obrázek 10.5 – Hledání nejmenší hodnoty funkce

Níže na obrázku 10.6 je algoritmus pro úpravu oblasti ve které se generují náhodné body. Cyklus for prohledá všechny hodnoty funkce ve vygenerovaných bodech a porovnává je s nejlepší hodnotou funkce z minulé iterace v hlavním cyklu. Po každé lepší hodnotě přičte

k proměnné locSearch +1. Když je lepších hodnot víc než jedna pětina všech bodů, tak se oblast násobí hodnotu 0,8, když je lepších bodů méně, tak se oblast zvětší násobením 1,15.

```
locSearch = 0;
for j = 1:length(F)
    if F(j) < oldMinF
        locSearch = locSearch + 1;
    end
end

if locSearch < 5/N
    area = area* 0.8;
else
    area = area* 1.15;
end
```

Obrázek 10.6 – Úprava oblasti hledání

Nejmenší hodnota funkce této populace bodů se ukládá do proměnné oldMinF pro budoucí porovnání v dalším kole cyklu. Proměnná i se zmenší o hodnotu -1, aby nedošlo k zacyklení hlavní smyčky a ukončí se hlavní smyčka cyklu while značkou end - viz obr. 10.7.

```
    oldMinF = minF;
    i = i - 1;
end
```

Obrázek 10.7 – Ukládání nejmenší hodnoty

Na konec se do konzole vypíše bod, který je nejblíže minimu funkce a hodnota funkce v tomto bodě. Dále se vypíše čas trvání a počet výpočtů funkce pro porovnání s ostatními metodami.

10.2 Kód metody generování náhodných úseček

Tento algoritmus je podobný algoritmu generování náhodných bodů, ale jsou zde výrazné změny, a proto bude vysvětlen i tento algoritmus. Nejprve je potřeba definovat a nastavit hodnoty proměnným, které jsou nutné pro správný chod scriptu - viz obr. 10.8.

```
% Nastavení parametrů
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f = 2; % typ funkce
            %1 Rosenborckova funkce
            %2 Trigonometrická funkce
            %3 Dvoubodový okrajový problém
x0 = [1,1,1]; % počáteční bod hledání
d = 0.01; % vzdálenost mezi body v úsečce
i = 10000000; % Nastavení počtu iterací
presnost = 1e-4; %1e-4
info = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
minF = inf;
oldMinF = inf;
iNow = 1;
locSearchReduction = 0;
```

Obrázek 10.8 – Definice proměnných

Poté se do proměnné k uloží vektor směru úsečky. Do proměnné N se uloží hodnota počtu hodnot v proměnné k a do proměnné n se uloží hodnota počtu hodnot vektoru, ve kterém je uložený počáteční bod hledání x0. Dále se vytváří pole U o rozměrech N krát n. Následuje hlavní smyčka, ve které se nejprve vygeneruje náhodný směr a poté se v cyklu for vytvoří samotná úsečka, která obsahuje počáteční bod hledání x0 a je násobena náhodným směrem randDir a délkou d. Výsledná úsečka se uloží do pole U. a v dalším cyklu for je vypočítána hodnota účelové funkce pro každý bod v úsečce - viz obr. 10.9.

```
k = [-1, 0, 1, 2]; % směr
N = length(k);
n = length(x0);
U = zeros(N,n);
% Hlavní smyčka
while i >= iNow && minF > presnost

    % Generování bodů úsečky
    randDir = randn(1, n);

    for j = 1:length(k)
        U(j,:) = x0 + k(j) * randDir * d;
    end

    % Vyhodnocení funkce pro každý bod
F = zeros(N,1);
for j = 1:N
    switch f
        case 1
            F(j) = BP_rosen(U(j,:)); % testovací funkce
        case 2
            F(j) = BP_trigon(U(j,:)); % % testovací funkce
        case 3
            F(j) = BP_dvoubodovy_okrajovy_problem(U(j,:)); % testovací funkce
    end
end
```

Obrázek 10.9 – Definice proměnných a hlavní smyčka

Funkce min() hledá nejmenší hodnotu v poli F a ukládá jí do proměnné minF a proměnná minIndex ukládá pozici na kterém se minimální hodnota nachází. Proměnná x0 ukládá bod ve kterém se nejlepší hodnota funkce nachází - viz obr. 10.10.

```
[minF, minIndex] = min(F);
x0 = U(minIndex,:);
```

Obrázek 10.10 – Hledání nejmenší hodnoty funkce

Následující část kódu se stará o proložení polynomu 3. řádu. Na začátku je definovaná matice A o rozměrech 3x3 a vektoru b**.** Následuje řešení soustavy lineálních rovnic pomocí zpětného lomítka (\), což je operátor pro řešení soustavy lineárních rovnic v MATLABu. Tím se vypočítají neznámé hodnoty vektoru ak. Dále se vypočítá diskriminant D a nastaví se hodnota proměnné tmin na 0 - viz obr. 10.11.

```
A=[-1 1 -1; 1 1 1; 2 4 8];
b=[F(1)-F(2); F(3)-F(2); F(4)-F(2)];
ak=A\b;
D=(2*ak(2))^2-4*3*ak(3)*ak(1);
tmin=0;
```

Obrázek 10.11 – Výpočet diskriminantu

Zkontroluje se, zda je hodnota diskriminantu větší než nula. Pokud ano, máme dvě různé hodnoty t1 a t2, které mohou být řešením kvadratické rovnice. Vypočítáme hodnoty t1 a t2 na základě vzorců pro řešení kvadratické rovnice. Pro každou z hodnot t1 a t2 zkontrolujeme, zda druhá derivace kvadratické funkce je kladná. To zajišťuje, že jsme našli minimum kvadratické křivky. Pokud je druhá derivace kladná pro dané t1 nebo t2, aktualizujeme hodnotu tmin na menší z těchto dvou hodnot - viz obr. 10.12.

```
if D>0
    t1=(-2*ak(2)+sqrt(D))/(2*3*ak(3));
    if(6*ak(3)*t1+2*ak(2)>0)  %2.derivace
        tmin=t1;
    end
    t2=(-2*ak(2)-sqrt(D))/(2*3*ak(3));
    if(6*ak(3)*t2+2*ak(2)>0)  %2.derivace
        tmin=t2;
    end
end
```

Obrázek 10.12 – Řešení kvadratické rovnice a 2. derivace

Na obrazku 11.1 je podmínka pro proměnnou tmin, které nesmí být rovné 0. Poté se vytvoří nový vektor xnew z x0 a příčtením tmin, které je násobeno vektorem s náhodným směrem a délkou d. Následuje vyhodnocení účelové funkce pro xnew a případné uložení xnew do proměnné x0 za předpokladu, že je splněná podmínka, že nová hodnota účelové funkce je lepší než předešlé hodnoty účelové funkce, která je uložena v proměnné minF - viz obr. 10.13.

```
if(tmin~=0)
  xnew = x0 + tmin*randDir * d;

  switch f
        case 1
            fnew = BP_rosen(xnew); % testovací funkce
        case 2
            fnew = BP_trigon(xnew); % testovací funkce
        case 3
            fnew = BP_dvoubodovy_okrajovy_problem(xnew); % testovací funkce
   end

  if(fnew<minF)
     x0=xnew;
     minF=fnew;
  end
end
```

Obrázek 10.13 – Výpočet a porovnání hodnoty účelové funkce polynomu

Níže na obrázku 10.6 je algoritmus pro úpravu délky úsečky, který je důležitý pro podrobnější hledání minima účelové funkce. Podmínka porovnává, jestli nově nalezené minimum účelové funkce je lepší než předešlé, když je tato podmínka splněna, tak se hodnota proměnné locSearchReduction zvětší o 1. Další podmínka je splněna pouze, když proměnná locSearchReduction nabyde 3 krát větší hodnotě než je počet dimenzí, pro které je hledáno minimum funkce účelové funkce. Poté, když je tato podmínka splněna, tak se násobí délka úsešky d číslem 0,9 - viz obr. 10.14.

```
% Úprava parametru délky úsečky

if oldMinF == minF
    locSearchReduction = locSearchReduction + 1;
else
    locSearchReduction = 0;
end

if locSearchReduction == n*3
    d = d * 0.9;
    locSearchReduction = 0;
end
```

Obrázek 10.14 – Úprava délky úsečky

Nejmenší hodnota účelové funkce se ukládá do proměnné oldMinF pro budoucí porovnání v dalším kole cyklu. Proměnná iNow se zvětší o hodnotu 1, aby nedošlo k zacyklení hlavní smyčky a ukončí se hlavní smyčka cyklu while značkou end - viz obr. 10.15.

```
oldMinF = minF;
iNow = iNow + 1;
end
```

Obrázek 10.15 – Ukládání nejmenší hodnoty

Na konec se do konzole vypíše bod, který je nejblíže minimu funkce a hodnota funkce v tomto bodě. Dále se vypíše čas trvání a počet výpočtů funkce pro porovnání s ostatními metodami.

10.3 Kód metody Diferenciální evoluce

V Této části práce je vysvětlen algoritmus diferenciální evoluce. Nejprve je potřeba definovat a nastavit hodnoty proměnným, které jsou nutné pro správný chod scriptu. Poté se definuje globální proměnná `callCount`, která se využívá pro počítání volání testovací funkce – viz obr. 10.16.

```
f = 1; % typ funkce
                %1 Rosenbrockova funkce
                %2 Dvoubodovy_okrajovy_problem
                %3 Trigonometrická funkce
n = 3; % Počet proměnných
N = 10*n; % Velikost populace
i = 1000; % Maximální počet generací
lowerBound = -5; % Dolní mez hledaného intervalu
upperBound = 5; % Horní mez hledaného intervalu
paraF = 0.8; % Parametr F
paraCR = 0.5; % Parametr CR
global callCount
callCount = 0;
```

Obrázek 10.16 – Definice proměnných

První populace se generuje ještě před hlavní smyčkou, protože je to populace, která je kompletně náhodná a poté se vylepšuje v hlavní smyčce – viz obr. 10.17.

```
X = lowerBound + (upperBound - lowerBound) * rand(N, n);
```

Obrázek 10.17 – Definice první populace

Následně přichází hlavní smyčka scriptu, která se vykonává dokud proměnná i s informací o počtu maximálních iterací je větší než 0. Od této proměnné se na konci hlavní smyčky odčítá pokaždé -1. Dále se definuje pole F, které slouží pro ukládání jednotlivých hodnot funkce pro vygenerovanou populaci bodů a nastaví své hodnoty na hodnotu „0“. Následně se opakuje cyklus for dokud pole F nebude mít všechny hodnoty funkce pro dané body. Funkce switch přijímá hodnotu proměnné f a určuje pro jakou funkci je počítána hodnota. Poté už se jen do pole F ukládá daná hodnota funkce na určitou pozici - viz obr. 10.18.

```
while i > 0
    F = zeros(N,1);
    for j = 1:N
        switch f
            case 1
                F(j) = BP_rosen(X(j,:)); % testovací funkce
            case 2
                F(j) = BP_dvoubodovy_okrajovy_problem(X(j,:)); % testovací funkce
            case 3
                F(j) = BP_trigonometrickaFce(X(j,:)); % testovací funkce
        end
    end
```

Obrázek 10.18 – Hlavní cyklus a funkce switch

Zde funkce min() hledá nejmenší hodnotu v poli F a ukládá jí do proměnné minF a proměnná minIndex ukládá hodnotu pozice na kterém se minimální hodnota nachází. Proměnná x0 ukládá bod ve kterém se nejlepší hodnota funkce nachází - viz obr. 10.19.

```
[minF, minIndex] = min(F);
x0 = X(minIndex, :);
```

Obrázek 10.19 – Hledání nejmenší hodnoty funkce

Vytváří se nová populace a přebírá hodnoty staré, které se poté budou vylepšovat - viz obr. 10.20.

```
newPopulation = X;
```

Obrázek 10.20 – Vytváření nové populace s hodnotami ze staré

Následuje cyklus for pro křížení a mutaci bodů ze staré populace. Vybírají se 3 náhodné body ze staré populace - viz obr. 10.21.

```
for j = 1:N
    % Výběr tří náhodných jedinců
    indices = randperm(N, 3);
    a = X(indices(1), :);
    b = X(indices(2), :);
    c = X(indices(3), :);
```

Obrázek 10.21 – Výběr tří náhodných bodů ze staré populace

Vytváří se mutant, podle postupu označovaným RAND, který generuje bod ze tří náhodných bodů ze staré generace populace - viz obr. 10.22.

```
mutant = a + paraF * (b - c);
```

Obrázek 10.22 – Vytváření mutanta

Křížení je proces při kterém vzniká bod do nové generace populace tak, že jakákoliv jeho složka je nahrazena hodnotou mutanta s pravděpodobností paraCR. Nový potomek vytvořený mutací a křížením je uložen do proměnné trial - viz obr. 10.23.

```
crossoverPoints = rand(1, n) < paraCR;
trial = crossoverPoints .* mutant + (1 - crossoverPoints) .* X(j, :);
```

Obrázek 10.23 – Křížení

Poté se vyhodnotí hodnota funkce pro nového potomka. Následuje porovnání hodnoty funkce pro nového potomka s bodem z předešlé generace. Když je nový potomek lepší, tak se finálně zařazuje do nové generace a aktualizuje se pole F s hodnotami funkce pro populaci – viz obr. 10.24.

```
if trialScore < F(j)
    newPopulation(j, :) = trial;
    F(j) = trialScore; % Aktualizace skóre
End
```

Obrázek 10.24 – Porovnání potomka

Poté se aktualizuje i populace. Hlavní smyčka je ukončena zmenšením proměnné i s hodnotou počtu iterací o -1 a cyklus je zakončen značkou end – viz obr. 10.25.

```
    X = newPopulation;
    i = i - 1;
end
```

Obrázek 10.25 – Aktualizace populace

Na konec se do konzole vypíše bod, který je nejblíže minimu funkce a hodnota funkce v tomto bodě. Dále se vypíše čas trvání a počet výpočtů funkce pro porovnání s ostatními metodami.

11 POROVNÁNÍ METOD

V této kapitole jsou porovnány jednotlivé algoritmy mezi sebou a s metodou fminsearch, která je k dispozici v prostředí MATLAB. Nejprve jsou porovnány pomocí grafu a poté podle počtu vyhodnocení účelové funkce pro zvolenou přesnost.

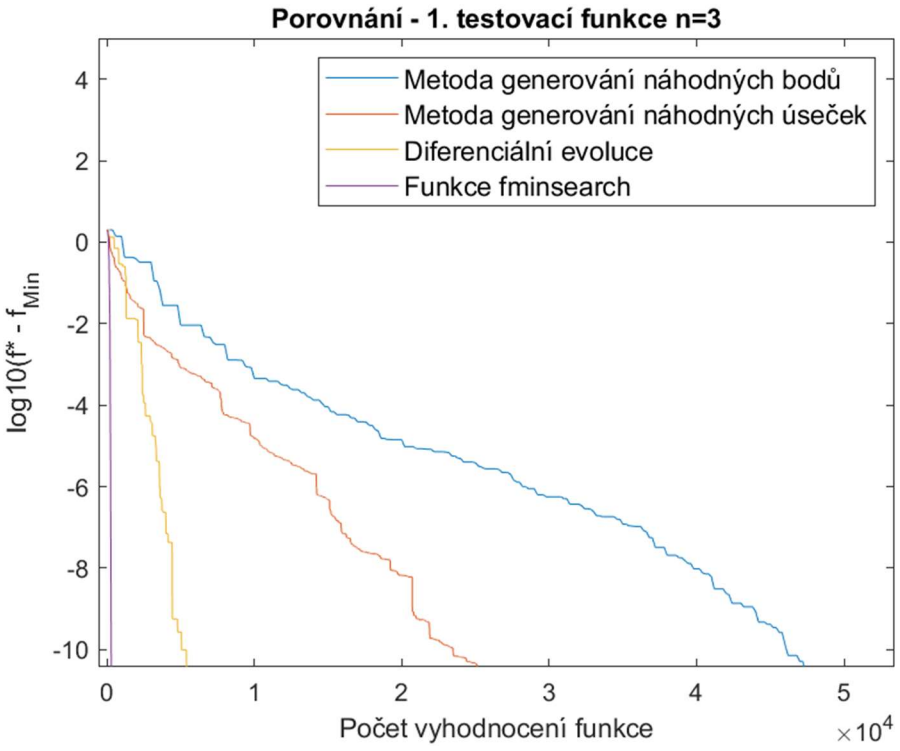
11.1 Vizuální porovnání metod v grafu

Pro porovnání jednotlivých algoritmů se zohledňuje čas trvání a počet volání funkcí. Přesnost nalezení řešení je ohodnocena výrazem

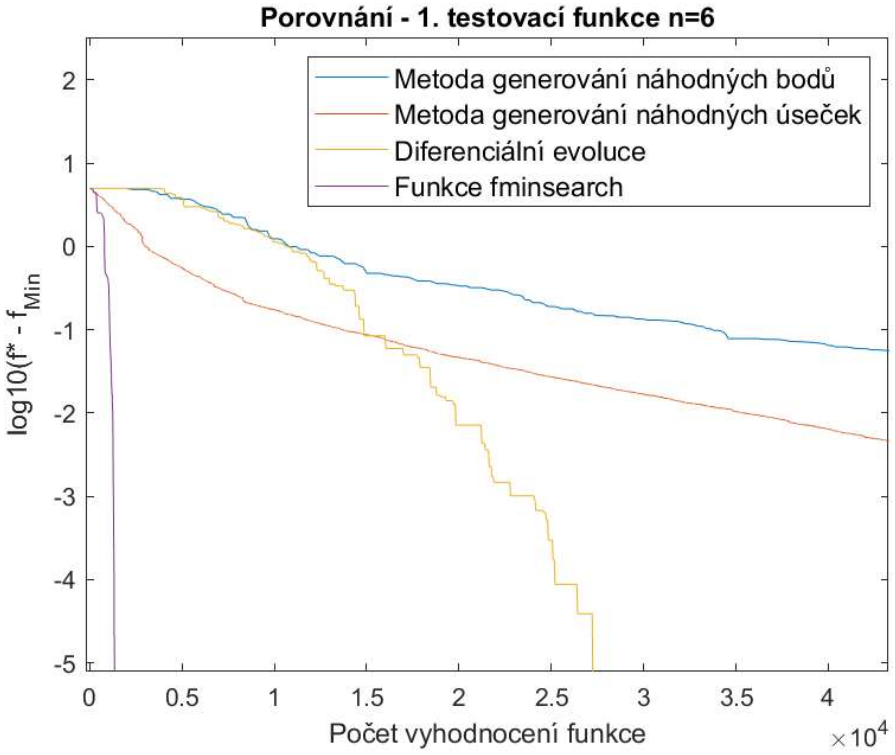
$$\log(f_* - f_{min}) \tag{11.1}$$

kde f_* je nejmenší hodnota účelové funkce pomocí jednotlivých metod optimalizace a f_{min} je skutečné minimum funkce.

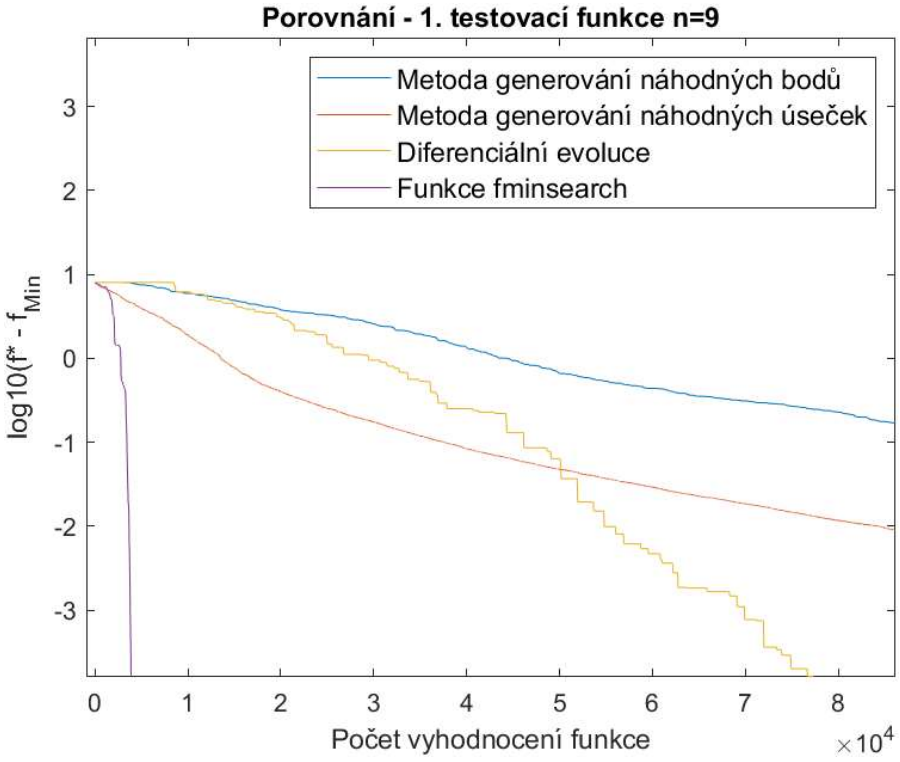
Obrázky 11.1 - 11.9 ukazují průběh přesnosti řešení v závislosti na počtu vyhodnocení jednotlivých testovacích funkcí, pro n=3, n=6 a n=9.



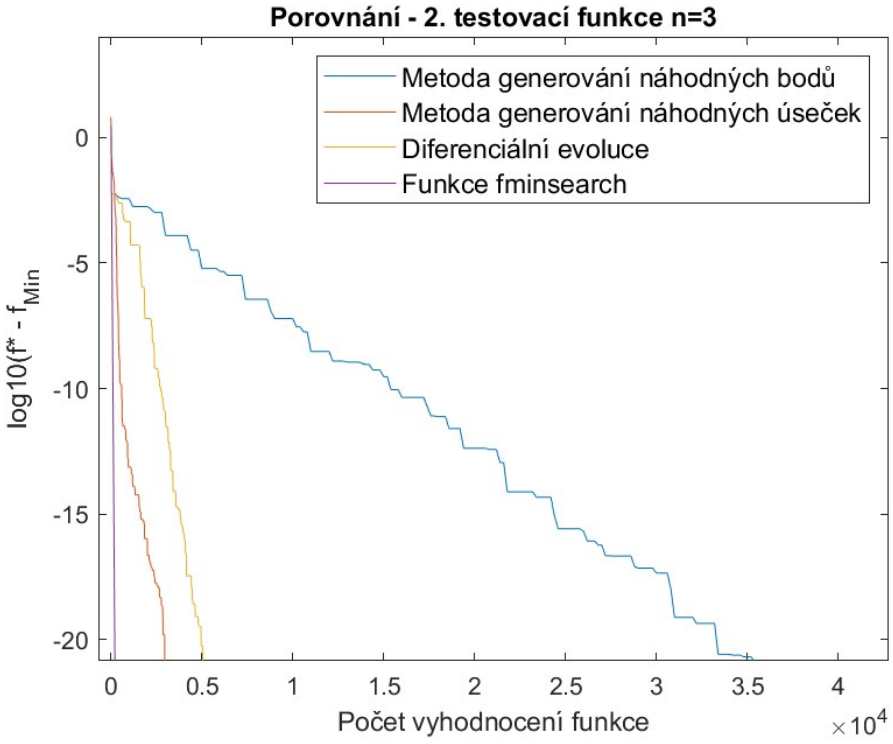
Obrázek 11.1 – Porovnání pomocí Rosenbrockovy funkce pro n=3



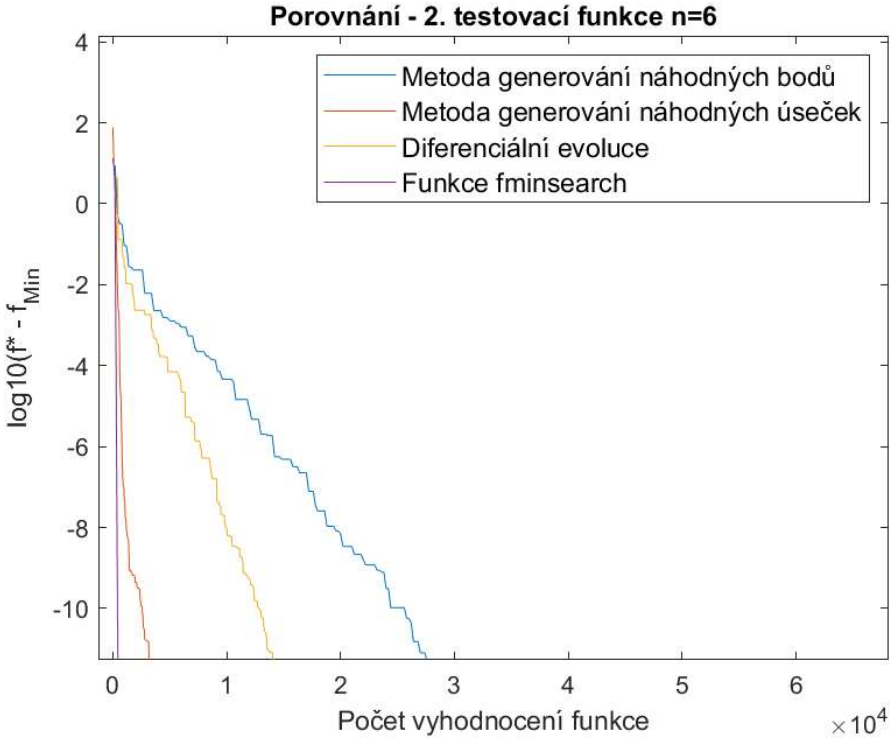
Obrázek 11.2 – Porovnání pomocí Rosenbrockovy funkce pro n=6



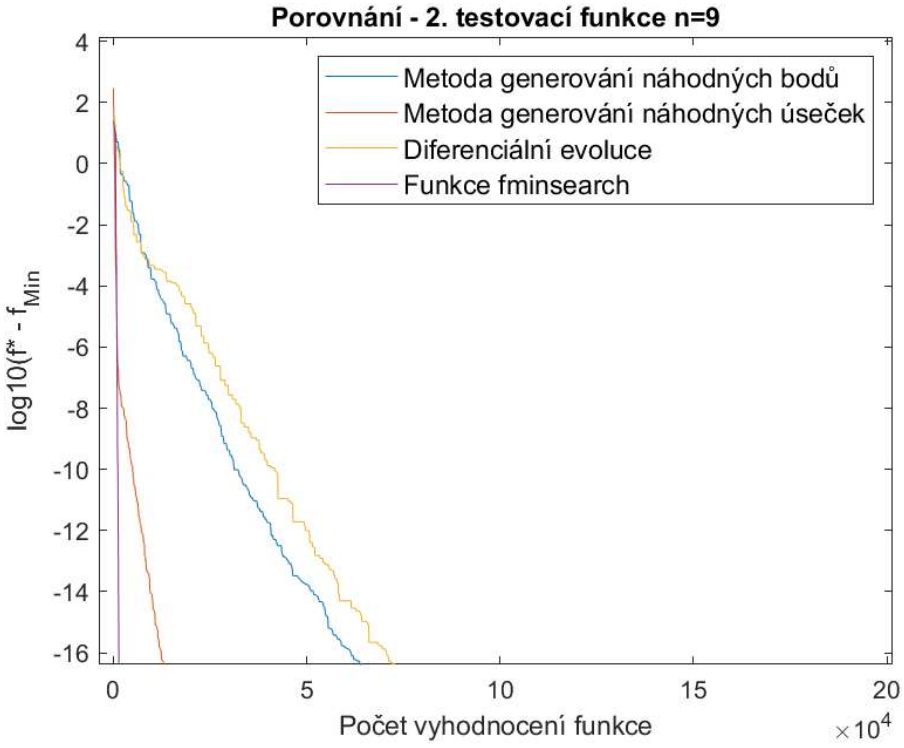
Obrázek 11.3 – Porovnání pomocí Rosenbrockovy funkce pro n=9



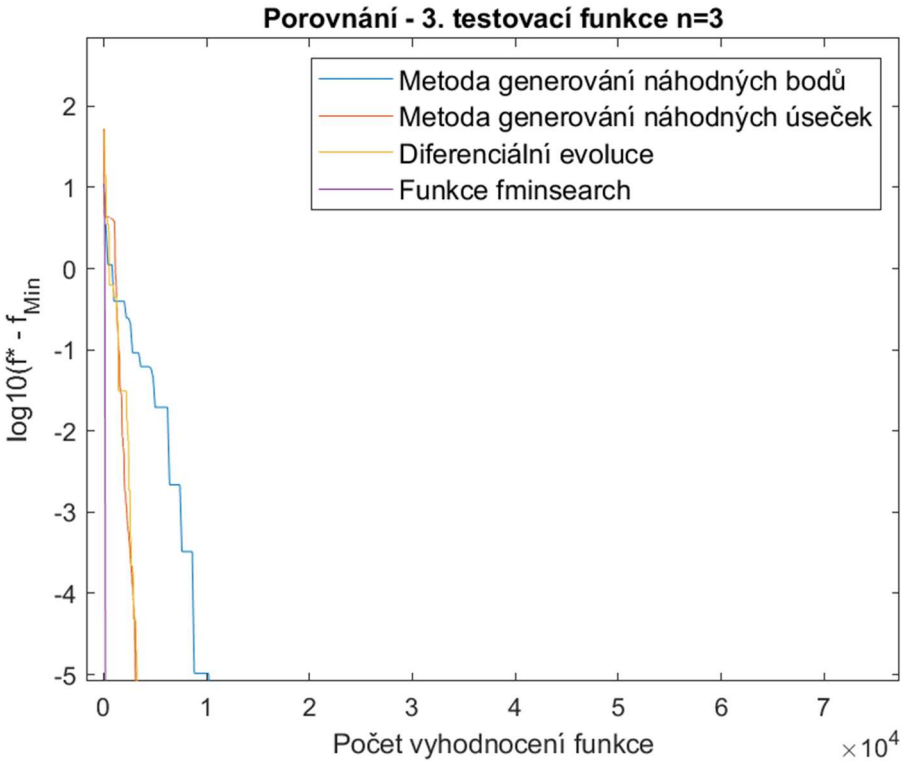
Obrázek 11.4 – Porovnání pomocí Trigonometrické funkce pro n=3



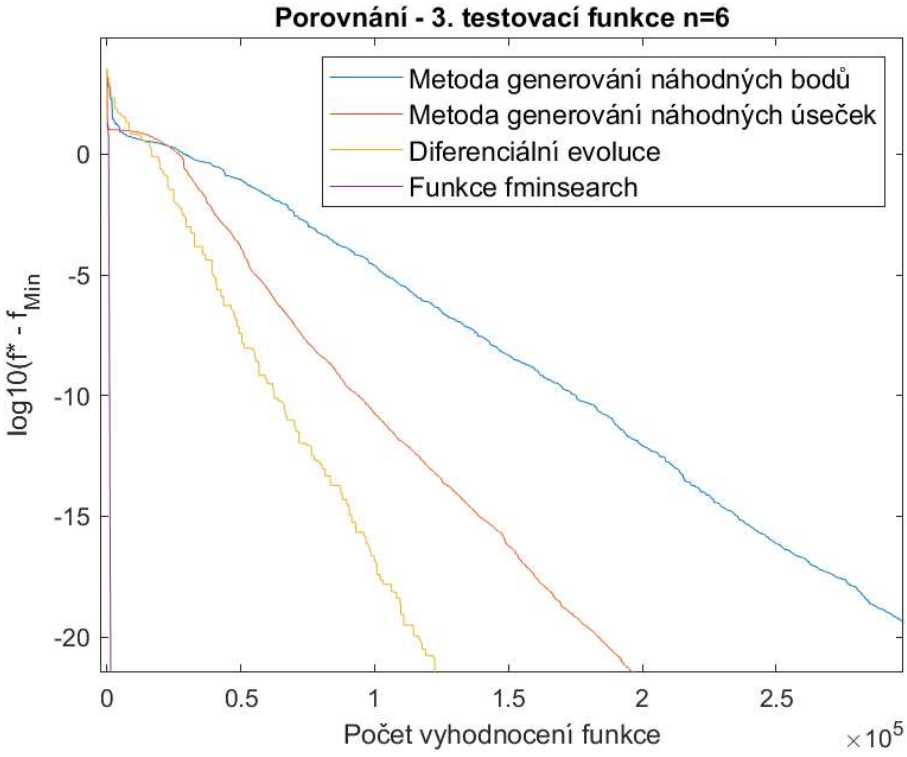
Obrázek 11.5 – Porovnání pomocí Trigonometrické funkce pro n=6



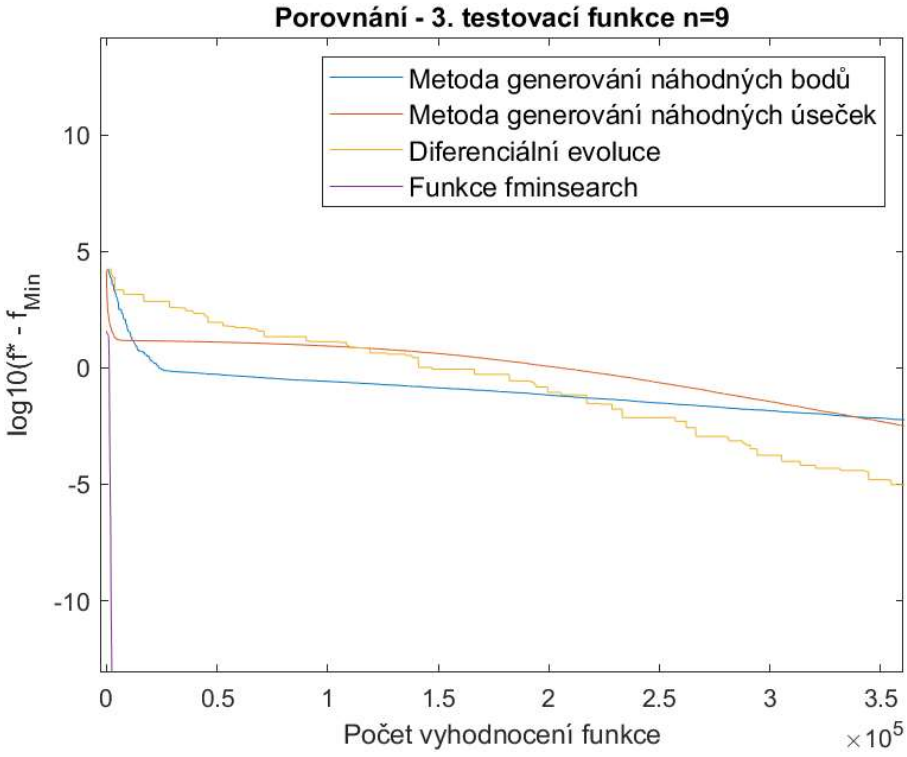
Obrázek 11.6 – Porovnání pomocí Trigonometrické funkce pro n=9



Obrázek 11.7 – Porovnání pomocí dvoubodového okrajového problému pro n=3



Obrázek 11.8 – Porovnání pomocí dvoubodového okrajového problému pro n=6



Obrázek 11.9 – Porovnání pomocí dvoubodového okrajového problému pro n=9

Na Obr. 11.1 - 11.9 je vidět, že podstatně nejrychlejší je metoda flexibilního simplexu v MATLABu, druhá nejlepší vychází metoda generování náhodných úseček a Diferenfiální evoluce. Nejpomalejší je často metoda generování náhodných bodů, ale je možné říci, že často dojde k podobně přesnému výsledku.

11.2 Porovnání podle počtu vyhodnocení účelové funkce

Pro porovnání jednotlivých algoritmů se zohledňuje počet vyhodnocení účelové funkce pro zvolenou přesnost 10^{-4}. Všechny měření jsou prováděna 10 krát a poté byl vypočítán aritmetický průměr podle vzorce 11.2.

$$(x_1 + x_2 + x_3 + \cdots + x_{10}) : 10 \qquad (11.2)$$

Kde x_1, x_2, x_3, … x_{10} jsou počty vyhodnocení účelové funkce.

Tabulky 11.1 - 11.3 ukazují počet vyhodnocení jednotlivých testovacích funkcí, pro n=3, n=6 a n=9 pro zvolenou přesnost 10^{-4}.

	Metoda generování náhodných bodů	Metoda generování náhodných úseček	Diferenciální evoluce	Funkce fminsearch
n=3	9420	6970,3	4374,4	238,7
n=6	207830	92520,4	22428,7	1320,4
n=9	610260	199850,8	72414,2	3918,9

Tab. 11.1 – Porovnání pomocí počtu vyhodnocení Rosenbrockovy funkce

	Metoda generování náhodných bodů	Metoda generování náhodných úseček	Diferenciální evoluce	Funkce fminsearch
n=3	3670	180,2	720,2	55,4
n=6	7840	654,6	4572,3	285,7
n=9	12850	770,2	13284,8	823,5

Tab. 11.2 – Porovnání pomocí počtu vyhodnocení Trigonometrické funkce

	Metoda generování náhodných bodů	Metoda generování náhodných úseček	Diferenciální evoluce	Funkce fminsearch
n=3	9160	1465,1	2513,8	152,8
n=6	171150	66535,2	33857,4	732,5
n=9	1019890	449820,7	325801,6	1557,1

Tab. 11.3 – Porovnání pomocí počtu vyhodnocení dvoubodového okrajového problému

ZÁVĚR

Tato práce se zabývá problematikou optimalizace funkce bez výpočtu derivace. V prostředí MATLAB bylo implementováno několik algoritmů, které tento problém řeší. Na závěr byla získána data pro porovnání těchto algoritmů a bylo provedeno porovnání.

V teoretické části práce jsou vysvětlené základní pojmy týkající se optimalizace, se zaměřením na popis základních přístupů k optimalizaci bez výpočtu derivací. Podrobně jsou popsány metody využité v praktické části. Popsány byly také některé metody, které nejsou využity v praktické části.

Praktická část práce obsahuje kódy naprogramovaných scriptů v jazyce MATLAB s komentářem, vybrání testovacích funkcí a následné porovnání a funkcí fminsearch, která už je v MATLABu k dispozici.

Z výsledků kritérii lze zjistit, že funkce fminsearch, která je k dispozici v MATLABu, je nejlepší. Naprogramované scripty pro optimalizaci funkce bez výpočtu derivací jsou použitelné, ale pro každý problém se hodí jiná metoda. Obecně je možné říci, že pokud je třeba rychle zjistit minimum funkce a nezáleží tolik na přesnosti, je možné zvolit metodu generování náhodných úseček. Pokud jsou ale potřeba přesné hodnoty, je vhodné zvolit Diferenciální evoluci nebo generování náhodných bodů, které ovšem potřebují delší čas na konečný výsledek.

LITERATURA

CVEJN, Jan. *Algorithms of optimization: Elektronický studijní materiál k předmětu Algoritmy optimalizace*. Pardubice: Univerzita Pardubice, FEI, 2021.

KROESE, Dirk P., Thomas TAIMRE a Zdravko I. BOTEV. *Handbook of Monte Carlo methods*. Hoboken, NJ: Wiley, 2011. Wiley series in probability and statistics. ISBN 978-0-470-17793-8.

MORÉ, Jorge J., Burton S. GARBOW a Kenneth E. HILLSTROM. Testing Unconstrained Optimization Software. *ACM Transactions on Mathematical Software* [online]. 1981, **7**(1), 17-41 [cit. 2023-08-25]. ISSN 0098-3500. Dostupné z: doi:10.1145/355934.355936

NEMHAUSER, George a Laurence A. WOLSEY. *Integer and Combinatorial Optimization*. New York: John Wiley, 1988. ISBN 0-471-35943-2.

OŠMERA, Pavel. *Evoluční algoritmy a jejich aplikace: Evolutionary algorithms and their applications*. V Praze: České vysoké učení technické, 2008. ISBN 978-80-01-04192-5.

PRESS, William H. *Numerical recipes: the art of scientific computing*. 3rd ed. Cambridge, United Kingdom: Cambridge University Press, 2007. ISBN 978-0-521-88068-8.

PRICE, Kenneth, Rainer M. STORN a Jouni A. LAMPINEN. *Differential Evolution: A Practical Approach to Global Optimization*. Springer Science & Business Media, 2005. ISBN 3642424163.

PTICHY, Petr. *Náhodné metody v optimalizaci* [online]. Matfyzpress, 2015 [cit. 2023-08-29]. Dostupné z: https://www.karlin.mff.cuni.cz/~ptichy/blogs/nmo/NMO_01.pdf

RUBINSTEIN, Reuven Y. a Dirk P. KROESE. *Simulation and the monte carlo method*. 2nd ed. Hoboken: John Wiley, 2008. ISBN 978-0-470-17794-5.

TVRDÍK, Josef. *Evoluční algoritmy* [online]. Ostravská univerzita, Přírodovědecká fakulta., 2010 [cit. 2023-08-29]. Dostupné z: https://docplayer.cz/656774-Ucebni-texty-ostravske-univerzity-prirodovedecka-fakulta-evolucni-algoritmy-josef-tvrdik.html

PŘÍLOHY

A-CD

Příloha A

Příloha k bakalářské práci

METODY OPTIMALIZACE BEZ VÝPOČTU DERIVACÍ

Tomáš Hladík

CD

OBSAH