VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
DEPARTMENT OF AUTOMATION AND COMPUTER SCIENCE

MATLAB/GLOBAL OPTIMIZATION TOOLBOX: ŘEŠENÍ OPTIMALIZAČNÍCH PROBLÉMŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KATEŘINA ŠVIHÁLKOVÁ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
DEPARTMENT OF AUTOMATION AND COMPUTER SCIENCE

MATLAB/GLOBAL OPTIMIZATION TOOLBOX: ŘEŠENÍ OPTIMALIZAČNÍCH PROBLÉMŮ

MATLAB/GLOBAL OPTIMIZATION TOOLBOX: SOLVING OPTIMIZATION PROBLEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — KATEŘINA ŠVIHÁLKOVÁ
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. RADOMIL MATOUŠEK, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Bakalářská práce popisuje základní možnosti globálních optimalizačních metod implementovaných ve výpočetním prostředí MATLAB. V první části jsou popsány aplikační knihovny MATLABU – Optimization Toolbox a Global Optimization Toolbox. Práce dále popisuje vybrané algoritmy z obou knihoven, konkrétně genetické algoritmy, simulované žíhání, pattern search a fminsearch a také popisuje způsob jejich implementace. Poslední část je zaměřena na samotné řešení optimalizačních úloh. Rosenbrockova funkce, Rastriginova funkce, problém obchodního cestujícího a konzola s proměnnou průřezovou charakteristikou jsou řešeny uvedenými metodami. Dosažené výsledky u jednotlivých úloh jsou vzájemně porovnány a vyhodnoceny.

Abstract

The bachelor thesis deals with basic description of global optimization methods which are implemented in MATLAB environment. First part of this thesis describes two MATLAB's toolboxes – Optimization Toolbox and Global Optimization Toolbox. This study also describes chosen algorithms from those toolboxes (Genetic Algorithms, Simulated Annealing, Pattern Search and Fminsearch) and describes it's implementation. The last part is focused on the solving of optimization problems. Rosenbrock's function, Rastrigin's function, Traveling Salesman Problem and Stepped Cantilever Beam Design Problem are being solved by methods mentioned above. Reached results of every task are compared to each other and evaluated.

Klíčová slova

MATLAB, Optimization Toolbox, Global Optimization Toolbox, genetické algoritmy, simulované žíhání, pattern search, Nelder-Mead algoritmus, optimalizační problémy

Keywords

MATLAB, Optimization Toolbox, Global Optimization Toolbox, Genetic Algorithm, Simulated Annealing, Pattern Search, Nelder-Mead algorithm, optimization problems

Citace

Kateřina Švihálková: MATLAB/Global Optimization Toolbox: řešení optimalizačních problémů, bakalářská práce, Brno, FSI VUT v Brně, 2014
Vedoucí práce: doc. Ing. Radomil Matoušek, Ph.D.

MATLAB/Global Optimization Toolbox: řešení optimalizačních problémů

Prohlášení

Prohlašuji, že předložená bakalářská práce je původní a zpracovala jsem ji samostatně. Prohlašuji, že citace pramenů je úplná, že jsem ve své práci neporušila autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech související s právem autorským).

.....
Kateřina Švihálková
5. června 2014

Poděkování

Děkuji zejména vedoucímu mé bakalářské práce doc. Ing. Radomilu Matouškovi, Ph.D. za odborné vedení, poskytnutý čas, cenné rady a ochotu. Dále děkuji Ing. Františku Horákovi a Ing. Petru Šoustkovi za korekturu a pomoc při psaní této práce. V neposlední řadě děkuji rodině, která mi byla oporou během celého studia.

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2013/2014

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Kateřina Švihálková

který/která studuje v **bakalářském studijním programu**

obor: **Aplikovaná informatika a řízení (3902R001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

MATLAB/Global Optimization Toolbox: řešení optimalizačních problémů

v anglickém jazyce:

MATLAB/ Global Optimization Toolbox: Solving Optimization Problems

Stručná charakteristika problematiky úkolu:

Bakalářská práce stručným, ovšem edukativním, způsobem pojedná o možnostech využití optimalizačních nástrojů systému MATLAB/Global Optimization Toolbox pro řešení vybraných optimalizačních problémů. Dosažená řešení budou rovněž porovnána s řešeními dosaženými s využitím vybraných klasických optimalizačních algoritmů implementovaných v knihovně MATLAB/Optimization Toolbox.

Cíle bakalářské práce:

- Stručný popis knihovny MATLAB/Global Optimization Toolbox.
- Stručný popis knihovny MATLAB/Optimization Toolbox.
- Základní popis užitých optimalizačních algoritmů a příklad jejich implementace.
- Řešení daných optimalizačních úloh s využitím MATLAB/Global Optimization Toolbox a vyhodnocení efektivity užitých řešičů (k porovnání bude vyžit i zvolený řešič z knihovny MATLAB/Optimization Toolbox).

Seznam odborné literatury:

[01] Mathworks [online] http://www.mathworks.com/products/global-optimization/
[02] Mathworks [online] http://www.mathworks.com/products/optimization/
[03] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, 1989.

Vedoucí bakalářské práce: doc. Ing. Radomil Matoušek, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2013/2014.

V Brně, dne 4.4.2014

L.S.

Ing. Jan Roupec, Ph.D. Ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. Děkan fakulty

Obsah

Úvod

Existence matematicky pojatých optimalizačních metod se datuje již od dob Newtona, Lagrange, Cauchyho či Eulera. Newton s Leibnitzem položili základy diferenciálního počtu, což významně přispělo k rozvoji optimalizačních metod. Matematickou analýzu, která se zabývala minimalizací funkce, rozvinuli Bernoulli, Euler, Lagrange a Weistrass.

Významný rozvoj matematické optimalizace nastal v podstatě pod "rouškou"tzv. operačního výzkumu v průběhu II. světové války, kdy bylo třeba řešit rozsáhle optimalizační úlohy v oblasti logistiky, plánování výroby apod. Pojem matematické programování vznikl ve 40. letech 20. století s rozvojem výpočetní techniky a zásluhou amerického vědce Georgie B. Dantzinga [5]. Matematické programování představuje studium matematické struktury optimalizačních problémů, jejich řešení pomocí matematických metod a realizaci řešení pomocí počítačů. S rozvíjející se technikou se výrazně rozšířil rozsah a složitost optimalizačních problémů. Vývoj optimalizačních technik probíhal současně nejen v informatice, ale také v operačním výzkumu, numerické analýze, teorii her, matematické ekonomii, teorii řízení a kombinatorice. S touhou vědců a inženýrů řešit stále složitější problémy vyvstaly otázky řešitelnosti kombinatorických problémů spadajících do třídy tzv. NP problémů. Řešení takových problému, respektive nalezení jejich optimálního řešení, znamená v podstatě enumerativní vyčíslení všech možností, což není pro reálné úlohy mnohdy možné. V tomto ohledu začaly do vývoje optimalizačních metod významně zasahovat přístupy z oblasti tzv. umělé inteligence. Do této skupiny jsou zařazovány tzv. optimalizační metaheuristiky, které čerpají často inspiraci v chování přírodních systémů. Velká skupina těchto metod a v podstatě přístupů k řešení problémů je označovaná jako Soft Computing. S rozvojem výpočetní techniky byly převážně v 80. a 90. letech 20 ho století vytvořeny algoritmy typu genetický algoritmus, simulované žíhání, diferenciální evoluce aj. Tyto algoritmy sice z principu negarantují nalezení optimálních výsledků, ale na druhou stranu dokazují, že jsou v akceptovatelném čase schopny najít dostatečně dobré výsledky, či volněji řečeno nalézt inženýrsky optimální řešení daných úloh. Prvotně čistě matematické softwarové nástroje typu MATLAB nebo Mathematica dnes běžně implementují uvedené typy metaheuristik právě z důvodu možnosti řešit úlohy, které ze své povahy využití exaktních nástrojů matematické optimalizace komplikují, nebo neumožňují. Principálně může jít o rozsáhlé NP úlohy, funkce které nejsou spojité a hladké, obsahují mnoho extrémů ve velkém počtu dimenzí apod. Předložená bakalářská práce se stručně zabývá představením optimalizačních nástrojů ze skupiny metod soft computing, které jsou dostupné v systému MATLAB v knihovně Global Optimization Toolbox (GOTbx). Práce krom podrobnějšího popisu knihovny GOTbx obsahuje i základní charakteristiku k MATLABU dostupné knihovny Optimization Toolbox (OTbx), která je dobře využitelná pro řešení problémů lineárního programování a některých nelineárních úloh. Bakalářská práce je členěna do 4 základních kapitol. Z principu rozsahu práce se jedná o základní úvod do problematiky metod dostupných v GOTbx, doplněný sadou příkladů vhodných k proniknutí do případné vlastní implementace. Sou-

částí je rovněž základní studie chování genetického algoritmu (GA, Genetic Algorithms) a metody simulovaného žíhání (SA, Simulated Anealing) při řešení testovací úlohy multimodálního charakteru ozn. jako F6 (Rastriginova funkce). Efektivně parametrizované metody GA a SA jsou srovnány s výsledky pokročilých algoritmů Petern Search a exaktního N-M algoritmu. Na základě provedené jednoduché empirické studie jsou učiněny závěry. K další prezentované úloze patří modelový problém obchodního cestujícího definovaný na kružnici bodů. Závěr práce je doplněn příkladem řešení úlohy technického charakteru, která typově spadá do oblasti smíšeného celočíselného programování.

Kapitola 1

Optimization Toolbox

MATLAB je interaktivní prostředí a skriptovací jazyk na vysoké úrovni, který byl vytvořen pro vědeckotechnické výpočty, simulaci, vizualizaci, návrhy systémů a další. MATLAB a Simulink mohou využívat aplikační knihovny (toolboxy), které jsou rozčleněny podle technických odvětví. Pro řešení optimalizačních úloh disponuje MATLAB mnoha nástroji. Exaktní optimalizační metody (metody matematického programování) jsou integrovány v Optimalizačním Toolboxu, který bude dále popsán.

1.1 Funkce MATLAB Optimization Toolbox

Optimalizační metody jsou implementovány formou parametrizovatelných metod, které lze v kontextu optimalizace označovat rovněž jako takzvané řešiče. Tyto metody si nyní stručně popíšeme a rozdělíme dle zvyklosti matematického programování. Obecně je možné říci, že identifikátory metod vystihují jejich účel.

Kvadratické a lineární programování (Linear and Quadratic Programming)

- linprog – lineární programování
- quadprog – kvadratické programování

Nelineární optimalizace (Nonlinear minimization)

- fminbnd – nelineární minimalizace s danými mezemi
- fminsearch – vícedimenzionální nepodmíněná nelineární optimalizace, využívá Nelder-Mead algoritmus
- fminunc – vícedimenzionální nepodmíněná nelineární optimalizace
- fseminf – vícedimenzionální podmíněná nelineární optimalizace se semi-infinitními omezujícími podmínkami
- fmincon – vícedimenzionální podmíněná nelineární optimalizace

Vícekriteriální optimalizace (Multiobjective Optimization)

- fgoalattain – Vícekriteriální optimalizace s penalizacemi.
- fminimax – Vícekriteriální optimalizace – úlohy minimaxu.

Optimalizace využívající metody nejmenších čtverců (Least-Squares Optimization)

- lsqlin – lineární metoda nejmenších čtverců s lineárními omezujícími podmínkami
- lsqnonneg – lineární metoda nejmenších čtverců s nezápornými omezujícími podmínkami
- lsqcurvefit – nelineární metoda aproximace křivek s mezemi
- lsqnonlin – nelineární metoda nejmenších čtverců s mezemi

Řešení soustav nelineárních rovnic (nonlinear system of equation solving)

- fsolve – řešení soustav nelineárních rovnic
- fzero – hledání kořenu nelineární skalární funkce

Výše uvedené metody jsou uloženy jako MATLAB m-soubory a jejich kód si je možné prohlédnout příkazem `type function_name`. Optimization Toolbox dále umožňuje napsání uživatelských m-souborů. MATLAB umožňuje přistoupit k metodám pomocí uživatelem vytvořeného m-skriptu nebo za pomocí GUI aplikace (Optimization app) implementované v toolboxu. Aplikace efektivně zjednodušuje definici problému a parametrizaci zvolených řešičů. Rovněž umožňuje otestování metod pomocí předdefinovaných optimalizačních problémů.

Charakteristické vlastnosti aplikace:

- Vybrat řešič a definovat optimalizační úlohu.
- Nastavit optimalizační volby a výchozí hodnoty pro vybraný řešič.
- Spustit úlohu a následně vykreslit průběžné a závěrečné výsledky.
- Zobrazit řešič – jeho specifickou dokumentaci.
- Importovat a exportovat zadefinovaný problém, nastavení algoritmu a výsledku mezi MATLAB workspace a Optimization app.
- Vygenerování příslušného m-kódu s možností dalšího využití.
- Propojit s metodami Global Optimization Toolbox.

V rámci implementace Optimization Toolbox lze efektivně využít Parallel Computing Toolbox. Od verze M2012 lze využít až 12 CPU jader, pokud by byl požadavek na paralelizaci většího rozsahu, je nutné využít distribuované výpočty.

1.1.1 Lineární programování (Linear Programming)

Lineární programování [6] se zabývá maximalizací nebo minimalizací lineární kriteriální funkce, jejíž omezující podmínky mají tvar lineárních rovnic a nerovností. Lineárního programování je využíváno ve finančnictví, operačním výzkumu a dalších oborech.

Formulace úlohy lineárního programování:

$$\min\{\mathbf{c}^T\mathbf{x}\} \tag{1.1}$$

za podmínek

$$A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \in M, \tag{1.2}$$

kde $\mathbf{x}$ je vektor proměnných, A matice koeficientů omezujících podmínek, $\mathbf{b}$ i $\mathbf{c}$ jsou sloupcové vektory koeficientů v kriteriální funkci a odpovídajících proměnných a M je množina přípustných řešení.

K dispozici jsou tři algoritmy implementované v metodě `linprog`:

- Simplexový algoritmus (simplex) – Systém lineárních omezení definuje mnohostěn jako množinu přípustných řešení. Simplexový algoritmus se pohybuje po okrajích mnohostěnu od vrcholu k vrcholu, dokud nedosáhne optimálního řešení. Je to jeden z nejvyužívanějších nástrojů lineárního programování [5].
- Interrior point algoritmus – Pracuje na principu primárně-duální a prediktor-korektor metody [5].
- Active set algoritmus – Varianta sekvenčního kvadratického algoritmu modifikovaného pro lineární programování [14].

1.1.2 Kvadratické programování (Quadratic Programming)

Kvadratické programování [14] umožňuje řešit problémy minimalizace vícerozměrné kvadratické funkce s případnými lineárními omezeními. Časté optimalizační úlohy vedoucí na problém kvadratického programování jsou z oblasti financí, optimalizace výroby energie pro elektrické nástroje a dalších oborů.

Úloha kvadratického programování:

$$\min\left\{\frac{1}{2}\mathbf{x}^T H\mathbf{x} + \mathbf{c}^\mathbf{T}\mathbf{x}\right\} \tag{1.3}$$

za podmínek

$$A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \in M, \tag{1.4}$$

kde $\mathbf{x}$ je vektor proměnných, A matice koeficientů omezujících podmínek, H matice koeficientů kvadratických členů, $\mathbf{b}$ i $\mathbf{c}$ jsou sloupcové vektory koeficientů v kriteriální funkci a odpovídajících proměnných a M je množina přípustných řešení.

K dispozici jsou tři algoritmy implementované v metodě `quadprog`:

- Interior-point-convex algoritmus – Řeší konvexní problémy s různými kombinacemi omezení [5].
- Trust-region-reflective algoritmus – Řeší problémy s danými mezemi nebo lineárními omezeními[18].
- Active-set algoritmus – Řeší problémy s různými kombinacemi omezení[14].

1.1.3 Nelineární optimalizace(Nonlinear Optimization)

Optimization Toolbox poskytuje široce využívané algoritmy pro řešení úloh nelineárního programování. Toolbox obsahuje řešiče pro nepodmíněnou a podmíněnou nelineární optimalizaci a řešiče pro metodu nejmenších čtverců.

Podmíněná nelineární optimalizace (Constrained nonlinear optimization)

Podmíněné nelineární optimalizační problémy [3] jsou složeny z lineární nebo nelineární kriteriální funkce s případnými omezujícími podmínkami.

Úloha podmíněné nelineární optimalizace:

$$\min_{\mathbf{x}\in\mathbf{M}} f(\mathbf{x}) \tag{1.5}$$

$$g_i\mathbf{x} = c_i, i = 1, \ldots, n \tag{1.6}$$

$$h_j\mathbf{x} \geq d_j, j = 1, \ldots, m \tag{1.7}$$

kde $\mathbf{x}$ je vektor proměnných, g a h obecné funkce, c a d koeficenty pravých stran a M je množina přípustných řešení.

Metoda `fmincon` obsahuje čtyři algoritmy:

- Trust-Region Reflective algoritmus – Je využíván pro podmíněné úlohy s mezemi nebo omezeními ve tvaru nerovností [18].
- Active-set algoritmus – Je využíván pro obecné problémy nelineární optimalizace [14].
- SQP algoritmus – Je iterativní metoda, která minimalizuje sekvenci optimalizačních subproblémů [14].
- Interior point algoritmus – Řeší podmíněné nelineární problémy jako sekvenci aproximovaných minimalizačních problémů. Využívá bariérové funkce pro udržení běhu optimalizace na definované množině přípustných řešení [23].

V metodě `fminbnd` je implementovaný algoritmus, který hledáná lokální minimum v jednodimenzionálním problému s danými mezemi. K vyhledávání používá metodu zlatého řezu a parabolickou interpolaci.

Metoda `fseminf` implementuje algoritmus, který řeší problém minimalizace semi-infinitní podmíněné nelineární funkce.

Nepodmíněná nelineární optimalizace (Unconstrained Nonlinear Optimization)

Nepodmíněná nelineární optimalizace [3] řeší problémy minimalizace kriteriální funkce, přičemž nejsou kladena žádná omezení na definiční obor funkce. Úloha nepodmíněné nelineární optimalizace:

$$\min_{\mathbf{x}\in\mathbf{M}} f(\mathbf{x}) \tag{1.8}$$

kde $\mathbf{x}$ je vektor proměnných a M je množina přípustných řešení.

V metodě `fminunc` jsou dostupné dva algoritmy:

- Quasi-Newton algoritmus – Kombinuje kvadratický a kubický line search postup a Broyden-Fletcher-Goldfarb-Shannovu formulaci pro aktualizaci aproximace Hessovy matice [14].
- Trust-region algoritmus – Je využíván pro nepodmíněné nelineární problémy[18].

V metodě `fminsearch` je implementován Nelder-mead algoritmus [13], který spadá do kategorie metod přímého vyhledávání (direct search)[16]. Pracuje pouze s hodnotami kriteriální funkce, nevyžaduje její derivace. Je vhodný i pro funkce, které nejsou hladké.

1.1.4 Lineární a nelineární metody nejmenších čtverců a Data Fitting (Linear and Nonlinear Least-Squares Optimization and Data Fitting)

Základní úloha metody nejmenších čtverců [4] spočívá v nalezení vektoru x, který minimalizuje sumu druhých mocnin kriteriální funkce $f(\mathbf{x})$ s omezujícími podmínkami:

$$\min_{\mathbf{x}\in M} \sum_i f_i^2(\mathbf{x}) \tag{1.9}$$

kde $\mathbf{x}$ je vektor proměnných, f a h obecné funkce, c a d koeficenty pravých stran a M je množina přípustných řešení.

Podmíněné lineární least-squares problémy

V toolboxu jsou obsaženy metody `lsqnonneg` a `lsqlin`, které využívají algoritmy:

- Active set algoritmus – Řeší ohraničené problémy s omezeními ve tvaru rovností nebo nerovností [14].
- Trust-region-reflective algoritmus - Je vhodný pro úlohy, které mají vázaná omezení[18].

Nelineární metody nejmenších čtverců

V metodě `lqnonlin` a `lsqcurvefit` jsou implementovány algoritmy využívající modifikovanou Gauss-Newton metodu, označovanou jako Levenberg-Marquardt algoritmus [11].

Datta fitting

Toolbox poskytuje specializované rozhraní pro data fitting problémy, ve kterých chceme najít členy nelineárních funkcí, které nejvíce zapadají do množiny datových bodů. Pro data fitting problémy toolbox používá stejné algoritmy jako v případě nelineárních least-squares problémů.

1.1.5 Rešení soustav nelineárních rovnic (Nonlinear system of equation solving)

Úkolem je nalezení kořenů soustavy nelineárních rovnic. V metodě `fsolve` je implementován dogleg trust-regions algoritmus pro řešení n nelineárních rovnic o n neznámých. Funkce dále umožňuje řešit problémy za pomoci trust-region reflective[18] a Levenberg-Marquardt[11] algoritmů.

Metoda `fzero` poskytuje možnost nalezení kořenu kriteriální funkce. K jeho nalezení využívá kombinace intervalové bisekce, lineární interpolace a inverzní kvadratické interpolace.

1.1.6 Vícekriteriální optimalizace (Multiobjective Optimization)

Vícekriteriální optimalizace se zabývá minimalizací více kriteriálních funkcí s omezeními. Optimization Toolbox poskytuje funkce pro dva různé typy problému vícekriteriální optimalizace:

- `fgoalattain` – Goal attainment problém snižuje hodnotu lineární nebo nelineární vektorové funkce k dosažení cílové hodnoty dané cílovým vektorem. Relativní význam cílů je indikován za použití hmotnostního vektoru. Úloha může mít lineární nebo nelineární omezení.
- `minimax` – Minimax řeší problém minimalizace nejhorší možnosti ze zadaných víceproměnných funkcí s případnými lineárními nebo nelineárními omezujícími podmínkami.

Výše popsané problémy vícekriteriální optimalizace mohou být převedeny na standardní podmíněný optimalizační problém, který je řešen pomocí active-set přístupu.

Kapitola 2

Global Optimization Toolbox

Pro řešení optimalizačních problémů, které mají více extrémů, je v MATLABU k dispozici další aplikační knihovna, tzv. Global Optimization Toolbox. Knihovna obsahuje úlohy založené na strategiích z oblasti Soft computingu.

Konkrétně jsou implementovány algoritmy:

- *global search*
- *multistart*[22]
- *pattern search*[21]
- *genetic algorithm*[7]
- *simulated annealing*[8]

Tyto metody jsou využitelné v problémech, které nelze řešit exaktními metodami matematické optimalizace. Například v úlohách kombinatorické optimalizace, kde nalezení optimálního řešení je velmi časově náročné.

Dále je vhodné uvést, že Global Optimization Toolbox implementuje jak uvedené metody Soft computingu, tak umožňuje využít exaktní metody matematické optimalizace implementované v Optimization Toolboxu.

Global Optimization Toolbox poskytuje možnost přistupovat k metodám přímo z příkazového řádku nebo využitím Optimization app v Optimization Toolboxu. Základní podmínkou využití implementovaných optimalizačních metod je schopnost definice problému ve formě kriteriální funkce. Kriteriální funkce je definována pomocí standardní m-funkce, přičemž jsou možné i varianty založené na volání jiných knihoven, simulinkového kódu, či tzv. mex-files (kompilované programy z m-kódu). Pro vlastní proces optimalizace je možná relativně variabilní parametrizace řešiče. Rovněž je umožněno volání specifických funkcí pro grafickou reprezentaci optimalizačního běhu. Rychlost výpočtu lze ve většině případů zrychlit využitím paralelního výpočetního modelu. Tato funkcionalita je umožněna pomocí Parallel Computing Toolboxu.

2.1 Global Search a Multistart Solvers

Global search a multistart řešiče používají gradientně založené metody, implementované v metodě `fmincon`, pro hledání extrému. Řešiče začínají vyhledávání z několika výchozích bodů. Výsledky uchovávají během celého vyhledávácího procesu.

Global search – využívá scatter-search algoritmus pro generování výchozích bodů, které závisí na hodnotě kriteriální funkce a uchovaných výsledků.
Multistart – výchozí body jsou rovnoměrně rozložené v předem stanovených mezích nebo jsou uživatelsky definované. Lokální řešiče běží ze všech startovních bodů, běh může být sériový nebo paralelní. Multistart solver poskytuje na výběr různé lokální řešiče nelineárních problémů.

2.2 Genetický algoritmus (Genetic algorithm)

Genetický algoritmus (GA)[7] napodobuje biologické principy evoluce, konkrétně Darwinovu teorii přirozeného výběru a Mendelovu genetiku. Upravuje populaci v následujících generacích podle pravidel biologické reprodukce. Díky využití svých principů genetický algoritmus podstatně zvyšuje šance na nalezení globálního řešení. Metody genetických algoritmů jsou využitelné k řešení nepodmíněných, vázaně omezených a obecných optimalizačních problémů.

Následující tabulka 2.1 znázorňuje možnosti genetických algoritmů, které implementuje Global Optimization Toolbox:

Skupina metod GA	**Implementované metody GA** [1]
Vytvoření	Uniform, feasible
Fitness škálování	Rank-based, proportional, top (truncation), shift linear
Výběr	Roulette, stochastic uniform selection (SUS), tournament, uniform, remainder
Křížení	Arithmetic, heuristic, intermediate, scattered, single-point, two-point
Mutace	Adaptive feasible, Gaussian, uniform
Vykreslení	Best fitness, best individual, distance among individuals, diversity of population, expectation of individuals, max constraint, range, selection index, stopping conditions

Tabulka 2.1: GA ([1]Zavedené pojmy jsou ponechány v anglickém jazyce.)

Dále je možné specifikovat velikost populace (tj. počet potenciálních řešení), elitismu, poměr křížení a migraci.

Operátory výběru, mutace a křížení jsou implementovány jako m-funkce, které lze modifikovat nebo vytvářet vlastní v různých datových formátech. Podobně platí i pro další metody genetických algoritmů jako je například vlastní volba ukončovacích kriterií, které závisí na čase, ustálení, fitness limitu a počtu generací. Pro zvýšení rychlosti optimalizačního procesu je možná vektorizace problému.

2.3 Vícekriteriální genetický algoritmus (Multiobjective Genetic Algorithm Solver)

Vícekriteriální optimalizace [7] se zabývá minimalizací více kriteriálních funkcí s danými omezeními. Vícekriteriální genetický algoritmus reší vícekriteriální problémy pomocí identifikace *pareto front* - množina rovnoměrně rozložených nedominantních optimálních řešení. Metoda je použitelná při řešení hladkých nebo nehladkých optimalizačních problémů s případnými mezemi a lineárními omezeními. Algoritmus nevyžaduje, aby funkce byla diferencovatelná nebo spojitá. Následující tabulka 2.2 znázorňuje možnosti standardních vícekriteriálních genetických algoritmů poskytovaných Global Optimization Toolbox:

Skupina metod GA	**Implementované metody vícekriteriálního GA** [2]
Vytvoření	Uniform, feasible
Fitness škálování	Rank-based, proportional, top (truncation), linear scaling, shift
Výběr	Tournament
Křížení	Arithmetic, heuristic, intermediate, scattered, single-point, two-point
Mutace	Adaptive feasible, Gaussian, uniform
Vykreslení	Average Pareto distance, average Pareto spread, distance among individuals, diversity of population, expectation of individuals, Pareto front, rank histogram, selection index, stopping conditions

Tabulka 2.2: Možnosti vícekriterálního genetického algoritmu ([2]Zavedené pojmy jsou ponechány v anglickém jazyce.)

Dále je možné specifikovat velikost populace, elitismu, poměr křížení a migraci. Možnosti uživatelského nastavení jsou stejné jako v případě Genetických algoritmů.

2.4 Pattern Search Solver

Global Optimization Toolbox obsahuje tři algoritmy přímého vyhledávání:

- Generalized pattern search(GPS)[2]
- Generating set search (GSS)[10]
- Mesh adaptive search (MADS)[1]

Zatímco tradiční optimalizační algoritmy používají přesné nebo aproximované informace o gradientu nebo derivacích vyššího řádu k hledání optima, výše uvedené algoritmy implementují pro vyhledávání metody využívající minimální a maximální pozitivní základní vzor. Pattern search algoritmus řeší problémy s nelineárními, lineárními nebo vázanými omezeními a nevyžaduje, aby funkce byla diferencovatelná nebo spojitá.

Možnosti pattern search algoritmu ukazuje následující tabulka 2.3:

Možnosti PS	**Popis**
Polling metody	Rozhodují, jak generovat a vyhodnocovat body daného vzoru a nejvyšší počet bodu generovaných v každém kroku. Umožňuje rovněž ovlivňovat pořadí bodu za účelem zefektivnění.
Vyhledávací metody	Výběr volitelného vyhledávacího kroku muže být více efektivní než poll krok. Lze provést vyhledávání ve vzoru nebo v celém prohledávaném prostoru. K získání dobrého počátečního bodu je možné použít globální vyhledávací metody (kupříkladu genetické algoritmy).
Mesh	Určuje, jak se vzor mění v průběhu iterací a upravuje mesh pro problémy, které se liší v měřítku napříč dimenzemi. Lze vybrat počáteční mesh velikost, expanzní faktor nebo kontrakční faktor. Mesh akcelerátor urychluje konvergenci, když je poblíž minima.
Cache	Ukládá body, ke kterým se dospělo v průběhu optimalizace výpočetně náročných účelových funkcí. Je možné nadefinovat velikost a toleranci cache, kterou pattern search algoritmus využívá a měnit toleranci cashe za běhu, aby optimalizace byla co nejrychlejší a nejefektivnější.
Nastavení nelineárního omezení	Specifikuje parametr penalizace pro nelineární omezení a update-faktor penalizace.

Tabulka 2.3: Možnosti pattern search

2.5 Simulované žíhání (Simulated Annealing Solver)

Simulované žíhání (SA)[8] je pravděpodobnostní optimalizační metoda, která napodobuje fyzikální proces žíhání. Materiál je zahříván na vysokou teplotu a poté pomalým postupným snižováním teploty jsou eliminovány vnitřní vady tělesa, čímž se snižuje celková energie tělesa. Metoda se analogicky v každé iteraci snaží vylepšit stávající minimum zmenšením vyhledávacího prostoru a akceptuje nejen nové body, které snižují hodnotu kriteruální funkce, ale s určitou pravděpodobností i body, které řešení zhoršují. Toto zhoršení zabraňuje uvíznutí algoritmu v lokálním minimu na počátku iterací.

Simulované žíhání umožňuje řešit nepodmíněné nebo vázaně omezené optimalizační problémy a nevyžaduje, aby funkce byla diferencovatelná nebo spojitá.

Pomocí příkazového řádku nebo aplikace je možno využít následujících možností:

- Řešit problém adaptivním simulovaným žíháním, Boltzmannovým žíháním nebo rychlým žíhacím procesem.
- Vytvořit uživatelskou funkci pro definování: žíhacího procesu, nastavení teploty, kritéria přijetí, vykreslení funkce, simulaci žíhání a vlastní formát dat.
- Použít hybridní řešení. Základní SA je kombinováno s další metodou.

Kapitola 3

Implementace

V následující kapitole jsou popsány vybrané metody, které byly použity v praktickém řešení optimalizačních úloh.

3.1 Pattern search

Pattern search[21] je numerická optimalizační metoda, která pracuje pouze s hodnotou kriteriální funkce a nevyžaduje znalost gradientu. Optimalizační metody tohoto typu jsou známy jako direct search metody[16].

Pattern search hledá posloupnost bodů x_0,x_1,x_2,..., které se blíží k optimálnímu bodu. Při přechodu na další bod se hodnota kriteriální funkce snižuje nebo zůstává stejná. *Pattern* je množina vektorů, které algoritmus v každé iteraci používá při hledání dalších bodů, nazývané *mesh* body. Množina vektorů je dána počtem nezávislých proměnných v kriteriální funkci, N, a pozitivním základním souborem. Dva základní soubory, které algoritmus používá, jsou maximální základ s $2N$ vektory a minimální základ s $N + 1$ vektory.

GPS tvoří vektory o velikosti $2N$ pevně daného směru. V GSS je pattern shodný s GPS pattern až na případy, kdy jsou dána lineární omezení a aktuální bod se nachází na hranicích těchto omezení. MADS tvoří sadu náhodně vybraných vektorů. V závislosti na *poll* metodě, je počet vektorů $2N$ nebo $N + 1$. Jako v případě GPS, sada $2N$ se skládá z N kladných a N záporných vektorů, zatímco sada $N + 1$ se skládá z N vektorů a jednoho záporného vektoru, který je dán sumací předchozích.
Pro názaronost bude níže popsán běh GPS algoritmu:
Pattern search začíná v uživateli zadaném výchozím bodu, například $x_0 = [2.1\ 1.7]$.

Iterace 1

V první iteraci, `patternsearch` vygeneruje pattern vektory a rozšíří je o *mesh size*, což je požadovaná vzdálenost mesh bodu od výchozího bodu. V první iteraci je mesh size 1. GPS algoritmus přičte vektory k výchozímu bodu a získá mesh body:

$$\begin{aligned} 1 * [1\ 0] + x_0 &= [3.1\ 1.7] \\ 1 * [0\ 1] + x_0 &= [2.1\ 2.7] \\ 1 * [-1\ 0] + x_0 &= [1.1\ 1.7] \\ 1 * [0\ -1] + x_0 &= [2.1\ 0.7] \end{aligned}$$

Algoritmus spočítá hodnotu kriteriální funkce v mesh bodech, dokud nenarazí na jeden, v němž hodnota kriteriální funkce je menší než ve výchozím bodu. Pokud takový bod existuje, iterace je úspěšná a tento bod se stává výchozím. Výpočet hodnoty kriteriální funkce v mesh bodě je označován jako *poll* (dotazování). První takový bod je [1.1 1.7] nastavnen jako $x_1 = [1.1\ 1.7]$.

Iterace 2

Po úspěšném akceptováním bodu, algortimus rozšíří pattern vektory dvakrát a získá další body:

$$2 * [1\ 0] + x1 = [3.1\ 1.7]$$
$$2 * [0\ 1] + x1 = [1.1\ 3.7]$$
$$2 * [-1\ 0] + x1 = [-0.9\ 1.7]$$
$$2 * [0\ -1] + x1 = [1.1\ -0.3]$$

Algoritmus opět spočítá hodnotu kriteriální funkce v těchto bodech. První bod, v němž hodnota kriteriální funkce je menší než v bodě x_1, je $[-0.9\ 1.7]$ nastaven jako $x2 = [-0.9\ 1.7]$. Protože iterace byla úspěšná, ve třetí iteraci algoritmus rozšíří pattern vektory čtyřikrát a najde bod x_3.

Iterace 4

Stávající bod je $x_3 = [-4.9\ 1.7]$ a mesh size je 8, potom mesh body jsou:

$$8 * [1\ 0] + x3 = [3.1\ 1.7]$$
$$8 * [0\ 1] + x3 = [-4.9\ 9.7]$$
$$8 * [-1\ 0] + x3 = [-12.9\ 1.7]$$
$$8 * [0\ -1] + x3 = [-4.9\ -1.3]$$

V této iteraci v žádném bodě hodnota kriteriální funkce není menší než v bodě x_3 a iterace byla neúspěšná. V takovém případě algoritmus nezmění aktuální bod, v další iteraci $x_4 = x_3$ a sníží mesh size o polovinu.

Algoritmus opakuje tyto kroky tak dlouho, dokud nenajde hledané optimum, nebo dokud nejsou splněna ukončovací kritéria.

3.1.1 Implementace Pattern search v MATLABU

Pattern search vyžaduje alespoň dva vstupní parametry, konkrétně kriteriální funkci a výchozí bod. Algoritmus využívá implementovanou funkci `patternsearch`, která má následující syntax:

```
x = patternsearch(fun,x0)
[x,fval] = patternsearch(fun,x0, ...)
[x,fval,exitflag,output] = patternsearch(fun, ...,options)
```

Kde `fun` je kriteriální funkce a `x0` je výchozí bod. Dále je možné specifikovat omezení a ohraničení. Výstupními argumenty funkce jsou hodnota kriteriální funkce – `fval`, optimum – `x`, `exitflag` – stav při ukončení a `output` – informace o optimalizaci. Funkce poskytuje možnost uživatelského nastavení vytvořením `options` struktury využívající funkci `psoptimset` ve tvaru:

```
options = psoptimset('Param1',value1,'Param2',value2, ...);
```

Ve funkci psoptimset je možné specifikovat:

Volba	**Popis**	**Hodnota**
CompletePoll	Dotáže se na všechny body v aktuální iteraci	'on' {'off'}
InitialMeshSize	Počáteční mesh size pro PS	Kladný skalár {1.0}
MaxIter	Maximální počet iterací	{100*počet_proměnných}
MeshContraction	Mesh contraction faktor, zmenší mesh v případě neúspěšné iterace	Kladný skalár {0.5}
MeshExpansion	Mesh expansion faktor, zvětší mesh v případě úspěšné iterace	Kladný skalár {2.0}
OutputFcns	Specifikuje uživatelskou funkci, která je volána v každé iteraci	Ukazatel na funkci, nebo pole ukazatelů na funkce
PlotFcns	Možnosti vizualizace optimalizačního procesu	@psplotbestf @psplotmeshsize @psplotfuncount @psplotbestx
PollMethod	Polling metoda využitá v PS	{'GPSPositiveBasis2N'} 'GPSPositiveBasisNp1' 'GSSPositiveBasis2N' 'GSSPositiveBasisNp1' 'MADSPositiveBasis2N' 'MADSPositiveBasisNp1'

Tabulka 3.1: Možnosti nastavení psoptimset. (Hodnoty ve složených závorkách jsou defaultní.)

Metoda dále umožňujě vektorizaci úlohy a využití paralelních výpočtů. Podrobnější informace o nastavení viz nápověda systému MATLAB.

Příklad použití metody pattern seach

Níže bude uveden příklad použití pattern search pro minimalizaci kriteriální funkce s nelineárními omezujícími podmínkami a ohraničením:

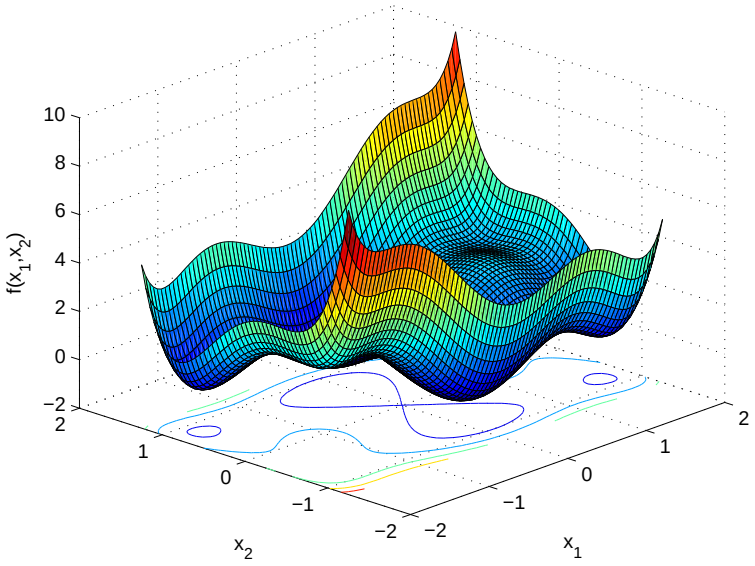
Minimalizujeme kriteriální funkci dvou proměnných x_1 a x_2,

$$\min_x f(x) = (4 - 2.1 * x1^2 + x1^4/3) * x1^2 + x1 * x2 + (-4 + 4 * x2^2) * x2^2 \tag{3.1}$$

takových, které vyhovují následujícím nelineárním podmínkám a ohraničení:

$$x1 * x2 + x1 - x2 + 1.5 <= 0$$
$$10 - x1 * x2 <= 0$$
$$0 <= x1 <= 1$$
$$0 <= x2 <= 13$$

Funkce je známá jako *Six-hump camel back function* a její 3-D mapa je znázorněna v následujícím grafu 3.1:



Obrázek 3.1: Hodnota kriteriální funkce a nalezené optimum (PS).

Vytvoříme m-skript obsahující kriteriální funkci `my_fun.m`:

```
function y = my_fun(x)
   y = (4 - 2.1*x(1)^2 + x(1)^4/3)*x(1)^2 + x(1)*x(2) + ...
   (-4 + 4*x(2)^2)*x(2)^2;
```

Funkce očekává jeden vstupní argument x, kde x má tolik prvků, jako je počet proměnných v úloze. Funkce vypočítá hodnotu kriteriální funkce a vrátí tuto skalární hodnotu jako výstupní argument y.
Dále vytvoříme m-skript s nelineárními omezeními `my_constraint.m`:

```
function [c, ceq] = my_constraint(x)
   c = [1.5 + x(1)*x(2) + x(1) - x(2); -x(1)*x(2) + 10];
   ceq = [];
```

Funkce má jeden vstupní parametr x, kde x má tolik prvků, jako je počet proměnných v úloze. Funkce vypočítá všechna omezení, vrátí vektor nerovnostních omezení c a vektor rovnostních omezení ceq.

Pro minimalizování kriteriální funkce použijeme `patternsearch` funkci a specifikujeme výchozí bod jako druhý vstupní argument. Dále zadefinujeme ohraničení a ukazatele na funkce:

```
ObjectiveFun = @my_fun;
x0 = [0 0];   % výchozí bod
lb = [0 0];   % dolní hranice
ub = [1 13];  % horní hranice
```

```
ConstraintFun = @my_constraint;
[x,fval] = patternsearch(ObjectiveFun,x0,[],[],[],[],lb,ub,ConstraintFun)
```

Po ukončení programu se v příkazovém řádku objeví hodnota kriteriální funkce a optimum:

```
Optimization terminated: mesh size less than options.TolMesh
 and constraint violation is less than options.TolCon.

x = 0.8122   12.3122

fval = 9.1324e+04
```

Pro lepší názornost probíhající optimalizace lze využít vizualizaci zadefinováním `options` struktury. Vykreslíme hodnotu kriteriální funkce v každé iteraci a nalezené optimum, graf 3.2:

```
options = psoptimset('PlotFcns',{@psplotbestf,@psplotbestx});
```

Potom volání funkce je ve tvaru:

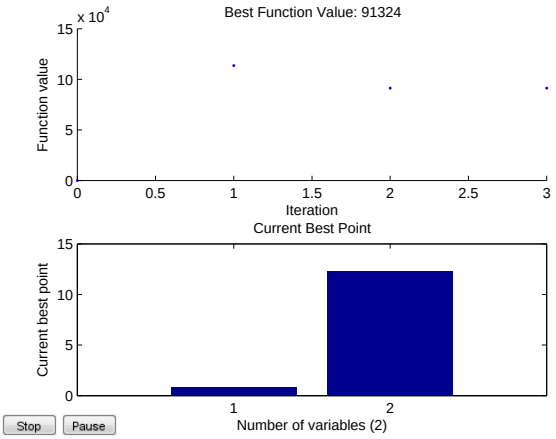
```
[x,fval] = patternsearch(ObjectiveFunction,x0,[],[],[],[],lb,ub,...
ConstraintFunction,options)
```

Výsledek optimalizačního procesu:

```
Optimization terminated: mesh size less than options.TolMesh
 and constraint violation is less than options.TolCon.

x = 0.8122   12.3122

fval = 9.1324e+04
```



Obrázek 3.2: Závislost hodnoty kriteriální funkce na počtu iterací a nalezené optimum (PS).

3.2 Simulované žíhaní (Simulated annealing)

Metoda simulovaného žíhání (SA) [8] patří mezi pravděpodobnostní optimalizační algoritmy. První jednoduché návrhy algoritmu simulovaného žíhání, které pracovaly na principu Monte Carlo, byly zpracovány v roce 1953 N. Metropolisem a jeho kolegy [12]. Kirkpatrick [8] je rozvinul a úspěšně aplikoval na kombinatorické problémy v roce 1983. Inspirací k vytvoření algoritmu byl fyzikální proces žíhání tuhého tělesa. Těleso je zahříváno na vysokou teplotu a poté postupným snižováním teploty ochlazováno. Při vysokých teplotách jsou částice v neuspořádaném stavu a vnitřní energie tělesa vzroste. Postupným snižování teploty vnitří energie tělesa klesá a částice se dostávají do rovnovážného stavu, čímž jsou odstraňovány defekty v tělese. Jestliže je ochlazování dostatečně pomalé, pak rozložení pravděpodobnosti výskytu stavů je určeno Botzmannovým rozdělením. Podle tohoto rozdělení je pravděpodobnost toho, že při teplotě T je těleso ve stavu i s energií E_i, rovna:

$$W_t(E_i) = \frac{1}{Q(T)} e^{-\frac{E_i}{kT}}, \tag{3.2}$$

kde T je Boltzmannova konstanta, a $Q(T)$

$$Q(T) = \sum_i e^{-\frac{E_i}{kT}}, \tag{3.3}$$

Následující tabulka 3.2 ukazuje spojitost mezi fyzikálním procesem žíhání tuhého tělesa a řešením optimalizačních problémů: Převzato z knihy P. Ošmera, Miloš Šeda a spol [15].

žíhání	kombinatorická optimalizace
stav systému	přípustné řešení
energie stavu systému	hodnota kriteriální funkce
změna stavu systému	přechod k sousednímu řešení
teplota	řídící parametr
ustálený stav systému	herustické řešení

Tabulka 3.2: Analogie procesu žíhání a kombinatorické optimalizace

Princip algoritmu

Na počátku je zvolen výchozí bod a inicializována teplota. V každé iteraci je náhodně vybrán bod v blízkosti výchozího bodu a vypočítána hodnota kriteriální funkce v tomto bodě. Délka kroku je v závislosti na teplotě. Jestliže je hodnota kriteriální funkce menší než v předchozím bodě, bod je akceptován a zvolen jako výchozí. Pokud je hodnota kriteriální funkce menší, bod je akceptován s pravděpodobností závisející na rozdílu hodnot kriteriální funkce a aktuální teplotě. Pravděpodobnost akceptování bodu:

$$\frac{1}{1 + exp(\frac{\Delta}{max(T)})} \tag{3.4}$$

Teplota se v každém kroku snižuje tak dlouho, dokud není přijat daný počet bodů. Následně dochází ke znovužíhání. Při znovužíhání se teplota zvýší, což zabraňuje tomu, aby algoritmus uvízl v lokálním minimu. Tento proces se opakuje dokud nejsou splněna ukončovací kritéria.

3.2.1 Implementace simulovaného žíhání v MATLABU

K nalezení optima používá algoritmus simulovaného žíhání implementovanou funkci `simulannealbnd`, která má dva vstupní argumenty, kriteriální funkci a výchozí bod. Funkce má následující sytnax:

```
x = simulannealbnd(fun,x0)
[x,fval] = simulannealbnd(fun,x0,...)
[x,fval,exitflag,output] = simulannealbnd(fun,...,options)
```

Kde `fun` je kriteriální funkce a `x0` je výchozí bod. Dále je možné specifikovat omezení a ohraničení. Výstupními argumenty funkce jsou hodnota kriteriální funkce – `fval`, optimum – `x`, `exitflag` – stav při ukončení a `output` – informace o optimalizaci. Funkce poskytuje možnost uživatelského nastavení vytvořením `options` struktury využívající funkci `saoptimset` ve tvaru:

```
options = saoptimset('Param1',value1,'Param2',value2,...);
```

Následující tabulka 3.3 uvádí možnosti nastavení funkce `saoptimset`:

Volba	**Popis**	**Hodnota**
AcceptanceFcn	Funkce pravděpodobnosti přijetí nového bodu	Uživatelská funkce {@acceptancesa}
AnnealingFcn	Generování nových bodů v další iteraci	@annealingboltz {@annealingfast}
InitialTemperature	Startovní tepolota	Kladný skalár {100}
MaxFunEvals	Maximální počet vyhodnocení kriteriální funkce	Kladné číslo {3000*počet_proměnných}
MaxIter	Maximální počet iterací	Kladné číslo {inf}
PlotFcns	Vykreslovací funkce volané v průběhu iterací	Ukazatel na funkci, nebo na cell pole funkcí @saplotbestf @saplotbestx @saplotf @saplotstopping @saplottemperature
ReannealInterval	Interval znovužíhání	Kladné číslo {100}
TemperatureFcn	Funkce pro snížení teploty v každé iteraci	@temperatureboltz @temperaturefast {@temperatureexp}

Tabulka 3.3: Možnosti nastavení saoptimset. (Hodnoty ve složených závorkách jsou defaultní.)

Podrobnější informace o nastavení viz nápověda systému MATLAB.

Příklad použití metody simulovaného žíhání

Tento příklad ukazuje minimalizaci kriteriální funkce pomocí simulovaného žíhání:

Minimalizujeme kriteriální funkci 3.1 dvou proměnných x_1 a x_2. Pro minimalizaci kriteriální funkce využijeme již zadefinovaný m-skript `my_fun.m` popsaný v metodě pattern search. Máme tedy jeden vstupní argument pro funkci `simulannealbnd` a druhý určíme specifikací výchozího bodu x_0. Dále deklarujeme ukazatel na funkci.

```
ObjectiveFun = @my_fun;
x0 = [-1 1];
[x,fval] = simulannealbnd(ObjectiveFun,x0)
```

Po ukončení programu se v příkazovém řádku objeví hodnota kriteriální funkce a optimum:

```
Optimization terminated: change in best function value less
 than options.TolFun.

x = 0.0902   -0.7134

fval = -1.0316
```

Funkce `simulannealbnd` může být použita pro problémy s ohraničením. Horní ub a dolní lb mez je zde definována jako vektor. V každé dimenzi i řešič akceptuje bod x, který vyhovuje omezení $lb(i) <= x(i) <= ub(i)$. V našem případě bude x z intervalu $< -20, 20 >$, potom ohraničení bude ve tvaru:

```
lb = [-20 -20];
ub = [20 20];
```

Změnu hodnoty kriteriální funkce v každé iteraci, můžeme vykreslit pomocí grafu. Pro porování můžeme také vykreslit nejlepší hodnotu kriteriální funkce, kterou metoda `simulannealbnd` uchovává. Definujeme `options` strukturu ve tvaru:

```
options = saoptimset('PlotFcns',{@saplotf,@saplotbestf})
```

Nyní můžeme opět spustit řešič:

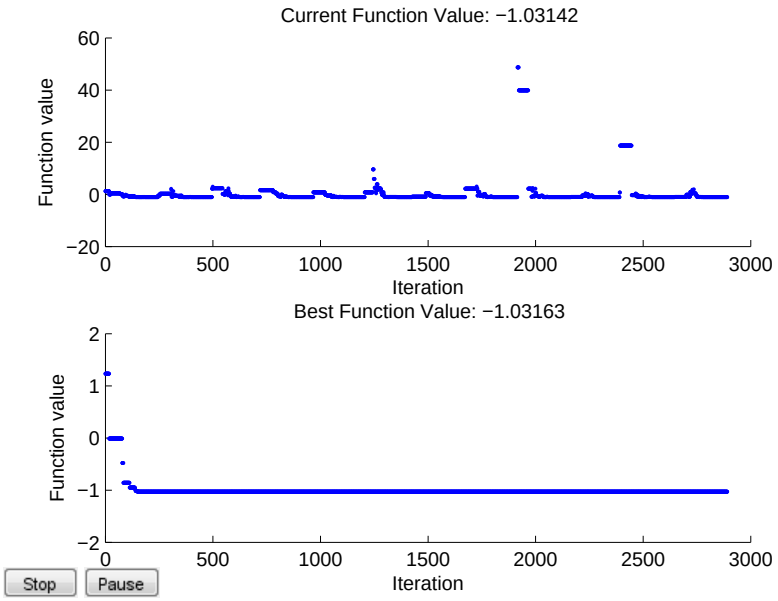
```
[x,fval,exitFlag,output] = simulannealbnd(ObjectiveFun,x0,lb,ub,options);
```

Výsledek optimalizačního procesu a graf 3.3:

```
Optimization terminated: change in best function value less
 than options.TolFun.

x = -0.0881    0.7140

fval = -1.0316
```



Obrázek 3.3: Závislost hodnoty kriteriální funkce na počtu iterací a nejlepší nalezené hodnota kriteriální funkce (SA).

3.3 Genetický algoritmus (Genetic algorithm)

Genetické algoritmy (GA) [7] jsou jedny z nejznámějších evolučních výpočetních technik (EVT). Byly poprvé představeny Johnem Holandem v roce 1975 a vycházejí z principů Darwinovy a Mendelovy teorie evoluce. GA disponují výkonnými vyhledávacími postupy, které vyžadují minimální informace o problému. Staly se efektivní strategií pro řešení optimalizačních problémů.

Pseudokód pro jednoduchý genetický algoritmus je zobrazen níže:

Algorithm 1 Pseudokód průběhu genetického algoritmnu.

```
begin
    Choose a coding to represent variables
    Initialize population
    Evaluate population
    repeat
        Reproduction
        Crossover
        Mutation
        Evaluate population
    until the termination criteria is met
end.
```

Terminologie

Kvůli základnímu principu GA, byla většina terminologie převzata z biologie. Každý jedinec má chromozom, který se skládá z genů tedy parametrů. Soubor genů je nazýván genotyp.

Vhodnost jedince pro evoluční vývoj je názývána *fitness* funkce. Ve standardních optimalizačních algoritmech je známa jako kriteriální funkce. Jedinec je bod, na který můžeme aplikovat fitness funkci. Množina jedinců je nazývána *populace*. V každé iteraci GA provádí sérii výpočtů k produkci nové populace, která je označena jako *nová generace*. K vytvoření další generace jsou vybráni jedinci, nazýváni *rodiče*. Výběr rodičů probíhá pomocí *selekce* a *elitismu*, který jedince s nejlepší hodnotou fitness funkce umístí do nové populace. Po úspěšném výběru rodičů nastává proces *reprodukce*. V první fázi proběhne *křížení* dvou rodičů a v druhé fázi *mutace*, kdy u jednoho rodiče dojde k náhodné změně genomu. Vytvoří se jedinci nové generace, nazývány *děti* a proces reprodukce se opakuje.

3.3.1 Implementace genetického algoritmu v MATLABU

K nalezení optima používá genetický algoritmus implementovanou funkci `ga`, která má následující sytnax:

```
x = ga(fitnessfcn,nvars)
[x,fval] = ga(fitnessfcn,nvars,...)
[x,fval,exitflag,output,population,scores] = ga(fitnessfcn, ...
 nvars,...,options)
```

Kde `fitnessfcn` je fitness funkce a `nvars` je počet proměnných fitness funkce. Dále je možné specifikovat omezení a ohraničení. Výstupními argumenty funkce jsou nejlepší hodnota fitness funkce – `fval`, optimum – `x`, stav při ukončení – `exitflag`, informace o optimalizaci – `output`, matice populace – `population` a vektor hodnot fitness funkce stávající populace – `scores`. Funkce poskytuje možnost uživatelského nastavení vytvořením `options` struktury využívající funkci `gaoptimset` ve tvaru:

```
options = gaoptimset('Param1', value1, 'Param2', value2, ...);
```

V následující tabulce 3.4 jsou uvedeny parametry, které lze nastavit ve funkci `gaoptimset`:

Metoda poskytuje možnost využití pralelního procesu ve smyslu vytvoření několika subpopulací. Přesouvání jedinců mezi subpopulacemi je specifikováno operátorem migrace. Dále je možné vektorizovat úlohu a využit paralelních výpočtů. Podrobnější informace o nastavení viz nápověda systému MATLAB.

Příklad použití genetických algoritmů

Níže bude popsán příklad použití funkce metody genetických algoritmů:

Minimalizujeme kriteriální funkci dvou proměnných x_1 a x_2.

$$\min_x f(x) = (4 - 2.1 * x1^2 + x1^4/3) * x1^2 + x1 * x2 + (-4 + 4 * x2^2) * x2^2$$

Pro minimalizaci kriteriální funkce využijeme již zadefinovaný m-skript `my_fun.m` popsaný v metodě pattern search. Máme jeden vstupní argument pro funkci `ga`. Druhý vstupní argument je $nvars$, v našem případě $nvars = 2$. Dále deklarujeme ukazatel na funkci.

Volba	**Popis**	**Hodnota**
CreationFcn	Funkce k vytvoření počáteční populace	{@gacreationuniform} {@gacreationlinearfeasible}
CrossoverFcn	Funkce křížení	@crossoverheuristic {@crossoverscattered} {@crossoverintermediate} @crossoverarithmetic
EliteCount	Počet elitních jedinců	{floor(.05*PopulationSize)}
Generations	Maximální počet generací	{100*počet_proměnných}
MutationFcn	Funkce mutace	@mutationuniform {@mutationadaptfeasible} {@mutationgaussian}
PlotFcns	Vykreslovací funkce	@gaplotbestf @gaplotdistance @gaplotgenealogy @gaplotmaxconstr @gaplotrange @gaplotselection @gaplotstopping
PopInitRange	Specifikace rozsahu počáteční populace	Matice nebo vektor
PopulationSize	Velikost populace	Kladné číslo
SelectionFcn	Funkce výběru	@selectiontournament @selectionuniform {@selectionstochunif} @selectionroulette

Tabulka 3.4: Možnosti nastavení gaoptimset

```
FitnessFun = @my_fun;
nvars = 2;
[x,fval] = ga(FitnessFun,nvars)
```

Po ukončení programu se v příkazovém řádku objeví hodnota kriteriální funkce a optimum:

```
  Optimization terminated: average change in the fitness value less
  than options.TolFun.

x = -0.1044    0.7148

fval = -1.0308
```

Aby byl výsledek optimalizace názornější, vykreslíme nejlepší a průměrnou hodnotu fitness funkce v každé generaci přidáním `options` struktury:

```
options = gaoptimset('PlotFcns',@gaplotbestf)
```

Opět spustíme řešič:

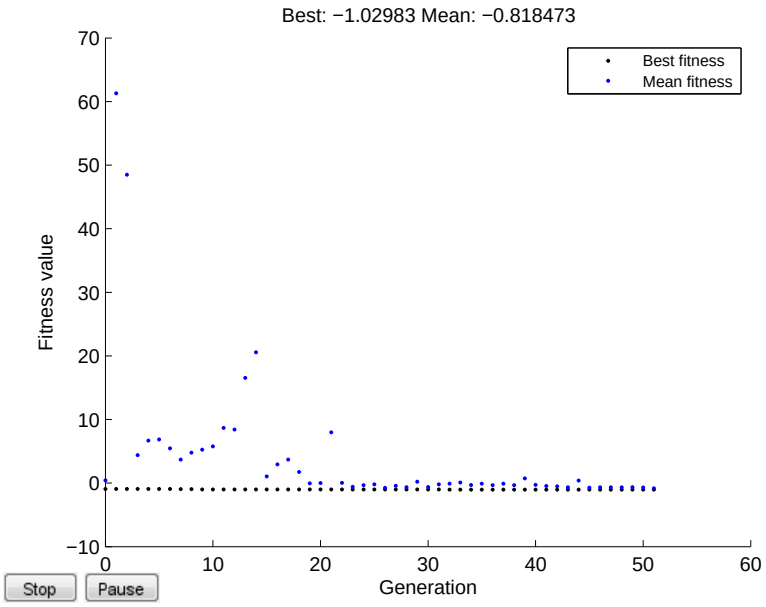
```
[x,fval] = ga(FitnessFun,nvars,options);
```

Výsledek optimalizačního procesu a graf 3.4:

```
  Optimization terminated: average change in the fitness value less
  than options.TolFun.

x = -0.0722    0.7030

fval = -1.0298
```



Obrázek 3.4: Závistlost hodnoty kriteriální funkce na počtu generací (GA).

3.4 Fminsearch – Nelder-Mead algorithm

Fminsearch používá Nelder-Mead algoritmus, který je podrobně popsán v Lagarias [9]. V roce 1962 navrhli Spendely, Hext a Himsworth[19] simplexovou metodu $n + 1$ bodů v n dimenzionálním vektorovém prostoru pro řešení nelineárních optimalizačních úloh. Metoda je založena na principu postupného nahrazování bodu s největší hodnotou kriteriální funkce. Nový bod je zrcadlovým obrazem vzhledem k těžišti původního bodu. Velikost simplexu se může měnit, ale tvar je stále zachován. John Nelder a Roger Mead [13] v roce 1965 publikovali modifikovanou simplexovou metodu. Navrhli přizpůsobení tvaru simplexu podle povrchu kriteriální funkce a úspěšnosti zobrazeného bodu. Tento algoritmus se stal jednou z nejpoužívanejších metod pro vícedimenzionální nepodmíněné optimalizační problémy. Patří mezi přímé vyhledávávací metody. Pro úplné definování metody musí být

specifikovány parametry *reflection* (ρ), *expansion* (χ), *contraction* (γ) a *shrinkage* (σ). Univerzální výběr parametrů ve standardním Nelder-Mead algoritmu je:

$$\rho = 1, \chi = 2, \gamma = \frac{1}{2}, \sigma = \frac{1}{2} \tag{3.5}$$

Jedna iterace algoritmu Nelder-Mead

1. **Order**. Seřadí $n+1$ vrcholů tak, aby platilo $f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$.

2. **Reflect**. Vypočítá *reflection point* $\mathbf{x}_r$ jako

   $$\mathbf{x}_r = \overline{\mathbf{x}} + \rho(\overline{\mathbf{x}} - \mathbf{x}_{n+1}), \tag{3.6}$$

   kde $\overline{\mathbf{x}}$ je průměr všech vrcholů vyjma $\mathbf{x}_{n+1}$. Jestliže $f_1 \leq f_r < f_n$, pak je reflexní bod $\mathbf{x}_r$ přijat a iterace je ukončena.

3. **Expand** Jestliže $f_r < f_1$, pak je vypočítán *expansion point* $\mathbf{x}_e$ vztahem

   $$\mathbf{x}_e = \overline{\mathbf{x}} + \chi(\mathbf{x}_r - \overline{\mathbf{x}}), \tag{3.7}$$

   a vypočítá funkční hodnotu v bodě $\mathbf{x_e}$. Jestliže $f_e < f_r$, bod $\mathbf{x}_e$ je přijat a iterace ukončena, jinak (je-li $f_e \geq f_r$) je přijat bod $\mathbf{x_r}$ a iterace je ukončena.

4. **Contract**. Jestliže $f_r \geq f_n$, je provedena *contraction* mezi $\overline{\mathbf{x}}$ a lepším z $\mathbf{x}_{n+1}$ a $\mathbf{x}_r$ dle hodnoty kriteriální funkce.

   (a) **Outside**. Jestliže $f_n \leq f_r < f_{n+1}$ provede se *outside contraction*:

   $$\mathbf{x}_c = \overline{\mathbf{x}} + \gamma(\mathbf{x}_r - \overline{\mathbf{x}}) \tag{3.8}$$

   a vypočítá funkční hodnotu v bodě $\mathbf{x}_c$. Jestliže $f_c \leq f_r$, bod $\mathbf{x}_c$ je přijat a iterace je ukončena, jinak přejde na krok 5.

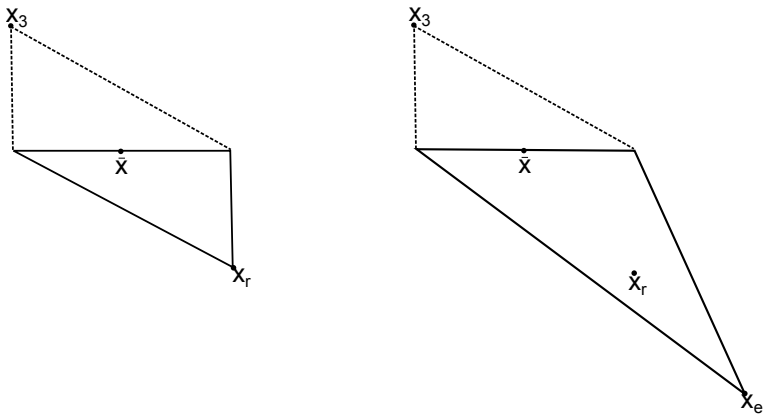
   (b) **Inside** Jestliže $f_r \geq f_{n+1}$ provede se *inside contraction*:

   $$\mathbf{x}_{cc} = \overline{\mathbf{x}} - \gamma(\overline{\mathbf{x}} - \mathbf{x}_{n+1}) \tag{3.9}$$

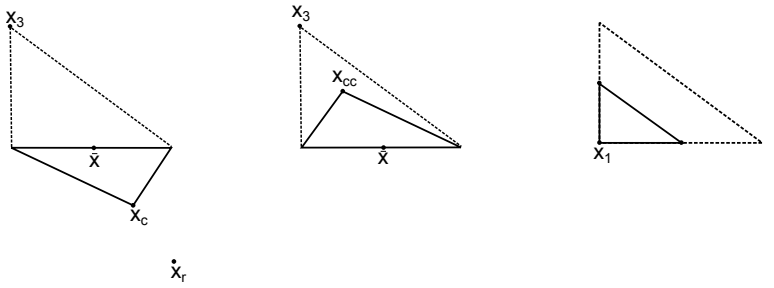
   a vypočítá funkční hodnotu v bodě $\mathbf{x}_{cc}$. Jestliže $f_{cc} < f_{n+1}$ bod $\mathbf{x}_{cc}$ je přijat a iterace je ukončena, jinak přejde na krok 5.

5. **Shrink** Vypočítá hodnotu kriteriální funkce pro všechny vrcholy $\mathbf{v}_i$, $\mathbf{v}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1), i = 2, \ldots, n+1$. Neseřazené vrcholy simplexu jsou v další iteraci složeny z $\mathbf{x}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n+1}$

Následující obrázky 3.5, 3.6 znázorňují kroky Nelder-Mead algoritmu pro 2D problém při použití koeficientů ze vzorce 3.5.



Obrázek 3.5: Stav Nelder-Meadova algoritmu po reflection a expansion. Původní trojúhelník je zobrazen čárkovaně.



Obrázek 3.6: Stav Nelder-Meadova algoritmu po outside contraction, inside contraction a shrink. Původní trojúhelník je zobrazen čárkovaně.

3.4.1 Implementace fminsearch v MATLABU

Fminsearch může nalézt pouze lokální řešení a to reálných funkcí. Pokud funkce má komplexní proměnné, je nutné ji rozdělit na reálnou a imaginární část. K nalezení optima používá algoritmus implementovanou funkci `fminsearch`, která má následující syntax:

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

Kde `fun` je kriteriální funkce a `x0` je výchozí bod. Výstupními argumenty funkce jsou hodnota kriteriální funkce – `fval`, optimum – `x`, `exitflag` – stav při ukončení a `output` – informace o optimalizaci.
Funkce poskytuje možnost uživatelského nastavení vytvořením `options` struktury využívající funkci `optimset` ve tvaru:

```
options = optimset('param1',value1,'param2',value2,...);
```

Následující tabulka 3.5 uvádí možnosti nastavení funkce `optimset`:

Volba	**Popis**	**Hodnota**
MaxFunEvals	Maximální počet vyhodnocení kriteriální funkce	Kladné číslo
MaxIter	Maximální počet iterací	Kladné číslo
PlotFcns	Vykreslovací funkce, které jsou volány v každé iteraci	Uživatelské funkce @optimplotx @optimplotfval @optimplotfunccount

Tabulka 3.5: Možnosti nastavení optimset

Podrobnější informace o nastavení viz nápověda systému MATLAB.

Příklad použití metody fminsearch

Tento příklad ukazuje minimalizaci kriteriální funkce pomocí fminsearch:

Minimalizujeme kriteriální funkci dvou proměnných x_1 a x_2.

$$\min_x f(x) = (4 - 2.1 * x1^2 + x1^4/3) * x1^2 + x1 * x2 + (-4 + 4 * x2^2) * x2^2$$

Pro minimalizaci kriteriální funkce využijeme již zadefinovaný m-skript `my_fun.m` popsaný v metodě pattern search. Máme tedy jeden vstupní argument pro funci `fminsearch` a druhý určíme specifikací výchozího bodu x_0. Dále deklarujeme ukazatel na funkci a pomocí výstupních argumentů `exitflag` a `output` vypíšeme stav při ukončení a informace o optimaliazci.

```
ObjectiveFun = @my_fun;
x0 = [-1 1];
[x,fval,output] = fminsearch(ObjectiveFun,x0)
```

Po ukončení programu se v příkazovém řádku objeví hodnota kriteriální funkce, optimum, stav při ukončení a informace o optimalizaci:

```
x = -0.0898
    0.7127

fval = -1.0316

exitflag = 1

output =
    iterations: 44
     funcCount: 83
     algorithm: 'Nelder-Mead simplex direct search'
       message: [1x194 char]
```

Pro lepší názornost optimalizace vykreslíme změnu hodnoty kriteriální funkce v každé iteraci. Definujeme `options` strukturu ve tvaru:

```
options = optimset('PlotFcns',@optimplotfval)
```

Opět spustíme řešič:

```
[x,fval,exitflag, output] = fminsearch(ObjectiveFun,x0,options);
```

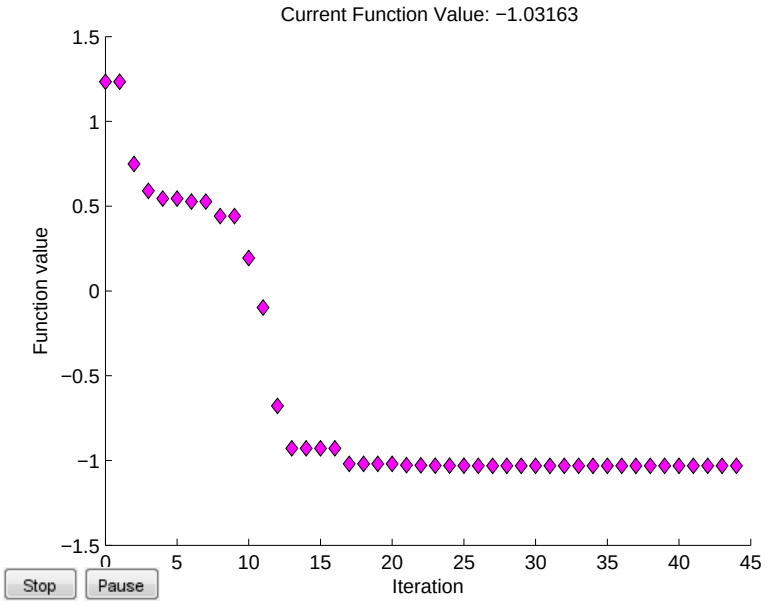
Výsledek optimalizačního procesu a graf 3.7:

```
x = -0.0898
    0.7127

fval = -1.0316

exitflag = 1

output =
    iterations: 44
     funcCount: 83
     algorithm: 'Nelder-Mead simplex direct search'
       message: [1x194 char]
```



Obrázek 3.7: Závistlost hodnoty kriteriální funkce na počtu iterací (fminsearch).

Kapitola 4

Experiment

Tato kapitola je věnovaná řešení optimalizačních úloh metodami popsaných v kapitole implementace. Primárně se jedná o metody implementované v Global Optimization Toolbox, avšak pro srovnání dosažených výsledků byl vybrán jeden exaktní matematický algoritmus.

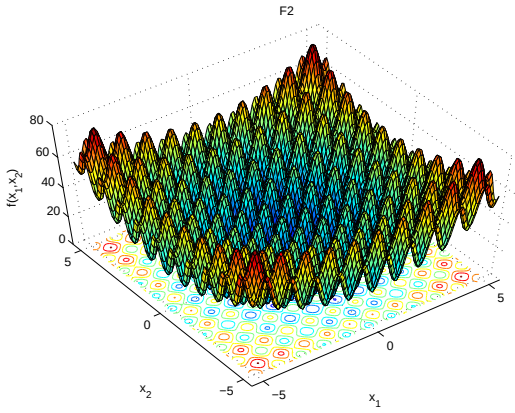
4.1 Rastriginova funkce

Pro vyhodnocení efektivity algoritmu se v matematické optimalizaci používají různé testovací funkce. Zpravidla se jedná o funkce, u kterých je řešení známé, ale numericky náročné. První ze zvolených testovacích funkcí je Rastriginova funkce, označovaná jako *F6*, která má více lokálních minim. Jako 2D funkce byla navržena L.A. Rastriginem a až později zobecněna. Nalezení globálního optima je obtížné kvůli velkému počtu extrémů.

Definice funkce pro n dimenzí:

$$f(x,y) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right], \tag{4.1}$$

kde $A = 10$ a $x_i \in < -5.12\ 5.12 >$. Globální minimum se nachazí v bodě [0 0] a hodnota kriteriální funkce je v tomto bodě nulová. Pojedem dimenze bude ztotožňován s počtem optimálních parametrů úlohy. 3D zobrazení Rastriginovy funkce je na obrázku 4.1.



Obrázek 4.1: Rastriginova funkce

Funkce F6 byla využita pro testování genetických algoritmů, metody simulovaného žíhání, pattern search a fminsearch. Výsledky jednotlivých metod byly zpracovány a na závěr srovnány. Všechny testy byly provedeny stokrát kvůli objektivnosti výsledků dosažených v případě heuristických algoritmů.

Testování genetických algoritmů

Genetické algoritmy umožňují velmi variabilní nastavení svých parametrů, proto proběhlo širší testování vhodných hodnot parametrů pro daný optimalizační problém.

Nastavení GA

Níže je uvedeno možné nastavení genetických algoritmů:

```
PopulationSize = 100
PopInitRange = [-5.12 5.12]
SelectionFcn = selectiontournament
SelectionSize = 2
EliteCount = 2
MutationFcn = mutationadaptfeasible
CrossoverFcn = crossoverscattered
Generations = 1000
StallGenLimit = Inf
```

Testy proběhly pro dvě různá nastavení metod křížení (crossoverheuristic, crossoverarithmetic) a pro tři varianty turnajové selekce s velikostí turnaje 2, 4, 8 a tři varianty elitismu o velikostech 2, 5 a 10 jedinců. V tabulce 4.1 jsou zobrazeny výsledky hodnot fitness funkce reprezentované její průměrnou hodnotou (mean), směrodatnou odchylkou (std), mediánem (median) a minimem (min) pro zvolené dimenze.

	crossover	**heuristic**			**arithmetic**		
F6	**sel**	**2**	**4**	**8**	**2**	**4**	**8**
2D	**mean**	0,03	6,97E-10	2,81E-07	0,01	1,80E-11	6.,05E-08
	std	0,17	3,30E-09	2,72E-07	0,1	1,23E-11	1,76E-07
	median	0	6,77E-11	2,32E-07	0	0	7.74E-13
	min	0	0	0	0	0	0
5D	**mean**	0	7,05E-08	7,62E-07	2.49E-01	6,00E-02	4,74E-02
	std	0	1,11E-07	3,29E-07	6,82E-01	2,85E-02	4,74E-02
	median	0	2,23E-08	7,42E-07	0	2,94E-09	6,41E-07
	min	0	1,31E-06	5,77E-08	0	0	1,99E-13
10D	**mean**	1,20E-07	2,46E-05	3,63E-06	1,07	2,84E-1	1,56E-2
	std	6,85E-07	8,37E-05	5,00E-06	1,45	6,93E-01	1,23E-01
	median	2,13E-14	2,69E-06	1,98E-06	1	3,98E-07	1,65E-06
	min	0	1,12E-07	5,25E-07	1,02E-08	1,44E-08	1,60E-07
50D	**mean**	3,85E1	1,25E1	1,30E1	2,10E2	9,06E1	3,48E1
	std	1,59E1	6,46	4,26	4,84E1	3,04E1	1,34E1
	median	3,48E1	1,15E1	1,22E1	2,10E1	8,50E1	3,15E1
	min	1,02E1	1,76	4,64	1,07E2	3,87E1	1,42E1

Tabulka 4.1: Srovnání vlivu metody křížení a selekčního tlaku při nastavení elitismu 2 a populace 100 (úloha F6).

Další experimenty jsou poplatny heuristickému křížení. Také byl zkoumán vliv počtu generací na zlepšení hodnoty fitness funkce. Maximální počet generací byl nastaven na 1000, 5000 a 10000. V tabulkách 4.2, 4.3, 4.4, 4.5 jsou uvedeny výsledky testů.

Zvýrazněné hodnoty v tabulkách jsou nejlepší z hlediska hodnoty fitness funkce a budou dále použity pro srovnání dosažených výsledků.

	elitism	2			5			10		
2D	selection	2	4	8	2	4	8	2	4	8
1k	mean	0,03	6,97E-10	2,81E-07	0,01	1,61E-09	3,31E-07	0	4,02E-09	2,07E-07
	std	0,17	3,30E-09	2,72E-07	0,1	1,12E-08	2,67E-07	0	9,02E-09	2,45E-07
	median	0	6,77E-11	2,32E-07	0	9,95E-11	2,67E-07	0	5,71E-10	1,14E-07
	min	0	0	0	0	0	1,07E-11	0	0	1,27E-11
5k	mean	0,01	9,80E-11	2,21E-07	0,01	1,61E-10	2,85E-07	0,01	2,28E-10	2,22E-07
	std	0,1	2,24E-10	2,16E-07	0,1	2,81E-10	2,40E-07	0,1	2,79E-10	2,26E-07
	median	0	2,99E-12	1,54E-07	0	1,51E-11	2,34E-07	0	1,18E-10	1,45E-07
	min	0	0	5,83E-13	0	0	2,22E-11	0	0	6,29E-11
10k	mean	0,01	**1,05E-10**	2,25E-07	0,01	1,82E-10	2,76E-07	0	2,16E-10	2,92E-07
	std	0,1	**2,49E-10**	2,41E-07	0,1	3,04E-10	2,53E-07	0	3,30E-10	2,72E-07
	median	0	**0**	1,27E-07	0	1,59E-11	2,11E-07	0	4,50E011	2,58E-07
	min	0	**0**	4,12E-13	0	0	7,82E-13	0	0	0

Tabulka 4.2: Výsledky testů GA na úloze F6 pro 2 dimenze

	elitism	2			5			10		
5D	selection	2	4	8	2	4	8	2	4	8
1k	mean	0	7,05E-08	7,62E-07	0	1,01E-07	7,98E-07	2,49E-15	1,41E-07	7,88E-07
	std	0	1,11E-07	3,29E-07	0	1,25E-07	3,60E-07	1,90E-14	1,59E-07	3,33E-07
	median	0	2,23E-08	7,42E-07	0	4,40E-08	7,93E-07	0	6,24E-08	7,96E-07
	min	0	1,31E-06	5,77E-08	0	1,07E-09	1,13E-07	0	6,53E-10	7,45E-08
5k	mean	0	8,03E-10	7,99E-07	0	1,16E-09	7,28E-07	0	2,67E-08	8,07E-07
	std	0	6,28E-10	,19E-07	0	1,16E-09	2,96E-07	0	3,32E-09	3,26E-07
	median	0	6,94E-10	7,95E-07	0	9,60E-10	6,99E-07	0	1,58E-09	7,57E-07
	min	0	0	1,31E-07	0	2,78E-11	8,22E-08	0	5,99E-12	1,11E-07
10k	mean	**0**	5,82E-10	7,94E-07	0	7,80E-10	7,68E-07	0	9,34E-10	7,84E-07
	std	**0**	5,16E-10	3,30E-07	0	5,82E-10	3,11E-07	0	7,31E-10	3,14E-07
	median	**0**	4,96E-10	7,98E-07	0	6,27E-10	7,77E-07	0	8,10E-10	7,82E-07
	min	**0**	7,11E-15	3,50E-08	0	4,69E-13	5,66E-08	0	1,37E-12	1,03E-07

Tabulka 4.3: Výsledky testů GA na úloze F6 pro 5 dimenzí

	elitism	2			5			10		
10D	selection	2	4	8	2	4	8	2	4	8
1k	mean	1,20E-07	2,46E-05	3,63E-06	1,88E-11	2,24E-05	6,71E-06	5,41E-08	4,01E-04	3,62E-06
	std	6,85E-07	8,37E-05	5,00E-06	9,19E-11	1,08E-04	2,42E-05	1,76E-07	1,42E-04	5,60E-06
	median	2,13E-14	2,69E-06	1,98E-06	8,31E-13	3,70E-06	1,82E-06	1,48E-08	6,91E-06	1,88E-06
	min	0	1,31E-07	5,25E-07	0	2,26E-07	2,01E-07	4,62E-11	5,71E-07	6,22E-07
5k	mean	0	1,05E-08	1,46E-06	0	1,78E-08	1,39E-06	4,26E-16	2,58E-08	1,48E-06
	std	0	7,95E-09	4,15E-07	0	1,09E-08	4,31E-08	2,44E-15	1,43E-08	4,20E-07
	median	0	7,44E-09	1,40E-06	0	1,68E-08	1,42E-06		2,25E-08	1,47E-06
	min	0	6,71E-10	5,88E-07	0	1,68E-09	4,58E-07	0	2,63E-09	4,39E-07
10k	mean	0	2,71E-09	1,37E-06	**0**	4,47E-09	1,43E-06	0	8,20E-09	1,43E-06
	std	0	2,11E-09	4,01E-07	**0**	3,62E-09	4,33E-07	0	6,06E-09	4,36E-07
	median	0	2,30E-09	1,38E-06	**0**	3,33E-09	1,34E-06	0	6,45E-09	1,39E-06
	min	0	1,16E-10	4,50E-07	**0**	3,87E-12	2,14E-07	0	5,08E-10	4,69E-07

Tabulka 4.4: Výsledky testů GA na úloze F6 pro 10 dimenzí

	elitism	2			5			10		
50D	selection	2	4	8	2	4	8	2	4	8
1k	mean	3,85E01	1,28E01	1,30E01	2,02E01	1,06E01	1,49E0	2,01E01	1,36E01	1,75E01
	std	1,59E01	6,46E01	4,26	9,66	6,1	5,1	1,03E01	6,28	5,13
	median	3,48E01	1,15E01	1,22E01	1,80E01	9,11	1,39E01	1,89E01	1,19E01	1,74E1
	min	1,02E01	1,76	4,64	5,07	2,35	5,56	4,11	2,77	8,09
5k	mean	01,10E-01	7,40E-04	5,00E-02	7,29E-09	0	6,00E-2	2,85E-06	0	9,00E-02
	std	4,75E-1	6,07E-04	1,01E-01	1,56E-08	0	5,26E-02	6,55E-06	0	7,59E-02
	median	6,27E-08	5,85E-04	3,67E-02	2,28E-09	0	5,95E-02	1,38E-06	0	1,23E-02
	min	3,31E-11	6,59E-05	0	5,88E-11	1,06E-04	0	3,07E-07	1,89E-04	1,42E-02
10k	mean	**1,42E-04**	1,55E-06	6,76E-05	1,20E-13	2,70E-06	1,24E-04	7,05E-04	3,82E-06	2,37E-04
	std	**3,56E-14**	6,01E-07	9,81E-05	6,90E-14	1,02E-06	1,98E-04	1,19E-10	1,19E-10	2,71E-04
	median	**0**	1,46E-06	3,08E-05	1,14E-13	2,71E-06	5,65E-05	3,23E-13	3,69E-06	1,44E-04
	min	**0**	7,07E-07	8,18E-06	0	8,03E-07	6,13E-06	1,88E-12	1,80E-06	7,45E-06

Tabulka 4.5: Výsledky testů GA na úloze F6 pro 50 dimenzí

Testování metody simulovaného žíhání

Metoda byla otestována pro různé počáteční teploty a pro sto různých výchozích bodů z intervalu $x_0 = [-5.12\ 5.12]$. Dále byly otestovány různé kombince dvou parametrů. První z nich `AnnealingFcn` (annealingfast, annealingboltz), který generuje, v případě nastavení annealingfast, náhodně nové body s rovnoměrným rozložením v délce kroku odpovídající teplotě a v případě annealingboltz v délce kroku odpovádající druhé odmocnině z teploty. Druhý parametr je funkce pro aktualizaci teploty `TemperatureFcn` (temperatureexp, temperaturefast, temperatureboltz). Testování proběhlo pro různý počet dimenzí, konkrétně

pro 2, 5 a 10. Pro větší počet dimenzí v případě úlohy F6 dosahuje algoritmus natolik nepřesných výsledků, že je nelze srovnat s ostatními metodami. V tabulkách 4.6, 4.7 a 4.8 je uveden medián a minimum kriteriální funkce. Počáteční teploty jsou uvedeny v závorkách u příslušného nastavení.

2D		**boltz**	**exp**	**fast**
boltz	**mean**	1,05(100)	7,78E-01(2000)	1,5E-01(1000)
	min	1,35E-03(800)	0(200)	6,63E-4(1500)
fast	**mean**	1,02(100)	1,07E-06(400)	2,32E-01(1500)
	min	0(100)	1,29E-10(1500)	1,55E0-7(1)

Tabulka 4.6: Výsledky testu SA na úloze F6 pro 2 dimenze

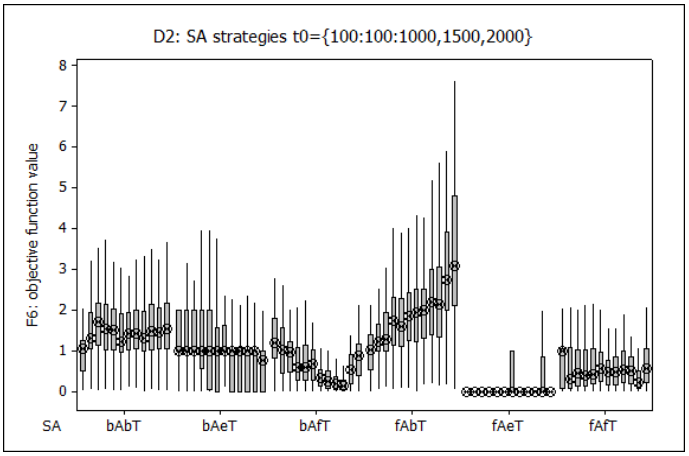
5D		**boltz**	**exp**	**fast**
boltz	**mean**	1,53E01(100)	6,96(2000)	9,73(500)
	min	5,85(100)	6,51E-12(2000)	1,68(200)
fast	**mean**	1,66E01(100)	2,99(400)	6,22(600)
	min	2,41(1)	2,80E-05(300)	2,54E-01(900)

Tabulka 4.7: Výsledky testů SA na úloze F6 pro 5 dimenzí

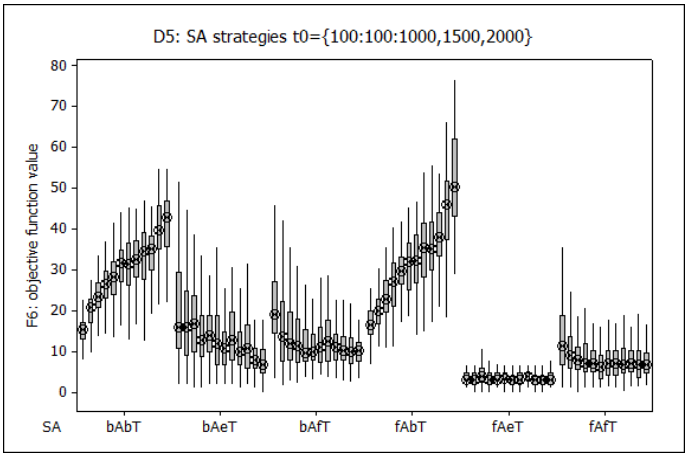
10D		**boltz**	**exp**	**fast**
boltz	**mean**	5,95E01(100)	2,98E01(400)	3,21E01(2000)
	min	2,79E01(1)	2,80E-05(400)	2,56E-01(1000)
fast	**mean**	7,08E01(100)	2,58E01(200)	2,31E01(1500)
	min	3,15E01(1)	5,54(300)	5,76(1500)

Tabulka 4.8: Výsledky testů SA na úloze F6 pro 10 dimenzí

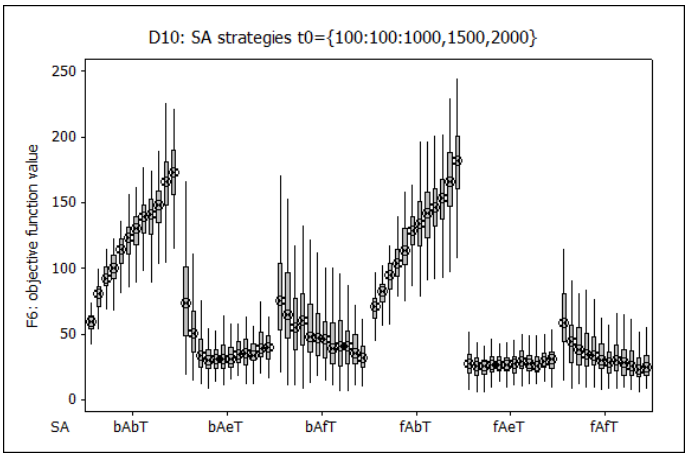
Do grafů 4.2, 4.3 a 4.4 byly vyneseny hodnoty kriteriální funkce v závislosti na nastavení metody pro $T_0 = 100, 200, 1000, 1500, 2000$. Zkratky použité v grafu: annealingfast – fA, annealingboltz – bA, temperatureexp – eT, temperaturefast – fT, temperatureboltz – bT.



Obrázek 4.2: Výsledné testy SA pro různá nastavení v 2D



Obrázek 4.3: Výsledné testy SA pro různá nastavení v 5D



Obrázek 4.4: Výsledné testy SA pro různá nastavení v 10D

Ve většině případů je nastavení `annealingfast` a `temperatureexp` nejvýhodnější. Výsledky tohoto nastavení budou použity pro srovnání s dalšími metodami.

Testování metody pattern search

Metody byla otestována pro sto různých počátečních bodů z intervalu $x_0 \in < -5.12,\ 5.12 >$ a pro dimenze 2, 5, 10 a 50.

Nastavení metody pattern search je následující:

```
CompletePoll = on
MaxIter = 1000
PollMethod = MADSPositiveBasis2N
TolFun = 1e-10
```

Níže je uvedena tabulka 4.9 výsledků testování:

F6	2D	5D	10D	50D
mean	2,19E-01	1,09E-01	1,89E-01	1,89E-01
std	4,38E-01	3,13E-01	4,17E-01	4,40E-01
median	9,61E-10	3,04E-09	5,17E-09	1,73E-07
min	2,12E-12	2,72E-10	1,31E-10	1,82E-08

Tabulka 4.9: Výsledky PS testů

Testování metody fminsearch

Metody byla otestována pro sto různých počátečních bodů z intervalu $x_0 = [-5.12\ 5.12]$ a pro dimenze 2, 5, 10 a 50.
V následující tabulce 4.10 jsou zobrazeny výsledky testování.

F6	2D	5D	10D	50D
mean	1,88E-10	4,87E-10	9,43E-10	4,87E-09
std	2,11E-10	3,38E-10	4,45E10	1,13E-09
median	9,84E-11	3,96E-10	9,46E-10	4,74E-09
min	1,35E-13	2,23E-11	1,92E-10	2,46E-09

Tabulka 4.10: Výsledky fminsearch testů

Srovnání výsledků

Z genetických algoritmů a metody simulovaného žíhání byly vybrány nejlepší hodnoty dosažené testováním. Exaktní matematickou metodu zde reprezentuje algoritmus fminsearch dále v tabulkách označenou jako (N-M).

V tabulkách 4.11, 4.12, 4.13, 4.14, 4.14 jsou uvedeny dosažené výsledky pro dimenze 2, 5, 10 a 50 jednotlivými metoda:

2D	GA	SA	PS	N-M
median	0	1,07e-06	9,61E-10	9,84E-11
min	0	1,29e-10	2,12E-12	2,11E-10

Tabulka 4.11: Srovnání jednotlivých metod na úloze F6 pro 2 dimenze.

5D	GA	SA	PS	N-M
median	0	2,99	3,04E-09	3,96E-10
min	0	2,80E-05	2,72E-10	3,38E-10

Tabulka 4.12: Srovnání jednotlivých metod na úloze F6 pro 5 dimenzí.

10D	GA	SA	PS	N-M
median	0	2,58E01	5,17E-09	9,46E-10
mini	0	5,54	1,31E-10	4,45E-10

Tabulka 4.13: Srovnání jednotlivých metod na úloze F6 pro 10 dimenzí.

50D	GA	PS	N-M
median	0	1,73E-07	4,74E-09
min	0	1,82E-08	1,13E-09

Tabulka 4.14: Srovnání jednotlivých metod na úloze F6 pro 50 dimenzí.

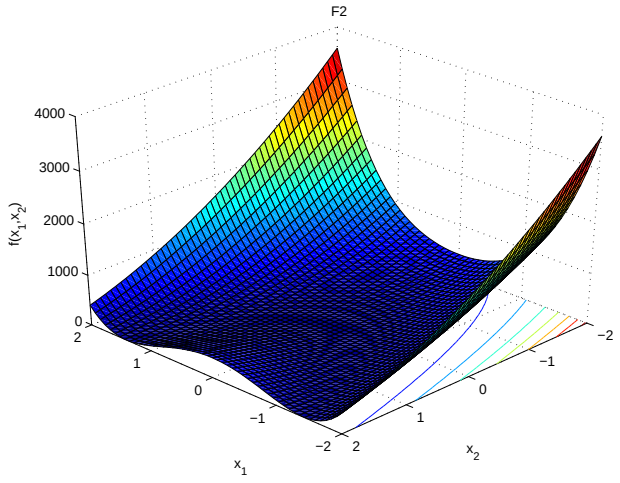
4.2 Rosenbrockova funkce

Další z testovací funkcí je Rosenbrockova funkce, také označována *F2*, kterou v roce 1960 popsal Howard H. Rosenbrock[17]. Globální minimum funkce leží v dlouhém, uzkém údolí parabolického tvaru a jeho nalezení je obtížné.

Definice funkce:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{4.2}$$

Globální minimum se nachazí v bodě [0 0] a hodnota kriteriální funkce je v tomto bodě nulová. 3D mapa Rosenbrockovy funkce je na následujícím obrázku:



Obrázek 4.5: Rosenbrockova funkce

Funkce byla využita pro testování genetických algoritmů, metody simulovaného žíhání, pattern search a fminsearch. Výsledky jednotlivých metod byly zpracovány a vyhodnoceny. Všechny testy byly provedeny na sto opakování kvůli objektivnosti výsledků dosažených v případě heuristických algoritmů.

Testování genetických algoritmů

Při řešení Rastriginovy funkce se ukázal dvě varianty nastavení jako efektivní. Pro řešení Rosenbrockovy funkce byla vybrána varianta `elitismus = 5` při stejném počtu dimenzí.

V tabulkách 4.15, 4.16, 4.17, 4.18 je uveden vliv selekce na hodnotu kriteriální funkce reprezentované její průměrnou hodnotou, minimální hodnotou, mediánem a směrodatnou odchylkou. Maximální počet generací byl 10000.

2D	2	4	8
mean	0	1,34E-02	3,60E-02
std	0	7,67E-02	1,15E-01
median	0	1,07E-10	1,30E-03
min	0	6,55E-17	2,74E-10

Tabulka 4.15: Výsledky GA pro 2 dimenze

5D	2	4	8
mean	2,04E-01	1,34	1,22
std	7,59E-01	1,88	1,93
median	5,10E-03	8,53E-02	1,19E-02
min	3,07E-05	1,49E-04	9,62E-07

Tabulka 4.16: Výsledky GA pro 5 dimenzí

10D	2	4	8
mean	8,44E-01	2,13	1,46
std	1,33	1,96	1,85
median	3,71E-01	1,8	5,18E-01
min	4,83E-05	2,60E-03	4,58E-05

Tabulka 4.17: Výsledky GA pro 10 dimenzí

50D	2	4	8
mean	5,75E01	9,56E01	6,08E01
std	2,33E01	3,99E01	c3,42E01
median	4,70E01	8,64E01	4,98E01
min	3,83E01	2,00E01	1,54E01

Tabulka 4.18: Výsledky Ga pro 50 dimenzí

Testování metody simulovaného žíhání

Nejlepší nastavení při řešení Rastriginovy funkce bylo použito i pro řešení Rosenbrockovy funkce. Metoda byla otestována pro dvě různé počáteční teploty, pro sto různých výchozích bodů z intervalu $x_0 = [-2\ 2]$ a pro zvolené dimenze 2, 5, 10 a 50.

Nastavení SA

Níže je uvedeno nastavení metody simulovaného žíhání:

```
InitialRemperature = 100
TemperatureFcn = temperatureexp
AnnealingFcn = annealingfast
TolFun = 1e-10
```

V tabulkách 4.19, 4.20, 4.21, 4.22 je uveden vliv teploty na hodnotu kriteriální funkce reprezentované její průměrnou hodnotou, minimální hodnotou, mediánem a směrodatnou odchylkou.

2D	$t = 100$	$t = 150$
mean	1,06E-01	9,94E-03
std	1,05E+00	8,47E-02
median	3,70E-05	4,34E-05
min	8,15E-08	1,41E-08

Tabulka 4.19: Výsledky SA testů pro 2 dimenze

5D	$t = 100$	$t = 150$
mean	3,92E-01	2,76E-03
atd	2,82E+00	6,78E-03
median	2,73E-04	3,49E-04
min	5,75E-07	2,09E-08

Tabulka 4.20: Výsledky SA testů pro 5 dimenzí

10D	$t = 100$	$t = 150$
mean	1,76E-02	1,03E-01
std	9,44E-02	8,62E-01
median	6,24E-04	7,52E-04
min	4,97E-08	8,91E-08

Tabulka 4.21: Výsledky SA testů pro 10 dimenzí

50D	$t = 100$	$t = 150$
mean	2,15E+00	3,42E+00
std	9,60E+00	1,52E+01
median	8,94E-03	8,80E-03
min	2,34E-07	6,14E-08

Tabulka 4.22: Výsledky SA testů pro 50 dimenzí

Testování metody pattern search

Metody byla otestována pro sto různých počátečních bodů z intervalu $x_0 = [-2\ 2]$ a pro dimenze 2, 5, 10 a 50.

Nastavení PS

Nastavení metody pattern search je následující:

```
CompletePoll = on
MaxIter = 1000
PollMethod = MADSPositiveBasis2N
TolFun = 1e-10
```

Pro uvedené nastavení je v tabulce 4.23 zobrazena hodnota kriteriální fukce reprezentovaná její průměrnou hodnotou, minimální hodnotou, mediánem a směrodatnou odchylkou.

F2	**2D**	**5D**	**10D**	**50D**
mean	1,26E-03	2,89E-03	2,83E-03	5,27E-03
std	1,04E-02	1,24E-02	1,23E-02	1,49E-02
median	8,42E-09	3,08E-08	6,15E-08	1,10E-07
min	2,76E-11	1,66E-12	7,57E-12	1,11E-10

Tabulka 4.23: Hodnota kriteriální funkce na úloze F2 pro 2, 5, 10 a 50 dimenzí

Testování metody fminsearch

Metody byla otestována pro sto různých počátečních bodů z intervalu $x_0 = [-2\ 2]$. V následující tabulce 4.24 jsou zobrazeny výsledky testování.

F2	2D	5D	10D	50D
mean	5,29E-02	2,32E-01	2,61E-01	9,31E-01
std	1,29E-01	1,14E+00	1,00E+00	2,16E+00
median	3,73E-03	3,24E-03	2,32E-02	1,48E-01
min	1,77E-06	3,17E-06	1,91E-05	2,89E-04

Tabulka 4.24: Výsledky fminsearch testu

Srovnání výsledků

Z genetických algoritmů a metody simulovaného žíhání byly vybrány nejlepší hodnoty dosažené testováním. V případě genetických algoritmů bylo obtížné vybrat adekvátní nastavení, proto jsou pro srování uvedeny dvě různé varianty možného nastavení. Exaktní matematickou metodu zde reprezentuje algoritmus fminsearch dále v tabulkách označenou jako (N-M).
V tabulkách 4.25, 4.26, 4.27, 4.28 jsou uvedeny dosažené výsledky pro dimenze 2, 5, 10 a 50 jednotlivými metodami:

2D	GA	SA	PS	N-M
median	0	3,70E-05	8,42E-09	3,73E-03
min	0	8,15E-08	2,76E-11	1,77E-06

Tabulka 4.25: Srovnání jednotlivých metod pro 2 dimenze.

5D	GA2	SA	PS	N-M
median	1,19E-02	3,49E-04	3,08E-08	3,24E-03
min	9,62E-07	2,09E-08	1,66E-12	3,17E-06

Tabulka 4.26: Srovnání jednotlivých metod pro 5 dimenzí.

10D	GA	SA	PS	N-M
median	3,71E-01	6,24E-04	6,15E-08	2,32E-02
min	4,83E-05	4,97E-08	7,57E-12	1,91E-05

Tabulka 4.27: Srovnání jednotlivých metod pro 10 dimenzí.

50D	GA	SA	PS	N-M
median	4,90E01	8,80E-03	1,10E-07	1,48E-01
min	1,54E01	6,14E-08	1,11E-10	2,89E-04

Tabulka 4.28: Srovnání jednotlivých metod pro 50 dimenzí.

4.3 Problém obchodního cestujícího (Traveling salesman problem)

Problém obchodního cestujícího (TSP) je jednou z typicky řešených úloh matematické optimalizace. Je to kombinatorický problém, důležitý v operačním výzkumu a teoretické in-

formatice. Poprvé byl formulován v roce 1930 a stal se jedním z nejčastějších nástrojů pro určení kvality optimalizačních metod. Výpočtová náročnost se rapidně zvyšuje s rostoucím počtem měst. Tímto problém se mimo jiné také zabýval již zmíňený B. G. Dantzing [5].

Formulace TSP: Mámě množinu n měst, přičemž vzdálenost mezi jednotlivými městy je známá. Úkolem obchodníka je navštívit všechny města právě jednou a vrátit se do výchozího města po nejkratší možné cestě.
Problém obchodního cestujícího byl řešen pomocí genetických algoritmů a metody simulovaného žíhání. Rozložení měst bylo zvoleno na obvodu kruhu (o poloměru 100) kvůli ověření správnosti dosažených výsledků. Počet měst byl zvolen 20 a 100.

Řešení pomocí genetických algoritmů

Genetické algoritmy [7] řeší otpimalizační problémy reprezentované binárním řetězcem nebo datech formátu double. Jestliže je potřebný jiný formát dat, je možné vytvořit pole buněk požadovaného datového typu. V tomto případě je nutné nadefinovat vlastní funkce pro vytvoření populace, křížení a mutaci. Uživatelská funkce pro vytvoření populace sestaví pole buněk P velikosti populace, pričemž každý prvek pole P reprezentuje množinu měst jako permutační vektor. To znamená, že každý jedinec specifikuje pořadí měst, v jakém je bude obchodník projíždět. Funkce pro křížení vrátí pole buněk dětí, které jsou kombinací rodičů. Při mutaci dojde k přeuspořádání pořadí měst, funkce tedy vrátí zmutovaný permutační vektor. Nezbytná je také definice fitness funkce, která pro každého jedince spočítá délku trasy určenou jeho permutačním vektor. Fitness funkce musí mít k dispozici informaci o vzdálenosti jednotlivých měst.

TSP pro 20 měst

Při řešení problému obchodního cestujícího byl zkoumán vliv selekce a elitismu na celkovou délku trasy. Hodnota selekce byla testována z množiny $[4, 6, 8, 10]$ a hodnota elitismu z množiny $[4, 6, 8]$.

Níže je uvedeno nastavení genetických algoritmů:

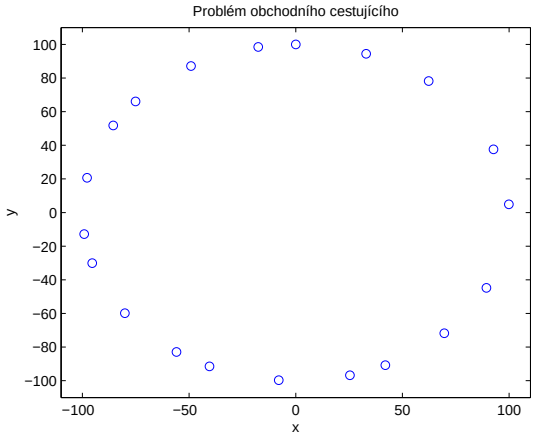
```
cities = 20;
PopInitRange = [1;cities]
PopulationSize = 100
SelectionFcn = selectiontournament
EliteCount = 4
Generations = 100
StallGenLimit = Inf
```

Vizualizace rozložení měst a optimalizačního procesu v 5, 25 a 100 generacích. Obrázky 4.6 až 4.9:

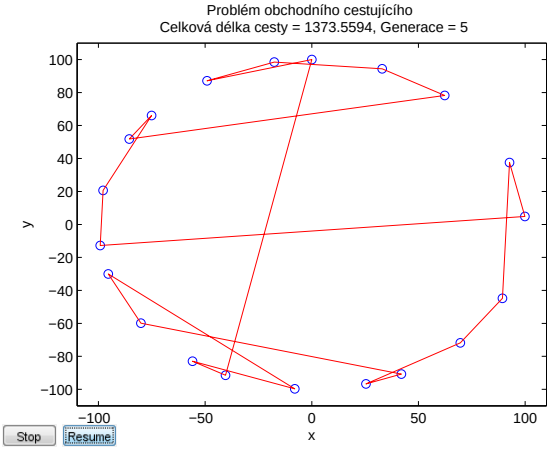
Úloha je natolik jednoduchá, že algoritmus našel správnou sestu při všech možnostech nastavení. V tabulce 4.29 jsou uvedeny výsledky fitness funkce reprezentované její průměrnou hodnotou, minimem, mediánem a směrodatnou odchylkou pro vybrané nastavení:

TSP pro 100 měst

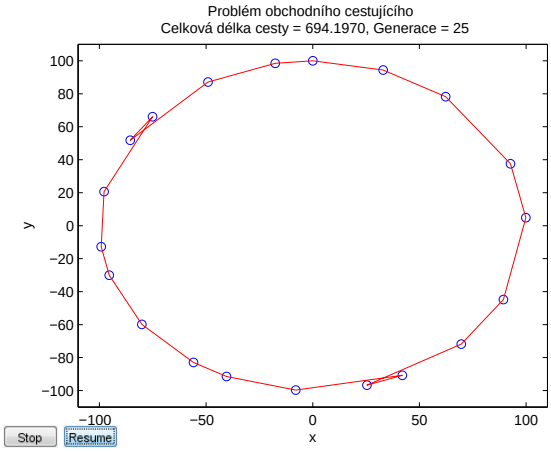
Při řešení problému obchodního cestujícího byl zkoumán vliv selekce a elitismu na celkovou délku cesty. Hodnota selekce a elitismu odpovídá předchozímu nastavení. Nalezení nejkratší



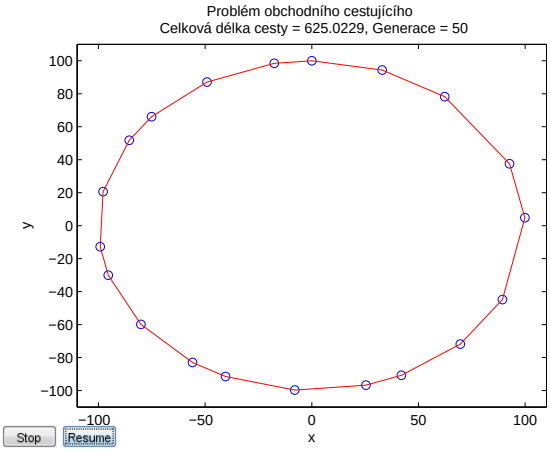
Obrázek 4.6: TSP20 - rozložení měst.



Obrázek 4.7: TSP20 - průběh optimalizace v 5ti generacích (GA).



Obrázek 4.8: TSP20 - průběh optimalizace v 25ti generacích (GA).



Obrázek 4.9: TSP20 - průběh optimalizace v 50ti generacích (GA).

selection	4	6	8	10
mean	6,25E+02	6,25E+02	6,25E+02	6,25E+02
min	6,25E+02	6,25E+02	6,25E+02	6,25E+02
median	6,25E+02	6,25E+02	6,25E+02	6,25E+02
std	1,02E-12	1,02E-12	1,02E-12	1,02E-12

Tabulka 4.29: Výsledky fitness funkce genetických algoritmů pro 20 měst.

možné cesty pro 100 měst je výrazně složitější, proto byla zvětšena hodnota počáteční populace a počet generací.

Níže je uvedeno nastavení genetických algoritmů:

```
cities = 100;
PopInitRange = [1;cities]
PopulationSize = 500
```

```
SelectionFcn = selectiontournament
EliteCount = 4
Generations = 1500
StallGenLimit = Inf
```

Vizualizace rozložení měst a optimalizačního procesu v 50, 150 a 1500 generacích:

V tabulkách 4.30, 4.31, 4.32 jsou uvedeny výsledky fitness funkce reprezentované její průměrnou hodnotou, minimem, mediánem a směrodatnou odchylkou pro různé kombinace selekce a elitismu.

TSP	**elitism 4**			
	6	**8**	**10**	**12**
mean	6,81E+02	6,35E+02	6,28E+02	6,28E+02
min	6,28E+02	6,28E+02	6,28E+02	6,28E+02
median	6,31E+02	6,28E+02	6,28E+02	6,28E+02
std	9,68E+01	1,41E+01	1,88E+00	1,34E+00

Tabulka 4.30: Výsledky fitness funkce pro genetické algoritmy s elitismem 4.

TSP	**elitism 6**			
	6	**8**	**10**	**12**
mean	6,74E+02	6,30E+02	6,28E+02	6,28E+02
min	6,28E+02	6,28E+02	6,28E+02	6,28E+02
median	6,28E+02	6,28E+02	6,28E+02	6,28E+02
std	9,53E+01	7,31E+00	1,34E+00	1,14E-12

Tabulka 4.31: Výsledky fitness funkce pro genetické algoritmy s elitismem 6.

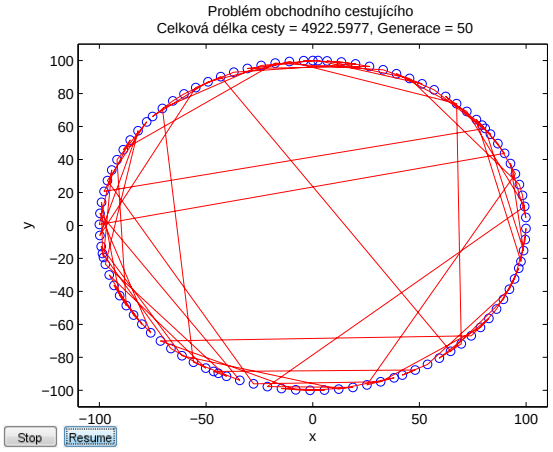
TSP	**elitism 8**			
	6	**8**	**10**	**12**
mean	6,52E+02	6,28E+02	6,28E+02	6,28E+02
min	6,28E+02	6,28E+02	6,28E+02	6,28E+02
median	6,28E+02	6,28E+02	6,28E+02	6,28E+02
std	6,17E+01	1,14E-12	1,14E-12	1,14E-12

Tabulka 4.32: Výsledky fitness funkce pro genetické algoritmy s elitismem 8.

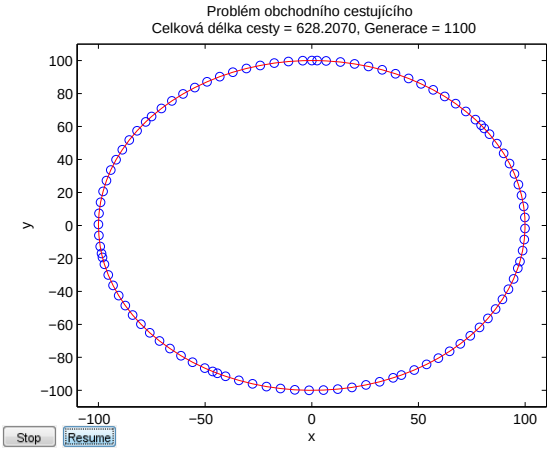
Vizualizace optimalizačního procesu v 50 a 1100 generacích. Obrázky 4.10 a 4.11.

Řešení pomocí metody simulovaného žíhání

Je dán počet měst, pričemž jejich pořadí udává posloupnost stavů, ve kterých se atom, tedy obchodní cestující nachází. Na počátku je náhodně dáno pořadí měst, které splňuje podmínku, že každé město může být navštíveno pouze jednou. V každé iteraci je pořadí stávajících měst náhodně přehozeno ve snaze nalézt nejkratší možnou cestu. Je tedy nutné znát vzdálenost mezi jednotlivými městy.



Obrázek 4.10: TSP100 - průběh optimalizace v 50ti generacích (GA).



Obrázek 4.11: TSP100 - průběh optimalizace v 1100ti generacích (GA).

TSP pro 20 měst

Problém byl řešen pro tři nastavení počáteční teploty a to pro $t = 50$, $t = 100$ a $t = 150$. Dále byl zvolen počet měst, jejichž pořadí bude prohozeno. Pro aktualizaci teploty v každé iteraci slouží `AnnealingParameter`, který byl zvolen 0.8.

Nastavení metody simulovaného žíhání:

```
cities = 20;
InitialTemperature = 50
CitiestoSwap=2
MaxIteration = 1000
AnnealingParameter = 0.8
```

V tabulce 4.33 jsou uvedeny výsledky testování (průměrná délka cesty, minimální délka cesty, medián a směrodatná odchylka):

TSP	**t = 50**		**t = 100**		**t = 150**	
	swap2	**swap4**	**swap2**	**swap4**	**swap2**	**swap4**
mean	7,23E+02	7,18E+02	7,12E+02	7,11E+02	6,99E+02	6,94E+02
min	6,25E+02	6,25E+02	6,25E+02	6,25E+02	6,25E+02	6,25E+02
median	6,25E+02	6,25E+02	6,25E+02	6,25E+02	6,25E+02	6,25E+02
std	1,80E+02	1,76E+02	1,74E+02	1,72E+02	1,63E+02	1,59E+02

Tabulka 4.33: Výsledky simulovaného žíhání pro 20 měst.

TSP pro 100 měst

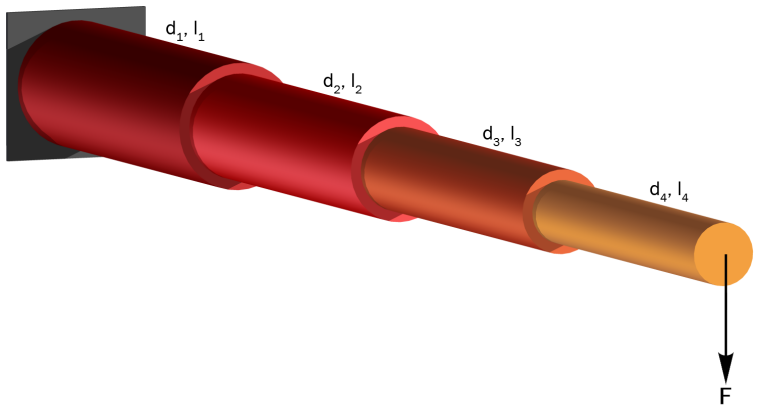
Nastavení metody simulovaného žíhání bylo stejné jako v případě TSP pro 20 měst až `MaxIteration`, v našem případě `MaxIteration=5000`. V tabulce 4.34 jsou uvedeny výsledky testování (průměrná délka cesty, minimální délka cesty, medián a směrodatná odchylka):

TSP	t = 50		t = 100		t = 150	
	swap2	swap4	swap2	swap4	swap2	swap4
mean	2,38E+03	2,52E+03	2,44E+03	2,46E+03	2,45E+03	2,46E+03
min	1,43E+03	1,73E+03	1,37E+03	1,52E+03	1,44E+03	1,57E+03
median	2,37E+03	2,55E+03	2,41E+03	2,51E+03	2,46E+03	2,47E+03
std	3,99E+02	4,16E+02	4,25E+02	4,76E+02	4,08E+02	4,16E+02

Tabulka 4.34: Výsledky simulovaného žíhání pro 100 měst

4.4 Konzola s proměnnou průřezovou charakteristikou

Tento problém zahrnuje konstrukci konzoly s volným koncem [20]. Konzola musí být především schopna unést předepsané zatížení. Úkolem je minimalizace objemu konzoly při různých konstrukčních omezeních. Konzola je složena z válců proměnné délky (l_i) a proměnného průměru (d_i) a je zatížena silou **F** na jejím volném konci. Na nákresu 4.12 je zobrazena modelová situace.



Obrázek 4.12: Modelová situace stupňovítého konzolového nosníku.

Kriteriální funkci zde reprezentuje objem nosníku složeného ze čtyř válců:

$$V = \frac{\pi}{4}(d_1^2 l_1 + d_2^2 l_2 + d_3^2 l_3 + d_4^2 l_4) \tag{4.3}$$

Napětí v bodě (x, y, z) ve středu průřezu jednotlivých částí je dán vztahem:

$$\sigma_i = \frac{M(x) y_i}{W_i} \tag{4.4}$$

Kde $M(x)$ je ohybový moment v bodě x, x je vzdálenost od volného konce a W modul průřezu v ohybu. V případě kruhové plochy je modul průřezu v ohybu roven vztahu:

$$\sigma_i = \frac{J(x)}{e} \tag{4.5}$$

Pro kruhový průřez platí:

$$J(x) = \frac{\pi d^4}{64}, e = \frac{d}{2}. \tag{4.6}$$

Ohybové napětí v každé části konzoly nesmí překročit maximální dovolené napětí $\sigma_{\max}$. V důsledku toho můžeme uvést čtyři podmínky pro napětí v ohybu:

$$\begin{aligned}
\frac{32Fl_4}{\pi d_4^3} &\leq \sigma_{\max} \\
\frac{32Fl_3}{\pi d_3^3} &\leq \sigma_{\max} \\
\frac{32Fl_2}{\pi d_2^3} &\leq \sigma_{\max} \\
\frac{32Fl_1}{\pi d_1^3} &\leq \sigma_{\max}
\end{aligned} \tag{4.7}$$

Průhyb volného konce nosníku δ lze spočítat podle Castiglianovy věty:

$$\delta = \frac{\partial U}{\partial F} \tag{4.8}$$

Kde U je deformační energie pro kterou platí vztah:

$$U = \int_0^L \frac{M^2}{2EI} dx \tag{4.9}$$

M je moment síly F v bodě x. Protože $M = Fx$, můžeme deformační energii psát ve tvaru:

$$\begin{aligned}
U = \quad & \tfrac{F^2}{2E} \Big[\textstyle\int_0^{l_4} \frac{x^2}{I_4} dx + \\
& \textstyle\int_0^{l_3} \frac{(x+l_4)^2}{I_3} dx + \\
& \textstyle\int_0^{l_2} \frac{(x+l_4+l_3)^2}{I_2} dx + \\
& \textstyle\int_0^{l_1} \frac{(x+l_4+l_3+l_2)^2}{I_1} dx \Big]
\end{aligned} \tag{4.10}$$

Kde I je kvadratický moment průřezu dané části. Po integraci dostáváme následující rovnici:

$$\begin{aligned}
U = \quad & \tfrac{F^2}{2E} \Big[\tfrac{(l_1+l_2+l_3+l_4)^3}{3I_1} - \tfrac{(l_2+l_3+l_4)^3}{3I_1} + \\
& \tfrac{(l_2+l_3+l_4)^3}{3I_2} - \tfrac{(l_3+l_4)^3}{3I_2} + \\
& \tfrac{(l_3+l_4)^3}{3I_3} - \tfrac{(l_4)^3}{3I_3} + \\
& \tfrac{(l_4)^3}{3I_4} \Big]
\end{aligned} \tag{4.11}$$

Aplikací Castiglianovy vět, průhyb volného konce bude:

$$\begin{aligned}
\delta = \quad & \tfrac{F}{2E} \Big[\tfrac{(l_1+l_2+l_3+l_4)^3}{3I_1} - \tfrac{(l_2+l_3+l_4)^3}{3I_1} + \\
& \tfrac{(l_2+l_3+l_4)^3}{3I_2} - \tfrac{(l_3+l_4)^3}{3I_2} + \\
& \tfrac{(l_3+l_4)^3}{3I_3} - \tfrac{(l_4)^3}{3I_3} + \\
& \tfrac{(l_4)^3}{3I_4} \Big]
\end{aligned} \tag{4.12}$$

Průhyb volného konce δ musí být menší než maximalní dovolený průhyb $\delta_{\max}$, potom další omezující podmínka bude:

$$\frac{F}{2E}\left(\frac{(l_1+l_2+l_3+l_4)^3}{3I_1}-\frac{(l_2+l_3+l_4)^3}{3I_1}+\frac{(l_2+l_3+l_4)^3}{3I_2}-\frac{(l_3+l_4)^3}{3I_2}+\frac{(l_3+l_4)^3}{3I_3}-\frac{(l_4)^3}{3I_3}+\frac{(l_4)^3}{3I_4}\right)\leq\delta_{\max} \tag{4.13}$$

Nyní je možné nalézt optimální parametry nosníku s příhlédnutím k jeho omezení. Položíme

$$x_1=d_1, x_2=d_2, x_3=d_3, x_4=d_4, x_5=l_1, x_6=l_2, x_7=l_3, x_8=l_4 \tag{4.14}$$

Proměnné dosadíme do rovnic 4.3, 4.7, ze kterých nám vyplyne kriteriální funkce a omezující podmínky.

Dále je nutné vyhovět požadavkům na rozměry válců, které přidávájí další omezení, které jsou popsány:

$$15\leq x_1\leq 30\wedge x_1\in Z \tag{4.15}$$
$$15\leq x_2\leq 19\wedge x_2\in Z \tag{4.16}$$
$$15\leq x_3\leq 19\wedge x_3\in Z \tag{4.17}$$
$$5\leq x_4\leq 15 \tag{4.18}$$
$$x_5, x_6, x_7, x_8\in[50, 70, 100, 120, 150] \tag{4.19}$$
$$x_5+x_6+x_7+x_8\leq 500 \tag{4.20}$$

Parametry problému:
Síla, kterou je nosník namáhán, $F=80000N$
Maximální dovolený průhyb volného konce, $\delta_{\max}=3cm$
Maximální dovolené napětí v jednotlivých členech, $\sigma_{\max}=14000N/cm^2$
Yongův modul, $E=2\times10^7N/cm^2$

Řešení pomocí genetických algoritmů

K vyřešení problému využijeme implementovanou funkci `ga`. Proměnné x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 jsou dané diskrétní množinou, proto se tento problém nazývá *Mixed Integer*. Aby bylo možné zadefinovat horní a dolní mez, vytvoříme funkci pro transformování diskrétních proměnných. Tyto proměnné přetransformujeme na celé číslo v rozsahu $[1,\ldots,5]$ (proměnné jsou z množiny pěti prvků). Dále vyvoříme m-skript obsahující fitness funkci `cantilever_volume.m` a m-skript omezujících podmínek `my_cantilever_constraints.m`. Horní a dolní mez je ve tvaru:

```
lb = [15 1 1 5 1 1 1 1] % dolní mez
ub = [30 5 5 15 5 5 5 5] % horní mez
```

Dále nastavíme `options` strukturu:

```
options = gaoptimset(...
          'PopulationSize', 300, ...
          'Generations', 500, ...
          'EliteCount', 8, ...
          'TolFun', 1e-10, ...
          'StallGenLimit', Inf);
```

Pro volání funkce ga specifikujeme vektor indexů proměnných, které mají diskrétní hodnoty:

```
options = gaoptimset(...
[best_size, volume] = ga(@cantilever_volume, ...
    8, [], [ ], [], [], lb, ub, @my_cantilever_constraints, ...
    [1 2 3 5 6 7 8 ], options);
```

Funkce byla spuštěna dvacetkrát a z výsledku byly vybrány minimální rozměry nosníku při daném maximálním napětí a maximálním ohybu.
Výsledné rozměry konzoly jsou $d_1 = 24$, $l_1 = 70$, $d_2 = 19$, $l_2 = 50$, $d_3 = 18$, $l_3 = 50$, $d_4 = 14.42$, $l_4 = 50$.

Závěr

Předložená bakalářská práce představila vybrané optimalizační nástroje dostupné v systému MATLAB. Stručná rešerše dvou hlavních optimalizačních knihoven označených jako Optimization Toolbox (OTbx) a Global Optimization Toolbox (GOTbx) poukázala na možnosti tohoto výpočetního prostředí. V tomto ohledu mohou příslušné kapitoly sloužit jako rychlé referenční příručky pro orientaci v implementovaných optimalizačních metodách.

Práce se dále zaměřila na detailnější popis optimalizačních *Soft Computing* nástrojů implementovaných v knihovně GOTbx. Právě tato knihovna je využitelná pro řešení obtížných nelineárních či NP úloh. Představeny byly vlastní metody, jejich základní syntaxe a parametrizace a uvedeny příklady jejich využití. Na sadě testovacích úloh byla prezentována metodika vhodné parametrizace genetických algoritmů (GA) a simulovaného žíhání (SA). Testovací úlohy F2 a F6 reprezentovaly známé úlohy využívané právě k otestování výkonu optimalizačních algoritmů. K dále užitým úlohám patřila v tomto případě uměle modifikovaná úloha obchodního cestujícího, a konečně praktická úloha z oblasti celočíselného smíšeného programování, která modelově řešila optimální návrh parametrů vetknutého nosníku. Testovací úlohy byly řešeny již zmíněnými algoritmy GA a SA, a dále v případě úloh F6 a F2 metodou petern search (PS) a komparativně exaktním Nelder-Mead (NM) algoritmem. Simulační experimenty byly vyhodnocovány jednak ve vlastním prostředí MATLAB, tak bylo využito sofistikovaného statistického software Minitab. Získané výsledky potvrdili schopnost GA nalézat globální extrémy i na rozsáhlých multimodálních úlohách (F6), v případe SA se naopak ukázaly jisté meze této metody ve vztahu k množství lokálních extrémů. Experimenty dále ukázaly vliv některých parametrů GA a SA, což potvrdilo vhodnost optimalizace těchto parametrů ve vztahu k dané úloze.

Vybrané a experimentálně podložené závěry jsou následující:

- V případě genetických algoritmů je vliv síly selekce na konvergenční vlastnosti algoritmu signifikantní. U multimodálních úloh vede zvýšený selekční tlak k předčasné konfergenci. U jednoextremálních úloh je pochopitelně nasazení GA závislé na dalších "patologiích"účelové funkce, nic méně je efektivně možné.

- Velikost populace v diskutovaném rozsahu 100 až 500 jedinců ovlivňuje kvalitu nalezených řešení podstatně méně než počet generací, které má algoritmus k dispozici za účelem dosažení optimálního řešení.

- Velikost výběru elitních jedinců je dostatečná do 5% velikosti populace. Pochopitelně jsou mnohé parametry závislé, tedy je toto konstatování vztaženo ověřovanému selekčnímu tlaku generovanému turnajovým výběrem o maximu 8% jedinců.

- Technika SA je efektivní u jednoextremálních úloh s omezením, u více extrémů je efektivní pouze pro menší počet optimalizovaných parametrů (testováno 2, 5 a 10 parametrů). Z principu je algoritmus SA rychlejší než ekvivalentní GA.

- U algoritmu SA byly testovány všechny dostupné varianty parametrizace metod žíhání a změny teploty. Jako optimální se ukázalo využití kombinace rychlého žíhání (nazev) a exponenciální změny teploty (nazev).
- Algoritmy PS potvrdily předpoklady a byly dobrým konkurentem metod GA v multimodální úloze F6. Rovněž komparativní porovnání s exaktním algoritmem Nelder-Mead (NM) bylo vhodným doplněním pro představu možností standardních algoritmů na daných typech úloh.
- Na obtížné multimodální úloze F6 neměl GA konkurenci a potvrdil adekvátnost užití evolučních metod v tomto ohledu. Algoritmy PS a NM podaly přibližně obdobný výkon. Algoritmus SA dopadl nejhůře.
- Na úloze F2, byly nejefektivnější algoritmy PS a NM, v tesném sousledu úspěšnosti následoval algoritmus SA. GA v tomto ohledu našel dostečně dobré řešení, přesto se umístil na posledním místě. Tento výsledek je třeba vztáhnout k užité metodě mutace a užitému kódování, nikoliv ho zevšeobecňovat. Přesto je v ohledu doporučení metaheuristik na daný typ problému, využít jako algoritmus první volby PS nebo SA.

Závěrem je vhodné poznamenat, že doména zadání by mohla indukovat značně rozsáhlou práci s vědeckým potenciálem analýzy chování vybraných algoritmů, a to vzhledem k jejich parametrizaci, rozsahu a povaze optimalizační úlohy. Cílem práce bylo stručným rešeršním způsobem pojednat o diskutovaných metodách, ukázat některé jejich vlastnosti a možnosti užití. V tomto ohledu je modelový příklad výpočtu optimálních parametrů nosníku vhodnou závěrečnou úlohou, která naznačuje, kde mohou být diskutované algoritmy využity. Je třeba připomenout, že právě v inženýrské praxi je velmi zásadní obdržet dostatečně přijatelná řešení v přijatelném čase a není mnohdy možné čekat na optimální řešení v matematickém kontextu. Z tohoto důvodů jsem i vzhledem k svému studiu na technické škole ráda, že jsem se mohla daným metodám věnovat, a pochopila tak alespoň z části základní vlastnosti metod, které v podstatě vyvinula a v podstatě ještě vyvíjí sama příroda.

Literatura

[1] Audet, C.; Dennis, J.: Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization*, ročník 17, č. 1, 2006: s. 188–217, doi:10.1137/040603371, <http://epubs.siam.org/doi/pdf/10.1137/040603371>. URL <http://epubs.siam.org/doi/abs/10.1137/040603371>

[2] Audet, C.; Dennis, J. E.: Analysis of generalized pattern searches. *SIAM Journal on Optimization*, ročník 13, č. 3, 2002: s. 889–903.

[3] Bertsekas, D. P.: *Nonlinear Programming: 2nd Edition*. Cambridge: Athena Scientific, 1999, ISBN 1-886529-00-0.

[4] Björck, A.: *Numerical Methods for Least Squares Problems*. Philadelphia: SIAM, 1996, ISBN 1-886529-00-0.

[5] Dantzig, G. B.: *Linear Programming and Extensions*. Princeton, New Jersey: Princeton University Press, 1998, ISBN 0-691-08000-3, 633 s.

[6] Doc. RNDr. Jindřich Klapka, C.; RNDr. Jiří Dvořák, C.; RNDr. Pavel Popela, P.: *Metody operačního výzkumu*. Brno: VUTIUM, 2001, ISBN 80-214-1839-7.

[7] Goldberg, D.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence, Addison-Wesley, 1989, ISBN 9780201157673. URL <http://books.google.cz/books?id=3_RQAAAAMAAJ>

[8] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P.: Optimization by simulated annealing. *SCIENCE*, ročník 220, č. 4598, 1983: s. 671–680.

[9] Lagarias, J. C.; Reeds, J. A.; Wright, M. H.; aj.: Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM Journal of Optimization*, ročník 9, 1998: s. 112–147.

[10] Lewis, R.; Shepherd, A.; Torczon, V.: Implementing Generating Set Search Methods for Linearly Constrained Minimization. *SIAM Journal on Scientific Computing*, ročník 29, č. 6, 2007: s. 2507–2530, doi:10.1137/050635432, <http://epubs.siam.org/doi/pdf/10.1137/050635432>. URL <http://epubs.siam.org/doi/abs/10.1137/050635432>

[11] Marquardt, D. W.: An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, ročník 11, č. 2, 1963: s. 431–441.

[12] Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.: Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, ročník 21, 1953.

[13] Nelder, J. A.; Mead, R.: A Simplex Method for Function Minimization. *Computer Journal*, ročník 7, č. 4, 1965: s. 308–313.

[14] Nocedal, J.; Wright, S. J.: *Numerical Optimization (2nd ed.)*. Berlin: New York: Springer-Verlag, 2006, ISBN 978-0-387-30303.

[15] Oplatková, Z.; Ošmera, P.; Šeda, M.; aj.: *Evoluční výpočetní techniky – principy a aplikace*. Praha: BEN-Technická literatura, 2008, ISBN 80-7300-218-3.

[16] Powell, M. J. D.: Direct Search Algorithms for Optimization Calculations. 1998.

[17] Rosenbrock, H. H.: An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, ročník 3, č. 3, 1960: s. 175–184.
URL `<http://dx.doi.org/10.1093/comjnl/3.3.175>`

[18] Sorensen, D. C.: Newton's Method with a Model Trust Region Modification. *SIAM Journal on Numerical Analysis*, ročník 19, č. 2, 1982: s. 409–426.
URL `<http://www.maths.tcd.ie/~mnl/store/Sorensen1982a.pdf>`

[19] Spendley, W.; Hext, G. R.; Himsworth, F. R.: Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation. *Technometrics*, ročník 4, č. 4, 1962: s. 441–461.
URL `<http://www.ams.org/mathscinet-getitem?mr=32:1849>`

[20] Thanedar, P.; Vanderplaats, G.: Survey of Discrete Variable Optimization for Structural Design. *Journal of Structural Engineering*, ročník 121, č. 2, 1995: s. 301–306.

[21] Torczon, V.: On the Convergence of Pattern Search Algorithms. *SIAM Journal on Optimization*, ročník 7, č. 1, 1997: s. 1–25.
URL `<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.4715&rep=rep1&type=pdf>`

[22] Urgay, Z.; Lasdon, L.; Plummer, J.; aj.: Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *INFORMS Journal on Computing*, ročník 19, č. 3, 2007: s. 328–340.

[23] Waltz, R.; Morales, J.; Nocedal, J.; aj.: An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, ročník 107, č. 3, 2006: s. 391–408, ISSN 0025-5610, doi:10.1007/s10107-004-0560-5.
URL `<http://dx.doi.org/10.1007/s10107-004-0560-5>`