

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

**Metódy sedlového bodu na riešenie
úloh konvexného programovania**

DIPLOMOVÁ PRÁCA

BRATISLAVA 2012

BC. TOMÁŠ ROSENBERG

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA APLIKOVANEJ MATEMATIKY A ŠTATISTIKY



Metódy sedlového bodu na riešenie úloh konvexného programovania

DIPLOMOVÁ PRÁCA

Študijný program : Ekonomická a finančná matematika

Študijný odbor : 1114 Aplikovaná matematika

Skoliteľ : doc. RNDr. Milan Hamala, CSc.

BRATISLAVA 2012

BC. TOMÁŠ ROSENBERG



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Rosenberg
Študijný program: ekonomická a finančná matematika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.1.9. aplikovaná matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Metódy sedlového bodu na riešenie úloh konvexného programovania

Cieľ: Metódy sedlového bodu na riešenie úloh konvexného programovania

Vedúci: doc. RNDr. Milan Hamala, CSc.

Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky

Dátum zadania: 13.01.2011

Dátum schválenia: 14.01.2011

prof. RNDr. Daniel Ševčovič, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Prehlásenie

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry a s odbornou pomocou školiteľa.

Bratislava, 25. apríla 2012

.....
Bc. Tomáš Rosenberg

PodĎakovanie

Na tomto mieste by som sa chcel srdečne poĎakovať vedúcemu diplomovej práce doc. RNDr. Milanovi Hamalovi, CSc., za obrovské množstvo času, ktoré mi venoval a za jeho cenné rady, nápady, pripomienky a usmernenia, ktoré mi veľmi pomohli pri písaní tejto práce. Ďakujem taktiež svojej rodine a priateľom za trpezlivosť a podporu.

Abstrakt

ROSENBERG, Tomáš, Bc., Metódy sedlového bodu na riešenie úloh konvexného programovania

Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky.

Vedúci diplomovej práce: doc. RNDr. Milan Hamala, CSc.

Diplomová práca, 2012.

V práci sa venujeme najmä metódam rozšírených Lagrangeových funkcií, ktoré sú jedným z nástrojov na riešenie úloh nelineárneho programovania. Ich podstatou je nahradiť riešenie danej úlohy postupnosťou úloh na voľnú optimalizáciu. Okrem klasickej metódy Hestenesa, Powella a Rockafellara bolo navrhnutých množstvo alternatívnych metód, vrátane techniky nelineárneho škálovania. V posledných dvadsiatich rokoch boli navyše navrhnuté postupy, ktoré v záverečnej fáze algoritmu nahradia minimalizáciu rozšírenej Lagrangeovej funkcie riešením špeciálnej sústavy rovníc. Výsledkom je asymptoticky vyššia rýchlosť konvergencie. Okrem študovania týchto postupov sa pokúsime aplikovať novú metódu založenú na komplementárnych funkciách. Naším hlavným cieľom je implementácia uvedených algoritmov a vykonanie numerických experimentov, na ktoré použijeme náhodne generované konvexné úlohy kvadratického programovania a vybranú množinu úloh z kolekcie CUTEr.

Kľúčové slová: nelineárne programovanie, rozšírené Lagrangeove funkcie, nelineárne škálovanie, primárno-duálny systém, Fischerova funkcia, numerické experimenty.

Abstract

ROSENBERG, Tomáš, Bc., Saddle point methods for solving convex optimization problems

Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics.

Supervisor: doc. RNDr. Milan Hamala, CSc.

Master Thesis, 2012.

In this master thesis we study in particular the augmented Lagrangian methods, which represent one of the well-known approaches to solving constrained optimization problems. Their purpose is to replace the solving of a given problem by sequence of unconstrained optimization subproblems. Aside to the classic Powell-Hestenes-Rockafellar algorithm many other methods of this kind were proposed, notably of the nonlinear-rescalling framework. Also in the last two decades some special procedures were suggested, which replaced the unconstrained optimization by solving particular system of equations in the last phase of the main algorithm. This results in an asymptotically faster rate of convergence. In addition to studying these methods, we try to apply a new method based on complementarity-problem functions. Our major goal is to implement aforementioned algorithms and to perform numerical experiments on randomly generated convex quadratic programming problems and selected problems from the CUTEr collection.

Keywords: nonlinear programming, augmented Lagrangian functions, nonlinear-rescalling, primal-dual system, Fischer function, numerical experiments.

Obsah

Úvod	1
Zoznam hlavných symbolov	3
1 Úvod do matematického programovania	4
1.1 Úloha s ohraničeniami v tvare rovníc	5
1.2 Úloha s ohraničeniami v tvare nerovníc	7
1.3 Úloha konvexného programovania a teória duality	11
1.3.1 Postačujúce podmienky optimality	12
1.3.2 Duálna úloha a teória duality	13
2 Metódy rozšírených Lagrangeových funkcií	16
2.1 Metóda Hestenesa a Powella pre úlohu s ohraničeniami v tvare rovníc	18
2.1.1 Iterácie multiplikátorov	23
2.2 Metóda Rockafellara pre úlohu s ohraničeniami v tvare nerovníc . . .	27
2.2.1 Odvodenie odhadu druhého rádu	31
2.3 Všeobecná penalizačná funkcia	34
2.3.1 Penalizačné funkcie pre úlohu (\mathcal{PR})	35
2.3.2 Penalizačné funkcie pre úlohu (\mathcal{PNR})	36
3 Nelineárne škálovanie a Fischerova metóda	41
3.1 Metóda nelineárneho škálovania	42
3.2 Primárno-duálna metóda Polyaka a Grivu	45
3.3 Vylepšená primárno-duálna schéma	49
3.4 Fischerova schéma	52

4	Numerické experimenty	57
4.1	Generátor úloh	57
4.2	Implementácie algoritmov	60
4.2.1	Rockafellarov algoritmus	60
4.2.2	Nelineárne škálovanie	62
4.2.3	Algorimus PDNRD Polyaka a Grivu	63
4.2.4	Fischerove algoritmy	64
4.3	Výsledky experimentov	65
4.3.1	Ilustračný príklad	67
4.3.2	Test č. 1: Vplyv rozmerov úlohy	67
4.3.3	Test č. 2: Zmena počtu ohraničení	69
4.3.4	Test č. 3: Zmena počtu aktívnych ohraničení	70
4.3.5	Test č. 4: Zmena počtu kvadratických ohraničení	71
4.3.6	Test č. 5: Zmena čísla podmienenosti	73
4.3.7	Zbierka CUTEr	74
	Záver	75
	Literatúra	77
	Príloha	81
A.	Newtonova minimalizačná metóda a vyhľadávanie na lúči	81
B.	Výsledky testovania príkladov CUTEr	84
C.	Zdrojový kód	95

Úvod

Požiadavka optimalizácie je stále častejšie sa vyskytujúcim problémom v technickej a vedeckej praxi. Veľký úspech lineárneho programovania v polovici minulého storočia bol podnetom pre formuláciu a štúdium zložitejších modelov. Tento fakt bol podporený zisteniami z praxe, kde sa ukázalo, že lineárne modely niekedy nedokážu dostatočne dobre zachytiť podstatu problému a boli tak používané skôr ako východisko z núdze. Výsledkom bol vznik nelineárneho programovania, ktoré dnes nachádza široké praktické uplatnenie (napr. modelovanie chemických a fyzikálnych javov, optimálny výber portfólia, úlohy plánovania a optimálneho riadenia).

Jednou zo základných metód na riešenie nelineárnych optimalizačných úloh sú metódy rozšírených Lagrangeových funkcií, ktorých podstatou je nahradenie riešenia zložitej optimalizačnej úlohy postupnosťou riešení jednoduchších minimalizačných úloh s cieľom nájdania sedlového bodu týchto funkcií. Spomínané metódy sú nástupcami penalizačných metód a odstraňujú niektoré ich nedostatky zavedením vektora Lagrangeových multiplikátorov. Ich vznik sa datuje do roku 1969, kedy bola Powellom [35] a nezávisle Hestenesom [22] uverejnená prvá metóda tohto druhu pre úlohy s ohraničeniami v tvare rovníc. Rockafellar [40] neskôr danú metódu rozšíril aj pre ohraničenia v tvare nerovnic a postupom času vzniklo mnoho rôznych tvarov rozšírených Lagrangeových funkcií [4]. Veľký teoretický aj praktický záujem o tieto metódy bol badateľný najmä v 70-tych a 80-tych rokoch 20. storočia, kde medzi popredných autorov tejto oblasti patril najmä Bertsekas [1]. V nasledujúcich rokoch sa pozornosť mierne vytratila a upriamila sa na metódy vnútorného bodu.

O čiastočný návrat rozšírených Lagrangeových funkcií sa postarali Polyak a Griva, ktorí v sérii článkov [30], [32], [33], [34] rozpracovali základy tzv. primárno-duálnej nelineárne škálovacej metódy. Tento prístup je založený na použití metódy špeciálnej rozšírenej Lagrangeovej funkcie a modifikácii jej záverečnej výpočtovej fázy. Polyak a Griva navrhli nahradiť minimalizáciu Lagrangeovej funkcie riešením tzv. primárno-

duálneho systému, čoho výsledkom je väčšia rýchlosť konverencie. Do pozornosti sa dostáva aj alternatívna metodika, založená na transformácii KKT systému Fischerovou transformačnou funkciou [14].

Cieľom práce je oboznámiť sa s metódami sedlového bodu a ich algoritmickou realizáciou, implementovať tieto algoritmy a vykonať numerický experiment s cieľom ich vzájomného porovnania. Tieto ciele sú realizované v štyroch hlavných kapitolách.

V prvej kapitole uvádzame základné poznatky z teórie nelineárneho programovania.

V druhej kapitole sa venujeme teoretickému rozboru metód rozšírených Lagrangeových funkcií. Spomíname Powellovu-Hestenesovu metódu, z ktorej postupne odvodíme Rockafellarovu metódu a takisto uvedieme všeobecné triedy rozšírených Lagrangeových funkcií.

V tretej kapitole nadviažeme na výsledky druhej kapitoly a uvedieme primárnoduálnu nelineárne škálovaciu metódu Polyaka a Grivu a takisto Fischerovu metódu.

Štvrtú kapitolu venujeme numerickým experimentom. Spomenieme schematické implementácie testovaných algoritmov, spôsob generovania náhodných úloh konvexného bikvadratického programovania. Numerické testy vykonáme na spomínaných náhodne generovaných úlohách a takisto na vybraných úlohách zo štandardného testovacieho balíka CUTEr.

Nasleduje zoznam použitej literatúry, výsledky experimentov na CUTEr príkladoch a časť zdrojového kódu pre prostredie MATLAB. Úplný zdrojový kód je obsiahnutý na priloženom médiu.

Zoznam hlavných symbolov

$\mathbf{A}, \mathbf{C}, \mathbf{D}$	matice
\mathbf{I}, \mathbf{I}_n	jednotková matica (dimenzie $n \times n$)
$\mathbf{x}, \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u}, \mathbf{v}, \boldsymbol{\xi}, \boldsymbol{\mu}$	vektory (stĺpcové)
$\mathbf{A}^T, \mathbf{x}^T$	transponovaná matica k matici \mathbf{A} , transponovaný vektor k vektoru \mathbf{x} (výsledkom je riadkový vektor)
$\mathbf{x}^T \mathbf{y}$	skalárny súčin vektorov \mathbf{x}, \mathbf{y}
$\mathbf{y} \geq 0$	vektor, ktorého všetky zložky sú väčšie alebo rovné nule
$\mathbb{S}, \mathbb{M}, \mathbb{X}, \mathbb{Y}$	množiny
\mathbb{R}	množina reálnych čísel
$\mathbb{R}_n, \mathbb{R}_n^+, \mathbb{R}_n^{++}$	n -rozmerný Euklidovský priestor, jeho nezáporný, resp. kladný ortant
$f(\mathbf{x})$	reálna funkcia n -premenných $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$
$\nabla f(\mathbf{x})$	gradient funkcie $f(\mathbf{x})$, t.j. stĺpcový vektor, ktorého zložky sú prvé parciálne derivácie funkcie $f(\mathbf{x})$, čiže $(\nabla f(\mathbf{x}))_i = \frac{\partial f(\mathbf{x})}{\partial x_i}$
$\nabla^2 f(\mathbf{x})$	Hessova matica funkcie $f(\mathbf{x})$, t.j. matica druhých parciálnych derivácií, $(\nabla^2 f(\mathbf{x}))_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$
$F(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))^T$	vektorová funkcia premennej \mathbf{x} , $F : \mathbb{X} \rightarrow \mathbb{R}^m$
$\nabla F(\mathbf{x})$	Jakobiho matica funkcie $F(\mathbf{x})$, t.j. matica, ktorej riadkami sú gradienty funkcií $f_i(\mathbf{x})$, pre i -ty riadok $\nabla F(\mathbf{x})$ platí $(\nabla F(\mathbf{x}))_i = \nabla f_i(\mathbf{x})^T$
$\mathcal{L}(\mathbf{x}, \mathbf{u}, \mathbf{v}), \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{z})$	označenie pre klasickú, resp. rozšírenú Lagrangeovu funkciu
$\ \mathbf{x}\ $	norma vektora \mathbf{x} , najčastejšie $\ \mathbf{x}\ _2 = \sqrt{\mathbf{x}^T \mathbf{x}}$
$\text{diag}(\mathbf{x})$	diagonálna matica, ktorej diagonálu tvoria prvky vektora \mathbf{x}
$\text{rank}(\mathbf{A})$	hodnosť matice \mathbf{A} , t.j. počet jej lineárne nezávislých riadkov (stĺpcov)
$(\mathcal{PR}), (\mathcal{PNR}), \dots$	označenia úloh matematického programovania

Kapitola 1

Úvod do matematického programovania

Pod ľubovoľným problémom optimalizácie rozumieme výber „najlepšieho“ riešenia spomedzi všetkých dostupných riešení. Na určenie kvality týchto riešení používame *účelovú* funkciu, ktorú v závislosti od filozofie a podstaty problému minimalizujeme alebo maximalizujeme. Takýto problém môžeme zapísať ako

$$\begin{aligned} &\text{minimalizovať} && f(\mathbf{x}) \\ &\text{za podmienok} && \mathbf{x} \in \mathbb{M}, \end{aligned} \tag{1.1}$$

kde $f : \mathbb{X} \rightarrow \mathbb{R}$ je už spomínaná *účelová* funkcia, a množinu $\mathbb{M} \subset \mathbb{X}$ nazývame množinou *prípustných* riešení. Keďže maximalizácia funkcie f je ekvivalentná minimalizácii $-f$, na základe konvencie sa preto obmedzíme na úlohy formulované v uvedenom tvare. Vektor $\hat{\mathbf{x}}$ nazveme *lokálnym* minimom (*lokálnym* riešením) úlohy (1.1), ak existuje $\delta > 0$ také, že

$$f(\hat{\mathbf{x}}) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{M}, \|\hat{\mathbf{x}} - \mathbf{x}\| < \delta, \tag{1.2}$$

ostrým lokálnym minimom (*ostrým lokálnym* riešením), ak v (1.2) platí ostrá nerovnosť, a nakoniec *globálnym* minimom (*globálnym* riešením), ak

$$f(\hat{\mathbf{x}}) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{M}.$$

V prípade, že $\mathbb{M} = \mathbb{R}^n$, dostávame úlohu *voľnej* minimalizácie. Naším záujmom je však zaoberať sa problémami, kde $\mathbb{M} \subset \mathbb{R}^n$.

V celej práci budeme predpokladať nasledovné:

Predpoklad 1.1. *Množina optimálnych riešení*

$$\widehat{\mathbb{M}} = \{\hat{\mathbf{x}} \mid f(\hat{\mathbf{x}}) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathbb{M}\}$$

je neprázdna a ohraničená.

Z matematického hľadiska je zaujímavé popísať \mathbb{M} ako množinu všetkých bodov $\mathbf{x} \in \mathbb{X}$, ktoré spĺňajú nejakú sústavu rovníc a nerovnic, čiže

$$\mathbb{M} = \left\{ \mathbf{x} \in \mathbb{X} \mid \begin{array}{l} g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, m \\ h_j(\mathbf{x}) = 0, \quad j = 1, \dots, p \end{array} \right\},$$

pričom ohraničenia g_i, h_j sú reálne funkcie, $g_i : \mathbb{X} \rightarrow \mathbb{R}$ ($i = 1, \dots, m$), $h_j : \mathbb{X} \rightarrow \mathbb{R}$ ($j = 1, \dots, p$), a navyše o týchto funkciách, a takisto o funkcii f predpokladáme, že sú spojito diferencovateľné. Kvôli zjednodušeniu zápisov a úvah predpokladajme, že $\mathbb{X} = \mathbb{R}^n$. Dostávame sa tak ku klasickej formulácii úlohy *matematického programovania*.

V ďalšom texte sa budeme venovať variáciám tejto úlohy a sformulujeme niektoré základné tvrdenia o ich optimálnom riešení. Začneme úlohou, ktorej množina prípustných riešení \mathbb{M} je tvorená iba rovnicami (t.j. $m = 0$), a na základe výsledkov odvodených pre túto úlohu vyslovíme ich ekvivalenty aj pre úlohu s ohraničeniami v tvare nerovnic ($p = 0$).

1.1 Úloha s ohraničeniami v tvare rovníc

Uvažujme úlohu matematického programovania v tvare

$$\begin{array}{ll} \text{minimalizovať} & f(\mathbf{x}) \\ \text{za podmienok} & h_j(\mathbf{x}) = 0, \quad j = 1, \dots, p. \end{array} \quad (\mathcal{PR})$$

Nech $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ ($j = 1, \dots, p$), a takisto $f \in C^1, h_j \in C^1$ pre všetky j . Pre zjednodušenie niektorých zápisov označme

$$h(\mathbf{x}) = (h_1(\mathbf{x}), \dots, h_p(\mathbf{x}))^T,$$

potom $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$, a predpokladajme, že $p \leq n$. Pripomeňme, že symbolom $\nabla h(\mathbf{x})$ označujeme Jakobiho maticu funkcie h .

Množina prípustných riešení \mathbb{M} úlohy (\mathcal{PR}) má tvar

$$\mathbb{M} = \{\mathbf{x} \mid h(\mathbf{x}) = 0\}.$$

Pred vyslovením základného tvrdenia (Lagrange, 1760) o nutných podmienkach optimálneho riešenia úlohy (\mathcal{PR}) uvedieme definíciu *regulárneho* bodu, ktorú budeme potrebovať pri jej vyslovení.

Definícia 1.1. Bod $\hat{\mathbf{x}} \in \mathbb{M}$ nazývame *regulárnym*, ak sú gradienty ohraničení $\nabla h_1(\hat{\mathbf{x}}), \dots, \nabla h_p(\hat{\mathbf{x}})$ lineárne nezávislé, t. j.

$$\text{rank}(\nabla h(\hat{\mathbf{x}})) = p.$$

Teraz môžeme pristúpiť k vysloveniu tvrdenia.

Veta 1.1. Nech $\hat{\mathbf{x}}$ je lokálnym riešením úlohy (\mathcal{PR}) a takisto regulárnym bodom. Potom existuje (jednoznačne určený) vektor Lagrangeových multiplikátorov $\hat{\mathbf{v}} = (\hat{v}_1, \dots, \hat{v}_p)^T$ taký, že platí

$$\nabla f(\hat{\mathbf{x}}) + \sum_{j=1}^p \hat{v}_j \nabla h_j(\hat{\mathbf{x}}) = 0, \quad (1.3)$$

$$h(\hat{\mathbf{x}}) = 0. \quad (1.4)$$

Ak navyše $f \in C^2$, $h \in C^2$, potom platí

$$\boldsymbol{\omega}^T \left[\nabla^2 f(\hat{\mathbf{x}}) + \sum_{j=1}^p \hat{v}_j \nabla^2 h_j(\hat{\mathbf{x}}) \right] \boldsymbol{\omega} \geq 0 \quad (1.5)$$

pre všetky $\boldsymbol{\omega}$ také, že $\nabla h(\hat{\mathbf{x}})\boldsymbol{\omega} = 0$.

DÔKAZ: Je možné nájsť v [3], str. 281-283, prípadne [24], str. 327. □

Sústavu $n + p$ rovníc (1.3), (1.4) zvyčajne zapisujeme pomocou klasickej *Lagrangeovej funkcie*, ktorú definujeme predpisom

$$L(\mathbf{x}, \mathbf{v}) = f(\mathbf{x}) + \sum_{j=1}^p v_j h_j(\mathbf{x}) = f(\mathbf{x}) + h(\mathbf{x})^T \mathbf{v}, \quad (1.6)$$

kde $L : \mathbb{R}^n \times \mathbb{R}^p \rightarrow \mathbb{R}$. Potom pre nutné podmienky prvého rádu (1.3), (1.4) dostávame

$$\nabla_x L(\hat{\mathbf{x}}, \hat{\mathbf{v}}) = 0,$$

$$\nabla_v L(\hat{\mathbf{x}}, \hat{\mathbf{v}}) = 0,$$

čiže bod $(\hat{\mathbf{x}}, \hat{\mathbf{v}})$ je kritickým bodom Lagrangeovej funkcie na $\mathbb{R}^n \times \mathbb{R}^p$.

Ukazuje sa, že tvrdenia (1.3) - (1.5) vety 1.1 môžeme využiť pri formulácii postačujúcich podmienok pre $\hat{\mathbf{x}}$ takto: ak $\hat{\mathbf{x}}$ a $\hat{\mathbf{v}}$ spĺňajú sústavu (1.3), (1.4) a v podmienke (1.5) platí pre všetky $\boldsymbol{\omega} \neq 0$ ostrá nerovnosť, potom $\hat{\mathbf{x}}$ je ostrým (lokálnym) optimálnym riešením úlohy (\mathcal{PR}) . Navyše nie je nutné predpokladať regularitu bodu $\hat{\mathbf{x}}$.

Dôležitou vlastnosťou bodu $(\hat{\mathbf{x}}, \hat{\mathbf{v}})$ vo vzťahu k Lagrangeovej funkcii (1.6) je jeho sedlovosť, ako vyplýva z nasledujúcej vety (predpoklad $f \in C^1, h \in C^1$ nie je potrebný).

Veta 1.2. *Nech bod $(\hat{\mathbf{x}}, \hat{\mathbf{v}}) \in \mathbb{R}^n \times \mathbb{R}^p$ spĺňa*

$$L(\hat{\mathbf{x}}, \mathbf{v}) \leq L(\hat{\mathbf{x}}, \hat{\mathbf{v}}) \leq L(\mathbf{x}, \hat{\mathbf{v}}) \quad \forall \mathbf{x} \in \mathbb{R}^n, \forall \mathbf{v} \in \mathbb{R}^p, \quad (1.7)$$

čiže je sedlovým bodom Lagrangeovej funkcie (1.6). Potom $\hat{\mathbf{x}}$ je optimálnym riešením úlohy (\mathcal{PR}) .

DÔKAZ: Dá sa dokázať analogicky ako Tvrdenie 1 v [41]. □

Uvedené výsledky použijeme na odvodenie nutných a postačujúcich podmienok pre úlohu s ohraničením v tvare nerovností.

1.2 Úloha s ohraničeniami v tvare nerovniíc

Na tomto mieste sa budeme zaoberať úlohou

$$\begin{aligned} &\text{minimalizovať } f(\mathbf{x}) \\ &\text{za podmienok } g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, m. \end{aligned} \quad (\mathcal{PNR})$$

Analogicky ako v časti 1.1 predpokladajme $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ($i = 1, \dots, m$), $f \in C^1, g_i \in C^1$ pre všetky i , a takisto zavedme označenie

$$g(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))^T,$$

čiže $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Množina prípustných riešení úlohy (\mathcal{PNR}) nadobúda tvar

$$\mathbb{M} = \{\mathbf{x} \mid g(\mathbf{x}) \geq 0\}.$$

Pre potreby ďalšej analýzy zaveďme pre $\mathbf{x} \in \mathbb{M}$ označenie

$$I_A(\mathbf{x}) = \{i \mid g_i(\mathbf{x}) = 0\} \quad (1.8)$$

pre množinu indexov aktívnych ohraničení v bode \mathbf{x} , a takisto definujme *regulárny* bod.

Definícia 1.2. Bod $\hat{\mathbf{x}} \in \mathbb{M}$ nazývame *regulárnym*, ak sú gradienty aktívnych ohraničení $\nabla g_i(\hat{\mathbf{x}}), \forall i \in I_A(\hat{\mathbf{x}})$ lineárne nezávislé.

Označme $q = |I_A(\hat{\mathbf{x}})|$ počet aktívnych ohraničení v $\hat{\mathbf{x}}$ a podobne ako v časti 1.1 predpokladajme $q \leq n$. Pre odvodenie nasledujúcich tvrdení najskôr pretransformujeme úlohu (\mathcal{PNR}) zavedením doplnkových premenných $\xi_i, i = 1, \dots, m$ na úlohu (\mathcal{PR}) a použijeme vetu 1.1 na túto transformovanú úlohu. Výsledkom budú tzv. Karush-Kuhn-Tuckerove (KKT) podmienky (pre detaily tohto postupu viď [3], str. 312 - 313).

Definujme transformovaný problém

$$\begin{aligned} &\text{minimalizovať} && f(\mathbf{x}) \\ &\text{za podmienok} && \xi_i^2 - g_i(\mathbf{x}) = 0, \quad i = 1, \dots, m. \end{aligned} \quad (1.9)$$

Nech $\hat{\mathbf{x}}$ je lokálnym riešením úlohy (\mathcal{PNR}) a takisto nech je regulárnym bodom (podľa def. 1.2), potom bod $(\hat{\mathbf{x}}, \hat{\boldsymbol{\xi}}) = (\hat{\mathbf{x}}, \sqrt{g_1(\hat{\mathbf{x}})}, \dots, \sqrt{g_m(\hat{\mathbf{x}})})$ je lokálnym riešením úlohy (1.9) a takisto je regulárnym bodom tejto úlohy (def. 1.1), čo ľahko ukážeme. Ak označíme $r_i(\mathbf{x}, \boldsymbol{\xi}) = \xi_i^2 - g_i(\mathbf{x}), i = 1, \dots, m$ a použijeme pomocné označenie $r(\mathbf{x}, \boldsymbol{\xi}) = (r_1(\mathbf{x}, \boldsymbol{\xi}), \dots, r_m(\mathbf{x}, \boldsymbol{\xi}))^T$, potom matica

$$\nabla r(\hat{\mathbf{x}}, \hat{\boldsymbol{\xi}}) = \begin{bmatrix} -\nabla g_1(\hat{\mathbf{x}})^T & 2\hat{\xi}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\nabla g_m(\hat{\mathbf{x}})^T & 0 & \cdots & 2\hat{\xi}_m \end{bmatrix}$$

má hodnotu m . Aplikujme teda nutné podmienky prvého rádu vety 1.1 na úlohu (1.9), podľa ktorých existuje (jednoznačne určený) vektor Lagrangeových multiplikátorov $\hat{\mathbf{u}} = (\hat{u}_1, \dots, \hat{u}_m)^T$ taký, že platí

$$\begin{aligned} \nabla f(\hat{\mathbf{x}}) + \sum_{i=1}^m \hat{u}_i \nabla r_i(\hat{\mathbf{x}}) &= 0, \\ \hat{\xi}_i &= \sqrt{g_i(\hat{\mathbf{x}})}, \quad i = 1, \dots, m, \end{aligned}$$

čo je ekvivalentné s

$$\nabla f(\hat{\mathbf{x}}) - \sum_{i=1}^m \hat{\mathbf{u}}_i \nabla g_i(\hat{\mathbf{x}}) = 0, \quad (1.10)$$

$$2\hat{\mathbf{u}}_i \sqrt{g_i(\hat{\mathbf{x}})} = 0, \quad i = 1, \dots, m. \quad (1.11)$$

Z (1.11) máme $\hat{\mathbf{u}}_i = 0$ pre $i \notin I_A(\mathbf{x})$, čo môžeme zapísať ako

$$\hat{\mathbf{u}}_i g_i(\hat{\mathbf{x}}) = 0, \quad i = 1, \dots, m. \quad (1.12)$$

Ak navyše predpokladáme $f \in C^2, g \in C^2$, potom aplikovaním podmienky druhého rádu (1.5) vety 1.1 máme

$$\begin{bmatrix} \boldsymbol{\omega}^T & \boldsymbol{\lambda}^T \end{bmatrix} \begin{bmatrix} \nabla_{xx}^2 L(\hat{\mathbf{x}}, \hat{\mathbf{u}}) & 0 \\ 0 & 2 \operatorname{diag}(\hat{\mathbf{u}}) \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{\lambda} \end{bmatrix} \geq 0 \quad (1.13)$$

pre všetky $(\boldsymbol{\omega}, \boldsymbol{\lambda}) \in \mathbb{R}^{n+m}$ spĺňajúce

$$2\hat{\boldsymbol{\xi}}_i \boldsymbol{\lambda}_i - \nabla g_i(\hat{\mathbf{x}})^T \boldsymbol{\omega} = 0, \quad i = 1, \dots, m. \quad (1.14)$$

pričom sme označili

$$L(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) - \sum_{i=1}^m \mathbf{u}_i g_i(\mathbf{x}), \quad L : \mathbb{R}^n \times \mathbb{R}_+^m \longrightarrow \mathbb{R}. \quad (1.15)$$

Ak zvolíme $\boldsymbol{\lambda}_i = 0$ pre $i \in I_A(\hat{\mathbf{x}})$ a využitím (1.12) a (1.13) dostaneme

$$\boldsymbol{\omega}^T \nabla_{xx}^2 L(\hat{\mathbf{x}}, \hat{\mathbf{u}}) \boldsymbol{\omega} \geq 0$$

pre $\boldsymbol{\omega}$ spĺňajúce $\nabla g_i(\hat{\mathbf{x}})^T \boldsymbol{\omega} = 0$ pre $i \in I_A(\hat{\mathbf{x}})$. Ďalej môžeme pre každé $i \in I_A(\hat{\mathbf{x}})$ vo vzťahu (1.13) zvoliť $\boldsymbol{\omega} = 0, \boldsymbol{\lambda}_i \neq 0$ a $\boldsymbol{\lambda}_j = 0$ pre $i \neq j$ a dostaneme

$$\hat{\mathbf{u}}_i \geq 0, \quad i \in I_A(\hat{\mathbf{x}}). \quad (1.16)$$

Týmto názorným postupom sme dostali sústavu podmienok (1.10), (1.12) a (1.16), ktoré spolu s podmienkou prípustnosti $g(\hat{\mathbf{x}}) \geq 0$ tvoria nutné podmienky prvého rádu pre úlohu (\mathcal{PNR}) (spomínané Karush-Kuhn-Tuckerove podmienky). Museli sme však dodatočne predpokladať, že $f \in C^2, g \in C^2$. Karush-Kuhn-Tuckerove podmienky však platia aj bez tohto predpokladu, čo nám umožňuje sformulovať nasledujúcu vetu.

Veta 1.3. *Nech $\hat{\mathbf{x}}$ je lokálnym riešením úlohy (\mathcal{PNR}) a takisto regulárnym bodom. Potom existuje (jednoznačne určený) vektor Lagrangeových multiplikátorov $\hat{\mathbf{u}} = (\hat{\mathbf{u}}_1, \dots, \hat{\mathbf{u}}_m)^T$ taký, že platí*

$$\nabla f(\hat{\mathbf{x}}) - \sum_{i=1}^m \hat{\mathbf{u}}_i \nabla g_i(\hat{\mathbf{x}}) = 0, \quad (1.17)$$

$$g(\hat{\mathbf{x}}) \geq 0, \quad (1.18)$$

$$\hat{\mathbf{u}}^T g(\hat{\mathbf{x}}) = 0, \quad (1.19)$$

$$\hat{\mathbf{u}} \geq 0. \quad (1.20)$$

Ak navyše $f \in C^2$, $g \in C^2$, potom platí

$$\boldsymbol{\omega}^T \left[\nabla^2 f(\hat{\mathbf{x}}) - \sum_{i=1}^m \hat{\mathbf{u}}_i \nabla^2 g_i(\hat{\mathbf{x}}) \right] \boldsymbol{\omega} \geq 0 \quad (1.21)$$

pre všetky $\boldsymbol{\omega}$ také, že $\nabla g_i(\hat{\mathbf{x}})^T \boldsymbol{\omega} = 0$ pre $i \in I_A(\hat{\mathbf{x}})$.

DÔKAZ: Priamy dôkaz bez transformácie na úlohu (\mathcal{PR}) a bez nutnosti predpokladu $f \in C^2, g \in C^2$ pre odvodenie vzťahov (1.17) - (1.20) je možné nájsť v [24], str. 342-343. \square

Keďže platí $\hat{\mathbf{u}} \geq 0$ a $g(\hat{\mathbf{x}}) \geq 0$, potom (1.19) je ekvivalentné s

$$\hat{\mathbf{u}}_i g_i(\hat{\mathbf{x}}) = 0, \quad i = 1, \dots, m. \quad (1.22)$$

Tento výraz nazývame *podmienkou komplementarity*, keďže z neho vyplýva, že aspoň jeden z výrazov $\hat{\mathbf{u}}_i, g_i(\hat{\mathbf{x}})$ je nulový. Ak je nulový práve jeden, čiže ak

$$\hat{\mathbf{u}}_i > 0 \quad \text{pre } i \in I_A(\hat{\mathbf{x}}), \quad (1.23)$$

hovoríme, že platí *podmienka ostrej komplementarity*.

Podobne ako v časti 1.1 môžeme formulovať postačujúce podmienky takto: Nech bod $(\hat{\mathbf{x}}, \hat{\mathbf{u}})$ spĺňa (1.17) - (1.20), v podmienke (1.21) platí pre všetky $\boldsymbol{\omega} \neq 0$ ostrá nerovnosť a navyše je splnená podmienka ostrej komplementarity. Potom $\hat{\mathbf{x}}$ je ostrým (lokálnym) optimálnym riešením úlohy (\mathcal{PNR}) .

Rovnako platí, že sedlovosť bodu $(\hat{\mathbf{x}}, \hat{\mathbf{u}})$ implikuje optimalitu $\hat{\mathbf{x}}$.

Veta 1.4. *Nech bod $(\hat{\mathbf{x}}, \hat{\mathbf{u}}) \in \mathbb{R}^n \times \mathbb{R}_+^m$ splňa*

$$L(\hat{\mathbf{x}}, \mathbf{u}) \leq L(\hat{\mathbf{x}}, \hat{\mathbf{u}}) \leq L(\mathbf{x}, \hat{\mathbf{u}}) \quad \forall \mathbf{x} \in \mathbb{R}^n, \forall \mathbf{u} \in \mathbb{R}_+^m, \quad (1.24)$$

čiže je sedlovým bodom Lagrangeovej funkcie (1.15). Potom $\hat{\mathbf{x}}$ je riešením úlohy (PNR).

DÔKAZ: Veta je prvým tvrdením v [41]. □

Je nutné poznamenať, že pre všeobecné funkcie f, g a h nevieme dokázať viac než len *lokálnu* platnosť uvedených viet (okrem viet o sedlových bodoch). K dokázaniu globálnej platnosti je potrebné, aby bola v istom zmysle „pekná“ nielen účelová funkcia, ale aj množina prípustných riešení \mathbb{M} . Tieto neurčité požiadavky je možné splniť napríklad predpokladaním konvexnosti účelovej funkcie a množiny \mathbb{M} . Dostaneme tak formuláciu úlohy *konvexného* programovania.

1.3 Úloha konvexného programovania a teória duality

Úlohou konvexného programovania je minimalizácia konvexnej funkcie na konvexnej množine. Uvedme najprv niektoré poznatky z konvexnej analýzy. Výborným zdrojom z tejto oblasti je publikácia [38].

Množinu \mathcal{C} nazývame *konvexnou*, ak \mathcal{C} obsahuje celú priamku spájajúcu ľubovoľné dva body, ktoré v nej ležia, alebo inak

$$\alpha \mathbf{x} + (1 - \alpha) \mathbf{y} \in \mathcal{C}, \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}, \alpha \in (0, 1). \quad (1.25)$$

Množina, ktorá vznikne prienikom ľubovoľného počtu konvexných množín, je konvexná.

Vektorový súčet

$$\alpha_1 \mathbf{x}_1 + \dots + \alpha_k \mathbf{x}_k$$

nazývame *konvexnou kombináciou* vektorov $\mathbf{x}_1, \dots, \mathbf{x}_k$, ak platí $\alpha_1 + \dots + \alpha_k = 1$ a $\alpha_i \geq 0$ ($i = 1, \dots, k$). Potom množina \mathcal{C} je konvexná práve vtedy, ak obsahuje všetky konvexné kombinácie svojich prvkov.

Funkciu $f : \mathbb{S} \rightarrow \mathbb{R}$, $\mathbb{S} \subseteq \mathbb{R}^n$ nazveme *konvexnou*, ak jej *epigraf*

$$\text{epi } f = \{(\mathbf{x}, \mu) \mid \mathbf{x} \in \mathbb{S}, \mu \in \mathbb{R}, f(\mathbf{x}) \leq \mu\}$$

je konvexná množina ako podmnožina \mathbb{R}^{n+1} . Takisto platí

$$f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y}), \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{S}, \alpha \in (0, 1), \quad (1.26)$$

čiže úsečka spájajúca ľubovoľné dva body funkcie f leží nad jej grafom. Pre $f \in C^1$ platí

$$f(\mathbf{x}) \geq f(\mathbf{y}) + \nabla f(\mathbf{y})^T (\mathbf{x} - \mathbf{y}), \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{S}. \quad (1.27)$$

Graf funkcie f teda leží nad jej ľubovoľnou dotykovou nadrovinou.

Pre *rýdzokonvexnú* funkciu platia vzťahy (1.26) a (1.27) s ostrými nerovnosťami. Dodajme ešte, že funkciu f nazývame *konkávnu*, ak $-f$ je konvexná.

Ak f je konvexná, potom jej Hessova matica je kladne semidefinitná, Hessián rýdzokonvexnej funkcie je kladne definitný.

Do nasledujúcej vety zaradíme dva dôležité poznatky o konvexných funkciách.

Veta 1.5. *Platí:*

(a) *Nezáporná kombinácia konvexných funkcií je konvexnou funkciou.*

(b) *Nech $f : \mathbb{R}^n \rightarrow \mathbb{R}$ je konvexná funkcia a \mathcal{C} je konvexná množina. Potom lokálne minimum funkcie f na množine \mathcal{C} je tiež globálnym minimom. Ak je f rýdzokonvexná, potom existuje najviac jedno globálne minimum f na \mathcal{C} .*

DÔKAZ: Tvrdenie (a) možno nájsť v [20], str. 78, časť (b) nájdeme napríklad v [2], str. 87. □

1.3.1 Postačujúce podmienky optimality

Teraz už môžeme sformulovať úlohu konvexného programovania

$$\begin{aligned} &\text{minimalizovať} && f(\mathbf{x}) \\ &\text{za podmienok} && g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, m, \end{aligned} \quad (\mathcal{PNR}_C)$$

kde predpokladáme $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ($i = 1, \dots, m$), f je konvexná a g_i sú konkávne funkcie. Na základe vety 1.5 môžeme očakávať, že obdobné tvrdenia o

optimálnom riešení z časti 1.2 budú pre úlohu (\mathcal{PNR}_C) platiť v globálnom meradle. Skutočne, Karush-Kuhn-Tuckerove podmienky budú postačujúcou podmienkou pre optimálne riešenie, ako vyplýva z nasledujúceho tvrdenia.

Veta 1.6. *Majme úlohu konvexného programovania (\mathcal{PNR}_C) a predpokladajme, že $f \in C^1, g \in C^1$. Nech bod $(\hat{\mathbf{x}}, \hat{\mathbf{u}}) \in \mathbb{R}^n \times \mathbb{R}_+^m$ spĺňa*

$$\nabla f(\hat{\mathbf{x}}) - \sum_{i=1}^m \hat{u}_i \nabla g_i(\hat{\mathbf{x}}) = 0, \quad (1.28)$$

$$g(\hat{\mathbf{x}}) \geq 0, \quad (1.29)$$

$$\hat{\mathbf{u}}^T \mathbf{g}(\hat{\mathbf{x}}) = 0, \quad (1.30)$$

$$\hat{\mathbf{u}} \geq 0. \quad (1.31)$$

Potom $\hat{\mathbf{x}}$ je (globálnym) optimálnym riešením.

DÔKAZ: Nájdeme v [21]. □

Vidíme, že regularita bodu $\hat{\mathbf{x}}$ nie je potrebná a takisto je automaticky splnená podmienka druhého rádu (1.21) analogickej vety 1.3, keďže Lagrangeova funkcia

$$L(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) - \sum_{i=1}^m u_i g_i(\mathbf{x}), \quad L : \mathbb{R}^n \times \mathbb{R}_+^m \longrightarrow \mathbb{R}, \quad (1.32)$$

je konvexná (podľa vety 1.5, tvrdenie **(a)**). Niektoré ďalšie tvrdenia týkajúce sa úlohy konvexného programovania uvedieme v súvislosti s jej *duálnou* úlohou.

1.3.2 Duálna úloha a teória duality

V tejto časti spomenieme niekoľko výsledkov a tvrdení *teórie duality*, ktorá skúma vlastnosti tzv. *duálnej úlohy*. Túto úlohu možno chápať ako „doplnkovú“ úlohu k úlohe (\mathcal{PNR}_C) v premennej \mathbf{u} , pričom vzájomný vzťah týchto úloh je daný bodom $(\hat{\mathbf{x}}, \hat{\mathbf{u}})$. Takýto pohľad na problematiku umožňuje nazerať na vektor multiplikátorov \mathbf{u} ako na rovnocennú veličinu s primárnym vektorom \mathbf{x} (pre podrobnosti pozri [2], kap. 5).

Budeme sa venovať konvexnej úlohe

$$\begin{aligned} &\text{minimalizovať} && f(\mathbf{x}) \\ &\text{za podmienok} && g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, m, \end{aligned} \quad (\mathcal{PNR}_C)$$

no niektoré vlastnosti duality je možné rozšíriť aj na všeobecnú úlohu (\mathcal{PNR}) , prípadne (\mathcal{PR}) . Pre zjednotenie niektorých zápisov označme

$$\mathbb{M} = \{\mathbf{x} \mid g_i(\mathbf{x}) \geq 0, i = 1, \dots, m\}$$

množinu prípustných riešení úlohy (\mathcal{PNR}_C) . Predpokladajme navyše, že $f(\hat{\mathbf{x}}) > -\infty$. Úlohu (\mathcal{PNR}_C) nazveme v kontexte duality *primárnou* a označíme

$$\hat{f} = \min_{\mathbf{x} \in \mathbb{M}} f(\mathbf{x})$$

jej optimálnu hodnotu. Vďaka nej môžeme uviesť alternatívnu definíciu optimálneho vektora Lagrangeových multiplikátorov.

Definícia 1.3 (Lagrangeov multiplikátor). *Vektor $\hat{\mathbf{u}}$ nazveme optimálnym vektorom Lagrangeových multiplikátorov pre primárnu úlohu (\mathcal{PNR}_C) , ak $\hat{\mathbf{u}} \geq 0$ a*

$$\hat{f} = \min_{\mathbf{x} \in \mathbb{R}^n} L(\mathbf{x}, \hat{\mathbf{u}}).$$

Definujme *duálnu funkciu* d predpisom

$$d(\mathbf{u}) = \min_{\mathbf{x} \in \mathbb{R}^n} L(\mathbf{x}, \mathbf{u}) \tag{1.33}$$

a priradíme k nej *duálnu úlohu*

$$\begin{aligned} &\text{maximalizovať} && d(\mathbf{u}) \\ &\text{za podmienok} && \mathbf{u} \geq 0. \end{aligned} \tag{DNRC}$$

Z definície funkcie d vyplýva, že je konkávna, a označme

$$\hat{d} = \max_{\mathbf{u} \geq 0} d(\mathbf{u})$$

jej optimálnu hodnotu. Vzťah úloh (\mathcal{PNR}_C) a (\mathcal{DNRC}) popíšeme pomocou *slabej vety o dualite*.

Veta 1.7 (Slabá veta o dualite). *Platí*

$$\hat{d} \leq \hat{f}.$$

Ak $\hat{d} = \hat{f}$, hovoríme, že platí *silná dualita*, a ak $\hat{d} < \hat{f}$, hovoríme o výskyte *duálnej medzery*. Existencia optimálneho vektora Lagrangeových multiplikátorov z definície 1.3 implikuje platnosť silnej duality, a v tomto prípade takisto platí, že $\hat{\mathbf{u}}$ je

optimálnym vektorom Lagrangeových multiplikátorov pre (\mathcal{PNR}_C) práve vtedy, ak je optimálnym riešením úlohy (\mathcal{DNR}_C) .

Ak existuje duálna medzera, potom je množina optimálnych vektorov Lagrangeových multiplikátorov prázdna. Za určitých doplnujúcich predpokladov (napr. existencia vnútorného bodu, t.j. $\exists \bar{x} : g(\bar{x}) > 0 \forall i$) platí silná dualita pre úlohy (\mathcal{PNR}_C) a (\mathcal{DNR}_C) . Potom namiesto riešenia jednej z nich môžeme exvivalentne riešiť druhú, napríklad s cieľom využiť jej výhodnejšiu štruktúru (napríklad lineárne a kvadratické programovanie). Moderným prístupom je riešiť oba problémy „súčasne“, na základe čoho ich potom nazývame primárno-duálnymi metódami.

Na záver ešte uvedieme dve vety.

Veta 1.8. *Vektor $\hat{x} \in \mathbb{R}^n$ je optimálnym riešením úlohy (\mathcal{PNR}_C) a $\hat{u} \in \mathbb{R}_+^m$ je optimálnym riešením úlohy (\mathcal{DNR}_C) vtedy a len vtedy, ak (\hat{x}, \hat{u}) je sedlovým bodom Lagrangeovej funkcie*

$$L(\hat{x}, u) \leq L(\hat{x}, \hat{u}) \leq L(x, \hat{u}), \quad \forall x \in \mathbb{R}^n, \forall u \in \mathbb{R}_+^m.$$

Záverečná veta predstavuje postačujúce podmienky pre všeobecnú úlohu (\mathcal{PNR}) cez globálne minimum Lagrangeovej funkcie $L(\cdot, \hat{u})$.

Veta 1.9. *Nech \hat{u} je optimálny vektor Lagrangeových multiplikátorov. Potom \hat{x} je globálnym riešením primárnej úlohy (\mathcal{PNR}) vtedy a len vtedy, ak $\hat{x} \in \mathbb{M}$ a*

$$\hat{x} = \arg \min_{x \in \mathbb{R}^n} L(x, \hat{u}), \quad \hat{u}^T g(\hat{x}) = 0.$$

Sedlový bod Lagrangeovej funkcie je kľúčovým objektom pri hľadaní optimálneho riešenia úlohy (\mathcal{PNR}) , prípadne (\mathcal{PNR}_C) . Algoritmické využitie klasickej Lagrangeovej funkcie však nie je postačujúce. Jedným z možných spôsobov je použitie tzv. *rozšírených Lagrangeových funkcií*, ktorých hlavné myšlienky si predstavíme v nasledujúcej časti.

Kapitola 2

Metódy rozšírených Lagrangeových funkcií

V predchádzajúcej kapitole sme vyjadrili vo vete 1.3 pomocou klasickej Lagrangeovej funkcie nutné a v prípade konvexnej úlohy aj postačujúce podmienky pre optimálne riešenie úlohy (\mathcal{PNR}) , ktoré sú realizované sústavou rovníc a nerovníc (1.17) - (1.20). Každá dvojica vektorov $(\hat{\mathbf{x}}, \hat{\mathbf{u}})$, ktorá rieši tento systém, predstavuje kandidátov na (lokálne) optimálne riešenie. Snaha o algoritmické hľadanie takejto dvojice pomocou sedlového bodu klasickej Lagrangeovej funkcie má niekoľko nedostatkov. V tejto časti sa preto budeme zaoberať skupinou metód, ktorých hlavným nástrojom je *rozšírená* Lagrangeova funkcia.

Hlavnou ideou metód rozšírených Lagrangeových funkcií je aproximovať pôvodnú úlohu (\mathcal{PNR}) sekvenciou subproblémov, ktorých riešenie je výrazne jednoduchšie. Riešením uvedených subproblémov budeme generovať postupnosť bodov $\{\mathbf{x}^k\}$, $\{\mathbf{u}^k\}$, ktoré budú stále lepšími aproximáciami optimálneho riešenia. Hoci táto schéma vytvára nekonečnú postupnosť aproximácií, z praktického hľadiska ju však stačí ukončiť, ak sme dostatočne spokojní s kvalitou aproximácie.

Koncepcia metód rozšírených Lagrangeových funkcií je veľmi podobná klasickým penalizačným metódam z 50. a 60. rokov minulého storočia. V týchto metódach zanedbáme niektoré alebo všetky ohraničenia a k hodnote účelovej funkcie pripočítame penalizačný člen, ktorý priradí vysokú hodnotu všetkým neprípustným vektorom \mathbf{x} . K penalizačnému členu je priradený parameter r^k , ktorý kontroluje veľkosť penalizácie. Cieľom je potom generovať postupnosť $\{r^k\} \rightarrow \infty$, ktorá zaručí, že $\{\mathbf{x}^k\} \rightarrow \hat{\mathbf{x}}$,

kde \mathbf{x}^k určíme ako minimum príslušnej penalizovanej funkcie. Avšak práve požiadavka $\{r^k\} \rightarrow \infty$, ktorá spôsobuje výrazné zhoršenie podmienenosti problému a taktiež pomerne pomalá konvergencia sú hlavnými nedostatkami penalizačných metód. Tieto fakty boli jedným z dôvodov, ktoré podnietili vznik prvých metód rozšírených Lagrangeových funkcií, ktoré tieto negatíva do značnej miery eliminujú.

Predtým, ako sa dostaneme k bližšej analýze niektorých metód z triedy rozšírených Lagrangeových funkcií, uvedieme pomerne všeobecnú vetu o sedlovom bode.

Veta 2.1 (Roode). *Majme všeobecnú úlohu matematického programovania*

$$\begin{aligned} & \text{minimalizovať } f(\mathbf{x}) \\ & \text{za podmienok } \mathbf{x} \in \mathbb{M} \end{aligned} \tag{2.1}$$

kde $\mathbb{M} \subset \mathbb{X} \subseteq \mathbb{R}^n$ a $f : \mathbb{X} \rightarrow \mathbb{R}$. Nech $\mathbb{Y} \subseteq \mathbb{R}^m$ a $\Gamma : \mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{R}$ má nasledovné vlastnosti:

$$(A1) \quad \forall \boldsymbol{\mu} \in \mathbb{Y}, \quad \forall \mathbf{x} \in \mathbb{M} : \quad \Gamma(\mathbf{x}, \boldsymbol{\mu}) \leq 0$$

$$(A2) \quad \exists \bar{\boldsymbol{\mu}} \in \mathbb{Y}, \quad \forall \mathbf{x} \in \mathbb{M} : \quad \Gamma(\mathbf{x}, \bar{\boldsymbol{\mu}}) = 0$$

$$(A3) \quad \forall \mathbf{x} \notin \mathbb{M} : \quad \sup_{\boldsymbol{\mu} \in \mathbb{Y}} \Gamma(\mathbf{x}, \boldsymbol{\mu}) = \infty$$

Nech $(\hat{\mathbf{x}}, \hat{\boldsymbol{\mu}}) \in \mathbb{X} \times \mathbb{Y}$ je sedlovým bodom rozšírenej Lagrangeovej funkcie

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) = f(\mathbf{x}) + \Gamma(\mathbf{x}, \boldsymbol{\mu}), \tag{2.2}$$

$\mathcal{L} : \mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{R}$ v zmysle

$$\mathcal{L}(\hat{\mathbf{x}}, \boldsymbol{\mu}) \leq \mathcal{L}(\hat{\mathbf{x}}, \hat{\boldsymbol{\mu}}) \leq \mathcal{L}(\mathbf{x}, \hat{\boldsymbol{\mu}}), \quad \forall \mathbf{x} \in \mathbb{X}, \quad \forall \boldsymbol{\mu} \in \mathbb{Y}. \tag{2.3}$$

Potom $\hat{\mathbf{x}}$ je (lokálnym) optimálnym riešením úlohy (2.1).

DÔKAZ: Je možné nájsť v [21]. □

Analógia s vetami o sedlovom bode z predošlých častí teda platí aj pre rozšírené Lagrangeove funkcie definované vzťahom (2.2), pričom ich penalizačná funkcia Γ musí spĺňať podmienky (A1)-(A3), ktoré budeme nazývať *Roodeho axiómy*.

Prvú metódu z triedy rozšírených Lagrangeových funkcií rozpracovali Hestenes [22] a nezávisle Powell [35] pre úlohu (\mathcal{PR}). Práve tejto metóde, v literatúre niekedy označovanej ako klasická metóda multiplikátorov [1], venujeme nasledujúcu časť.

2.1 Metóda Hestenesa a Powella pre úlohu s ohra- ničeniami v tvare rovníc

V tejto časti budeme uvažovať úlohu v tvare

$$\begin{aligned} & \text{minimalizovať } f(\mathbf{x}) \\ & \text{za podmienok } h(\mathbf{x}) = 0 \end{aligned} \tag{PR}$$

kde $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$, $f, h \in C^2$ sú dané funkcie, $h(\mathbf{x}) = (h_1(\mathbf{x}), \dots, h_p(\mathbf{x}))^T$. Na tomto mieste zatiaľ nebudeme požadovať konvexnosť úlohy (PR) a stručný prehľad metódy multiplikátorov urobíme vo všeobecnosti aj pre nekonvexné úlohy. Spomenieme však výhody a teoretické zjednodušenia, ktoré prináša konvexná úloha (v tomto prípade je potrebné, aby f bola konvexná a všetky funkcie h_j afinné).

Predpokladajme, že optimálne riešenie $\hat{\mathbf{x}}$ úlohy (PR) spĺňa nasledujúci predpoklad.

Predpoklad 2.1. *Vektor $\hat{\mathbf{x}}$ je ostrým lokálnym minimom a regulárnym bodom úlohy (PR), pričom platí*

$$\boldsymbol{\omega}^T \mathbf{L}(\hat{\mathbf{x}}, \hat{\mathbf{v}}) \boldsymbol{\omega} > 0$$

pre všetky $\boldsymbol{\omega} \neq 0$ spĺňajúce $\nabla h(\hat{\mathbf{x}}) \boldsymbol{\omega} = 0$

Tento predpoklad zabezpečuje ostrú lokálnu konvexnosť klasickej Lagrangeovej funkcie (1.6) na vybranom podpriestore. Definujme *rozšírenú Lagrangeovu funkciu* pre úlohu (PR) ako

$$\begin{aligned} \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) &= f(\mathbf{x}) + \sum_{i=1}^p \left[z_i h_i(\mathbf{x}) + \frac{r}{2} h_i^2(\mathbf{x}) \right] \\ &= f(\mathbf{x}) + h(\mathbf{x})^T \mathbf{z} + \frac{r}{2} h(\mathbf{x})^T h(\mathbf{x}), \end{aligned} \tag{2.4}$$

kde r je kladný penalizačný parameter a \mathbf{z} označuje vektor zovšeobecnených Lagrangeových multiplikátorov. V súlade s vetou 2.1 máme $\Gamma(\mathbf{x}, \mathbf{z}) = h(\mathbf{x})^T \mathbf{z} + \frac{r}{2} h(\mathbf{x})^T h(\mathbf{x})$, $\mathbb{X} = \mathbb{R}^n$, $\mathbb{Y} = \mathbb{R}^p$ a $\mathbb{M} = \{\mathbf{x} \mid h(\mathbf{x}) = 0\}$, z čoho možno jednoducho overiť platnosť Roodeho axióm.

Funkcia (2.4) predstavuje určité *rozšírenie* klasickej Lagrangeovej funkcie, keďže

$$\mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) = \mathbf{L}(\mathbf{x}, \mathbf{z}) + \frac{r}{2} h(\mathbf{x})^T h(\mathbf{x}).$$

Takisto je zrejmá podobnosť s pôvodnou penalizačnou funkciou, ktorá zodpovedá $\mathcal{L}_{\text{PH}}(\mathbf{x}, 0)$. Ďalej si všimnime, že platí

$$\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}, \boldsymbol{\mu}) = \nabla_x L(\mathbf{x}, \bar{\boldsymbol{\mu}}), \quad \text{pre } \bar{\boldsymbol{\mu}} = \boldsymbol{\mu} + rh(\mathbf{x}),$$

a keďže pre optimálne riešenie $\hat{\mathbf{x}}$ platí $h(\hat{\mathbf{x}}) = 0$, máme $\nabla_x \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}) = \nabla_x L(\hat{\mathbf{x}}, \hat{\mathbf{v}}) = 0$, z čoho nakoniec dostávame, že $\hat{\mathbf{z}} = \hat{\mathbf{v}}$. Vektory multiplikátorov \mathbf{v} a \mathbf{z} sa v optimálnom bode rovnajú.

Formálny popis jedného kroku klasického algoritmu Powell-Hestenesa je možné sformulovať takto:

Algoritmus (Powell-Hestenes). *Nech je daný Lagrangeov multiplikátor \mathbf{z}^k a penalizačný parameter r^k . Vektor \mathbf{x}^k získame minimalizáciou funkcie $\mathcal{L}_{\text{PH}}(\cdot, \mathbf{z}^k)$ na priestore \mathbb{R}^n . Potom položíme*

$$\mathbf{z}^{k+1} = \mathbf{z}^k + r^k h(\mathbf{x}^k), \quad (2.5)$$

zvolíme nový penalizačný parameter $r^{k+1} \geq r^k$ a proces zopakujeme.

Počiatočný vektor \mathbf{z}^0 môže byť ľubovoľný a postupnosť r^k môže byť zvolená a priori, alebo na základe rýchlosti konverencie podľa nejakého pravidla.

Je samozrejmé sa pýtať, či je algoritmus Powell-Hestenesa dobre definovaný, pričom otázkou môže byť najmä existencia minima funkcie $\mathcal{L}_{\text{PH}}(\cdot, \mathbf{z}^k)$, prípadne ako sa mení vzdialenosť týchto miním od bodu $\hat{\mathbf{x}}$ v závislosti od hodnoty multiplikátora \mathbf{z} a parametra r . Keďže máme

$$\nabla_x \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}) = \nabla f(\hat{\mathbf{x}}) + \nabla h(\hat{\mathbf{x}})^T [\hat{\mathbf{z}} + rh(\hat{\mathbf{z}})] = 0, \quad (2.6)$$

$$\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}) = \nabla_{xx}^2 L(\hat{\mathbf{x}}, \hat{\mathbf{z}}) + r \nabla h(\hat{\mathbf{x}})^T \nabla h(\hat{\mathbf{x}}), \quad (2.7)$$

potom sa dá ukazať (napr. pomocou Lemmy 3.2.1 z [3]), že

$$\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}) > 0 \quad \forall r \geq \bar{r}, \quad (2.8)$$

kde \bar{r} je kritická hodnota penalizačného parametra (vopred neznáma). Teda na základe (2.6) a (2.8) vidíme, že $\hat{\mathbf{x}}$ je ostrým lokálnym minimom $\mathcal{L}_{\text{PH}}(\cdot, \hat{\mathbf{z}})$ pre všetky $r \geq \bar{r}$. Môžeme sa teda domnievať, že pre všetky \mathbf{z} dostatočne blízke k $\hat{\mathbf{z}}$ existuje minimum $\mathcal{L}_{\text{PH}}(\cdot, \mathbf{z})$ pre všetky $r \geq \bar{r}$. Ako uvidíme z nasledujúceho tvrdenia, minimum $\mathcal{L}_{\text{PH}}(\cdot, \mathbf{z})$ bude existovať aj pre \mathbf{z} vzdialené od $\hat{\mathbf{z}}$, ak je paramater r dostatočne veľký.

Veta 2.2. *Nech je dané \bar{r} také, že*

$$\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}) > 0$$

a nech platí predpoklad 2.1. Potom existujú kladné konštanty δ, ε, K , že platí

(a) *Pre všetky $(\mathbf{z}, r) \in \mathbb{D} \subset \mathbb{R}^{p+1}$, kde \mathbb{D} je daná ako*

$$\mathbb{D} = \{(\mathbf{z}, r) \mid \|\mathbf{z} - \hat{\mathbf{z}}\| < \delta r, r \geq \bar{r}\},$$

má úloha

$$\begin{aligned} &\text{minimalizovať } \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \\ &\text{za podmienok } \|\mathbf{x} - \hat{\mathbf{x}}\| < \varepsilon \end{aligned}$$

jediné riešenie $\mathbf{x}(\mathbf{z})$. Funkcia $\mathbf{x}(\mathbf{z})$ je spojito diferencovateľná vo vnútri množiny \mathbb{D} .

(b) *Pre všetky $(\mathbf{z}, r) \in \mathbb{D}$ máme*

$$\|\mathbf{x}(\mathbf{z}) - \hat{\mathbf{x}}\| \leq \frac{K}{r} \|\mathbf{z} - \hat{\mathbf{z}}\| \quad (2.9)$$

a

$$\|\bar{\mathbf{z}} - \hat{\mathbf{z}}\| \leq \frac{K}{r} \|\mathbf{z} - \hat{\mathbf{z}}\|, \quad (2.10)$$

kde

$$\bar{\mathbf{z}} = \mathbf{z} + rh(\mathbf{x}(\mathbf{z})),$$

(c) *Pre všetky $(\mathbf{z}, r) \in \mathbb{D}$ je matica $\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z})$ kladne definitná a $\nabla h(\mathbf{x}(\mathbf{z}))$ má hodnotu p .*

DÔKAZ: Dôkaz tvrdenia možno nájsť v [1], str. 108-111, alebo postupovať rovnako ako v [28]. □

Skutočne vidíme, že aj keď zvolíme počiatočný vektor \mathbf{z}^0 príliš ďaleko od $\hat{\mathbf{z}}$, konvergenciu zaručíme voľbou dostatočne veľkého parametra $r \geq \bar{r}$. Pomocou tejto vety môžeme dokázať konvergenciu metódy a ukázať rýchlosť konvergenie v prípade, že iterácia multiplikátorov je daná vzťahom

$$\mathbf{z}^{k+1} = \mathbf{z}^k + r^k h(\mathbf{x}^k).$$

Presnejšie, ak označíme

$$\mathbf{B} = \left[\nabla h(\hat{\mathbf{x}}) (\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}))^{-1} \nabla h(\hat{\mathbf{x}})^{\text{T}} \right]^{-1} - r \mathbf{I} \quad (2.11)$$

pre všetky r také, že $(\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\hat{\mathbf{x}}, \hat{\mathbf{z}}))^{-1}$ existuje a ak

$$\bar{r} > \max_{i=1, \dots, p} \{ -2e_i \},$$

kde e_i sú vlastné čísla matice \mathbf{B} , potom za určitých dodatočných predpokladoch o \mathbf{z}^0 a $r^0 \geq \bar{r}$ vieme ukázať, že $\{\mathbf{z}^k\} \rightarrow \hat{\mathbf{z}}$ a $\{\mathbf{x}(\mathbf{z}^k, r^k)\} \rightarrow \hat{\mathbf{x}}$. Navyiac ak $\limsup_{k \rightarrow \infty} r^k = \hat{r} < \infty$ a $\mathbf{z}^k \neq \hat{\mathbf{z}}$ pre všetky k , potom

$$\limsup_{k \rightarrow \infty} \frac{\|\mathbf{z}^{k+1} - \hat{\mathbf{z}}\|}{\|\mathbf{z}^k - \hat{\mathbf{z}}\|} \leq \max_{i=1, \dots, p} \left| \frac{e_i}{e_i + \hat{r}} \right|, \quad (2.12)$$

a ak $\{r^k\} \rightarrow \infty$ a $\mathbf{z}^k \neq \hat{\mathbf{z}}$ pre všetky k , potom

$$\limsup_{k \rightarrow \infty} \frac{\|\mathbf{z}^{k+1} - \hat{\mathbf{z}}\|}{\|\mathbf{z}^k - \hat{\mathbf{z}}\|} = 0. \quad (2.13)$$

Teda z (2.12) vidíme, že algoritmus dosahuje lineárnu konvergenciu, ak je postupnosť r^k ohraničená a superlineárnu konvergenciu (podľa (2.13)), ak $r^k \rightarrow \infty$. Tieto odhady však vo všeobecnosti už nemožno zlepšiť.

Niekoľko poznámok k Powell-Hestenesovmu algoritmu:

- (a) Opakovanie jednotlivých krokov Powell-Hestenesovho algoritmu je vo všeobecnosti nekonečný proces. Kvôli implementovateľnosti a výpočtovej náročnosti sa pri riešení jednotlivých minimalizačných subproblémov uspokojíme aj s približným riešením, teda takým, ktoré spĺňa

$$\|\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}_\alpha(\mathbf{z}), \mathbf{z})\| \leq \alpha.$$

Pre takto definovaný algoritmus vieme dokázať podobnú vetu ako je Veta 2.2, v ktorej uvažujeme množinu \mathbb{D} definovanú ako

$$\mathbb{D} = \left\{ (\mathbf{z}, r, \alpha) \left| \sqrt{\frac{\|\mathbf{z} - \hat{\mathbf{z}}\|^2}{r^2} + \|\alpha\|^2} < \delta, r \geq \bar{r} \right. \right\}.$$

Potom sa pravé strany odhadov (2.9) a (2.10) zmenia na

$$\frac{K}{r} \sqrt{\|\mathbf{z} - \hat{\mathbf{z}}\|^2 + \|r\alpha\|^2}.$$

V platnosti zostanú aj odhady rýchlosti konvergenzie (2.12) a (2.13).

- (b) Pre prípad konvexného programovania dostávame samozrejme silnejšie výsledky. V tomto prípade stačí napríklad uvažovať ľubovoľné \bar{r} kladné ako kritickú hodnotu parametra r . Minimum funkcie $\mathcal{L}_{\text{PH}}(\cdot, \mathbf{z})$ existuje pre ľubovoľnú dvojicu $(\mathbf{z}, r) \in \mathbb{R}^p \times \mathbb{R}_{++}$. Takisto pre iteráciu multiplikátorov

$$\mathbf{z}^{k+1} = \mathbf{z}^k + r^k h(\mathbf{x}^k),$$

platí $\{\mathbf{z}^k\} \rightarrow \hat{\mathbf{z}}$ pre ľubovoľné hodnoty \mathbf{z}^0 a r^0 . Rýchlosť konvergenzie zostáva v platnosti.

- (c) Dôležitou otázkou z hľadiska praktických výpočtov zostáva, ako určiť postupnosť r^k . Počiatočná hodnota r^0 nesmie byť príliš veľká kvôli zhoršeniu podmienenosti úlohy v prvej minimalizačnej iterácii (Hestenes navrhuje použiť $\mathbf{z} = 0$ a teda prvá iterácia zodpovedá minimalizácii klasickej penalizačnej funkcie), a jeho hodnota nesmie rásť príliš rýchlo. Často sa používa predpis $r^{k+1} = \gamma r^k$, napríklad pre $\gamma \in (1, 8]$. Inou možnosťou je zväčšiť r^k iba vtedy, ak $\|h(\mathbf{x}(\mathbf{z}^k))\|$ klesá k nule príliš „pomaly“. Ďalšou možnosťou je použiť vektor penalizačných parametrov, každý priradený jednému ohraničeniu a meniť len tie, ktorých ohraničenia nespĺňajú $\|h_i(\mathbf{x}(\mathbf{z}^{k+1}))\| < \beta \|h_i(\mathbf{x}(\mathbf{z}^k))\|$ pre $\beta \in (0, 1)$. Pre konvexné úlohy je možné ponechať $r^k = r^0$ pre všetky k . Idey zmeny penalizačného parametra sú uvedené napr. v [8].
- (d) Otázka požadovanej presnosti minimalizácie vyjadrenej pomocou $\|\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}^k), \mathbf{z}^k)\|$ je takisto otvorená. Jednou z možností je požadovať, aby $\|\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}^k), \mathbf{z}^k)\| < \varepsilon^k$, kde $\{\varepsilon^k\} \rightarrow 0$, resp. $\varepsilon^k = \varepsilon^0 \ll 1 \forall k$. Bertsekas [1] navrhuje použiť

$$\varepsilon^k = \frac{\eta^k r^k}{2} \|h(\mathbf{x}(\mathbf{z}^k))\|^2,$$

pričom $\{\eta^k\} \rightarrow 0$ je vopred zvolená nezáporná postupnosť, a Polyak [33] používa

$$\varepsilon^k = \frac{\sigma}{r^k} \|\bar{\mathbf{z}} - \mathbf{z}^k\|,$$

kde $\bar{\mathbf{z}} = \mathbf{z}^k + r^k h(\mathbf{x}(\mathbf{z}^k))$ a $\sigma > 0$ je zvolená konštanta.

Doposiaľ sme sa príliš nevenovali motivácii pre iteráciu multiplikátorov v tvare

$$\mathbf{z}^{k+1} = \mathbf{z}^k + r^k h(\mathbf{x}^k).$$

Jednou z nich je fakt, že pre takto definovaný odhad multiplikátora \mathbf{z}^{k+1} platí $\nabla_{\mathbf{x}}L(\mathbf{x}^k, \mathbf{z}^{k+1}) = 0$. Ako uvidíme v ďalšej časti, z hľadiska duálnej funkcie pre problém (\mathcal{PR}) ide o iteráciu gradientnej metódy s krokom r^k .

2.1.1 Iterácie multiplikátorov

V tejto časti sa pokúsime zhrnúť základné poznatky o rôznych metódach odhadovania Lagrangeových multiplikátorov. V algoritme Powell-Hestenesa sme sa zatiaľ zmienili iba o jednej z nich (2.5), často nazývanej ako metóda prvého rádu. Ukážeme, že je možné odvodiť aj metódy, ktoré využívajú informáciu Hessovej matice rozšírenej Lagrangeovej funkcie, a preto sa s nimi môžeme v literatúre stretnúť pod názvom metódy druhého rádu. Ukážeme odvodenie týchto metód pomocou duálnej funkcie problému (\mathcal{PR}) , a ako sa neskôr ukáže, metóda bude zodpovedať jednej iterácii Newtonovej metódy aplikovanej na primárno-duálny systém rovníc. Dobrým zdrojom sú články Byrda [7], Tapiu [16], [43], a monografia Bertsekasa [1].

Nech \bar{r}, δ a ε sú ako vo vete (2.2), a pre dvojicu (\mathbf{z}, r) patriacu do množiny

$$\mathbb{D} = \{(\mathbf{z}, r) \mid \|\mathbf{z} - \hat{\mathbf{z}}\| < \delta r, r \geq \bar{r}\} \quad (2.14)$$

definujme duálnu funkciu predpisom

$$d_{\text{PH}}(\mathbf{z}) = \min_{\|\mathbf{x} - \hat{\mathbf{x}}\| < \varepsilon} \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}(\mathbf{z})) + h(\mathbf{x}(\mathbf{z}))^T \mathbf{z} + \frac{r}{2} h(\mathbf{x}(\mathbf{z}))^T h(\mathbf{x}(\mathbf{z})). \quad (2.15)$$

Keďže z vety (2.2) vieme, že $\mathbf{x}(\cdot)$ je diferencovateľná, potom to isté platí aj o $d_{\text{PH}}(\cdot)$. Spočítajme gradient a Hessovu maticu funkcie d_{PH} . Dostávame

$$\nabla_{\mathbf{z}} d_{\text{PH}} = \nabla_{\mathbf{z}} \mathbf{x}(\mathbf{z}) \nabla_{\mathbf{x}} \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) + h(\mathbf{x}(\mathbf{z})) \quad (2.16)$$

Keďže

$$\nabla_{\mathbf{x}} \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) = 0, \quad (2.17)$$

máme

$$\nabla_{\mathbf{z}} d_{\text{PH}} = h(\mathbf{x}(\mathbf{z})). \quad (2.18)$$

Pre Hessovu maticu dostávame

$$\nabla_{\mathbf{z}\mathbf{z}}^2 d_{\text{PH}} = \nabla_{\mathbf{z}} \mathbf{x}(\mathbf{z}) \nabla h(\mathbf{x}(\mathbf{z}))^T. \quad (2.19)$$

S využitím, že pre všetky dvojice $(\mathbf{z}, r) \in \mathbb{D}$ máme (2.17), potom po derivovaní tohto vzťahu podľa \mathbf{z} máme

$$\nabla_{\mathbf{z}} \mathbf{x}(\mathbf{z}) \nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) + \nabla_{xz}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) = 0$$

a s využitím

$$\nabla_{xz}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z})) = \nabla h(\mathbf{x}(\mathbf{z})).$$

dostávame

$$\nabla_{\mathbf{z}} \mathbf{x}(\mathbf{z}) = -\nabla h(\mathbf{x}(\mathbf{z})) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) \right]^{-1}$$

a napokon

$$\nabla_{zz}^2 d_{\text{PH}} = -\nabla h(\mathbf{x}(\mathbf{z})) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) \right]^{-1} \nabla h(\mathbf{x}(\mathbf{z}))^{\text{T}}. \quad (2.20)$$

Keďže z vety (2.2) máme, že $\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}(\mathbf{z}), \mathbf{z}) > 0$ a matica $\nabla h(\mathbf{x}(\mathbf{z}))$ má hodnotu p pre $(\mathbf{z}, r) \in \mathbb{D}$, dostávame z (2.20), že $\nabla_{zz}^2 d_{\text{PH}}(\mathbf{z}) < 0$ pre $(\mathbf{z}, r) \in \mathbb{D}$. Naviac, z (2.18) máme pre $r \geq \bar{r}$

$$\nabla d_{\text{PH}}(\hat{\mathbf{z}}) = h(\mathbf{x}(\hat{\mathbf{z}})) = h(\hat{\mathbf{x}}) = 0,$$

Teda, pre všetky $r \geq \bar{r}$, $\hat{\mathbf{z}}$ maximalizuje funkciu d_{PH} na množine $\{\mathbf{z} \mid \|\mathbf{z} - \hat{\mathbf{z}}\| < \delta r\}$. Potom môžeme iteráciu (2.5) zapísať ako

$$\mathbf{z}^{k+1} = \mathbf{z}^k + r^k \nabla d_{\text{PH}}(\mathbf{z}^k), \quad (2.21)$$

čiže iterácia (2.5) skutočne zodpovedá iterácii gradientnej metódy s krokom r^k .

Rovnako ako sme uvažovali iteráciu v tvare

$$\mathbf{z}^{k+1} = \mathbf{z}^k + r^k \nabla d_{\text{PH}}(\mathbf{z}^k),$$

môžeme uvažovať Newtonovu iteráciu (pozri prílohu A) na maximalizovanie funkcie d_{PH}

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \left[\nabla_{zz}^2 d_{\text{PH}}(\mathbf{z}^k) \right]^{-1} \nabla d_{\text{PH}}(\mathbf{z}^k).$$

Po využití (2.18) a (2.20) dostávame vzťah

$$\mathbf{z}^{k+1} = \mathbf{z}^k + \left[\nabla h(\mathbf{x}^k) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}^k, \mathbf{z}^k) \right]^{-1} \nabla h(\mathbf{x}^k)^{\text{T}} \right]^{-1} h(\mathbf{x}^k). \quad (2.22)$$

Kvôli analýze konvergenzie tohto odhadu uvažujme Newtonovu metódu na riešenie sústavy rovníc

$$\begin{aligned}\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) &= 0, \\ h(\mathbf{x}) &= 0,\end{aligned}$$

čiže (použili sme označenie $\Delta \mathbf{x} = \bar{\mathbf{x}} - \mathbf{x}$, $\Delta \mathbf{z} = \bar{\mathbf{z}} - \mathbf{z}$)

$$\begin{bmatrix} \nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) & \nabla h(\mathbf{x})^\top \\ \nabla h(\mathbf{x}) & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{z} \end{bmatrix} = \begin{bmatrix} -\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \\ -h(\mathbf{x}) \end{bmatrix} \quad (2.23)$$

Za predpokladov invertovateľnosti matice $\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z})$ a plnej hodnosti matice $\nabla h(\mathbf{x})$ (hodnosť p) vieme tento systém explicitne vyriešiť. Ľahko sa možno presvedčiť, že $\bar{\mathbf{x}}$, $\bar{\mathbf{z}}$ sú dané vzťahmi

$$\bar{\mathbf{z}} = \mathbf{z} + \left[\nabla h(\mathbf{x}) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \right]^{-1} \nabla h(\mathbf{x})^\top \right]^{-1} \times \quad (2.24)$$

$$\left[h(\mathbf{x}) - \nabla h(\mathbf{x}) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \right]^{-1} \nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \right]. \quad (2.25)$$

$$\bar{\mathbf{x}} = \mathbf{x} - \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \right]^{-1} \nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \quad (2.26)$$

Ak použijeme fakt, že $\nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) = 0$, zistíme, že (2.22) zodpovedá iterácii (2.24), čo nám pomôže pri analýze konvergenzie. Podobne ako pre iteráciu (2.5) môžeme sformulovať nasledujúcu vetu.

Veta 2.3. *Nech platí predpoklad 2.1 a nech \bar{r} a δ sú totožné ako v (2.2). Potom k ľubovolnej konštante γ existuje konštanta δ_2 , kde $0 < \delta_2 \leq \delta$ taká, že pre všetky dvojice (\mathbf{z}, r) z množiny \mathbb{D}_2 definovanej ako*

$$\mathbb{D}_2 = \{(\mathbf{z}, r) \mid \|\mathbf{z} - \hat{\mathbf{z}}\| < \delta_2 r, r \geq \bar{r}\},$$

dostávame nasledovné odhady:

(a) Platí

$$\|\bar{\mathbf{z}} - \hat{\mathbf{z}}\| \leq \frac{\gamma}{r} \|\mathbf{z} - \hat{\mathbf{z}}\| \quad (2.27)$$

kde

$$\bar{\mathbf{z}} = \mathbf{z} + \left[\nabla h(\mathbf{x}) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}, \mathbf{z}) \right]^{-1} \nabla h(\mathbf{x})^\top \right]^{-1} h(\mathbf{x})$$

(b) Ak sú naviac Hessove matice funkcií f, h_i Lipschitzovsky spojité, t.j.

$$\begin{aligned}\|\nabla^2 f(\mathbf{x}_1) - \nabla^2 f(\mathbf{x}_2)\| &\leq L_0 \|\mathbf{x}_1 - \mathbf{x}_2\| \\ \|\nabla^2 h_i(\mathbf{x}_1) - \nabla^2 h_i(\mathbf{x}_2)\| &\leq L_i \|\mathbf{x}_1 - \mathbf{x}_2\|, \quad i = 1, \dots, p\end{aligned}$$

na nejakom okolí bodu $\hat{\mathbf{x}}$, potom existuje konštanta K_2 taká, že pre všetky $(\mathbf{z}, r) \in \mathbb{D}_2$ máme

$$\|\bar{\mathbf{z}} - \hat{\mathbf{z}}\| \leq \frac{K_2}{r^2} \|\mathbf{z} - \hat{\mathbf{z}}\|^2. \quad (2.28)$$

DÔKAZ: Dôkaz tvrdenia nájdeme v [1], str. 136. \square

Rovnako ako pre odhad prvého rádu aj pre odhad (2.22) platí, že za určitých dodatočných predpokladoch o \mathbf{z}^0 a $r^0 \geq \bar{r}$ vieme ukázať, že $\{\mathbf{z}^k\} \rightarrow \hat{\mathbf{z}}$ a $\{\mathbf{x}(\mathbf{z}^k)\} \rightarrow \hat{\mathbf{x}}$. Rýchlosť konvergencie je v prípade $\limsup_{k \rightarrow \infty} r^k = \hat{r} < \infty$ a $\mathbf{z}^k \neq \hat{\mathbf{z}}$ superlineárna a v prípade $\{r^k\} \rightarrow \infty$ a $\mathbf{z}^k \neq \hat{\mathbf{z}}$ kvadratická.

Pri odvodení tejto iterácie sme predpokladali, že minimalizácia funkcie $\mathcal{L}_{\text{PH}}(\cdot, \mathbf{z})$ je vykonaná exaktne, čiže máme $\nabla_x \mathcal{L}_{\text{PH}}(\cdot, \mathbf{z}) = 0$. Ak namiesto toho budeme predpokladať, že platí $\|\nabla_x \mathcal{L}_{\text{PH}}(\cdot, \mathbf{z})\| \leq \alpha, \alpha \geq 0$, potom môžeme uvažovať odhad

$$\begin{aligned}\mathbf{z}^{k+1} = \mathbf{z}^k + &\left[\nabla h(\mathbf{x}^k) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}^k, \mathbf{z}^k) \right]^{-1} \nabla h(\mathbf{x}^k)^\top \right]^{-1} \times \\ &\left[h(\mathbf{x}^k) - \nabla h(\mathbf{x}^k) \left[\nabla_{xx}^2 \mathcal{L}_{\text{PH}}(\mathbf{x}^k, \mathbf{z}^k) \right]^{-1} \nabla_x \mathcal{L}_{\text{PH}}(\mathbf{x}^k, \mathbf{z}^k) \right].\end{aligned} \quad (2.29)$$

ktorý môžeme považovať za verziu odhadu (2.22) v prípade, ak minimalizačná procedúra nie je exaktná. Na tomto mieste je nutné spomenúť, že v praktických výpočtoch dosahuje (2.29) lepšie výsledky ako (2.22).

Zaujímavé je porovnanie výhod a nevýhod týchto odhadov (2.5) a (2.22). Z viet (2.2) a (2.3) vidíme, že oblasť konvergencie je v oboch prípadoch podobná, no dá sa ukázať, že kritická hodnota \bar{r} je v prípade odhadu prvého rádu (2.5) väčšia ako v prípade odhadu druhého rádu (2.22). Presnejšie, ak matica \mathbf{B} definovaná podľa (2.11) má zápornú vlastnú hodnotu a ak označíme \bar{e} najzápornejšiu z nich, potom iterácia prvého rádu vyžaduje $\bar{r} > -2\bar{e}$, zatiaľčo pre metódu druhého rádu stačí, aby $\bar{r} > -\bar{e}$. Naviac, iterácia druhého rádu má vyšší rád konvergencie. Na druhej strane, iterácia prvého rádu je výpočtovo jednoduchšia a v prípade konvexnej úlohy konverguje pre ľubovoľnú vstupnú dvojicu (\mathbf{z}^0, r^0) dokonca aj bez predpokladu diferencovateľnosti,

zatiaľčo metóda druhého rádu potrebuje druhé derivácie a koverguje len pre niektoré vstupné dvojice. Pre úlohu konvexného programovania je teda metóda prvého rádu robustnejšia.

Uvedené poznatky o metóde multiplikátorov Powella a Hestenesa aplikovanej na úlohu (\mathcal{PR}) využijeme pri odvodení a analýze tejto metódy aplikovanej na úlohu s ohraňeniami v tvare nerovností.

2.2 Metóda Rockafellara pre úlohu s ohraňeniami v tvare nerovníc

V tejto časti použijeme doposiaľ známe výsledky Powell-Hestenesovej metódy na odvodenie tzv. Rockafellarovej metódy pre úlohu v tvare

$$\begin{aligned} &\text{minimalizovať} && f(\mathbf{x}) \\ &\text{za podmienok} && g(\mathbf{x}) \geq 0 \end{aligned} \tag{PNR}$$

kde $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $g(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))^T$. Pripomeňme, že klasická Lagrangeova funkcia má pre túto úlohu tvar

$$L(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) - g(\mathbf{x})^T \mathbf{u}, \quad L : \mathbb{R}^n \times \mathbb{R}_+^m \rightarrow \mathbb{R}. \tag{2.30}$$

Tento problém je možné transformovať zavedením doplnkových premenných $\boldsymbol{\xi} = (\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_m)^T$ na ekvivalentnú úlohu

$$\begin{aligned} &\text{minimalizovať} && f(\mathbf{x}), \mathbf{x} \in \mathbb{R}^n \\ &\text{za podmienok} && \boldsymbol{\xi}_i^2 - g_i(\mathbf{x}) = 0, \quad i = 1, \dots, m. \end{aligned} \tag{2.31}$$

Naviac máme, že $\hat{\mathbf{x}}$ je lokálnym (globálnym) minimom úlohy (\mathcal{PNR}) vtedy a len vtedy, ak $(\hat{\mathbf{x}}, \hat{\boldsymbol{\xi}}_1, \dots, \hat{\boldsymbol{\xi}}_m)$, kde $\hat{\boldsymbol{\xi}}_i = \sqrt{g_i(\hat{\mathbf{x}})}$, $i = 1, \dots, m$ je lokálnym (globálnym) minimom úlohy (2.31). Pomocou tejto transformácie odvodíme tzv. Rockafellarovu [40] rozšírenú Lagrangeovu funkciu pre úlohu (\mathcal{PNR}) .

Uvažujme najprv rozšírenú Lagrangeovu funkciu

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi}) = f(\mathbf{x}) + \sum_{i=1}^m \left[\mathbf{y}_i (\boldsymbol{\xi}_i^2 - g_i(\mathbf{x})) + r (\boldsymbol{\xi}_i^2 - g_i(\mathbf{x}))^2 \right]. \tag{2.32}$$

Pri aplikovaní algoritmu Powell-Hestenesa na úlohu (2.31) musíme minimalizovať funkciu (2.32) vzhľadom na $(\mathbf{x}, \boldsymbol{\xi}) \in \mathbb{R}^{n+m}$ pre rôzne, zafixované hodnoty multiplikátora \mathbf{y} . Ukážeme, že minimalizácia $\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi})$ vzhľadom na $\boldsymbol{\xi}$ sa dá vykonať explicitne pre zafixovanú hodnotu vektora \mathbf{x} .

Keďže máme

$$\min_{\boldsymbol{\xi}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi}) = f(\mathbf{x}) + \min_{\boldsymbol{\xi}} \sum_{i=1}^m \left[\mathbf{y}_i (\boldsymbol{\xi}_i^2 - g_i(\mathbf{x})) + \frac{r}{2} (\boldsymbol{\xi}_i^2 - g_i(\mathbf{x}))^2 \right],$$

vidíme, že minimalizácia $\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi})$ vzhľadom na $\boldsymbol{\xi}_i$ je ekvivalentná s

$$\min_{\boldsymbol{\xi}_i} \left[\mathbf{y}_i (\boldsymbol{\xi}_i^2 - g_i(\mathbf{x})) + \frac{r}{2} (\boldsymbol{\xi}_i^2 - g_i(\mathbf{x}))^2 \right],$$

prípadne

$$\min_{\boldsymbol{\eta}_i \geq 0} \left[\mathbf{y}_i (\boldsymbol{\eta}_i - g_i(\mathbf{x})) + \frac{r}{2} (\boldsymbol{\eta}_i - g_i(\mathbf{x}))^2 \right]. \quad (2.33)$$

Keďže minimalizovaná funkcia v (2.33) je kvadratická v premennej $\boldsymbol{\eta}_i$, jej globálne minimum vzhľadom na $\boldsymbol{\eta}_i \in \mathbb{R}$ sa nadobúda v bode

$$\bar{\boldsymbol{\eta}}_i = g_i(\mathbf{x}) - \frac{\mathbf{y}_i}{r}.$$

Musíme uvažovať dve možnosti: ak $\bar{\boldsymbol{\eta}}_i \geq 0$, potom $\bar{\boldsymbol{\eta}}_i$ rieši minimalizačný problém (2.33) a teda $\hat{\boldsymbol{\eta}}_i = \bar{\boldsymbol{\eta}}_i$, v opačnom prípade máme $\hat{\boldsymbol{\eta}}_i = 0$. Sumárne dostávame pre $\hat{\boldsymbol{\eta}}_i$ vzťah

$$\hat{\boldsymbol{\eta}}_i = \max \left[0, g_i(\mathbf{x}) - \frac{\mathbf{y}_i}{r} \right],$$

a takisto máme

$$\hat{\boldsymbol{\eta}}_i - g_i(\mathbf{x}) = \max \left[-g_i(\mathbf{x}), -\frac{\mathbf{y}_i}{r} \right]. \quad (2.34)$$

Ak zavedieme pomocné označenia

$$g_i^+(\mathbf{x}, \mathbf{y}_i) = \max \left[-g_i(\mathbf{x}), -\frac{\mathbf{y}_i}{r} \right] = -\min \left[g_i(\mathbf{x}), \frac{\mathbf{y}_i}{r} \right], \quad (2.35)$$

a

$$g^+(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} g_1^+(\mathbf{x}, \mathbf{y}_1) \\ \vdots \\ g_m^+(\mathbf{x}, \mathbf{y}_m) \end{bmatrix},$$

potom z (2.33), (2.34) máme

$$\min_{\boldsymbol{\xi}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi}) = f(\mathbf{x}) + g^+(\mathbf{x}, \mathbf{y})^T \mathbf{y} + \frac{r}{2} g^+(\mathbf{x}, \mathbf{y})^T g^+(\mathbf{x}, \mathbf{y}). \quad (2.36)$$

Rozšířená Lagrangeova funkcia pre úlohu (\mathcal{PNR}) je teda daná vzťahom

$$\mathcal{L}_R(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + g^+(\mathbf{x}, \mathbf{y})^T \mathbf{y} + \frac{r}{2} g^+(\mathbf{x}, \mathbf{y})^T g^+(\mathbf{x}, \mathbf{y}).$$

Tento výraz je možné upraviť na ekvivalentný tvar

$$\mathcal{L}_R(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \frac{1}{2r} \sum_{i=1}^m \left[\max^2(0, \mathbf{y}_i - r g_i(\mathbf{x})) - \mathbf{y}_i^2 \right], \quad (2.37)$$

ktorý sa často používa v literatúre.

Uvedené výsledky možno sumarizovať takto: Úloha minimalizácie $\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi})$ vzhľadom na $(\mathbf{x}, \boldsymbol{\xi}) \in \mathbb{R}^{n+m}$ je ekvivalentná minimalizácii $\mathcal{L}_R(\mathbf{x}, \mathbf{y})$ v premennej $\mathbf{x} \in \mathbb{R}^n$, a teda bod $(\mathbf{x}(\mathbf{y}), \boldsymbol{\xi}(\mathbf{y}))$ je riešením prvého z týchto problémov vtedy a len vtedy, ak $\mathbf{x}(\mathbf{y})$ je riešením druhého a platí

$$|\boldsymbol{\xi}_i(\mathbf{y})| = \sqrt{\max \left[0, g_i(\mathbf{x}(\mathbf{y})) - \frac{\mathbf{y}_i}{r} \right]}, \quad i = 1, \dots, m.$$

Algoritmus Powell-Hestenesa teda môžeme aplikovať na úlohu (\mathcal{PNR}) po jej pretransformovaní na ekvivalentný problém (2.31), no pri výpočte minima funkcie $\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\xi})$ môžeme vynechať doplnkové premenné $\boldsymbol{\xi}_i, i = 1, \dots, m$, keďže namiesto tohto výpočtu môžeme ekvivalentne minimalizovať funkciu $\mathcal{L}_R(\mathbf{x}, \mathbf{y})$.

Pre funkciu

$$\Gamma(\mathbf{x}, \mathbf{y}) = \frac{1}{2r} \left[\max^2(0, \mathbf{y}_i - r g_i(\mathbf{x})) - \mathbf{y}_i^2 \right]$$

môžeme ľahko overiť platnosť Roodeho axióm z vety 2.1, kde použijeme $\mathbb{X} = \mathbb{R}^n$, $\mathbb{Y} = \mathbb{R}_+^m$ a $\mathbb{M} = \{\mathbf{x} \mid g(\mathbf{x}) \geq 0\}$. Rovnako ako v prípade rozšírenej funkcie Hestenesa a Powella platí

$$\nabla_x \mathcal{L}_R(\mathbf{x}, \boldsymbol{\mu}) = \nabla_x L(\mathbf{x}, \bar{\boldsymbol{\mu}}), \quad \text{pre } \bar{\boldsymbol{\mu}}_i = -\max[0, \boldsymbol{\mu}_i - r g_i(\mathbf{x})], \quad i = 1, \dots, m$$

a keďže $\nabla_x \mathcal{L}_R(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \nabla_x L(\hat{\mathbf{x}}, \hat{\mathbf{u}}) = 0$ a takisto $g(\hat{\mathbf{x}}) \geq 0, \hat{\mathbf{y}}^T g(\hat{\mathbf{x}}) = 0, \hat{\mathbf{y}} \geq 0$, potom $\hat{\mathbf{y}} = \hat{\mathbf{u}}$. Opäť dostávame rovnosť multiplikátorov v optimálnom bode.

Takmer všetky teoretické výsledky Powell-Hestenesovho algoritmu pre úlohu (\mathcal{PR}) je možné mechanicky rozšíriť aj pre jeho aplikáciu na úlohu (\mathcal{PNR}) . V tomto prípade budeme od optimálneho riešenia $\hat{\mathbf{x}}$ úlohy (\mathcal{PNR}) požadovať splnenie nasledovného predpokladu:

Predpoklad 2.2. Vektor $\hat{\mathbf{x}}$ je ostrým lokálnym minimom a regulárnym bodom problému (\mathcal{PNR}) , pričom platí

$$\boldsymbol{\omega}^T \mathbf{L}(\hat{\mathbf{x}}, \hat{\mathbf{u}}) \boldsymbol{\omega} > 0$$

pre všetky $\boldsymbol{\omega} \neq 0$ spĺňajúce $\nabla g_i(\hat{\mathbf{x}})^T \boldsymbol{\omega} = 0$ pre všetky $i \in I_A(\hat{\mathbf{x}}) = \{i \mid g_i(\hat{\mathbf{x}}) = 0\}$, kde $\mathbf{L}(\cdot, \cdot)$ je definovaná v (2.30). Navyiac, $\hat{\mathbf{u}}$ spĺňa podmienku ostrej komplementarity, čiže

$$\hat{u}_i > 0 \quad \text{pre všetky } i \in I_A(\hat{\mathbf{x}}).$$

Ak $\hat{\mathbf{x}}$ spĺňa predpoklad 2.2, potom lokálne minimum $(\hat{\mathbf{x}}, \sqrt{g_1(\hat{\mathbf{x}})}, \dots, \sqrt{g_m(\hat{\mathbf{x}})})$ problému (2.31) spĺňa predpoklad 2.1. Tento fakt umožňuje použiť algoritmus a jeho teoretické výsledky (napr. veta (2.2)) z časti (2.1) najprv na úlohu (2.31) a potom transformovaním na (\mathcal{PNR}) .

V prípade úlohy (\mathcal{PR}) sme definovali iteráciu multiplikátorov prvého rádu (2.5) ako

$$\mathbf{z}_j^{k+1} = \mathbf{z}_j^k + r^k h_j(\mathbf{x}^k), \quad j = 1, \dots, p,$$

podobne pre úlohu (\mathcal{PNR}) definujeme

$$\mathbf{y}_i^{k+1} = \mathbf{y}_i^k + r^k g_i^+(\mathbf{x}^k, \mathbf{y}_i^k), \quad i = 1, \dots, m. \quad (2.38)$$

Kedže máme (2.35), môžeme pre všetky $i = 1, \dots, m$ písať

$$\mathbf{y}_i^{k+1} = \mathbf{y}_i^k + r^k \max \left[-g_i(\mathbf{x}^k), -\frac{\mathbf{y}_i^k}{r} \right],$$

a nakoniec

$$\mathbf{y}_i^{k+1} = \max [0, \mathbf{y}_i^k - r^k g_i(\mathbf{x}^k)]. \quad (2.39)$$

Iterácia (2.39) je analógom k (2.5) pre úlohu (\mathcal{PNR}) . Výraz (2.39) zároveň zaručí nezápornosť \mathbf{y}^k pre ľubovoľné $k > 1$.

Schematicky potom môžeme popísať tzv. Rockafellarov algoritmus takto:

Algoritmus (Rockafellar). *Nech je daný rozšírený Lagrangeov multiplikátor \mathbf{y}^k a penalizačný parameter r^k . Vektor \mathbf{x}^k získame minimalizáciou funkcie $\mathcal{L}_{\mathbf{R}}(\cdot, \mathbf{y}^k)$ na priestore \mathbb{R}^n . Ďalej položíme*

$$\mathbf{y}_i^{k+1} = \max [0, \mathbf{y}_i^k - r^k g_i(\mathbf{x}^k)], \quad i = 1, \dots, m,$$

zvolíme nový penalizačný parameter $r^{k+1} \geq r^k$ a proces zopakujeme.

Tento algoritmus býva často nazývaný aj Powell-Hestenes-Rockafellarov [4].

2.2.1 Odvodenie odhadu druhého rádu

Na tomto mieste odvodíme rovnako ako pre úlohu (\mathcal{PR}) iteráciu druhého rádu k úlohe (\mathcal{PNR}) . K tomuto odvodeniu použijeme duálnu funkciu pre problém (\mathcal{PNR}) . Budeme postupovať rovnako ako v časti (2.1.1) (pre details pozri [1], [36]). Za platnosti predpokladu 2.2 definujeme duálnu funkciu

$$d_{\mathbb{R}}(\mathbf{y}) = \min_{\|\mathbf{x} - \hat{\mathbf{x}}\| < \varepsilon} \mathcal{L}_{\mathbb{R}}(\mathbf{x}, \mathbf{y}). \quad (2.40)$$

pre všetky (\mathbf{y}, r) patriace do množiny

$$\mathbb{D} = \{(\mathbf{y}, r) \mid \|\mathbf{y} - \hat{\mathbf{y}}\| < \delta r, r \geq \bar{r}\}, \quad (2.41)$$

kde δ, ε , a \bar{r} sú rovnaké ako vo Vete (2.2) aplikovanej na úlohu (2.31). Potom pre všetky $(\mathbf{y}, r) \in \mathbb{D}$ máme

$$\nabla_{\mathbf{y}} d_{\mathbb{R}}(\mathbf{y}) = g^+(\mathbf{x}(\mathbf{y}), \mathbf{y}), \quad (2.42)$$

kde $\mathbf{x}(\mathbf{y})$ rieši (2.40). Ďalej odvodíme Hessovu maticu funkcie $d_{\mathbb{R}}$. Ak $\mathbf{x}(\mathbf{y})$ patrí do množiny $\hat{\mathbb{S}}_{\mathbf{y}, r}$, kde

$$\hat{\mathbb{S}}_{\mathbf{y}, r} = \left\{ \mathbf{x} \mid g_i(\mathbf{x}) \neq \frac{\mathbf{y}_i}{r}, i = 1, \dots, m \right\},$$

potom je $d_{\mathbb{R}}$ dvakrát spojito diferencovateľná na nejakom okolí (\mathbf{y}, r) . Ak rozpíšeme výraz (2.42) po zložkách, dostaneme

$$\frac{\partial d_{\mathbb{R}}(\mathbf{y})}{\partial \mathbf{y}_i} = \max \left[-g_i(\mathbf{x}(\mathbf{y}_i)), -\frac{\mathbf{y}_i}{r} \right], \quad i = 1, \dots, m,$$

potom ľahko spočítame aj druhé parciálne derivácie funkcie $d_{\mathbb{R}}$. Pre všetky indexy i také, že $g_i(\mathbf{x}(\mathbf{y})) \geq \frac{\mathbf{y}_i}{r}$, máme

$$\frac{\partial^2 d_{\mathbb{R}}(\mathbf{y})}{\partial \mathbf{y}_i \partial \mathbf{y}_j} = 0, \quad i \neq j, \quad (2.43)$$

$$\frac{\partial^2 d_{\mathbb{R}}(\mathbf{y})}{\partial \mathbf{y}_i^2} = -\frac{1}{r}, \quad (2.44)$$

a pre zvyšné indexy i počítame druhé derivácie rovnako ako v (2.1.1), pričom považujeme $g_i(\mathbf{x}(\mathbf{y}))$ za rovnosti. Pre zjednodušenie sumárneho zápisu na chvíľu predpokladajme, že existuje index q taký, že platí $g_i(\mathbf{x}(\mathbf{y})) \geq \frac{\mathbf{y}_i}{r}$ pre $i = 1, \dots, q$ a

$g_i(\mathbf{x}(\mathbf{y})) < \frac{\mathbf{y}_i}{r}$ pre $i = q + 1, \dots, m$, a označme $g_{(m-q)}(\mathbf{x}) = (g_{q+1}(\mathbf{x}), \dots, g_m(\mathbf{x}))^\top$.
Potom

$$\nabla_{yy}^2 d_R = \begin{bmatrix} -\frac{1}{r}\mathbf{I}_q & 0 \\ 0 & \mathbf{H}(\mathbf{x}(\mathbf{y}), \mathbf{y}) \end{bmatrix}, \quad (2.45)$$

kde \mathbf{I}_q je $q \times q$ identická matica, a $\mathbf{H}(\mathbf{x}(\mathbf{y}), \mathbf{y})$ je matica rozmeru $(m - q) \times (m - q)$, pričom

$$\mathbf{H}(\mathbf{x}(\mathbf{y}), \mathbf{y}) = -\nabla g_{(m-q)}(\mathbf{x}(\mathbf{y})) \left[\nabla_{xx}^2 \mathcal{L}_R(\mathbf{x}(\mathbf{y}), \mathbf{y}) \right]^{-1} \nabla g_{(m-q)}(\mathbf{x}(\mathbf{y}))^\top,$$

Rovnako možno písať

$$\nabla_{yy}^2 d_R = - \left[\nabla g(\mathbf{x}(\mathbf{y})) \mathbf{D}_1 \left[\nabla_{xx}^2 \mathcal{L}_R(\mathbf{x}(\mathbf{y}), \mathbf{y}) \right]^{-1} \mathbf{D}_1 \nabla g(\mathbf{x}(\mathbf{y}))^\top - \mathbf{D}_2 \right], \quad (2.46)$$

pre

$$\mathbf{D}_1 = \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{I}_{m-q} \end{bmatrix}, \quad \mathbf{D}_2 = \begin{bmatrix} \frac{1}{r}\mathbf{I}_q & 0 \\ 0 & 0 \end{bmatrix},$$

pričom \mathbf{I}_q a \mathbf{I}_{m-q} sú identické matice príslušných rozmerov. Následná iterácia má potom tvar

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \left[\nabla_{yy}^2 d_R(\mathbf{y}^k) \right]^{-1} \nabla d_R(\mathbf{y}^k). \quad (2.47)$$

S využitím (2.42) a (2.45), resp. (2.46) vidíme, že

$$\mathbf{y}_i^{k+1} = 0 \quad \text{pre } i = 1, \dots, q, \quad (2.47a)$$

a

$$\mathbf{y}_{(m-q)}^{k+1} = \mathbf{y}_{(m-q)}^k - \left[\nabla g_{(m-q)}(\mathbf{x}^k) \left[\nabla_{xx}^2 \mathcal{L}_R(\mathbf{x}^k, \mathbf{y}^k) \right]^{-1} \nabla g_{(m-q)}(\mathbf{x}^k)^\top \right]^{-1} g_{(m-q)}(\mathbf{x}^k), \quad (2.47b)$$

kde sme označili

$$\mathbf{y}_{(m-q)} = \begin{bmatrix} \mathbf{y}_{q+1} \\ \vdots \\ \mathbf{y}_m \end{bmatrix}.$$

Ak označíme

$$N_k = \left\{ i \mid g_i(\mathbf{x}(\mathbf{y}^k)) \geq \frac{\mathbf{y}_i}{r} \right\},$$

čiže N_k je množina indexov tých ohraničení, o ktorých predpokladáme, že budú neaktívne v $\hat{\mathbf{x}}$, potom iteráciu (2.47) môžeme s pomocou (2.47a) a (2.47b) opísať takto: Všetky Lagrangeove multiplikátory, ktoré zodpovedajú nerovniciam o ktorých

predpokladáme, že budú neaktívne v bode optima $\hat{\mathbf{x}}$, položíme rovné 0 (t.j. $\mathbf{y}_i^{k+1} = 0 \forall i \in N_k$), a zvyšné ohraňovania budeme považovať za rovnice a na základe tohto predpokladu odhadneme k nim prislúchajúce multiplikátory \mathbf{y}_i^{k+1} , $i \notin N_k$.

Niekoľko poznámok k metóde druhého rádu:

- (a) Vo všeobecnosti sa môže stať, že niektoré \mathbf{y}_i získané vzťahom (2.47) budú záporné, no z KKT podmienok vieme, že $\hat{\mathbf{y}} \geq 0$. Preto sa zdá byť rozumné použiť namiesto (2.47) upravenú iteráciu, ktorú definujeme ako

$$\bar{\mathbf{y}}^{k+1} = \mathbf{y}^k - \left[\nabla_{yy}^2 d_R(\mathbf{y}^k) \right]^{-1} \nabla d_R(\mathbf{y}^k)$$

a

$$\mathbf{y}^{k+1} = \begin{bmatrix} \max[0, \bar{\mathbf{y}}_1^{k+1}] \\ \vdots \\ \max[0, \bar{\mathbf{y}}_m^{k+1}] \end{bmatrix}, \quad (2.48)$$

čiže všetky záporné multiplikátory \mathbf{y}_i^{k+1} položíme rovné 0. Dá sa ukázať, že platí

$$\|\mathbf{y}^{k+1} - \hat{\mathbf{y}}\| \leq \|\bar{\mathbf{y}}^{k+1} - \hat{\mathbf{y}}\|,$$

a taktiež pre ľubovoľné (\mathbf{x}, \mathbf{y}) máme

$$\mathcal{L}_R(\mathbf{x}, \mathbf{y}) \leq \mathcal{L}_R(\mathbf{x}, \mathbf{y}^{k+1}),$$

z čoho plynie

$$d_R(\bar{\mathbf{y}}^{k+1}) \leq d_R(\mathbf{y}^{k+1}).$$

Vektor \mathbf{y}^{k+1} je teda bližšie k riešeniu $\hat{\mathbf{y}}$ ako $\bar{\mathbf{y}}^{k+1}$, a zároveň hodnotu duálnej funkcie nemôžeme zmenšiť, ak vymeníme $\bar{\mathbf{y}}^{k+1}$ za \mathbf{y}^{k+1} . Tento poznatok je dôležitý pri dôkazoch konvergenčných viet, ako je Veta (2.3).

- (b) Podobne ako v prípade úlohy (\mathcal{PR}) môžeme dať iteráciu (2.47) do súvisu s riešením primárno-duálnej sústavy Newtonovou metódou. Vzťah (2.47) zodpovedá prípadu, keď je minimalizácia funkcie $\mathcal{L}_R(\cdot, \mathbf{y})$ presná. Rovnako ako v časti 2.1.1 môžeme definovať

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \left[\nabla_{yy} d_R(\mathbf{y}^k) \right]^{-1} \times \left[\nabla_y d_R(\mathbf{y}^k) - \nabla g(\mathbf{x}^k) \left[\nabla_{xx}^2 \mathcal{L}_R(\mathbf{x}^k, \mathbf{y}^k) \right]^{-1} \nabla_x \mathcal{L}_R(\mathbf{x}^k, \mathbf{y}^k) \right]. \quad (2.49)$$

zodpovedajúcu neexaktnej minimalizácii funkcie $\mathcal{L}_R(\cdot, \mathbf{y})$. Takisto možno pozorovať lepšie praktické výsledky oproti iterácii (2.47)

- (c) Výsledky odvodené pre iteráciu druhého rádu v časti 2.1.1 platia aj pre iterácie (2.39) a (2.47). Na zreteli musíme mať najmä globálnu konvergenciu iterácie (2.39) a iba lokálnu konvergenciu (2.47) pre úlohy konvexného programovania.

Podarilo sa nám odvodiť kvadratické penalizačné funkcie pre úlohy (\mathcal{PR}) a (\mathcal{PNR}). V nasledujúcom texte poukážeme na niektoré teoretické nedostatky týchto metód a definujeme triedy všeobecných penalizačných funkcií.

2.3 Všeobecná penalizačná funkcia

Doposiaľ uvažované kvadratické penalizačné funkcie a k nim pridružené rozšírené Lagrangeove funkcie sú najrozšírenejšie pre túto triedu algoritmov. V istých špeciálnych prípadoch však môže byť žiaduce použiť inú ako kvadratickú penalizačnú funkciu. Tieto dôvody môžu byť napríklad takéto:

- (a) Existujú jednoduché príklady úloh (nekonvexného programovania), v ktorých je síce účelová funkcia na množine prípustných riešení zdola ohraničená, no príslušná rozšírená Lagrangeova funkcia túto vlastnosť nemá (na \mathbb{R}^n). Ako príklad uvažujme jednoduchú úlohu jednej premennej [1]

$$\begin{aligned} \text{minimalizovať} \quad & -x^{2\rho}, \quad x \in \mathbb{R} \\ \text{za podmienky} \quad & x = 0, \end{aligned} \tag{2.50}$$

pričom $\rho \in \mathbb{N}$. Hestenesova funkcia má potom tvar

$$\mathcal{L}_{\mathbb{R}}(x, z) = -x^{2\rho} + xz + \frac{r}{2}x^2$$

a vidíme, že je zdola neohraničená pre ľubovoľné r , ak $\rho \geq 2$. Potom voľná minimalizácia tejto funkcie môže zlyhať s výnimkou, ak je štartovací bod dostatočne blízko bodu lokálneho minima Powell-Hestenesovej funkcie. Tento nedostatok sa však dá napraviť voľbou inej penalizačnej funkcie (ich konkrétne tvary budeme uvažovať neskôr).

- (b) K úlohe (\mathcal{PNR}) sme jej transformáciou na úlohu (\mathcal{PR}) a následnou úpravou odvodili Rockafellarovu rozšírenú Lagrangeovu funkciu. Tento postup však spôsobuje, že druhá derivácia funkcie $\mathcal{L}_{\mathbb{R}}(\cdot, \mathbf{y})$ je nespojitá pre tie \mathbf{x} , ktoré spĺňajú $g_i(\mathbf{x}) = \frac{\mathbf{y}_i}{r}, i = 1, \dots, m$. Na druhej strane, minimalizačné metódy, ktoré prichádzajú do úvahy, vyžadujú spojitost' druhej derivácie na garanciu konverencie.

V praktickej implementácii a riešení problémov sa však ukazuje, že pri použití Newtonovej, prípadne kvázinewtonovských metód tento nedostatok nemá negatívny vplyv na správanie sa algoritmu. Je tu však teoretická možnosť zlyhania a preto je potrebné sa ňou zaoberať. V častiach, ktoré budú nasledovať, odvodíme triedu penalizačných, dvakrát spojitou diferencovateľných funkcií.

- (c) Otázka rýchlosti konvergenzie je takisto dôležitá. Táto rýchlosť sa môže drasticky meniť v závislosti od použitej penalizačnej funkcie.

Všetky funkcie, ktoré budeme spomínať, budú spĺňať axiomy Roodeho vety 2.1, a takisto bude platiť, podobne ako v prípade funkcií Powell-Hestenesa a Rockafellara, rovnosť medzi klasickým a zovšeobecným Lagrangeovým multiplikátorom v optimálnom bode ($\hat{\mathbf{z}} = \hat{\mathbf{v}}$ pre úlohu (\mathcal{PR}) a $\hat{\mathbf{y}} = \hat{\mathbf{u}}$ pre úlohu (\mathcal{PNR})).

Kvôli jednoduchosti začneme úlohou (\mathcal{PR}) a definujeme veľmi všeobecné podmienky na tvar penalizačných funkcií.

2.3.1 Penalizačné funkcie pre úlohu (\mathcal{PR})

K úlohe (\mathcal{PR}) definujeme triedu penalizačných funkcií takto:

Trieda funkcií $\mathcal{P}_{\mathbf{EQ}}$: Do tejto triedy zaradíme všetky funkcie $\phi : \mathbb{R} \rightarrow \mathbb{R}$ spĺňajúce tieto podmienky:

- (i) $\phi(\xi)$ je spojitou diferencovateľná a ostro konvexná v \mathbb{R} ,
- (ii) $\phi(0) = 0$, $\phi'(0) = 0$,
- (iii) $\lim_{\xi \rightarrow \infty} \phi'(\xi) = \infty$, $\lim_{\xi \rightarrow -\infty} \phi'(\xi) = -\infty$.

K danej penalizačnej funkcii ϕ z triedy funkcií $\mathcal{P}_{\mathbf{EQ}}$ priradíme rozšírenú Lagrangeovu funkciu $\mathcal{L}_{\mathbf{EQ}}(\mathbf{x}, \mathbf{z})$ predpisom

$$\mathcal{L}_{\mathbf{EQ}}(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}) + h(\mathbf{x})^T \mathbf{z} + \frac{1}{r} \sum_{j=1}^p \phi(rh_j(\mathbf{x})). \quad (2.51)$$

Metóda multiplikátorov potom pozostáva z voľnej minimalizácie funkcie $\mathcal{L}_{\mathbf{EQ}}(\cdot, \mathbf{z}^k)$ na \mathbb{R}^n , ktorou získame vektor \mathbf{x}^k . Po tejto minimalizácii nasleduje iterácia multiplikátorov (v tomto prípade prvého rádu)

$$\mathbf{z}_j^{k+1} = \mathbf{z}_j^k + \phi'(rh_j(\mathbf{x}^k)), \quad j = 1, \dots, p. \quad (2.52)$$

Je možné odvodiť iteráciu multiplikátorov druhého rádu úplne analogickým postupom ako v časti 2.1.1, jej tvar bude rovnaký ako v (2.22) so zámenou $\mathcal{L}_{\mathbf{PH}}(\cdot, \cdot) \rightarrow$

$\mathcal{L}_{\text{EQ}}(\cdot, \cdot)$. To isté platí aj pre definovanie iterácie v tvare (2.29), ktorá zodpovedá iterácii druhého rádu po neúplnej minimalizácii.

Zostáva ešte spomenúť príklady niektorých funkcií patriacich do triedy funkcií \mathcal{P}_{EQ} . Máme napríklad:

(a) $\phi(\xi) = \frac{1}{2}\xi^2$,

(b) $\phi(\xi) = \frac{1}{\nu}|\xi|^\nu$, $\nu > 1$,

(c) $\phi(\xi) = \frac{1}{\nu}|\xi|^\nu + \frac{1}{2}\xi^2$, $\nu \in (1, 2)$,

(d) $\phi(\xi) = Q(\xi)$, Q je polynóm stupňa 2ν , $\nu \in \mathbb{N}$, spĺňajúci $Q(0) = 0$ a $Q'(0) = 0$.

Môžeme si všimnúť, že pri použití (a) dostaneme Powell-Hestenesovu rozšírenú Lagrangeovu funkciu. Ak použijeme na definovanie rozšírenej Lagrangeovej funkcie funkciu (b), prípadne (d) pre ν dostatočne veľké, potom ju môžeme s úspechom aplikovať na riešenie problematickej úlohy (2.50). V literatúre sa často môžeme stretnúť aj s exponenciálnymi penalizačnými funkciami, v ich prípade si však treba dať pozor pri implementácii. Veľmi ľahko môže dôjsť k pretečeniu vo floating-point aritmetike, preto sa pristupuje k ich extrapolácii polynomiálnymi funkciami.

Takisto hodnota parametra ν funkcie (b) a (d) nesmie byť príliš veľká. Zvýšená rýchlosť konvergencie je v tomto prípade kompenzovaná zhoršujúcou sa podmienenosťou Hesseovej matice rozšírenej Lagrangeovej funkcie, čo môže mať negatívny vplyv na proces jej minimalizácie. Funkcia (c) dosahuje superlineárnu rýchlosť konvergencie, no vykazuje nespojitosť druhej derivácie. Existujú príklady úloh, v ktorých však dosahuje lepšie výsledky ako Powell-Hestenesova funkcia (a).

2.3.2 Penalizačné funkcie pre úlohu (\mathcal{PNR})

V prípade úlohy (\mathcal{PNR}) uvedieme jednu veľkú triedu funkcií, z ktorej vyberieme dve rôzne podskupiny funkcií, ktoré sa často vyskytujú a používajú v literatúre [4].

Trieda funkcií \mathcal{P}_{NEQ} : Do tejto triedy zaradíme všetky funkcie $P : \mathbb{R}^2 \rightarrow \mathbb{R}$ spĺňajúce tieto podmienky (použijeme označenie $P'(\xi, \eta) = \frac{\partial}{\partial \xi} P(\xi, \eta)$):

- (i) $P(\xi, \eta)$ je spojitá na $\mathbb{R} \times [0, \infty)$ a spojitou diferencovateľná na $\mathbb{R} \times [0, \infty)$, $P(\cdot, 0)$ je spojitou diferencovateľná v premennej ξ ,
- (ii) $P(\xi, \cdot)$ je konkávna na $[0, \infty]$ pre každé pevné ξ ,
- (iii) Pre každé $\eta \geq 0$ je $P(\cdot, \eta)$ konvexná v ξ na \mathbb{R} ,
- (iv) $P(0, \eta) = 0$ pre všetky $\eta \geq 0$,

- (v) $P'(0, \eta) = -\eta$ pre všetky $\eta \geq 0$,
- (vi) $\lim_{\xi \rightarrow \infty} P'(\xi, \eta) = 0$ pre všetky $\eta \geq 0$,
- (vii) $\lim_{\xi \rightarrow -\infty} P'(\xi, \eta) = -\infty$ pre všetky $\eta \geq 0$,
- (viii) $\inf_{\xi \in \mathbb{R}} P(\xi, \eta) > -\infty$ pre všetky $\eta \geq 0$.

Premennú ξ možno interpretovať ako výraz $rg_i(\mathbf{x})$, premennú η zasa považujeme za multiplikátor \mathbf{y}_i , pre nejaké i . Funkcia $P(\xi, \eta)$ prechádza počiatkom a rastie do nekonečna pre $\xi \rightarrow -\infty$ (nepripustná oblasť, keďže $g_i(\mathbf{x}) < 0$), naviac sklon tejto funkcie rastie bez obmedzenia. Pre $\xi \rightarrow \infty$ funkcia postupne klesá k svojmu infimu (ktoré je konečné, a z konvexnosti funkcie vyplýva, že aj menšie alebo rovné 0). Multiplikátor η mení sklon funkcie P pri prechode počiatkom, kde je tento sklon rovný $-\eta$ (dôležité pre rovnosť medzi klasickým a rozšíreným vektorom Lagrangeových multiplikátorov v optimálnom bode).

Ku každej funkcii P priradíme rozšírenú Lagrangeovu funkciu predpisom

$$\mathcal{L}_{\text{NEQ}}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \frac{1}{r} \sum_{i=1}^m P(rg_i(\mathbf{x}), \mathbf{y}_i). \quad (2.53)$$

Klasické metódy multiplikátorov môžeme prirodzene rozšíriť na použitie funkcie (2.53). Metóda prvého rádu teda bude zodpovedať voľnej minimalizácii funkcie $\mathcal{L}_{\text{NEQ}}(\cdot, \mathbf{y}^k)$ na \mathbb{R}^n , ktorou získame vektor \mathbf{x}^k . Po tejto minimalizácii nasleduje iterácia multiplikátorov prvého rádu

$$\mathbf{y}_j^{k+1} = -P'(rg_j(\mathbf{x}^k), \mathbf{y}_j), \quad j = 1, \dots, m. \quad (2.54)$$

Ak štartovací Lagrangeov multiplikátor spĺňa $\mathbf{y}^0 \geq 0$, potom z vlastností (iii), (vi), (vi) funkcie P máme, že $\mathbf{y}^k \geq 0$ pre všetky k .

Keďže máme

$$\nabla_{\mathbf{x}} \mathcal{L}_{\text{NEQ}}(\mathbf{x}, \mathbf{y}) = \nabla f(\mathbf{x}) - \sum_{i=1}^m P'(rg_i(\mathbf{x}), \mathbf{y}_i) \nabla g_i(\mathbf{x}), \quad (2.55)$$

tak z (2.54) vidíme, že platí

$$\nabla_{\mathbf{x}} \mathcal{L}_{\text{NEQ}}(\mathbf{x}^k, \mathbf{y}^k) = \nabla_{\mathbf{x}} L(\mathbf{x}^k, \mathbf{y}^{k+1}),$$

kde $L(\mathbf{x}, \mathbf{y})$ je klasická Lagrangeova funkcia definovaná vzťahom (2.30).

Prvá z podtried funkcií, ktorú budeme uvažovať pre problém (\mathcal{PNR}) , je transformovanou triedou \mathcal{PEQ} . Získame ju rovnakým postupom, akým sme postupovali

pri odvodení Rockafellarovej funkcie z Powell-Hestenesovej funkcie v časti 2.2, čiže zavedením doplnkových premenných $\xi_i, i = 1, \dots, m$, transformáciou problému a následnou minimalizáciou príslušnej rozšírenej Lagrangeovej funkcie (2.51) priradenej k penalizačnej funkcii z triedy $\mathcal{P}_{\mathbf{EQ}}$. Dostaneme tak odvodenú triedu funkcií $\mathcal{P}_{\mathbf{NEQ}}$ pre úlohu (\mathcal{PNR}) .

Trieda funkcií $\mathcal{P}_{\mathbf{NEQ}}^1$: Do tejto triedy funkcií patria všetky funkcie $P : \mathbb{R}^2 \rightarrow \mathbb{R}$ definované vzťahom

$$P(\xi, \eta) = \begin{cases} \phi(\xi) - \xi\eta & \text{pre } \phi'(\xi) < \eta \\ \min_{\theta \in \mathbb{R}} \{ \phi(\theta) - \theta\eta \} & \text{pre } \phi'(\xi) \geq \eta \end{cases}, \quad (2.56)$$

kde ϕ je penalizačná funkcia z triedy $\mathcal{P}_{\mathbf{EQ}}$ pre úlohu (\mathcal{PR}) . Existencia minima je zabezpečená vlastnosťami **(i)** a **(iii)** funkcie ϕ . Trieda $\mathcal{P}_{\mathbf{NEQ}}^1$ teda zodpovedá metóde multiplikátorov pre úlohu (\mathcal{PNR}) po tom, čo ju pretransformujeme na úlohu (\mathcal{PR}) .

Použitím (2.56) dostaneme

$$\frac{\partial}{\partial \xi} P(\xi, \eta) = P'(\xi, \eta) = \begin{cases} \phi'(\xi) - \eta & \text{pre } \phi'(\xi) < \eta \\ 0 & \text{pre } \phi'(\xi) \geq \eta \end{cases},$$

a teda iterácia multiplikátorov (2.54) má pre triedu funkcií (2.56) tvar

$$\mathbf{y}_i^{k+1} = -P'(rg_i(\mathbf{x}^k), \mathbf{y}_i^k) = \max [0, \mathbf{y}_i^k - \phi'(rg_i(\mathbf{x}^k))].$$

Vezmime napríklad $\phi(\xi) = \frac{1}{2}\xi^2$, čiže klasickú kvadratickú penalizačnú funkciu patriacu do triedy $\mathcal{P}_{\mathbf{EQ}}$. Podľa (2.56) máme

$$P_{\mathbf{R}}(\xi, \eta) = \begin{cases} \frac{1}{2}\xi^2 - \xi\eta & \text{pre } \xi < \eta \\ -\frac{\eta^2}{2} & \text{pre } \xi \geq \eta \end{cases},$$

a podľa (2.53) dostávame

$$\begin{aligned} \mathcal{L}_{\mathbf{R}}(\mathbf{x}, \mathbf{y}) &= f(\mathbf{x}) + \frac{1}{r} \sum_{i=1}^m P_{\mathbf{R}}(rg_i(\mathbf{x}), \mathbf{y}_i) \\ &= f(\mathbf{x}) + \frac{1}{2r} \sum_{i=1}^m \left[\max^2(0, \mathbf{y}_i - rg_i(\mathbf{x})) - \mathbf{y}_i^2 \right], \end{aligned}$$

čiže Rockafellarovu rozšírenú Lagrangeovu funkciu (2.37). Rovnako pre iteráciu multiplikátorov prvého rádu máme

$$\mathbf{y}_i^{k+1} = \max [0, \mathbf{y}_i^k - rg_i(\mathbf{x}^k)].$$

Rovnakým postupom získame ďalšie funkcie P pomocou ľubovoľnej funkcie $\phi \in \mathcal{P}_{\mathbf{EQ}}$, pre ktorú vieme nájsť minimum vo výraze (2.56) analyticky. Takisto je možné odvodiť iteráciu multiplikátorov druhého rádu opakovaním postupu z časti 2.2.1.

Keďže odvodenie celej triedy funkcií zodpovedalo postupu odvodenia Rockafellarovej penalizačnej funkcie z klasickej kvadratickej penalizačnej funkcie Powell-Hestenesa, metódy používajúce penalizačné funkcie z triedy $\mathcal{P}_{\mathbf{NEQ}}^1$ budú mať podobné vlastnosti ako sú uvedené v časti 2.2. Zároveň však zostane v platnosti aj nespojitosť druhých derivácií v bodoch, kde

$$\phi'(rg_i(\mathbf{x})) = \mathbf{y}_i, \quad i = 1, \dots, m.$$

Sformulujeme preto ešte jednu podtriedu triedy $\mathcal{P}_{\mathbf{NEQ}}$, ktorá bude zostrojená špeciálne pre úlohu (\mathcal{PNR}) , ktorá je v literatúre veľmi často spomínaná. Funkcie z tejto triedy budú mať spojité druhé derivácie.

Trieda funkcií $\mathcal{P}_{\mathbf{NEQ}}^2$: Do tejto triedy patria všetky funkcie $P : \mathbb{R}^2 \rightarrow \mathbb{R}$ tvaru

$$P(\xi, \eta) = \eta\psi(\xi),$$

kde $\psi : \mathbb{R} \rightarrow \mathbb{R}$ spĺňa nasledujúce požiadavky:

- (i) $\psi \in C^2$, ψ je ostro konvexná na \mathbb{R} ,
- (ii) $\psi(0) = 0$, $\psi'(0) = -1$,
- (iii) $\lim_{\xi \rightarrow -\infty} \psi(\xi) = \infty$, $\lim_{\xi \rightarrow \infty} \psi(\xi) > -\infty$,
- (iv) $\lim_{\xi \rightarrow -\infty} \psi'(\xi) = -\infty$, $\lim_{\xi \rightarrow \infty} \psi'(\xi) = 0$,

Penalizačné funkcie z tejto triedy sú lineárne v multiplikátore \mathbf{y} , zároveň vidíme, že z vlastnosti (i) vyplýva, že P má spojité druhé derivácie. Vlastnosť $\psi'(0) = -1$ zabezpečí rovnosť medzi klasickým a rozšíreným optimálnym vektorom Lagrangeových multiplikátorov.

Typickým zástupcom funkcií z tejto triedy je exponenciálna penalizačná funkcia

$$\psi(\xi) = e^{-\xi} - 1,$$

konkrétnou tvorbou ďalších funkcií patriacich do $\mathcal{P}_{\mathbf{NEQ}}^2$ sa budeme zaoberať v ďalšej kapitole.

Rozšírená Lagrangeova funkcia prislúchajúca k triede penalizačných funkcií $\mathcal{P}_{\mathbf{NEQ}}^2$

je daná ako

$$\begin{aligned}\mathcal{L}_{\text{NEQ}}(\mathbf{x}, \mathbf{y}) &= f(\mathbf{x}) + \frac{1}{r} \sum_{i=1}^m P(r g_i(\mathbf{x}), \mathbf{y}_i) \\ &= f(\mathbf{x}) + \frac{1}{r} \sum_{i=1}^m \mathbf{y}_i \psi(r g_i(\mathbf{x})),\end{aligned}\tag{2.57}$$

a iterácia multiplikátorov prvého rádu je daná vzťahom

$$\mathbf{y}_i^{k+1} = -\mathbf{y}_i^k \psi'(r g_i(\mathbf{x}^k)), \quad i = 1, \dots, m.\tag{2.58}$$

Metódy rozšírených Lagrangeových funkcií používajúce funkciu (2.57) sa niekedy nazývajú metódami *nelineárneho škálovania* a parameter r škálovacím parametrom ([34]).

Je nutné poznamenať, že v prípade funkcií z triedy $\mathcal{P}_{\text{NEQ}}^2$ je nevyhnutné, aby $\mathbf{y}^0 > 0$. Potom na základe (2.58) a vlastností (i), (ii) a (iii) funkcie ψ platí $\mathbf{y}^k > 0$ pre všetky k . Pre úlohu konvexného programovania je možné pomocou teórie Fenchelovej duality ukázať, že bod \mathbf{y}^{k+1} definovaný pomocou (2.58) maximalizuje penalizovanú duálnu funkciu úlohy (\mathcal{PNR}) odvodenú pomocou funkcie (2.57). Odhady multiplikátorov druhého rádu pre túto funkciu nie sú známe.

Napriek tomu, že teoretické výsledky pre funkcie z triedy $\mathcal{P}_{\text{NEQ}}^2$ nie sú také silné ako v prípade triedy $\mathcal{P}_{\text{NEQ}}^1$, ich algoritmické využitie takmer identické. Na druhej strane odstraňujú nedostatok funkcií z triedy $\mathcal{P}_{\text{NEQ}}^1$ v podobe nespojitosti druhej derivácie, a aj preto môžu byť použité ako súčasť metód, pre ktoré je táto vlastnosť nevyhnutná. Využijeme ich aj v nasledujúcej kapitole, v ktorej predstavíme algoritmus R. Polyaka a I. Grivu.

Kapitola 3

Nelineárne škálovanie a Fischerova metóda

V predošlej kapitole sme sa venovali teoretickému popisu klasických metód rozšírených Lagrangeových funkcií. Každý krok týchto metód spočíval v minimalizácii rozšírenej Lagrangeovej funkcie na priestore primárnej premennej \mathbf{x} , po ktorej nasledoval odhad vektora zovšeobecnených Lagrangeových multiplikátorov \mathbf{y} . Penalizačný parameter r môže byť zafixovaný alebo zmenený na konci iterácie. Práve možnosť fixovania parametra r často zabraňuje zhoršovaniu podmienenosti Hessovej matice minimalizovanej funkcie. Aktívna prítomnosť Lagrangeových multiplikátorov je pre konvergenciu pri fixovanom r rozhodujúca. Ak sú splnené podmienky optimality druhého rádu (podmienky predpokladu 2.2 z časti 2.2), metódy konvergujú lineárne k (lokálnemu) optimálnemu riešeniu pri fixovanom, no dostatočne veľkom penalizačnom parametri $r \geq \bar{r}$ (veta 2.2).

Za účelom zlepšenia rýchlosti konverencie je nutné zvyšovať hodnotu penalizačného parametra po každej iterácii, čo môže viesť k výrazným numerickým ťažkostiam. Minimalizácia rozšírenej Lagrangeovej funkcie sa tak stáva náročnejšou. Tento problém študovali R. Polyak a I. Griva a publikovali v sérii článkov ([29], [30], [31], [32], [33], [34]), ktorých výsledkom bolo popísanie tzv. primárno-duálnej, nelineárne škálovacej metódy. Táto nahradzuje voľnú minimalizáciu rozšírenej Lagrangeovej funkcie a následný odhad multiplikátorov riešením tzv. primárno-duálnej nelineárnej sústavy rovníc, čo umožňuje neobmedzené zvyšovanie penalizačného parametra bez narušenia numerickej stability. V prípade špeciálnej voľby zmeny penalizačného parametra

je možné dosiahnuť superlineárnu rýchlosť konvergencie.

Keďže na riešenie spomínaného primárno-duálneho systému rovníc použijeme Newtonovu metódu, musí byť štartovací bod dostatočne blízko optimálnemu primárno-duálnemu riešeniu $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$. Preto výsledný algoritmus bude špeciálnym spojením klasického algoritmu rozšírených Lagrangeových funkcií a riešenia primárno-duálnej sústavy, kde využijeme „to najlepšie“ z oboch uvedených metód.

Táto kapitola je organizovaná nasledovne: Najprv sformulujeme základné predpoklady a zopakujeme metódu nelineárneho škálovania, potom odvodíme tzv. primárno-duálnu nelineárne škálovaciu metódu s dynamickou zmenou penalizačného parametra (PDNRD metóda, [30], [33]), a poukážeme na možné zlepšenia uvedeného prístupu (PDEPM metóda, [32]). Na záver ukážeme odlišný spôsob formulácie primárno-duálnej sústavy využitím tzv. Fischerovej funkcie [14], ktorá umožňuje prepísať KKT podmienky na systém rovníc [11], [12].

3.1 Metóda nelineárneho škálovania

Budeme sa zaoberať problémom konvexného programovania v tvare

$$\begin{aligned} & \text{minimalizovať} && f(\mathbf{x}) \\ & \text{za podmienok} && g(\mathbf{x}) \geq 0 \end{aligned} \quad (\mathcal{PNRC})$$

kde $f : \mathbb{R}^n \rightarrow \mathbb{R}$ je konvexná, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ konkávna funkcia, a predpokladajme, že $f, g \in C^2$. Metódy uvedené v tejto časti je možné odvodiť aj pre ohraničenia typu

$$h(\mathbf{x}) = 0, \quad h : \mathbb{R}^n \rightarrow \mathbb{R}^p,$$

no tieto pre zjednodušenie výkladu vynecháme (odvodenie spočíva v mechanickom zopakovaní postupov, ktoré neskôr uvedieme). Niektoré vlastnosti uvedených metód platia aj bez predpokladu konvexnosti.

Pre jednoduchosť zápisov označme

$$I_A(\hat{\mathbf{x}}) = \{i \mid g_i(\hat{\mathbf{x}}) = 0\} = \{1, \dots, q\},$$

kde $q \leq \min\{n, m\}$ a takisto použijeme označenie $g_{(q)}(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_q(\mathbf{x}))^T$ so štandardným predpokladom $\text{rank}(\nabla g_{(q)}(\hat{\mathbf{x}})) = q$. Lagrangeovu funkciu zapisujeme v tvare

$$\mathbf{L}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) - g(\mathbf{x})^T \mathbf{y}, \quad \mathbf{L} : \mathbb{R}^n \times \mathbb{R}_+^m \rightarrow \mathbb{R}. \quad (3.1)$$

V tejto kapitole budeme uvažovať, že platia nasledujúce predpoklady.

Predpoklad 3.1. Platí Slaterova podmienka, čiže existuje $\bar{\mathbf{x}}$ také, že $g_i(\bar{\mathbf{x}}) > 0, i = 1, \dots, m$.

Predpoklad 3.2. Vektor $\hat{\mathbf{x}}$ je ostrým globálnym minimom a regulárnym bodom problému (\mathcal{PNR}_C) , pričom platí

$$\boldsymbol{\omega}^T \mathbf{L}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \boldsymbol{\omega} > 0$$

pre všetky $\boldsymbol{\omega} \neq 0$ spĺňajúce $\nabla g_i(\hat{\mathbf{x}})^T \boldsymbol{\omega} = 0$ pre $i \in \{1, \dots, q\}$, kde $\mathbf{L}(\cdot, \cdot)$ je definovaná v (3.1). Navyiac, $\hat{\mathbf{y}}$ spĺňa podmienku ostrej komplementarity, čiže

$$\hat{\mathbf{y}}_i > 0 \quad \text{pre všetky } i \in I_A(\hat{\mathbf{x}}).$$

Uvažujme rozšírenú Lagrangeovu funkciu s predpisom

$$\mathcal{L}_{\text{NR}}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \frac{1}{r} \sum_{i=1}^m \mathbf{y}_i \psi(r g_i(\mathbf{x})), \quad (3.2)$$

kde transformačné funkcie ψ spĺňajú tieto podmienky:

- (i) $\psi'(\xi) < 0, \psi''(\xi) > 0, \psi \in C^2$,
- (ii) $\psi(0) = 0, \psi'(0) = -1$,
- (iii) $\psi(\xi) \geq \alpha \xi^2$, pre $\xi < 0$ a nejaké $\alpha > 0$,
- (iv) $-\psi'(\xi) \leq \frac{\beta}{\xi+1}, \psi''(\xi) \leq \frac{\gamma}{(\xi+1)^2}$, pre $\xi > 0$ a nejaké $\beta > 0, \gamma > 0$.

Funkcie ψ s uvedenými vlastnosťami tvoria podmnožinu funkcií patriacich do triedy $\mathcal{P}_{\text{NEQ}}^2$ definovanej v kapitole 2, keďže konkrétne špecifikujeme rýchlosť ich rastu pre $\xi < 0$, resp. rýchlosť poklesu pre $\xi > 0$. Medzi najčastejšie používané transformačné funkcie ψ patria

$$\text{(a)} \psi_1(\xi) = e^{-\xi} - 1, \quad \text{(b)} \psi_2(\xi) = \frac{1}{1+\xi} - 1, \quad \text{(c)} \psi_3(\xi) = -\ln(1+\xi),$$

no tieto funkcie ešte nie sú vhodné pre praktické použitie. Funkcie ψ_2 a ψ_3 sú definované iba pre $\xi \in (-1, \infty)$ a exponenciálny rast funkcie ψ_1 môže byť až príliš veľký, preto je možné pristúpiť ku kvadratickej extrapolácii. Pre ľubovoľné $\tau \in (-1, 0]$ definujeme

$$\psi_{p_i}(\xi) = \begin{cases} p_i(\xi) = a_i \xi^2 + b_i \xi + c_i & \xi \leq \tau, \\ \psi_i(\xi) & \xi > \tau. \end{cases} \quad (3.3)$$

Konštanty a_i, b_i, c_i polynómu $p_i(\xi)$ určíme podľa

$$\begin{aligned} a_i &= \frac{\psi_i''(\tau)}{2}, \\ b_i &= \psi_i'(\tau) - \tau\psi_i''(\tau), \\ c_i &= \psi_i(\tau) - \tau\psi_i'(\tau) + \frac{\tau^2\psi_i''(\tau)}{2}, \end{aligned}$$

keďže práve takáto voľba jednoznačne zabezpečí, že $\psi_{p_i} \in C^2$.

Algoritmus nelineárneho škálovania je potom klasickým zástupcom algoritmov rozšírených Lagrangeových funkcií.

Algoritmus (Nelineárne škálovanie). *Predpokladajme, že poznáme rozšírený Lagrangeov multiplikátor \mathbf{y}^k a penalizačný (škálovací) parameter r^k . Vektor \mathbf{x}^{k+1} získame minimalizáciou funkcie $\mathcal{L}_{\text{NR}}(\cdot, \mathbf{y}^k)$ na priestore \mathbb{R}^n . Potom odhadneme multiplikátory vzťahom*

$$\mathbf{y}_i^{k+1} = -\mathbf{y}_i^k \psi'(r^k g_i(\mathbf{x}^{k+1})), \quad i = 1, \dots, m, \quad (3.4)$$

zvolíme nový penalizačný (škálovací) parameter $r^{k+1} \geq r^k$ a proces zopakujeme.

Algoritmus je dobre definovaný vďaka vlastnostiam **(i)**, **(iii)** a **(iv)** funkcie ψ , za predpokladu konvexnosti úlohy (\mathcal{PNRC}) a predpokladu 3.1 (napr. [30], tvrdenie 1.).

Uvedený algoritmus vytvára postupnosť bodov $\mathbf{x}^k, \mathbf{y}^k$ podľa

$$\mathbf{x}^{k+1} : \nabla_{\mathbf{x}} \mathcal{L}_{\text{NR}}(\mathbf{x}^{k+1}, \mathbf{y}^k) = \nabla f(\mathbf{x}^{k+1}) + \sum_{i=1}^m \mathbf{y}_i^k \psi'(r^k g_i(\mathbf{x}^{k+1})) \nabla g_i(\mathbf{x}^{k+1}) = 0 \quad (3.5)$$

a

$$\mathbf{y}_i^{k+1} = -\mathbf{y}_i^k \psi'(r^k g_i(\mathbf{x}^{k+1})), \quad i = 1, \dots, m. \quad (3.6)$$

Ak označíme

$$\Psi'(r^k g(\mathbf{x}^{k+1})) = \text{diag} [\psi'(r^k g_i(\mathbf{x}^{k+1}))]_{i=1}^m = \begin{bmatrix} \psi'(r^k g_1(\mathbf{x}^{k+1})) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \psi'(r^k g_m(\mathbf{x}^{k+1})) \end{bmatrix},$$

potom môžeme (3.6) zapísať ako

$$\mathbf{y}^{k+1} = -\Psi'(r^k g(\mathbf{x}^{k+1})) \mathbf{y}^k. \quad (3.7)$$

Za platnosti predpokladu 3.2 algoritmus nelineárneho škálovania vytvára trajektóriu bodov $\{\mathbf{x}^k\}, \{\mathbf{y}^k\}$, ktorá konverguje k optimálnemu riešeniu lineárne pre

fixovaný prenalizačný parameter r^k , a superlineárne, ak $r^k \rightarrow \infty$. Platia nasledovné ohraňčenia

$$\|\mathbf{x}^{k+1} - \hat{\mathbf{x}}\| \leq \frac{K}{r^k} \|\mathbf{y}^k - \hat{\mathbf{y}}\|, \quad \|\mathbf{y}^{k+1} - \hat{\mathbf{y}}\| \leq \frac{K}{r^k} \|\mathbf{y}^k - \hat{\mathbf{y}}\|, \quad (3.8)$$

kde konštanta K nezávisí od r^k .

Nájdenie presnej hodnoty \mathbf{x}^{k+1} je vo všeobecnosti nemožné, preto sa uspokojíme s jej aproximáciou $\tilde{\mathbf{x}}^{k+1}$. Ak nová aproximatívna dvojica $\tilde{\mathbf{x}}^{k+1}, \tilde{\mathbf{y}}^{k+1}$ generovaná algoritmom nelineárneho škálovania bude spĺňať

$$\tilde{\mathbf{x}}^{k+1} : \|\nabla_x \mathcal{L}_{\text{NR}}(\tilde{\mathbf{x}}^{k+1}, \tilde{\mathbf{y}}^k)\| \leq \frac{\sigma}{r^k} \|\Psi'(r^k g(\tilde{\mathbf{x}}^{k+1})) - \tilde{\mathbf{y}}^k\|, \quad (3.9)$$

$$\tilde{\mathbf{y}}^{k+1} = -\Psi'(r^k g(\tilde{\mathbf{x}}^{k+1})) \tilde{\mathbf{y}}^k, \quad (3.10)$$

potom sa dá ukázať ([31], s. 444-449), že platia podobné odhady ako vo vzťahu (3.8). Dvojicu $\tilde{\mathbf{x}}^{k+1}, \tilde{\mathbf{y}}^{k+1}$ vieme nájsť konečným počtom operácií.

Sústava rovníc (3.5), (3.7) bude základom pre odvodenie primárno-duálnej metódy Polyaka a Grivu ([32], [33], [34]).

3.2 Primárno-duálna metóda Polyaka a Grivu

Uvažujme už spomínanú sústavu rovníc

$$\nabla_x \mathcal{L}_{\text{NR}}(\bar{\mathbf{x}}, \mathbf{y}) = \nabla f(\bar{\mathbf{x}}) + \sum_{i=1}^m \mathbf{y}_i \psi'(r g_i(\bar{\mathbf{x}})) \nabla g_i(\bar{\mathbf{x}}) = \nabla_x \mathbf{L}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = 0, \quad (3.11)$$

$$\bar{\mathbf{y}} = -\Psi'(r g(\bar{\mathbf{x}})) \mathbf{y}. \quad (3.12)$$

Namiesto minimalizovania funkcie (3.1) a následným odhadom multiplikátorov podľa (3.6), budeme riešiť systém rovníc (3.11), (3.12). Na jeho riešenie aplikujeme Newtonovu metódu, pričom použijeme (\mathbf{x}, \mathbf{y}) ako štartovací bod.

Ak predpokladáme $\bar{\mathbf{x}} = \mathbf{x} + \Delta \mathbf{x}$, $\bar{\mathbf{y}} = \mathbf{y} + \Delta \mathbf{y}$, potom linearizáciou systému (3.11), (3.12), zanedbaním členov druhých a vyšších rádov a označením $\tilde{\mathbf{y}} = -\Psi'(r g(\mathbf{x})) \mathbf{y}$ dostaneme

$$\nabla_{xx}^2 \mathbf{L}(\mathbf{x}, \mathbf{y}) \Delta \mathbf{x} - \nabla g(\mathbf{x})^T \Delta \mathbf{y} = -\nabla_x \mathbf{L}(\mathbf{x}, \mathbf{y}), \quad (3.13)$$

$$r \psi''(r g_i(\mathbf{x})) \mathbf{y}_i \nabla g_i(\mathbf{x})^T \Delta \mathbf{x} + \Delta \mathbf{y}_i = \tilde{\mathbf{y}}_i - \mathbf{y}_i, \quad i = 1, \dots, m. \quad (3.14)$$

Ak označíme

$$\mathbf{Y} = \text{diag} [\mathbf{y}_i]_{i=1}^m = \begin{bmatrix} \mathbf{y}_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{y}_m \end{bmatrix},$$

potom môžeme ekvivalentne písať

$$\begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) & -\nabla g(\mathbf{x})^T \\ r\Psi''(rg(\mathbf{x}))\mathbf{Y}\nabla g(\mathbf{x}) & \mathbf{I}_m \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}, \mathbf{y}) \\ \tilde{\mathbf{y}} - \mathbf{y} \end{bmatrix} \quad (3.15)$$

Označme

$$N(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) & -\nabla g(\mathbf{x})^T \\ r\Psi''(rg(\mathbf{x}))\mathbf{Y}\nabla g(\mathbf{x}) & \mathbf{I}_m \end{bmatrix}, \quad (3.16)$$

potom z (3.15) máme

$$N(\mathbf{x}, \mathbf{y}) \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}, \mathbf{y}) \\ \tilde{\mathbf{y}} - \mathbf{y} \end{bmatrix}. \quad (3.17)$$

Maticu $N_k(\mathbf{x}, \mathbf{y})$ budeme regularizovať, aby sme zabezpečili jej regularitu pre ľubovoľnú dvojicu (\mathbf{x}, \mathbf{y}) :

$$N_k(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) + \frac{1}{r^2}\mathbf{I}_n & -\nabla g(\mathbf{x})^T \\ r\Psi''(rg(\mathbf{x}))\mathbf{Y}\nabla g(\mathbf{x}) & \mathbf{I}_m \end{bmatrix}. \quad (3.18)$$

Táto konkrétna voľba regularizácie totiž nebude mať vplyv na rýchlosť konvergencie algoritmu. Predtým, ako popíšeme krok primárno-duálneho algoritmu, zdefinujme tzv. merit funkciu, ktorá bude určovať „vzdialenosť“ od riešenia $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$,

$$\nu(\mathbf{x}, \mathbf{y}) = \max \left\{ \|\nabla_x L(\mathbf{x}, \mathbf{y})\|, -\min_{1 \leq i \leq m} \{g_i(\mathbf{x})\}, \sum_{i=1}^m |\mathbf{y}_i g_i(\mathbf{x})|, -\min_{1 \leq i \leq m} \{\mathbf{y}_i\} \right\}, \quad (3.19)$$

s vlastnosťou $\nu(\mathbf{x}, \mathbf{y}) \geq 0$ a $\nu(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = 0$.

Jeden krok primárno-duálneho algoritmu potom spočíva v nasledovnom.

Algoritmus (Primárno-duálny). *Nech je daná primárno-duálna dvojica $\mathbf{x}^k, \mathbf{y}^k$. Jeden krok primárno-duálneho algoritmu potom spočíva v nasledovnom postupe:*

(1) *Zo sústavy*

$$N_k(\mathbf{x}^k, \mathbf{y}^k) \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}^k, \mathbf{y}^k) \\ \tilde{\mathbf{y}}^k - \mathbf{y}^k \end{bmatrix}$$

nájdem primárno-duálne korektory $\Delta \mathbf{x}, \Delta \mathbf{y}$.

(2) Položíme $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}$, $\mathbf{y}^{k+1} = \mathbf{y}^k + \Delta \mathbf{y}$.

(3) Zmeníme penalizačný parameter r^{k+1} podľa pravidla

$$r^{k+1} = \frac{1}{\sqrt{\nu(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})}}. \quad (3.20)$$

Uvedený postup je dobre definovaný, ak je matica $N_k(\mathbf{x}, \mathbf{y})$ regulárna pre ľubovoľné \mathbf{x}, \mathbf{y} . Ak by nebola, potom by musel existovať nenulový vektor $(\boldsymbol{\omega}, \boldsymbol{\chi})^\top$ taký, že

$$N_k(\mathbf{x}, \mathbf{y}) \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{\chi} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Vzhľadom na štruktúru systému (3.15) môžeme vyjadriť $\boldsymbol{\chi}$ vzťahom

$$\boldsymbol{\chi} = -r\Psi''(rg(\mathbf{x}))\mathbf{Y}\nabla g(\mathbf{x})\boldsymbol{\omega}.$$

Dosadením do prvej rovnice tejto sústavy máme

$$\left(\nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) + \frac{1}{r^2} \mathbf{I}_n + r\nabla g(\mathbf{x})^\top \Psi''(rg(\mathbf{x}))\mathbf{Y}\nabla g(\mathbf{x}) \right) \boldsymbol{\omega} = M_k(\mathbf{x}, \mathbf{y})\boldsymbol{\omega} = 0.$$

Ak vezmeme do úvahy konvexnosť úlohy (\mathcal{PNR}_C) a vlastnosť (i) funkcie ψ , dostaneme, že matica $M_k(\mathbf{x}, \mathbf{y})$ je kladne definitná, z čoho máme $\boldsymbol{\omega} = 0$ a následne $\boldsymbol{\chi} = 0$. Matica $N_k(\mathbf{x}, \mathbf{y})$ je teda regulárna.

Uvedený postup ukazuje, že vzhľadom na štruktúru uvedeného systému môžeme riešenie sústavy zjednodušiť. Najprv nájdeme primárny korektor $\Delta \mathbf{x}$ riešením

$$M_k(\mathbf{x}^k, \mathbf{y}^k)\Delta \mathbf{x} = -\nabla_x \mathcal{L}_{\text{NR}}(\mathbf{x}^k, \mathbf{y}^k),$$

potom položíme

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{x}^k + \Delta \mathbf{x}, \\ \mathbf{y}^{k+1} &= \tilde{\mathbf{y}}^k - r\Psi''(rg(\mathbf{x}^k))\mathbf{Y}\nabla g(\mathbf{x}^k)\Delta \mathbf{x}. \end{aligned}$$

Pre potreby ďalšej analýzy označme

$$\mathbf{w} = (\mathbf{x}, \mathbf{y}), \quad \mathbb{M}_\varepsilon = \{ \mathbf{w} \mid \|\mathbf{w} - \hat{\mathbf{w}}\| \leq \varepsilon \}$$

Potom platí

Veta 3.1. *Nech platí predpoklad 3.2 a nech sú Hessove matice funkcií f, g_i Lipschitzovsy spojité, t.j.*

$$\begin{aligned}\|\nabla^2 f(\mathbf{x}_1) - \nabla^2 f(\mathbf{x}_2)\| &\leq L_0 \|\mathbf{x}_1 - \mathbf{x}_2\| \\ \|\nabla^2 g_i(\mathbf{x}_1) - \nabla^2 g_i(\mathbf{x}_2)\| &\leq L_i \|\mathbf{x}_1 - \mathbf{x}_2\|, \quad i = 1, \dots, m.\end{aligned}$$

Potom existuje ε_0 dostatočne malé tak, že pre ľubovoľnú dvojicu $\mathbf{w} = (\mathbf{x}, \mathbf{y}) \in \mathbb{M}_{\varepsilon_0}$ stačí jediný krok primárno-duálneho algoritmu na nájdenie novej primárno-duálnej dvojice $\bar{\mathbf{w}} = (\bar{\mathbf{x}}, \bar{\mathbf{y}})$ takej, že platí

$$\|\bar{\mathbf{w}} - \hat{\mathbf{w}}\| \leq C \|\mathbf{w} - \hat{\mathbf{w}}\|^{\frac{3}{2}},$$

kde C nezávisí od (\mathbf{x}, \mathbf{y}) .

DÔKAZ: Nájdeime v [33], s. 247-251. □

Vidíme, že pokiaľ sa štartovací bod nachádza v blízkosti optimálneho bodu, primárno-duálny algoritmus konverguje lokálne superlineárne. Dôležitým faktom v dôkaze je práve špeciálna voľba penalizačného parametra r^k . Konvergencia je však iba lokálna, preto spojením primárno-duálnej metódy a nelineárne-škálovacej metódy navrhneime globálne konvergentný algoritmus.

Majme ľubovoľnú dvojicu $(\mathbf{x}^k, \mathbf{y}^k)$. Nový skúšobný bod $(\mathbf{x}_T^{k+1}, \mathbf{y}_T^{k+1})$ najprv nájdeme použitím primárno-duálneho algoritmu. Ak tento bod neznižuje superlineárne hodnotu merit funkcie (t.j. $\nu(\mathbf{x}_T^{k+1}, \mathbf{y}_T^{k+1}) > \nu(\mathbf{x}^k, \mathbf{y}^k)^{1.5}$), znamená to, že bod $(\mathbf{x}^k, \mathbf{y}^k)$ ešte nie je dostatočne blízko optimálneho riešenia (nepatrí do $\mathbb{M}_{\varepsilon_0}$). V tomto prípade zamietneme skúšobný bod $(\mathbf{x}_T^{k+1}, \mathbf{y}_T^{k+1})$ a nový bod $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$ nájdeme nelineárne škálovacím algoritmom - minimalizovaním rozšírenej Lagrangeovej funkcie a odhadom multiplikátorov. K tejto minimalizácii navyše môžeme použiť smer $\Delta \mathbf{x}$ nájdený primárno-duálnym algoritmom, keďže tento smer je vďaka kladnej definitnosti matice $M_k(\mathbf{x}^k, \mathbf{y}^k)$ spádový (t.j. $\nabla_x \mathcal{L}_{NR}(\mathbf{x}^k, \mathbf{y}^k)^T \Delta \mathbf{x} < 0$).

Spočiatku sa teda algoritmus správa rovnako ako nelineárne škálovací algoritmus a v tomto „móde“ vykoná niekoľko iterácií, generujúcich postupnosť bodov $\mathbf{x}^k, \mathbf{y}^k$, ktoré budú čoraz bližšie okoliu $\mathbb{M}_{\varepsilon_0}$ primárno-duálneho riešenia. Potom sa „prepne“ na primárno-duálny algoritmus, kde naplno využije jeho superlineárnu rýchlosť konvergencie (veta 3.1).

Môžeme si všimnúť, že k výraznému zväčšeniu penalizačného parametra r^k dochádza iba počas vykonávania primárno-duálnych krokov v záverečnej fáze algoritmu (vzťah (3.20)). Táto zmena nespôsobuje numerické ťažkosti ako v prípade voľnej minimalizácie, práve naopak: smer $(\Delta \mathbf{x}, \Delta \mathbf{y})$, ktorý získame riešením systému (3.18), sa čoraz viac podobá smeru, ktorý získame Newtonovou metódou aplikovanou na Lagrangeov systém rovníc zodpovedajúci aktívnym ohraničeniam. Algoritmus tak využíva najlepšie vlastnosti nelineárne-škálovacej metódy pre body vzdialené od riešenia $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ a primárno-duálnej metódy pre body v okolí $\mathbb{M}_{\varepsilon_0}$. Zároveň tak obchádza ich základné nedostatky.

Tento výsledný kombinovaný algoritmus Polyak a Griva nazývajú PDNRD algoritmus (primal-dual nonlinear-rescaling method with dynamic scaling parameter update, [33]). V článku [32] rozpracovali zlepšenie primárno-duálnej schémy zakladajúce sa na identifikácii nulových multiplikátorov a zavedení vektora penalizačných parametrov, ktoré teraz uvedieme.

3.3 Vylepšená primárno-duálna schéma

Výsledkom spomínaného zlepšenia bude primárno-duálna metóda s asymptoticky kvadratickou rýchlosťou konvergence. Základom bude identifikácia aktívnych ohraničení a nulových Lagrangeových multiplikátorov počas vykonávania algoritmu, k čomu posluží fakt, že Lagrangeove multiplikátory prislúchajúce pasívnym ohraničeniam konvergujú k nule kvadraticky. Takisto zavedieme vektor škálovacích parametrov $\mathbf{p} = (\mathbf{p}_1, \dots, \mathbf{p}_m)^T$, ktorý bude v úzkom spojení s penalizačným parametrom r .

Budeme vychádzať zo sústavy

$$\nabla_x \mathcal{L}_{\text{NR}}(\bar{\mathbf{x}}, \mathbf{y}) = \nabla f(\bar{\mathbf{x}}) + \sum_{i=1}^m \mathbf{y}_i \psi'(\mathbf{p}_i g_i(\bar{\mathbf{x}})) \nabla g_i(\bar{\mathbf{x}}) = \nabla_x L(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = 0, \quad (3.21)$$

$$\bar{\mathbf{y}} = -\Psi'(\mathbf{p}g(\bar{\mathbf{x}}))\mathbf{y}, \quad (3.22)$$

$$\mathbf{p}_i = \frac{r}{\mathbf{y}_i}, \quad i = 1, \dots, m, \quad (3.23)$$

kde

$$\Psi'(\mathbf{p}g(\mathbf{x})) = \text{diag} [\psi'(\mathbf{p}_i g_i(\mathbf{x}))]_{i=1}^m = \begin{bmatrix} \psi'(\mathbf{p}_1 g_1(\mathbf{x})) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \psi'(\mathbf{p}_m g_m(\mathbf{x})) \end{bmatrix}.$$

Definujme indexové množiny

$$I_+(\mathbf{x}, \mathbf{y}) = \{i \mid \mathbf{y}_i > \nu(\mathbf{x}, \mathbf{y})\}, \quad I_0(\mathbf{x}, \mathbf{y}) = \{i \mid \mathbf{y}_i < \nu(\mathbf{x}, \mathbf{y})\},$$

reprezentujúce množiny „veľkých“ a „malých“ Lagrangeových multiplikátorov. Rozdelíme systém (3.22) na dva podsystémy

$$\bar{\mathbf{y}}_i = -\mathbf{y}_i \psi'(\mathbf{p}_i g_i(\bar{\mathbf{x}})), \quad i \in I_+(\mathbf{x}, \mathbf{y}), \quad (3.24)$$

$$\bar{\mathbf{y}}_i = -\mathbf{y}_i \psi'(\mathbf{p}_i g_i(\bar{\mathbf{x}})), \quad i \in I_0(\mathbf{x}, \mathbf{y}). \quad (3.25)$$

Pri linearizácii systému rovníc (3.21), (3.24), (3.25) budeme prihliadať na odlišnosť týchto systémov. Uvedieme len výslednú linearizovanú schému, keďže jej odvodenie vyžaduje určité znalosti teórie Fenchelovej konvexnej transformácie. Označme

$$\left. \begin{aligned} g_+(\mathbf{x}) &= (g_i(\mathbf{x}))^T \\ \mathbf{y}_+ &= (\mathbf{y}_i)^T \\ \mathbf{p}_+ &= (\mathbf{p}_i)^T \end{aligned} \right\} i \in I_+(\mathbf{x}, \mathbf{y}),$$

$$\left. \begin{aligned} g_0(\mathbf{x}) &= (g_i(\mathbf{x}))^T \\ \mathbf{y}_0 &= (\mathbf{y}_i)^T \\ \mathbf{p}_0 &= (\mathbf{p}_i)^T \end{aligned} \right\} i \in I_0(\mathbf{x}, \mathbf{y}),$$

$$\tilde{\mathbf{y}}_0 = -\Psi'(\mathbf{p}_0 g_0(\mathbf{x})) \mathbf{y}_0, \quad (3.26)$$

potom dostávame (regularizovanú) sústavu (pre detaily odvodenia pozri [32])

$$\begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) + \frac{1}{r} \mathbf{I}_n & -\nabla g_+(\mathbf{x})^T & -\nabla g_0(\mathbf{x})^T \\ \nabla g_+(\mathbf{x}) & -\frac{\varphi''(1)}{r} \mathbf{I}_+ & 0 \\ r\Psi''(\mathbf{p}_0 g_0(\mathbf{x})) \mathbf{Y}_0 \nabla g_0(\mathbf{x}) & 0 & \mathbf{I}_0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y}_+ \\ \Delta \mathbf{y}_0 \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}, \mathbf{y}) \\ -g_+(\mathbf{x}) \\ \tilde{\mathbf{y}}_0 - \mathbf{y}_0 \end{bmatrix},$$

kde φ je záporne vzatá Fenchelova transformácia funkcie ψ . Opäť označíme

$$N_k(\mathbf{x}, \mathbf{y}_+, \mathbf{y}_0) = \begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) + \frac{1}{r} \mathbf{I}_n & -\nabla g_+(\mathbf{x})^T & -\nabla g_0(\mathbf{x})^T \\ \nabla g_+(\mathbf{x}) & -\frac{\varphi''(1)}{r} \mathbf{I}_+ & 0 \\ r\Psi''(\mathbf{p}_0 g_0(\mathbf{x})) \mathbf{Y}_0 \nabla g_0(\mathbf{x}) & 0 & \mathbf{I}_0 \end{bmatrix} \quad (3.27)$$

a popíšeme jeden krok modifikovanej primárno-duálnej metódy.

Algoritmus (Modifikovaný primárno-duálny). *Nech je daná primárno-duálna dvojica $\mathbf{x}^k, \mathbf{y}_+^k, \mathbf{y}_0^k$. Jeden krok modifikovaného primárno-duálneho algoritmu potom spočíva v nasledovnom postupe:*

(1) Zo sústavy

$$N_k(\mathbf{x}^k, \mathbf{y}_+, \mathbf{y}_0) \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y}_+ \\ \Delta \mathbf{y}_0 \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}^k, \mathbf{y}^k) \\ -g_+(\mathbf{x}^k) \\ \tilde{\mathbf{y}}_0^k - \mathbf{y}_0^k \end{bmatrix},$$

nájďeme primárno-duálne korektory $\Delta \mathbf{x}, \Delta \mathbf{y}_+, \Delta \mathbf{y}_0$.

(2) Položíme $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}$, $\mathbf{y}_+^{k+1} = \mathbf{y}_+^k + \Delta \mathbf{y}_+$, $\mathbf{y}_0^{k+1} = \mathbf{y}_0^k + \Delta \mathbf{y}_0$

(3) Zmeníme penalizačný parameter r^{k+1} podľa pravidla

$$r^{k+1} = \frac{1}{\nu(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})} \quad (3.28)$$

a položíme

$$\mathbf{p}_i^{k+1} = \frac{r^{k+1}}{\mathbf{y}_i^{k+1}}, \quad i = 1, \dots, m.$$

Oproti pôvodnému algoritmu si môžeme všimnúť aj pozmenené pravidlo zmeny penalizačného parametra.

Globálne konvergentný algoritmus opäť dostaneme spojením nelineárne škálovacieho algoritmu a modifikovaného primárno-duálneho algoritmu. Nelineárne škálovací algoritmus posluží v úvodnej fáze na získanie bodov $\mathbf{x}^k, \mathbf{y}^k$ blízko optimálneho riešenia, modifikovaný primárno-duálny algoritmus potom spúšťame v záverečnej výpočtovej fáze. Tento algoritmus Polyak a Griva nazývajú PDEPM (primal-dual exterior point method, [32]). PDEMP vykazuje asymptoticky kvadratickú rýchlosť konvergenzie.

Veta 3.2. *Nech platí predpoklad 3.2 a nech sú Hessove matice funkcií f, g_i Lipschitzovsky spojité, t.j.*

$$\|\nabla^2 f(\mathbf{x}_1) - \nabla^2 f(\mathbf{x}_2)\| \leq L_0 \|\mathbf{x}_1 - \mathbf{x}_2\|$$

$$\|\nabla^2 g_i(\mathbf{x}_1) - \nabla^2 g_i(\mathbf{x}_2)\| \leq L_i \|\mathbf{x}_1 - \mathbf{x}_2\|, \quad i = 1, \dots, m.$$

potom algoritmus PDEPM generuje globálne konvergentnú postupnosť bodov $\mathbf{x}^k, \mathbf{y}^k$ konvergujúcu k optimálnemu riešeniu $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ s asymptoticky kvadratickou rýchlosťou.

DÔKAZ: Nájďeme v [32], s. 155-158. □

V poslednej časti tejto kapitoly ukážeme spôsob, ktorým môžeme previesť KKT systém na ekvivalentný systém, ktorý však bude obsahovať iba rovnice. Optimálny bod $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ potom nájdeme riešením tohto systému.

3.4 Fischerova schéma

Doposiaľ sme sa zaoberali metódami, ktoré hľadajú sedlový bod (rozšírenej) Lagrangeovej funkcie prostredníctvom ich minimalizácie na priestore primárnej premennej, nasledovanej voľbou nového vektora Lagrangeových multiplikátorov, ktorý maximalizuje penalizovanú duálnu funkciu príslušnej duálnej úlohy. Metódy uvedené v častiach 3.2 a 3.3 tento bod hľadajú prostredníctvom riešenia primárno-duálnej sústavy, celková filozofia metód však zostáva zachovaná. V tejto časti spomenieme odlišný prístup, založený na transformácii KKT sústavy na ekvivalentnú sústavu rovníc.

Pripomeňme uvažovanú úlohu

$$\begin{aligned} & \text{minimalizovať} && f(\mathbf{x}) \\ & \text{za podmienok} && g(\mathbf{x}) \geq 0 \end{aligned} \tag{PNRC}$$

kde $f : \mathbb{R}^n \rightarrow \mathbb{R}$ je konvexná, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ konkávna vektorová funkcia. Predpokladajme, že $f, g \in C^2$, a definujme pomocou

$$L(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) - g(\mathbf{x})^T \mathbf{y}$$

príslušnú Lagrangeovu funkciu. Ohraničenia typu

$$h(\mathbf{x}) = 0$$

neuvažujeme iba kvôli zjednodušeniu výkladu.

Nutné a postačujúce podmienky optimality pre úlohu (PNRC) v podobe Karush-Kuhn-Tuckerových podmienok majú tvar (veta 1.6)

$$\begin{aligned} \nabla f(\mathbf{x}) - \sum_{i=1}^m \mathbf{y}_i \nabla g_i(\mathbf{x}) &= 0, \\ g(\mathbf{x}) &\geq 0, \\ \mathbf{y}^T g(\mathbf{x}) &= 0, \\ \mathbf{y} &\geq 0, \end{aligned}$$

ktoré zapíšeme ako

$$\nabla_x L(\mathbf{x}, \mathbf{y}) = 0, \tag{3.29}$$

$$g(\mathbf{x}) \geq 0, \mathbf{y} \geq 0, g(\mathbf{x})^T \mathbf{y} = 0. \tag{3.30}$$

Vzťah (3.30) je podobný formulácii úloh o komplementarite, ktoré je možné pretransformovať na systém rovníc pomocou tzv. CP-funkcií (complementarity problem functions, [27]).

Definícia 3.1 (CP-funkcia). *Funkciu $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}$ nazveme CP-funkciou, ak platí*

$$\phi(\alpha, \beta) = 0 \iff \alpha \geq 0, \beta \geq 0, \alpha\beta = 0. \quad (3.31)$$

Z uvedenej definície je zrejmé, že pomocou CP-funkcií môžeme transformovať systém (3.29), (3.30) na ekvivalentný systém

$$\nabla_x L(\mathbf{x}, \mathbf{y}) = 0, \quad (3.32)$$

$$\phi(g_i(\mathbf{x}), \mathbf{y}_i) = 0, \quad i = 1, \dots, m. \quad (3.33)$$

V minulosti bolo navrhnutých viacero takýchto funkcií (spomínané napr. v [13]), no v poslednej dobe sa teší popularite najmä CP-funkcia zavedená Fischerom ([14], [15]).

Veta 3.3. *Nech je funkcia ϕ_F definovaná vzťahom*

$$\phi_F(\alpha, \beta) = \alpha + \beta - \sqrt{\alpha^2 + \beta^2}.$$

Potom ϕ_F je CP-funkcia.

DÔKAZ: (\implies) Nech platí $\phi_F(\alpha, \beta) = 0$. Ak $\alpha + \beta < 0$, potom $\phi_F(\alpha, \beta) < 0$ pre ľubovoľné α, β spĺňajúce uvedenú nerovnosť. Predpokladajme teda, že $\alpha + \beta \geq 0$. Potom namiesto výrazu $\phi_F(\alpha, \beta) = 0$ môžeme ekvivalentne písať

$$(\alpha + \beta)^2 = \alpha^2 + \beta^2,$$

z čoho plynie $\alpha\beta = 0$. Využitím výrazov $\alpha + \beta \geq 0$ a $\alpha\beta = 0$ už ľahko dostaneme, že musí platiť $\alpha \geq 0$ a takisto $\beta \geq 0$.

(\impliedby) Ak platí $\alpha\beta = 0$, potom môžeme písať

$$\begin{aligned} \phi_F(\alpha, \beta) &= \alpha + \beta - \sqrt{\alpha^2 + \beta^2 + 2\alpha\beta} = \alpha + \beta - \sqrt{(\alpha + \beta)^2} \\ &= \alpha + \beta - |\alpha + \beta|. \end{aligned}$$

Keďže predpokladáme $\alpha \geq 0, \beta \geq 0$, potom $\phi_F(\alpha, \beta) = 0$, čo sme chceli dokázať. \square

Funkcia ϕ_F je spojitá a pre jej parciálne derivácie platí

$$\frac{\partial \phi_F(\alpha, \beta)}{\partial \alpha} = 1 - \frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}, \quad \frac{\partial \phi_F(\alpha, \beta)}{\partial \beta} = 1 - \frac{\beta}{\sqrt{\alpha^2 + \beta^2}}. \quad (3.34)$$

Problém so spojitosťou prvých parciálnych derivácií (3.34) v bode $(0, 0)$ je možné obísť zavedením regularizovanej Fischerovej funkcie

$$\phi_{F\mu}(\alpha, \beta) = \alpha + \beta - \sqrt{\alpha^2 + \beta^2 + 2\mu} \quad (3.35)$$

s vlastnosťou

$$\phi_{F\mu}(\alpha, \beta) = 0 \iff \alpha > 0, \beta > 0, \alpha\beta = \mu, \quad (3.36)$$

pre $\mu > 0$, ktorú dokážeme rovnako ako vo vete 3.3. Označme

$$\Phi_{F\mu}(g(\mathbf{x}), \mathbf{y}) = \begin{bmatrix} \phi_{F\mu}(g_1(\mathbf{x}), \mathbf{y}_1) \\ \vdots \\ \phi_{F\mu}(g_m(\mathbf{x}), \mathbf{y}_m) \end{bmatrix},$$

potom budeme aproximovať systém (3.32), (3.33) systémom

$$\nabla_x L(\mathbf{x}, \mathbf{y}) = 0 \quad (3.37)$$

$$\Phi_{F\mu}(g(\mathbf{x}), \mathbf{y}) = 0. \quad (3.38)$$

Jedným zo spôsobov riešenia tohto systému je minimalizácia jeho normy. V tomto prípade vznikajú menšie problémy aj pri ostro konvexných úlohách [12], preto na jeho riešenie aplikujeme Newtonovu metódu. Linearizovaním uvedenej sústavy a zanedbaním členov druhého a vyššieho rádu dostávame (predpokladáme $\bar{\mathbf{x}} = \mathbf{x} + \Delta\mathbf{x}$, $\bar{\mathbf{y}} = \mathbf{y} + \Delta\mathbf{y}$)

$$\begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) & -\nabla g(\mathbf{x})^T \\ \mathbf{D}_\alpha \nabla g(\mathbf{x}) & \mathbf{D}_\beta \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}, \mathbf{y}) \\ -\Phi_{F\mu}(g(\mathbf{x}), \mathbf{y}) \end{bmatrix}. \quad (3.39)$$

kde používame označenie

$$\mathbf{D}_\alpha = \text{diag} [\phi'_{F\mu;\alpha}(g(\mathbf{x}), \mathbf{y})]_{i=1}^m = \begin{bmatrix} \phi'_{F\mu;\alpha}(g_1(\mathbf{x}), \mathbf{y}_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \phi'_{F\mu;\alpha}(g_m(\mathbf{x}), \mathbf{y}_m) \end{bmatrix},$$

$$\mathbf{D}_\beta = \text{diag} [\phi'_{F\mu;\beta}(g(\mathbf{x}), \mathbf{y})]_{i=1}^m = \begin{bmatrix} \phi'_{F\mu;\beta}(g_1(\mathbf{x}), \mathbf{y}_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \phi'_{F\mu;\beta}(g_m(\mathbf{x}), \mathbf{y}_m) \end{bmatrix},$$

pričom

$$\begin{aligned}\phi'_{F\mu;\alpha}(g_i(\mathbf{x}), \mathbf{y}_i) &= \left. \frac{\partial \phi_{F\mu}(\alpha, \beta)}{\partial \alpha} \right|_{\alpha=g_i(\mathbf{x}), \beta=\mathbf{y}_i}, \\ \phi'_{F\mu;\beta}(g_i(\mathbf{x}), \mathbf{y}_i) &= \left. \frac{\partial \phi_{F\mu}(\alpha, \beta)}{\partial \beta} \right|_{\alpha=g_i(\mathbf{x}), \beta=\mathbf{y}_i}.\end{aligned}$$

Označme

$$N_\gamma(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) + \gamma \mathbf{I}_n & -\nabla g(\mathbf{x})^\top \\ \mathbf{D}_\alpha \nabla g(\mathbf{x}) & \mathbf{D}_\beta \end{bmatrix}, \quad \gamma > 0 \quad (3.40)$$

regularizovanú maticu sústavy (3.39). Newtonovské smery $(\Delta \mathbf{x}, \Delta \mathbf{y})$ budú touto sústavou dobre definované, ak $N_\gamma(\mathbf{x}, \mathbf{y})$ bude regulárna matica pre ľubovoľné (\mathbf{x}, \mathbf{y}) . Ak by nebola regulárna, potom by muselo platiť

$$N_\gamma(\mathbf{x}, \mathbf{y}) \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{\chi} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

pre nenulový vektor $(\boldsymbol{\omega}, \boldsymbol{\chi})^\top$. Z druhej rovnice pre $\boldsymbol{\chi}$ dostávame

$$\boldsymbol{\chi} = -\mathbf{D}_\beta^{-1} \mathbf{D}_\alpha \nabla g(\mathbf{x}) \boldsymbol{\omega}$$

a dosadením do prvej rovnice máme

$$\left(\nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) + \gamma \mathbf{I}_n + \nabla g(\mathbf{x})^\top \mathbf{D}_\beta^{-1} \mathbf{D}_\alpha \nabla g(\mathbf{x}) \right) \boldsymbol{\omega} = M_\gamma(\mathbf{x}, \mathbf{y}) \boldsymbol{\omega} = 0.$$

Matica $M_\gamma(\mathbf{x}, \mathbf{y})$ je kladne definitná (konvexnosť úlohy (\mathcal{PNR}_C) a kladnosť prvkov diagonálnych matic $\mathbf{D}_\alpha, \mathbf{D}_\beta$), čiže $\boldsymbol{\omega} = 0$, a následne $\boldsymbol{\chi} = 0$. Matica $N_\gamma(\mathbf{x}, \mathbf{y})$ je regulárna.

Základnú ideu hľadania optimálneho riešenia $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ potom môžeme zhrnúť takto:

Algoritmus (Fischerov). *Nech je daná dvojica vektorov $\mathbf{x}^k, \mathbf{y}^k$. Jeden krok Fischerovho algoritmu potom spočíva v nasledovnom postupe:*

(1) *Zo sústavy*

$$N_\gamma(\mathbf{x}^k, \mathbf{y}^k) \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}^k, \mathbf{y}^k) \\ -\Phi_{F\mu}(g(\mathbf{x}^k), \mathbf{y}^k) \end{bmatrix}$$

nájdem Newtonovské smery $\Delta \mathbf{x}, \Delta \mathbf{y}$.

(2) *Položíme $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}$, $\mathbf{y}^{k+1} = \mathbf{y}^k + \Delta \mathbf{y}$ a zvolíme $\gamma^{k+1} \leq \gamma^k$.*

Uvedený algoritmus však konverguje len pre body $(\mathbf{x}^k, \mathbf{y}^k)$ patriace do nejakého okolia optimálneho riešenia $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, pričom veľkosť tohto okolia závisí od dát úlohy. Pri rozšírení algoritmu na globálny sa ponúkajú dve možnosti

(a) Krok (2) uvedeného algoritmu nahradíme

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \Delta \mathbf{x}, \quad \mathbf{y}^{k+1} = \mathbf{y}^k + \lambda^k \Delta \mathbf{y},$$

kde λ^k získame vyhľadávaním na lúči normy riešeného systému. Ak označíme $\mathbf{w} = (\mathbf{x}, \mathbf{y})$, $\mathbf{s} = (\Delta \mathbf{x}, \Delta \mathbf{y})$ a

$$F(\mathbf{w}) = \begin{bmatrix} \nabla_x L(\mathbf{x}, \mathbf{y}) \\ \Phi_{F\mu}(g(\mathbf{x}), \mathbf{y}) \end{bmatrix},$$

potom λ^k získame približným riešením úlohy

$$\lambda^k = \arg \min_{\lambda > 0} \{ \|F(\mathbf{w} + \lambda \mathbf{s})\| \}.$$

(b) Na získanie potrebných bodov $(\mathbf{x}^k, \mathbf{y}^k)$ môžeme využiť napríklad nelineárne škálovací algoritmus. Fischerovu metódu potom spustíme až v čase, keď tieto body budú ležať v požadovanom okolí optimálneho riešenia $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$.

Ani jednu z uvedených modifikácií nepodporíme dôkazom konvergenzie, no pokúsime sa ich otestovať rozličnými numerickými experimentami.

Kapitola 4

Numerické experimenty

Závěrečnú kapitolu tejto práce venujeme numerickým experimentom, ktorých cieľom bude vzájomné porovnanie rôznych metód k riešeniu úlohy konvexného programovania v tvare

$$\begin{array}{ll} \text{minimalizovať} & f(\mathbf{x}) \\ \text{za podmienok} & g(\mathbf{x}) \geq 0 \end{array} \quad (\mathcal{PNRC})$$

kde $f : \mathbb{R}^n \rightarrow \mathbb{R}$ je konvexná, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ je konkávna vektorová funkcia, $g(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))^T$. Za týmto účelom využijeme náhodne generované úlohy konvexného bikvadratického (kvadratického) programovania, a takisto vybrané úlohy zo zbierky testovacích príkladov CUTEr [18].

Kapitola je rozdelená do troch logických celkov. V prvom popíšeme spôsob náhodného generovania úloh, v druhom sa budeme venovať implementácii uvažovaných algoritmov, a v treťom uvedieme výsledky navrhnutých experimentov, ktoré stručne okomentujeme.

4.1 Generátor úloh

Dôležitým predpokladom pre tvorbu automatického generátora úloh (\mathcal{PNRC}) je voľba konkrétnej triedy úloh, pre ktorú budeme môcť takýto generátor ľahko implementovať. My sme si zvolili triedu úloh bikvadratického programovania. Konkrétna

voľba funkcií f, g teda bude

$$f(\mathbf{x}) = \frac{1}{4}(\mathbf{x}^T \mathbf{D} \mathbf{x})^2 + \frac{1}{2} \mathbf{x}^T \mathbf{G} \mathbf{x} + \mathbf{h}^T \mathbf{x},$$

$$g_i(\mathbf{x}) = \begin{cases} \frac{1}{2} \mathbf{x}^T \mathbf{G}_i \mathbf{x} + \mathbf{h}_i^T \mathbf{x} - t_i, & i = 1, \dots, q, \\ \mathbf{a}_i^T \mathbf{x} - b_i, & i = 1, \dots, m - q. \end{cases} \quad (4.1)$$

pre nejaké kladne definitné matice \mathbf{D}, \mathbf{G} (\mathbf{D} diagonálna) a záporne definitné matice $\mathbf{G}_i, i = 1, \dots, q$, kde predpokladáme $0 \leq q \leq m$. Vektory $\mathbf{h}, \mathbf{h}_i, \mathbf{t}_i, \mathbf{a}_i, \mathbf{b}_i$ volíme „ľubovoľne“ (drobné obmedzenie špecifikujeme neskôr). Z riadkových vektorov \mathbf{a}_i^T utvoríme maticu \mathbf{A} . Pre takto zvolené funkcie a matice bude úloha (\mathcal{PNRC}) rýdzo konvexná. Jej primárno-duálne optimálne riešenie $(\hat{\mathbf{x}}, \hat{\mathbf{u}})$ je potom jednoznačne determinované sústavou KKT podmienok, ktoré nadobúdajú tvar

$$(\hat{\mathbf{x}}^T \mathbf{D} \hat{\mathbf{x}}) \mathbf{D} \hat{\mathbf{x}} + \mathbf{G} \hat{\mathbf{x}} + \mathbf{h} - \mathbf{A}^T \hat{\mathbf{u}}_{(m-q)} - \sum_{i=1}^q \hat{\mathbf{u}}_i (\mathbf{G}_i \hat{\mathbf{x}} + \mathbf{h}_i) = 0 \quad (4.2)$$

$$\frac{1}{2} \hat{\mathbf{x}}^T \mathbf{G}_i \hat{\mathbf{x}} + \mathbf{h}_i^T \hat{\mathbf{x}} \geq t_i, \quad i = 1, \dots, q, \quad (4.3)$$

$$\mathbf{u}_i \left(\frac{1}{2} \hat{\mathbf{x}}^T \mathbf{G}_i \hat{\mathbf{x}} + \mathbf{h}_i^T \hat{\mathbf{x}} - t_i \right) = 0, \quad i = 1, \dots, q, \quad (4.4)$$

$$\mathbf{A} \hat{\mathbf{x}} \geq \mathbf{b}, \quad (4.5)$$

$$\hat{\mathbf{u}}_{(m-q)}^T (\mathbf{A} \hat{\mathbf{x}} - \mathbf{b}) = 0, \quad (4.6)$$

$$\hat{\mathbf{u}} \geq 0, \quad (4.7)$$

kde označujeme $\mathbf{u}_{(m-q)} = (\mathbf{u}_{q+1}, \dots, \mathbf{u}_m)^T$ a $\mathbf{u} = (\mathbf{u}_1, \dots, \mathbf{u}_q, \mathbf{u}_{(m-q)})^T$. Potrebujeme ešte zabezpečiť lineárnu nezávislosť gradientov $\nabla g_i(\hat{\mathbf{x}})$ prislúchajúcich aktívnym ohraničeniam. Predpokladajme, že existujú $1 \leq q_1 \leq q, 1 \leq q_2 \leq m - q, q_1 + q_2 \leq n$ také, že prvých q_1 kvadratických ohraničení a prvých q_2 lineárnych ohraničení je aktívnych v $\hat{\mathbf{x}}$. Potom musí platiť

$$\text{rank}(\nabla g_{(q_1+q_2)}(\hat{\mathbf{x}})) = \min\{q_1 + q_2, n\}, \quad \nabla g_{(q_1+q_2)}(\hat{\mathbf{x}}) = \begin{bmatrix} (\mathbf{G}_1 \hat{\mathbf{x}} + \mathbf{h}_1)^T \\ \vdots \\ (\mathbf{G}_{q_1} \hat{\mathbf{x}} + \mathbf{h}_{q_1})^T \\ \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_{q_2}^T \end{bmatrix}, \quad (4.8)$$

čo môžeme dosiahnuť vhodnou voľbou matíc úlohy.

Sústava rovníc a nerovnic (4.2) - (4.7) ponúka pomerne jednoduchý spôsob na generovanie úloh (\mathcal{PNRC}) v požadovanom tvare: Najprv zvolíme ľubovoľné $\hat{\mathbf{x}}$ a $\hat{\mathbf{y}} \geq 0$, nato zvolíme matice a vektory $\mathbf{G}_i, \mathbf{h}_i, \mathbf{A}, i = 1, \dots, q$ (\mathbf{G}_i záporne definitné) tak, aby platilo (4.8). Následne zvolíme ľubovoľné, kladne definitné matice \mathbf{D} a \mathbf{G} , a zo vzťahov (4.3) - (4.6) dopočítame vektory \mathbf{b}, \mathbf{t} . Na záver určíme \mathbf{h} z rovnice (4.2). Schematický popis implementácie tohto postupu je nasledovný:

Krok 1 : Inicializácia. Zvoľ n (počet premenných), m (počet ohraničení), $0 \leq q \leq m$ (počet kvadratických ohraničení), $m_a < n$ (počet aktívnych ohraničení v bode optima), ďalej nastav $\rho > 0$ (vzdialenosť bodu \mathbf{x}^0 od optimálneho riešenia) a $cond > 1$ (číslo podmienenosti matíc $\mathbf{D}, \mathbf{G}, \mathbf{G}_i$).

Krok 2 : Vygeneruj ľubovoľné $\hat{\mathbf{x}} \in \mathbb{R}^n$ a ľubovoľné také $\hat{\mathbf{y}} \geq 0$, ktoré obsahuje $m - m_a$ nulových prvkov a m_a kladných prvkov.

Krok 3 : Vytvor maticu $\mathbf{G} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$, kde \mathbf{Q} je ľubovoľná ortogonálna matica a $\mathbf{\Lambda}$ je diagonálna matica s prvkami $\lambda_i > 0$ zvolenými tak, aby platilo $\frac{\lambda_{\max}}{\lambda_{\min}} = cond$. Rovnako určí $\mathbf{G}_i = \mathbf{Q}_i\mathbf{\Lambda}_i\mathbf{Q}_i^T$, pre $\mathbf{\Lambda}_i$ diagonálne s prvkami $\lambda_{ij} < 0$ zvolenými tak, aby platilo $\frac{\lambda_{i,\max}}{\lambda_{i,\min}} = cond$.

Krok 4 : Zvoľ $d > 0$ tak, aby $\frac{d_{\max}}{d_{\min}} = cond$, a polož $\mathbf{D} = \text{diag}(d)$.

Krok 5 : Vygeneruj maticu \mathbf{A} a vektory $\mathbf{h}_i, i = 1, \dots, q$ tak, aby platilo (4.8), a dopočítaj vektory \mathbf{b}, \mathbf{t} podľa

$$\left. \begin{aligned} t_i &= \frac{1}{2}\hat{\mathbf{x}}^T \mathbf{G}_i \hat{\mathbf{x}} + \mathbf{h}_i^T \hat{\mathbf{x}} - \theta_i & \text{ak } u_i = 0 \\ t_i &= \frac{1}{2}\hat{\mathbf{x}}^T \mathbf{G}_i \hat{\mathbf{x}} + \mathbf{h}_i^T \hat{\mathbf{x}} & \text{ak } u_i > 0 \end{aligned} \right\} i = 1, \dots, q,$$

$$\left. \begin{aligned} b_i &= \mathbf{a}^T \hat{\mathbf{x}} - \theta_i & \text{ak } u_{q+i} = 0 \\ b_i &= \mathbf{a}^T \hat{\mathbf{x}} & \text{ak } u_{q+i} > 0 \end{aligned} \right\} i = 1, \dots, m - q,$$

pričom $\theta_i > 0$ sú náhodné čísla.

Krok 6 : Z (4.2) dopočítaj \mathbf{h} .

Krok 7 : Zvoľ náhodne vektor \mathbf{s} a polož $\mathbf{x}^0 = \hat{\mathbf{x}} + \rho \frac{\mathbf{s}}{\|\mathbf{s}\|}$.

Krok 8 : Výstup : Matice a vektory dát $\hat{\mathbf{x}}, \hat{\mathbf{u}}, \mathbf{D}, \mathbf{G}, \mathbf{h}, \mathbf{G}_i, \mathbf{h}_i, \mathbf{t}, \mathbf{A}, \mathbf{b}, \mathbf{x}^0$.

Parametrami generátora sú konštanty $\{n, m, m_a, q, \rho, cond\}$, ktorých rozličným nastavením budú vznikať úlohy s rôznymi charakteristikami. Toto využijeme pri vykonávaní numerických experimentov.

Všimnime si ešte, že algoritmus nášho generátora úloh zabezpečí platnosť podmienky ostrej komplementarity (1.23). Maticu \mathbf{D} pri generovaní volíme diagonálnu.

4.2 Implementácie algoritmov

Na tomto mieste sa budeme venovať popisu algoritmov, ktoré použijeme na numerické experimentovanie. Konkrétne sa bude jednať o:

- (a) Klasický algoritmus Rockafellara, kde použijeme dva rôzne prístupy pre odhad multiplikátorov: odhad prvého rádu a odhad druhého rádu.
- (b) Algoritmus nelineárneho škálovania pre vybranú transformačnú funkciu $\psi \in \mathcal{P}_{\text{NEQ}}^2$.
- (c) Algoritmus PDNRD Polyaka a Grivu.
- (d) „Fischerov“ algoritmus v dvoch verziách, ktoré nazveme „klasická“ a „modifikovaná“.

Detaily Newtonovho algoritmu na voľnú minimalizáciu a algoritmov na vyhľadávanie na lúči možno nájsť v Prílohe. Predtým, ako pristúpime ku konkrétnym implementáciám algoritmov, zdefinujme funkciu

$$\nu(\mathbf{x}, \mathbf{y}) = \max \left\{ \|\nabla_{\mathbf{x}} L(\mathbf{x}, \mathbf{y})\|, -\min_{1 \leq i \leq m} \{g_i(\mathbf{x})\}, \sum_{i=1}^m |\mathbf{y}_i g_i(\mathbf{x})|, -\min_{1 \leq i \leq m} \{\mathbf{y}_i\} \right\}, \quad (4.9)$$

ktorú nazveme „merit“ funkciou a budeme ňou merať kvalitu aproximácie jednotlivých iteračných bodov $(\mathbf{x}^k, \mathbf{y}^k)$, keďže pre všetky uvažované algoritmy platí $\hat{\mathbf{u}} = \hat{\mathbf{y}}$. Predpokladáme, že vstup je vhodne škálovaný.

Na záver prezradíme, že všetky algoritmy sme naprogramovali v prostredí MATLAB (verzia 7.12). Kvôli snahe o zrýchlenie niektorých výpočtov kombinujeme klasické .m súbory so skompilovanými časťami kódu jazyku C v podobe .mex súborov, v ktorých využívame FORTRANovské funkcie z knižnice BLAS. Keďže sme chceli použiť na testovanie aj spomínaný testovací balík CUTEr, museli sme pracovať pod operačným systémom LINUX (distribúcia Ubuntu, v11.04). Implementácia, ktorá je špeciálne určená pre výstup generátora úloh je však po vhodnom prekompilovaní spustiteľná aj pod operačným systémom Windows.

4.2.1 Rockafellarov algoritmus

Pripomeňme označenia Rockafellarovej rozšírenej Lagrangeovej funkcie

$$\mathcal{L}_R(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \sum_{i=1}^m P_R(g_i(\mathbf{x}), \mathbf{y}_i),$$

s Rockafellarovou penalizačnou funkciou

$$P_R(\xi, \eta) = \begin{cases} \frac{1}{2}\xi^2 - \xi\eta & \text{pre } \xi < \eta \\ -\frac{\eta^2}{2} & \text{pre } \xi \geq \eta \end{cases}.$$

Algoritmus Rockafellara, ktorý sme teoreticky popísali v časti 2.2, sme implementovali nasledovne:

- Krok 1 :** Inicializácia. Urči konštanty ε (terminačné kritérium), $r_{\max} < 10^6$, $r_{\text{stp}} \in [2, 10]$ (konštanty použité na zmenu parametra r), δ (očakávaný pokles porušenia prípustnosti), konštanty $0 < \nu_{\text{crit}} \leq 1$, $odhad = 1$ a $0 < L < 1$, a urči hodnotu $r^0 > 0$. Polož $\mathbf{y}^0 = (1, \dots, 1)^T$ a $k = 0$.
- Krok 2 :** AK $\nu(\mathbf{x}^k, \mathbf{y}^k) < \varepsilon$, STOP, **Výstup :** $\mathbf{x}^k, \mathbf{y}^k$.
- Krok 3 :** Polož $\tilde{\mathbf{y}} = \text{multiplier}(\mathbf{x}^k, \mathbf{y}^k, odhad)$, označ $\mathbf{g}^k = \|\nabla_x \mathcal{L}_R(\mathbf{x}^k, \mathbf{y}^k)\|$ a použi Newtonovu metódu na približnú minimalizáciu funkcie $\mathcal{L}_R(\cdot, \mathbf{y}^k)$ na \mathbb{R}^n , bod minima označ \mathbf{x}^{k+1} . V tomto bode musí platiť

$$\|\nabla_x \mathcal{L}_R(\mathbf{x}^{k+1}, \mathbf{y}^k)\| < \frac{L}{r^k} \|\tilde{\mathbf{y}} - \mathbf{y}^k\| + \max\{1, \sqrt{r^k}\|\mathbf{g}^k\|\}\varepsilon.$$

Krok 4 : Polož $\mathbf{y}^{k+1} = \text{multiplier}(\mathbf{x}^{k+1}, \mathbf{y}^k, odhad)$.

Krok 5 : Označ $\delta_k = \frac{\max_i \{-g_i(\mathbf{x}^{k+1})\}}{\max_i \{-g_i(\mathbf{x}^k)\}}$, AK $\delta_k > \delta$, POTOM $r^{k+1} = \min\{r_{\text{stp}}r^k, r\frac{\delta_k}{\delta}, r_{\max}\}$, INAK $r^{k+1} = r^k$.

Krok 6 : AK $\nu(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}) < \nu_{\text{crit}}$, POTOM $odhad = 2$.

Krok 7 : $k = k + 1$, CHOĎ NA KROK 2.

Pomocná funkcia $\text{multiplier}(\mathbf{x}, \mathbf{y}, odhad)$:

Krok 1 : AK $odhad = 1$, potom $\tilde{\mathbf{y}}_i = \max[0, \mathbf{y}_i - r g_i(\mathbf{x})]$, $i = 1, \dots, m$.

AK $odhad = 2$, potom

$$\tilde{\mathbf{y}} = \mathbf{y} - \left[\nabla_{\mathbf{y}\mathbf{y}} d_R(\mathbf{y}) \right]^{-1} \times \left[\nabla_{\mathbf{y}} d_R(\mathbf{y}) - \nabla g(\mathbf{x}) \left[\nabla_{\mathbf{x}\mathbf{x}}^2 \mathcal{L}_R(\mathbf{x}, \mathbf{y}) \right]^{-1} \nabla_x \mathcal{L}_R(\mathbf{x}, \mathbf{y}) \right],$$

a $\tilde{\mathbf{y}}_i = \max[0, \tilde{\mathbf{y}}_i]$, $i = 1, \dots, m$, pričom význam a hodnoty symbolov $\nabla_{\mathbf{y}} d_R(\mathbf{y})$ a $\nabla_{\mathbf{y}\mathbf{y}} d_R(\mathbf{y})$ nájdeme definované v časti 2.2.1.

Krok 2 : **Výstup :** $\tilde{\mathbf{y}}$.

Krátky komentár k uvedenej implementácii: Najväčšia výpočtová náročnosť algoritmu sa sústreďuje v kroku 3, v ktorom minimalizujeme Rockafellarovu rozšírenú Lagrangeovu funkciu, pričom terminačné kritérium je inšpirované vetou 2.2. Parameter ν_{crit} určuje, ktorú metódu odhadu multiplikátorov použijeme. Ak sa nachádzame

„ďaleko“ od optimálneho riešenia ($\nu(\mathbf{x}^k, \mathbf{y}^k)$ je veľké), použijeme odhad prvého rádu, pre malé hodnoty merit funkcie použijeme odhad druhého rádu. Parameter ν_{crit} teda určuje, kedy považujeme vzdialenosť za malú a kedy za veľkú. Pri experimentoch budeme používať dva varianty: v prvom zvolíme $\nu_{\text{crit}} = 0$ (vždy použijeme odhad prvého rádu), v druhom $\nu_{\text{crit}} = 1$ (použijeme oba odhady).

Parameter r meníme, ak algoritmus vykazuje veľmi pomalé znižovanie neprípustnosti bodov $(\mathbf{x}^k, \mathbf{y}^k)$. Ak pomer hodnôt neprípustnosti dvoch po sebe idúcich iterácií je väčší ako očakávaná hodnota δ , zmeníme r podľa vzťahu v kroku 5, inak ponecháme r bezo zmeny (pre detaily pozri [8]).

V experimentoch používame $r_{\text{max}} = 5 \cdot 10^5$, $r_{\text{stp}} = 6$, $\delta = 0.5$, $L = 0.9$, $r^0 = 10$.

4.2.2 Nelineárne škálovanie

Implementácia tohto algoritmu je veľmi podobná ako implementácia Rockafellarovho algoritmu. Pripomeňme, že uvažujeme

$$\mathcal{L}_{\text{NR}}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \frac{1}{r} \sum_{i=1}^m \mathbf{y}_i \psi(r g_i(\mathbf{x})), \quad (4.10)$$

kde sme zvolili

$$\psi(\xi) = \begin{cases} \xi^2 - \xi & \text{pre } \xi \leq 0, \\ \frac{1}{1+\xi} - 1 & \text{pre } \xi > 0. \end{cases} \quad (4.11)$$

Schematická implementáci algoritmu:

Krok 1 : Inicializácia. Urči konštanty ε (terminačné kritérium), $r_{\text{max}} < 10^6$, $r_{\text{stp}} \in [2, 10]$ (konštanty použité na zmenu parametra r), δ (očakávaný pokles porušenia prípustnosti), $0 < L < 1$, a urči hodnotu $r^0 > 0$. Polož $\mathbf{y}^0 = (1, \dots, 1)^T$ a $k = 0$.

Krok 2 : AK $\nu(\mathbf{x}^k, \mathbf{y}^k) < \varepsilon$, STOP, **Výstup :** $\mathbf{x}^k, \mathbf{y}^k$.

Krok 3 : Polož $\tilde{\mathbf{y}} = -\Psi'(r g(\mathbf{x}^k)) \mathbf{y}^k$, označ $\mathbf{g}^k = \|\nabla_x \mathcal{L}_{\text{NR}}(\mathbf{x}^k, \mathbf{y}^k)\|$ a použi Newtonovu metódu na približnú minimalizáciu funkcie $\mathcal{L}_{\text{NR}}(\cdot, \mathbf{y}^k)$ na \mathbb{R}^n , bod minima označ \mathbf{x}^{k+1} . V tomto bode musí platiť

$$\|\nabla_x \mathcal{L}_{\text{NR}}(\mathbf{x}^{k+1}, \mathbf{y}^k)\| < \frac{L}{r^k} \|\tilde{\mathbf{y}} - \mathbf{y}^k\| + \max\{1, \sqrt{r^k}\} \|\mathbf{g}^k\| \varepsilon.$$

Krok 4 : Polož $\mathbf{y}^{k+1} = -\Psi'(r g(\mathbf{x}^{k+1})) \mathbf{y}^k$.

Krok 5 : Označ $\delta_k = \frac{\max_i \{-g_i(\mathbf{x}^{k+1})\}}{\max_i \{-g_i(\mathbf{x}^k)\}}$, AK $\delta_k > \delta$, POTOM $r^{k+1} = \min\{r_{\text{stp}} r^k, r \frac{\delta_k}{\delta}, r_{\text{max}}\}$, INAK $r^{k+1} = r^k$.

Krok 6 : $k = k + 1$, CHOĎ NA KROK 2.

Vidíme, že schematický algoritmus je takmer totožný ako Rockafellarov algoritmus. Uvažujeme iba odhad multiplikátorov prvého rádu. Rovnako ako v prípade Rockafellarovho algoritmu používame $r_{\max} = 5 \cdot 10^5$, $r_{\text{stp}} = 6$, $\delta = 0.5$, $L = 0.9$, $r^0 = 10$.

4.2.3 Algoritmus PDNRD Polyaka a Grivu

Schematický popis uvedeného algoritmu Polyaka a Grivu sa bude od predošlých prípadov mierne odlišovať. V algoritme využijeme funkcie definované vzťahmi (4.10) a (4.11):

Krok 1 : Inicializácia. Urči konštanty ε (terminačné kritérium), $r_{\max} < 10^6$, $r_{\text{stp}} \in [2, 10]$ (konštanty použité na zmenu parametra r), δ (očakávaný pokles porušenia prípustnosti), $0 < L < 1$, $0 < \theta \leq 0.5$, a urči hodnotu $r^0 > 0$. Polož $\mathbf{y}^0 = (1, \dots, 1)^T$ a $k = 0$.

Krok 2 : AK $\nu(\mathbf{x}^k, \mathbf{y}^k) < \varepsilon$, STOP, **Výstup :** $\mathbf{x}^k, \mathbf{y}^k$.

Krok 3 : Nájdi $\Delta \mathbf{x}, \Delta \mathbf{y}$ zo sústavy

$$\begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}^k, \mathbf{y}^k) & -\nabla g(\mathbf{x}^k)^T \\ r\Psi''(rg(\mathbf{x}^k))\mathbf{Y}\nabla g(\mathbf{x}^k) & \mathbf{I}_m \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}^k, \mathbf{y}^k) \\ \tilde{\mathbf{y}} - \mathbf{y}^k \end{bmatrix}, \quad (4.12)$$

kde $\mathbf{Y} = \text{diag}(\mathbf{y}^k)$ a $\tilde{\mathbf{y}} = -\Psi'(rg(\mathbf{x}^k))\mathbf{y}^k$. Polož $\mathbf{x}_T^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}$, $\mathbf{y}_T^{k+1} = \mathbf{y}^k + \Delta \mathbf{y}$.

Krok 4 : AK $\nu(\mathbf{x}_T^{k+1}, \mathbf{y}_T^{k+1}) < \min\{1 - \theta, \nu(\mathbf{x}^k, \mathbf{y}^k)^{1.5-\theta}\}$, POTOM polož $\mathbf{x}^{k+1} = \mathbf{x}_T^{k+1}$, $\mathbf{y}^{k+1} = \mathbf{y}_T^{k+1}$ a $r^{k+1} = \nu(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})^{-1}$, $k = k + 1$, CHOĎ NA KROK 2.

Krok 5 : Polož $\tilde{\mathbf{y}} = -\Psi'(rg(\mathbf{x}^k))\mathbf{y}^k$, označ $\mathbf{g}^k = \|\nabla_x \mathcal{L}_{\text{NR}}(\mathbf{x}^k, \mathbf{y}^k)\|$ a použi Newtonovu metódu na približnú minimalizáciu funkcie $\mathcal{L}_{\text{NR}}(\cdot, \mathbf{y}^k)$ na \mathbb{R}^n , bod minima označ \mathbf{x}^{k+1} . V tomto bode musí platiť

$$\|\nabla_x \mathcal{L}_{\text{NR}}(\mathbf{x}^{k+1}, \mathbf{y}^k)\| < \frac{L}{r^k} \|\tilde{\mathbf{y}} - \mathbf{y}^k\| + \max\{1, \sqrt{r^k}\|\mathbf{g}^k\|\}\varepsilon.$$

Krok 6 : Polož $\mathbf{y}^{k+1} = -\Psi'(rg(\mathbf{x}^{k+1}))\mathbf{y}^k$.

Krok 7 : Označ $\delta_k = \frac{\max_i\{-g_i(\mathbf{x}^{k+1})\}}{\max_i\{-g_i(\mathbf{x}^k)\}}$, AK $\delta_k > \delta$, POTOM $r^{k+1} = \min\{r_{\text{stp}}r^k, r\frac{\delta_k}{\delta}, r_{\max}\}$, INAK $r^{k+1} = r^k$.

Krok 8 : $k = k + 1$, CHOĎ NA KROK 2.

Algoritmus je vlastne kombináciou primárno-duálneho algoritmu a nelineárneho škálovania. V každej iterácii sa pokúsime najprv spustiť primárno-duálnu metódu. Ak nový iteračný bod generovaný touto metódou dostatočne nezniží hodnotu merit funkcie (krok 4), znamená to, že sa ešte nachádzame príliš ďaleko od primárno-duálneho riešenia. Na získanie nového bodu $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$ potom použijeme klasickú nelineárne škálovaciu metódu tak, ako sme popísali v závere časti 3.2. Môžeme si všimnúť, že zo sústavy (4.12) vynechávame regularizačný člen (porovnaj s (3.18)). Dôvod je ten, že

všetky úlohy generované algoritmom z časti 4.1 spĺňajú $\nabla_{xx}^2 L(\mathbf{x}, \mathbf{y}) > 0$ pre ľubovoľné \mathbf{x} a $\mathbf{y} \geq 0$.

Parametre algoritmu v experimentoch volíme $r_{\max} = 5 \cdot 10^5$, $r_{\text{stp}} = 6$, $\delta = 0.5$, $L = 0.9$, $r^0 = 10$, $\theta = 0.35$.

4.2.4 Fischerove algoritmy

Algoritmy uvedené v časti 3.4 sú založené na použití Fischerovej funkcie na transformáciu KKT sústavy. Prvý algoritmus, „klasický“, bude založený výlučne na riešení tohto transformovaného systému.

Krok 1 : Inicializácia. Urči konštanty ε (terminačné kritérium), $0 < \mu$ (korekcia Fischerovej funkcie). Polož $\mathbf{y}^0 = (1, \dots, 1)^T$ a $k = 0$.

Krok 2 : AK $\nu(\mathbf{x}^k, \mathbf{y}^k) < \varepsilon$, STOP, **Výstup :** $\mathbf{x}^k, \mathbf{y}^k$.

Krok 3 : Nájdi $\Delta \mathbf{x}, \Delta \mathbf{y}$ zo sústavy

$$\begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}^k, \mathbf{y}^k) & -\nabla g(\mathbf{x}^k)^T \\ \mathbf{D}_\alpha \nabla g(\mathbf{x}^k) & \mathbf{D}_\beta \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}^k, \mathbf{y}^k) \\ -\Phi_{F\mu}(g(\mathbf{x}^k), \mathbf{y}^k) \end{bmatrix}, \quad (4.13)$$

kde presné definície diagonálnych matíc a funkcie $\Phi_{F\mu}$ nájdeme v časti 3.4.

Krok 4 : Nájdi približné riešenie λ^k úlohy

$$\lambda^k = \arg \min_{\lambda > 0} \{ \|F(\mathbf{w}^k + \lambda \mathbf{s}^k)\| \}, \quad F(\mathbf{w}) = \begin{bmatrix} \nabla_x L(\mathbf{x}, \mathbf{y}) \\ \Phi_{F\mu}(g(\mathbf{x}), \mathbf{y}) \end{bmatrix},$$

kde $\mathbf{w}^k = (\mathbf{x}^k, \mathbf{y}^k)$ a $\mathbf{s}^k = (\Delta \mathbf{x}, \Delta \mathbf{y})$.

Krok 5 : Polož $\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \Delta \mathbf{x}$, $\mathbf{y}_i^{k+1} = \max[0, \mathbf{y}_i^k + \lambda^k (\Delta \mathbf{y})_i]$, $i = 1, \dots, m$.

Krok 6 : $k = k + 1$, CHOĎ NA KROK 2.

Algoritmus je vlastne implementáciou Newtonovej metódy na riešenie sústavy rovníc (3.37), (3.38). Zároveň „cielené“ udržiavame body \mathbf{y}^k nezáporné a takisto upúšťame od regularizácie sústavy (4.13) (rovnaké dôvody ako pri algoritme PDNRD v predchádzajúcej časti). V experimentoch volíme $\mu = m_\varepsilon = 2^{-52}$.

Druhá implementácia Fischerovho algoritmu bude založená na kombinácii predošlého algoritmu a nelineárne škálovacieho algoritmu.

Krok 1 : Inicializácia. Urči konštanty ε (terminačné kritérium), $r_{\max} < 10^6$, $r_{\text{stp}} \in [2, 10]$ (konštanty použité na zmenu parametra r), δ (očakávaný pokles porušenia prípustnosti), $0 < L < 1$, $0 < \mu$ (korekcia Fischerovej funkcie), $0 < \nu_{\text{crit}} \leq 1$, $alg = 2$ a urči r^0 . Polož $\mathbf{y}^0 = (1, \dots, 1)^T$ a $k = 0$.

Krok 2 : AK $\nu(\mathbf{x}^k, \mathbf{y}^k) < \varepsilon$, STOP, **Výstup :** $\mathbf{x}^k, \mathbf{y}^k$.

Krok 3 : AK $alg = 1$, POTOM CHOĎ NA KROK 3, INAK CHOĎ NA KROK 6.

Krok 4 : Nájdi $\Delta \mathbf{x}, \Delta \mathbf{y}$ zo sústavy

$$\begin{bmatrix} \nabla_{xx}^2 L(\mathbf{x}^k, \mathbf{y}^k) & -\nabla g(\mathbf{x}^k)^T \\ \mathbf{D}_\alpha \nabla g(\mathbf{x}^k) & \mathbf{D}_\beta \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} -\nabla_x L(\mathbf{x}^k, \mathbf{y}^k) \\ -\Phi_{F\mu}(g(\mathbf{x}^k), \mathbf{y}^k) \end{bmatrix}, \quad (4.14)$$

kde presné definície diagonálnych matíc a funkcie $\Phi_{F\mu}$ nájdeme v časti 3.4.

Krok 5 : Polož $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}$, $\mathbf{y}_i^{k+1} = \max[0, \mathbf{y}_i^k + (\Delta \mathbf{y})_i]$, $i = 1, \dots, m$, $k = k + 1$, CHOĎ NA KROK 2.

Krok 6 : Polož $\tilde{\mathbf{y}} = -\Psi'(rg(\mathbf{x}^k))\mathbf{y}^k$, označ $\mathbf{g}^k = \|\nabla_x \mathcal{L}_{NR}(\mathbf{x}^k, \mathbf{y}^k)\|$ a použi Newtonovu metódu na približnú minimalizáciu funkcie $\mathcal{L}_{NR}(\cdot, \mathbf{y}^k)$ na \mathbb{R}^n , bod minima označ \mathbf{x}^{k+1} . V tomto bode musí platiť

$$\|\nabla_x \mathcal{L}_{NR}(\mathbf{x}^{k+1}, \mathbf{y}^k)\| < \frac{L}{r^k} \|\tilde{\mathbf{y}} - \mathbf{y}^k\| + \max\{1, \sqrt{r^k}\|\mathbf{g}^k\|\}\varepsilon.$$

Krok 7 : Polož $\mathbf{y}^{k+1} = -\Psi'(rg(\mathbf{x}^{k+1}))\mathbf{y}^k$.

Krok 8 : Označ $\delta_k = \frac{\max_i\{-g_i(\mathbf{x}^{k+1})\}}{\max_i\{-g_i(\mathbf{x}^k)\}}$, AK $\delta_k > \delta$, POTOM $r^{k+1} = \min\{r_{stp}r^k, r\frac{\delta_k}{\delta}, r_{max}\}$, INAK $r^{k+1} = r^k$.

Krok 9 : AK $\nu(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}) < \nu_{crit}$, POTOM polož $alg = 1$.

Krok 10 : $k = k + 1$, CHOĎ NA KROK 2.

Tento postup je vlastne heuristickým spojením dvoch odlišných algoritmov. V prvých iteráciách budeme najprv vykonávať klasický algoritmus nelineárneho škálovania, pokým nedostaneme dvojicu $\mathbf{x}^k, \mathbf{y}^k$, ktorej merit funkcia má menšiu hodnotu ako ν_{crit} . Potom prepneme na Fischerov algoritmus (v testoch volíme $\nu_{crit} = 0.1$, všetky ostatné konštanty volíme rovnako ako v pôvodných algoritmoch).

Ostatné parametre už tradične volíme $r_{max} = 5 \cdot 10^5$, $r_{stp} = 6$, $\delta = 0.5$, $L = 0.9$, $\mu = 2^{-52}$, $r^0 = 10$.

4.3 Výsledky experimentov

Po stručnom popísaní implementácie jednotlivých algoritmov môžeme pristúpiť k vykonaniu jednotlivých numerických experimentov. Budeme si všímať najmä vplyv zmeny veľkosti úlohy (vyjadrenú počtom premenných n a počtom ohraničení m), pomeru aktívnych ohraničení k ich celkovému počtu (t.j. bude nás zaujímať pomer $\frac{m_a}{m}$), pomeru kvadratických ohraničení (t.j. $\frac{q}{m}$) a nakoniec vplyv zmeny čísla podmienosti matíc (determinujúcich funkcie (4.1)) na kvalitu dosiahnutých riešení. Kvalitu budeme posudzovať týmito kritériami:

- (a) Absolútne odchýlky polohy dosiahnutých aproximácií $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ od známych optimálnych riešení $\hat{\mathbf{x}}, \hat{\mathbf{y}}$, t.j. budú nás zaujímať vzdialenosti $\|\bar{\mathbf{x}} - \hat{\mathbf{x}}\|$ a $\|\bar{\mathbf{y}} - \hat{\mathbf{y}}\|$.
- (b) Výsledná norma gradientu klasickej Lagrangeovej funkcie $\|\nabla_x L(\bar{\mathbf{x}}, \bar{\mathbf{y}})\|$ (označíme GRAD).
- (c) Porušenie komplementarity, vyjadrené pomocou $\sum_{i=1}^m |\bar{\mathbf{y}}_i g_i(\bar{\mathbf{x}})|$ (KOMP).
- (d) Porušenie prípustnosti, t.j. $\max \{ \max_i \{-g_i(\bar{\mathbf{x}})\}, \max_i \{-\bar{\mathbf{y}}_i\} \}$ (PRIP).
- (e) Časová náročnosť v sekundách.
- (f) Počet vykonaných makroiterácií (hodnota konštanty k vo všetkých algoritmoch).
- (g) Počet vykonaných mikroiterácií (počet iterácií vykonaných Newtonovou minimalizačnou metódou plus počet riešení špeciálnych sústav (4.12), (4.13) a (4.14)).

V každom experimente vykonáme pozorovania na sérii 10 náhodne generovaných úloh. Výsledky testov potom uvedieme v tabuľkách, v ktorých každý z riadkov bude obsahovať priemerné údaje z testovanej série. V tabuľkách budeme označovať implementované algoritmy skratkami:

AL_NR - Algoritmus rozšírenej Lagrangeovej funkcie používajúci funkcie nelineárneho škálovania.

AL_ROC1 - Algoritmus rozšírenej Lagrangeovej funkcie používajúci Rockafellarovu funkciu a odhad multiplikátorov prvého rádu.

AL_ROC2 - Algoritmus rozšírenej Lagrangeovej funkcie používajúci Rockafellarovu funkciu a kombinovaný odhad multiplikátorov (prvého aj druhého rádu).

FIS1 - Klasická verzia Fischerovho algoritmu.

FIS2 - Modifikovaná verzia Fischerovho algoritmu.

POL - Polyakov a Grivov algoritmus PDNRD.

Algoritmy sme testovali na počítači s procesorom Pentium(R) Dual-Core 2.10 GHz a s 3Gb operačnej pamäte typu DDR3, používajúci operačný systém Linux (distribúcia Ubuntu, v11.04).

Predtým, ako pristúpime k vykonaniu spomínaných experimentov, demonštrujeme funkčnosť algorimov na príklade malých rozmerov.

4.3.1 Ilustračný príklad

Na ilustráciu sme vybrali príklad zo známej zbierky testovacích úloh Hocka a Schittkowského (pôvodné zadania je možné nájsť v [42]), konkrétne príklad s označením HS65 (príklad malých rozmerov s $n = 3$ a $m = 7$). Jeho formulácia je nasledovná:

$$\begin{aligned} \text{minimalizovať} \quad & (\mathbf{x}_1 - \mathbf{x}_2)^2 + \frac{(\mathbf{x}_1 + \mathbf{x}_2 - 10)^2}{9} + (\mathbf{x}_3 - 5)^2 \\ \text{za podmienok} \quad & 48 - \mathbf{x}_1^2 - \mathbf{x}_2^2 - \mathbf{x}_3^2 \geq 0 \\ & -4.5 \leq \mathbf{x}_i \leq 4.5, \quad i = 1, 2 \\ & -5 \leq \mathbf{x}_3 \leq 5. \end{aligned} \tag{4.15}$$

Keďže analytický výpočet presného riešenia je pomerne náročný (vyžaduje riešiť polynómy štvrtého stupňa), v nasledujúcej tabuľke namiesto odchýliek polôh riešenia (kritérium **(a)**) uvedieme funkčnú hodnotu vo výslednom bode (označíme $\bar{f} \equiv f(\bar{\mathbf{x}})$). Použili sme terminačné kritérium $\varepsilon = 10^{-9}$, štartovací bod je určený ako $\mathbf{x}^0 = (-5, 5, 0)^T$.

HOCK & SCHITTKOWSKI, p. 65							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
AL_NR	9.535E-01	2.254E-11	1.330E-12	8.043E-270	4	11	0.052
AL_ROC1	9.535E-01	3.178E-11	3.502E-15	2.737E-231	4	11	0.049
AL_ROC2	9.535E-01	8.882E-16	8.756E-16	8.043E-270	3	10	0.048
FIS1	9.535E-01	3.528E-11	2.032E-14	2.309E-13	8	8	0.076
FIS2	9.535E-01	2.554E-15	2.382E-14	2.736E-13	7	11	0.037
POL	9.535E-01	3.748E-11	9.862E-31	8.043E-270	4	16	0.070

Tabuľka 4.1: Riešenie ilustračného príkladu

Vidíme, že všetky použité algoritmy konvergujú veľmi rýchlo k optimálnemu riešeniu úlohy. Metóda Rockafellara dosahuje presnejšie výsledky pri použití odhadu druhého rádu, čo sme intuitívne očakávali. Veľmi dobre sa ukazujú aj Fischerove metódy a takisto PDNRD algoritmus Polyaka a Grivu.

4.3.2 Test č. 1: Vplyv rozmerov úlohy

Prvý test, ktorý vykonáme, sa zameriava na súvis zmeny veľkosti úlohy (počet premenných n a počet ohraňení m) so zmenou výkonu algoritmov. Budeme si všimnúť najmä zmenu počtu iterácií potrebných na konvergenciu a takisto na presnosť dosiahnutých riešení. Nastavenie parametrov generátora bude nasledovné: $n =$

100, 200, 300, $m = 2n$, $m_a = 0.25m$, $q = 0.25m$, $\rho = 5n$ a $cond = 10^2$, ako tolerančné kritérium zvolíme $\varepsilon = 10^{-6}$. V tabuľke sumerizujeme výsledky:

TEST č. 1, VÝSLEDKY								
ALG	$\ \bar{x} - \hat{x}\ $	$\ \bar{y} - \hat{y}\ $	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
$n = 100, m = 200, m_a = 50, m_q = 50, \rho = 500$								
AL_NR	1.451E-09	2.994E-06	2.034E-07	5.044E-07	3.633E-08	10.10	32.50	2.639
AL_ROC1	5.000E-10	1.075E-06	5.438E-08	2.800E-07	4.298E-09	14.30	30.40	2.273
AL_ROC2	3.061E-10	8.144E-07	1.589E-08	2.566E-07	5.169E-09	6.90	24.60	1.846
FIS1	2.311E-12	4.546E-09	1.310E-10	5.028E-10	1.298E-10	19.80	19.80	3.754
FIS2	1.084E-13	2.071E-10	1.164E-10	2.434E-11	7.105E-12	8.60	28.10	2.215
POL	2.026E-11	3.752E-08	5.821E-11	1.441E-08	7.388E-10	9.20	30.20	2.122
$n = 200, m = 400, m_a = 100, m_q = 100, \rho = 1000$								
AL_NR	1.236E-09	4.694E-06	1.180E-06	8.743E-07	5.553E-08	11.10	35.30	23.668
AL_ROC1	1.404E-10	3.264E-07	4.036E-07	2.191E-07	2.166E-09	14.40	35.70	21.149
AL_ROC2	2.462E-11	1.663E-11	1.198E-06	4.752E-07	6.705E-09	8.10	30.10	16.846
FIS1	1.103E-12	2.824E-09	1.164E-10	4.970E-10	6.435E-11	21.90	21.90	31.987
FIS2	1.510E-14	1.908E-11	3.492E-10	9.862E-11	3.638E-12	8.70	32.40	17.184
POL	2.241E-11	5.145E-08	2.328E-10	1.305E-08	1.226E-09	9.20	33.00	16.991
$n = 300, m = 600, m_a = 150, m_q = 150, \rho = 1500$								
AL_NR	5.005E-10	1.865E-06	2.426E-06	6.524E-07	2.846E-08	11.30	37.50	226.601
AL_ROC1	1.021E-10	2.304E-07	5.632E-07	2.728E-07	1.826E-09	14.50	39.20	179.374
AL_ROC2	4.320E-11	3.214E-11	1.502E-06	5.190E-07	4.158E-09	8.40	32.80	141.009
FIS1	2.486E-09	1.543E-05	4.064E-07	8.822E-07	4.397E-07	23.30	23.30	253.049
FIS2	1.181E-13	6.614E-10	3.492E-10	2.447E-10	1.762E-11	8.30	34.70	162.137
POL	4.920E-11	1.577E-07	2.328E-10	7.108E-08	5.298E-09	9.10	33.80	154.684

Tabuľka 4.2: Súhrnné výsledky pre test č. 1

Výsledky do veľkej miery potvrdzujú naše očakávania. Príjemným konštatovaním je fakt, že požadovaná presnosť bola dosiahnutá vo všetkých prípadoch, odchýlky presnosti polohy \bar{x} a \bar{y} sú takisto veľmi dobré. V tomto smere sa sľubne ukazujú metódy študované v Kapitole 3, čiže PDNRD algoritmus Polyaka a Grivu a obe modifikácie Fischerovho algoritmu. Všimnúť si môžeme aj vplyv použitia odhadu multiplikátorov druhého rádu na presnosť vektora multiplikátorov \bar{y} oproti použitiu odhadu prvého rádu v prípade Rockafellarovho algoritmu.

Zväčšujúci sa rozmer úlohy však viac alebo menej vplýva na počet vykonaných iterácií, ale najmä na časovú náročnosť. Počty hlavných iterácií sa zväčšujú len nepatrne, sumárny počet Newtonovských iterácií stúpol najviac v prípade Rockafellarovho algoritmu (2nd order update), konkrétne o 33%. Rápidny vzrast si všimneme

v prípade časovej náročnosti. Tento je spôsobený jednak zmenou rozmerov úlohy, a takisto zväčšujúcim sa počtom kvadratických ohraničení, ktoré je nutné vyčíslovať v každej iterácii.

4.3.3 Test č. 2: Zmena počtu ohraničení

V tomto teste sa zameriame výlučne na zmenu počtu ohraničení, kým počet premenných ponecháme konštantný. Budeme uvažovať $n = 100$ a postupne budeme voliť $m = 100, 300, 500$. Ku každej kombinácii potom priradíme ostatné parametre generátora predpisom $q = 0.25m, m_a = 0.6n, \rho = 5n$ a $cond = 10^2$. Podobne ako v predošlom teste zvolíme $\varepsilon = 10^{-6}$.

TEST č. 2, VÝSLEDKY								
ALG	$\ \bar{x} - \hat{x}\ $	$\ \bar{y} - \hat{y}\ $	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
$n = 100, m = 100, m_a = 60, m_q = 25, \rho = 500$								
AL_NR	1.110E-09	3.323E-06	2.539E-07	3.649E-07	2.095E-08	11.40	32.50	1.485
AL_ROC1	6.008E-10	1.567E-06	1.321E-07	3.673E-07	4.690E-09	14.20	30.40	1.246
AL_ROC2	3.190E-10	5.275E-07	1.345E-06	2.254E-07	3.138E-09	7.80	24.20	1.015
FIS1	1.717E-09	4.834E-06	1.328E-07	1.535E-07	7.430E-08	18.30	18.30	1.971
FIS2	4.410E-12	1.081E-08	4.578E-09	1.970E-09	2.735E-10	7.70	25.90	1.104
POL	2.492E-10	5.010E-07	1.110E-10	1.173E-07	8.925E-09	9.50	29.40	1.100
$n = 100, m = 300, m_a = 60, m_q = 75, \rho = 1000$								
AL_NR	3.665E-08	2.948E-05	9.813E-06	4.209E-07	2.045E-08	11.30	34.70	4.175
AL_ROC1	7.116E-10	2.058E-06	9.482E-08	5.649E-07	7.176E-09	13.80	32.80	3.596
AL_ROC2	7.545E-11	1.490E-07	4.058E-07	9.406E-08	1.564E-09	8.50	27.60	3.141
FIS1	4.559E-09	5.459E-06	8.165E-06	1.883E-07	9.350E-08	21.30	21.30	6.424
FIS2	1.236E-11	2.643E-08	6.275E-08	1.864E-08	9.334E-10	8.00	28.90	3.210
POL	2.290E-09	5.618E-06	2.832E-10	3.725E-07	3.503E-08	9.00	31.40	3.193
$n = 100, m = 500, m_a = 60, m_q = 125, \rho = 1000$								
AL_NR	2.967E-09	6.626E-06	3.704E-07	2.999E-07	1.271E-08	11.40	34.20	6.481
AL_ROC1	7.771E-10	2.024E-06	7.080E-08	5.149E-07	8.199E-09	13.80	32.50	5.654
AL_ROC2	9.382E-11	1.723E-07	3.505E-08	9.180E-08	1.188E-09	8.00	27.10	4.952
FIS1	1.289E-09	2.125E-06	3.307E-06	2.125E-07	6.695E-08	22.90	22.90	11.915
FIS2	4.189E-09	5.152E-06	6.274E-05	1.545E-07	6.452E-09	8.40	28.80	5.185
POL	6.019E-10	1.284E-06	2.197E-10	2.053E-07	1.566E-08	9.70	31.30	5.081

Tabuľka 4.3: Súhrnné výsledky pre test č. 2

Vidíme, že presnosť dosiahnutých riešení primárnej premennej \bar{x} aj duálnej premennej \bar{y} je v každom z nastavení parametrov takmer rovnaká. Zaujímavá je najmä

takmer rovnaká dosiahnutá presnosť pre $\bar{\mathbf{y}}$, keďže práve počet multiplikátorov sa v rámci testu zvyšuje. Všetky algoritmy produkujú porovnateľne presné riešenia, hoci v niektorých prípadoch nedosahuje norma gradientu Lagrangeovej funkcie požadovanú presnosť.

Časová náročnosť má podľa očakávania rastúcu tendenciu, keďže viac ohraničení znamená viac ich vyčíslení. Nárast však nie je taký markantný ako v prípade predchádzajúceho testu. Situácia ohľadom počtu vykonaných iterácií nie je jednoznačná: v niektorých prípadoch pozorujeme mierny nárast, v iných prípadoch stagnáciu či dokonca mierny pokles. Prevažujúci trend je však mierne rastúci.

V rámci tohto testu teda uspeli všetky porovnávané algoritmy, čo je uspokojivé.

4.3.4 Test č. 3: Zmena počtu aktívnych ohraničení

V doterajších testoch sme sa príliš nezameriavali na pomer počtu aktívnych ohraničení k ich celkovému počtu. V prvom teste bol tento pomer konštantný, v druhom sa vplyvom zmeny počtu ohraničení nepriamo menil. V tomto teste sa zameriame výlučne na skúmanie vplyvu daného pomeru na iteračnú a časovú náročnosť, a takisto na presnosť dosiahnutých riešení. Zvolíme $n = 100, m = 100, q = 50, \rho = 500, cond = 10^2$ a budeme uvažovať pomery $\frac{m_a}{m} = 0.1, 0.4, 0.7, 0.95$. Tradične budeme voliť $\varepsilon = 10^{-6}$.

Z výsledkov uvedených v tabuľke 4.4 opäť vidíme, že dosiahnutá presnosť či už primárnej premennej $\bar{\mathbf{x}}$ alebo duálnej premennej $\bar{\mathbf{y}}$ je vo všetkých skúmaných scenároch takmer rovnaká. Mierny pokles si môžeme všimnúť až pri nastavení $\frac{m_a}{m} = 0.95$.

Práve v tomto teste je najlepšie vidieť rast počtu iterácií, či už hlavných iterácií alebo sumárneho počtu Newtonovských iterácií. Pri zvyšovaní počtu aktívnych ohraničení sa stále viac ohraničení realizuje v optimálnom bode ako rovnosti, a tento fakt zrejme sťažuje konvergenciu algoritmov. Najlepšie je tento fakt vidieť pri prechode $\frac{m_a}{m} : 0.7 \rightarrow 0.95$. Pri tomto nastavení v niektorých prípadoch mierne klesá dosiahnutá presnosť vektora multiplikátorov a takisto normy gradientu Lagrangeovej funkcie. Priamym dôsledkom zvyšovania počtu iterácií je aj zvýšenie časovej náročnosti.

TEST č. 3, VÝSLEDKY								
ALG	$\ \bar{x} - \hat{x}\ $	$\ \bar{y} - \hat{y}\ $	GRAD	KOMP	PRIP	počet iterací	\sum mikroit.	časová náročnosť
$n = 100, m = 100, m_a = 10, q = 50, \rho = 1000$								
AL_NR	7.865E-08	1.354E-05	1.889E-06	5.512E-07	1.917E-08	10.40	30.10	2.467
AL_ROC1	1.629E-09	6.107E-07	3.148E-09	4.634E-07	4.068E-08	11.50	27.90	2.096
AL_ROC2	5.889E-10	1.462E-07	6.425E-07	1.408E-07	1.222E-08	6.00	22.50	1.760
FIS1	1.518E-11	2.178E-09	7.233E-09	7.272E-09	1.441E-09	16.90	16.90	3.096
FIS2	4.585E-13	5.916E-11	5.937E-10	4.540E-10	4.354E-11	5.80	24.60	1.836
POL	8.523E-10	2.339E-07	5.251E-10	1.743E-07	2.600E-08	6.30	25.30	1.816
$n = 100, m = 100, m_a = 40, q = 50, \rho = 1000$								
AL_NR	1.940E-09	3.252E-06	1.344E-07	4.551E-07	3.934E-08	10.40	31.90	2.548
AL_ROC1	5.438E-10	8.460E-07	3.024E-08	4.007E-07	1.111E-08	13.10	29.90	2.137
AL_ROC2	1.268E-10	1.880E-07	1.005E-08	1.186E-07	2.424E-09	7.70	24.70	1.807
FIS1	1.961E-10	2.101E-07	1.779E-08	8.152E-08	8.718E-10	18.60	18.60	3.359
FIS2	3.309E-12	1.830E-09	2.856E-08	1.733E-08	3.879E-10	7.20	26.60	1.958
POL	2.433E-10	3.228E-07	1.135E-10	9.221E-08	1.142E-08	8.40	28.10	1.887

Tabuľka 4.4: Súhrnné výsledky pre test č. 3

TEST č. 4, VÝSLEDKY								
ALG	$\ \bar{x} - \hat{x}\ $	$\ \bar{y} - \hat{y}\ $	GRAD	KOMP	PRIP	počet iterací	\sum mikroit.	časová náročnosť
$n = 100, m = 100, m_a = 70, q = 50, \rho = 1000$								
AL_NR	1.290E-09	4.342E-06	8.152E-07	3.547E-07	1.730E-08	11.10	34.10	2.678
AL_ROC1	7.262E-10	2.410E-06	1.971E-07	4.204E-07	7.085E-09	14.10	33.00	2.311
AL_ROC2	6.470E-10	8.283E-07	7.048E-06	4.872E-07	6.288E-09	7.60	26.20	1.893
FIS1	6.411E-10	1.641E-06	3.309E-07	1.388E-07	2.225E-08	19.60	19.60	3.605
FIS2	1.104E-09	2.286E-06	6.493E-05	2.931E-08	1.393E-09	8.00	28.50	2.066
POL	2.418E-10	6.405E-07	1.019E-10	7.732E-08	7.827E-09	9.90	31.40	2.025
$n = 100, m = 100, m_a = 95, q = 50, \rho = 1000$								
AL_NR	4.120E-09	7.758E-05	2.381E-05	4.426E-07	1.461E-08	13.60	39.70	3.099
AL_ROC1	2.210E-09	4.007E-05	7.493E-06	3.723E-07	3.827E-09	15.60	36.70	2.499
AL_ROC2	1.367E-09	8.513E-06	2.332E-05	2.993E-07	3.149E-09	9.00	28.60	2.031
FIS1	1.466E-09	1.563E-05	4.061E-08	1.208E-07	1.723E-08	30.50	30.50	5.486
FIS2	3.132E-10	2.665E-06	6.430E-07	6.247E-08	6.630E-09	8.60	29.10	2.102
POL	2.836E-09	5.838E-05	2.119E-05	3.335E-07	1.352E-08	14.00	39.10	2.417

Tabuľka 4.4: Súhrnné výsledky pre test č. 3, pokračovanie

4.3.5 Test č. 4: Zmena počtu kvadratických ohraničení

Teraz sa zameriame na sledovanie vplyvu pomeru kvadratických ohraničení k celkovému počtu ohraničení. Dá sa očakávať, že úloha s väčším podielom nelineárnych

funkcií bude ťažšie riešiteľná. Za týmto účelom zafixujeme $n = 100, m = 100, m_a = 40, \rho = 500, cond = 10^2$, postupne budeme voliť $q = 10, 30, 70, 95$ a uvidíme, či sa naše domnienky potvrdia.

TEST č. 4, VÝSLEDKY								
ALG	$\ \bar{x} - \hat{x}\ $	$\ \bar{y} - \hat{y}\ $	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
$n = 100, m = 100, m_a = 50, q = 10, \rho = 1000$								
AL_NR	1.597E-09	3.247E-06	1.254E-07	4.973E-07	2.563E-08	10.90	29.10	0.749
AL_ROC1	7.266E-10	1.444E-06	3.559E-08	5.284E-07	7.750E-09	13.50	28.30	0.698
AL_ROC2	2.957E-10	9.136E-06	1.084E-06	3.831E-07	2.988E-08	7.40	22.40	0.536
FIS1	6.925E-10	1.588E-06	4.069E-08	1.448E-07	3.371E-08	16.90	16.90	1.023
FIS2	2.956E-12	6.128E-09	1.717E-10	2.725E-10	1.507E-10	7.40	23.00	0.552
POL	7.880E-10	1.509E-06	1.019E-10	2.199E-07	3.608E-08	8.70	26.00	0.553
$n = 100, m = 100, m_a = 50, q = 40, \rho = 1000$								
AL_NR	1.564E-09	3.054E-06	2.606E-07	4.911E-07	3.215E-08	10.30	32.20	2.145
AL_ROC1	7.809E-10	1.282E-06	5.090E-08	5.295E-07	1.016E-08	13.20	30.30	1.786
AL_ROC2	8.742E-11	1.130E-07	1.313E-08	7.994E-08	1.321E-09	7.90	25.50	1.604
FIS1	5.972E-10	1.101E-06	6.025E-08	2.262E-07	4.484E-09	17.90	17.90	2.760
FIS2	4.141E-11	5.472E-08	5.219E-08	2.394E-08	2.212E-09	7.30	26.30	1.607
POL	2.334E-10	2.888E-07	1.048E-10	6.979E-08	6.905E-09	8.80	29.80	1.638
$n = 100, m = 100, m_a = 50, q = 70, \rho = 1000$								
AL_NR	1.157E-09	2.453E-06	2.575E-07	3.424E-07	3.415E-08	10.40	32.10	3.532
AL_ROC1	7.036E-10	1.446E-06	5.795E-08	5.515E-07	1.578E-08	13.00	30.40	3.196
AL_ROC2	7.041E-11	1.152E-07	1.856E-08	6.677E-08	1.852E-09	7.90	25.30	2.569
FIS1	1.896E-10	2.416E-07	2.196E-08	1.003E-07	2.734E-09	18.40	18.40	4.864
FIS2	1.821E-11	2.565E-08	1.175E-07	5.222E-08	1.139E-09	7.00	26.70	2.742
POL	6.808E-10	9.970E-07	1.426E-10	1.790E-07	2.840E-08	8.20	28.80	2.930
$n = 100, m = 100, m_a = 50, q = 95, \rho = 1000$								
AL_NR	1.655E-09	1.376E-06	5.020E-08	3.736E-07	3.770E-08	10.00	32.20	4.577
AL_ROC1	7.738E-10	1.096E-06	2.468E-08	4.672E-07	3.109E-08	12.10	29.70	3.821
AL_ROC2	8.417E-11	1.111E-07	1.256E-08	1.046E-07	2.687E-09	6.50	24.20	3.359
FIS1	8.125E-11	8.729E-08	6.015E-07	7.058E-08	3.353E-09	18.80	18.80	6.329
FIS2	1.305E-13	1.909E-10	4.540E-10	4.197E-10	8.728E-12	5.90	26.20	3.520
POL	5.942E-10	7.886E-07	3.347E-10	1.448E-07	2.981E-08	6.90	27.70	3.500

Tabuľka 4.5: Súhrnné výsledky pre test č. 4

Výsledky testu sú viac ako zaujímavé. Ukazuje sa totiž, že okrem zvyšovania časovej náročnosti (čo sa očakávalo, keďže vyčíslenie kvadratických ohraničení je náročnejšie ako lineárnych) nedochádza k žiadnej výraznej zmene vo všetkých ostatných sledovaných veličinách. Pri použití metód využívajúcich informácie druhého rádu však v prípade kvadratických funkcií nedochádza k strate informácie. To je

jedno z možných vysvetlení výsledkov.

Dosiahnuté výsledky sú opäť veľmi presné, všetky algoritmy skonvergovali k dostatočne presným aproximáciám skutočných riešení, pričom počet iterácií zostal relatívne malý. Uvidíme, či sa situácia zmení, ak na testovanie použijeme úlohy, kde Hessove matice funkcií týchto úloh budú zle podmienené.

4.3.6 Test č. 5: Zmena čísla podmienenosti

Na záver ešte vykonáme test na zle podmienených úlohách. Zafixujeme preto $n = 100, m = 200, m_a = 80, q = 50, \rho = 500$ a budeme voliť rôzne čísla podmienenosti matíc vystupujúcich v definíciách funkcií (4.1), konkrétne vyskúšame $cond = 10, 10^6, 10^{12}$. Výsledky sumarizuje nasledujúca tabuľka.

TEST č. 5, VÝSLEDKY								
ALG	$\ \bar{x} - \hat{x}\ $	$\ \bar{y} - \hat{y}\ $	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
$n = 100, m = 200, m_a = 80, q = 50, \rho = 1000, cond = 10$								
AL_NR	1.624E-09	3.347E-06	1.587E-06	4.086E-07	1.421E-08	11.30	40.30	3.210
AL_ROC1	1.243E-09	2.846E-06	4.636E-07	5.089E-07	5.311E-09	14.20	39.30	2.844
AL_ROC2	6.874E-10	1.035E-06	5.768E-07	4.903E-07	5.132E-09	7.60	31.60	2.394
FIS1	3.263E-10	5.064E-07	2.467E-08	6.824E-08	5.679E-09	25.30	25.30	4.924
FIS2	1.894E-12	3.896E-09	2.101E-10	3.111E-10	7.325E-11	8.50	34.80	2.567
POL	4.273E-10	7.497E-07	7.331E-11	1.461E-07	7.603E-09	10.40	37.20	2.485
$n = 100, m = 200, m_a = 80, q = 50, \rho = 1000, cond = 10^6$								
AL_NR	1.519E-09	7.074E-05	9.180E-05	4.364E-07	1.575E-08	12.90	33.70	2.544
AL_ROC1	1.276E-09	5.674E-05	8.837E-07	5.920E-07	6.178E-09	15.10	29.60	2.057
AL_ROC2	1.144E-10	1.204E-05	1.656E-03	1.934E-07	3.638E-09	8.90	23.20	1.666
FIS1	5.426E-10	1.735E-05	4.734E-08	1.487E-07	1.833E-08	27.50	27.50	5.407
FIS2	8.476E-12	1.415E-07	2.055E-06	1.226E-08	3.414E-10	9.50	25.80	1.764
POL	6.868E-10	2.703E-05	7.451E-10	1.588E-07	7.110E-09	14.20	35.20	2.089
$n = 100, m = 200, m_a = 80, q = 50, \rho = 1000, cond = 10^{12}$								
AL_NR	1.590E-09	3.039E-04	8.654E-04	3.712E-07	8.150E-09	13.00	32.20	2.428
AL_ROC1	8.974E-10	1.685E-04	2.968E-06	4.074E-07	3.976E-09	15.70	30.70	2.090
AL_ROC2	1.895E-11	3.217E-06	2.636E-04	2.681E-08	5.238E-10	8.60	21.30	1.517
FIS1	4.147E-11	5.254E-06	2.500E-09	9.421E-09	1.084E-09	31.90	31.90	6.368
FIS2	2.528E-12	4.415E-07	2.073E-08	1.777E-10	1.330E-10	10.00	24.90	1.650
POL	1.252E-09	2.612E-04	6.925E-04	3.870E-07	8.808E-09	15.50	36.00	2.185

Tabuľka 4.6: Súhrnné výsledky pre test č. 5

Výsledky sú opäť zaujímavé. Výrazné zvýšenie čísla podmienenosti nevedlo v

niektorých prípadoch k takmer žiadnemu zhoršeniu výkonu algoritmov. Zhoršenie podmienosti matíc determinujúcich funkcie úlohy toriž nemusí automaticky znamenať zhoršenie podmienosti Hessovej matice klasickej či rozšírenej Lagrangeovej funkcie.

Zaznamenali sme mierny pokles presnosti aproximácií multiplikátorov $\bar{\mathbf{y}}$ (napr. AL_NR alebo POL), a takisto veľkosť normy gradientu Lagrangeovej funkcie v niektorých prípadoch nedosahuje požadovanú presnosť 10^{-6} . Súhrnne v tomto teste celkom dobre dopadli Fisherove algoritmy.

4.3.7 Zbierka CUTEr

Na záver vykonáme testy na niekoľkých vybraných úlohách zo zbierky príkladov CUTEr [18]. Jedná sa o zborník obsahujúci príklady na voľnú aj viazanú optimalizáciu. Všeobecný tvar úloh obsiahnutých v tomto zborníku je

$$\begin{aligned} & \text{minimalizovať} && f(\mathbf{x}) \\ & \text{za podmienok} && \mathbf{b}_i \leq g_i(\mathbf{x}) \leq \mathbf{c}_i, && i = 1, \dots, m \\ & && h_i(\mathbf{x}) = \mathbf{d}_i, && i = 1, \dots, p \\ & && \boldsymbol{\alpha}_i \leq \mathbf{x}_i \leq \boldsymbol{\beta}_i, && i = 1, \dots, n. \end{aligned} \tag{4.16}$$

pre $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ($i = 1, \dots, m$) a $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ($i = 1, \dots, p$). Niektoré prvky vektorov \mathbf{b} , $\boldsymbol{\alpha}$ môže byť $-\infty$ a analogicky môže byť niektoré prvky \mathbf{c} , $\boldsymbol{\beta}$ rovné ∞ . Úlohy vykazujú rôzne stupne nelinearity, zbierka taksito obsahuje množstvo nekonvexných úloh.

Naše testovanie zameriame na niektoré vybrané úlohy konvexného programovania. Aby sme mohli lepšie využiť štruktúru úlohy 4.16, rozšírili sme algoritmy z časti (4.2) aj pre zvládnutie ohraničení v tvare rovností. Keďže toto rozšírenie je pomerne priamočiare, preto detaily na tomto mieste uvádzať nebudeme. Výsledky numerického experimentu sú uvedené v druhej časti Prílohy a zdrojové kódy portu algoritmov na testovanie CUTEr úloh sú obsiahnuté na priloženom médiu.

Väčšinu úloh dokázali vyriešiť všetky algoritmy. Klasický Fischerov algoritmus mal spomedzi testovaných algoritmov najväčšie problémy. Naopak najlepšie si viedli PDNRD algoritmus Polyaka a Grivu a modifikovaný Fischerov algoritmus.

Záver

V tejto práci sme sa venovali teoretickému popisu a implementácii algoritmov na riešenie úlohy konvexného programovania (\mathcal{PNR}_c) a ich experimentálnemu porovnaniu. Prevažnú časť algoritmov tvorili metódy rozšírených Lagrangeových funkcií, konkrétne Rockafellarov algoritmus (časť 2.2) a všeobecný algoritmus nelineárneho škálovania (2.3.2 a 3.1). Spolu s uvedenými algoritmami sme študovali primárnoduálny nelineárne škálovací algoritmus (PDNRD, časti 3.2, 3.3) Polyaka a Grivu a dva varianty Fischerovho algoritmu (časť 3.4). Algoritmy sme implementovali prostredie MATLAB, pričom využívame mex funkcie (skompilovaný kód jazyka C) a procedúry knižnice BLAS v snahe urýchliť niektoré výpočty. Ukážky zdrojových kódov sú priložené v Prílohe.

Tieto algoritmy sme podrobili testovaniu na sériach náhodne generovaných úloh bikvadratického programovania a takisto na vybraných úlohách zo známeho testovacieho prostredia CUTer.

Na testoch pozostávajúcich z náhodne generovaných úloh dosahovali všetky algoritmy podobné výsledky, dosiahnutá presnosť riešení bola postačujúca. Môžeme si však všimnúť zväčšenú presnosť a miernu iteračnú a časovú úsporu algoritmov PDNRD, modifikovaného Fischerovho algoritmu a Rockafellarovho algoritmu s použitím odhadu multiplikátorov druhého rádu v porovnaní s ostatnými metódami. Prístup kombinovania klasických algoritmov rozšírených Lagrangeových funkcií a riešenia špeciálnych sústav v záverečnej časti výpočtového procesu, ktoré PDNRD a modifikovaný Fischerov algoritmus využívajú, sa zdá byť užitočný. Takisto sa prejavuje vplyv odhadu multiplikátorov druhého rádu.

V testovaniach na príkladoch zo zborníka CUTer si najlepšie viedli klasický algoritmus nelineárneho škálovania, PDNRD algoritmus a modifikovaný Fischerov algoritmus, ktoré vyriešili takmer všetky úlohy. Veľkú väčšinu úloh však vyriešili veľmi presne všetky testované algoritmy.

Niektoré dôležité otázky zostávajú zatiaľ nezodpovedané: porovnanie teoretickej rýchlosti konvergenzie PDNRD algoritmu s Fischerovým algoritmom a prípadná reformulácia týchto algoritmov na riešenie úloh nekonvexného programovania. V týchto oblastiach je priestor pre ďalší výskum.

Literatúra

- [1] Bertsekas D. P., *Constrained Optimization and Lagrange Multiplier Methods*, Athena Scientific, 1996
- [2] Bertsekas D. P., Nedić A., Ozdaglar A. E., *Convex Analysis and Optimization*, Athena Scientific, Massachusetts, 560 s., 2003
- [3] Bertsekas D. P., *Nonlinear Programming*, Athena Scientific, 2nd edition, 1999
- [4] Birgin E. G., Castillo R. A., Martínez J. M., *Numerical Comparison of Augmented Lagrangian Algorithms for Nonconvex Problems*, Computational Optimization and Applications, Vol. 31, No. 1, s. 31-55, 2005.
- [5] Boyd S., Vanderberghe L., *Convex Optimization*, Cambridge University Press, 730 s., 2004
- [6] Brent R. P., *Algorithms for Minimization Without Derivatives*, Prentice-Hall, Eaglewood Cliffs, N. J., 208 s., 1973
- [7] Byrd R. H., *Local Convergence of the Diagonalized Method of Multipliers*, Journal of Optimization Theory and Applications, Vol. 26, s. 485-500, 1978
- [8] Delbos F., Gilbert J. Ch., *Global Linear Convergence of an Augmented Lagrangian Algorithm to Solve Convex Quadratic Optimization Problems*, Journal of Convex Analysis, Volume 12, No. 1, s. 45-69, 2005
- [9] Dennis J. E., Schnabel R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Society for Industrial Mathematics, 394 s., 1987
- [10] Fletcher R., *Practical Methods Of Optimization*, 2nd Edition, John Wiley & Sons, 450 s., 1987

- [11] Facchinei F., Fischer A., Kanzow C., *A Semismooth Newton Method For Variational Inequalities, The Case of Box Constraints*, Complementarity and Variational Problems: State of the Art. SIAM, s. 76-90, 1997
- [12] Facchinei F., Fischer A., Kanzow C., Peng J., *A Simply Constrained Optimization Reformulation of KKT Systems Arising From Variational Inequalities*, Applied Mathematics & Optimization, Vol. 40, No. 1, s. 19-37, 1999
- [13] Facchinei F., Soares J., *Testing a New Class of Algorithms for Nonlinear Complementarity Problems*, DIS Technical Report 15.94, 1994
- [14] Fischer A., *A Special Newton-Type Optimization Method*, Optimization, Vol. 24, s. 269-284, 1992
- [15] Fischer A., *A Newton-type Method for Positive-Semidefinite Linear Complementarity Problems* Journal of Optimization Theory and Applications, Vol. 86, No. 3, s. 585-608, 1995
- [16] Fontecilla R., Steihaug T., Tapia R. A., *Convergence Theory for a Class of Quasi-Newton Methods for Constrained Optimization*, SIAM Journal on Numerical Analysis, Vol. 24, No. 5, s. 1133-1151, 1987
- [17] Gill P. E., Murray W., Wright M. H., *Practical Optimization*, Academic Press, London, 402 s., 1981
- [18] Gould N., Orban D., Toint P. L., *CUTEr and SifDec: A Constrained and Unconstrained Testing Environment, Revisited*, ACM Transactions on Mathematical Software, Vol. 29, No. 4, s. 373-394, 2003
- [19] Griva I., *Numerical Experiments With an Interior-Exterior Point Method for Nonlinear Programming*, Computational Optimization and Applications, Vol. 29, No. 2, s. 173-195, 2004
- [20] Hamala M., *Nelineárne Programovanie*, Alfa, Bratislava, 2. vydanie, 1976.
- [21] Hamala M., *Prednášky z Nelineárneho programovania*, 2009/2010.
- [22] Hestenes M. R., *Multiplier and Gradient Methods*, Journal of Optimization Theory and Applications, Vol. 4, s. 303-320, 1969.

- [23] Kelley C., *Solving Nonlinear Equations With Newton's Method*, Society for Industrial and Applied Mathematics, 119 s., 2003
- [24] Luenberger D. G., Ye Y., *Linear and Nonlinear Programming*, 3rd edition, Springer, 2008.
- [25] More J., Thuente D., *Line Search Algorithms With Guaranteed Sufficient Decrease*, ACM Transactions on Mathematical Software, Vol. 20, No. 3, s. 286-307, 1994
- [26] Nocedal J., Wright S., *Numerical Optimization*, Springer, 656 s., 2000
- [27] Peng J., Roos C., Terlaky T., *A Logarithmic Barrier Approach to Fischer Function*, Nonlinear Optimization and Applications, Vol. 2, s. 277-298, 1999
- [28] Polyak R., *Modified Barrier Functions (Theory and Methods)*, Mathematical Programming, Vol. 54, s. 177-222, 1992
- [29] Polyak R., *On the Local Quadratic Convergence of the Primal-Dual Augmented Lagrangian Method*, Optimization Methods and Software, Vol. 24, No. 3, s. 369-379, 2009
- [30] Polyak R., Griva I., *1.5-Q-Superlinear Convergence of an Exterior-Point Method for Constrained Optimization*, Journal of Global Optimization, Vol. 40, No. 4, 2008
- [31] Polyak R., Griva I., *Log-Sigmoid Multipliers Method in Constrained Optimization*, Annals of Operations Research, Vol. 101, s. 427-460, 2001
- [32] Polyak R., Griva I., *Primal-Dual Exterior Point Method for Convex Optimization*, Optimization Methods and Software, Vol. 23, No. 1, s. 141-160, 2008
- [33] Polyak R., Griva I., *Primal-Dual Nonlinear Rescaling Method With Dynamic Scaling Parameter Update*, Mathematical Programming, Vol. 106, s. 237-259, 2005
- [34] Polyak R., Griva I., *Primal-Dual Nonlinear Rescaling Method for Convex Optimization*, Journal of Optimization Theory and Applications, Vol. 122, No. 1, s. 111-156, 2004

- [35] Powell M. J. D., *A Method for Nonlinear Constraints in Minimization Problems*, Optimization, Edited by R. Fletcher, Academic Press, New York, s. 283-298, 1969.
- [36] Powell M. J. D., *Algorithms For Nonlinear Constraints That Use Lagrangian Functions*, Mathematical Programming, Vol. 14, s. 224-248, 1978.
- [37] Press H. P., Teukolsky S. A., Vetterling W. T., Flannery B. P., *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 3rd Edition, 1256 s., 2007
- [38] Rockafellar R. T., *Convex Analysis*, Princeton University Press, New Jersey, 2nd Edition, 1972
- [39] Rockafellar R. T., *Lagrange Multipliers and Optimality*, SIAM Review, Vol. 35, No. 2, s. 183-238, 1993
- [40] Rockafellar R. T., *The Multiplier Method of Hestenes and Powell Applied to Convex Programming*, Journal of Optimization Theory and Applications, Vol. 12, s. 555-562, 1973.
- [41] Roode J. D., *Generalized Lagrangian Functions and Mathematical Programming*, in Optimization, ed. R. Fletcher, Academic Press, s. 327-338, 1969.
- [42] Schittkowski K., *Test Examples for Nonlinear Programming Codes - All Problems from the Hock-Schittkowski-Collection*, Report, Department of Computer Science, University of Bayeruth, 2009
- [43] Tapia R. A., *On Secant Updates for Use in General Constrained Optimization*, Mathematics of Computation, Vol. 51, No. 183, s. 181-202, 1988

Príloha

A. Newtonova minimalizačná metóda a vyhľadávanie na lúči

V tejto časti budeme uvažovať úlohu na voľnú minimalizáciu v tvare

$$\text{minimalizovať } F(\mathbf{x}), \mathbf{x} \in \mathbb{R}^n \quad (5.1)$$

kde $F : \mathbb{R}^n \rightarrow \mathbb{R}$, $F \in C^2$. Na úvod predpokladajme, že F je rýdzokonvexná.

Majme bod \mathbf{x}^k , ktorý je približným riešením úlohy (5.1). Na okolí tohto bodu budeme aproximovať funkciu F Taylorovým polynómom druhého stupňa. Dostávame

$$F(\mathbf{x}) \simeq F(\mathbf{x}^k) + \nabla F(\mathbf{x}^k)^T(\mathbf{x} - \mathbf{x}^k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^k)^T \nabla^2 F(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k). \quad (5.2)$$

Novú aproximáciu minima \mathbf{x}^{k+1} definujeme ako bod minimalizujúci (5.2). Derivovaním (5.2) a dosadením \mathbf{x}^{k+1} dostávame

$$\nabla F(\mathbf{x}^k) + \nabla^2 F(\mathbf{x}^k)(\mathbf{x}^{k+1} - \mathbf{x}^k) = 0,$$

z čoho máme

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \nabla^2 F(\mathbf{x}^k)^{-1} \nabla F(\mathbf{x}^k). \quad (5.3)$$

Tento iteračný predpis určuje jeden krok *klasickkej Newtonovej metódy*. Uvedená metóda patrí do triedy metód, v ktorých sa nový iteračný krok určuje ako

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \mathbf{s}^k,$$

pre voľbu

$$\mathbf{s}^k = -\nabla^2 F(\mathbf{x}^k)^{-1} \nabla F(\mathbf{x}^k), \quad \lambda^k = 1. \quad (5.4)$$

Vidíme, že Newtonovský smer \mathbf{s}^k je *spádový*, čiže platí $\nabla F(\mathbf{x}^k)^T \mathbf{s}^k < 0$ (keďže $\nabla^2 F(\mathbf{x}^k)$ je kladne definitná). Výsledná metóda však nemusí byť spádová v zmysle

$F(\mathbf{x}^{k+1}) < F(\mathbf{x}^k)$. Preto základom *modifikovanej Newtonovej metódy* bude regulácia kroku λ^k .

Do polovice 60-tych rokov sa presadzovala myšlienka optimalizácie kroku λ^k , dôsledkom čoho bolo nutné riešiť minimalizačnú úlohu

$$\lambda^k = \arg \min_{\lambda > 0} \{F(\mathbf{x}^k + \lambda \mathbf{s}^k)\},$$

ktorá je však časovo náročná. Aj preto sa neskôr od tejto požiadavky upúšťalo a cieľom bolo hľadať „približne optimálny“ krok. Kvôli zabráneniu akceptovateľnosti príliš malých alebo veľkých krokov boli neskôr formulované podmienky, ktoré musí krok λ^k spĺňať. Takýmito podmienkami sú napr. *silné Wolfeho podmienky*

$$F(\mathbf{x}^k + \lambda^k \mathbf{s}^k) \leq F(\mathbf{x}^k) + c_1 \lambda^k \nabla F(\mathbf{x}^k)^T \mathbf{s}^k, \quad (5.5)$$

$$|\nabla F(\mathbf{x}^k + \lambda^k \mathbf{s}^k)^T \mathbf{s}^k| \leq c_2 |\nabla F(\mathbf{x}^k)^T \mathbf{s}^k|, \quad (5.6)$$

pre $0 < c_1 < c_2 < 1$. Podmienka (5.5) zabezpečuje hornú hranicu pre krok λ^k , podmienka (5.6) naopak dolnú hranicu. Konštanty c_1, c_2 je možné voliť napr. $c_1 = 10^{-4}, c_2 = 0.9$ [26]. Sústava podmienok (5.5), (5.6) zabezpečí, že krok λ^k bude ležať blízko optimálneho kroku. Známym algoritmom na hľadanie krokov spĺňajúcich (5.5), (5.6) je algoritmus J. Morého a D. Thuenteho [25]. Pre iné algoritmy pozri napr. [9].

Doteraz sme predpokladali ostrú konvexnosť funkcie F . Pre funkcie, ktoré nie sú konvexné už nie je iterácia 5.3 dobre definovaná: smer \mathbf{s}^k nemusí byť spádový, a ak neexistuje $\nabla^2 F(\mathbf{x}^k)^{-1}$, tak \mathbf{s}^k nie je ani definovaný. Tieto problémy môžu nastať aj v prípade konvexných funkcií, ak je matica $\nabla F(\mathbf{x}^k)$ singulárna alebo takmer singulárna. V tomto prípade je možné modifikovanú Newtonovu metódu opraviť uvažovaním matice $\nabla^2 F(\mathbf{x}^k) + \mathbf{E}^k$ namiesto $\nabla^2 F(\mathbf{x}^k)$, kde \mathbf{E}^k je diagonálna matica zvolená tak, že $\nabla^2 F(\mathbf{x}^k) + \mathbf{E}^k$ je zaručene kladne definitná (niektorí autori uvažujú $\mathbf{E}^k = \mu^k \mathbf{I}$, $\mathbf{E}^k = 0$ ak $\nabla^2 F(\mathbf{x}^k)$ je dostatočne kladne definitná). Známym algoritmom na hľadanie matice \mathbf{E}^k je algoritmus GMW81 (pozri [26], [17]) založený na modifikovanom Choleskeho rozklade matice $\nabla^2 F(\mathbf{x}^k)$ (algoritmus v skutočnosti vykoná Choleskeho rozklad $\nabla^2 F(\mathbf{x})$ tak, že platí $\mathbf{L}\mathbf{L}^T = \nabla^2 F(\mathbf{x}) + \mathbf{E}$).

V tejto práci sa často stretávame s potrebou riešiť problém (5.1), k čomu využijeme modifikovanú Newtonovu metódu nasledovnej implementácie.

Krok 1 : Inicializácia. Urči konštantu ε a polož $k = 0$.

Krok 2 : AK $\|\nabla F(\mathbf{x}^k)\| < \varepsilon$, STOP, **Výstup :** \mathbf{x}^k .

Krok 3 : Nájdi \mathbf{s}^k z rovnice $(\nabla^2 F(\mathbf{x}^k) + \mathbf{E}^k)\mathbf{s}^k = -\nabla F(\mathbf{x}^k)$, kde \mathbf{E}^k je určená pomocou GMW81 algoritmu.

Krok 4 : Nájdi približné riešenie úlohy $\lambda^k = \arg \min_{\lambda > 0} \{F(\mathbf{x}^k + \lambda \mathbf{s}^k)\}$ spĺňajúce (5.5), (5.6).

Krok 5 : Polož $\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \mathbf{s}^k$, $k = k + 1$, CHOĎ NA KROK 2.

Na riešenie sústavy v kroku 3 používame Choleskeho rozklad, krok 4 kvôli jednoduhosti riešime Brentovou metódou [6] (testovali sme aj algoritmus MoreThuente [25]). Kódy všetkých procedúr sú priložené k diplomovej práci.

B. Výsledky testovania príkladov CUTEr

V tabuľkách uvádzame výsledky algoritmov dosiahnuté pri riešení vybranej vzorky príkladov zo zborníka CUTEr. Ku každému príkladu pridávame jeho názov a takisto rozmery (n - počet premenných, m - počet nerovníc, p - počet rovníc). Údaje o absolútnej odchýlke bodov $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ od optimálnych riešení nahradíme údajom o funkčnej hodnote v bode $\bar{\mathbf{x}}$ ($\bar{f} = f(\bar{\mathbf{x}})$). Požadovaná tolerancia bola nastavená na $\varepsilon = 10^{-8}$.

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha '3PK', $n = 30, m = 30, p = 0$							
AL_NR	1.720E+00	3.849E-12	9.481E-13	2.737E-231	4	4	0.231
AL_ROC1	1.720E+00	2.821E-12	0.000E+00	2.737E-231	1	1	0.106
AL_ROC2	1.720E+00	2.821E-12	0.000E+00	8.043E-270	1	1	0.007
FIS1	1.720E+00	1.614E-06	6.642E-15	8.043E-270	32	32	0.419
FIS2	1.720E+00	7.174E-07	1.085E-12	8.043E-270	3	3	0.140
POL	1.720E+00	9.174E-12	1.853E-09	8.043E-270	3	4	0.140
úloha 'AIRCRFTA', $n = 8, m = 0, p = 8$							
AL_NR	0.000E+00	6.777E-14	0.000E+00	2.388E-16	1	7	0.030
AL_ROC1	0.000E+00	6.777E-14	0.000E+00	2.388E-16	1	7	0.039
AL_ROC2	0.000E+00	6.777E-14	0.000E+00	2.388E-16	1	7	0.034
FIS1	0.000E+00	1.000E-13	0.000E+00	2.388E-16	4	4	0.054
FIS2	0.000E+00	6.777E-14	0.000E+00	2.388E-16	1	7	0.031
POL	0.000E+00	6.777E-14	0.000E+00	2.388E-16	1	7	0.030
úloha 'AIRCRFTB', $n = 8, m = 0, p = 3$							
AL_NR	1.672E-25	4.397E-11	0.000E+00	1.368E-14	1	23	0.104
AL_ROC1	1.672E-25	4.397E-11	0.000E+00	1.368E-14	1	23	0.123
AL_ROC2	1.672E-25	4.397E-11	0.000E+00	1.368E-14	1	23	0.127
FIS1	8.109E-01	6.994E-14	0.000E+00	0.000E+00	34	34	0.257
FIS2	1.672E-25	4.397E-11	0.000E+00	1.368E-14	1	23	0.110
POL	1.672E-25	4.397E-11	0.000E+00	1.368E-14	1	23	0.107
úloha 'AIRPORT', $n = 84, m = 210, p = 0$							
AL_NR	4.795E+04	3.683E-11	7.262E-09	2.247E-12	17	33	0.323
AL_ROC1	4.795E+04	1.013E-10	5.643E-09	2.184E-12	21	42	0.419
AL_ROC2	4.795E+04	1.933E-12	3.208E-12	4.747E-16	15	36	0.405
FIS1	4.795E+04	1.592E-12	7.512E-10	8.597E-14	13	13	0.321
FIS2	4.795E+04	4.547E-12	1.177E-09	9.164E-14	14	29	0.297
POL	4.795E+04	2.274E-12	3.260E-12	4.747E-16	13	34	0.314

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr

VÝSLEDKY TESTOVANIA CUTE _r PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'ALLINQP', $n = 100, m = 134, p = 28$							
AL_NR	-9.159E+00	1.026E-11	2.254E-09	2.515E-09	14	36	0.292
AL_ROC1	-9.159E+00	6.479E-09	9.171E-09	5.949E-09	11	34	0.288
AL_ROC2	-9.159E+00	5.551E-16	2.637E-17	2.220E-16	3	14	0.164
FIS1	-9.159E+00	2.578E-10	5.346E-14	2.220E-16	10	10	0.208
FIS2	-9.159E+00	9.290E-16	4.628E-14	2.220E-16	9	17	0.134
POL	-9.159E+00	2.876E-11	5.012E-11	8.242E-11	9	18	0.113
úloha 'AUG2DCP', $n = 220, m = 220, p = 110$							
AL_NR	3.040E+02	6.106E-10	2.897E-09	1.955E-09	12	22	0.487
AL_ROC1	3.040E+02	3.026E-13	2.260E-09	1.578E-11	13	19	0.554
AL_ROC2	3.040E+02	2.331E-15	3.955E-16	8.882E-16	6	12	0.437
FIS1	-	-	-	-	-	-	-
FIS2	3.040E+02	6.982E-12	3.620E-12	1.934E-10	8	11	0.389
POL	3.040E+02	1.308E-10	2.887E-10	7.195E-09	8	13	0.347
úloha 'AVGASA', $n = 8, m = 26, p = 0$							
AL_NR	-4.632E+00	1.545E-13	4.811E-09	2.438E-09	9	13	0.050
AL_ROC1	-4.632E+00	6.917E-14	3.125E-09	1.282E-09	11	14	0.059
AL_ROC2	-4.632E+00	8.882E-16	1.371E-16	5.551E-17	3	6	0.035
FIS1	-4.632E+00	4.237E-12	1.081E-14	7.757E-17	8	8	0.105
FIS2	-4.632E+00	1.422E-10	1.055E-14	8.043E-270	5	9	0.035
POL	-4.632E+00	1.518E-10	4.758E-10	5.392E-10	5	10	0.038
úloha 'AVGASB', $n = 8, m = 26, p = 0$							
AL_NR	-4.483E+00	4.885E-14	7.231E-09	2.527E-09	11	17	0.073
AL_ROC1	-4.483E+00	1.423E-13	8.562E-09	3.327E-09	12	16	0.068
AL_ROC2	-4.483E+00	8.882E-16	9.889E-17	2.220E-16	4	7	0.041
FIS1	-4.483E+00	4.289E-09	1.131E-14	1.212E-16	8	8	0.102
FIS2	-4.483E+00	1.948E-09	1.138E-14	2.737E-231	5	10	0.040
POL	-4.483E+00	7.994E-15	3.841E-14	1.477E-14	6	12	0.044
úloha 'BDEXP', $n = 500, m = 500, p = 0$							
AL_NR	0.000E+00	2.058E-18	5.034E-15	2.737E-231	3	3	5.085
AL_ROC1	7.670E-28	9.971E-27	0.000E+00	2.737E-231	1	1	1.593
AL_ROC2	7.670E-28	9.971E-27	0.000E+00	2.737E-231	1	1	1.598
FIS1	4.036E-07	1.414E-08	9.238E-16	8.043E-270	74	74	24.486
FIS2	0.000E+00	3.078E-15	1.589E-12	2.737E-231	3	3	3.428
POL	8.286E-28	2.639E-21	1.279E-18	2.737E-231	3	3	2.344

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTE_r, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'BIGGSB1', $n = 1000, m = 1998, p = 0$							
AL_NR	1.500E-02	7.995E-13	2.155E-09	5.887E-09	12	28	79.869
AL_ROC1	1.500E-02	1.006E-13	2.430E-09	8.043E-270	4	11	35.235
AL_ROC2	1.500E-02	2.220E-16	0.000E+00	2.737E-231	3	9	58.144
FIS1	1.500E-02	9.516E-09	7.390E-13	8.043E-270	37	37	181.446
FIS2	1.500E-02	6.366E-09	4.101E-13	2.737E-231	8	13	44.048
POL	1.500E-02	2.166E-11	1.458E-13	7.324E-13	8	14	55.102
úloha 'BOX2', $n = 3, m = 0, p = 1$							
AL_NR	8.027E-19	9.126E-10	0.000E+00	1.033E-10	1	5	0.068
AL_ROC1	8.027E-19	9.126E-10	0.000E+00	1.033E-10	1	5	0.039
AL_ROC2	8.027E-19	1.727E-09	0.000E+00	1.033E-10	1	5	0.023
FIS1	1.829E-16	1.244E-09	0.000E+00	0.000E+00	13	13	0.118
FIS2	8.027E-19	9.126E-10	0.000E+00	1.033E-10	1	5	0.019
POL	8.027E-19	9.126E-10	0.000E+00	1.033E-10	1	5	0.019
úloha 'CONGIGMZ', $n = 3, m = 5, p = 0$							
AL_NR	2.800E+01	2.758E-10	2.448E-09	8.043E-270	11	31	0.128
AL_ROC1	1.968E+02	1.846E+20	3.969E+21	4.407E+02	99	5000	25.742
AL_ROC2	1.968E+02	1.846E+20	3.969E+21	4.407E+02	99	5000	25.558
FIS1	2.800E+01	3.259E-12	1.602E-14	7.105E-15	8	8	0.094
FIS2	2.800E+01	9.907E-16	7.423E-15	2.737E-231	5	16	0.075
POL	2.800E+01	3.794E-15	7.106E-15	7.105E-15	6	18	0.078
úloha 'CVXBQP1', $n = 100, m = 200, p = 0$							
AL_NR	2.272E+02	3.411E-13	6.497E-09	2.179E-12	8	10	0.075
AL_ROC1	2.272E+02	4.263E-14	2.031E-09	1.042E-11	16	19	0.150
AL_ROC2	2.273E+02	0.000E+00	0.000E+00	2.737E-231	8	11	0.119
FIS1	2.273E+02	1.142E-10	1.107E-11	6.106E-15	5	5	0.108
FIS2	2.273E+02	1.421E-14	1.004E-11	6.467E-15	7	9	0.078
POL	2.272E+02	1.421E-13	2.192E-09	7.567E-13	7	11	0.077
úloha 'CVXQP1', $n = 100, m = 200, p = 50$							
AL_NR	1.159E+04	4.614E-10	1.845E-09	2.061E-09	13	39	0.312
AL_ROC1	1.159E+04	1.771E-10	9.889E-09	3.787E-11	19	30	0.264
AL_ROC2	1.159E+04	1.771E-10	9.889E-09	3.787E-11	19	30	0.481
FIS1	1.159E+04	2.274E-13	2.716E-11	1.125E-14	11	11	0.332
FIS2	1.159E+04	7.958E-13	2.990E-11	2.177E-14	12	35	0.318
POL	1.159E+04	2.274E-13	1.388E-11	8.692E-12	12	38	0.289

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'CVXQP2', $n = 100, m = 200, p = 25$							
AL_NR	8.121E+03	1.637E-11	5.525E-10	2.414E-10	11	27	0.196
AL_ROC1	8.121E+03	1.307E-11	9.028E-09	2.308E-11	19	26	0.225
AL_ROC2	8.121E+03	1.137E-13	1.380E-14	8.882E-16	10	17	0.172
FIS1	8.121E+03	1.787E-12	3.383E-11	1.324E-14	17	17	0.543
FIS2	8.121E+03	1.137E-13	4.861E-11	2.637E-14	12	28	0.425
POL	8.121E+03	2.274E-13	2.734E-13	1.021E-14	10	26	0.195
úloha 'DTCIL', $n = 598, m = 0, p = 400$							
AL_NR	4.254E-01	5.770E-15	0.000E+00	8.051E-10	6	9	4.669
AL_ROC1	4.254E-01	5.770E-15	0.000E+00	8.051E-10	6	9	5.335
AL_ROC2	4.254E-01	8.246E-09	0.000E+00	5.653E-09	3	7	5.175
FIS1	4.254E-01	1.372E-10	0.000E+00	9.714E-17	9	9	3.318
FIS2	4.254E-01	2.040E-10	0.000E+00	9.021E-17	6	7	2.203
POL	4.254E-01	8.209E-13	0.000E+00	1.211E-12	4	8	4.128
úloha 'DUAL1', $n = 85, m = 170, p = 1$							
AL_NR	3.501E-02	7.149E-10	1.793E-12	3.868E-12	18	35	0.358
AL_ROC1	3.501E-02	9.507E-14	5.090E-10	1.125E-09	11	30	0.297
AL_ROC2	3.501E-02	2.352E-15	6.753E-19	2.164E-16	6	16	0.239
FIS1	3.501E-02	6.103E-10	6.653E-13	1.341E-09	12	12	0.342
FIS2	3.501E-02	1.113E-12	6.654E-13	1.341E-09	18	20	0.315
POL	3.501E-02	2.852E-15	1.949E-15	8.003E-14	16	37	0.366
úloha 'DUAL2', $n = 96, m = 192, p = 1$							
AL_NR	3.373E-02	3.772E-11	1.640E-09	1.815E-09	16	30	0.305
AL_ROC1	3.373E-02	2.754E-14	1.248E-09	3.076E-11	10	19	0.267
AL_ROC2	3.373E-02	1.638E-15	1.323E-22	3.886E-16	7	11	0.244
FIS1	3.373E-02	1.614E-10	8.864E-14	1.094E-13	8	8	0.274
FIS2	3.373E-02	2.033E-15	8.716E-14	2.914E-14	14	17	0.236
POL	3.373E-02	1.235E-15	7.739E-15	5.377E-13	12	29	0.364
úloha 'EXPQUAD', $n = 120, m = 20, p = 0$							
AL_NR	-3.626E+06	6.594E-12	8.207E-10	2.726E-11	7	22	0.133
AL_ROC1	-3.626E+06	5.912E-12	2.461E-09	6.214E-11	13	36	0.216
AL_ROC2	-3.626E+06	8.422E-08	0.000E+00	8.043E-270	6	29	0.180
FIS1	-	-	-	-	-	-	-
FIS2	-3.626E+06	6.366E-12	2.263E-13	1.776E-15	5	20	0.167
POL	-3.626E+06	6.995E-12	9.914E-11	3.306E-12	6	21	0.117

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'GENHS28', $n = 10, m = 0, p = 8$							
AL_NR	9.272E-01	3.775E-15	0.000E+00	2.066E-09	6	6	0.022
AL_ROC1	9.272E-01	3.775E-15	0.000E+00	2.066E-09	6	6	0.022
AL_ROC2	9.272E-01	2.220E-16	0.000E+00	1.110E-16	3	3	0.014
FIS1	9.272E-01	1.186E-10	0.000E+00	1.110E-16	7	7	0.094
FIS2	9.272E-01	5.274E-16	0.000E+00	1.665E-16	4	4	0.010
POL	9.272E-01	2.220E-15	0.000E+00	3.331E-16	4	4	0.012
úloha 'GILBERT', $n = 50, m = 1, p = 1$							
AL_NR	2.115E+01	1.055E-14	9.442E-17	7.392E-09	11	15	0.127
AL_ROC1	2.115E+01	9.881E-15	0.000E+00	7.391E-09	11	15	0.100
AL_ROC2	2.115E+01	1.946E-12	0.000E+00	2.550E-12	5	10	0.068
FIS1	2.115E+01	6.112E-09	4.414E-16	1.962E-09	13	13	0.107
FIS2	2.115E+01	2.929E-11	4.391E-16	3.304E-12	5	9	0.050
POL	2.115E+01	1.359E-10	9.158E-20	9.411E-10	6	10	0.054
úloha 'GOULDQP3', $n = 699, m = 1398, p = 349$							
AL_NR	2.028E+00	4.815E-10	1.354E-10	6.475E-09	11	24	25.149
AL_ROC1	2.028E+00	4.968E-11	2.750E-10	2.795E-10	7	32	36.587
AL_ROC2	2.028E+00	3.057E-15	0.000E+00	3.553E-15	6	31	51.286
FIS1	2.018E+00	3.338E-08	8.978E-03	4.375E-03	99	99	232.696
FIS2	1.962E+00	1.174E-04	1.758E-02	1.074E-01	99	108	255.446
POL	2.028E+00	2.693E-13	2.766E-12	2.301E-10	11	22	41.565
úloha 'GRIDGENA', $n = 578, m = 780, p = 188$							
AL_NR	1.920E+03	4.448E-13	1.840E-29	4.494E-09	9	10	6.857
AL_ROC1	1.920E+03	3.389E-13	0.000E+00	3.297E-09	8	9	6.894
AL_ROC2	1.920E+03	2.953E-09	0.000E+00	5.421E-20	3	4	5.423
FIS1	1.920E+03	6.067E-09	4.152E-14	1.023E-19	5	5	4.136
FIS2	1.920E+03	7.200E-10	3.088E-13	7.790E-18	4	5	3.508
POL	1.920E+03	7.239E-13	3.439E-26	7.565E-13	5	6	4.241
úloha 'GRIDNETA', $n = 180, m = 41, p = 148$							
AL_NR	9.524E+01	3.624E-13	3.701E-09	2.821E-09	15	21	0.330
AL_ROC1	9.524E+01	2.010E-13	3.448E-09	2.208E-10	14	15	0.341
AL_ROC2	9.524E+01	2.010E-13	3.448E-09	2.208E-10	14	15	0.557
FIS1	9.524E+01	4.000E+00	8.533E-15	2.665E-15	99	99	9.702
FIS2	9.524E+01	2.070E-13	1.002E-14	9.194E-16	9	15	0.638
POL	9.524E+01	7.097E-11	2.328E-10	1.744E-10	8	15	0.400

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'GRIDNETI', $n = 924, m = 308, p = 484$							
AL_NR	8.787E+01	1.866E-11	2.194E-09	5.103E-09	16	28	65.608
AL_ROC1	8.787E+01	5.262E-14	2.330E-09	4.822E-09	14	28	69.615
AL_ROC2	8.787E+01	5.262E-14	2.330E-09	4.822E-09	14	28	87.785
FIS1	8.787E+01	9.058E-09	1.394E-13	5.025E-11	10	10	13.095
FIS2	8.787E+01	5.284E-14	1.715E-13	5.024E-11	12	16	26.136
POL	8.787E+01	8.882E-16	5.356E-15	1.245E-12	12	22	49.327
úloha 'HANGING', $n = 308, m = 180, p = 12$							
AL_NR	-6.202E+02	7.553E-12	5.016E-09	1.776E-10	11	35	2.226
AL_ROC1	-5.378E+00	2.050E-02	6.931E+11	4.035E+00	99	2281	139.942
AL_ROC2	-5.378E+00	2.050E-02	6.931E+11	4.035E+00	99	2281	171.870
FIS1	-6.202E+02	3.858E-11	1.129E-11	1.833E-11	26	26	2.278
FIS2	-6.202E+02	1.180E-13	3.260E-13	1.910E-14	6	26	1.568
POL	-6.202E+02	1.625E-11	2.438E-09	2.891E-10	7	25	1.360
úloha 'HAGER1', $n = 201, m = 0, p = 101$							
AL_NR	8.808E-01	2.976E-11	0.000E+00	5.868E-09	10	10	0.166
AL_ROC1	8.808E-01	2.976E-11	0.000E+00	5.868E-09	10	10	0.201
AL_ROC2	8.808E-01	4.783E-13	0.000E+00	2.620E-13	3	3	0.093
FIS1	8.808E-01	8.049E-09	0.000E+00	2.198E-14	11	11	0.386
FIS2	8.808E-01	3.152E-09	0.000E+00	2.254E-14	5	5	0.112
POL	8.808E-01	4.018E-11	0.000E+00	3.115E-13	6	7	0.101
úloha 'HART6', $n = 6, m = 12, p = 0$							
AL_NR	-3.323E+00	3.022E-12	9.918E-10	8.043E-270	4	7	0.020
AL_ROC1	-3.323E+00	3.475E-14	0.000E+00	8.043E-270	2	6	0.023
AL_ROC2	-3.323E+00	3.475E-14	0.000E+00	2.737E-231	2	6	0.027
FIS1	-6.687E-04	7.078E-03	2.670E-15	8.043E-270	99	99	0.944
FIS2	-3.323E+00	6.155E-11	5.277E-15	2.737E-231	4	7	0.021
POL	-3.323E+00	2.640E-10	8.164E-13	8.043E-270	4	7	0.021
úloha 'HS11', $n = 2, m = 1, p = 0$							
AL_NR	-8.498E+00	1.219E-12	6.953E-09	2.280E-09	5	11	0.100
AL_ROC1	-8.498E+00	2.665E-15	7.035E-09	2.307E-09	8	14	0.081
AL_ROC2	-8.498E+00	1.763E-11	4.166E-12	1.366E-12	4	11	0.054
FIS1	-8.498E+00	1.423E-09	1.022E-13	3.353E-14	7	7	0.075
FIS2	-8.498E+00	6.350E-13	5.552E-14	1.821E-14	4	10	0.042
POL	-8.498E+00	2.984E-11	7.733E-09	2.536E-09	4	8	0.041

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'HS14', $n = 2, m = 1, p = 1$							
AL_NR	1.393E+00	2.531E-14	3.501E-09	6.940E-09	10	14	0.049
AL_ROC1	1.393E+00	2.087E-14	6.913E-09	3.744E-09	10	13	0.047
AL_ROC2	1.393E+00	1.023E-08	6.187E-09	3.350E-09	4	9	0.034
FIS1	1.393E+00	7.162E-09	4.309E-12	2.333E-12	6	6	0.059
FIS2	1.393E+00	6.861E-09	7.936E-09	4.298E-09	4	7	0.021
POL	1.393E+00	1.386E-13	1.597E-12	2.048E-12	5	7	0.017
úloha 'HS34', $n = 3, m = 8, p = 0$							
AL_NR	-8.340E-01	8.107E-10	1.915E-11	2.737E-231	5	19	0.086
AL_ROC1	-	-	-	-	-	-	-
AL_ROC2	-	-	-	-	-	-	-
FIS1	-8.340E-01	1.280E-11	1.516E-14	3.233E-13	11	11	0.203
FIS2	-8.340E-01	1.443E-15	8.837E-15	1.723E-13	8	15	0.067
POL	-8.340E-01	4.408E-39	7.715E-17	2.737E-231	5	23	0.126
úloha 'HS37', $n = 3, m = 8, p = 0$							
AL_NR	-3.456E+03	8.572E-10	1.023E-12	7.105E-15	5	9	0.051
AL_ROC1	-3.456E+03	6.730E-11	3.070E-12	8.043E-270	8	12	0.071
AL_ROC2	-3.456E+03	2.842E-14	0.000E+00	8.043E-270	6	10	0.069
FIS1	-1.597E-22	1.436E-09	1.864E-16	8.043E-270	19	19	0.296
FIS2	-3.456E+03	1.188E-11	1.029E-12	8.043E-270	4	8	0.044
POL	-3.456E+03	0.000E+00	2.053E-48	8.043E-270	6	10	0.044
úloha 'HS44NEW', $n = 10, m = 16, p = 0$							
AL_NR	-1.500E+01	3.115E-11	4.050E-11	2.871E-12	5	12	0.095
AL_ROC1	-1.500E+01	8.334E-12	1.545E-09	8.043E-270	6	10	0.085
AL_ROC2	-1.500E+01	8.334E-12	1.545E-09	2.737E-231	6	10	0.112
FIS1	-3.000E+00	1.859E-10	4.111E-15	2.737E-231	11	11	0.189
FIS2	-1.500E+01	4.121E-11	1.066E-12	4.547E-13	4	9	0.074
POL	-1.500E+01	7.105E-15	1.338E-14	8.043E-270	6	10	0.066
úloha 'HS100', $n = 7, m = 4, p = 0$							
AL_NR	6.806E+02	6.065E-09	5.377E-12	4.121E-13	4	18	0.125
AL_ROC1	6.806E+02	5.556E-11	7.152E-11	8.043E-270	4	17	0.111
AL_ROC2	6.806E+02	4.558E-08	6.159E-11	1.053E-10	3	16	0.111
FIS1	6.806E+02	2.931E-12	3.311E-14	2.376E-14	14	14	0.214
FIS2	6.806E+02	1.230E-10	1.669E-09	4.449E-09	4	14	0.086
POL	6.806E+02	3.553E-14	6.666E-13	5.342E-13	4	20	0.121

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'HS108', $n = 9, m = 14, p = 0$							
AL_NR	-8.660E-01	5.868E-09	8.302E-10	3.990E-13	8	175	1.033
AL_ROC1	-8.660E-01	4.596E-11	7.834E-10	2.513E-10	6	86	0.511
AL_ROC2	-8.660E-01	4.596E-11	7.834E-10	2.513E-10	6	86	0.562
FIS1	-8.660E-01	2.844E-09	1.892E-12	2.471E-12	69	69	0.981
FIS2	-8.660E-01	3.113E-09	3.868E-15	7.617E-20	6	25	0.151
POL	-8.660E-01	4.520E-10	1.274E-13	2.283E-13	6	32	0.170
úloha 'HS113', $n = 10, m = 8, p = 0$							
AL_NR	2.431E+01	5.204E-10	8.898E-11	2.409E-11	4	20	0.117
AL_ROC1	2.431E+01	1.879E-11	2.910E-09	1.036E-09	4	19	0.114
AL_ROC2	2.431E+01	3.143E-08	3.417E-10	2.856E-10	3	18	0.123
FIS1	2.431E+01	4.370E-13	9.369E-15	1.421E-14	11	11	0.137
FIS2	2.431E+01	5.760E-13	1.052E-11	5.064E-10	4	19	0.109
POL	2.431E+01	6.598E-09	6.803E-11	5.143E-11	3	23	0.125
úloha 'HS118', $n = 15, m = 59, p = 0$							
AL_NR	6.648E+02	4.736E-12	1.460E-10	1.795E-10	4	24	0.245
AL_ROC1	6.648E+02	1.348E-12	1.502E-09	3.050E-10	3	42	0.396
AL_ROC2	6.648E+02	2.802E-13	1.502E-09	3.050E-10	3	42	0.393
FIS1	1.798E+02	4.341E+01	1.359E+03	1.466E+03	99	99	2.062
FIS2	6.648E+02	5.286E-11	2.383E-11	4.471E-10	4	23	0.173
POL	6.648E+02	1.776E-15	8.943E-11	3.007E-10	4	27	0.214
úloha 'JANNSON3', $n = 200, m = 402, p = 1$							
AL_NR	1.985E+02	1.378E-08	7.892E-09	1.430E-10	13	17	0.646
AL_ROC1	1.985E+02	1.081E-13	0.000E+00	1.193E-10	6	10	0.573
AL_ROC2	1.985E+02	2.220E-16	0.000E+00	1.089E-16	5	8	0.969
FIS1	1.985E+02	9.427E-09	1.780E-13	1.119E-15	5	5	1.084
FIS2	1.985E+02	3.708E-14	1.781E-13	1.366E-16	6	7	0.932
POL	1.985E+02	1.452E-08	1.737E-15	4.187E-14	7	11	0.785
úloha 'MATRIX2', $n = 6, m = 6, p = 0$							
AL_NR	1.754E-13	3.687E-09	3.399E-13	1.244E-14	22	35	0.222
AL_ROC1	1.717E-11	4.272E-18	3.959E-11	3.175E-12	6	28	0.181
AL_ROC2	4.830E-19	1.387E-09	4.316E-20	3.285E-09	5	14	0.109
FIS1	1.154E-16	4.234E-09	2.234E-16	2.693E-17	6	6	0.089
FIS2	1.375E-09	3.419E-09	6.438E-09	2.331E-09	13	17	0.050
POL	1.844E-22	1.506E-18	3.688E-22	6.329E-10	91	127	0.893

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'MCCORMCK', $n = 1000, m = 2000, p = 0$							
AL_NR	-9.137E+02	1.018E-13	6.904E-09	6.377E-09	5	8	24.310
AL_ROC1	-9.137E+02	8.882E-13	8.139E-10	8.043E-270	4	48	152.777
AL_ROC2	-9.137E+02	1.554E-15	0.000E+00	8.043E-270	4	48	170.406
FIS1	-9.137E+02	4.090E-12	4.521E-13	2.737E-231	11	11	183.033
FIS2	-9.137E+02	1.812E-14	5.334E-13	2.737E-231	4	7	61.171
POL	-9.137E+02	9.473E-10	1.099E-09	1.176E-09	4	6	26.415
úloha 'MINC44', $n = 51, m = 66, p = 40$							
AL_NR	4.297E-02	4.477E-13	3.357E-13	3.043E-09	11	62	0.466
AL_ROC1	4.297E-02	1.976E-11	0.000E+00	8.681E-09	8	55	0.443
AL_ROC2	1.177E+10	5.961E+20	4.436E+23	8.233E+06	99	4918	64.134
FIS1	-	-	-	-	-	-	-
FIS2	4.297E-02	4.120E-11	2.778E-14	5.242E-11	9	69	0.545
POL	4.297E-02	4.636E-11	2.393E-18	6.885E-11	7	39	0.268
úloha 'OBSTCLBM', $n = 529, m = 882, p = 88$							
AL_NR	6.519E+00	5.650E-10	1.855E-09	6.603E-09	11	29	12.581
AL_ROC1	6.519E+00	1.998E-14	7.468E-10	2.825E-10	7	27	13.721
AL_ROC2	6.519E+00	1.388E-16	6.291E-18	6.939E-18	4	21	16.531
FIS1	6.519E+00	5.489E-10	4.362E-13	2.431E-13	9	9	6.940
FIS2	6.519E+00	2.474E-13	4.530E-13	2.760E-13	11	16	9.670
POL	6.519E+00	2.220E-16	4.760E-14	9.748E-14	12	24	15.045
úloha 'OPTCTRL6', $n = 302, m = 0, p = 203$							
AL_NR	5.039E+03	1.177E-08	0.000E+00	9.108E-09	17	32	1.687
AL_ROC1	5.039E+03	1.177E-08	0.000E+00	9.108E-09	17	32	1.995
AL_ROC2	5.039E+03	2.774E-11	0.000E+00	1.632E-15	16	78	5.547
FIS1	5.039E+03	3.881E-07	0.000E+00	1.161E-13	5	5	0.670
FIS2	5.039E+03	1.356E-06	0.000E+00	5.348E-11	11	14	1.486
POL	5.039E+03	2.728E-11	0.000E+00	1.570E-11	17	43	3.580
úloha 'PRIMAL1', $n = 325, m = 86, p = 0$							
AL_NR	-3.501E-02	4.197E-09	3.325E-11	3.517E-09	9	28	3.124
AL_ROC1	-3.501E-02	7.653E-14	1.851E-09	5.899E-09	4	23	2.671
AL_ROC2	-3.501E-02	5.375E-17	9.325E-18	4.510E-17	4	23	3.469
FIS1	-3.501E-02	5.803E-11	9.985E-13	1.664E-09	11	11	1.928
FIS2	-3.501E-02	6.940E-14	4.006E-13	1.050E-09	13	19	2.101
POL	-3.501E-02	2.220E-16	3.086E-16	1.950E-13	9	31	3.242

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTEr PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'PRIMAL2', $n = 649, m = 97, p = 0$							
AL_NR	-3.373E-02	1.116E-09	5.588E-11	6.158E-10	8	23	14.301
AL_ROC1	-3.373E-02	1.082E-13	2.048E-09	5.639E-09	4	16	9.780
AL_ROC2	-3.373E-02	1.061E-15	9.579E-18	5.594E-17	3	15	10.657
FIS1	-3.373E-02	1.452E-10	7.419E-14	1.570E-13	7	7	1.316
FIS2	-3.373E-02	6.217E-15	7.597E-14	1.570E-13	9	15	6.148
POL	-3.373E-02	1.173E-09	1.583E-13	2.879E-15	8	25	14.609
úloha 'S268', $n = 5, m = 5, p = 0$							
AL_NR	4.657E-10	7.335E-10	9.273E-10	2.737E-231	5	9	0.044
AL_ROC1	1.455E-11	7.276E-12	0.000E+00	2.737E-231	2	3	0.020
AL_ROC2	1.455E-11	7.276E-12	0.000E+00	2.737E-231	2	3	0.024
FIS1	-3.638E-12	7.269E-07	3.207E-17	2.737E-231	8	8	0.098
FIS2	1.091E-11	1.231E-06	4.332E-18	2.737E-231	4	7	0.028
POL	1.091E-11	7.724E-07	5.103E-18	2.737E-231	4	7	0.029
úloha 'SIMPLIPA', $n = 2, m = 4, p = 0$							
AL_NR	1.000E+00	1.360E-12	2.138E-11	1.395E-11	5	10	0.048
AL_ROC1	1.000E+00	0.000E+00	0.000E+00	8.043E-270	3	5	0.024
AL_ROC2	1.000E+00	0.000E+00	0.000E+00	2.737E-231	3	5	0.026
FIS1	1.000E+00	5.946E-09	4.634E-12	4.633E-12	8	8	0.094
FIS2	1.000E+00	6.521E-09	1.954E-11	1.855E-11	3	5	0.023
POL	1.000E+00	0.000E+00	1.278E-11	1.278E-11	5	7	0.027
úloha 'STCQP1', $n = 1025, m = 2025, p = 510$							
AL_NR	2.618E+04	9.026E-09	1.508E-75	2.331E-15	13	20	69.879
AL_ROC1	2.618E+04	8.754E-12	2.184E-09	2.437E-11	18	24	101.653
AL_ROC2	2.618E+04	8.754E-12	2.184E-09	2.437E-11	18	24	272.656
FIS1	2.618E+04	8.160E-12	8.930E-13	2.454E-14	9	9	72.859
FIS2	2.618E+04	9.030E-11	7.999E-11	1.598E-10	14	23	124.668
POL	2.618E+04	4.547E-13	2.242E-12	1.039E-12	10	18	85.552
úloha 'UBH1', $n = 909, m = 606, p = 612$							
AL_NR	1.116E+00	1.811E-11	1.970E-10	2.700E-11	6	6	12.935
AL_ROC1	1.116E+00	2.080E-11	0.000E+00	1.625E-09	5	5	15.141
AL_ROC2	1.116E+00	2.129E-12	0.000E+00	3.837E-13	3	3	12.521
FIS1	1.799E+00	2.219E-05	2.657E-13	1.812E-13	99	99	222.896
FIS2	1.116E+00	6.341E-10	2.691E-13	4.214E-13	5	5	11.556
POL	1.116E+00	2.189E-06	0.000E+00	7.310E-13	99	102	307.053

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTEr, pokračovanie

VÝSLEDKY TESTOVANIA CUTE _r PRÍKLADOV							
ALG	\bar{f}	GRAD	KOMP	PRIP	počet iterácií	\sum mikroit.	časová náročnosť
úloha 'WACHBIEG', $n = 3, m = 2, p = 2$							
AL_NR	1.000E+00	1.021E-14	3.602E-10	7.063E-10	7	15	0.056
AL_ROC1	1.000E+00	3.828E-13	2.450E-09	4.903E-09	6	15	0.064
AL_ROC2	1.000E+00	8.882E-16	3.218E-16	1.110E-15	4	14	0.066
FIS1	1.000E+00	1.522E-09	1.158E-12	2.316E-12	10	10	0.106
FIS2	1.000E+00	1.611E-11	7.514E-12	1.503E-11	4	11	0.039
POL	1.000E+00	1.887E-15	2.449E-16	8.882E-16	5	15	0.055

Tabuľka 5.7: Súhrnné výsledky pre testovanie úloh zo zbierky CUTE_r, pokračovanie

C. Zdrojový kód

Záverečnú časť prílohy venujeme zdrojovým kódom algoritmov. Vypíšeme niektoré z algoritmov uvažovaných v časti 4.2 a s ich subprocedúrami, celý súbor funkcií je možné nájsť na priloženom médiu. Funkcie v jazyku C je nutné pred použitím skompilovať pomocou MATLABBovského príkazu mex s parametrom `-lmwblas` kvôli zaradeniu hlavičiek funkcií z knižnice BLAS.

Generátor úloh z časti 4.1

```
1 function [varargout] = generUQuad(n, m, ma, mq, meq, dist, condn, gdiag, outopt)
2
3 % [x_hat,u_hat,v_hat,D,G,h,Gin,hin,rin,A,b,Aeq,beq,x0] =
4 % generUQuad(n,m,ma,mq,meq,dist,rcond,gdiag,outopt)
5 % funkcia generuje ulohu bikvadratickeho konvexneho programovania v tvare
6 %
7 % min 0.25*(x'*D*x).^2 + 0.5*x'*G*x + h'*x
8 % x
9 % za podmienok 0.5*x'*Gin*x+hin'*x >= rin
10 % A*x >= b
11 % Aeq*x == beq
12 %
13 % Vstupne parametre specifikuju rozsah ulohy: n - pocet premennych,
14 % m - pocet ohraniceni v tvare nerovnosti, mq - pocet kvadratickych
15 % ohraniceni (mq <= m), ma je pocet ohraniceni v tvare nerovnosti,
16 % ktore su aktivne v bode optima (predpoklada sa ma <= min(m,n,mq)),
17 % meq - pocet ohraniceni v tvare rovnice, dist - volitelny parameter, ktory
18 % udava vzdialenost startovacieho bodu x0 od optimalneho riesenia ulohy
19 % x_hat. V pripade nezadania dist je vo vystupe x0 = [], rcond -
20 % pozadovane cislo podmienosti matice D,G, pripadne Gin, gdiag -
21 % udava, ci su matice Gin diagonalne, alebo nie,
22 % options - struktura udavajuca vystupne nastavenia generatora
23 %
24 % STRUKTURA VYSTUPU:
25 % Vstupna premenna outopt specifikuje format vystupu. Volba 's' sposobí,
26 % ze v pripade meq = 0 alebo m = 0 budu vo vystupe vynechane v_hat, Aeq,
27 % beq (ak meq = 0), pripadne u_hat, A, b, Gin, hin, rin (ak m = 0).
28 % Teda napr. pri volbe outopt = 's', a meq = 0 je vystup vo forme
29 % [x_hat, u_hat, D, G, h, A, b, Gin, hin, rin, x0].
30
31 % kontrola vstupnych parametrov
32
33 if nargin < 9
34 outopt = 's';
35 if nargin < 8
36 gdiag = [];
37 if nargin < 6
38 dist = [];
39 if nargin < 5
40 meq = 0;
41 if nargin < 4
42 meq = 0;
43 if nargin < 3
44 error('Prilis malo vstupnych parametrov, minimum je 3');
45 end
46 end
47 end
48 end
49 end
50 end
51
52 % kontroly chybnych vstupov
53 if isempty(n) || isempty(m) || isempty(ma) || isempty(mq) || isempty(meq)
54 error('Vstupne parametre n, m, ma, mq a meq musia byt ciselne udaje');
55 end
56 if isempty(condn)
57 condn = 5;
58 end
59
60 n = max(0, floor(n)); m = max(0, floor(m));
61 ma = max(0, floor(ma)); meq = max(0, floor(meq));
62 mq = max(0, floor(mq)); dist = max(0, dist);
63 condn = max(2, condn); rcondn = 1/condn;
64
65 if meq + ma > n
66 error('Pocet aktivnych ohraniceni (%i), ...
67 meq + ma, n);
68 end
69 if mq > m
70 error('Ziadany pocet kvadratickych ohraniceni presahuje ich pocet')
71 end
72
73 % kontrola vstupnych parametrov, pripad, ked je poziadavka na vynechanie
74 % nepotrebnych vystupov
75 outoptindex = strcmpi(outopt, 's');
76 if outoptindex
77 if meq && nargin > 11
78 error('Pocet vstupnych parametrov je nekonzistentny so vstupom ulohy')
79 end
80 if m && nargin > 8
81 error('Pocet vstupnych parametrov je nekonzistentny so vstupom ulohy')
82 end
83 if meq && m && nargin > 4
84 error('Pocet vstupnych parametrov je nekonzistentny so vstupom ulohy')
85 end
86 else
87 if nargin > 14
88 error('Pocet vstupnych parametrov je nekonzistentny so vstupom ulohy')
89 end
90 end
91
92 % globalne parametre
93 rozsah = 10;
94 dens = 0.4;
% ciselny rozsah niektorých poli
% hustota matice G
```



```

95 decp = 16;
96
97 % heuristika pre urcenia lmax a lmin na zaklade znalosti condn
98 lmax = log(condn + 1);
99 lmin = lmax*condn;
100
101 lvec = lmin + (lmax-lmin)*rand(n, 1);
102 indl = randi([1, n]); indu = randi([1, n]);
103 while indl == indu && n > 1
104     indu = randi([1, n]);
105 end
106 lvec(indl) = lmin; lvec(indu) = lmax;
107
108 % generovanie optimalnych rieseni, u_hat ma rovnomerne rozdelenie
109 % nulove a nenulove hodnoty
110 mt = m - ma; mat = mat; p = ma/m;
111 x_hat = randi([-rozsah, rozsah], n, 1); % optimalne riesenie
112 v_hat = randi([-rozsah, rozsah], meq, 1); % multiplikator rovnosti
113 u_hat = rand(m, 1); % multiplikator nerovnosti
114
115 for i = 1:m
116     if u_hat(i) < p
117         mat = mat - 1;
118     else
119         mt = mt - 1; u_hat(i) = 0;
120     end
121     p = mat/(mat+mt);
122 end
123 p = u_hat > 0;
124 u_hat(p) = floor(0.75*rozsah*rand(nuz(p), 1)) + 1;
125
126 % generovanie matic ulohy
127 D = lvec;
128 G = full(sprandsym(n, dens, rcondn, 2));
129
130 if gdiag == 0
131     gin = zeros(n, n*mq);
132 else
133     gin = zeros(n, mq);
134 end
135 hin = zeros(n, mq);
136 rin = zeros(mq, 1);
137 tgrad = zeros(m, n);
138 for i=1:mq
139     if gdiag == 0
140         Gin(:, n*(i-1)+1:n*i) = -full(sprandsym(n, dens, rcondn, 2));
141         hin(:, i) = randi([-0.5*rozsah, 0.5*rozsah], n, 1);
142         rin(i) = 0.5*x_hat'*Gin(:, n*(i-1)+1:n*i)*x_hat + hin(:, i)'*x_hat;
143         tgrad(i, :) = (Gin(:, n*(i-1)+1:n*i)*x_hat + hin(:, i))';
144     else
145         lvec = lmin + (lmax-lmin)*rand(n, 1);
146         indl = randi([1, n]); indu = randi([1, n]);
147         while indl == indu && n > 1
148             indu = randi([1, n]);
149         end
150         lvec(indl) = lmin; lvec(indu) = lmax;
151         Gin(:, i) = -lvec;
152         hin(:, i) = randi([-0.5*rozsah, 0.5*rozsah], n, 1);
153         rin(i) = 0.5*x_hat'*Gin(:, i)*x_hat + hin(:, i)'*x_hat;
154         tgrad(i, :) = (Gin(:, i)*x_hat + hin(:, i))';
155     end
156 end
157
158 % matice A, Aeq a vektory b, beq
159 A = randi([-0.5*rozsah, 0.5*rozsah], m-mq, n);
160 b = A*x_hat;
161 tgrad(mq+1:m, :) = A;
162 Aeq = randi([-0.5*rozsah, 0.5*rozsah], meq, n);
163 beq = Aeq*x_hat;
164
165 % dopocitanie zvisnych vektorov
166 pt = 'p(1:mq)'; p = 'p(mq+1:m)';
167
168 rin(pt) = rin(pt) - 0.75*rozsah*rand(size(rin(pt)))+10*(-decp/2);
169 b(p) = b(p) - 0.75*rozsah*rand(size(b(p)))+10*(-decp/2);
170 h = tgrad'*u_hat - Aeq'*v_hat - (G*x_hat + (x_hat'*D.*x_hat))*(0.*x_hat);
171
172 % ak je zadane dist, dopocitanie startovacieho bodu x0
173 if isempty(dist)
174     pom = 2*rozsah*rand(n, 1) - rozsah;
175     x0 = x_hat + (abs(dist)/norm(pom))*pom;
176 else
177     x0 = [];
178 end
179
180 % uprava vystupnych parametrov
181 varargout{1} = x_hat;
182
183 % 1. vystup = [x_hat, D, G, h, x0]
184 if 'meq' && outoptindex
185     varargout{2} = G; varargout{3} = h; varargout{4} = x0;
186     return
187 end
188 if 'm'
189     % 2a. vystup = [x_hat, v_hat, D, G, h, Aeq, beq, x0]
190     % 2b. vystup = [x_hat, [] ,v_hat, D, G, h, Gin, hin, rin, A, b, [], [], x0]
191     if outoptindex
192         varargout{2} = v_hat; varargout{3} = D;
193         varargout{4} = G; varargout{5} = h;
194         varargout{6} = Aeq; varargout{7} = beq;
195         varargout{8} = x0;
196         return
197     end
198 end
199 if 'meq'
200     % 3a. vystup = [x_hat, u_hat, D, G, h, Gin, hin, rin, A, b, Aeq, beq, x0]
201     % 3b. vystup = [x_hat, u_hat, [], D, G, h, Gin, hin, rin, A, b, [], [], x0]
202     if outoptindex
203         varargout{2} = u_hat; varargout{3} = D;
204         varargout{4} = G; varargout{5} = h;
205         varargout{6} = Gin; varargout{7} = hin;
206         varargout{8} = rin; varargout{9} = A;
207         varargout{10} = b; varargout{11} = x0;
208         return
209     end
210 end
211
212 % [x_hat, u_hat, v_hat, D, G, h, Gin, hin, rin, A, b, Aeq, beq, x0]
213 varargout{2} = u_hat; varargout{3} = v_hat;
214 varargout{4} = D; varargout{5} = G;
215 varargout{6} = h; varargout{7} = Gin;
216 varargout{8} = hin; varargout{9} = rin;
217 varargout{10} = A; varargout{11} = b;
218 varargout{12} = Aeq; varargout{13} = beq;
219 varargout{14} = x0;

```

Funkcia Rockafellarovho algoritmu (rieši úlohy vytvorené generátorom úloh, verzia na riešenie úloh CUTer je analogická a možno ju nájsť na priloženom médiu).

```

1 function [xopt, yopt, zopt, output] = ulohaUPHR_bq(D,G,h,Gin,hin,rin,...
2     A,b,Aeq,beq,lb,ub,x0,varargin)
3
4 % kontrola nezadaných vstupov
5 if nargin < 14
6     options = [];
7     if nargin < 13
8         x0 = [];
9         if nargin < 12
10            ub = [];
11            if nargin < 11
12                lb = [];
13            end
14        end
15    end
16 end
17
18 b = b(:);
19 h = h(:);
20 beq = beq(:);
21 rin = rin(:);
22
23 % kontroly rozmerov ulohy
24 n = max([length(G), length(h), length(D)]);
25 mq = length(rin);
26 ml = length(b);
27 meq = length(beq);
28 m = ml + meq;
29
30 if (~all(size(A) == [ml, n]) && ~isempty(A))
31     error('Matica A ma nespravny rozmer, ocakavam (yu x %u)', ml, n);
32 elseif (~all(size(Aeq) == [meq, n]) && ~isempty(Aeq))
33     error('Matica AEq ma nespravny rozmer, ocakavam (yu x %u)', meq, n);
34 elseif (length(D) ~= n && ~isempty(D)) || (length(lb) ~= n && ~isempty(lb))
35     error('Diagonálna matica D alebo vektor H maju nespravny rozmer. ');
36 elseif all(size(hin) == [n, mq])
37     error('Matica HIN ma nespravny rozmer. ');
38 end
39
40 % nastavenia parametrov
41 if ~isempty(varargin)
42     options = setoptions(varargin);
43 end
44
45 defaults = struct('Tolerancia', 1e-06, 'HlavneIteracieMax', 100, ...
46     'NewtonovskeIteracieMax', 50, 'Rparameter', 10, 'CasVypoctov', ...
47     1, 'Verbosity', 0, 'UpdateMethod', 1);
48 MaxMainIter = setting(options, 'HlavneIteracieMax', defaults);
49 MaxNewtonIter = setting(options, 'NewtonovskeIteracieMax', defaults);
50 r = setting(options, 'Rparameter', defaults);
51 epsilon = setting(options, 'Tolerancia', defaults);
52 Verb = setting(options, 'Verbosity', defaults);
53 Time = setting(options, 'CasVypoctov', defaults);
54 MultMethod = setting(options, 'UpdateMethod', defaults);
55 PSI_L = 'psi_L_PHR';
56 PSI_E = 'psi_E';
57
58 % v prípade, že A, Aeq, b, beq su prazdne, prenavstavime na spravne
59 % rozmary kvoli nasobeniu
60 if isempty(A)
61     A = zeros(0, n);

```

```

62 end
63 if isempty(Aeq)
64     Aeq = zeros(0, n);
65 end
66 if isempty(b)
67     b = zeros(0, 1);
68 end
69 if isempty(beq)
70     beq = zeros(0, 1);
71 end
72
73 [Aep, bap] = kontrolaohr(lb, ub, n);
74 A = [A; Aep];
75 b = [b; bap];
76
77 % zistime rozmary kvaratickych ohraniceni
78 size_g = size(Gin, 2);
79 if size_g == mq
80     gdiag = 1;
81 elseif size_g == mq*n
82     gdiag = 0;
83 else
84     error('Matica Gin nema pozadovane rozmary');
85 end
86
87 % startovacie hodnoty multiplikatorov
88 y = ones(m, 1);
89 z = zeros(meq, 1);
90 if isempty(x0)
91     x = zeros(n, 1);
92 elseif length(x0) == n
93     x = zeros(n, 1);
94     warning('UlohaU3: x0rozmer', 'Vektor x0 ma iny rozmer ako n, zacinam z [0; ... ; 0]');
95 else
96     x = x0(:);
97 end
98
99 output = struct('HlavneIteracie', 0, 'NewtonovskeIteracie', 0, ...
100     'Tolerancia', epsilon, 'Merit', Inf, 'VypoctovyCas', 0);
101 NewtonIterationCount = 0;
102 NormType = Inf; NormTypeMerit = 'max';
103
104 % spustime meranie casu
105 if Time == 0
106     tic;
107 end
108
109 % vypocty pociatocnych dat v x0
110 [g, H] = Lagrang_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r, ...
111     'Psi_E', 'PSI_L', 'gradhess');
112 [fin, feq, jin, Jeq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
113 [rrt = merit_comp(fin, feq, y, NormTypeMerit);
114 con_viol = const_viol(fin, feq, NormTypeMerit);
115 rho_des = 0.25; % ocakavany pokles porusenia pripustnosti
116 r_max = 5e+165; % maximalny parameter r
117 r_max_mupt = 6;
118 L = 0.9; % konstanta pri vypocte tolerancie toll
119 min_toll = 0.25; % TOLL = min(TULLI, min_toll) + TULL2
120 grel_toll_cls = max(sqrt(norm(h, NormType)), 1);
121 grel_toll = sqrt(r) * grel_toll_cls;
122 grad_flag = 0;

```

```

123 mrt_trhoid = 1; % miesto prepnutia odhadov
124
125 % test pouzitia metody odhadu
126 if MultMethod == 1
127 met = 1;
128 elseif MultMethod == 2
129 if mrtt < mrt_trhoid
130 met = 2;
131 else
132 met = 1;
133 end
134 else
135 error('Neznama metoda odhadu parametrov');
136 end
137
138 % hlavna iteracia, ich pocet je limitovany parametrom MaxMainIter
139 %
140 % Proces updatovania multiplikatorov funguje nasledovne : ak je zvolene
141 % nastavenie UPDATEREIHOD = 1, potom sa pouziva 1st order update, pri
142 % volbe UPDATEREIHOD = 2 sa najprv pouzije 1st order, a ked merit
143 % dostatočne poklesne (pod hodnotu MRT_THRHOID), zacne sa pouzivat
144 % 2nd order update
145
146 for k=1:MaxMainIter % HLAVNA ITERACIA
147
148 % kontrola optimality: ak je merit funkcia dostatočne mala (t.j.
149 % porusenia pripustnosti a podmienka komplementarity
150 % dosahuju male hodnoty, takisto norma gradientu je relativne mala),
151 % ukoncime algoritmus
152 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
153 if verb
154 fprintf('***** ITERACIA %u *****\n', k);
155 fprintf('<PRIP. A KOMPLEMENTARITA> %e\n', mrtt);
156 fprintf('<NORMA GRAD. KLAS. LAGR.> %e\n', norm(LagrClas_quad(x,...
157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
158 y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,grad'), NormType));
159 end
160 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
161 % skalovany gradient v merit funkcii
162 if mrtt < epsilon && grad_flag
163 break
164 end
165
166 % stanovime terminacne kriterium pomocou rozdielu medzi prediktorom
167 % multiplikatora a konkrétnym multiplikatorom, predeleny prislusnym
168 % penalizacnym parametrom r
169 [yt, zt] = mult_update(x,y,z,fin,feq,jin,Jeq,r,g,H,PSI_I,met);
170
171 upd_diff = norm([yt - y; zt - z], NormType);
172 toll = min(L / r * upd_diff, min_toll);
173 if verb
174 fprintf('<TERM. KRITERIUM NEWTON M.> %e\n', toll + epsilon*...
175 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
176 lam, norm(g, NormType));
177 end
178 % pocitadlo Newtonovskych iteracií
179 NewtonIterationCount = NewtonIterationCount + 1;
180 end % KONIEC NEWTONOVSKAJ ITERACIE
181
182 % Update multiplikatorov, 1st order update
183 z = zt;
184 y = yt;
185
186 % Zistime, ci treba zvyisit hodnotu penalizacneho parametra r na
187 % zaklade poklesu neprípustnosti aktualneho bodu, nova hodnota
188 % parametra bude z intervalu [R_CURRENT, R_MAX]

```

```

257 %
258
259 con_viol_old = con_viol;
260 con_viol = const_viol(fin, feq, 'max');
261 rho = con_viol / con_viol_old;
262 if rho > rho_des
263     rt = min([2 * r * rho / rho_des, r_max_mupd * r, r_max]);
264     grel_toll = sqrt(rt / r) * grel_toll;
265     r = rt;
266 end
267 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
268 if verb
269     fprintf('****** KONEC NEWT_ITER *****\n');
270     fprintf('<POCET NEWTON_ITERACT> %\n', j - 1);
271     fprintf('<HYSLIENKA NORMA GRAD_AUFLAG_F> %3.4e\n', norm(g, NormType));
272     fprintf('<PORUSENIE PRIPUSTNOSTI> %\n', con_viol);
273     fprintf('<PORUSENIE KOMPONENTNOSTI> %\n', sum(abs(y.*fin)));
274     fprintf('<PENALIZ. PARAMETER> %\n', r);
275 end
276 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
277 % zistime hodnotu merit funkcie v novom iteracnom bode
278 % (xt, yt, zt), takisto spoctame hodnotu gradientu v tomto bode
279
280 [g, H] = LagrAug_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r, ...
281     PSI_E, PSI_I, 'gradhess');
282 [fin, feq, Jin, Jeq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
283 mrtt = merit_comp(fin, feq, y, NormTypeMerit);
284
285 if norm(LagrClas_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'grad'), ...
286     NormType) < grel_toll * epsilon
287     grad_flag = 1;
288 else grad_flag = 0;
289     % gradient nedosahuje pozadovane hod.
290 end
291
292 % zistime, v akaj metode odhadov pokračovat
293 if MultMethod == 1
294     % ziadany je odhad prveho radu
295     met = 1;
296 else
297     % ziadany je odhad druhého radu
298     met = 2;
299     else
300         met = 1;
301     end
302 end
303
304 % meranie casu ukoncene
305 if Time
306     output.VpocovyCas = toc;
307 end
308
309 % zistime, v akaj metode odhadov pokračovat
310 if MultMethod == 1
311     % ziadany je odhad prveho radu
312     met = 1;
313 else
314     % ziadany je odhad druhého radu
315     met = 2;
316     end
317 end
318
319 % zistime, v akaj metode odhadov pokračovat
320 if Time
321     output.VpocovyCas = toc;
322 end
323
324 % zistime, v akaj metode odhadov pokračovat
325 if MultMethod == 1
326     % ziadany je odhad prveho radu
327     met = 1;
328 else
329     % ziadany je odhad druhého radu
330     met = 2;
331     end
332 end
333
334 % zistime, v akaj metode odhadov pokračovat
335 if MultMethod == 1
336     % ziadany je odhad prveho radu
337     met = 1;
338 else
339     % ziadany je odhad druhého radu
340     met = 2;
341     end
342 end
343
344 % zistime, v akaj metode odhadov pokračovat
345 if MultMethod == 1
346     % ziadany je odhad prveho radu
347     met = 1;
348 else
349     % ziadany je odhad druhého radu
350     met = 2;
351     end
352 end
353
354 % zistime, v akaj metode odhadov pokračovat
355 if MultMethod == 1
356     % ziadany je odhad prveho radu
357     met = 1;
358 else
359     % ziadany je odhad druhého radu
360     met = 2;
361     end
362 end
363
364 % zistime, v akaj metode odhadov pokračovat
365 if MultMethod == 1
366     % ziadany je odhad prveho radu
367     met = 1;
368 else
369     % ziadany je odhad druhého radu
370     met = 2;
371     end
372 end
373
374 % zistime, v akaj metode odhadov pokračovat
375 if MultMethod == 1
376     % ziadany je odhad prveho radu
377     met = 1;
378 else
379     % ziadany je odhad druhého radu
380     met = 2;
381     end
382 end
383
384 % zistime, v akaj metode odhadov pokračovat
385 if MultMethod == 1
386     % ziadany je odhad prveho radu
387     met = 1;
388 else
389     % ziadany je odhad druhého radu
390     met = 2;
391     end
392 end
393
394 % zistime, v akaj metode odhadov pokračovat
395 if MultMethod == 1
396     % ziadany je odhad prveho radu
397     met = 1;
398 else
399     % ziadany je odhad druhého radu
400     met = 2;
401     end
402 end
403
404 % zistime, v akaj metode odhadov pokračovat
405 if MultMethod == 1
406     % ziadany je odhad prveho radu
407     met = 1;
408 else
409     % ziadany je odhad druhého radu
410     met = 2;
411     end
412 end
413
414 % zistime, v akaj metode odhadov pokračovat
415 if MultMethod == 1
416     % ziadany je odhad prveho radu
417     met = 1;
418 else
419     % ziadany je odhad druhého radu
420     met = 2;
421     end
422 end
423
424 % zistime, v akaj metode odhadov pokračovat
425 if MultMethod == 1
426     % ziadany je odhad prveho radu
427     met = 1;
428 else
429     % ziadany je odhad druhého radu
430     met = 2;
431     end
432 end
433
434 % zistime, v akaj metode odhadov pokračovat
435 if MultMethod == 1
436     % ziadany je odhad prveho radu
437     met = 1;
438 else
439     % ziadany je odhad druhého radu
440     met = 2;
441     end
442 end
443
444 % zistime, v akaj metode odhadov pokračovat
445 if MultMethod == 1
446     % ziadany je odhad prveho radu
447     met = 1;
448 else
449     % ziadany je odhad druhého radu
450     met = 2;
451     end
452 end
453
454 % zistime, v akaj metode odhadov pokračovat
455 if MultMethod == 1
456     % ziadany je odhad prveho radu
457     met = 1;
458 else
459     % ziadany je odhad druhého radu
460     met = 2;
461     end
462 end
463
464 % zistime, v akaj metode odhadov pokračovat
465 if MultMethod == 1
466     % ziadany je odhad prveho radu
467     met = 1;
468 else
469     % ziadany je odhad druhého radu
470     met = 2;
471     end
472 end
473
474 % zistime, v akaj metode odhadov pokračovat
475 if MultMethod == 1
476     % ziadany je odhad prveho radu
477     met = 1;
478 else
479     % ziadany je odhad druhého radu
480     met = 2;
481     end
482 end
483
484 % zistime, v akaj metode odhadov pokračovat
485 if MultMethod == 1
486     % ziadany je odhad prveho radu
487     met = 1;
488 else
489     % ziadany je odhad druhého radu
490     met = 2;
491     end
492 end
493
494 % zistime, v akaj metode odhadov pokračovat
495 if MultMethod == 1
496     % ziadany je odhad prveho radu
497     met = 1;
498 else
499     % ziadany je odhad druhého radu
500     met = 2;
501     end
502 end
503
504 % zistime, v akaj metode odhadov pokračovat
505 if MultMethod == 1
506     % ziadany je odhad prveho radu
507     met = 1;
508 else
509     % ziadany je odhad druhého radu
510     met = 2;
511     end
512 end
513
514 % zistime, v akaj metode odhadov pokračovat
515 if MultMethod == 1
516     % ziadany je odhad prveho radu
517     met = 1;
518 else
519     % ziadany je odhad druhého radu
520     met = 2;
521     end
522 end
523
524 % zistime, v akaj metode odhadov pokračovat
525 if MultMethod == 1
526     % ziadany je odhad prveho radu
527     met = 1;
528 else
529     % ziadany je odhad druhého radu
530     met = 2;
531     end
532 end
533
534 % zistime, v akaj metode odhadov pokračovat
535 if MultMethod == 1
536     % ziadany je odhad prveho radu
537     met = 1;
538 else
539     % ziadany je odhad druhého radu
540     met = 2;
541     end
542 end
543
544 % zistime, v akaj metode odhadov pokračovat
545 if MultMethod == 1
546     % ziadany je odhad prveho radu
547     met = 1;
548 else
549     % ziadany je odhad druhého radu
550     met = 2;
551     end
552 end
553
554 % zistime, v akaj metode odhadov pokračovat
555 if MultMethod == 1
556     % ziadany je odhad prveho radu
557     met = 1;
558 else
559     % ziadany je odhad druhého radu
560     met = 2;
561     end
562 end
563
564 % zistime, v akaj metode odhadov pokračovat
565 if MultMethod == 1
566     % ziadany je odhad prveho radu
567     met = 1;
568 else
569     % ziadany je odhad druhého radu
570     met = 2;
571     end
572 end
573
574 % zistime, v akaj metode odhadov pokračovat
575 if MultMethod == 1
576     % ziadany je odhad prveho radu
577     met = 1;
578 else
579     % ziadany je odhad druhého radu
580     met = 2;
581     end
582 end
583
584 % zistime, v akaj metode odhadov pokračovat
585 if MultMethod == 1
586     % ziadany je odhad prveho radu
587     met = 1;
588 else
589     % ziadany je odhad druhého radu
590     met = 2;
591     end
592 end
593
594 % zistime, v akaj metode odhadov pokračovat
595 if MultMethod == 1
596     % ziadany je odhad prveho radu
597     met = 1;
598 else
599     % ziadany je odhad druhého radu
600     met = 2;
601     end
602 end
603
604 % zistime, v akaj metode odhadov pokračovat
605 if MultMethod == 1
606     % ziadany je odhad prveho radu
607     met = 1;
608 else
609     % ziadany je odhad druhého radu
610     met = 2;
611     end
612 end
613
614 % zistime, v akaj metode odhadov pokračovat
615 if MultMethod == 1
616     % ziadany je odhad prveho radu
617     met = 1;
618 else
619     % ziadany je odhad druhého radu
620     met = 2;
621     end
622 end
623
624 % zistime, v akaj metode odhadov pokračovat
625 if MultMethod == 1
626     % ziadany je odhad prveho radu
627     met = 1;
628 else
629     % ziadany je odhad druhého radu
630     met = 2;
631     end
632 end
633
634 % zistime, v akaj metode odhadov pokračovat
635 if MultMethod == 1
636     % ziadany je odhad prveho radu
637     met = 1;
638 else
639     % ziadany je odhad druhého radu
640     met = 2;
641     end
642 end
643
644 % zistime, v akaj metode odhadov pokračovat
645 if MultMethod == 1
646     % ziadany je odhad prveho radu
647     met = 1;
648 else
649     % ziadany je odhad druhého radu
650     met = 2;
651     end
652 end
653
654 % zistime, v akaj metode odhadov pokračovat
655 if MultMethod == 1
656     % ziadany je odhad prveho radu
657     met = 1;
658 else
659     % ziadany je odhad druhého radu
660     met = 2;
661     end
662 end
663
664 % zistime, v akaj metode odhadov pokračovat
665 if MultMethod == 1
666     % ziadany je odhad prveho radu
667     met = 1;
668 else
669     % ziadany je odhad druhého radu
670     met = 2;
671     end
672 end
673
674 % zistime, v akaj metode odhadov pokračovat
675 if MultMethod == 1
676     % ziadany je odhad prveho radu
677     met = 1;
678 else
679     % ziadany je odhad druhého radu
680     met = 2;
681     end
682 end
683
684 % zistime, v akaj metode odhadov pokračovat
685 if MultMethod == 1
686     % ziadany je odhad prveho radu
687     met = 1;
688 else
689     % ziadany je odhad druhého radu
690     met = 2;
691     end
692 end
693
694 % zistime, v akaj metode odhadov pokračovat
695 if MultMethod == 1
696     % ziadany je odhad prveho radu
697     met = 1;
698 else
699     % ziadany je odhad druhého radu
700     met = 2;
701     end
702 end
703
704 % zistime, v akaj metode odhadov pokračovat
705 if MultMethod == 1
706     % ziadany je odhad prveho radu
707     met = 1;
708 else
709     % ziadany je odhad druhého radu
710     met = 2;
711     end
712 end
713
714 % zistime, v akaj metode odhadov pokračovat
715 if MultMethod == 1
716     % ziadany je odhad prveho radu
717     met = 1;
718 else
719     % ziadany je odhad druhého radu
720     met = 2;
721     end
722 end
723
724 % zistime, v akaj metode odhadov pokračovat
725 if MultMethod == 1
726     % ziadany je odhad prveho radu
727     met = 1;
728 else
729     % ziadany je odhad druhého radu
730     met = 2;
731     end
732 end
733
734 % zistime, v akaj metode odhadov pokračovat
735 if MultMethod == 1
736     % ziadany je odhad prveho radu
737     met = 1;
738 else
739     % ziadany je odhad druhého radu
740     met = 2;
741     end
742 end
743
744 % zistime, v akaj metode odhadov pokračovat
745 if MultMethod == 1
746     % ziadany je odhad prveho radu
747     met = 1;
748 else
749     % ziadany je odhad druhého radu
750     met = 2;
751     end
752 end
753
754 % zistime, v akaj metode odhadov pokračovat
755 if MultMethod == 1
756     % ziadany je odhad prveho radu
757     met = 1;
758 else
759     % ziadany je odhad druhého radu
760     met = 2;
761     end
762 end
763
764 % zistime, v akaj metode odhadov pokračovat
765 if MultMethod == 1
766     % ziadany je odhad prveho radu
767     met = 1;
768 else
769     % ziadany je odhad druhého radu
770     met = 2;
771     end
772 end
773
774 % zistime, v akaj metode odhadov pokračovat
775 if MultMethod == 1
776     % ziadany je odhad prveho radu
777     met = 1;
778 else
779     % ziadany je odhad druhého radu
780     met = 2;
781     end
782 end
783
784 % zistime, v akaj metode odhadov pokračovat
785 if MultMethod == 1
786     % ziadany je odhad prveho radu
787     met = 1;
788 else
789     % ziadany je odhad druhého radu
790     met = 2;
791     end
792 end
793
794 % zistime, v akaj metode odhadov pokračovat
795 if MultMethod == 1
796     % ziadany je odhad prveho radu
797     met = 1;
798 else
799     % ziadany je odhad druhého radu
800     met = 2;
801     end
802 end
803
804 % zistime, v akaj metode odhadov pokračovat
805 if MultMethod == 1
806     % ziadany je odhad prveho radu
807     met = 1;
808 else
809     % ziadany je odhad druhého radu
810     met = 2;
811     end
812 end
813
814 % zistime, v akaj metode odhadov pokračovat
815 if MultMethod == 1
816     % ziadany je odhad prveho radu
817     met = 1;
818 else
819     % ziadany je odhad druhého radu
820     met = 2;
821     end
822 end
823
824 % zistime, v akaj metode odhadov pokračovat
825 if MultMethod == 1
826     % ziadany je odhad prveho radu
827     met = 1;
828 else
829     % ziadany je odhad druhého radu
830     met = 2;
831     end
832 end
833
834 % zistime, v akaj metode odhadov pokračovat
835 if MultMethod == 1
836     % ziadany je odhad prveho radu
837     met = 1;
838 else
839     % ziadany je odhad druhého radu
840     met = 2;
841     end
842 end
843
844 % zistime, v akaj metode odhadov pokračovat
845 if MultMethod == 1
846     % ziadany je odhad prveho radu
847     met = 1;
848 else
849     % ziadany je odhad druhého radu
850     met = 2;
851     end
852 end
853
854 % zistime, v akaj metode odhadov pokračovat
855 if MultMethod == 1
856     % ziadany je odhad prveho radu
857     met = 1;
858 else
859     % ziadany je odhad druhého radu
860     met = 2;
861     end
862 end
863
864 % zistime, v akaj metode odhadov pokračovat
865 if MultMethod == 1
866     % ziadany je odhad prveho radu
867     met = 1;
868 else
869     % ziadany je odhad druhého radu
870     met = 2;
871     end
872 end
873
874 % zistime, v akaj metode odhadov pokračovat
875 if MultMethod == 1
876     % ziadany je odhad prveho radu
877     met = 1;
878 else
879     % ziadany je odhad druhého radu
880     met = 2;
881     end
882 end
883
884 % zistime, v akaj metode odhadov pokračovat
885 if MultMethod == 1
886     % ziadany je odhad prveho radu
887     met = 1;
888 else
889     % ziadany je odhad druhého radu
890     met = 2;
891     end
892 end
893
894 % zistime, v akaj metode odhadov pokračovat
895 if MultMethod == 1
896     % ziadany je odhad prveho radu
897     met = 1;
898 else
899     % ziadany je odhad druhého radu
900     met = 2;
901     end
902 end
903
904 % zistime, v akaj metode odhadov pokračovat
905 if MultMethod == 1
906     % ziadany je odhad prveho radu
907     met = 1;
908 else
909     % ziadany je odhad druhého radu
910     met = 2;
911     end
912 end
913
914 % zistime, v akaj metode odhadov pokračovat
915 if MultMethod == 1
916     % ziadany je odhad prveho radu
917     met = 1;
918 else
919     % ziadany je odhad druhého radu
920     met = 2;
921     end
922 end
923
924 % zistime, v akaj metode odhadov pokračovat
925 if MultMethod == 1
926     % ziadany je odhad prveho radu
927     met = 1;
928 else
929     % ziadany je odhad druhého radu
930     met = 2;
931     end
932 end
933
934 % zistime, v akaj metode odhadov pokračovat
935 if MultMethod == 1
936     % ziadany je odhad prveho radu
937     met = 1;
938 else
939     % ziadany je odhad druhého radu
940     met = 2;
941     end
942 end
943
944 % zistime, v akaj metode odhadov pokračovat
945 if MultMethod == 1
946     % ziadany je odhad prveho radu
947     met = 1;
948 else
949     % ziadany je odhad druhého radu
950     met = 2;
951     end
952 end
953
954 % zistime, v akaj metode odhadov pokračovat
955 if MultMethod == 1
956     % ziadany je odhad prveho radu
957     met = 1;
958 else
959     % ziadany je odhad druhého radu
960     met = 2;
961     end
962 end
963
964 % zistime, v akaj metode odhadov pokračovat
965 if MultMethod == 1
966     % ziadany je odhad prveho radu
967     met = 1;
968 else
969     % ziadany je odhad druhého radu
970     met = 2;
971     end
972 end
973
974 % zistime, v akaj metode odhadov pokračovat
975 if MultMethod == 1
976     % ziadany je odhad prveho radu
977     met = 1;
978 else
979     % ziadany je odhad druhého radu
980     met = 2;
981     end
982 end
983
984 % zistime, v akaj metode odhadov pokračovat
985 if MultMethod == 1
986     % ziadany je odhad prveho radu
987     met = 1;
988 else
989     % ziadany je odhad druhého radu
990     met = 2;
991     end
992 end
993
994 % zistime, v akaj metode odhadov pokračovat
995 if MultMethod == 1
996     % ziadany je odhad prveho radu
997     met = 1;
998 else
999     % ziadany je odhad druhého radu
1000     met = 2;
1001     end
1002 end

```

```

391 if length(ub) > varc
392 warning('kontrolaobr:ubrozmer', 'length(ub) > pocet premennych, prebytočne ignorujem. ');
393 ub = ub(1:varc);
394 elseif length(ub) < varc
395 ub = [ub; Inf*ones(varc - length(ub), 1)];
396 end
397
398 if length(lb) > varc
399 warning('kontrola:lbrozmer', 'length(lb) > pocet premennych, prebytočne ignorujem. ');
400 lb = lb(1:varc);
401 elseif length(lb) < varc
402 lb = [lb; -Inf*ones(varc - length(lb), 1)];
403 end
404
405 if any(lb>ub)
406 error('Nekonzistentne ohranicenia, niektore z dolnych ohraniceni je\su vacsie ako horne');
407 elseif any(lb == Inf)
408 error('Dolne ohranicenie nemoze byt Inf');
409 elseif any(ub == -Inf)
410 error('Horne ohranicenie nemoze byt -Inf');
411 end
412
413 Iden = eye(varc);
414 Aap = [Iden(isfinite(lb), :); -Iden(isfinite(ub), :)];
415 bap = [lb(isfinite(lb)); -ub(isfinite(ub))];
416
417 PSI_I = setting(options, 'PsiFunction', defaults);
418 PSI_E = 'psi_E';
419
420 % v pripade, ze A, Aeq, b, beq su prazdne, prenavstavime na spravne
421 % rozmary
422 if isempty(A)
423     A = zeros(0, n);
424 end
425 if isempty(Aeq)
426     Aeq = zeros(0, n);
427 end
428 if isempty(b)
429     b = zeros(0, 1);
430 end
431 if isempty(beq)
432     beq = zeros(0, 1);
433 end
434
435 % zistime rozmary kvaratickych ohraniceni
436 size_g = size(Gin, 2);
437 if size_g == mq
438     gdiag = 1;
439 else
440     size_g = mqpn;
441     gdiag = 0;
442 end
443 error('Matica Gin nema pozadovane rozmary');
444
445 % startovacie hodnoty multiplikatorov
446 y = ones(m, 1);
447 z = zeros(meq, 1);
448 if isempty(x0)
449     x = zeros(n, 1);
450 else
451     x = x0;
452 end
453 warning('UlohaU3:x0rozmer', 'Vektor x0 ma iny rozmer ako n, zacinam z [0; ...; 0]');
454 x = x0(:);
455
456 output = struct('HlavneIteracie', 0, 'NewtonovskeIteracie', 0, 'Tolerancia', epsilon, ...
457     'Merit', Inf, 'VypoctovyCas', 0);
458 MainIteration = 0;
459 NewtonIteration = 0;
460 NormType = Inf; NormTypeMerit = 'max';
461 % spustime meranie casu

```

```

1 function [xopt, yopt, zopt, output] = polysak_bq_q(D,G,h,Gin,hin,rin,A,b,...
2     Aeq,beq,x0,varargin)
3
4 % kontrola nezadanych vstupov
5 if nargin < 12
6     options = [];
7     if nargin < 11
8         x0 = [];
9     end
10 end
11
12 b = b(:);
13 h = h(:);
14 beq = beq(:);
15 rin = rin(:);
16
17 % kontroly rozmerov ulohy
18 n = max([length(G), length(h), length(D)]);
19 mq = length(cin);
20 ml = length(b);
21 meq = length(beq);
22 m = ml + mq;
23
24 if (~all(size(A) == [ml, n]) && ~isempty(A))
25     error('Matica A ma nespravny rozmer, ocakavam (%u x %u)', ml, n);
26 elseif (~all(size(Aeq) == [meq, n]) && isempty(Aeq))
27     error('Matica Aeq ma nespravny rozmer, ocakavam (%u x %u)', meq, n);
28 elseif (length(D) == n && ~isempty(D)) || (length(h) == n && ~isempty(h))
29     error('Diagonalna matica D alebo vektor h maju nespravny rozmer');
30 elseif ~all(size(hin) == [n, mq])
31     error('Matica HIN ma nespravny rozmer');
32 end
33
34 % nastavenia parametrov
35 if ~isempty(varargin)
36     options = setoptions(varargin);
37 end
38
39 defaults = struct('Tolerancia', 1e-06, 'HlavneIteracieMax', 100, ...
40     'NewtonovskeIteracieMax', 50, 'RParameter', 10, ...
41     'Verbosity', 0, 'PsiFunction', 'psi_I_bvp');
42 MaxMainIteration = setting(options, 'PsiFunction', defaults);
43 MaxNewtonIteration = setting(options, 'HlavneIteracieMax', defaults);
44 pen_k = setting(options, 'NewtonovskeIteracieMax', defaults);
45 epsilon = setting(options, 'RParameter', defaults);
46 time = setting(options, 'Tolerancia', defaults);
47 verb = setting(options, 'Verbosity', defaults);

```

Funkcia Polyakovho a Grivovho algoritmu

```

95 if Time == 0
96 tic;
97 end
98
99 [fin, feq] = const_quad(x, G_in, hin, rin, A, b, Aeq, beq, gdiag);
100 rval = merit(LagrClass_quad(x, y, z, D, G, h, G_in, hin, rin, A, b, Aeq, beq, ...
101 'grad'), fin, feq, y, NormTypeMerit);
102 % parametre
103 con_viol = const_viol(fin, feq, NormTypeMerit);
104 rho_des = 0.25;
105 pen_max = 5*1e5;
106 pen_max_mupd = 6;
107 L = 0.9;
108 grel_toll_cls = max(sqrt(norm(h, NormType)), 1);
109 grel_toll = sqrt(pen_k) * grel_toll_cls;
110 grad_flag = 0;
111
112 alg = 'MINIM';
113 gamma = 0.75;
114
115 for k=1:MaxMainIteration % HLAVNA ITERACIA
116
117 % zacneme novu iteraciu. Na zaklade hodnoty merit funkcie vykoname
118 % minimalizacny krok klasického Aug-lag algoritmu, alebo spustime
119 % Polyakov PD algoritmus
120 MainIteration = MainIteration + 1;
121
122 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
123 if verb
124 fprintf('h***** ITERACIA %u *****\n', k);
125 fprintf('<PRIP. A KOMPLEMENTARITA> %e\n', con_viol);
126 fprintf('<NORMA GRAD. KLAS. LAGR.> %e\n', norm(LagrClass_quad(x, ...
127 y, z, D, G, h, G_in, hin, rin, A, b, Aeq, beq, 'grad'), NormType));
128
129 end
130 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
131 % skalovany gradient v merit funkcii
132 if rval <= epsilon && grad_flag
133 break % output x, y
134 end
135
136 % na tomto mieste sa pokusime vyriesit Polyakovu sustavu rovnice a
137 % updatovat vektory x, y, z pomocou smerov ziskanych z tejto sustavy
138 if strcmp(alg, 'POLYAK')
139 % ziskame riesenie PD sustavy a odhadneme nove vektory
140 [dx, dy, dz] = PD_direction(x, y, z, D, G, h, G_in, hin, rin, A, b, Aeq, beq, ...
141 pen_k, PSI_I);
142
143 xt = x + dx;
144 yt = y + dy;
145 zt = z + dz;
146 NewtonIteration = NewtonIteration + 1;
147
148 % ziskame hodnoty ohraneni v týchto bodoch, a nasledne vypocitame
149 % hodnotu merit funkcie
150 [fin, feq] = const_quad(xt, G_in, hin, rin, A, b, Aeq, beq, gdiag);
151 rval_tmp = merit_comp(fin, feq, yt, NormTypeMerit);
152 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
153 if verb
154 fprintf('POLYAK PD. MERIT> %e\n', rval_tmp);
155 end
156
157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
158 % zistime, ci doslo k pozadovanemu poklesu merit funkcie
159 if rval_tmp < min([rval-(1.15), 0.65]) % KROK PRIJATY
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb
fprintf('<PRIJIMAM POLYAK. RIESENIE>\n');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x = xt;
y = yt;
z = zt;
rval = rval_tmp;
pen_k = max(1 / rval, pen_k);
grel_toll = sqrt(pen_k / pen_kt) * grel_toll;
con_viol = const_viol(fin, feq, NormTypeMerit);
else
alg = 'MINIM';
end
end
% strmpi(alg, 'MINIM')
% nastava faza minimalizacie AugLag funkcie na priestore premennej
% x, po vykonani tejto minimalizacie pouzijeme na vektory
% multiplikatorov odhady prveho radu
[fin, feq] = const_quad(x, G_in, hin, rin, A, b, Aeq, beq, gdiag);
% prediktory multiplikatorov
yt = - feval(PSI_I, pen_k*fin, y, 'der1');
zt = feval(PSI_E, pen_k*feq, z, 'der1');
% tolerancia pre Newtonovu metodu
upd_diff = norm([yt - y; zt - z], NormType);
toll = min(L / pen_k * upd_diff, 0.25);
% vypocet gradientu a Hesseovej matice
[G, H] = LagrAug_quad(x, y, z, D, G, h, G_in, hin, rin, A, b, Aeq, beq, pen_k, ...
PSI_E, PSI_I, 'gradhess');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb
fprintf('<TERM. KRITERIUM NEWTON M.> %e\n', toll + epsilon*...
grel_toll);
fprintf('<PRIJIMAM MINIMALIZACNY ALGORITMUS>\n');
fprintf('<NORMA GRADIENTU AUGLAG F.> %e\n', norm(g, NormType));
fprintf('***** NEWTON. ITERACIA *****\n');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:MaxNewtoniter
% podmienka konvergence pre Newtonovu iteraciu
if ( norm(g, NormType) < toll + epsilon*grel_toll) && j > 1
break
end
% spocitame Newtonovsky krok S, krok LAM stanovime ciastocnou
% minimalizaciou na lucci X + LAM*S niektorou z dostupnych metod
dx = -H \ g;
lam = brenterc(x, y, z, D, G, h, G_in, hin, rin, A, b, Aeq, beq, pen_k, dx, ...
PSI_E, PSI_I);
% upravime primarny vektor x
x = x + lam*dx;
[fin, feq] = const_quad(x, G_in, hin, rin, A, b, Aeq, beq, gdiag);
% prediktory multiplikatorov
yt = - feval(PSI_I, pen_k*fin, y, 'der1');

```

```

229 zt = feval(Psi_E, pen_k*feq, z, 'der1');
230
231 % prepočitame tolerancne kriterium
232 upd_diff = norm([ yt - y; zt - z], NormType);
233
234 toll = min(L / pen_k * upd_diff, 0.25);
235
236 % získame nove data na urcenie smeru DX
237 [g, H] = LagrAug_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,pen_k,...
238     Psi_E, Psi_I, 'gradhes');
239
240 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
241 if verb
242     fprintf(' <TERM. KRITERIUM> %e\n', toll + epsilon*grel_toll);
243     fprintf(' <DLZKA KROKU> %f, <NORMA GRAD. AUGLAG. F.> %e\n', ...
244         lam, norm(g, NormType));
245 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
246
247 NewtonIteration = NewtonIteration + 1;
248 end
249
250 %% vypocitame hodnotu merit funkcie v novom iteracnom bode %%
251 rval_tmp = merit_comp(fin,feq,yt,NormTypeMerit);
252
253 z = yt;
254 y = zt;
255
256 % podla poklesu merit funkcie urcime, aku cast algoritmu vykoname
257 % dalej
258 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
259 if verb
260     fprintf('***** KONEC NEWT. ITER. *****\n');
261     fprintf(' <POCET NEWTON. ITERACII> %d\n', j - 1);
262     fprintf(' <VYSLEDNA NORMA GRAD. AUGLAG. F.> %3.4e\n', norm(g, NormType));
263 end
264 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
265
266 con_viol_old = con_viol;
267 con_viol = con_viol(fin, feq, NormTypeMerit); % maximum norm
268 rho = con_viol / con_viol_old;
269 if rho > rho_des
270     pen_kt = min([2 * pen_k * rho / rho_des, pen_max_mupd * pen_k, pen_max]);
271 else
272     pen_kt = pen_k;
273 end
274
275 if rval_tmp < gamma * rval % DOSTATOCNY POKLES, POLYAK. ALG.
276     alg = 'POLYAK';
277     rval = rval_tmp;
278     pen_kt = max(1 / rval, pen_kt);
279     grel_toll = sqrt(pen_kt / pen_k) * grel_toll;
280     pen_k = pen_kt;
281 else
282     % NEDOSTATOCNY POKLES, MINIMAL. ALG.
283     % Zistime, ci treba zvyisit hodnotu penalizacneho parametra r na
284     % zaklade zmeny pripustnosti aktualneho bodu
285     alg = 'MINIM';
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
grel_toll = sqrt(pen_kt / pen_k) * grel_toll;
pen_k = pen_kt;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb
fprintf(' <PORUSENIE PRIJISTNOSTI> %e\n', con_viol);
fprintf(' <PORUSENIE KOMPLEMENTARITY> %e\n', sum(abs(y.*fin)));
fprintf(' <PENALIZ. PARAMETER> %f\n', pen_k);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if norm(LagrClass_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'grad'), ...
    NormType) < grel_toll * epsilon
    grad_flag = 1; % optimalny gradient, 1 KKT podmienka
else
    grad_flag = 0; % gradient nedosahuje pozadovane hod.
end
end
307 end
308
309 % meranie casu ukoncene
310 if Time
311     output.VpocetoVyCas = toc;
312 end
313
314 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
315 if verb
316     fprintf(' <CELKOVY POCET ITERACII> %d\n', k - 1);
317     fprintf(' <CELKOVY POCET NEWT. ITERACII> %d\n', NewtonIteration);
318     fprintf(' <CASOVA NAROCNOST V SEK.> %f\n', output.VpocetoVyCas);
319 end
320 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
321
322 xopt = x;
323 yopt = y;
324 zopt = z;
325 output.HlavneIteracie = k - 1;
326 output.NewtonovskeIteracie = NewtonIteration;
327 output.Merit = rval
328
329 function [dx,dy,dz] = PD_direction(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,...
330     pen_k,Psi_I)
331
332 % funkcia riesi Polyakovu sustavu, predpoklada sa Psi_E = 'psi_E'
333 n = length(x);
334
335 [fin,feq,jin,Je] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq);
336 D_xixi = diag(feval(Psi_I, pen_k*fin, y, 'der2'));
337 ybar = - feval(Psi_I, pen_k*fin, y, 'der1');
338 zbar = feval('psi_E', pen_k*feq, z, 'der1');
339
340 Lxx = LagrClass_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'hess') + ...
341     1/pen_k^2 * eye(n);
342 Lx = LagrAug_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,pen_k,...
343     'psi_E', Psi_I, 'grad');
344 dx = chololve([Lxx+1/pen_k^2*eye(n) pen_k*jin *D_xixi*jin+pen_k*(Je'*Je)],Lx);
345 dy = ybar-y;pen_k*D_xixi*jin+dz;
346 dz = zbar-z;pen_k*Je+dx;

```


Funkcia modifikovaného Fischerovho algoritmu.

```

1 function [xopt, yopt, zopt, output] = fisher2_bq(D,G,h,Gin,hin,rin,A,b,Aeq,beq,...
2     x0,varargin)
3
4 % kontrola nezadaných vstupov
5 if nargin < 12
6     options = [];
7     if nargin < 11
8         x0 = [];
9     end
10 end
11
12 b = b(:);
13 h = h(:);
14 beq = beq(:);
15 rin = rin(:);
16
17 % kontroly rozmerov ulohy
18 n = max(length(G), length(h), length(D));
19 m = length(rin);
20 ml = length(b);
21 meq = length(beq);
22 m = ml + meq;
23
24 if (~all(size(A) == [ml, n]) && isempty(A))
25     error('Matica A ma nespravny rozmer, ockavam (%u x %u)', ml, n);
26 elseif (~all(size(Aeq) == [meq, n]) && isempty(Aeq))
27     error('Matica Aeq ma nespravny rozmer, ockavam (%u x %u)', meq, n);
28 elseif (length(D) ~= n && isempty(D)) || (length(h) ~= n && isempty(h))
29     error('Diagonálna matica D alebo vektor h maju nespravny rozmer');
30 elseif ~all(size(hin) == [n, meq])
31     error('Matica HIN ma nespravny rozmer');
32 end
33
34 % nastavenia parametrov
35 if isempty(varargin)
36     options = setoptions(varargin);
37 end
38
39 defaults = struct('Tolerancia', 1e-06, 'HlavneIteracieMax', 100, ...
40     'NewtonovskeIteracieMax', 50, 'Rparameter', 10, 'CasVypoctov', ...
41     1, 'Verbosity', 0, 'PsiFunction', 'psi_hyp');
42
43 MaxMainIter = setting(options, 'HlavneIteracieMax', defaults);
44 MaxNewtonIter = setting(options, 'NewtonovskeIteracieMax', defaults);
45 epsilon = setting(options, 'Rparameter', defaults);
46 Time = setting(options, 'Tolerancia', defaults);
47 verb = setting(options, 'CasVypoctov', defaults);
48 PSI_I = setting(options, 'Verbosity', defaults);
49 PSI_E = 'psi_E';
50
51 % v pripade, ze A, Aeq, b, beq su prazdne, prenavstavime na spravne
52 % rozmary
53 if isempty(A)
54     A = zeros(0, n);
55 end
56 if isempty(Aeq)
57     Aeq = zeros(0, n);
58 end
59 if isempty(b)
60     b = zeros(0, 1);
61 end
62 if isempty(beq)
63     beq = zeros(0, 1);
64 end
65
66 % zistime rozmary kvaratických ohraniceni
67 size_g = size(Gin, 2);
68 if size_g == mq
69     gdiag = 1;
70 elseif size_g == mq*n
71     gdiag = 0;
72 else
73     error('Matica Gin nema pozadovane rozmary');
74 end
75
76 % startovacie hodnoty multiplikatorov
77 y = ones(n, 1);
78 z = zeros(meq, 1);
79 if isempty(x0)
80     x = zeros(n, 1);
81 elseif length(x0) == n
82     x = zeros(n, 1);
83     warning('UI:ohau3:x0Rozmer', 'Vektor x0 ma iny rozmer ako n, zacinam z [0; ... ;0]');
84 else
85     x = x0(:);
86 end
87
88 output = struct('HlavneIteracie', 0, 'NewtonovskeIteracie', 0, ...
89     'Tolerancia', epsilon, 'Epsilon', Inf, 'VypoctovyCas', 0);
90 NormType = Inf; NormTypeMerit = 'max';
91 NewtonIt = 0;
92
93 % spustime meranie casu
94 if Time == 0
95     tic;
96 end
97
98 % definovanie pouzitych konstant pri vypoctoch
99 [fin, feq, lin, leq] = const_quad(x, Gin, hin, rin, A, b, Aeq, beq, gdiag);
100 con_viol = const_viol(fin, feq, NormTypeMerit);
101 rho_des = 0.5; % ockavany pokles merit funkcie
102 r_max = 5*eps; % maximalny parameter r
103 r_max_mupd = 6;
104 L = 0.9; % konstanta pri vypocte tolerancie toll
105 min_toll = 0.25; % TOLL = min(TULL1, min_toll) + TULL2
106 grel_toll_cls = max(sqrt(norm(h, NormType)), 1);
107 grel_toll = sqrt(r) * grel_toll_cls;
108 grad_flag = 0;
109 nu = 2 * eps;
110 flag = 0;
111 mrt = merit_comp(fin, feq, y, NormTypeMerit);
112 mrt_thold = 0.1; % uroven prepnutia algoritmu
113
114 for k=1:MaxMainIter
115     % hlavna iteracia algoritmu. V prvej fazi riesime minimaliacnu cast,
116     % spustame klasicky AugLag algoritmus. Ak hodnota merit funkcie
117     % poklesne pod urcitu kriticku uroven kontrolovane parametrom
118     % MRT_THOLD, zmenime vykonavany algoritmus na Fischerov.
119
120     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121     if verb
122         fprintf('\n***** ITERACIA %u *****\n', k);
123         fprintf('<PRIP. A KOMPLEMENTARITA> %e\n', mrt);
124         fprintf('<OROVA GRD. KLAS. LAGR.> %e\n', norm(LagrClass_quad(x,...
125             y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq, 'grad'), NormType));
126     end
127     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
128

```



```

129 % skalovany gradient v merit funkcii
130 if mrt < epsilon && grad_flag
131     break
132 end
133
134 % podla identifikatora FLAG urcime, ktory algoritmus sa bude vykonavat
135 if flag == 1
136     % vykoname Fischerovu cast algoritmu pre klasicku Lagr. funkciu
137     [Lx, Lxx] = LagrClass_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'gradhess');
138
139     % najdeme korekcie [dx, dy, dz] riesenim Fischerovej sustavy
140     [dx,dy,dz,prec] = solvesystem(Lxx,Lx,fin,feq,Jin,Jeq,y,r,mm,PSI_I);
141     NewtonIt = NewtonIt + 1;
142
143     % minimalizacia normy Fischerovej sustavy
144     % smer = [dx;dy,dz];
145     lam = 1;
146
147     % predikcia noveho bodu
148     x = x + lam*dx;
149     y = max(0, y + lam*dy);
150     z = z + lam*dz;
151
152     % vypocitame hodnotu ohraničeni a zisitime zmenu pripustnosti
153     [fin, feq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
154     % a komplementarity v novom bode
155     [Jin,feq,Jin,Jeq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
156     mrt = merit_comp(fin,feq,y, NormTypeMerit);
157     con_viol = const_viol(fin, feq, NormTypeMerit);
158
159     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
160     if verb
161         fprintf('<PREZNOST RIESENIA SUSTAVY> %e\n', prec);
162         fprintf('<NOVA HODNOTA PRIPUSTNOSTI> %e\n', mrt);
163     end
164     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
165
166 else
167     % budeme minimalizovat AugLag v x a pouzijeme 1st order multiplier
168     % update, zvycajne prvych niekoľko iteraci sa vykona v rezime
169     % minimalizacie, pokym merit funkcia nebude dostatočne mala
170
171     % stanovime terminačne kriterium pomocou rozdielu medzi prediktorem
172     % multiplikatora a konkrétnym multiplikatorom, predeľeny príslušným
173     % penalizačným parametrom r
174     yt = - feval(PSI_I,r*fin,y,'der1');
175     zt = feval(PSI_E,r*feq,z,'der1');
176
177     upd_diff = norm([yt - y; zt - z], NormType );
178     toll = min(L / r * upd_diff, min_toll);
179
180     [Lx, Lxx] = LagrAug_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r,...
181     PSI_E,PSI_I,'gradhess');
182
183     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
184     if verb
185         fprintf('<TERM. KRITERIUM NEWTON M.> %e\n', toll + epsilon*grl_toll);
186         fprintf('<DILZKA KROKU> %f, <NORMA GRAD. AUGLAG. F.> %e\n', ...
187         lam, norm(Lx, NormType));
188     end
189     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VYPISY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
190
191     for j=1:MaxNewtonIter % ZACIATOK NEWTONOVSKJEJ ITERACIE
192         % podmienka konvergenencie pre Newtonovsku iteraciju
193         if verb
194
195

```



```

1 function roptions = setoptions(index)
2
3 % funkcia sluzi na spracovanie volitelneho vstupu do testovacich
4 % funkci ULOHA3, ULOHAU3PHR, FISCHER, FISCHER2, POLYAK, POLYAK2
5
6 % uprava vstupu
7 n = numel(index);
8 input = reshape(index, 1, n);
9
10 if n == 1
11 % na vstupe je jeden parameter, musi byt STRUCT
12 if isstruct(input{1})
13 roptions = input{1};
14 else
15 error('Pole parametrov OPTIONS musi byt typu struct');
16 end
17 else
18 % vstup je organizovany ako sled dvojic, kde prvý prvok je názov
19 % parametra (alebo pole parametrov) a druhý prvok udáva jeho (ich)
20 % hodnotu. V prípade, že je parametrov alebo hodnôt rozny počet,
21 % prebytočne sa ignorujú
22 if mod(n, 2) == 1
23 warning('SetOptions:nSize', 'Rozne rozmery vstupnych poli, prebytočne ignorujem');
24 input(n) = [];
25 end
26 for i=1:n/2
27 % nacitame nazov parametra a jeho hodnotu
28 if iscell(input{2*i-1})
29 proc = input{2*i-1}; % parameter
30 else
31 proc = input{2*i-1};
32 end
33 if iscell(input{2*i})
34 val = input{2*i}; % hodnota parametra
35 else val = input{2*i};
36 end
37
38 NumOfVals = length(val);
39 NumOfProc = length(proc);
40
41 % Polia PROC a VAL zvektorizujeme preskupenim udajov
42 val = reshape(val, 1, NumOfVals);
43 try
44 proc = reshape(strtrim(proc), 1, NumOfProc);
45 catch err
46 if strcmp(err.identifier, 'MATLAB:strtrim:InputClass')
47 error('Nazvy parametrov musia byt typu string');
48 else
49 rethrow(err);
50 end
51 end
52
53
54 % ak maju nacitane polia rozne rozmery, prebytočne udaje ignorujeme
55 if NumOfVals ~= NumOfProc
56 warning('SetOptions:nSize', '[Rozne rozmery nazov parametrov', ...
57 % a ich hodnot, prebytočne ignorujem.]);
58 end
59
60 % prejdeme všetky prvky poli PROC a VAL a vytvorime strukturu k
61 % ďalšiemu spracovaniu
62 for j = 1:min(NumOfVals, NumOfProc)
63 tproc = proc{j}; tval = val{j};
64 if iscell(tproc) || iscell(tval)
65 error('Prilis mnoho vnorených objektov typu cell');
66 end
67
68 % vytvorime prvok v strukture roptions s nazvom tproc a
69 % hodnotou tval
70 roptions.(proc{j}) = val{j};
71 end
72 end
73 end
74
75 function hod = setting(opts, name, def)
76
77 % funkcia na získanie nastavení zo struktury OPTS. NAME je identifikátor
78 % parametra, ktorý hľadáme, a DEF je jeho defaultná hodnota (struktúra
79 % DEF musí obsahovať pole NAME).
80
81 if ~isfield(def, name)
82 error('Struktúra DEF neobsahuje pole NAME');
83 end
84 if isempty(opts)
85 hod = def.(name);
86 return;
87 end
88
89 % najdeme ziadanu hodnotu parametra
90 NameInStr = fieldnames(opts); % všetky mozne uvazovane polia
91 NumberOfEntries = length(NameInStr);
92 for i=1:NumberOfEntries
93 NameToCmp = NameInStr{i};
94 ind = find(strcmp(NameToCmp, name, length(NameToCmp)));
95 if ~isempty(ind)
96 ind = i;
97 break;
98 end
99 end
100
101 % ak sme ju nasi, vytiahneme hodnotu z pola OPTS
102 if isempty(ind)
103 hod = def.(name);
104 else
105 hod = opts.(NameToCmp);
106 end

```

Funkcie na výpočet klasickej a rozšírenej Lagrangeovej funkcie, a funkcia na výpočet ohraničení a ich Jakobianov (funkcie pre balík CUTer sú analogické).

```

1 function varargout = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag)
2
3 % [fin, feq, Jin, Jeq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag)
4 %
5 % funkcia pocita funkcie hodnoty a Jakobiani ohraniceni.
6 % Pocet vstupnych parametrov 1-4.
7 %
8 % fin - funkcia hodnoty ohraniceni v tvare nerovnosti
9 % feq - funkcia hodnoty ohraniceni v tvare rovnosti
10 % Jin - Jakobian ohraniceni v tvare nerovnosti
11 % Jeq - Jakobian ohraniceni v tvare rovnosti
12
13 if nargin < 9
14     gdiag = [];
15
16 if nargin < 8
17     error('Funkcia const_quad potrebuje 8 vstupnych argumentov');
18 end
19
20 % rozmery prislusnych matic
21 sz_n = length(x);
22 sz_mq = length(rin);
23 sz_m = sz_mq + size(A, 1);
24
25 % kontrola rozmerov matice Gin
26 if isempty(gdiag)
27     sz_g = size(Gin, 2);
28     if sz_g == sz_mq;
29         gdiag = 1;
30     elseif sz_g == sz_mq * sz_n
31         gdiag = 0;
32     else
33         error('Matica Gin nema pozadovane rozmery');
34     end
35 end
36
37 fin = zeros(sz_m, 1);
38 if nargin < 3
39     % vypocet kvadratickych ohraniceni
40     if gdiag == 1
41         for i=1:sz_mq
42             fin(i) = x*(0.5*Gin(i, i)*x + hin(i, i)) - rin(i);
43         end
44     else
45         for i=1:sz_mq
46             fin(i) = x*(0.5*Gin(i, i)*x + hin(i, i)) - rin(i);
47         end
48     end
49
50 % priradenie linearnych ohraniceni
51 fin(sz_mq+1:sz_m) = A*x-b;
52 feq = Aeq*x-beq;
53
54 % priradenie vstupu
55 varargout{1} = fin;
56 varargout{2} = feq;
57 else
58     % vypocet kvadratickych ohraniceni
59     Jin = zeros(sz_m, sz_n);
60     if gdiag == 1
61         for i=1:sz_mq
62             temp = Gin(i, i).*x;
63             fin(i) = x*(0.5*temp + hin(i, i)) - rin(i);
64             Jin(i, :) = (temp + hin(i, i));
65         end
66     else
67         for i=1:sz_mq

```

```

68         temp = Gin(i, sz_n*(i-1)+1:sz_n*i)*x;
69         fin(i) = x*(0.5*temp + hin(i, i)) - rin(i);
70         Jin(i, :) = (temp + hin(i, i));
71     end
72 end
73
74 % priradenie linearnych ohraniceni
75 fin(sz_mq+1:sz_m) = A*x-b;
76 Jin(sz_mq+1:sz_m, :) = A;
77 feq = Aeq*x-beq;
78 Jeq = Aeq;
79
80 % priradenie vstupu
81 varargout{1} = fin;
82 varargout{2} = feq;
83 varargout{3} = Jin;
84 varargout{4} = Jeq;
85 end

```

```

1 function varargout = LagrClas_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,met)
2
3 % funkcia pocita funkciu hodnotu, gradient alebo Hessovu maticu
4 % klasickej Lagrangeovej funkcie definovanej ako
5 %  $L(x,y,z) = f(x) - y'*g(x) + z'*h(x)$ 
6 % navratove hodnoty zavisi od hodnoty premennej met, ktora moze byt
7 % PHOD - pocita sa funkcia hodnoty Lagrangeovej funkcie
8 % GRAD - pocita sa gradient Lagrangeovej funkcie
9 % HESS - pocita sa Hessova matica Lagrangeovej funkcie
10 % GRADHESS - pocita sa gradient a Hessova matica Lagrangeovej funkcie
11 % FHODGRAD - pocita sa funkcia hodnoty a gradient Lagrangeovej funkcie
12
13 if nargin < 14
14     error('Funkcia LAGRCLAS_QUAD potrebuje 14 vstupnych argumentov');
15 end
16
17 % kontrola vstupnej premennej met
18 switch lower(met)
19     case 'fhod'
20         typ = 1;
21     case 'grad'
22         typ = 2;
23     case 'hess'
24         typ = 3;
25     case 'gradhess'
26         typ = 4;
27     case 'fhodgrad'
28         typ = 5;
29     otherwise
30         error('Met musi byt FHOD, GRAD, HESS, FHODGRAD, alebo GRADHESS');
31 end
32
33 % kontrola vstupnych parametrov
34 if typ < 4 && nargin > 1
35     error('Prilis vela vstupnych argumentov');
36 elseif typ >= 4 && nargin > 2
37     error('Prilis vela vstupnych argumentov');
38 end
39
40 % prislusne rozmery ulohy
41 sz_n = length(x);
42 sz_meq = length(z);
43 sz_mq = max(size(hin, 2), length(rin));
44
45 % kontrola rozmerov
46 if length(D) ~= sz_n && ~isempty(D)
47     error('Diagonala matice D ma nespravny rozmer');

```

```

48 elseif max(length(b), size(A, 1)) ~= length(y) - sz_mq
49 error('Vektor b alebo matrica A maju nespravny rozmer');
50 elseif max(length(beq), size(Aeq, 1)) ~= sz_meq
51 error('Vektor beq alebo matrica Aeq maju nespravny rozmer');
52 elseif max(length(G), length(h)) ~= sz_n
53 error('Vektor h alebo matrica G maju nespravny rozmer');
54 end
55
56 % kontrola a identifikacia zadania matice Gin
57 sz_g = size(Gin, 2);
58 if sz_g == sz_mq
59 gdiag = 1;
60 elseif sz_g == sz_n * sz_mq
61 gdiag = 0;
62 else
63 error('Matrica Gin ma nespravny rozmer');
64 end
65
66 % vykonanie pozadovanych vypoctov
67 if typ == 1
68 % pocita sa funkčna hodnota Lagrangeovej funkcie
69 [fin, feq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
70 res = x*(0.5*G*x + h) + feq*z - fin'*y;
71 if (~isempty(D))
72 res = res + 0.25*(x*(D.*x))^-2;
73 end
74
75 % priradenie
76 varargout{1} = res;
77
78 elseif typ == 2
79 % pocita sa gradient Lagrangeovej funkcie
80 [~,~,Jin,Je] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
81 res = G*x + h + Je*q*z - Jin'*y;
82 if (~isempty(D))
83 tmp = D.*x;
84 res = res + (x'*tmp)*tmp;
85 end
86
87 % priradenie
88 varargout{1} = res;
89
90 elseif typ == 3
91 % pocita sa Hessova matica Lagrangeovej funkcie
92 sel = (1:sz_n+1:sz_n^2)';
93 res = G;
94 if ~isempty(D)
95 tmp = D.*x;
96 res = res + 2*(tmp*tmp)';
97 res(sel) = res(sel) + x'*tmp*D;
98 end
99
100 if gdiag == 1
101 for i=1:sz_mq
102 res(sel) = res(sel) - y(i)*Gin(:, i);
103 end
104 else
105 for i=1:sz_mq
106 res = res - y(i)*Gin(:, sz_n*(i-1)+1:sz_n*i);
107 end
108
109 % priradenie
110 varargout{1} = res;
111
112 elseif typ == 4
113 % pocita sa gradient a Hessova matica Lagrangeovej funkcie,

```

```

115 % vo vstupe su zoradene ako [grad, hess]
116 [~,~,Jin,Je] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
117 sel = (1:sz_n+1:sz_n^2)';
118
119 res1 = G*x + h + Je*q*z - Jin'*y;
120 res2 = G;
121 if (~isempty(D))
122 tmp = D.*x;
123 res1 = res1 + (x'*tmp)*tmp;
124 res2 = res2 + 2*(tmp*tmp)';
125 res2(sel) = res2(sel) + x'*tmp*D;
126 end
127 if gdiag == 1
128 for i=1:sz_mq
129 res2(sel) = res2(sel) - y(i)*Gin(:, i);
130 end
131 else
132 for i=1:sz_mq
133 res2 = res2 - y(i)*Gin(:, sz_n*(i-1)+1:sz_n*i);
134 end
135
136 % priradenie
137 varargout{1} = res1;
138 varargout{2} = res2;
139
140 else
141 % pocita sa funkčna hodnota a gradient Lagrangeovej funkcie,
142 % vo vstupe su zoradene ako [fval, grad]
143 [fin, feq, Jin, Je] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
144 res1 = x*(0.5*G*x + h) + feq*z - fin'*y;
145 res2 = G*x + h + Je*q*z - Jin'*y;
146 if (~isempty(D))
147 tmp = D.*x;
148 res1 = res1 + 0.25*(x'*tmp)^2;
149 res2 = res2 + (x'*tmp)*tmp;
150 end
151
152 % priradenie
153 varargout{1} = res1;
154 varargout{2} = res2;
155 end

```

```

1 function varargout = LagrAug_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r,...
2 psi_E,psi_I,met)
3
4 % funkcia pocita funkčnu hodnotu, gradient alebo Hessovu maticu
5 % rozšírenej Lagrangeovej funkcie definovanej ako
6 % L_A(x,y,z) = f(x) + 1/r sum(psi_I(r*g(x),y,...)) + ...
7 % 1/r sum(psi_E(r*h(x),z,...))
8 % navratove hodnoty zavisia od hodnoty premennej met, ktora moze byt
9 % FVAL - pocita sa funkčna hodnota Lagrangeovej funkcie
10 % GRAD - pocita sa gradient Lagrangeovej funkcie
11 % HESS - pocita sa Hessova matica Lagrangeovej funkcie
12 % GRADHSS - pocita sa gradient a Hessova matica Lagrangeovej funkcie
13 % FVALGRAD - pocita sa funkčna hodnota a gradient Lagrangeovej funkcie
14 %
15 % PSI je string a urcuje nazov penalizacnej funkcie, ktora je
16 % priradená k rozšírenej Lagrangeovej funkcii, PSI_I zodpoveda pen.
17 % nerovnic a PSI_E patri pen. funkcii rovníc. Obe funkcie musia mat 3
18 % vstupne parametre.
19
20 if nargin < 17
21 error('Funkcia LAGRAUG_QUAD potrebuje 17 vstupnych argumentov');
22 end
23
24 % kontrola vstupnej premennej met

```

```

25 switch lover(met)
26 case 'fhd',
27     typ = 1;
28 case 'grad',
29     typ = 2;
30 case 'hess',
31     typ = 3;
32 case 'gradhess',
33     typ = 4;
34 case 'fhodgrad',
35     typ = 5;
36 otherwise
37     error('Met musi byt FHOD, GRAD, HESS, FHODGRAD, alebo GRADHESS');
38 end
39
40 % kontrola vstupnych parametrov
41 if typ < 4 && nargin > 1
42     error('Prilis vela vstupnych argumentov');
43 elseif typ >= 4 && nargin > 2
44     error('Prilis vela vstupnych argumentov');
45 end
46
47 % prislusne rozmery ulohy
48 sz_n = length(x);
49 sz_meq = length(z);
50 sz_mq = max(size(hin, 2), length(rin));
51
52 % kontroly rozmerov
53 if length(D) ~= sz_n && isempty(D)
54     error('Diagonala matice D ma nespravny rozmer');
55 elseif max(length(b), size(A, 1)) ~= length(y) - sz_mq
56     error('Vektor b alebo matica A maju nespravny rozmer');
57 elseif max(length(beq), size(Aeq, 1)) ~= sz_meq
58     error('Vektor beq alebo matica Aeq maju nespravny rozmer');
59 elseif max(length(G), length(b)) ~= sz_n
60     error('Vektor h alebo matica G maju nespravny rozmer');
61 end
62
63 % kontrola a identifikacia zadania matice Gin
64 sz_g = size(Gin, 2);
65 if sz_g == sz_mq
66     gdiag = 1;
67 elseif sz_g == sz_n * sz_mq
68     gdiag = 0;
69 else
70     error('Matice Gin ma nespravny rozmer');
71 end
72
73 % vykonanie pozadovanych vypoctov
74 if typ == 1
75     % pocita sa funkčna hodnota Lagrangeovej funkcie
76     [fin, feq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
77     res = x*(0.5*G*x + h) + (sum(feval(psi_I, r*fin,y,'fhod')) + ...
78         sum(feval(psi_E, r*feq,z,'fhod')))/r;
79     if (~isempty(D))
80         res = res + 0.25*(x'(D.*x))^-2;
81     end
82
83 % priradenie
84     varargout{1} = res;
85
86 elseif typ == 2
87     % pocita sa gradient Lagrangeovej funkcie
88     [fin, feq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
89     ybar = - feval(psi_I, r*fin, y, 'der1');
90     zbar = feval(psi_E, r*feq, z, 'der1');
91
92     res = LagrClass_quad(x,ybar,zbar,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'grad');
93
94 % priradenie
95     varargout{1} = res;
96
97 elseif typ == 3
98     % pocita sa Hessova matica Lagrangeovej funkcie
99     [fin, feq, jin, jeq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
100     ybar = - feval(psi_I, r*fin, y, 'der1');
101     zbar = feval(psi_E, r*feq, z, 'der1');
102
103     res = LagrClass_quad(x,ybar,zbar,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'hess') + ...
104         r * (jin'.*(ones(sz_n,1)*feval(psi_I,r*fin,y,'der2'))*jin + ...
105             jeq'.*(ones(sz_n,1)*feval(psi_E,r*feq,z,'der2'))*jeq);
106
107 % priradenie
108     varargout{1} = res;
109
110 elseif typ == 4
111     % pocita sa Hessova matica Lagrangeovej funkcie,
112     % vo vstupe su zoradene ako [grad, hess]
113     [fin, feq, jin, jeq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
114     ybar = - feval(psi_I, r*fin, y, 'der1');
115     zbar = feval(psi_E, r*feq, z, 'der1');
116
117     [res1, res2] = LagrClass_quad(x,ybar,zbar,D,G,h,Gin,hin,rin,A,b,...
118         Aeq,beq,'gradhess');
119     res2 = res2 + ...
120         r * (jin'.*(ones(sz_n,1)*feval(psi_I,r*fin,y,'der2'))*jin + ...
121             jeq'.*(ones(sz_n,1)*feval(psi_E,r*feq,z,'der2'))*jeq);
122
123 % priradenie
124     varargout{1} = res1;
125     varargout{2} = res2;
126 else
127     % pocita sa funkčna hodnota a gradient Lagrangeovej funkcie,
128     % vo vstupe su zoradene ako [fval, grad]
129     [fin, feq] = const_quad(x,Gin,hin,rin,A,b,Aeq,beq,gdiag);
130     res1 = x*(0.5*G*x + h) + 1/r*(sum(feval(psi_I, r*fin,y,'fhod')) + ...
131         sum(feval(psi_E, r*feq,z,'fhod')));
132     if (~isempty(D))
133         res1 = res1 + 0.25*(x'(D.*x))^-2;
134     end
135     ybar = - feval(psi_I, r*fin, y, 'der1');
136     zbar = feval(psi_E, r*feq, z, 'der1');
137     res2 = LagrClass_quad(x,ybar,zbar,D,G,h,Gin,hin,rin,A,b,Aeq,beq,'grad');
138
139 % priradenie
140     varargout{1} = res1;
141     varargout{2} = res2;
142 end

```

Transformačné funkcie (Rockafellarova, Powell-Hestenesova a hyperbolická ψ funkcia, označené `psi_I_PHR`, `psi_E`, `psi_I_hyp`). Pred použitím je nutné skompilovať.

```

1 /* Pocita funkcnu hodnotu, prvu a druhe derivacie Rockafellarovej
2 penalizacnej funkcie
3
4 PSI(xi, nu) = 0.5*[max(0, nu-xi)]^2 - nu^2 ]
5
6 LAG(x,y,...) = f(x) + 1/r*sum(psi(*g(x),y)) + ... */
7
8 #if !defined(MAX2)
9 #define MAX2(A, B) ((A) < (B) ? (B) : (A))
10 #endif
11
12 #include "mex.h"
13 #include <string.h>
14 #include <math.h>
15
16 /* vstupne argumenty musia byt v tomto poradí:
17 prhs[0] = vektor xi, double (m x 1), vektor hodnot ochraničeni
18 prhs[1] = vektor nu, double (m x 1), vektor multiplikátorov
19 prhs[2] = retazec 'typ', udava, co sa ma vykonat
20 vystupom je vektor
21 */
22
23 void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
24 {
25     /* inicializacie premennych*/
26     double *res, *xi, *nu;
27     char *type;
28     mwSize typelen;
29     mwIndex len, i, typ;
30
31     /*vstupne kontroly*/
32     if (nrhs != 3) {
33         mexErrMsgTxt("Funkcia potrebuje 3 vstupne parametre");
34     }
35     if (nlhs > 1) {
36         mexErrMsgTxt("Prilis vela vystupnych parametrov");
37     }
38     if ( ! mxIsChar(prhs[2]) || (mxGetM(prhs[2]) != 1) ) {
39         mexErrMsgTxt("Vstupny argument 'typ musi byt string");
40     }
41     if (mxGetM(prhs[0]) != 1 && mxGetN(prhs[0]) != 1) {
42         mexErrMsgTxt("Vstupna premenna xi musi byt vektor");
43     }
44     if (mxGetM(prhs[1]) != 1 && mxGetN(prhs[1]) != 1) {
45         mexErrMsgTxt("Vstupna premenna nu musi byt vektor");
46     }
47
48     /*kopirovanie retazca zo vstupnej premennej 'typ'*/
49     typelen = mxGetN(prhs[2])*sizeof(mxChar) + 1;
50     type = mxArrayMalloc(typelen);
51     if (mxGetString(prhs[2], type, typelen)) {
52         mexErrMsgTxt("Chyba pri konverzii premennej typ");
53     }
54
55     /*inicializacia smernikov*/
56     len = (mwIndex)MAX2(mxGetM(prhs[0]), mxGetN(prhs[0]));
57     if (len != (mwIndex)MAX2(mxGetM(prhs[1]), mxGetN(prhs[1]))) {
58         mexErrMsgTxt("Vektory xi a nu musia byt rovnakej dlzky");
59     }
60     if ( ! mxIsEmpty(prhs[0]) || ! mxIsEmpty(prhs[1]) ) {
61         len = 0;
62     }
63
64     /*kontrola typu operacie*/
65     if (strcmp(type, "fnod") == 0) {
66         typ = 1;
67     } else if (strcmp(type, "der1_xi") == 0 || strcmp(type, "der1") == 0) {

```

```

68         typ = 2;
69     } else if (strcmp(type, "der1_nu") == 0) {
70         typ = 3;
71     } else if (strcmp(type, "der2_xi") == 0 || strcmp(type, "der2") == 0) {
72         typ = 4;
73     } else if (strcmp(type, "der2_xinu") == 0) {
74         typ = 5;
75     } else if (strcmp(type, "der2_nunu") == 0) {
76         typ = 6;
77     } else {
78         mexErrMsgTxt("Chybne zadany typ operacie");
79     }
80
81     /*definovanie smernikov*/
82     plhs[0] = mxCreateDoubleMatrix(len, 1, mxREAL);
83     res = mxGetPr(plhs[0]);
84     xi = mxGetPr(prhs[0]);
85     nu = mxGetPr(prhs[1]);
86
87     if (typ == 1) {
88         /* pocita sa funkcná hodnota penalizacnej casti
89         Rockafellarovej funkcie */
90         for ( i=0; i<len; i++) {
91             if (xi[i] <= nu[i]) {
92                 res[i] = (0.5*xi[i] - nu[i])*xi[i];
93             } else {
94                 res[i] = -0.5*nu[i]*nu[i];
95             }
96         }
97     } else if (typ == 2) {
98         /* pocita sa prva derivacia penalizacnej casti
99         Rockafellarovej funkcie, podla xi*/
100        for ( i=0; i<len; i++) {
101            if (xi[i] <= nu[i]) {
102                res[i] = xi[i] - nu[i];
103            } else {
104                res[i] = 0;
105            }
106        }
107    } else if (typ == 3) {
108        /* pocita sa prva derivacia penalizacnej casti
109        Rockafellarovej funkcie, podla nu*/
110        for ( i=0; i<len; i++) {
111            if (xi[i] <= nu[i]) {
112                res[i] = -xi[i];
113            } else {
114                res[i] = -nu[i];
115            }
116        }
117    } else if (typ == 4) {
118        /* pocita sa druha derivacia penalizacnej casti
119        Rockafellarovej funkcie, podla xi,xi*/
120        for ( i=0; i<len; i++) {
121            if (xi[i] <= nu[i]) {
122                res[i] = 1;
123            } else {
124                res[i] = 0;
125            }
126        }
127    } else if (typ == 5) {
128        /* pocita sa druha derivacia penalizacnej casti
129        Rockafellarovej funkcie, podla xi,nu*/
130        for ( i=0; i<len; i++) {
131            if (xi[i] <= nu[i]) {
132                res[i] = -1;
133            } else {
134                res[i] = 0;

```

```

135     }
136 }
137 } else if (typ == 6) {
138     /* pocita sa druha derivacia penalizacnej casti
139     Rockafellarovej funkcie, podla nu,nu*/
140     for ( i=0; i<len; i++) {
141         if (xi[i] <= nu[i]) {
142             res[i] = 0;
143         } else {
144             res[i] = -1;
145         }
146     }
147 }
148 mxFree(type);
149 }
150 }

1 /* Pocita funkcnu hodnotu, prvu a druha derivacie Hestenesovej
2 penalizacnej funkcie. */
3
4 PSI(xi, nu) = xi*nu + 0.5*xi*xi
5 LAG(x,...,z) = f(x) + ... + 1/r*sum(PSI(r*h(x),z)) + ... */
6
7 #if defined(MAX2)
8 #define MAX2(A, B) ((A) < (B) ? (B) : (A))
9 #endif
10
11 #include "mex.h"
12 #include <string.h>
13 #include <math.h>
14
15 /* vstupne argumenty musia byt v tomto poradí:
16 prhs[0] = vektor xi, double (m x 1), vektor hodnot ohraniceni
17 prhs[1] = vektor nu, double (m x 1), vektor multiplikatorov
18 prhs[2] = retazec 'typ', udava, co sa ma vykonat
19 vystupom je vektor
20 */
21
22 void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
23 {
24     /* inicializacie premennych*/
25     double *res, *xi, *nu;
26     char *type;
27     mwSize typelen;
28     mwIndex len, i, typ;
29
30     /*vstupne kontroly*/
31     if (prhs != 3) {
32         mexErrMsgTxt("Funkcia potrebuje 3 vstupne parametre");
33     }
34     if (nlhs > 1) {
35         mexErrMsgTxt("Prilis vela vystupnych parametrov");
36     }
37     if ( ! mxIsChar(prhs[2]) || (mxGetM(prhs[2]) != 1) ) {
38         mexErrMsgTxt("Vstupny argument typ musi byt string");
39     }
40     if (mxGetM(prhs[0]) != 1 && mxGetM(prhs[0]) != 1) {
41         mexErrMsgTxt("Vstupna premenna xi musi byt vektor");
42     }
43     if (mxGetM(prhs[1]) != 1 && mxGetM(prhs[1]) != 1) {
44         mexErrMsgTxt("Vstupna premenna nu musi byt vektor");
45     }
46
47     /*kopirovanie retazca zo vstupnej premennej 'typ'*/
48     typelen = mxGetM(prhs[2])*sizeof(mxChar) + 1;
49     type = mxMalloc(typelen);
50     if (mxGetString(prhs[2], type, typelen)) {

```



```

16 prhs[0] = vektor xi, double (m x 1), vektor hodnot ohraničení
17 prhs[1] = vektor nu, double (m x 1), vektor multiplikátorov
18 prhs[2] = retazec 'typ', udava, co sa ma vykonat
19 vystupom je vektor
20 */
21
22 void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
23 {
24     /* inicializacie premennych*/
25     double *res, *xi, *nu;
26     char *type;
27     mSize typelen;
28     mIndex ien, i, typ;
29     /*vstupne kontroly*/
30     if (nrhs != 3) {
31         mexErrMsgTxt("Funkcia potrebuje 3 vstupne parametre");
32     }
33     if (nlhs > 1) {
34         mexErrMsgTxt("Prilis vela vystupnych parametrov");
35     }
36     if ( ! mxIsChar(prhs[2]) || mxGetM(prhs[2]) != 1 ) {
37         mexErrMsgTxt("Vstupny argument typ musi byt string");
38     }
39     if ( mxGetM(prhs[0]) != 1 && mxGetM(prhs[0]) != 1 ) {
40         mexErrMsgTxt("Vstupna premenna xi musi byt vektor");
41     }
42     if ( mxGetM(prhs[1]) != 1 && mxGetM(prhs[1]) != 1 ) {
43         mexErrMsgTxt("Vstupna premenna nu musi byt vektor");
44     }
45     }
46     /*kopirovanie retazca zo vstupnej premennej 'typ'*/
47     typelen = mxGetM(prhs[2])*sizeof(mxChar) + 1;
48     type = malloc(typelen);
49     if ( mxGetString(prhs[2], type, typelen) ) {
50         mexErrMsgTxt("Chyba pri konverzii premennej typ");
51     }
52     /*inicializacia smernikov*/
53     len = (mIndex)MAX2(mxGetM(prhs[0]),mxGetM(prhs[0]));
54     if ( len != (mIndex)MAX2(mxGetM(prhs[1]),mxGetM(prhs[1])) ) {
55         mexErrMsgTxt("Vektory xi a nu musia byt rovnakej dlzky");
56     }
57     if ( mxIsEmpty(prhs[0]) || mxIsEmpty(prhs[1]) ) {
58         len = 0;
59     }
60     /*kontrola typu operacie*/
61     if ( strcmp(type, "fhod") == 0 ) {
62
63
64

```

```

65     typ = 1;
66     } else if ( strcmp(type, "der1") == 0 ) {
67     typ = 2;
68     } else if ( strcmp(type, "der2") == 0 ) {
69     typ = 3;
70     } else {
71     mexErrMsgTxt("Chybne zadany typ operacie");
72     }
73     /*definovanie smernikov*/
74     plhs[0] = mxCreateDoubleMatrix(len, 1, mxREAL);
75     res = mxGetPr(plhs[0]);
76     xi = mxGetPr(prhs[0]);
77     nu = mxGetPr(prhs[1]);
78     if ( typ == 1 ) {
79     /* pocita sa funkčna hodnota penalizacnej casti
80     hyperbollickej funkcie */
81     for ( i=0; i<len; i++) {
82         if ( xi[i] <= 0.0) {
83             res[i] = nu[i] * ( xi[i] - 1 ) * xi[i];
84         } else {
85             res[i] = nu[i] * ( 1/(1 + xi[i]) - 1 );
86         }
87     }
88     }
89     } else if ( typ == 2 ) {
90     /* pocita sa prva derivacia penalizacnej casti
91     hyperbollickej funkcie, podľa xi*/
92     for ( i=0; i<len; i++) {
93         if ( xi[i] <= 0.0) {
94             res[i] = nu[i] * ( 2*xi[i] - 1 );
95         } else {
96             res[i] = nu[i] * ( -1/pow(1+xi[i], 2) );
97         }
98     }
99     }
100     } else if ( typ == 3 ) {
101     /* pocita sa druha derivacia penalizacnej casti
102     hyperbollickej funkcie, podľa xi,xi*/
103     for ( i=0; i<len; i++) {
104         if ( xi[i] <= 0.0) {
105             res[i] = nu[i] * ( 2 );
106         } else {
107             res[i] = nu[i] * ( 2/pow(1+xi[i], 3) );
108         }
109     }
110     mxFree(type);
111     }
112 }

```

```

9  /* Modifikovana Newtonova metoda pre riesenie minimalizacneho problemu.
10 Cholsolve je funkcia na hladanie spadoveho smeru s'k rieseniim sustavy
11 hess(L(x))*s = -grad(L(x)) kde sa pouzije modifikovany Choleskeho
12 rozklad Hesseovej matice podľa algoritmu GMW81 s diagonalnou
13 pivotaciou, zaroven sluzi na hladanie smeru d'k ako smeru 'negative
14 definitna', ktorý je nulovy, ak je matica hess(L(x)) dostatočne kladne
15 definitna, a nenulovy v opakom prípade. Pre tieto smery platia
16 nasledovne vzťahy:

```

```

9  s'*grad(L(x)) < 0,    d'*grad(L(x)) < 0,    d'*hess(L(x))*d < 0.
11
12 Vstupne argumenty:
13 prhs[0] : Hesseova matica sustavy
14 prhs[1] : Gradienty sustavy
15 prhs[2] : minimálny prvok, ktorý je v prípade nutnosti pripocitany
16 k diagonale Hesseovej matice

```

Subprocedúra Newtonovej metódy: funkcia `cholsolve` hľadá riešenie systému rovníc $(\nabla^2 F(\mathbf{x}) + \mathbf{E}) = -\nabla F(\mathbf{x})$ použitím modifikovaného Choleskeho rozkladu podľa algoritmu GMW81 [17], matice \mathbf{E} sa určí „za behu“. Nutné skompilovať.

```

17  Vstupne argumenty:
18  plhs[0] : spadovy smer 's'
19  plhs[1] : smer 'negative curvature' 'd'
20 */
21
22 /* definicie pre maximum z dvoch, resp. troch argumentov */
23 #if !defined(MAX2)
24 #define MAX2(A, B) ((A) < (B) ? (B) : (A))
25 #endif
26 #if !defined(MAX3)
27 #define MAX3(A, B, C) (((A) < (B)) ? ((B) < (C) ? (C) : ((A) < (C) ? (C) : (A)))
28 #endif
29
30 #if !defined(MIN32)
31 #define dnm2 dnm2 /* 2-norma */
32 #define ddot ddot /* skalarny sucin */
33 #define idamax idamax /* Index maximalneho prvku v abs. hodnote */
34 #endif
35
36 #include "mex.h"
37 #include "blas.h"
38 #include <math.h>
39 #include <float.h>
40
41 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {
42
43     mwIndex n, j, k, q, ind, *pperm, one;
44     double *C, *L, *A, *B, *s, *d, *y, *D, *E;
45     double norm_g, norm_E, betasq, delta, theta, temp, minE;
46     double max_diag, max_offdiag, temp_D;
47
48     /* pocet vstupnych a vystupnych argumentov */
49     if (nrhs < 2 || nrhs > 3) {
50         mexErrMsgTxt("Funkcia potrebuje 2-3 vstupne parametre");
51     }
52     if (nrhs > 2) {
53         mexErrMsgTxt("Prilis vela vystupnych argumentov");
54     }
55
56     /* kontrola vstupnych argumentov */
57     n = mxGetM(prhs[0]);
58     if (mxGetM(prhs[0]) != n) {
59         mexErrMsgTxt("vstupna matica musi byt stvorcova");
60     }
61     if (mxGetM(prhs[1]) != n || mxGetM(prhs[1]) != 1) {
62         mexErrMsgTxt("Vektor g musi byt typu n x 1");
63     }
64     if (nrhs == 3) { /* zadany je aj minimalny korektny parameter */
65         if (mxGetNumberOfElements(prhs[2]) != 1) {
66             mexErrMsgTxt("Parameter MU musi byt skalar");
67         } else {
68             minE = fabs(mxGetScalar(prhs[2]));
69         }
70     } else { /* parameter nie je zadany */
71         minE = 0.0;
72     }
73
74     /* inicializacia vstupnych parametrov a smernikov na vstup,
75     deklaracia dynamickyho poli */
76     plhs[0] = mxCreateDoubleMatrix(n, 1, mxREAL); /* smer 's' */
77     s = mxGetPr(plhs[0]);
78
79     L = (double *)mxMalloc(n*n, sizeof(double)); /* cholesekeho rozklad L */
80     C = (double *)mxMalloc(n*n, sizeof(double)); /* pomocna matica C */
81     D = (double *)mxMalloc(n, sizeof(double)); /* diagonala matice D */
82     E = (double *)mxMalloc(n, sizeof(double)); /* diagonala matice E */
83     y = (double *)mxMalloc(n, sizeof(double)); /* pomocny vektor y */
84
85     perm = (mwIndex *)mxMalloc(n, sizeof(mwIndex)); /* vektor permutacie */
86     A = mxGetPr(prhs[0]); /* zdrojova matica A, Hessian */
87     b = mxGetPr(prhs[1]); /* zdrojovy vektor b, gradient */
88     one = 1;
89
90     /* memcpy(C, mxGetPr(prhs[0]), n*n*sizeof(double));
91     /* inicializacia diagonaly */
92     for (i=0; i<n*n; i++) {
93         C[i] = A[i];
94     }
95
96     /* uvodna permutacia */
97     for (i=0; i<n; i++) {
98         perm[i] = i;
99     }
100
101     /* vypocet najvacsieho diagonalneho prvku |A(j,j)| */
102     k = n + 1;
103     ind = idamax(&n, C, &k) - 1; /* index najvacsieho prvku v abs. hodnote */
104     max_diagA = fabs(C[ind*n + ind]);
105
106     /* vypocet najvacsieho mimodiagonalneho prvku */
107     max_offdiagA = 0;
108     for (j=0; j<n-1; j++) {
109         for (i=j+1; i<n; i++) {
110             temp = fabs(C[j*n+i]);
111             if (temp > max_offdiagA) {
112                 max_offdiagA = temp;
113             }
114         }
115     }
116     temp = max_offdiagA;
117     max_offdiagA = n > 1 ? max_offdiagA/sqrt(n*n-1) : max_offdiagA;
118
119     /* vypocet beta-2 */
120     betasq = MAX3(max_offdiagA, max_diagA, DBL_EPSILON);
121
122     /* vypocet deita, ako MU, ak je na vstupe zadane, alebo ako
123     ||H|| * macheps, ak je MU nezadane */
124     temp = temp + max_diagA;
125     delta = MAX2(DBL_EPSILON * MAX2(temp, 1), minE);
126
127     /* vypocet normy gradientu */
128     norm_g = dnm2(&n, b, &one); /* 2-norma vektora b */
129
130     /* Choleskeho rozklad matice A na dolnotrojholnikovu maticu L */
131     for (j=0; j<n; j++) { /* prechadzanie po stlpcoch */
132         L[j*n+j] = 1; /* na diagonale jednotka */
133
134         /* najdeme najvacsiu diagonalnu hodnotu matice A */
135         q = j;
136         temp = fabs(C[j*n+j]);
137         for (k=j+1; k<n; k++) {
138             if (temp < fabs(C[k*n+k])) {
139                 temp = fabs(C[k*n+k]);
140                 q = k;
141             }
142         }
143
144         /* symetricka vymena riadkov a stlpcov matice C a A */
145         if (q != j) {
146             /* permutacia */
147             temp = perm[j];
148             perm[j] = perm[q];
149             perm[q] = temp;
150             /* vymena riadkov a stlpcov matice A a C */
151

```

```

151 for ( k=0; k<n; k++) {
152     temp = C[k*n+j];
153     C[k*n+j] = C[k*n+q];
154     C[k*n+q] = temp;
155 }
156 for ( k=0; k<n; k++) {
157     temp = C[j*n+k];
158     C[j*n+k] = C[q*n+k];
159     C[q*n+k] = temp;
160 }
161 }
162
163 /* vypočet dotacných premenných C_ij*/
164 for ( i=0; i<j; i++) {
165     L[i*n+j] = C[i*n+j]/D[i];
166 }
167
168 for ( i=j+1; i<n; i++) {
169     temp = C[j*n+i];
170     for ( k=0; k<j; k++) {
171         temp = temp - L[k*n+i]*C[k*n+i];
172     }
173     C[j*n+i] = temp;
174 }
175
176 /* maximum z hodnot |c(i,j)| = |l(i,j)*d(j)|, i = j, i>j */
177 ind = n - j - 1;
178 if ( ind != 0 ) {
179     ind = idamax(&ind, &C[j*n+i], &one);
180     theta = fabs(C[j*n+i+ind]);
181 } else {
182     theta = 0;
183 }
184
185 /* modifikovaná diagonála */
186 temp_D = MAX3(theta*theta/betasq, fabs(C[j*n+j]), delta);
187 D[j] = temp_D;
188 E[j] = temp_D - C[j*n+j];
189
190 /* uprava diagonalnych prvkov C*/
191 for ( i=j+1; i<n; i++) {
192     temp = C[j*n+i];
193     C[i*n+i] = C[i*n+i] - temp*temp/temp_D;
194 }
195 } /*MAIN LOOP*/
196
197 /* inf-norma matice E */
198 ind = idamax(&n, E, &one);
199 norm_E = E[ind-1];
200
201 /* ***** */
202 /* spatne hladanie smeru pomocou rieseni trojuhelnikovych systemov */
203 /* ***** */
204
205 /* hladanie spadoveho smeru s */
206 for ( i=0; i<n; i++) {
207     temp = b[perm[i]];
208     for (k=0; k<i; k++) {
209         temp = temp + L[k*n+i]*y[k];
210     }
211     y[i] = -temp;
212 }
213
214 /* predelenie diagonalou */
215 for ( i=0; i<n; i++) {

```

```

216     y[i] = y[i] / D[i];
217 }
218
219 /* spatna substitucia */
220 for ( i=n-1; i>=0; i-- ) {
221     temp = y[i];
222     for ( k=i+1; k<n; k++) {
223         temp = temp - L[i*n+k]*s[perm[k]];
224     }
225     s[perm[i]] = temp;
226 }
227
228 /* hladanie 'direction of negative curvature' */
229 if ( nlbs > 1 ) {
230     /* pozadjems aj tento vystup */
231     plus[L] = mcreateDoubleMatrix(n, 1, mxREAL); /* smer 'd' */
232     d = mxgetPr(plus[L]);
233
234     if ( norm_E != 0 ) {
235         ind = 0;
236         temp = D[0] - E[0];
237         for ( i=1; i<n; i++) {
238             if ( temp > D[i] - E[i] ) {
239                 temp = D[i] - E[i];
240                 ind = i;
241             }
242         }
243     }
244     /* riesime sustavu L'*I*s = e_j */
245     for ( i=n-1; i>ind; i--) {
246         d[perm[i]] = 0;
247     }
248     d[perm[ind]] = 1;
249     for ( i=ind-1; i>=0; i--) {
250         temp = 0;
251         for ( k=i+1; k<=ind; k++) {
252             temp = temp + L[i*n+k]*d[perm[k]];
253         }
254         d[perm[i]] = -temp;
255     }
256
257     /* formulacia vystupneho smeru */
258     if ( norm_g != 0 ) {
259         temp = ddot(&n, d, &one, b, &one);
260         if ( temp > 0 ) { /* d'*g ma kladne znamienko */
261             for ( i=0; i<n; i++) {
262                 d[i] = -d[i];
263             }
264         }
265     }
266     } else {
267         /* d = 0 */
268         for ( j=0; j<n; j++) {
269             d[j] = 0;
270         }
271     }
272 }
273
274 /* uvolnenie pamate */
275 mxFree(L);
276 mxFree(C);
277 mxFree(D);
278 mxFree(E);
279 mxFree(y);
280 } /* MEX FUNCTION */

```

Vyhľadávanie na lúči: Brentov algoritmus na hľadanie koreňa funkcie $s^T \nabla F(\mathbf{x} + \lambda \mathbf{s})$, použitá implementácia je prepisom algoritmu *zero* z [6], s. 58-59, zdrojom bola takisto publikácia [37], a algoritmus More a Thuente [25], ktorý je založený na subfunkcii MCSRCH fortranovskej funkcie LBFGS (obsiahnutá napr. v distribúcii MINPACK) od J. Nocedala, a na prepise tejto funkcie od D. O'Leary.

```

1 function alpha_t = brentzero(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r,s, ...
2     PSI_E, PSI_I)
3
4 % hľadanie optimálnej dlžky kroku Lambda, Brentova metóda na hľadanie
5 % koreňa gradientu funkcie Lagrange(x*Lam*s,y).
6 %
7 % Ref: Numerical Recipes in C: The Art of Scientific Computing, 3rd
8 % edition, strany 464-466.
9
10 epsilon = 1e-4;
11 EPSTOTAL = eps;
12 MaxIters = 20;
13
14 % najprv nasleduje pokus u zaskatulkovanie minima na intervale
15 % I = [alpha_l, alpha_t]
16 alpha_min = 0.001;
17 alpha_max = 8;
18
19 alpha_l = 0;
20 alpha_t = 1;
21 gt = s'*Lagrange_quad(x + s,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r, ...
22     PSI_E, PSI_I, 'grad');
23 while gt <= 0
24     alpha_l = alpha_t;
25     gl = gt;
26     alpha_t = 2.5*alpha_t;
27     gt = s'*Lagrange_quad(x + alpha_t*s,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r, ...
28         PSI_E, PSI_I, 'grad');
29 end
30
31 if alpha_l == 0
32     gl = s'*Lagrange_quad(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r, ...
33         PSI_E, PSI_I, 'grad');
34 end
35
36 gu = gt;
37 alpha_u = alpha_t;
38 e = alpha_t - alpha_l;
39 d = e;
40
41 for k=1:MaxIters
42     % USPORIADANIE: alpha_t je najlepší odhad lambda, koren je uzavretý na
43     % intervale medzi alpha_t a alpha_u, alpha_l je predosla hodnota
44     % alpha_t. Vynikmu tvorí prvá iterácia, kedy je alpha_u == alpha_t a
45     % alpha_l je opacny koniec intervalu.
46
47     if (gt>0 && gu>0) || (gt<=0 && gu<=0)
48         alpha_u = alpha_l;
49         gu = gl;
50         e = alpha_t - alpha_l;
51         d = e;
52     end
53
54     % zoradenie bodov
55     if abs(gu) < abs(gt)

```

```

56     alpha_l = alpha_t;
57     alpha_t = alpha_u;
58     alpha_u = alpha_l;
59     gl = gt;
60     gt = gu;
61     gu = gl;
62 end
63
64 % nastavenie konečnej tolerancie ako sučet reálnitvnej a absolútnej
65 % tolerancie
66 tol = 2*EPSTOTAL*abs(alpha_t) + 0.5*epsilon;
67 m = 0.5*(alpha_u - alpha_t);
68
69 % kontrola vstupu
70 if ( abs(m) <= tol || gt == 0 )
71     break;
72 end
73
74 % Analýza možných prípadov, v ktorých priradíme novú hodnotu kroku
75 % na základe bisekcie, lineárnej interpolácie alebo inverznej
76 % kvadratickej interpolácie z trojice dát [gl, gt, gu].
77 if ( abs(e) < tol || abs(gl) <= abs(gt) )
78     % Vykona sa bisekcia
79     e = m;
80     d = e;
81 else
82     S = gt/gl;
83     if alpha_l == alpha_u
84         % Vykona sa lineárna interpolácia
85         P = 2*m*S;
86         q = 1 - S;
87     else
88         % Vykona sa inverzná kvadratická interpolácia
89         q = gl/gu;
90         R = gt/gu;
91         P = S*(2*m*q*(q-R) - (alpha_t-alpha_l)*(R-1));
92         q = (S-1)*(q-1)*(R-1);
93     end
94     if P > 0
95         q = -q;
96     else
97         P = -P;
98     end
99     S = e;
100    e = d;
101
102    % podmienky na overenie vhodnosti interpolovaneho bodu
103    if (2*P<3*m*q - abs(tol*q) && P < abs(0.5*S*q))
104        d = P/q;
105    else
106        e = m;
107        d = e;
108    end
109 end
110

```

```

111 % alpha_1 je predchadzajuca hodnota alpha_t
112 alpha_1 = alpha_t;
113 gl = gt;
114
115 % nova hodnota alpha_t a nova gunkcna hodnota
116 if tol < abs(d)
117     alpha_t = alpha_t + d;
118 elseif m > 0
119     alpha_t = alpha_t + tol;
120 else
121     alpha_t = alpha_t - tol;
122 end
123 gt = s*Lagrange_quad(x + alpha_t*s,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r,...
124     PSI_E, PSI_I, 'grad');
125 end
126
127 % zabezpecenie, aby vystup bol v pozadovanych medziach
128 alpha_t = max(min(alpha_t, alpha_max), alpha_min);
129
130 function [alpha_t, grt] = morethue(x,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,...
131     beq,t,s,PSI_E, PSI_I)
132
133 % Funkcia MORETHUENTE je implementaciou algoritmu od J. MORE a D. THUENTE
134 % na hladanie vhodneho kroku alpha_t pri vyhladavani na luci funkcie
135 % Lagrange_quad(x+alpha*s,y,z,D,G,h,Gin,hin,rin,A,b,Aeq,beq,r,PSI_E,PSI_I).
136 % Algoritmus je navrhnuty na hladanie kroku alpha_t spinajucu Armijsu
137 % podmienku dostatočnemu poklesu
138 % Lagr(x+alpha_t*s) - Lagr(x) <= mu*alpha_t*(grad_Lagr(x)'s) (P1)
139 % a silnu Goldsteimovu podmienku zakrivenia
140 % abs(grad_Lagr(x+alpha_t*s)'s) <= eta*abs(grad_Lagr(x)'s), (P2)
141 % kde prislusne parametre volime tak, aby 0 < mu < eta < 1 a mu < 0.5
142 % (v nasom pripade mu = 1e-3 a eta = 0.9).
143
144 % Algoritmus v kazdom kroku upravuje interval neurcitosti I, v ktorom lezi
145 % odhad kroku alpha_t. Spociatku na jeho urcenie pouziva pomocnu funkciu
146 % Lagr(x+alpha*s) - Lagr(x) - mu*alpha*(grad_Lagr(x)'s), (1)
147 % po najdeni kroku, v ktorom ma funkcia (1) zapornu funkcnu hodnotu a
148 % kladnu derivaciu, sa na hladanie intervalu I pouzije funkcia
149 % Lagr(x+alpha*s). Pre detaily vid [1].
150
151 % OUTPUT FUNKCIE:
152 % alpha_t > najlepší odhad kroku alpha
153 % grt > gradient Lagrange funkcie v novom bode x + alpha_t*s
154
155 % Implementacia vychadza z FORTRANovskej implementacie tohto algoritmu
156 % v baliku MINPACK a z Jej MATLABovskeno prepisu od Dianne O'Leary.
157
158 % Zdroj:
159 % [1] MORE J. J., THUENTE D. J. - Line Search Algorithms with Guaranteed
160 % Sufficient Decrease, Argonne National Laboratory, 1983.
161
162 %
163 %
164 %
165 %
166 %
167 %
168 %
169 %
170 %
171 %
172 %
173 %
174 %
175 %
176 %
177 %
178 %
179 %
180 %
181 %
182 %
183 %
184 %
185 %
186 %
187 %
188 %
189 %
190 %
191 %
192 %
193 %
194 %
195 %
196 %
197 %
198 %
199 %
200 %
201 %
202 %
203 %
204 %
205 %
206 %
207 %
208 %
209 %
210 %
211 %
212 %
213 %
214 %
215 %
216 %
217 %
218 %
219 %
220 %
221 %
222 %
223 %
224 %
225 %
226 %
227 %
228 %
229 %
230 %
231 %
232 %
233 %
234 %
235 %
236 %
237 %
238 %
239 %
240 %
241 %
242 %
243 %
244 %
245 %
246 %
247 %
248 %
249 %
250 %
251 %
252 %
253 %
254 %
255 %
256 %
257 %
258 %
259 %
260 %
261 %
262 %
263 %
264 %
265 %
266 %
267 %
268 %
269 %
270 %
271 %
272 %
273 %
274 %
275 %
276 %
277 %
278 %
279 %
280 %
281 %
282 %
283 %
284 %
285 %
286 %
287 %
288 %
289 %
290 %
291 %
292 %
293 %
294 %
295 %
296 %
297 %
298 %
299 %
300 %
301 %
302 %
303 %
304 %
305 %
306 %
307 %
308 %
309 %
310 %
311 %
312 %
313 %
314 %
315 %
316 %
317 %
318 %
319 %
320 %
321 %
322 %
323 %
324 %
325 %
326 %
327 %
328 %
329 %
330 %
331 %
332 %
333 %
334 %
335 %
336 %
337 %
338 %
339 %
340 %
341 %
342 %
343 %
344 %
345 %
346 %
347 %
348 %
349 %
350 %
351 %
352 %
353 %
354 %
355 %
356 %
357 %
358 %
359 %
360 %
361 %
362 %
363 %
364 %
365 %
366 %
367 %
368 %
369 %
370 %
371 %
372 %
373 %
374 %
375 %
376 %
377 %
378 %
379 %
380 %
381 %
382 %
383 %
384 %
385 %
386 %
387 %
388 %
389 %
390 %
391 %
392 %
393 %
394 %
395 %
396 %
397 %
398 %
399 %
400 %
401 %
402 %
403 %
404 %
405 %
406 %
407 %
408 %
409 %
410 %
411 %
412 %
413 %
414 %
415 %
416 %
417 %
418 %
419 %
420 %
421 %
422 %
423 %
424 %
425 %
426 %
427 %
428 %
429 %
430 %
431 %
432 %
433 %
434 %
435 %
436 %
437 %
438 %
439 %
440 %
441 %
442 %
443 %
444 %
445 %
446 %
447 %
448 %
449 %
450 %
451 %
452 %
453 %
454 %
455 %
456 %
457 %
458 %
459 %
460 %
461 %
462 %
463 %
464 %
465 %
466 %
467 %
468 %
469 %
470 %
471 %
472 %
473 %
474 %
475 %
476 %
477 %
478 %
479 %
480 %
481 %
482 %
483 %
484 %
485 %
486 %
487 %
488 %
489 %
490 %
491 %
492 %
493 %
494 %
495 %
496 %
497 %
498 %
499 %
500 %
501 %
502 %
503 %
504 %
505 %
506 %
507 %
508 %
509 %
510 %
511 %
512 %
513 %
514 %
515 %
516 %
517 %
518 %
519 %
520 %
521 %
522 %
523 %
524 %
525 %
526 %
527 %
528 %
529 %
530 %
531 %
532 %
533 %
534 %
535 %
536 %
537 %
538 %
539 %
540 %
541 %
542 %
543 %
544 %
545 %
546 %
547 %
548 %
549 %
550 %
551 %
552 %
553 %
554 %
555 %
556 %
557 %
558 %
559 %
560 %
561 %
562 %
563 %
564 %
565 %
566 %
567 %
568 %
569 %
570 %
571 %
572 %
573 %
574 %
575 %
576 %
577 %
578 %
579 %
580 %
581 %
582 %
583 %
584 %
585 %
586 %
587 %
588 %
589 %
590 %
591 %
592 %
593 %
594 %
595 %
596 %
597 %
598 %
599 %
600 %
601 %
602 %
603 %
604 %
605 %
606 %
607 %
608 %
609 %
610 %
611 %
612 %
613 %
614 %
615 %
616 %
617 %
618 %
619 %
620 %
621 %
622 %
623 %
624 %
625 %
626 %
627 %
628 %
629 %
630 %
631 %
632 %
633 %
634 %
635 %
636 %
637 %
638 %
639 %
640 %
641 %
642 %
643 %
644 %
645 %
646 %
647 %
648 %
649 %
650 %
651 %
652 %
653 %
654 %
655 %
656 %
657 %
658 %
659 %
660 %
661 %
662 %
663 %
664 %
665 %
666 %
667 %
668 %
669 %
670 %
671 %
672 %
673 %
674 %
675 %
676 %
677 %
678 %
679 %
680 %
681 %
682 %
683 %
684 %
685 %
686 %
687 %
688 %
689 %
690 %
691 %
692 %
693 %
694 %
695 %
696 %
697 %
698 %
699 %
700 %
701 %
702 %
703 %
704 %
705 %
706 %
707 %
708 %
709 %
710 %
711 %
712 %
713 %
714 %
715 %
716 %
717 %
718 %
719 %
720 %
721 %
722 %
723 %
724 %
725 %
726 %
727 %
728 %
729 %
730 %
731 %
732 %
733 %
734 %
735 %
736 %
737 %
738 %
739 %
740 %
741 %
742 %
743 %
744 %
745 %
746 %
747 %
748 %
749 %
750 %
751 %
752 %
753 %
754 %
755 %
756 %
757 %
758 %
759 %
760 %
761 %
762 %
763 %
764 %
765 %
766 %
767 %
768 %
769 %
770 %
771 %
772 %
773 %
774 %
775 %
776 %
777 %
778 %
779 %
780 %
781 %
782 %
783 %
784 %
785 %
786 %
787 %
788 %
789 %
790 %
791 %
792 %
793 %
794 %
795 %
796 %
797 %
798 %
799 %
800 %
801 %
802 %
803 %
804 %
805 %
806 %
807 %
808 %
809 %
810 %
811 %
812 %
813 %
814 %
815 %
816 %
817 %
818 %
819 %
820 %
821 %
822 %
823 %
824 %
825 %
826 %
827 %
828 %
829 %
830 %
831 %
832 %
833 %
834 %
835 %
836 %
837 %
838 %
839 %
840 %
841 %
842 %
843 %
844 %
845 %
846 %
847 %
848 %
849 %
850 %
851 %
852 %
853 %
854 %
855 %
856 %
857 %
858 %
859 %
860 %
861 %
862 %
863 %
864 %
865 %
866 %
867 %
868 %
869 %
870 %
871 %
872 %
873 %
874 %
875 %
876 %
877 %
878 %
879 %
880 %
881 %
882 %
883 %
884 %
885 %
886 %
887 %
888 %
889 %
890 %
891 %
892 %
893 %
894 %
895 %
896 %
897 %
898 %
899 %
900 %
901 %
902 %
903 %
904 %
905 %
906 %
907 %
908 %
909 %
910 %
911 %
912 %
913 %
914 %
915 %
916 %
917 %
918 %
919 %
920 %
921 %
922 %
923 %
924 %
925 %
926 %
927 %
928 %
929 %
930 %
931 %
932 %
933 %
934 %
935 %
936 %
937 %
938 %
939 %
940 %
941 %
942 %
943 %
944 %
945 %
946 %
947 %
948 %
949 %
950 %
951 %
952 %
953 %
954 %
955 %
956 %
957 %
958 %
959 %
960 %
961 %
962 %
963 %
964 %
965 %
966 %
967 %
968 %
969 %
970 %
971 %
972 %
973 %
974 %
975 %
976 %
977 %
978 %
979 %
980 %
981 %
982 %
983 %
984 %
985 %
986 %
987 %
988 %
989 %
990 %
991 %
992 %
993 %
994 %
995 %
996 %
997 %
998 %
999 %
1000 %

```

```

115 % sirka je menej ako xtol
116 if ( bracketed && a_max-a_min <= xtol*a_max )
117     info = 2;
118 end
119 % pozadovany krok alpha_t najdeny, (P1) aj (P2) splnene
120 if ( ft <= fcrit && abs(gt) <= eta*(-dginitt) )
121     info = 1;
122 end
123
124 % ukoncenie hlavnej iteracie
125 if (info == 0)
126     return
127 end
128
129 % urcenie fazy algoritmu, ktora urcuje pouzitie pomocnej funkcie
130 % na odhadnutie noveho kroku
131 if (phase1 && ft <= fcrit && gt >= min(mu,eta)*dginitt)
132     phase1 = 0;
133 end
134
135 % zistime, ci k urcenu noveho kroku pouzijeme pomocnu funkciu alebo
136 % nie (aux urcuje pouzitie pomocnej funkcie)
137 if (phase1 && ft <= fl && ft > fcrit)
138     aux = 1;
139     ft = ft - alpha_t*gcrit;
140     fl = fl - alpha_l*gcrit;
141     fu = fu - alpha_u*gcrit;
142     gt = gt - gcrit;
143     gl = gl - gcrit;
144     gu = gu - gcrit;
145 else
146     aux = 0;
147 end
148
149 % Nasleduju vypočet odhadu noveho kroku na zaklade sucasnych udajov,
150 % dispozicii je aktualny odhad kroku alpha_t, a krajne body intervalu
151 % [alpha_a, alpha_t], v kazdom z nich mame udaj o funkcnjej hodnote a
152 % derivacii. Body alpha_l a alpha_u su zoradene tak, ze alpha_l je bod
153 % s nizsou funkcnou hodnotou (moze sa teda stat, ze alpha_l > alpha_u).
154 % Ak je minimum ohranicene bodmi alpha_l a alpha_u (bracketed = TRUE),
155 % potom alpha_t lezi v tomto intervale. V kazdom pripade musi platiť
156 %  $gl*(alpha_t - alpha_l) < 0$ , teda funkcia musi klesat v smere kroku alpha_l.
157
158 % A: funkčna hodnota ft je vacsia ako hodnota fl. K odhadu noveho kroku
159 % sa pouzije kvadraticka [gl, ft, fl] a kubicka interpolacia [fl, ft, gl, gt].
160 % Novy krok lezi v intervale I (bracketed = TRUE).
161 if ft > fl
162     bound = 1;
163     dalpha = alpha_t - alpha_l;
164     Z = gl + gt - 3*(ft-fl)/dalpha;
165     W = sqrt(Z^2 - gt*gl);
166     if dalpha > 0
167         temp = (W + Z - gl)/(2*W + gt - gl);
168     else
169         temp = (W + gl - Z)/(2*W + gl - gt);
170     end
171
172 % odhady pomocou kubickej a kvadratickej interpolacie
173 alpha_cub = alpha_l + dalpha*temp;
174 alpha_quad = alpha_l + 0.5*dalpha*gl/(gl - (ft-fl)/dalpha );
175
176 % stanovenie noveho odhadu kroku
177 if abs(alpha_cub - alpha_l) < abs(alpha_quad - alpha_l)
178     alpha_new = alpha_cub;
179 else
180     alpha_new = 0.5*(alpha_cub + alpha_quad);
181 end
182
183 % alpha_new lezi v intervale I
184 bracketed = 1;
185
186 % B: plati ft < fl, a zaroven gradienty v bodoch alpha_l a alpha_t
187 % maju opacne znamienka. Opat sa pouzije kvadraticka [gl, gt] a
188 % kubicka [fl, ft, gl, gt] interpolacia. Novy odhad lezi v intervale I
189 % (bracketed = TRUE).
190 elseif gl*gt < 0
191     bound = 0;
192     dalpha = alpha_t - alpha_l;
193     Z = gl + gt - 3*(ft-fl)/dalpha;
194     W = sqrt(Z^2 - gt*gl);
195     if dalpha > 0
196         temp = (W + Z - gl)/(2*W + gt - gl);
197     else
198         temp = (W + gl - Z)/(2*W + gl - gt);
199     end
200
201 % odhady pomocou kubickej a kvadratickej interpolacie
202 alpha_cub = alpha_l + dalpha*temp;
203 alpha_quad = alpha_l - dalpha*gl/(gt - gl);
204
205 % stanovenie noveho odhadu kroku
206 if abs(alpha_cub - alpha_t) >= abs(alpha_quad - alpha_t)
207     alpha_new = alpha_cub;
208 else
209     alpha_new = alpha_quad;
210 end
211
212 % alpha_new lezi v intervale I
213 bracketed = 1;
214
215 % C: plati ft < fl, gt*gl >= 0 a |gt| < |gl|. Gradienty su rovnakeho
216 % znamienka a ich absolutna hodnota sa zmenjuje. Kubicky krok sa vezme
217 % do uvahy v pripade, ak kubicka interpolacna funkcia ide do nekoneca
218 % v smere kroku, alebo ak je kubicky krok za krokom alpha_t. Pocita
219 % sa aj kvadraticky krok a v zavislosti od jeho polohy vzhľadom k
220 % alpha_l a od toho, ci je minimum obsadnute v intervale I, sa
221 % definuje novy odhad.
222 elseif abs(gt) < abs(gl)
223     bound = 1;
224     dalpha = alpha_t - alpha_l;
225     Z = gl + gt - 3*(ft-fl)/dalpha;
226     W = sqrt(max(0, Z^2 - gt*gl));
227     if dalpha > 0
228         temp = (W + Z - gl)/(2*W + gt - gl);
229     else
230         temp = (W + gl - Z)/(2*W + gl - gt);
231     end
232
233 % ak kubicka interpolacna funkcia ide do nekoneca v smere kroku,
234 % alebo kubicky krok lezi za alpha_t (v nasom pripade W > 0 a
235 % temp > 1, pripad W == 0 nastane, ak kubicka funkcia nejde do
236 % nekoneca v smere kroku ), prijmete kubicky krok, inak
237 % ho nastavime ako jeden z okrajov intervalu I.
238 if (temp > 1 && W == 0)
239     alpha_cub = alpha_l + dalpha*temp;
240 elseif dalpha > 0
241     alpha_cub = a_max;
242 else
243     alpha_cub = a_min;
244 end
245 alpha_quad = alpha_l - dalpha*gl/(gt - gl);
246
247 % stanovenie noveho odhadu kroku
248 if bracketed

```

```

249 if (abs(alpha_t-alpha_cub) < abs(alpha_t-alpha_quad))
250   alpha_new = alpha_cub;
251 else
252   alpha_new = alpha_quad;
253 end
254 else
255   if (abs(alpha_t-alpha_cub) > abs(alpha_t-alpha_quad))
256     alpha_new = alpha_cub;
257   else
258     alpha_new = alpha_quad;
259   end
260 end
261 % D; ft < fl, gt*gl >= 0, a |gt| >= |gl|, na urcenie minima pouzijeme
262 % kubicku interpolaciu bodov [ft, fu, gt, gu].
263 else
264   % ak alpha_t lezi v intervale I, nový krok urcime kubickou interpolaciou,
265   % inak ako nový krok zvolime jeden z krajnych bodov I
266   bound = 0;
267   if bracketed
268     dalpha = alpha_u - alpha_t;
269     Z = gt + gu - 3*(fu-ft)/dalpha;
270     W = sqrt(Z^2 - gu*gt);
271     if dalpha > 0
272       temp = (W + Z - gt)/(2*W + gu - gt);
273     else
274       temp = (W + gt - Z)/(2*W + gt - gu);
275     end
276     alpha_new = alpha_t + dalpha*temp;
277   elseif alpha_t > alpha_l
278     alpha_new = a_max;
279   else
280     alpha_new = a_min;
281   end
282 end
283 end
284
285 % uprava intervalu I a preznacenie bodov intervalu
286 if fl < ft
287   alpha_u = alpha_t;
288   gu = gt;
289   fu = ft;

```

```

290 else
291   if gt*gl < 0
292     alpha_u = alpha_l;
293     fu = fl;
294     gu = gl;
295   end
296   alpha_l = alpha_t;
297   fl = ft;
298   gl = gt;
299 end
300 % vypocet noveho kroku alpha_t
301 alpha_t = max(a_min, min(a_max, alpha_new));
302 if bracketed && bound
303   if (alpha_u > alpha_l)
304     alpha_t = min(alpha_l + 0.66*(alpha_u-alpha_l), alpha_t);
305   else
306     alpha_t = max(alpha_l + 0.66*(alpha_u-alpha_l), alpha_t);
307   end
308 end
309
310 % v pripade, ze sme pocitali nový krok cez pomocnu funkciu (aux == 1),
311 % vratime sa k vychodzim hodnotam
312 if aux
313   fl = fl + alpha_l*gcrit;
314   fu = fu + alpha_u*gcrit;
315   gl = gl + gcrit;
316   gu = gu + gcrit;
317 end
318
319 % v pripade, ze nedoslo k dostatočnému zuzeniu intervalu I, pristupime k
320 % bisekcii (iba ak je minimum obsahnute v I)
321 if bracketed
322   if (abs(alpha_u-alpha_l) >= 0.66 * widthl)
323     alpha_t = alpha_l + 0.5*(alpha_u - alpha_l);
324   end
325   widthl = width;
326   width = abs(alpha_u-alpha_l);
327 end
328 end
329 end

```

```

1 /* Pocita porušenie prípustnosti ohraničeni fin >= 0, feq == 0 . */
2 /* Pouzite funkcie z kniznice BLAS : DNRM2, IDAMAX */
3 /* Kompilovat pomocou mex -O -DDEFINEUNIX const_viol.c -lmblas*/
4
5 #if !defined(MIN32)
6 #define dnr2 dnr2
7 #define idamax idamax_
8 #define dasum dasum_
9 #endif
10
11 #include "mex.h"
12 #include "blas.h"
13 #include <string.h>
14 #include <math.h>
15
16 #if !defined(MAX2)
17 #define MAX2(A, B) ((A) < (B) ? (B) : (A))
18 #endif
19 #if !defined(MIN2)
20 #define MIN2(A, B) ((A) > (B) ? (B) : (A))
21 #endif
22
23 /* vstupne argumenty musia byt v tomto poradí:
24 prhs[1] = vektor (g_i(x)), (m x 1) double - rezidua nerovnosti
25 prhs[2] = vektor (h_j(x)), (meq x 1) double - rezidua rovnosti
26 prhs[3] = typ normy na urcenie veľkosti porus. - string
27 */
28
29 void mexFunction(int nlhs, mxArray *prhs[], int nrhs, const mxArray *prhs[])
30 {
31   /* inicializacie premenných*/
32   double *fin, *req; /*smerniky na vstupy*/

```

Funkcia `const_viol` počítajúca veľkosť porušenia prípustnosti vektora x , funkcia `merit_comp` počíta porušenie prípustnosti a podmienky komplementarity a funkcia `merit` počíta hodnotu merit funkcie (4.9). Nutné skompilovať.

```

33 double *res, *ppntr;
34 char *type;
35 mwIndex len, i, j, typ, spacing = 1;
36 mwSize m, meq; /*rozmary vstupnych vektorov*/
37 mwSize typelen;
38
39 /* kontrola vstupov a vystupov */
40 if (nrhs != 3) {
41     mexErrMsgTxt("Funkcia CONST_VIOL potrebuje 3 vstupne argumenty");
42 }
43 if (nlhs > 1) {
44     mexErrMsgTxt("Prilis vela vystupnych argumentov");
45 }
46
47 /* rozmary vektorov */
48 m = (mwSize)mxGetNumberOfElements(prhs[0]);
49 meq = (mwSize)mxGetNumberOfElements(prhs[1]);
50
51 /*kopirovanie retazca zo vstupnej premennej 'typ'*/
52 typeLen = mxGetM(prhs[2])*sizeof(mxChar) + 1;
53 type = mxMalloc(typeLen);
54 if (mxGetString(prhs[2], type, typeLen)) {
55     mexErrMsgTxt("Chyba pri konverzii premennej typ");
56 }
57
58 /*kontrola typu operacie*/
59 if ((strcmp(type, "euc") == 0) || (strcmp(type, "2") == 0)) {
60     typ = 1;
61 } else if ((strcmp(type, "sup") == 0) || (strcmp(type, "max") == 0)) {
62     typ = 2;
63 } else {
64     mexErrMsgTxt("Neznamy typ normy");
65 }
66
67 /* alokacia pomocneho pola */
68 res = (double *)mxMalloc(m*meq, sizeof(double));
69 plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
70 ppntr = mxGetPr(plhs[0]);
71 fin = mxGetPr(prhs[0]);
72 feq = mxGetPr(prhs[1]);
73
74 /* vytvorenie komplementarného vektora */
75 j = 0;
76 for (i=0; i<m; i++) {
77     if ( fin[i] < 0.0 ) {
78         res[j] = fin[i]; /* neprípustne hodnoty nerovnosti */
79         j = j + 1;
80     }
81 }
82 for (i=0; i<meq; i++) {
83     res[j+i] = feq[i]; /* neprípustne hodnoty rovnosti */
84 }
85 j = j + meq;
86
87 /* pocitanie pozadovanej normy */
88 if (typ == 1) { /* euklidovska norma */
89     ppntr[0] = dnm2(&j, res, &spacing);
90 } else if (typ == 2) { /* maxinova norma */
91     i = idamax(&j, res, &spacing);
92     ppntr[0] = fabs(res[i-1]);
93 }
94
95 /* uvolnenie premennej res */
96 mxFree(res);
97 }
98
99 1 /* Pocita porusenie 2 - 5 podmienky KKT sustavy

```



```

69 /*kopirovanie retazca zo vstupnej premennej 'typ'*/
70 typelen = mxGetM(prhs[3])*sizeof(mxChar) + 1;
71 type = mxMalloc(typelen);
72 if (mxGetString(prhs[3], type, typelen)) {
73     mexErrMsgTxt("Chyba pri konverzii premennej typ");
74 }
75
76 /*kontrola typu operacie*/
77 if ((strcmp(type, "euc") == 0) || (strcmp(type, "2") == 0)) {
78     typ = 1;
79 } else if ((strcmp(type, "sup") == 0) || (strcmp(type, "max") == 0)) {
80     typ = 2;
81 } else {
82     mexErrMsgTxt("Chyba zadany typ operacie");
83 }
84
85 /* alokacia pomocnych poli a vstupnej matice */
86 res = (double *)mxMalloc((1+2*m-meq, sizeof(double));
87 pptr = mxCreateDoubleMatrix(1, 1, mxREAL);
88 neq = mxGetPr(prhs[0]);
89 eq = mxGetPr(prhs[0]);
90 meq = mxGetPr(prhs[1]);
91 y = mxGetPr(prhs[2]);
92
93 /* vytvorenie komplementarneho vektora */
94 res[0] = 0.0;
95 for (i=0; i<m; i++) {
96     res[0] = res[0] + fabs(y[i]) * neq[i]; /* sucet abs. hodnot fin*y */
97 }
98 j = 1;
99
100 /* pripocitame hodnoty zapornych nerovnic a multiplikatorov */
101 for (i=0; i<m; i++) {
102     if (neq[i] < 0.0) {
103         res[j] = neq[i];
104         j = j + 1;
105     }
106     if (y[i] < 0.0) {
107         res[j] = y[i];
108         j = j + 1;
109     }
110 }
111
112 /* pripocitame hodnoty rovnic */
113 for (i=0; i<meq; i++) {
114     res[j+i] = eq[i];
115 }
116 j = j + meq;
117
118 /* pocitanie pozadovanej normy */
119 if (typ == 1) { /* euklidovska norma */
120     pptr[0] = dnm2(&j, res, &spacing);
121 } else if (typ == 2) { /* maximova norma */
122     i = idamax(&j, res, &spacing);
123     pptr[0] = fabs(res[i-1]);
124 }
125
126 /* uvolnenie premennej res */
127 mxFree(res);
128 }
129
130 /* Pocita merit funkciu pre (Aug)Lag algoritmus. */
131 /* Pouzite funkcie z kniznice BLAS : DNRM2, IDAMAX, DASUM*/
132 /* Kompilovat pomocou mex -O -DDEFINEUNIX normstap_mex.c -lmblas/
133 4
134 5 #if !defined(_MIN32)
135
136 #define dhrm2 dhrm2
137 #define idamax idamax
138 #define dasum dasum
139 #endif
140
141 #include "mex.h"
142 #include "blas.h"
143 #include <string.h>
144 #include <math.h>
145
146 #if !defined(MAX2)
147 #define MAX2(A, B) ((A) < (B) ? (B) : (A))
148 #endif
149 #if !defined(MIN2)
150 #define MIN2(A, B) ((A) > (B) ? (B) : (A))
151 #endif
152
153 /* vstupne argumenty musia byt v tomto poradí:
154 prhs[0] = gradient (aug.) Lagrangeovej funkcie, (n x 1) double
155 prhs[1] = vektor (g_1(x)), (m x 1) double - rezidua nerovnosti
156 prhs[2] = vektor (h_j(x)), (meq x 1) double - rezidua rovnosti
157 prhs[3] = vektor Y, (m x 1) double
158 */
159
160 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
161 {
162     /* inicializacie premennych*/
163     double *grad, *neq, *eq, *y;
164     double *res, *pptr;
165     char *type;
166     mwIndex i, j, typ, spacing = 1;
167     mwIndex n, m, meq;
168     mxSize typelen;
169
170     if (nrhs != 5) {
171         mexErrMsgTxt("Funkcia MERIT potrebuje 5 vstupnych argumentov");
172     }
173     if (nlhs > 1) {
174         mexErrMsgTxt("Prilis vela vystupnych argumentov");
175     }
176     /* rozmery vektorov */
177     n = MAX2((mwIndex)mxGetM(prhs[0]), (mwIndex)mxGetM(prhs[0])) * !mxIsEmpty(prhs[0]);
178     m = MAX2((mwIndex)mxGetM(prhs[1]), (mwIndex)mxGetM(prhs[1])) * !mxIsEmpty(prhs[1]);
179     meq = MAX2((mwIndex)mxGetM(prhs[2]), (mwIndex)mxGetM(prhs[2])) * !mxIsEmpty(prhs[2]);
180
181     if (MIN2((mwIndex)mxGetM(prhs[0]), (mwIndex)mxGetM(prhs[0])) > 1) {
182         mexErrMsgTxt("Premenna GRAD musi byt vektor");
183     }
184     if (MIN2((mwIndex)mxGetM(prhs[1]), (mwIndex)mxGetM(prhs[1])) > 1) {
185         mexErrMsgTxt("Premenna FIN musi byt vektor");
186     }
187     if (MIN2((mwIndex)mxGetM(prhs[2]), (mwIndex)mxGetM(prhs[2])) > 1) {
188         mexErrMsgTxt("Premenna FEQ musi byt vektor");
189     }
190     if (m != MAX2((mwIndex)mxGetM(prhs[3]), (mwIndex)mxGetM(prhs[3])) * !mxIsEmpty(prhs[3])) {
191         mexErrMsgTxt("Vektory FIN a Y maju rozny pocet prvkov");
192     }
193
194     /*kopirovanie retazca zo vstupnej premennej 'typ'*/
195     typelen = mxGetM(prhs[4])*sizeof(mxChar) + 1;
196     type = mxMalloc(typelen);
197     if (mxGetString(prhs[4], type, typelen)) {
198         mexErrMsgTxt("Chyba pri konverzii premennej typ");
199     }
200 }

```

```

73 /*kontrola typu operacie*/
74 if ((strcmp(type, "euc") == 0) || (strcmp(type, "2") == 0)) {
75     typ = 1;
76 } else if ((strcmp(type, "sup") == 0) || (strcmp(type, "max") == 0)) {
77     typ = 2;
78 } else {
79     mexErrMsgTxt("Chybne zadany typ operacie");
80 }
81 /* alokacia pomocnych poli a vystupnej matice */
82 res = (double *)mxMalloc(2*2*m*meq, sizeof(double));
83 ppls[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
84 pptr = mxGetPr(ppls[0]);
85 grad = mxGetPr(ppls[0]);
86 neq = mxGetPr(ppls[1]);
87 eq = mxGetPr(ppls[2]);
88 y = mxGetPr(ppls[3]);
89
90 /* vytvorenie komplementarneho vektora */
91 res[0] = 0.0;
92 for (i=0; i<m; i++) {
93     res[0] = res[0] + fabs(y[i] * neq[i]); /* sucet abs. hodnot fin*ty */
94 }
95
96 /* norma gradientu Lagr. f. */
97 if ( typ == 1 ) {
98     res[1] = dnm2(kn, grad, &spacing);
99 } else if ( typ == 2 ) {
100     /* maximova norma */

```

```

101     i = idamax(kn, grad, &spacing);
102     res[1] = fabs(grad[i-1]);
103 }
104 j = 2;
105
106 /* pripocitate hodnoty zapornych nerovnic a multiplikatorov */
107 for (i=0; i<m; i++) {
108     if ( neq[i] < 0.0 ) {
109         res[j] = neq[i];
110         j = j + 1;
111     }
112     if ( y[i] < 0.0 ) {
113         res[j] = y[i];
114         j = j + 1;
115     }
116 }
117
118 /* pripocitate hodnoty rovnici */
119 for (i=0; i<meq; i++) {
120     res[j] = eq[i];
121     j = j + 1;
122 }
123
124 /* vycitanie maxima a priradenie do vystupu */
125 i = idamax(kj, res, &spacing);
126 pptr[0] = fabs(res[i-1]);
127 mxFree(res);
128 }

```

Funkcia `f_comp` počíta funkčnú hodnotu, prvú a druhú deriváciu Fischerovej funkcie. Zdrojový kód je nutné skompilovať.

```

1 /* Pocita funkcnu hodnotu, prvú a druhú deriváciu Fischerovej funkcie
2 F(a, b) = a + b - sqrt(a.*2 + b.*2 + 2*mm)
3 kompilovat pomocou mex -DDEFINIEUNIX_f_comp.c -lmbblas */
4
5 /* Vstupne parametre >
6 ppls[0] = fin : hodnoty nerovnostnych ohraniceni
7 ppls[1] = y : vektor multiplikatorov nerovnosti
8 ppls[2] = mu : perturbacny parameter Fischerovej schemy
9 ppls[3] = typ : string, jedno z FHD, DERIA = DERI_A,
10 DERIB = DERI_B, ALL.
11
12 #if !defined(_WIN32)
13 #define dnm2 dnm2
14 #define idamax idamax
15 #define dasum dasum
16 #endif
17
18 #if !defined(MAX3)
19 #define MAX3(A, B, C) (((A) < (B)) ? ((B) < (C) ? (C) : ((A) < (C) ? (C) : (A)))
20 #endif
21
22 #include "mex.h"
23 #include "blas.h"
24 #include <string.h>
25 #include <math.h>
26
27 #if !defined(MAX2)
28 #define MAX2(A, B) ((A) < (B) ? (B) : (A))
29 #endif
30 #if !defined(MIN2)
31 #define MIN2(A, B) ((A) > (B) ? (B) : (A))
32 #endif

```

```

65 if (mxGetString(prhs[3], type, typelen)) {
66     mexErrMsgTxt("Chyba pri konverzii premennej typ");
67 }
68
69 /*kontrola typu operacie*/
70 if (strcmp(type, "fhod") == 0) {
71     typ = 1;
72 } else if ((strcmp(type, "deria") == 0) || (strcmp(type, "der1_a") == 0)) {
73     typ = 2;
74 } else if ((strcmp(type, "derib") == 0) || (strcmp(type, "der1_b") == 0)) {
75     typ = 3;
76 } else if (strcmp(type, "all") == 0) {
77     typ = 4;
78 } else {
79     mexErrMsgTxt("Chybne zadany typ operacie");
80 }
81
82 /* kontrola vstupnych premennych */
83 if (typ < 4 && nlhs > 1) {
84     mxArrayMsgTxt("Prilis vela vstupnych argumentov");
85 } else if (typ == 4 && nlhs > 3) {
86     mxArrayMsgTxt("Prilis vela vstupnych argumentov");
87 }
88
89 /* priradenie smernikov */
90 fin = mxGetPr(prhs[0]);
91 y = mxGetPr(prhs[1]);
92 mu = fabs(mxGetScalar(prhs[2]));
93 sqrt_2mu = sqrt(2.0*mu);
94
95 if ( typ < 4 ) {
96     /* alokacia pomocnych poli a vystupnej matice */
97     plhs[0] = mxCreateDoubleMatrix(m, 1, mxREAL);
98     pptr = mxGetPr(plhs[0]);
99
100     for ( i=0; i<m; i++ ) {
101
102         s = MAX3( fabs(fin[i]), fabs(y[i]), sqrt_2mu );
103         sqrt_f = s*sqrt( (fin[i]/s)*(fin[i]/s) +
104             2*mu/s/s );
105
106         if (typ == 1) {
107             pptr[i] = fin[i] + y[i] - sqrt_f; /* funkcia hodnota */
108         } else if (typ == 2) {
109             pptr[i] = 1 - fin[i] / sqrt_f; /* derivacia podla a */
110         } else if (typ == 3) {
111             pptr[i] = 1 - y[i] / sqrt_f; /* derivacia podla b */
112         }
113     }
114     } else if (typ == 4) {
115
116         /* alokacia pomocnych poli a vystupnej matice */
117         plhs[0] = mxCreateDoubleMatrix(m, 1, mxREAL);
118         pptr = mxGetPr(plhs[0]);
119
120         /* alokacia dodatocnej pamate pre ostatne vystupy */
121         double *ppntr1, *ppntr2;
122         plhs[1] = mxCreateDoubleMatrix(m, 1, mxREAL);
123         pptr1 = mxGetPr(plhs[1]);
124         plhs[2] = mxCreateDoubleMatrix(m, 1, mxREAL);
125         pptr2 = mxGetPr(plhs[2]);
126
127         for ( i=0; i<m; i++ ) {
128             s = MAX3( fabs(fin[i]), fabs(y[i]), sqrt_2mu );
129             sqrt_f = s*sqrt( (fin[i]/s)*(fin[i]/s) +
130                 (y[i]/s) * (y[i]/s) +
131                 2*mu/s/s );
132             pptr[i] = fin[i] + y[i] - sqrt_f; /* funkcia hodnota */
133             pptr1[i] = 1 - fin[i] / sqrt_f; /* derivacia podla a */
134             pptr2[i] = 1 - y[i] / sqrt_f; /* derivacia podla b */
135         }
136     }
137 }

```