UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA

DIPLOMOVÁ PRÁCE

Sekvencionální kvadratické programování

Katedra matematické analýzy a aplikací matematiky
Vedoucí bakalářské práce: **Mgr. Jana Burkotová, Ph.D.**
Vypracoval(a): **Bc. Martin Veselík**
Studijní program: N1103 Aplikovaná matematika
Studijní obor Aplikace matematiky v ekonomii
Forma studia: prezenční
Rok odevzdání: 2022

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Bc. Martin Veselík

Název práce: Sekvencionální kvadratické programování

Typ práce: Diplomová práce

Pracoviště: Katedra matematické analýzy a aplikací matematiky

Vedoucí práce: Mgr. Jana Burkotová, Ph.D.

Rok obhajoby práce: 2022

Abstrakt: Sekvencionální kvadratické programování (SQP) je jedna z nejefektivnějších metod nelineárního programování, kdy hledáme minimum nelineární funkce omezené nelineárními podmínkami. Základem této metody je znalost kvadratického programování a metod pro řešení úloh kvadratického programování. Následně se práce přímo zabývá základní formou SQP metody, odvozením a konstrukcí základního algoritmu. Předposlední kapitola řeší možnosti, jak zlepšit základní algoritmus o aproximaci hessiánu a případnou úpravu kroku. Poslední kapitola popisuje podmínky konvergence.

Klíčová slova: Nelineární programování, Kvadratické programování, Sekvencionální kvadratické programování, SQP, Optimalizace, Podmíněná optimalizace

Počet stran: 95

Počet příloh: 1

Jazyk: český

BIBLIOGRAPHICAL IDENTIFICATION

Author: Bc. Martin Veselík

Title: Sequential quadratic programming

Type of thesis: Diploma

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: Mgr. Jana Burkotová, Ph.D.

The year of presentation: 2022

Abstract: Sequential quadratic programming (SQP) is one of the most efficient methods of nonlinear programming, where we look for a minimum of a nonlinear function bounded by nonlinear constraints. The basis of this method is knowledge of quadratic programming and methods for solving quadratic programming problems. Subsequently, the work directly deals with the basic form of the SQP method, derivation and construction of the basic algorithm. The penultimate chapter addresses the possibilities of how to improve the basic algorithm for Hessian approximation and possible step adjustment. The last chapter describes the conditions of convergence.

Key words: Nonlinear Programming, Quadratic Programming, Sequential Quadratic Programming, SQP, Optimization, Constrained Optimization

Number of pages: 95

Number of appendices: 1

Language: Czech

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně pod vedením paní Mgr. Jany Burkotové, Ph.D. a všechny použité zdroje jsem uvedl v seznamu literatury.

V Olomouci dne ....

podpis

Obsah

Úvod **7**

1 Nelineární programování **9**
1.1 Podmínky optimality 12

2 Kvadratické programování **22**
2.1 Kvadratické programování s omezeními tvaru rovností 23
2.1.1 Metoda nulového prostoru 24
2.2 Kvadratické programování s omezeními tvaru nerovností 32
2.2.1 Metoda aktivní množiny 33

3 Sekvencionální kvadratické programování - SQP **50**
3.1 Úlohy nelineárního programování s rovnostními omezeními 50
3.1.1 Základní algoritmus SQP pro úlohy s podmínkami tvaru rovností 58
3.2 Úlohy nelineárního programování s rovnostními a nerovnostními omezeními 63
3.2.1 Základní algoritmus SQP pro obecnou úlohu nelineárního programování 64

4 Rozšíření základního algoritmu **74**
4.1 Aproximace hessiánu 74
4.1.1 Algoritmus SQP doplněný o BFGS 76
4.2 Úprava kroku 81
4.2.1 Algoritmus SQP s merit funkcí 84

5 Konvergence **90**

Literatura **96**

Poděkování

Rád bych poděkoval paní Mgr. Janě Burkotové, Ph.D. a paní doc. RNDr. Jitce Machalové, Ph.D. za odborné vedení, pomoc a rady, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnovaly. Dále děkuji rodině i kolegyním, jenž mne podporovali během psaní práce.

Úvod

Úloha nelineárního programování je nejčastější typ úlohy, kterou můžeme v rámci optimalizace řešit. Jedná se o úlohy, kdy minimalizujeme nelineární funkci a řešení je podmíněno omezujícími podmínkami, jenž jsou také obecně nelineární.

V této práci se zaměříme konkrétně na metodu Sekvencionálního kvadratického programování, zkráceně SQP. V první kapitole se seznámíme se základními pojmy nelineárního programování, jenž jsou nutné k pochopení celé problematiky optimalizace. Velký důraz je dán i na podmínky optimality, jenž jsou základním kamenem při hledání minima daných úloh a navíc si představíme i jejich speciální tvar pro úlohy konvexního programování.

Po seznámení se základními pojmy a podmínkami optimality si projdeme nejjednoduší úlohy nelineárního programování a to úlohy kvadratického programování. Na začátku budou uvedeny základní pojmy konkrétně pro tento typ úloh a následně se podíváme přímo na metody řešení úloh kvadratického programování. V první řadě půjde úlohy s omezeními tvaru rovností, k jejichž řešení můžeme využít například metodu nulového prostoru, která bude následně demonstrována na konkrétních příkladech. Následně půjde o úlohy kvadratického programování s omezeními tvaru nerovností, s nimiž je spojena metoda aktivní množiny.

Ve třetí kapitole se už přímo podíváme na metodu SQP. Odvození pricipu této metody bude provedeno na úlohách nelineárního programování s omezeními tvaru rovností. Což následně zobecníme na úlohy nelineáního programování s omezeními tvaru rovností i nerovností. U každého typu úloh bude uveden i základní tvar algoritmu a jeho použití na příkladu.

V předposlední kapitole ukážeme, jak lze upravit základní algoritmus, abychom usnadnili výpočty a urychlili konvergenci. Například aproximací hessiánu kvazi-newtonovou metodou nebo úpravou kroku pomocí merit funkcí. V poslední kapitole jsou uvedeny konvergenční věty pro algoritmy SQP.

Každou metodu, kterou zde budeme probírat jsem doplnil o program v MATLABU, pro jejich přímé použití v tomto softwaru. Navíc budou všechny naprogramované funkce ukázány na příkladech, aby bylo možné porovnat ruční výpočet s výpočtem v MATLABU.

Kapitola 1

Nelineární programování

V této kapitole se seznámíme se základními pojmy nelineárního programování a podmínkami optimality. Značení je převzato převážně z [3] a je i hlavním zdrojem pro tuto problematiku.

Základní pojmy

Úlohou nelineárního programování rozumíme úlohu, kdy hledáme bod v němž nelineární funkce nabývá lokálního nebo globálního minima, přičemž oblast na které takový bod hledáme je omezena rovnostními a nerovnostními podmínkami. Tato úloha nelineárního programování se dá zapsat jako

$$\begin{cases} \text{minimalizovat } f(x) & x \in X \\ \text{za podmínek } g_i(x) \leq 0, & i \in I = \{1, \ldots, m\} \\ \qquad\qquad\quad h_j(x) = 0, & j \in J = \{1, \ldots, r\} \end{cases} \tag{1.1}$$

kde $f(x)$, $g_i(x)$, $h_j(x)$ jsou dané funkce n proměnných, množina X je otevřenou podmnožinou $\mathbb{R}^n$. Funkce $g_i(x)$ představuje i-tou omezující podmínku tvaru nerovnosti a funkce $h_j(x)$ představuje j-tou omezující podmínku tvaru rovnosti. Budeme předpokládat, že funkce f, g_i, h_j jsou spojitě diferencovatelné. V dalším textu se budou omezující podmínky uvádět i pomocí vektorových funkcí

$$g(x) = \begin{pmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{pmatrix} \in \mathbb{R}^m, \; h(x) = \begin{pmatrix} h_1(x) \\ \vdots \\ h_r(x) \end{pmatrix} \in \mathbb{R}^r,$$

stejně tak i matice sestavené z jejich gradientů

$$\nabla g(x) = \begin{pmatrix} \nabla g_1(x)^T \\ \vdots \\ \nabla g_m(x)^T \end{pmatrix} \in \mathbb{R}^{m\times n}, \ \nabla h(x) = \begin{pmatrix} \nabla h_1(x)^T \\ \vdots \\ \nabla h_r(x)^T \end{pmatrix} \in \mathbb{R}^{r\times n}$$

V některých případech budeme předpokládat, že funkce f, g_i, h_j jsou dvakrát spojitě diferencovatelné. Z tohoto důvodu mohou být použity matice sestavené z jejich hessiánů

$$\nabla^2 g(x) = \begin{pmatrix} \nabla^2 g_1(x)^T \\ \vdots \\ \nabla^2 g_m(x)^T \end{pmatrix} \in \mathbb{R}^{m\times n}, \ \nabla^2 h(x) = \begin{pmatrix} \nabla^2 h_1(x)^T \\ \vdots \\ \nabla^2 h_r(x)^T \end{pmatrix} \in \mathbb{R}^{r\times n}.$$

V celém textu budeme uvažovat sloupcové vektory. K označení vektorů a matic nebude použito tučné písm, jak tomu bývá v jiných textech. Z kontextu bude vždy zřejmé, zda se jedná o matici, vektor nebo skalár.

V případě nepodmíněné optimalizace, bychom se zajímali o množinu $X \subset \mathbb{R}$ a přímo na ni bychom hledali minimum dané funkce. V případě nelineárního programování musíme brát v úvahu i omezující podmínky, které vymezují *přípustnou množinu*, na které budeme hledat minimum funkce.

Definice 1.1. *Přípustnou množinou úlohy nelineárního programování* (1.1) *budeme rozumnět množinu*

$$S = \{x \in X : g(x) \leq 0, h(x) = 0\}.$$

Obecně vždy musíme rozlišit o kterém minimu vlastně mluvíme. V první řadě jde o lokální minimum, kterých může mít minimalizovaná funkce nekonečně mnoho.

Definice 1.2. *Bod $x^* \in S$ nazveme bodem lokálního minima úlohy* (1.1)*, jestliže existuje $\delta > 0$ takové, že*

$$f(x^*) \leq f(x) \qquad \forall x \in S \cap B(x^*, \delta),$$

kde $B(x^*, \delta)$ je δ-okolí bodu x^*.

Platí-li pro $x \neq x^*$ ostrá nerovnost, hovoříme o ostrém lokálním minimu.

Druhým typem je globální minimum. Bodů ve kterých funkce nabývá globálního minima může být taktéž nekonečně mnoho, ale funkční hodnota, která těmto bodům odpovídá může být pouze jedna.

Definice 1.3. *Bod $x^* \in S$ nazveme bodem globálního minima úlohy (1.1), jestliže*

$$f(x^*) \leq f(x) \qquad \forall x \in S.$$

Platí-li pro $x \neq x^*$ ostrá nerovnost, hovoříme o ostrém globálním minimu.

V rámci nepodmíněné optimalizace bychom hledali lokální nebo globální minimum nelineární funkce f na celém $X \subset \mathbb{R}^n$, ale v případě nelineárního programování obsahuje úloha (1.1) omezující podmínky, které definují přípustnou množinu bodů S. V takovém případě bod řešení úlohy nelineárního programování (1.1) nemusí být totožný s bodem, ve kterém nelineární funkce f nabývá svého lokálního nebo globálního minima na $X \subset \mathbb{R}^n$.
Obecná úloha (1.1) je omezena podmínkami rovnostního i nerovnostního typu. Podíváme-li se na tvar nerovnostních omezujících podmínek můžeme se zamyslet nad situací, kdy budou splněny ve formě rovnosti $g_i(x) = 0$. Z geometrického hlediska se jedná o případ, kdy máme bod x, který leží na hranici přípustné množiny, která je definovaná nerovnostní omezující podmínkou $g_i(x) \leq 0$. Souhrně všechny nerovnostní podmínky, které v daném bodě budou splněny jako rovnostní nazýváme *aktivní podmínky.*

Definice 1.4. *Nechť $x \in S$. Množina*

$$I(x) = \{i \in I : g_i(x) = 0\}$$

značí množinu indexů aktivních podmínek tvaru nerovnosti v bodě x. Spolu s množinou indexů všech podmínek tvarů rovností tvoří množinu všech aktivních podmínek v bodě x.

S aktivními podmínkami pracují některé metody nelineárního programování, např. metoda aktivní množiny, která bude popsána v druhé kapitole 2.

1.1. Podmínky optimality

Základní myšlenkou optimalizace je nalézt řešení úlohy (1.1), tj. bod $x^* \in S$, ve kterém funkce f nabývá svého minima na přípustné množině S. V takovém případě nám ale vzniká otázka, jaké podmínky musí bod minima splňovat a případně, jak ho najít. Tuto problematiku řeší podmínky optimality, které mají význam jak v teorii, tak i pro praktické výpočty, jelikož některé metody nelineárního programování jsou z nich přímo odvozeny.
Obecně podmínky optimality dělíme na dvě skupiny a to na podmínky nutné a podmínky postačující. Z pohledu teorie nutné podmínky musí být splněny v každém řešení dané úlohy (1.1). Což nevylučuje možnost jejich splnění i v jiných bodech, to ale neplatí pro postačující podmínky. Z praktického hlediska hledáme všechny body ve kterých jsou splněny nutné podmínky optimality. Tím získáme množinu bodů,tzv. kandidátů, z nichž některé mohou být skutečným bodem minima. Postačující podmínky ověřují, který z kandidátů je skutečně řešením dané úlohy (1.1). Dále se podmínky dělí na podmínky prvního a druhé řádu. V případě že se jedná o podmínky 1. řádu jsou podmínky zaměřeny na ověřování maximálně první derivace neboli na gradienty spojené s úlohou. Podmínky optimality 2. řádu navíc řeší i vlastnosti hessiánů, které jsou s úlohou spojeny.

Karush-Kuhn-Tuckerovy podmínky, zkráceně označované jako KKT podmínky, jsou zobecněním podmínek optimality pro nepodmíněné úlohy. V dalším textu budou uvedeny KKT nutné podmínky 1.řádu a KKT postačující podmínky 2. řádu.

Věta 1.1. *[3, Věta 2.4][Karush-Kuhn-Tuckerovy nutné podmínky] Nechť $x^* \in S$ je bod lokálního minima úlohy (1.1). Nechť $I(x^*) = \{i \in I : g_i(x^*) = 0\}$ a nechť gradienty $\nabla g_i(x^*), i \in I(x^*)$, a $\nabla h_j(x^*), j = 1, \ldots, r$, jsou lineárně nezávislé. Potom existuje dvojice vektorů $(\lambda^*, \mu^*) \in \mathbb{R}^m \times \mathbb{R}^r$ taková, že platí*

1. $\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{j=1}^{r} \mu_j^* \nabla h_j(x^*) = 0,$

2. $\lambda_i^* \geq 0, \qquad \forall i = 1, \ldots, m,$

3. $\lambda_i^* g_i(x^*) = 0, \qquad \forall i = 1, \ldots, m.$

První KKT podmínka ve Větě 1.1, je zobecněním podmínky pro nepodmíněné úlohy, kde je nutné, aby gradient v bodě minima byl nulový. První podmínku můžeme zapsat v maticovém tvaru

$$\nabla f(x^*) + \nabla g(x^*)^T \lambda^* + \nabla h(x^*)^T \mu^* = 0.$$

Vektory λ a μ se nazývají Lagrangeovy multiplikátory. Jak uvádí Věta 1.1, pouze pro multiplikátory spojené s podmínkami tvaru nerovností je nutné ověřit i jejich znaménko. V literatuře, např. [5], se můžeme setkat i s úlohami, kde podmínky tvaru nerovností mají tvar $g(x) \geq 0$. V takovém případě bude druhá podmínka mít tvar $\lambda_i^* \leq 0, \forall i = 1, \ldots, m$ nebo je nutné upravit omezující podmínky do tvaru $-g(x) \leq 0$. Třetí KKT podmínka se někdy uvadí i pod pojmem podmínky komplementarity. Obecně můžeme říct, že multiplikátory spojené s omezujícími podmínkami tvaru nerovnosti, které nejsou v bodě x^* nejsou aktivní musí být nulové, tj.

$$\lambda_i^* = 0, \qquad \forall i \notin I(x^*).$$

V případě, že platí $\lambda_i^* > 0, \forall i \in I(x^*)$, potom mluvíme o striktní komplementaritě.

Právě nutné KKT podmínky jsou základem pro odvození některých metod nelineárního programování. Ověření KKT podmínek je názorně představeno na následujících příkladech.

Příklad 1.1. *[1] Máme úlohu nelineárního programování s omezeními tvaru nerovností*

$$\begin{cases} \textit{minimalizovat } f(x) = -2x_1 + x_2, \\ \textit{za podmínek} \quad (1 - x_1)^3 - x_2 \geq 0, \\ \qquad\qquad\quad x_2 + 0.25x_1^2 - 1 \geq 0. \end{cases}$$

V tomto případě upravíme nerovnostní podmínky a naše úloha bude mít poté tvar

$$\begin{cases} \textit{minimalizovat } f(x) = -2x_1 + x_2, \\ \textit{za podmínek} \quad x_2 - (1 - x_1)^3 \leq 0, \\ \qquad\qquad\quad 1 - x_2 - 0.25x_1^2 \leq 0. \end{cases} \tag{1.2}$$

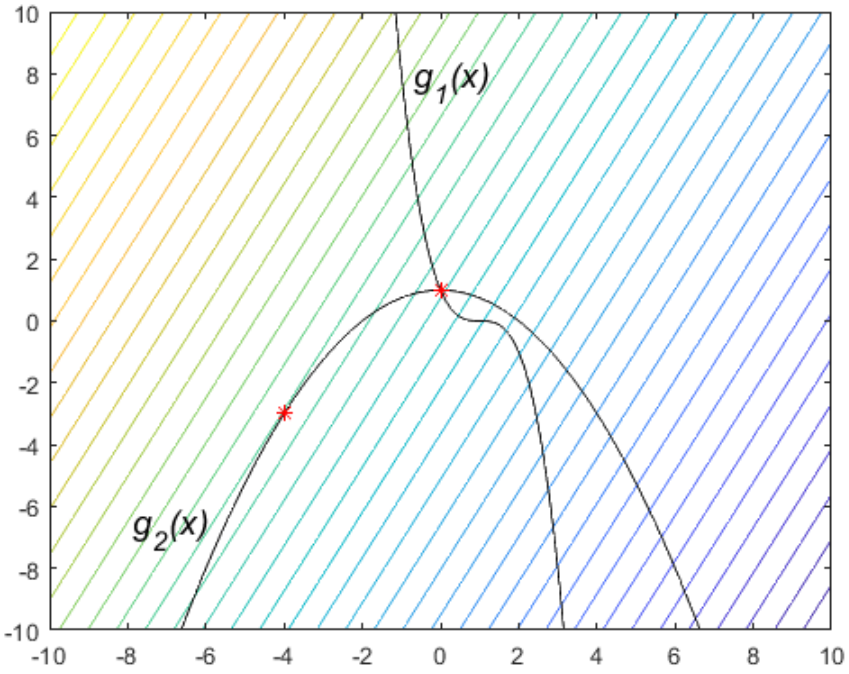
Nutné KKT podmínky pro tuto úlohy jsou ve tvaru

1. $\begin{aligned} &-2 + 3\lambda_1(1 - x_1)^2 - 0.5\lambda_2 x_1 = 0 \\ &1 + \lambda_1 - \lambda_2 = 0 \end{aligned}$

2. $\lambda_1 \geq 0,\ \lambda_2 \geq 0$

3. $\begin{aligned} &\lambda_1 \cdot (x_2 - (1 - x_1)^3) = 0 \\ &\lambda_2 \cdot (1 - 0.25x_1^2 - x_2) = 0 \end{aligned}$

Máme celkem čtyři rovnice o čtyřech neznámých. Podíváme-li se na třetí podmínku můžeme vzít v úvahu několik alternativ, případ kdy oba multiplikátory $\lambda_1 = 0$ *a* $\lambda_2 = 0$*, nemůže nastat jelikož by nebyla splněna druhá rovnice z první podmínky. Z toho můžeme odvodit, že minimálně jedna omezující podmínka bude aktivní. Případ, kdy obě podmínky budou aktivní nám dává bod* $x = (0, 1)^T$ *a* $\lambda = (\lambda_1, \lambda_2)^T = (\frac{2}{3}, \frac{5}{3})^T$*, který splňuje nutné KKT podmínky. Dále pro* $\lambda_1 = 0$*, bude* $\lambda_2 = 1$ *a odpovídající bod* $x = (-4, -3)^T$*, který také splňuje nutné KKT podmínky. Poslední možnost* $\lambda_2 = 0$*, kde* $\lambda_1 = -1$ *nesplňuje druhou podmínku, protože* λ_1 *je záporné. Na obrázku 1.1 je tato úloha zobrazena graficky. Přípustná množina je zde v levé části obrázku, která je vymezena omezujícími podmínkami, spolu s vrtevnicemi funkce* $f(x)$ *je zřejmé, že bod řešení úlohy bude ležet na hranici přípustné množiny. Dále jsou na obrázku 1.1 vykresleny dva body, ve kterých jsou splněny nutné KKT podmínky. Z obrázku je zřejmé, že pouze jeden z těchto bodů je bodem lokálního minima.*



Obrázek 1.1: Úloha (1.2)

Příklad 1.2. *[5] Máme úlohu nelineárního programování s omezením tvaru nerovnosti,*

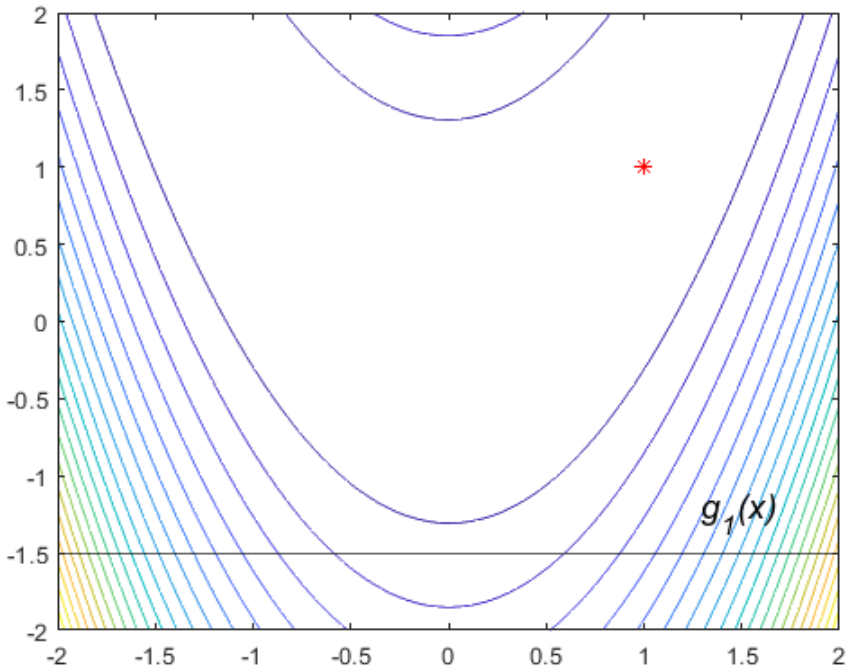
$$\begin{cases} \textit{minimalizovat } f(x) = 100(x_2 - (x_1)^2)^2 + (1 - x_1)^2 \\ \textit{za podmínek} \quad x_2 \geq -1.5. \end{cases} \tag{1.3}$$

konkrétně se jedná o minimalizaci Rosenbrokovy funkce pro dvě neznámé, nutné KKT podmínky pro tuto úlohy jsou ve tvaru

1. $-400x_1x_2 + 400(x_1)^3 + 2x_1 - 2 + \lambda_1 \cdot 0 = 0$
   $200(x_2 - (x_1)^2) - \lambda_1 = 0$
2. $\lambda_1 \geq 0$
3. $\lambda_1(-x_2 - 1.5) = 0$

Z třetí podmínky můžeme určit dvě možné alternativy, první možností je $x_2 = -1.5$. *Dosazením do první rovnice z první podmínky zjistíme, že pro* x_1 *nedostaneme výpočtem žádné reálné kořeny a tedy tato alternativa nevede k řešení.*

Druhou alternativou je $\lambda_1 = 0$*, dosazením do druhé rovnice první podmínky a substitucí do první rovnice získáme řešení* $x = (1, 1)^T$*, které splňuje nutné KKT podmínky. Tato úloha je graficky zobrazena na obrázku 1.2. Přípustná množina je zde celá polorovina nad vykreslenou funkcí* $g_1(x)$ *a dále je na obrázku 1.2 vykreslen bod, ve kterém jsou splněny nutné KKT podmínky.*



Obrázek 1.2: Úloha (1.3)

Lagrangeovy multiplikátory, případně KKT multiplikátory, přímo souvisí s Lagrangeovou funkcí, která velmi zjednodušuje zápis podmínek optimality, ale využívá se i v některých metodách optimalizace.

Definice 1.5. *[Lagrangeova funkce] Lagrangeovou funkcí nebo lagrangiánem příslušejícím úloze nelineárního programování* (1.1) *nazveme funkci*

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{r} \mu_j h_j(x)$$

V dalším textu se setkáme i s gradientem Lagrangeovy funkce

$$\nabla_x L(x, \lambda, \mu) = \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) + \sum_{j=1}^{r} \mu_j \nabla h_j(x)$$

a hessiánem Lagrangeovy funkce

$$\nabla^2_{xx} L(x, \lambda, \mu) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x) + \sum_{j=1}^{r} \mu_j \nabla^2 h_j(x).$$

Pro přehlednost lze první KKT podmínku lze psát ve tvaru

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0.$$

Tato podmínka odpovídá hledání stacionárního bodu Lagrangeovy funkce.

Poznámka 1 (Citlivost)**.** *Hodnota Lagrangeových multiplikátorů přímo souvisí s citlivostní analýzou, tj. jak je daný bod citlivý na přítomnost omezující podmínky. Obecně pro všechny neaktivní podmínky v daném bodě jsou multiplikátory nulové, což znamená, že daný bod není na tyto omezující podmínky citlivý. To platí i pro aktivní podmínky, kde je hodnota multiplikátoru nulová. Vezmeme-li příklad 1.1, kde jsme vypočítali bod* $x = (-4, -3)^T$ *a pro omezující podmínky* g_1*,* g_2 *odpovídající multiplikátory* $\lambda_1 = 0$*,* $\lambda_2 = 1$*. V tomto případě můžeme říct, že daný bod není citlivý na omezující podmínku* g_1*. Pro omezující podmínku* g_2 *nás může zajímat síla citlivosti. Pokud bude hodnota* $\lambda_i \|\nabla g_i(x)\|$ *je malá, pak citlivost není silná. Pro uvedený příklad bude tedy*

$$\lambda_2 \|\nabla g_2(x)\| = 2.2361.$$

Podrobnosti k analýze citlivosti lze najít např. v [1].

Jelikož KKT podmínky představené ve Větě 1.1 jsou nutnými podmínkami optimality musí platit v každém bodu lokálního minima úlohy nelineárního programování (1.1), což ale nevylučuje existenci jiných bodů z přípustné množiny ve kterých jsou také splněny KKT podmínky, jak bylo demonstrováno na Příkladu

1.2. Abychom mohli rozhodnout, které z určených bodů jsou skutečně bodem lokálního minima je třeba ověřit i platnost postačujících podmínek v daných bodech.

Pro definování těchto podmínek je třeba se nejdříve seznámit s dvěma pojmy a to množina *linearizovaných přípustných směrů* a *kritický kužel*.

Definice 1.6. *Nechť $x \in S$ je daný bod a $I(x)$ je příslušná množina indexů aktivních omezení tvaru nerovnosti v tomto bodě. Množinou linearizovaných přípustných směrů v bodě x budeme rozumět množinu*

$$LFD(x) = \left\{p \in \mathbb{R}^n : \nabla g_i(x)^T p \leq 0 \quad \forall i \in I(x),\, \nabla h_j(x)^T p = 0 \quad \forall j = 1, \ldots, r\right\}.$$

Definice 1.7. *Nechť $x^* \in S$ je bod lokálního minima úlohy (1.1), nechť λ^*, μ^* jsou odpovídající vektory Lagrangeových multiplikátorů splňující KKT podmínky. Kritickým kuželem v bodě x^* rozumíme množinu*

$$C(x^*) = \left\{p \in LFD(x^*) : \nabla g_i(x^*)^T p = 0\,, \forall i \in I(x^*)\, pro\ něž \lambda_i^* > 0\right\}.$$

Pomocí kritického kuželu můžeme následně zformulovat nutné a postačující podmínky optimality 2. řádu. Pro tyto podmínky je nutné, aby dané funkce f, g a h byly dvakrát spojitě diferencovatelné.

Věta 1.2. *[3, Věta 5.2] Nechť $x^* \in S$ je bod lokálního minima úlohy (1.1). Nechť v bodě x^* jsou gradienty $\nabla g_i(x^*)$, $i \in I(x^*)$, a $\nabla h_j(x^*)$, $j = 1, \ldots, r$ lineárně nezávislé. Nechť $\lambda^* \in \mathbb{R}^m$ a $\mu^* \in \mathbb{R}^r$ jsou vektory Lagrangeových multiplikátorů, které splňují KKT podmínky, tj. platí*

- $\nabla_x L(x^*, \lambda^*, \mu^*) = 0$,
- $g_i(x^*) \leq 0 \qquad \forall i = 1, \ldots, m$,
- $h_j(x^*) = 0 \qquad \forall j = 1, \ldots, r$,
- $\lambda_i^* \geq 0 \qquad \forall i = 1, \ldots, m$,
- $\lambda_i^* g_i(x^*) = 0 \qquad \forall i = 1, \ldots, m$. *Potom platí*

$$p^T \nabla^2_{xx} L(x^*, \lambda^*, \mu^*) p \geq 0 \qquad \forall p \in C(x^*).$$

Věta 1.3. *[3, Věta 5.3][Postačující podmínky optimality 2. řádu] Nechť $x^* \in S$ a nechť existují vektory Lagrangeových multiplikátorů $\lambda^* \in \mathbb{R}^m$, $\mu^* \in \mathbb{R}^r$ tak, že společně splňují KKT podmínky*

1. $\nabla_x L(x^*, \lambda^*, \mu^*) = 0$
2. $g_i(x^*) \leq 0 \qquad \forall i = 1, \ldots, m,$
3. $h_j(x^*) = 0 \qquad \forall j = 1, \ldots, r,$
4. $\lambda_i^* \geq 0 \qquad \forall i = 1, \ldots, m,$
5. $\lambda_i^* g_i(x^*) = 0 \qquad \forall i = 1, \ldots, m.$

Nechť navíc platí

$$p^T \nabla_{xx}^2 L(x^*, \lambda^*, \mu^*) p > 0 \qquad \forall p \in C(x^*),\ p \neq 0$$

Potom x^* je bodem ostrého lokálního minima úlohy (1.1).

Obě věty jsou uvedeny i ve zdroji [1], liší se pouze použitým značením. Rozdíl mezi těmito větami je hlavně v předpokladu na bod x^*, kdy u nutných podmínek předpokládáme, že bod x^* je bodem lokálního minima, ale u postačujících podmínek tento předpoklad není.

Příklad 1.3. *Vezmeme teď zadání předchozího příkladu 1.1. Kde jsme měli úlohu*

$$\begin{cases} \text{minimalizovat } f(x) = -2x_1 + x_2 \\ \text{za podmínek} \quad (1 - x_1)^3 - x_2 \geq 0 \\ \qquad\qquad\qquad\quad x_2 + 0.25x_1^2 - 1 \geq 0 \end{cases}$$

Při řešení nutných KKT podmínek jsme dostali dva body, které tyto podmínky splňují $x = (0, 1)^T$, $\lambda = (\frac{2}{3}, \frac{5}{3})^T$ a $x = (-4, -3)^T$, $\lambda = (0, 1)^T$.
Zaměříme se teď na bod $x = (-4, -3)^T$. Hodnoty multiplikátorů jsou zde $\lambda_1 = 0$ a $\lambda_2 = 1$. Jelikož jsou splněny nutné KKT podmínky je třeba ověřit zda hessián lagrangeovy funkce $\nabla_{xx}^2 L(x^*, \lambda^*)$ je pozitivně definitní na kritickém kuželu $C(x^*)$.

V tomto případě je kritický kužel dán množinou $C(x) = \{p \in \mathbb{R}^2 : \nabla g_2(x)^T p = 0\}$. *Určíme hessián lagrangeovy funkce, kde*

$$\nabla^2 f(x) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \nabla^2 g_1(x) = \begin{pmatrix} -6 + 6x_1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \nabla^2 g_2(x) = \begin{pmatrix} -0.5 & 0 \\ 0 & 0 \end{pmatrix}.$$

Hessián Lagrangeovy funkce v bodě $x = (-4, -3)^T$ *pak bude*

$$\nabla^2_{xx} L(x, \lambda) = \nabla^2 f(x) + \lambda_1 \nabla^2 g_1(x) + \lambda_2 \nabla^2 g_2(x) = \begin{pmatrix} -0.5 & 0 \\ 0 & 0 \end{pmatrix}.$$

Nyní ověříme, zda pro směr $p = (1, 2)^T \in C(x)$ *platí podmínka* $p^T \nabla^2_{xx} L(x, \lambda) p > 0$

$$(1,\ 2) \begin{pmatrix} -0.5 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = -0.5 < 0$$

Můžeme tedy říct, že bod $x = (-4, -3)^T$ *není bodem lokálního minima ikdyž splňuje nutné KKT podmínky 1. řádu, jelikož nesplňuje nutné podmínky 2. řádu. V případě bodu* $x = (0, 1)^T$ *se nejdříve podíváme na kritický kužel, kde*

$$\nabla g_1(x) = \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \quad \nabla g_2(x) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}.$$

V tomto případě kritický kužel obsahuje pouze jediný vektor $p = (0,\ 0)^T$. *Podíváme-li se na podmínku pro hessián Lagrangeovy funkce, můžeme říct , že pro nutné podmínky 2. řádu je splněna ve tvaru rovnosti. Pro postačující podmínky je množina nenulových vektorů, které patří do kritického kuželu, prázdná, proto můžeme říct, že podmínka je splněna. Z toho plyne, že bod* $x = (0, 1)^T$ *je bodem lokálního minima.*

Pro některé úlohy (1.1) může být složité vypočítat hessián Lagrangeovy funkce a nemusí být snadné ověřit zda je hessián pozitivně definitní nebo pozitivně semidefinitní na kritickém kuželu. Nejjednoduší formou úloh v takovém případě jsou úlohy konvexního programování, kde odpadá nutnost výpočtu hessiánu Lagrangeovy funkce, jak se můžeme přesvědčit v následující větě.

Definice 1.8. *[Úloha konvexního programování] Úloha nelineárního programování* (1.1)*, v níž $f(x)$ je konvexní funkce na S a přípustná množina S je konvexní podmnožina $\mathbb{R}^n$, se nazývá úloha konvexního programování.*

V rámci konvexního programování musí platit stejné podmínky jako u obecného tvaru úloh nelineárního programování (1.1). Konkrétně pro úlohy konvexního programování je možné zformulovat podmínky optimality, které pro tento konkrétní typ úloh budou jak nutné tak i postačující.

Věta 1.4. *[Nutné a postačující podmínky] Nechť $x^* \in S$ je přípustným bodem úlohy* (1.1) *a nechť existují vektory $(\lambda^*, \mu^*) \in \mathbb{R}^m \times \mathbb{R}^r$ takové, že platí*

1. $\nabla_x L(x^*, \lambda^*, \mu^*) = 0,$
2. $\lambda^* \geq 0,$
3. $\lambda_i^* g_i(x^*) = 0 \qquad \forall i = 1, \ldots, m.$

Jsou-li $f(x)$ a $g_i(x)$, $i = 1, \ldots, m$ konvexní, $h_j(x)$, $j = 1, \ldots, r$ lineární funkce a X je konvexní podmnožina $\mathbb{R}^n$, potom x^* je bodem globálního minima úlohy (1.1).

Pokud tedy úloha (1.1) splňuje definici 1.8, potom můžeme říct, že KKT podmínky pro úlohu konvexního pogramování jsou podmínkami nutnými i postačujícími. Tato věta plyne přímo z Věty 3.2 [3], kde se uvádí pod názvem *KKT postačující podmínky - pro speciální případ.* Jedná se o obecnější větu pro pseudokonvexní funkci $f(x)$, kvazikonvexní funkce $g_i(x)$, $i = 1, \ldots, m$ a lineární funkce $h_j(x)$, $j = 1, \ldots, r$. Jelikož v této práci se s takovými funkcemi nesetkáme postačí nám věta v konkrétnějším tvaru.

Kapitola 2

Kvadratické programování

Tato kapitola popisuje obecnou úlohu kvadratického programování. Zvlášť je řešena úloha s rovnostními podmínkami pomocí metody nulového prostoru a zvlášť úloha s podmínkami ve tvaru nerovností metodou aktivní množiny, kterou lze rozšířit i na obecnou úlohu kvadratického programování. Pro studium této problematiky jsem využil převážně knih [1], [2] a [3]. Důraz je kladen převážně na vlastnosti funkce $q(x)$ a omezujících podmínek, na jejichž základě jsou založeny metody kvadratického programování.

Úlohy kvadratického programování jsou nejjednodušší formou nelineárního programování. Jedná se o optimalizační úlohy s kvadratickou funkcí $q(x)$ a lineárními omezeními, které se obecně dají zapsat jako

$$\begin{cases} \text{minimalizovat } q(x) = \frac{1}{2}x^T Cx - d^T x \\ \text{za podmínky } \quad Ax = b \\ \qquad\qquad\quad\ \ Dx \leq e \end{cases} \tag{2.1}$$

kde $q(x)$ je daná kvadratická funkce o n proměnných. Matice $C \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{r\times n}$ a $D \in \mathbb{R}^{m\times n}$. Vektory $d, x \in \mathbb{R}^n$, $b \in \mathbb{R}^r$ a $e \in \mathbb{R}^m$.
Předpokládáme, že matice A má plnou hodnost, což znamená, že všechny dané omezující podmínky tvaru rovností jsou lineárně nezávislé. Matice C je symetrická, navíc je i hessiánem funkce q a zároveň určuje charakter této funkce. Jestliže je matice C

- pozitivně definitní/semidefinitní, pak funkce q je ryze konvexní/konvexní

- negativně definitní/semidefinitní, pak funkce q je ryze konkávní/konkávní
- nedefinitní, pak funkce je sedlo-bodová.

V zadání některých příkladů obsahuje kvadratická funkce $q($ absolutní člen $c \in \mathbb{R}^n$ a má tvar $q(x) = \frac{1}{2}x^T Cx - d^T x + c$. Tento člen však nemá žádný vliv na polohu bodu minima x^* a ovlivňuje pouze funkční hodnotu v tomto bodě. Z tohoto důvodu se při řešení úlohy vynechává.

Definice 2.1. *Přípustnou množinou úlohy kvadratického programování* (2.1) *budeme rozumnět množinu*

$$S = \{x \in X \subseteq \mathbb{R}^n : Ax - b = 0, Dx - e \leq 0\}.$$

Jelikož je množina S definovaná lineárními omezeními, jedná se o konvexní množinu.

Jelikož přípustná množina S je konvexní, je tedy pouze na kvadratické funkci q, zda se bude jednat o úlohu konvexního programování.

Definice 2.2. *Úloha kvadratického programování, kde kvadratická funkce $q(x)$ je konvexní na S a přípustná množina S je konvexní podmnožina $\mathbb{R}^n$, se nazývá úloha konvexního kvadratického programování.*

Metody kvadratického programování jsou důležitým nástrojem, jak pro řešení úloh kvadratického programování, tak i pro řešení pomocných obecných úloh nelineárního programování (1.1), například pro sekvencionální kvadratické programování v Kapitole 3. Dále v textu se budeme zabývat úlohou konvexního kvadratického programování.

2.1. Kvadratické programování s omezeními tvaru rovností

Jedná se o úlohu, kde omezující podmínky mají pouze tvar rovností. Obecný zápis této úlohy

$$\begin{cases} \text{minimalizovat } q(x) = \frac{1}{2}x^T Cx - d^T x \\ \text{za podmínky} \quad Ax = b. \end{cases} \tag{2.2}$$

Předpoklad, že matice A má plnou řádkovou hodnost, je obecnou formulací dvou předpokladů a to, že počet omezujících podmínek r je menší než počet neznámých n a matice A má plnou hodnost.
Z těchto předpokladů vyplývá, že všechny podmínky jsou lineárně nezávislé a podmínka $r < n$ přímo údává řešitelnost dané úlohy:

- pokud $r = n$, pak se nejedná o úlohu optimalizace, protože řešení musí splnit n lineárních rovnic o n neznámých a funkce q nemá na výsledek žádný vliv,
- pokud $r > n$, pak soustava rovnic omezujících podmínek nemá řešení,
- pokud $r < n$, pak soustava rovnic omezujících podmínek má nekonečně mnoho řešení a má smysl mezi nimi hledat bod, ve kterém funkce q nabývá nejmenší funkční hodnoty.

Jelikož je úloha omezena lineárními podmínkami je přípustná množina S konvexní. KKT podmínky pro tuto úlohu s podmínkou přípustnosti jsou ve tvaru

$$\begin{aligned} Cx^* - d + A^T\mu^* &= 0 \\ Ax^* - b &= 0, \end{aligned} \tag{2.3}$$

kde se vzu6ije, 6e $\nabla g(x) = Cx - d$ a $\nabla h(x) = A$. Tuto soustavu můžeme přepsat do maticového tvaru

$$\begin{pmatrix} C & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \mu \end{pmatrix} = \begin{pmatrix} d \\ b \end{pmatrix}$$

To za jakých podmínek bude mít soustav KKT podmínek (2.3) řešení popisuje Věta 8.1 [3].
Metody, používané k řešení úloh (2.2), jsou založeny na řešení KKT soustavy (2.3). V případě konvexního kvadratického programování je řešení KKT soustavy (2.3) a původní úlohy (2.2) totožné.

2.1.1. Metoda nulového prostoru

V této části bude popsána metoda nulového prostoru, která je převážně ze zdroje [3], s metodou nulového prostoru se můžeme setkat i v [1], kde je ale použit

rozdílný postup.

Metoda nulového prostoru slouží k řešení úloh kvadratického programování s omezeními tvaru rovností (2.2). Metoda přímo řeší KKT soustavu (2.3), užitím nulového prostoru matice A definovaného jako $N(A) = \{p \in \mathbb{R}, Ap = 0\}$ a partikulárního řešení $\tilde{x}$ soustavy omezujících podmínek. Všechna další řešení této soustavy můžeme souhrně zapsat jako

$$x = \tilde{x} + p, \qquad p \in N(A).$$

Dosadíme do rovnic KKT soustavy (2.3) dostaneme

$$\begin{aligned} C\tilde{x} + Cp + A^T\mu &= d \\ A\tilde{x} + Ap &= b \end{aligned}$$

Podíváme-li se na druhou rovnici, jelikož $\tilde{x} \in S$, tj. $A\tilde{x} = b$ pak daná rovnice bude bude platit pouze pokud $Ap = 0$, tj. $p \in N(A)$. V takovém případě přepíšeme soustavu do tvaru

$$\begin{aligned} Cp + A^T\mu &= d - C\tilde{x}, \\ Ap &= 0. \end{aligned}$$

A získáme upravenou KKT soustavu

$$\begin{pmatrix} C & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p \\ \mu \end{pmatrix} = \begin{pmatrix} \tilde{d} \\ b \end{pmatrix}$$

$$\tilde{d} = d - C\tilde{x}.$$

Z hlediska optimalizace přecházíme od minimalizace na konvexní množině $\{x \in \mathbb{R}^n : Ax = b\}$ k minimalizaci na podprostoru $N(A)$.

Dalším krokem je určení báze nulového prostoru, která je tvořena vektory $z_1, \ldots, z_{n-r} \in \mathbb{R}^n$. Z nich sestavená matice $Z \in \mathbb{R}^{n\times(n-r)}$ má plnou sloupcovou hodnost a platí pro ni

$$Z = (z_1, \ldots, z_{n-r}), \qquad AZ = 0.$$

Jelikož $Ap = 0$, pak vektor p musí být lineární kombinací bázových vektorů, což vyjádříme pomocí neznámého vektoru koeficientů $x_Z \in \mathbb{R}^{n-r}$ jako

$$p = Zx_Z.$$

V tomto bodě nám vzniká otázka, jak vypočítat neznámý bod x_Z. Dosazením za vektor p do první rovnice upravené KKT soustavy

$$Cp + A^T\mu = \tilde{d},$$

a vynasobenim Z^T zleva, obdržíme rovnici pro určení neznámého bodu x_Z

$$Z^T C Z x_Z = Z^T \tilde{d}.$$

Díky těmto úpravám získáme redukovanou úlohu nepodmíněné optimalizace

$$\begin{cases} \text{minimalizovat } \widehat{q}(x_Z) = \frac{1}{2} x_Z^T Z^T C Z x_Z - x_Z^T Z^T \tilde{d} \\ \text{pro} \qquad\qquad\qquad x_Z \in \mathbb{R}^{n-r}. \end{cases} \tag{2.4}$$

Podobně jako tomu bylo u původní funkce q, kde charakter funkce závisel na matici C i zde bude záležet na vlastnostech matice $Z^T C Z \in \mathbb{R}^{(n-r)\times(n-r)}$.

Věta 2.1. *[3, Věta 8.2] Nechť matice A má plnou hodnost a nechť $Z \in \mathbb{R}^{n\times(n-r)}$ je taková, že platí*

$$AZ = 0, \qquad rank(A^T \mid Z) = n.$$

Potom

1. *úloha (2.2) má právě jedno řešení tehdy a jen tehdy, když je matice $Z^T C Z$ pozitivně definitní;*

2. *úloha (2.2) má nekonečně mnoho řešení tehdy a jen tehdy, když je matice $Z^T C Z$ právě jen pozitivně semidefinitní a soustava $Z^T C Z x_Z = Z^T \tilde{d}$ má řešení;*

3. *úloha (2.2) nemá řešení v zbývajících případech*

K úplnému sestavení algoritmu metody nulového prostoru je třeba vyřešit výpočet multiplikátorů, který získáme z upravených KKT podmínek

$$A^T\mu = \tilde{d} - CZx_Z.$$

a partikulárního řešení, $\tilde{x} \in R(A^T)$ je sloupcový prostor matice A^T. Partikulární řešení lze tedy zapsat pomocí vektoru koeficientů

$$\tilde{x} = A^T x_A.$$

Pokud vynásobíme tuto rovnost maticí A zleva a využijeme, že $A\tilde{x} = b$, dostaneme, soustavu rovnic pro neznámý vektor koeficientů x_A

$$(AA^T)x_A = b.$$

Postup odvození hodnoty x_A je podrobněji popsán v literatuře [3]. Uvedeny postup je shrnut v následujícím algoritmu.

Algoritmus 2.1. *[3, str.73]*
Inicializace: Najdeme bázi $\{z_i\}_{i=1}^{n-r}$ *prostoru* $N(A)$ *a tyto vektory sestavíme do matice* $Z \in \mathbb{R}^{n\times(n-r)}$.
Krok 1: postupně sestavíme a vyřešíme následující soustavy

$$(AA^T)x_A = b,$$

$$(Z^TCZ)x_Z = Z^T(d - CA^Tx_A),$$

$$(AA^T)\mu = A(d - CA^Tx_A - CZx_Z).$$

Krok 2: vypočítáme

$$x = A^Tx_A + Zx_Z.$$

Pro použití metody nulového prostoru v MATLABU jsem naprogramoval funkci **nulprostor**, kde vstupem jsou matice C, A a vektory d, b. Výstupem metody je bod minima kvadratické úlohy (2.2) x a odpovídající hodnota multiplikátoru μ.

```
function [x,mi]=nulprostor(C,d,A,b)
%Vstupy
%matice a vektory odpovídající úloze ve tvaru
%   f(x)=1/2*x'*C*x-d'x
%       A*x=b
```

```
%
%Výstupy
%x0 - bod řešení dané úlohy
%mi - odpovídající hodnota multiplikátoru pro rovnostní omezení

%výpočet matice bázové matice
Z=null(A);
tA=A.';
tZ=Z.';

%postupné řešení jednotlivých rovnic
xa=linsolve(A*tA,b);
xz=linsolve(tZ*C*Z,tZ*(d-C*tA*xa));
mi=linsolve(A*tA,A*(d-C*tA*xa-C*Z*xz));

%výpočet bodu řešení úlohy
x=tA*xa+Z*xz;
end
```

Na následujícím příkladě bude ukázán výpočet pomocí algoritmu 2.1 a použití mnou nadefinované funkce **nulprostor**.

Příklad 2.1. *[6] Máme úlohu kvadratického porgramování s omezením tvaru rovnosti*

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}(x_1^2 + x_2^2) \\ \textit{za podmínky} \quad 2x_1 - x_2 = 5. \end{cases}$$

Úlohu upravíme pro další výpočty do maticového tvaru (2.2)*, kde*

$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $d = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $A = \begin{pmatrix} 2 \\ -1 \end{pmatrix}^T$, $b = 5$.

Inicializace:

Vektor $Z = \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix}$*, obecně mluvíme o matici v případě této úlohy půjde pouze*

vektor, který vypočítáme z rovnice $AZ = 0$. *Jelikož se jedná o rovnici o dvou neznámých existuje více vektorů, které tuto rovnost splňují. Pro tuto úlohu zvolíme* $Z_1 = 1$ *a výsledný vektor bude*

$$Z = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Krok 1:

Sestavíme rovnice

$5x_A = 5,$

$5x_Z = 0,$

$5\lambda = -5.$

Získáme řešení

$x_A = 1,\ x_Z = 0,\ \lambda = -1.$

Krok 2:

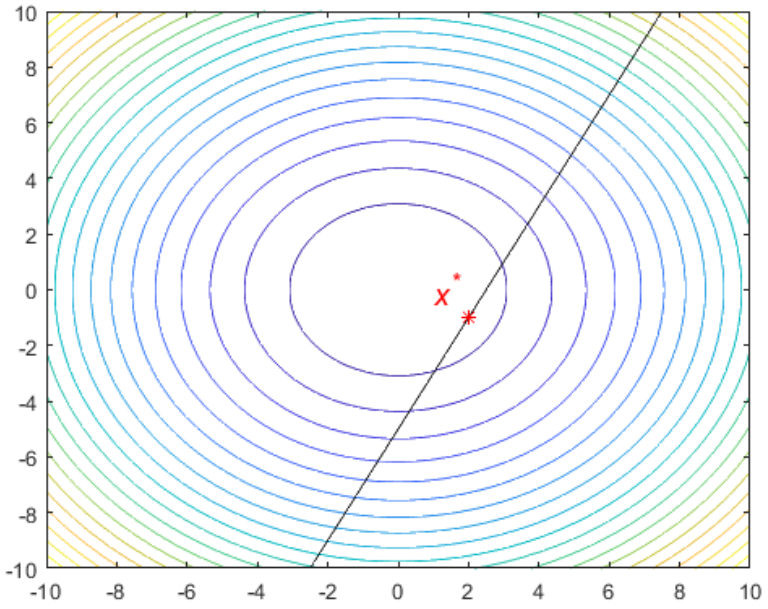
Vypočítáme

$$x^* = A^T x_A + Z x_Z = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \cdot 1 + \begin{pmatrix} 1 \\ -2 \end{pmatrix} \cdot 0 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}.$$

Při použití naprogramované funkce ***nulprostor*** *v MATLABU získáme výsledek ve tvaru*

```
[x,mi]=nulprostor([1 0;0 1],[0;0],[2 -1],5)


x =

     2
    -1


mi =

    -1
```

Příklad 2.1 je graficky znázorněn na obrázku 2.1.



Obrázek 2.1: Grafické znázornění Příkladu 2.1

Příklad 2.2. *[3] Máme úlohu kvadratického programování s omezeními tvaru rovnosti*

$$\begin{cases} \textit{minimalizovat } 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3 \\ \textit{za podmínky} \quad x_1 + x_3 = 3, \\ \qquad\qquad\quad x_2 + x_3 = 0. \end{cases}$$

Úlohu upravíme pro další výpočty do maticového tvaru (2.2), kde

$$C = \begin{pmatrix} 3 & 1 & 0.5 \\ 1 & 2.5 & 1 \\ 0.5 & 1 & 2 \end{pmatrix}, \; d = \begin{pmatrix} 8 \\ 3 \\ 3 \end{pmatrix}, \; A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 3 \\ 0 \end{pmatrix}.$$

Inicializace:

Vektor Z vypočítáme z rovnice $AZ = 0$. V této úloze bude tvaru

$$Z = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}.$$

Krok 1:

Sestavíme rovnice

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} x_A = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

$6.5x_Z = 4,$

$$\left(AA^T\right)\lambda = A\left(d - CA^T x_A - CZx_Z\right).$$

Postupným vyřešením jednotlivých rovnic získáme

$$x_A = \begin{pmatrix} 2 \\ -1 \end{pmatrix},\ x_Z = \tfrac{8}{13},\ \lambda = \begin{pmatrix} \frac{9}{26} \\ \frac{25}{26} \end{pmatrix}.$$

Krok 2:

Vypočítáme

$$x^* = A^T x_A + Z x_Z = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \cdot \tfrac{8}{13} = \begin{pmatrix} \frac{34}{13} \\ -\frac{5}{13} \\ \frac{5}{13} \end{pmatrix}$$

Při použití naprogramované funkce ***nulprostor*** *v MATLABU získáme výsledek ve tvaru*

```
[x,mi]=nulprostor([3 1 0.5;1 2.5 1;0.5 1 2],[8;3;3],
[1 0 1;0 1 1],[3;0])


x =

    2.6154
   -0.3846
    0.3846


mi =

    0.3462
    0.9615
```

Na základě uvedených příkladů můžeme říct, že ruční výpočet není příliš složitý pokud nemáme velké množství neznámých a omezujících podmínek. K řešení této problematiky můžeme využít i další metody, jako je například metoda projekce gradientu nebo metoda Schurova komplementu (Range space method).

2.2. Kvadratické programování s omezeními tvaru nerovností

V rámci kvadratického programování s omezeními tvaru nerovností se budeme zabývat úlohami, které nemají žádné rovnostní omezující podmínky. Obecný zápis této úlohy

$$\begin{cases} \text{minimalizovat } q(x) = \frac{1}{2}x^T Cx - d^T x \\ \text{za podmínky} \quad Dx \leq e \end{cases} \tag{2.5}$$

Stejně jako v předchozí části i zde je přípustná množina S definovaná lineárními omezujícími podmínkami a je tedy konvexní. Rozdílem je, že už podmínky řešitelnosti nezávisí na rozměru matice D. V případě, že máme konvexní funkci q, pak mohou nastat tři situace

- bod minima funkce q se bude nacházet mimo přípustnou množinu S a řešení úlohy (2.5) bude některý bod na hranici přípustné množiny,
- bod minima funkce q bude ležet na hranici přípustné množiny a bude totožný s řešením úlohy (2.5),
- bod minima funkce q bude ležet uvnitř přípustné množiny a bude totožný s řešením úlohy (2.5).

U posledních dvou možností se bude shodovat bod minima funkce q s bodem řešení x^* úlohy (2.5). Všechny možnosti mohou nastat i když funkce q bude nekonvexní, ale případ bude komplikovanější, jelikož můžeme mít více lokálních minim nebo můžeme mít přípustnou množinu neohraničenou.
Podmínky řešitelnosti úlohy (2.5) jsou popsány v následující větě

Věta 2.2. *[3, Věta 8.3] Nutnou podmínkou k tomu, aby bod $x^* \in S$ byl řešením dané úlohy (2.5) je, aby existovala čísla $\lambda_i^*, i = 1, \ldots, m$, tak, že platí*

1. $Cx^* + \sum_{i=1}^{m} \lambda_i^* d_i = d,$
2. $\lambda_i^*, \geq 0 \qquad \forall i = 1, \ldots, m,$

3. $\lambda_i^*(d_i^T x^* - b_i) = 0, \qquad \forall i = 1, \ldots, m,$

přičemž vektory $d_i^T, i = 1, \ldots, m$, *představují řádky matice* D.

Jedná se o upravené KKT podmínky a stejně tak platí, pokud funkce q je konvexní, tj. matice C je pozitivně definitní nebo semidefinitní, jsou tyto podmínky nutné i postačující.

Metody pro řešení úloh s omezeními tvaru rovností, byly založeny přímo na řešení KKT soustavy (2.3). V případě úloh s omezeními tvaru nerovností bychom takto mohli postupovat, pokud bychom v bodě řešení x^* věděli, které z podmínek jsou aktivní, pak bychom mohli přejít z původní úlohy (2.5) na úlohu (2.2). Jinak řečeno vynchali bychom všechny omezující podmínky, které nejsou splněny jako aktivní a poloha bodu řešení x^* na ně není nijak citlivá. Nová úloha pak by měla tvar

$$\begin{cases} \text{minimalizovat } q(x) = \frac{1}{2}x^T C x - d^T x \\ \text{za podmínky} \quad d_i^T x = e_i \qquad \forall i \in I(x^*). \end{cases} \tag{2.6}$$

V takovém případě můžeme k řešení úlohy (2.6) využít některou z metod pro řešení kvadratických úloh s omezeními tvaru rovností. Problémem zůstává, že množinu aktivních podmínek $I(x^*)$ v bodě x^* zpravidla neznáme a proto je tento přímočarý postup nepoužitelný. I přes uvedené problémy je možné ideu použít, jak ukážeme na metodě aktivní množiny.

2.2.1. Metoda aktivní množiny

Metoda aktivní množiny je založena na iteračním postupu, kdy se snažíme najít vhodnou kombinaci aktivních podmínek. Jelikož počet omezení je konečný, pak i počet jejich možných kombinací bude konečné číslo, které však exponenciálně roste s rostoucím počtem omezujících podmínek.
Podobně jako je tomu u jiných iteračních metod, i zde bude metoda generovat

posloupnost bodů $\{x^k\}$ tak, že

$$x^{k+1} = x^k + \alpha_k p^k, \qquad k = 0, 1, \ldots,$$

kde α_k je délka kroku a p^k je směr. V rámci iterací budeme zároveň aktualizovat i množinu aktivních podmínek pro daný bod x^k. Pro realizaci takovéhoto postupu se používají tyto dva postupy aktualizace aktivní množiny:

- vyřazování aktivních podmínek
- přidávání aktivních podmínek

Vynechání, neaktivních podmínek však přináší i jisté problémy. Jelikož při výpočtech úloh (2.6), nejsme omezeni všemi podmínkami jako je tomu u původní úlohy (2.5), může nastat situace, že nový bod x^{k+1} nebude spadat do přípustní množiny $x^{k+1} \notin S$. Kvůli tomuto nežádoucímu jevu je třeba zavést tzv. *kvazistacionární body.*

Definice 2.3. *Bod $\widehat{x} \in S$ nazveme kvazistacionárním bodem úlohy (2.5), jestliže je řešením tzv. kvazistacionárního problému:*

$$\begin{cases} \textit{minimalizovat } q(x) = \frac{1}{2}x^T C x - d^T x \\ \textit{za podmínky} \quad d_i^T x = e_i, \qquad \forall i \in I(\widehat{x}) \\ \qquad\qquad\quad\; d_i^T x \leq e_i, \qquad \forall i \notin I(\widehat{x}) \end{cases}$$

Stejně jako tomu bylo u Metody nulého prostoru, ani zde nebudeme hledat přímo bod x^{k+1}, ale pomocnou úlohu převedeme na úlohu pro výpočet směru p s použitím partikulárního řešení a postupu z předchozí části.

$$\begin{cases} \text{minimalizovat } \tilde{q}(p) = \frac{1}{2}p^T C p + (g^k)^T p \\ \text{za podmínky} \quad d_i^T p = 0, \qquad \forall i \in I_k \end{cases}$$

Kde množina I_k obsahuje všechny indexy omezujících podmínek, které jsou pro daný bod x^k aktivní. Vektor g^k je gradient funkce q v bodě x^k. Vyřešením této úlohy získáme směr p^k a následně položíme $\widehat{x} = x^k + p^k$, který je třeba otestovat, jelikož nemusí být akceptovatelný. Existují pouze dvě možnosti jak budeme dále pracovat s bodem $\widehat{x}$ a to

- Pokud $\widehat{x} \in S$, jedná o kvazistacionární bod, a položíme $x^{k+1} = \widehat{x}$
- Pokud $\widehat{x} \notin S$, nejedná se o kvazistacionární bod, a není tedy akceptovatelný.

V případě druhé situace, kdy získáme neakceptovatelný bod $\widehat{x}$, je třeba najít poslední akceptovatelný bod ve směru p^k, jenž zpravidla bude ležet na hranici některé z omezujících podmínek, která není obsažena v množině I_k. Jinak řečeno je třeba určit maximální přípustný krok α_k. Tento krok nám zabrání v opuštění přípustné množiny S. V případě první situace je tento krok roven $\alpha_k = 1$.
Pro všechna omezení, která nepatří do množiny aktivních podmínek vypočítáme hodnotu kroku pro kterou se daná podmínka stane aktivní. Z těchto hodnot vybereme tu nejmenší, pokud bude platit $\alpha_k < 1$, bude se jednat o tzv. blokující omezení a zavedeme toto omezení do aktivní množiny.

$$\widehat{\alpha_k} = \min_{i \notin \widehat{I}_k, d_i^T p^k > 0} \frac{e_i - d_i^T x^k}{d_i^T p^k}$$

Logicky nás bude zajímat maximální přípustný krok pouze pro omezující podmínky, které nepatří do množiny aktivních podmínek. Zároveň pro podmínky, kde platí $d_i^T p^k \leq 0$ také není třeba řešit hodnotu kroku, jelikož tyto podmínky nový bod x^{k+1} neporuší. To zda výsledný bod x^{k+1} bude řešením úlohy (2.5) budeme rozhodovat podle vektoru multiplikátorů $\widehat{\lambda}$. V případě nezápornosti všech složek tohoto vektoru bude možné x^{k+1} označit jako řešení úlohy (2.5). V opačném případě je nutné vyřadit podmínku, jejíž index odpovídá indexu záporné hodnoty vektoru $\widehat{\lambda}$. V případě že máme více kandidátů na vyřazení je doporučeno vyřadit ten s nejnižší hodnotou. Účelem je udržet aktivní množinu co možná nejmenší, což je důvod proč je doporučeno zařazovat nebo vyřazovat omezující podmínky z aktivní množiny vždy po jedné.
Problémem metody může být vetší počet omezení, která musíme procházet. Komplikace způsobí i když budeme v úloze mít lineárně závislé gradienty aktivních podmínek. Proto se zavádí pracovní množina, která narozdíl od aktivní množiny, problém lineárně závislých gradientů eliminujeme.

Definice 2.4. *Pracovní množinou $W_k = W(x^k)$ příslušející bodu x^k budeme rozumět takovou podmnožinu aktivní množiny $I_k = I(x^k)$, že všechny gradienty omezujících podmínek určených touto množinou jsou lineárně nezávislé.*

Pro určení aktivní množiny pro daný bod jsem v MATLABU naprogramoval funkci **UrceniAktMnoziny**, kde vstupem je bod $x0$ pro který hledáme aktivní množinu a matice D a vektor e, představující omezující podmínky. Výstupem jsou matice $W1$ a vektor $w1$, složené z řádků matice A a vektoru b, které odpovídají indexům podmínek, které jsou v daném bodě splněny jako rovnostní. Matice $W0$ a vektor $w0$ obsahují zbývající řádky odpovídající podmínkám, které nejsou splněny jako rovnostní.

```
function [W0,w0,W1,w1]=UrceniAktMnoziny(x0,D,e)
%W1,w1 - aktivní matice a vektor
%W0,w0 - pasivni matice a vektor
aktivni_podm=find(D*x0-e==0);
W0=D;
w0=e;
if isempty(aktivni_podm)
    W1=[];
    w1=[];
else
    W1=D(aktivni_podm(1),:);
    w1=e(aktivni_podm(1));
    W0(aktivni_podm,:)=[];
    w0(aktivni_podm,:)=[];
end
end
```

V případě, že bychom řešili úlohu s rovnostními i nerovnostními podmínkami nebude se postup řešení nijak lišit. Všechny omezující podmínky tvaru rovností budou součástí aktivní (pracovní) množiny a nebudou z ní vyřazovány.

Algoritmus 2.2. *[1, str.472][3, str.84]*

Inicializace: Určíme vhodný přípustný startovní bod $x^0 \in S$*, stanovíme aktivní množinu* $I(x^0)$ *a pracovní množinu* W_0 *a vypočítáme gradient ve startovním bodě* $g^0 = Cx^0 - d$*.*

HLAVNÍ CYKLUS:

pro $k = 0, 1, 2, \ldots$

určíme p^k *vyřešením úlohy*

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T C p + (g^k)^T p \\ \text{za podmínky} \quad a_i^T p = 0, \qquad \forall i \in W_k, \end{cases}$$

kde a_i *představují řádky matice* A *odpovídající indexům podmínek, které jsou zařazeny v pracovní množině* W_k*.*

- *POKUD* $p^k = 0$

  vypočítáme multiplikátory λ_i^k *ze vztahu*

  $$\sum_{i \in W_k} a_i \lambda_i^k = -Cx^k + d$$

  následně otestujeme jednotlivé multiplikátory

  - *pokud* $\lambda_i^k \geq 0 \quad \forall i \in W_k$*, potom položíme* $\lambda_i^k = 0 \quad \forall i \notin W_k$*,* $x^* = x^k$*,* $\lambda^* = \lambda^k$ *a ukončíme algoritmus*
  - *v opačném případě určíme* $\lambda_l = \min\limits_{i \in W_k} \left\{\lambda_i^k\right\}$*, položíme* $x^{k+1} = x^k$*,* $W_{k+1} = W_k - \{l\}$ *a vrátíme se na začátek hlavního cyklu*

- *JINAK* $\left(p^k \neq 0\right)$

  vypočítáme maximální přípustný krok ze vztahu

  $$\widehat{\alpha_k} = \min_{i \notin W_k, a_i^T p^k > 0} \frac{b_i - a_i^T x^k}{a_i^T p^k}$$

  určíme $\alpha_k = min\left\{1, \widehat{\alpha_k}\right\}$ *a vypočítáme* $x^{k+1} = x^k + \alpha_k p^k$*,* $g^{k+1} = Cx^{k+1} - d$

  následně krok α_k *otestujeme*

– *pokud* $\widehat{\alpha_k}$*, určíme index* r *takový, že* $a_r^T(x^k + \alpha_k p^k) = b_r$ *a položíme* $W_{k+1} = W_k \cup \{r\}$

– *jinak* $(\alpha_k = 1)$*, položíme* $W_{k+1} = W_k$

vrátíme se na začátek hlavního cyklu

Na vyřešení podúlohy, která se objevuje v algoritmu metody Aktivní množiny je třeba využít některou z metod kvadratického programování s omezeními tvaru rovností. V tomto textu budeme využívat konkrétně metodu nulového prostoru.

Pro použití metody aktivní množiny, jsem naprogramoval v MATLABU funkci **MetodaAktMnoziny**, kde vstupem je daný bod $x0$, matice A a C, vektory b a d. Výstupem je bod řešení x^* a odpovídající hodnota multiplikátorů λ^*.

```
function [x,lambda,mi]=MetodaAktMnoziny(x0,C,d,D,e,A,b)
%   f(x)=1/2*x'*C*x-d'x
%       D*x<=e
%       A*x=b

%Vstupy
%x0-startovací bod
%A-matice odpovídající omezujícím podmínkám
%b-vektor odpovídající omezujícím podmínkám
%C-matice definující minimalizovanou funkci
%d-vektor definující minimalizovanou funkci

%Výstupy
%x-bod řešení dané úlohy
%lambda-odpovídající hodnoty multiplikátorů

%určení, zda úloha má rovnostní podmínky
if (nargin==5)
```

```
A=[];
b=[];
end

%výpočet gradientu pro bod x0
g0=C*x0-d;

%určení aktivní množiny pro vektor x0
[Pmatice,Pvektor,Amatice,Avektor]=UrceniAktMnoziny(x0,D,e);

%cyklus funkce
while 1
%vypočítání pomocné úlohy pomocí metody nulového prostoru
[p,~]=nulprostor(C,-g0,[A;Amatice],zeros(length([b;Avektor]),1));

%testování směru p, zda je nulový
p(abs(p)<1e-10)=0;
if any(p)==1
       i=length(Pvektor);
       %výpočet možných kroků
       if isempty(Pvektor)
           alfa_k=[];
       else
            alfa_k=zeros(i,1);
            for j=1:i

            if (Pmatice(j,:)*p)>0
            alfa_k(j,1)=(Pvektor(j)-Pmatice(j,:)*x0)/(Pmatice(j,:)*p);
            else
            alfa_k(j,1)=Inf;
```

```
            end
            end
        end

    if isempty(alfa_k)
        alfa=1;
    else
        alfa=min(alfa_k);
    end
    %určení maximálního přípustného kroku
    alfa=min(1,alfa);
    %výpočet nového bodu a gradientu v tomto bodě
    x0=x0+alfa.*p;
    g0=C*x0-d;
    %určené blokující podmínky
    if alfa<1
        R=zeros(i,1);
        for k=1:i
            R(k)=Pmatice(k,:)*(x0)-Pvektor(k);
        end
        %zavedení blokující podmínky
        i_r=find(abs(R)<1e-10);
        Amatice=[Amatice;Pmatice(i_r(1),:)];
        Avektor=[Avektor;Pvektor(i_r(1))];
        Pmatice(i_r(1),:)=[];
        Pvektor(i_r(1))=[];
    end
else
    %výpočet multiplikátoruů
    if isempty([A;Amatice])
```

```
        multiplikatory=-C*x0+d;
    else
        multiplikatory=linsolve([A;Amatice].',-C*x0+d);
    end
    mi=multiplikatory(1:length(b));
    lambda=multiplikatory(length(b)+1:length(multiplikatory));
    if isempty(lambda)
        lambda=0;
    end
    %testování multiplikátorů
    if lambda>=0
        %dopočítání zbylých multiplikátorů určení řešení
        %a ukončení algoritmu

        rozdil_delky=length(e)-length(lambda);
        lambda=[lambda;zeros(rozdil_delky,1)];
        x=x0;
    break
    else
        %určení a odstranění blokující podmínky
        i_min=find(lambda==min(lambda));
        Pmatice=[Pmatice;Amatice(i_min(1),:)];
        Pvektor=[Pvektor;Avektor(i_min(1))];
        Amatice(i_min(1),:)=[];
        Avektor(i_min(1))=[];
    end
end
end
end
```

Příklad 2.3. *[5] Máme úlohu*

$$\begin{cases} \textit{minimalizovat } x_1^2 + x_2^2 - 5x_1 - 7x_2 + 5 \\ \textit{za podmínky} \quad 4x_1 + x_2 \leq 20 \\ \qquad\qquad\quad\; x_1 + 4x_2 \leq 20 \\ \qquad\qquad\quad\; x_1 \geq 0,\, x_2 \geq 0 \end{cases}$$

Jako počáteční bod budeme volit $x^0 = (0,\, 0)^T$.

Nejprve si určíme pracovní množinu $W_0 = \{3, 4\}$ *a dopočítáme gradient v počátečním bodě*

$$g^0 = Cx^0 + d = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 5 \\ 7 \end{pmatrix} = \begin{pmatrix} -5 \\ -7 \end{pmatrix}.$$

Dalším krokem je vyřešení pomocné úlohy pro výpočet směru p. *Pomocná úloha bude mít tvar*

$$\begin{cases} \textit{minimalizovat } \frac{1}{2} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}^T \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} -5 \\ -7 \end{pmatrix}^T \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \\ \textit{za podmínky} \quad p_1 = 0 \\ \qquad\qquad\quad\; p_2 = 0 \end{cases}$$

Tuto úlohu bychom řešili metodou kvadratického programování pro omezení tvaru rovnosti, v našem případě metodou nulového prostoru, ale omezující podmínky nám přímo vymezují řešení. Výsledný směr $p = (0,\, 0)^T$. *Jelikož je směr nulový vypočítáme odpovídající hodnoty multiplikátorů z rovnice*

$$\sum_{i \in W_0} a_i \lambda_i^0 = -Cx^0 + d$$

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}^T \begin{pmatrix} \lambda_3 \\ \lambda_4 \end{pmatrix} = - \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

Vyřešením rovnice získáme multiplikátory

$$\begin{pmatrix} \lambda_3 \\ \lambda_4 \end{pmatrix} = \begin{pmatrix} -5 \\ -7 \end{pmatrix}$$

Jelikož minimálně jedna hodnota multiplikátoru je záporná, musíme vyřadit jednu z podmínek z pracovní množiny. Multiplikátor odpovídající 4. podmínce je nejmenší,

z tohoto důvodu vyřadíme 4. podmínku $W_1 = W_0 - \{4\}$. *Bod* $x^1 = x^0$ *a gradient* $g^1 = g^0$.

Vrátíme se zpět k výpočtu směru p. Pomocná úloha nyní bude mít tvar

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}^T \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} -5 \\ -7 \end{pmatrix}^T \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \\ \textit{za podmínky} \quad p_1^1 = 0 \end{cases}$$

Vyřešením získáme směr $p = (0,\, 3.5)^T$. *Směr p je nenulový, proto musíme vypočítat maximální přípustný krok, abychom zabránili vystoupení z přípustné množiny S původní úlohy. Maximální přípustný krok vypočítáme ze vzorce*

$$\widehat{\alpha_1} = \underset{i \notin W_1,\, a_i^T p^1 > 0}{min} \frac{b_i - a_i^T x^1}{a_i^T p^1},$$

$$\widehat{\alpha_1} = 1.4286$$

Následně otestujeme, zda nám některá z podmínek neblokuje postup $\alpha_1 = min\,\{1, \widehat{\alpha_k}\}$. *Konečná hodnota kroku bude* $\alpha_1 = 1$, *což znamená, že žádná z omezujících podmínek nás neblokuje a není tedy třeba ji zavádět do pracovní množiny. Nový bod bude* $x^2 = x^1 + \alpha_1 p^1 = (0,\, 3.5)^T$ *a pracovní množina* $W_2 = W_1$. *Gradient pro nový bod je třeba vypočítat*

$$g^2 = Cx^2 + d = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 3.5 \end{pmatrix} - \begin{pmatrix} 5 \\ 7 \end{pmatrix} = \begin{pmatrix} -5 \\ 0 \end{pmatrix}.$$

Znovu sestavíme pomocnou úlohu

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}^T \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} -5 \\ 0 \end{pmatrix}^T \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \\ \textit{za podmínky} \quad p_1^2 = 0 \end{cases}$$

Řešením této úlohy bude směr $p = (0,\, 0)^T$. *Dopočítáme multiplikátor odpovídající omezující podmínce*

$$\begin{pmatrix} -1 & 0 \end{pmatrix}^T \lambda_3 = -\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

$$\lambda_3 = -5$$

Jelikož $\lambda_3 < 0$ *vyřadíme z pracovní množiny 3. podmínku. A nová pracovní množina bude* $W_3 = W_2 - \{3\}$. *Bod* $x^3 = x^2$ *a gradient* $g^3 = g^2$.
Sestavíme pomocnou úlohu pro směr p *ve tvaru*

$$\left\{ \textit{minimalizovat } \tfrac{1}{2} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}^T \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} -5 \\ 0 \end{pmatrix}^T \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \right.$$

Vyřešením dostaneme směr $p^3 = (2.5,\, 0)^T$. *Dopočítáme maximální přípustný krok* $\widehat{\alpha_3} = 1.65$ *a vybereme hodnotu která splňuje* $\alpha_3 = min\,\{1, \widehat{\alpha_3}\}$. *Výsledná hodnota kroku bude tedy* $\alpha_3 = 1$. *Nový bod* $x^4 = x^3 + \alpha_3 p^3 = (2.5,\, 3.5)^T$ *a pracovní množina* $W_4 = W_3 = \{\quad\}$. *Gradient v bodě* x^3 *bude ve tvaru*

$$g^3 = Cx^3 + d = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 2.5 \\ 3.5 \end{pmatrix} - \begin{pmatrix} 5 \\ 7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Sestavíme pomocnou úlohu pro výpočet směru p^4

$$\left\{ \textit{minimalizovat } \tfrac{1}{2} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}^T \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \right.$$

Vyřešením této úlohy dostaneme směr $p = (0,\, 0)^T$. *Jelikož pracovní množina* $W_4 = \{\quad\}$, *všechny multiplikátory odpovídající omezujícím podmínkám, které nepatří do pracovní množiny budou nulové. A řešení úlohy je tedy*

$$x^* = \begin{pmatrix} 2.5 \\ 3.5 \end{pmatrix}, \qquad \lambda^* = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

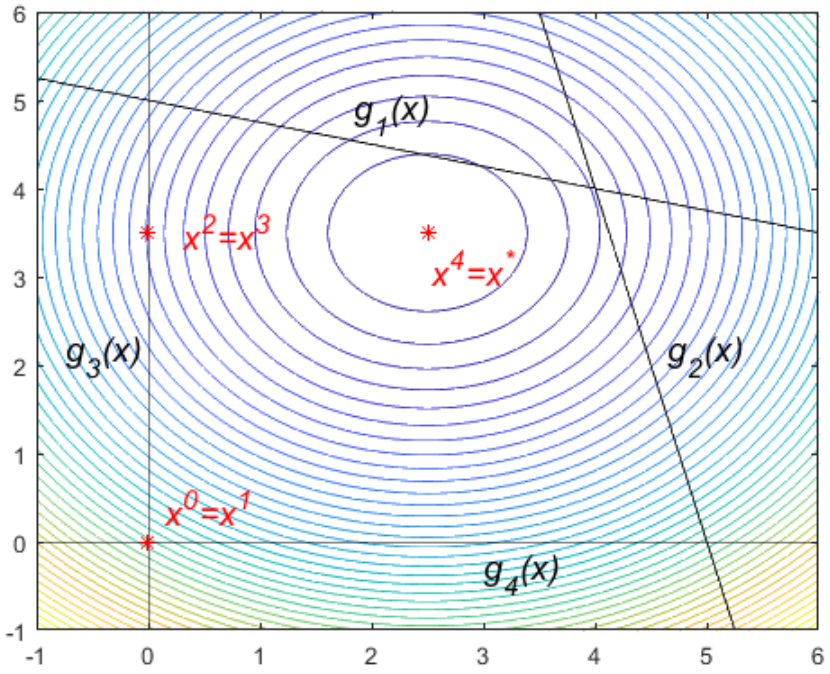
Grafické znázornění je na obrázku 2.2, kde jsou vykresleny jednotlivé iterace metody, a souhrn celého postupu je uveden v tabulce 2.1. Jak názorně můžeme vidět pro každou iteraci máme uvedený aktuální bod x^k, *jemu odpovídající vhodnou pracovní množinu, hodnoty multiplikátorů* λ_i *pro všechny omezující podmínky, vypočítaný směr* p^k *a případně hodnotu maximálně přípustného kroku* α_k, *pokud je potřebná.*

k	x^k	W_k	λ	p^k	α_k
0	(0, 0)	$\{3, 4\}$	$\lambda_1 = 0$ $\lambda_2 = 0$ $\lambda_3 = -5$ $\lambda_4 = -7$	(0, 0)	–
1	(0, 0)	$\{3\}$	$\lambda_1 = 0$ $\lambda_2 = 0$ $\lambda_3 = -5$ $\lambda_4 = 0$	(0, 3.5)	1
2	(0, 3.5)	$\{3\}$	$\lambda_1 = 0$ $\lambda_2 = 0$ $\lambda_3 = -5$ $\lambda_4 = 0$	(0, 0)	–
3	(0, 3.5)	$\{\}$	$\lambda_1 = 0$ $\lambda_2 = 0$ $\lambda_3 = 0$ $\lambda_4 = 0$	(2.5, 0)	1
4	(2.5, 3.5)	$\{\}$	$\lambda_1 = 0$ $\lambda_2 = 0$ $\lambda_3 = 0$ $\lambda_4 = 0$	(0, 0)	–

Tabulka 2.1: Postup metody aktivní množiny pro příklad 2.3

Při použití naprogramované funkce ***MetodaAktMnoziny*** *v MATLABU získáme výsledek ve tvaru*

```
[x,lambda]=MetodaAktMnoziny([0;0],[2 0;0 2],[5;7],[4 1;1 4;-1 0;0 -1],
[20;20;0;0])
x =
    2.5000
    3.5000
lambda =
     0
     0
     0
     0
```



Obrázek 2.2: Grafické znázornění Příkladu 2.3

Na následujícím případě ukážeme využití naprogramované funkce **MetodaAktMnoziny** na úloze s oběmi typy omezujících podmínek.

Příklad 2.4. *[7] Máme úlohu*

$$\begin{cases} \textit{minimalizovat } x_1^2 - x_2^2 \\ \textit{za podmínky} \quad x_1 - x_2 = 3 \\ \qquad\qquad\quad x_2 \geq 3. \end{cases}$$

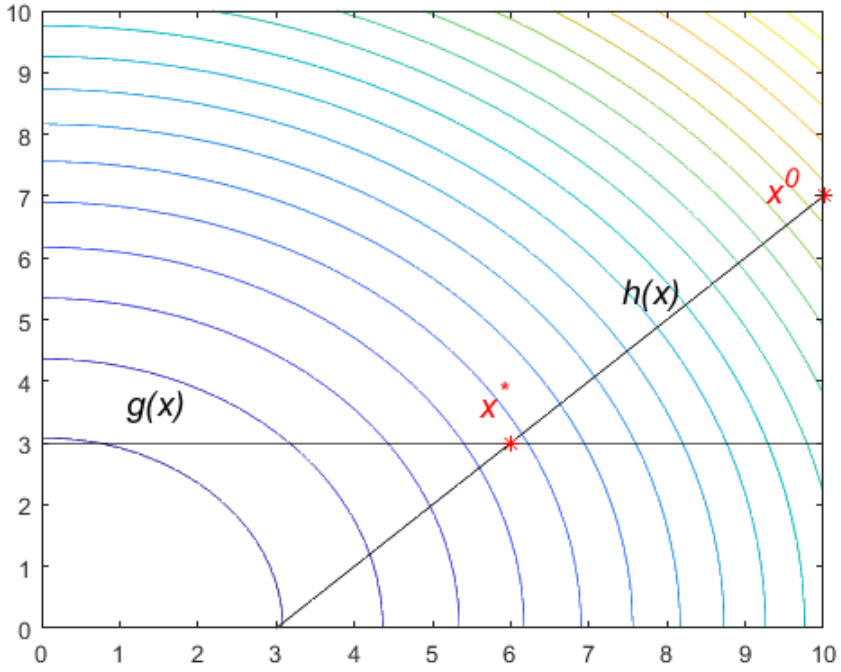
Nejdříve upravíme úlohu do tvaru (2.1)*, kde*

$$C = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \; d = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \; A = \begin{pmatrix} 1 & -1 \end{pmatrix}, \; b = 3, \; D = \begin{pmatrix} 0 & -1 \end{pmatrix}, \; e = -3.$$

Při použití naprogramované funkce ***MetodaAktMnoziny*** *v MATLABU získáme výsledek ve tvaru*

```
[x,lambda,mi]=MetodaAktMnoziny([10;7],[2 0;0 2],[0;0],[0 -1],[-3],
[1 -1],[3])
x =
    6.0000
    3.0000

lambda =
    18

mi =
  -12.0000
```

Tento výsledek nám potvrdí i obrázek 2.3, na kterém je úloha znázorněna. Jedná se pouze o ilustrační příklad, možnosti použití metody aktivní množiny pro úlohy kvadratického programování s oběmi typy omezujících podmínek.



Obrázek 2.3: Grafické znázornění Příkladu 2.4

Kapitola 3

Sekvencionální kvadratické programování - SQP

Základní ideou metody sekvencionálního kvadratického programování(SQP) je vyřešení úlohy nelineárního programování (1.1) za pomoci posloupnosti úloh kvadratického programování (2.1). Jedná se o jednu z nejúčinějších metod nelineárního programování, její největší předností je schopnost vypořádat se i s nelineárními omezeními. Tato kapitola je zaměřena na úlohy s omezeními ve tvaru rovností a na úlohy s omezeními ve tvaru rovností i nerovností. Úlohu pouze s nerovnostními omezeními v tomto textu nebudeme uvažovat, jelikož pro tento typ úloh jsou vhodnější jiné metody, například metoda vnitřních bodů. V praxi většinou není možné zaručit úlohy pouze s rovnostními omezeními, nejčastěji narazíme na úlohy se smíšenými omezujícími podmínkami, v takovém případě je pak na uživateli zda pro řešení takové úlohy využije SQP a nebo jinou metodu. Hlavním zdrojem pro tuto část je [1] a dále [3] a [4].

3.1. Úlohy nelineárního programování s rovnostními omezeními

V této kapitole se budeme věnovat úlohám nelineárního programování pouze s omezeními ve tvaru rovnosti, pro které je metoda nejvhodnější a proto je na nich odvozen i základní princip metody.

Uvažujme úlohu

$$\begin{cases} \text{minimalizovat } f(x) \\ \text{za podmínky } \quad h_i(x) = 0, \qquad i = 1, \ldots, m, \end{cases} \tag{3.1}$$

kde f je nelineární funkce o n proměnných, $x \in X \subseteq \mathbb{R}^n$ a $h_i(x) = 0$ je i-tá omezující podmínka tvaru rovnosti. Omezující podmínky se budou objevovat i ve vektorové formě

$$h(x) = 0.$$

Principem metody SQP je konstrukce posloupnosti $\{x^k\}$, která bude konvergovat k řešení úlohy (3.1). V každé iteraci budeme řešit úlohu kvadratického programování, jejím vyřešením získáme krok p, který je nutný k výpočtu nového bodu posloupnosti $x^{k+1} = x^k + p$. Úkolem je navrhnout pomocné úlohy kvadratického programování takovým způsobem, aby metoda měla dobré konvergenční vlastnosti. Podmínky za kterých metoda konverguje udává Věta 5.1 v poslední kapitole.

Pro odvození pomocné úlohy, máme k dispozici dva přístupy. Každý přístup je založen na jiné myšlence, první přístup využívá přímé aproximace a druhý přístup Newtonovu metodu pro řešení soustavy nelineárních rovnic.

1. přístup

První přístup je založen na přímé aproximaci funkce $f(x)$ v bodě $x^{k+1} = x^k + p$ Taylorovým polynomem druhého řádu

$$f(x^k + p) \doteq f(x^k) + \nabla f(x^k)^T p + \frac{1}{2} p^T \nabla^2 f(x^k) p$$

a omezujících podmínek Taylorovým polynomem prvního řádu v bodě x^{k+1}

$$h(x^k + p) \doteq h(x^k) + \nabla h(x^k) p.$$

Omezující podmínky v bodě x^{k+1} aproximujeme přímkou, to znamená, že v tomto bodě by se měli obě podmínky shodovat, proto můžeme říct

$$h(x^k) + \nabla h(x^k) p = 0.$$

V tuto chvíli již můžeme sestavit pomocnou úlohu, bude se jednat o úlohu kvadratického programování s omezeními tvaru rovností:

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2 f(x^k)p + \nabla f(x^k)^T p + f(x^k) \\ \text{za podmínky} \quad \nabla h(x^k)p + h(x^k) = 0. \end{cases}$$

Vyřešením této podúlohy za pomoci vhodné metody kvadratického programování bychom získali krok p, s jehož pomocí vypočítáme nový bod $x^{k+1} = x^k + p$ a pokračovali bychom v iteračním postupu opětovným odvozením dílčí podúlohy. Tento postup je sice jednoduchý a logický, ale problém může být, že podúlohy nemusí být řešitelné, jelikož nemusí být zdola ohraničeny. Což znamená, že nebude možné nalézt minimum funkce $f(x)$ na přípustné množině S. Tento problém bude demonstrován v následujícím příkladě.

Příklad 3.1. *[5] Máme úlohu*

$$\begin{cases} \textit{minimalizovat } (1-x_1)^2 - x_2^2 \\ \textit{za podmínky} \quad 10(x_2^2 - x_1) = 0 \end{cases}$$

Jako vstupní bod pro tento příklad použijeme $x^0 = (0,0)^T$. *Vypočítáme jednotlivé vektory a matice, potřebné pro pomocnou úlohu*

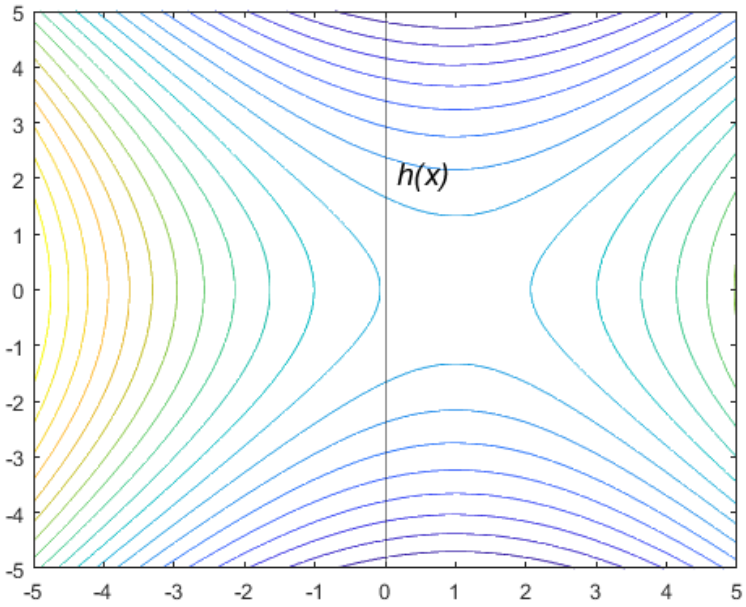
$$\nabla f(x) = \begin{pmatrix} -2(1-x_1) \\ -2x_2 \end{pmatrix}, \nabla^2 f(x) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}, \nabla h(x) = \begin{pmatrix} -10 \\ 20x_2 \end{pmatrix}.$$

$$\nabla f(x^0) = \begin{pmatrix} -2 \\ 0 \end{pmatrix}, \nabla^2 f(x^0) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}, \nabla h(x^0) = \begin{pmatrix} -10 \\ 0 \end{pmatrix}, h(x^0) = 0$$

Nyní můžeme sestavit pomocnou úlohu která bude mít v bodě x^0 *tvar*

$$\begin{cases} \textit{minimalizovat } p_1^2 - p_2^2 - 2p_1 + 1 \\ \textit{za podmínky} \quad -10p_2 = 0. \end{cases} \tag{3.2}$$

Úloha v tomto tvaru nemá řešení, jelikož je neohraničená. Tato úloha je znázorněna na obrázku 3.1.



Obrázek 3.1: Úloha (3.2)

Problém neohraničenosti zdola je možné řešit pomocí Lagrangeovy funkce a lagrangeových multiplikátorů. Vektor μ^* bude označovat hodnotu lagrangeových multiplikátorů, které přísluší k řešení x^* úlohy (3.1).
Upravená úloha, pak bude mít tvar

$$\begin{cases} \text{minimalizovat } L(x, \mu^*) = f(x) + (\mu^*)^T h(x) \\ \text{za podmínky} \quad h(x) = 0 \end{cases} \tag{3.3}$$

a její řešení x^* je totožné s původní úlohou (3.1).
Vzniká nový problém a tím je, že zpravidla neznáme optimální hodnotu multiplikátorů μ^*, které budeme vhodně aproximovat v k-té iteraci hodnotou μ^k. V tomto bodě využijeme stejný postup jako u původní úlohy (3.1). V bodě (x^k, μ^k) pak nahradíme Lagrangeovu funkci Taylorovým polynomem druhého řádu

$$L(x^k + p, \mu^k) \doteq \frac{1}{2} p^T \nabla^2_{xx} L(x^k, \mu^k) p + \nabla_x L(x^k, \mu^k)^T p + L(x^k, \mu^k).$$

Aproximace omezující podmínky bude mít stejný tvar. Po těchto úpravách můžeme

namodelovat pomocnou úlohu ve tvaru

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\mu^k)^Tp + \nabla_x L(x^k,\mu^k)^T p + L(x^k,\mu^k) \\ \text{za podmínky} \quad \nabla h(x^k)p + h(x^k) = 0. \end{cases} \tag{3.4}$$

Na následujícím příkladě bude demonstrováno, že použití Lagrangeovy funkce vyřešilo problém s neohraničeností.

Příklad 3.2. *Vezmeme znovu předchozí zadání příkladu 3.1*

$$\begin{cases} \textit{minimalizovat } (1-x_1)^2 - x_2^2 \\ \textit{za podmínky} \quad 10(x_2 - x_1^2) = 0, \end{cases}$$

a použijeme zde upravenou verzi dílčí úlohy. Pro její odvození potřebujeme vypočítat Lagrangeovu funkci a odpovídající gradient a hessián.

$$L(x^k,\mu^k) = (1-x_1^k)^2 - (x_2^k)^2 + 10\mu((x_2^k)^2 - x_1^k)$$

$$\nabla_x L(x^k,\mu^k) = \begin{pmatrix} -2 + 2x_1^k - 10\mu \\ -2x_2^k + \mu 20 x_2 \end{pmatrix}$$

$$\nabla^2_{xx}L(x^k,\mu^k) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} + \mu \begin{pmatrix} 0 & 0 \\ 0 & 20 \end{pmatrix}$$

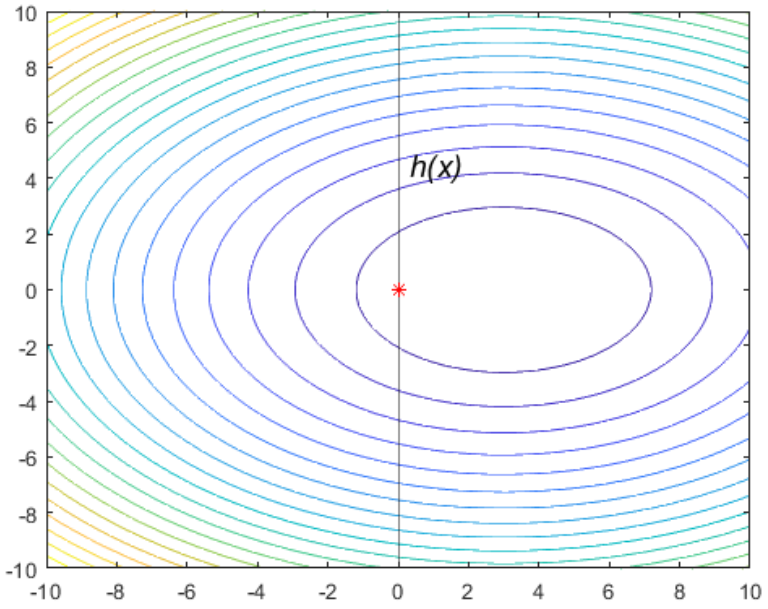
Hodnota multiplikátoru bude $\mu^0 = \frac{1}{2}$ *a počáteční bod* $x^0 = (0,\,0)^T$ *a jejich dosazením dostaneme*

$$L(x^0,\mu^0) = 1, \nabla_x L(x^0,\mu^0) = \begin{pmatrix} -12 \\ 0 \end{pmatrix}, \nabla^2_{xx}L(x^0,\mu^0) = \begin{pmatrix} 2 & 0 \\ 0 & 8 \end{pmatrix}$$

Nyní můžeme sestavit pomocnou úlohu, která bude mít v bodě x^0 *tvar*

$$\begin{cases} \textit{minimalizovat } p_1^2 + 4p_2^2 - 12p_1 + 1 \\ \textit{za podmínky} \quad -10p_1 = 0. \end{cases} \tag{3.5}$$

Vypočítáme úlohu, například pomocí metody Nulového prostoru a získáme řešení $p = (0,\,0)^T$. *Pomocná úloha* (3.5) *je znázorněna na obrázku 3.2.*



Obrázek 3.2: Úloha (3.5)

2. přístup

Druhý přístup je odvozen z použití Newtonovy metody na podmínky optimality 1. řádu pro řešenou úlohu. Princip Newtonovy metody pro soustavu nelineárnách rovnic je popsán např. v [4]. Stručně řečeno Newtonova metoda slouží k nalezení řešení soustavy rovnic $f_i(x) = 0$, $i = 1, \ldots, n$, za předpokladu, že známe derivace $f_i'(x)$. Soustava řešených rovnic se souhrně zapisuje jako $F(x) = 0$, kde

$$F(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix}.$$

Jednotlivé derivace $f_i'(x)$ dle všech proměnných definují Jakobiho matici $J_F(x)$. V tuto chvíli je možné sestavit rovnici k výpočtu řešení, Newtonovy soustavy

$$J_F(x)y = -F(x). \tag{3.6}$$

Jedná se o iterační postup ve kterém je třeba zadat počáteční aproximaci x^0. A pro nový bod bude platit $x^{k+1} = x^k + y$. Tento postup využijeme pří odvozování

pomocné úlohy pro metodu SQP. Podmínky optimality 1.řádu mají pro naši úlohu (3.1) tvar

$$\nabla_x L(x,\mu) = \nabla f(x) + \nabla h(x)^T \mu = 0$$

$$\nabla_\mu L(x,\mu) = h(x) = 0.$$

Definujeme vektorovou funkci $F(x,\mu)$

$$F(x,\mu) = \begin{pmatrix} \nabla f(x) + \nabla h(x)^T \mu \\ h(x) \end{pmatrix}$$

a budeme hledat nulové body této vektorové funkce, tj. řešíme soustavu nelineárních rovnic $F(x,\mu) = 0$

$$\begin{gathered} \nabla f(x) + \nabla h(x)^T \mu = 0 \\ h(x) = 0 \end{gathered}$$

Jacobiho matice pro funkci $F(x,\mu)$, pak bude mít tvar

$$J_F(x,\mu) = \begin{pmatrix} \nabla^2_{xx} L(x,\mu) & \nabla h(x)^T \\ \nabla h(x) & 0 \end{pmatrix}.$$

Pro nastartování metody je nutné zadat počáteční aproximace (x^0, μ^0). Nacházíme-li se v k-té iteraci, neboli v k-té podúloze metody Sekvencionálního kvadratického programování, máme bod (x^k, μ^k) jehož dosazením do Newtonovy soustavy (3.6) získáme

$$J_F(x^k,\mu^k) \begin{pmatrix} p^k \\ \triangle\mu^k \end{pmatrix} = -F(x^k,\mu^k). \tag{3.7}$$

Vyřešením soustavy získáme $(p^k, \triangle\mu^k)$. A pomocí následujícího předpisu vypočítáme body pro novou iteraci

$$\begin{pmatrix} x^{k+1} \\ \mu^{k+1} \end{pmatrix} = \begin{pmatrix} x^k + p^k \\ \mu^k + \triangle\mu^k \end{pmatrix}.$$

Budou-li splněny podmínky

(p1) $\nabla h(x^k)$ má plnou hodnost

(p2) $s^T \nabla^2_{xx} L(x^k,\mu^k) s > 0 \ \forall s : \nabla h(x^k) s = 0, s \neq 0,$

pak matice Newtonovy soustavy bude regulární. Takové podmínky budou splněny, pokud bude bod (x^k, μ^k) dostatečně blízko bodu (x^*, μ^*), v kterém platí podmínky optimality 2.řádu.
Nyní podíváme-li se na pomocnou úlohu odvozenou v prvním přístupu (3.4), která v *k-té* iteraci bude

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\mu^k)^Tp + \nabla_x L(x^k,\mu^k)^Tp + L(x^k,\mu^k) \\ \text{za podmínky} \quad \nabla h(x^k)p + h(x^k) = 0. \end{cases}$$

Pokud jsou splněny předpoklady (p1) a (p2) bude mít tato úloha jednoznačné řešení $(p_k, \Delta\mu_k)$, jenž bude splňovat KKT podmínky

$$\begin{aligned} \nabla^2_{xx}L(x^k,\mu^k)p^k + \nabla_x L(x^k,\mu^k) + \nabla h(x^k)^T\Delta\mu_k = 0, \\ \nabla h(x^k)p^k + h(x^k) = 0. \end{aligned}$$

Tuto soustavu zapíšeme do maticového tvaru

$$\begin{pmatrix} \nabla^2_{xx}L(x^k,\mu^k) & \nabla h(x^k)^T \\ \nabla h(x^k) & 0 \end{pmatrix} \begin{pmatrix} p^k \\ \Delta\mu_k \end{pmatrix} = - \begin{pmatrix} \nabla f_k + \nabla h(x^k)^T\mu_k \\ h(x^k) \end{pmatrix}.$$

Porovnáme-li tento maticový zápis s Newtonovou soustavou (3.7) můžeme říct, pokud jsou splněny předpoklady (p1) a (p2) potom řešení $(p^k, \Delta\mu_k)$ je pro obě soustavy totožné a pomocná úloha pro 2. přístup v bodě x^k bude mít tvar

$$\begin{cases} \text{minimalizovat } q_k(p) \\ \text{za podmínky} \quad \nabla h(x^k)p + h(x^k) = 0 \end{cases} \tag{3.8}$$

kde funkce $q_k(p)$ bude mít tvar

$$q_k(p) = \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\mu^k)p + \nabla_x L(x^k,\mu^k)^Tp + L(x^k,\mu^k).$$

Předpokládáme, že jsou splněny podmínky (p1) a (p2) na matici Newtonovy soustavy, bude úloha (3.8) konvexní, jelikož $\nabla^2_{xx}L(x^k,\mu^k)$ jakožto hessián funkce $q(x)$ je pozitivně definitní na kritickém kuželu, a bude se tedy jednat o úlohu konvexního kvadratického programování.

Dílčí úlohy (3.4) a (3.8) budeme zkráceně označovat jako QP1.

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\mu^k)p + \nabla_x L(x^k,\mu^k)^T p + L(x^k,\mu^k) \\ \text{za podmínky } \quad \nabla h(x^k)p + h(x^k) = 0 \end{cases} \tag{QP1}$$

Pomocí drobné úpravy můžeme odvodit nový tvar dílčích úloh. A to

$$\nabla_x L(x^k,\mu^k)^T p = \nabla f(x^k)^T p - (\mu^k)^T h(x^k),$$

$$\nabla_x L(x^k,\mu^k)^T p + L(x^k,\mu^k) = \nabla f(x^k)^T p + f(x^k).$$

V tomto případě pak dílčí úlohy, které budeme označovat jako QP2, budou mít tvar

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\mu^k)p + \nabla f(x^k)^T p + f(x^k) \\ \text{za podmínky } \quad \nabla h(x^k)p + h(x^k) = 0. \end{cases} \tag{QP2}$$

V tuto chvíli zbývá vyřešit, kterou z dílčích úloh QP1 nebo QP2 budeme v mětodě SQP používat. Pokud jde o řešení, z obou úloh dostaneme stejnou hodnotu řešení p^k, lišit se budou pouze hodnotou multiplikátorů. V této práci budeme k výpočtům využívat pomocnou úlohu (QP2), jelikož tento přístup se využívá u většiny zdrojů, např. [1], [3]. Z předchozích úvah a odvození je možné sestavit základní algoritmu metody SQP.

3.1.1. Základní algoritmus SQP pro úlohy s podmínkami tvaru rovností

Tento algoritmus je založen hlavně na aproximaci původní funkce a omezujících podmínek, z tohoto důvodu je označován pouze jako základní. V algoritmu jsou vynechány absolutní členy kvadratických aproximací funkce f, důvod je vysvětlen na začátku drhé kapitoly 2.

Algoritmus 3.1. *Zadáme počáteční aproximace x^0, μ^0, toleranci tol a položíme $k = 0$*

1. *Vypočítáme $\nabla^2_{xx}L(x^k,\mu^k), \nabla h(x^k)$ a $\nabla f(x^k)$*

2. Vyřešíme úlohu kvadratického programování

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\mu^k)p + \nabla f(x^k)^T p \\ \textit{za podmínky} \quad \nabla h(x^k)p + h(x^k) = 0 \end{cases}$$

a získáme řešení p^k a odpovídající hodnotu multiplikátorů $\Delta\mu^k$.

3. Provede se test ukončovacího kritéria

(a) v případě, že není splněno položíme $x^* = x^k$ a $\mu^* = \mu^k$ a ukončíme algoritmus

(b) v případě, že je splněno položíme $x^{k+1} = x^k + p^k$, $\mu^{k+1} = \mu^k + \Delta\mu^k$, $k = k + 1$ a přejdeme zpět na krok 1.

Algoritmus v tomto tvaru je pouze v základním tvaru. Pro lepší konvergenční vlastnosti metody může být výhodnější použití jiné délky kroku než $\alpha = 1$. Základní algoritmus jsem naprogramoval v MATLABU do funkce s názvem **SQP_rovnostni**, tento program je pouze základní a pro výpočty může být i náročný, jelikož jako vstupy je požadován hessián funkce i omezujících podmínek. Ukončovací kritérium je zvoleno jako $\|p\| < tol$, tolerance v daném kódu je volitelná.

```
function [x0,mi,k]=SQP_rovnostni(x0,mi,Gf,Hf,Ro,GRo,HRo,tol)
%Vstupy
%x0 - počáteční bod
%mi - počáteční hodnota multiplikátoru, pro rovnostní omezení
%Gf - gradient funkce
%Hf - hessián funkce
%Ro - rovnostní omezení
%GRo - gradient rovnostních omezení
%HRo - hessián rovnostních omezení
%tol - tolerance ukončovacího kritéria
```

```
%Výstupy
%x0 - bod řešení dané úlohy
%mi - odpovídající hodnota multiplikátoru
%k - počet iterací
imax=100;
k=1;
if (nargin<8)
    tol=1e-6;
end
while k<=imax
    HLf=Hf(x0)+(HRo(x0).'*mi);
    [p,p_mi]=nulprostor(HLf,-Gf(x0),GRo(x0).',-Ro(x0));

    if norm(p)<tol
        break
    else
    x0=x0+p;
    mi=mi+p_mi;
    k=k+1;
    end
end
if k==imax+1
    warning('Dosaženo maximálního počtu iterací')
end
```

Na následujícím příkladě bude demonstrováno použití základního algoritmu 3.1 a porovnání ručního výpočtu s výpočtem pomocí programu **SQP_rovnostni**.

Příklad 3.3. *Použití zakladního algoritmu 3.1 na příkladu*

$$\begin{cases} \textit{minimalizovat } (x_1-2)^4+(x_1-2x_2)^2 \\ \textit{za podmínky} \quad x_1^2-x_2=0 \end{cases}$$

Jako počáteční aproximace jsou použity $x^0 = (2,2)^T$ *a* $\mu^0 = 1$. *Pro namodelování pomocné úlohy* (QP2) *je třeba nejdřív vypočítat všechny potřebné matice a vektory, gradient funkce*

$$\nabla f(x) = \begin{pmatrix} 4(x_1-2)^3 + 2(x_1 - 2x_2) \\ -4x_1 + 8x_2 \end{pmatrix},$$

gradient omezující podmínky

$$\nabla h(x) = \begin{pmatrix} 2x_1 \\ -1 \end{pmatrix}^T$$

a hessián Lagrangeovy funkce

$$\nabla^2_{xx} L(x,\mu) = \nabla^2 f(x) + \mu \nabla^2 h(x) = \begin{pmatrix} 12x_1^2 - 48x_1 + 50 & -4 \\ -4 & 8 \end{pmatrix} + \mu \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}.$$

V bodě x^0 *bude*

$$\nabla^2_{xx} L(x^0,\mu^0) = \begin{pmatrix} 4 & -4 \\ -4 & 8 \end{pmatrix}, \ \nabla f(x^0) = \begin{pmatrix} -4 \\ 8 \end{pmatrix}, \ \nabla h(x^0) = \begin{pmatrix} 4 \\ -1 \end{pmatrix}, \ h(x^0) = 2.$$

Pomocná úloha pak bude tvaru

$$\begin{cases} \textit{minimalizovat } \frac{1}{2} p^T \begin{pmatrix} 4 & -4 \\ -4 & 8 \end{pmatrix} p + \begin{pmatrix} -4 \\ 8 \end{pmatrix}^T p \\ \\ \textit{za podmínky} \quad \begin{pmatrix} 4 \\ -1 \end{pmatrix}^T p + 2 = 0 \end{cases}$$

V tomto kroku vypočítáme pomocnou úlohu pomocí metody nulového prostoru, jako výsledek dostaneme $p^0 = (-0.84, -1.36)^T, \Delta\mu = 0.48$. *A následně vypočítáme nový bod a odpovídající multiplikátor*

$$x^1 = x^0 + p^0 = (1.16,\ 0.64)^T \qquad \mu^1 = \mu^0 + \Delta\mu = 1.48.$$

S novým bodem x^1 *a odpovídajícím multiplikátorem* μ^1 *přepočítáme potřebné matice a vektory*

$\nabla^2_{xx} L(x^1, \mu^1) = \begin{pmatrix} 13.4272 & -4 \\ -4 & 8 \end{pmatrix}$, $\nabla f(x^1) = \begin{pmatrix} -2.6108 \\ 0.48 \end{pmatrix}$, $\nabla h(x^1) = \begin{pmatrix} 2.32 \\ -1 \end{pmatrix}$,
$h(x^1) = 0.7056$.

Pomocná úloha pak bude mít tvar

$$\begin{cases} \textit{minimalizovat } \frac{1}{2} p^T \begin{pmatrix} 13.4272 & -4 \\ -4 & 8 \end{pmatrix} p + \begin{pmatrix} -2.6108 \\ 0.48 \end{pmatrix}^T p \\ \textit{za podmínky} \quad \begin{pmatrix} 2.32 \\ -1 \end{pmatrix}^T p + 0.7056 = 0 \end{cases}$$

Vypočítáním úlohy metodou nulového prostoru dostaneme řešení
$p^1 = (-0.2314,\, 0.1687)^T$, $\Delta\mu = 2.7556$. *Nový bod a odpovídající multiplikátor budou*

$$x^2 = x^1 + p^1 = (0.9286,\, 0.8087)^T \qquad \mu^2 = \mu^1 + \Delta\mu = 4.2356.$$

Další iterace budou popsány zkráceně a to

$k = 2:$ $\quad p^2 = (0.0162,\, 0.0836), \quad x^3 = (0.9448,\, 0.8923), \quad \mu^3 = 7.5951$

$k = 3:$ $\quad p^3 = (-0.0006,\, 0.0015), \quad x^4 = (0.9454,\, 0.8938), \quad \mu^4 = 10.9641$

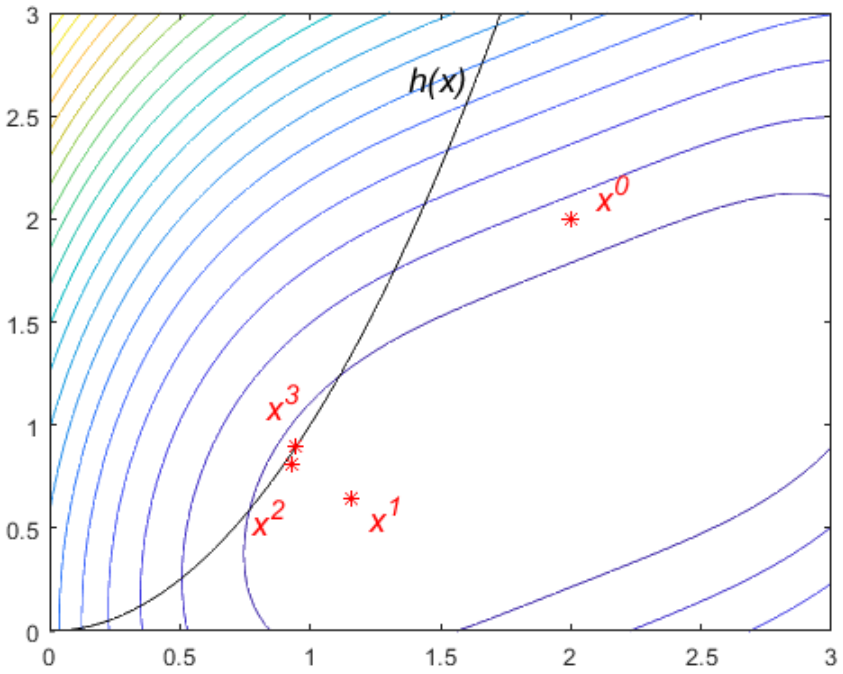
$k = 4:$ $\quad p^4 = (-0.0001,\, 0.0002), \quad x^5 = (0.9455,\, 0.8940), \quad \mu^5 = 14.3343.$

Postup této úlohy je zobrazen na obrázku 3.3.

Při použití naprogramované funkce ***SQP_rovnostni*** *v MATLABU získáme výsledek ve tvaru*

```
[x0,mi,k]=SQP_rovnostni([2;2],1,@(x)[4*((x(1)-2)^3)+2*(x(1)-2*x(2));
-4*x(1)+8*x(2)],@(x)[12*(x(1)^2)-48*x(1)+50 -4;-4 8],
@(x)(x(1)^2)-x(2),@(x)[2*x(1);-1],@(x)[2 0;0 0])


x0 =
    0.9456
    0.8941
```

```
mi =

   34.5580


k =

    11
```



Obrázek 3.3: Příklad 3.3

3.2. Úlohy nelineárního programování s rovnostními a nerovnostními omezeními

V praxi většinou není možné zaručit úlohy pouze s omezeními tvaru rovnosti. Pro na základě předchozích úvah a odvození, můžeme rovnostní verzi SQP zobecnit na obecnou úlohu nelineárního programování (1.1). V předchozí části jsme

odvodili dva tvary pomocné úlohy (QP1) a (QP2). Logickým postupem je využít nejdříve tvar (QP1), pomocná úloha bude v tomto případě ve tvaru

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\lambda^k,\mu^k)^Tp + \nabla_x L(x^k,\lambda^k,\mu^k)^Tp + L(x^k,\lambda^k,\mu^k) \\ \text{za podmínky } \quad \nabla h(x^k)p + h(x^k) = 0, \\ \qquad\qquad\quad \nabla g(x^k)p + g(x^k) \leq 0. \end{cases}$$

(QP1-Obecná)

V tuto chvíli se nám, ale objevuje problém, kdy bod x^* který je řešením pomocné úlohy (QP1-Obecná), nemusí být řešením původní úlohy, ale pouze sedlovým bodem. Zbývá nám tedy jediná možnost, a to využít druhý tvar pomocné úlohy (QP2). V takovém případě bude pomocná úloha

$$\begin{cases} \text{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\lambda^k,\mu^k)p + \nabla f(x^k)^Tp + f(x^k) \\ \text{za podmínky } \quad \nabla h(x^k)p + h(x^k) = 0, \\ \qquad\qquad\quad \nabla g(x^k)p + g(x^k) \leq 0. \end{cases} \qquad \text{(QP2-Obecná)}$$

V tuto chvíli můžeme zobecnit základní algoritmus pro obecnou úlohu nelineárního programování.

3.2.1. Základní algoritmus SQP pro obecnou úlohu nelineárního programování

Algoritmus je založen na tvorbě pomocných úloh tvaru (QP2-Obecná), pro výpočet takových úloh je vhodná metoda aktivní množiny.

Algoritmus 3.2. *Zadáme počáteční aproximace $x^0, \lambda^0, \mu^0, \lambda \geq 0$, toleranci tol a položíme $k = 0$*

1. *Vypočítáme $\nabla^2_{xx}L(x^k,\lambda^k,\mu^k), \nabla h(x^k)$ a $\nabla f(x^k)$*

2. *Vyřešíme úlohu kvadratického programování*

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\lambda^k,\mu^k)^Tp + \nabla_x f(x^k)^Tp \\ \textit{za podmínky} \quad \nabla g(x^k)p + g(x^k) \leq 0, \\ \qquad\qquad\quad \nabla h(x^k)p + h(x^k) = 0. \end{cases}$$

a získáme řešení p^k a odpovídající hodnoty multiplikátorů $\Delta\lambda^k, \Delta\mu^k$

3. Provede se test ukončovacího kritéria

   (a) v případě, že je splněno položíme $x^* = x^k$, $\lambda^* = \lambda^k$ *a* $\mu^* = \mu^k$ *a ukončíme algoritmus*

   (b) v případě, že není splněno položíme $x^{k+1} = x^k + p^k$, $\lambda^{k+1} = \lambda^k + \Delta\lambda^k$, $\mu^{k+1} = \mu^k + \Delta\mu^k$, $k = k + 1$ *a přejdeme zpět na krok 1.*

Základní algoritmus pro obecnou úlohu jsem naprogramoval v MATLABU do funkce s názvem **SQP_obecna**, tento program je určený pro řešení obecných úloh nelineárního programování, nevýhodou tohoto typu úloh je velké množství vstupů, které jsou nutné pro použití metody.

```
function [x0,lambda,mi,k]=SQP_obecna(x0,mi,Gf,Hf,Ro,GRo,HRo,lambda,
No,GNo,HNo,tol)
%Vstupy
%x0 - počáteční bod
%mi - počáteční hodnota multiplikátoru, pro rovnostní omezení
%Gf - gradient funkce
%Hf - hessián funkce
%Ro - rovnostní omezení
%GRo - gradient rovnostních omezení
%HRo - hessián rovnostních omezení
%lambda - počáteční hodnota multiplikátoru, pro nerovnostní omezení
%No - nerovnostní omezení
%GNo - gradient nerovnostních omezení
%HNo - hessián nerovnostních omezení
%tol - tolerance ukončovacího kritéria

%Výstupy
%x0 - bod řešení dané úlohy
%lambda - odpovídající hodnota multiplikátoru pro nerovnostní omezení
%mi - odpovídající hodnota multiplikátoru pro rovnostní omezení
```

```
%k - počet iterací

imax=100;
k=0;
if (nargin<12)
   tol=1e-6;
end
if (nargin<11)
    while k<=imax
    HLf=Hf(x0)+(HRo(x0).'*mi);
    [p,p_mi]=nulprostor(HLf,-Gf(x0),GRo(x0).',-Ro(x0));

    if norm(p)<tol
        lambda=[];
        break
    else
    x0=x0+p;
    mi=mi+p_mi;
    if k==imax
        lambda=[];
        warning('Dosaženo maximálního počtu iterací')
        break
    else
    k=k+1;
    end
    end
    end

else
    while k<=imax
```

```
    HLf=Hf(x0)+(HNo(x0).'*lambda)+(HRo(x0).'*mi);
    p0=linsolve([GRo(x0).';GNo(x0).'],[-Ro(x0);-No(x0)]);
    [p,p_lambda,p_mi]=MetodaAktMnoziny(p0,HLf,-Gf(x0),GNo(x0).',
    -No(x0),GRo(x0).',-Ro(x0));

    if norm(p)<tol
        break
    else
    x0=x0+p;
    lambda=lambda+p_lambda;
    mi=mi+p_mi;
    if k==imax
        warning('Dosaženo maximálního počtu iterací')
        break
    else
    k=k+1;
    end
    end
    end
end
end
```

Příklad 3.4. *Použití zakladního algoritmu 4.2 na příkladu, upravená verze příkladu 3.3,*

$$\begin{cases} \textit{minimalizovat } (x_1-2)^4+(x_1-2x_2)^2 \\ \textit{za podmínky} \quad x_1^2-x_2=0 \\ \qquad\qquad\quad x_2\geq 1 \end{cases}$$

Jako počáteční aproximaci jsou použity $x^0=(2,\,2)^T$, $\mu^0=1$ *a* $\lambda^0=1$. *Vypočítáme všechny potřebné matice a vektory pro namodelování pomocné úlohy* (QP2-Obecná). *Hessián Lagrangeovy funkce*

$$\nabla^2_{xx}L(x,\lambda,\mu)=\nabla^2 f(x)+\lambda\nabla^2 g(x)+\mu\nabla^2 h(x)=$$

$$= \begin{pmatrix} 12x_1^2 - 48x_1 + 50 & -4 \\ -4 & 8 \end{pmatrix} + \lambda \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \mu \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix},$$

gradient funkce

$$\nabla f(x) = \begin{pmatrix} 4(x_1 - 2)^3 + 2(x_1 - 2x_2) \\ -4x_1 + 8x_2 \end{pmatrix},$$

gradienty omezujících podmínek

$$\nabla h(x) = \begin{pmatrix} 2x_1 \\ -1 \end{pmatrix}^T, \nabla g(x) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}^T.$$

V bodě x^0 *bude*

$$\nabla^2_{xx} L(x^0, \lambda^0, \mu^0) = \begin{pmatrix} 4 & -4 \\ -4 & 8 \end{pmatrix}, \nabla f(x^0) = \begin{pmatrix} -4 \\ 8 \end{pmatrix}, \nabla h(x^0) = \begin{pmatrix} 4 \\ -1 \end{pmatrix}^T, h(x^0) = 2,$$

$$\nabla g(x^0) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}^T, g(x^0) = -1$$

Pomocná úloha pak bude mít tvar

$$\begin{cases} \textit{minimalizovat } \frac{1}{2} p^T \begin{pmatrix} 4 & -4 \\ -4 & 8 \end{pmatrix} p + \begin{pmatrix} -4 \\ 8 \end{pmatrix}^T p \\ \textit{za podmínky} \quad \begin{pmatrix} 4 \\ -1 \end{pmatrix}^T p + 2 = 0 \\ \qquad\qquad\quad \begin{pmatrix} 0 \\ -1 \end{pmatrix}^T p - 1 \leq 0 \end{cases}$$

Pro vypočítání pomocné úlohy pomocí metody aktivní množiny je třeba zadat počáteční bod $p^{poc} = (-\frac{3}{4}, -1)^T$. *Jako výsledek dostaneme* $p^0 = (-\frac{3}{4}, -1)$, $\Delta\lambda = 2.25$ *a* $\Delta\mu = 0.75$. *A následně vypočítáme nový bod a odpovídající multiplikátory*

$$x^1 = x^0 + p^0 = (1.25, 1)^T, \mu^1 = \mu^0 + \Delta\mu = 1.75, \lambda^1 = \lambda^0 + \Delta\lambda = 3.25.$$

S novým bodem x^1 *a odpovídajícími multiplikátory* μ^1 *a* λ^1 *přepočítáme potřebné matice a vektory*

$$\nabla^2_{xx} L(x^1, \lambda^1, \mu^1) = \begin{pmatrix} 12.5 & -4 \\ -4 & 8 \end{pmatrix}, \nabla f(x^1) = \begin{pmatrix} -3.1875 \\ 3 \end{pmatrix}, \nabla h(x^1) = \begin{pmatrix} 2.5 \\ -1 \end{pmatrix}^T$$

$$h(x^1) = 0.5625,\ \nabla g(x^1) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}^T,\ g(x^1) = 0$$

Pomocná úloha pak bude mít tvar

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}p^T \begin{pmatrix} 12.5 & -4 \\ -4 & 8 \end{pmatrix} p + \begin{pmatrix} -3.1875 \\ 3 \end{pmatrix}^T p \\ \textit{za podmínky} \quad \begin{pmatrix} 2.5 \\ -1 \end{pmatrix}^T p - 0.5625 = 0 \\ \qquad\qquad\quad \begin{pmatrix} 0 \\ -1 \end{pmatrix}^T p - 0 \leq 0 \end{cases}$$

Pro vypočítání úlohy použijeme jako počáteční bod $p^{poc} = (-0.2250,\ 0)^T$. *Výsledek bude* $p^1 = (-0.225,\ 0)^T$, $\nabla\lambda = 1.5225$ *a* $\nabla\mu = 2.3775$. *Vypočteme nový bod a odpovídající multiplikátory*

$$x^2 = x^1 + p^1 = (1.025,\ 1)^T,\ \mu^2 = \mu^1 + \nabla\mu = 4.1275,\ \lambda^2 = \lambda^1 + \nabla\lambda = 4.7725.$$

Další postup bude ukázán ve stručné bodobě

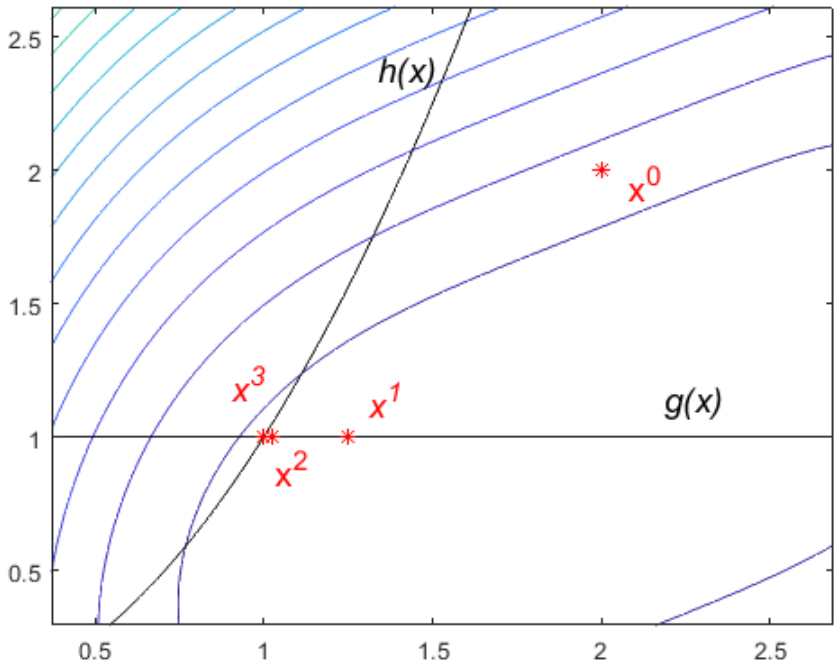
$k = 2:\quad p^2 = (-0.0247,\ 0),\quad x^3 = (1.0003,\ 1),\quad \mu^3 = 7.1482,\quad \lambda^3 = 5.7506$

$k = 3:\quad p^3 = (-0.0003,\ 0),\quad x^4 = (1,\ 1),\quad \mu^4 = 10.1494,\quad \lambda^4 = 6.7493$

Graficky je postup výpočtu této úlohy znázorněn na obrázku 3.4. Při použití naprogramované funkce ***SQP_obecna*** *v MATLABU získáme výsledek ve tvaru*

```
[x0,lambda,mi,k]=SQP_obecna([2;2],1,@(x)[4*((x(1)-2)^3)+2*(x(1)
-2*x(2));-4*x(1)+8*x(2)],@(x)[12*(x(1)^2)-48*x(1)+50 -4;-4 8],
@(x)(x(1)^2)-x(2),@(x)[2*x(1);-1],@(x)[2 0;0 0],1,@(x)1-x(2),
@(x)[0;-1],@(x)[0 0;0 0])


x0 =
    1.0000
    1.0000


lambda =
    6.7493
```

```
mi =

   10.1494


k =

     4
```



Obrázek 3.4: Úloha (3.4)

Příklad 3.5. *[5] Máme příklad*

$$\begin{cases} \textit{minimalizovat } (x_1-2)^2+(x_2-1)^2 \\ \textit{za podmínky} \quad x_1-2x_2+1=0 \\ \qquad\qquad\qquad\quad 0.25x_1^2+x_2^2-1\le 0 \end{cases}$$

Jako počáteční aproximace jsou použity $x^0=(-2,-2)^T$, $\mu^0=1$ *a* $\lambda^0=1$.

Vypočítáme všechny potřebné matice a vektory. Gradient funkce

$$\nabla f(x) = \begin{pmatrix} 2x_1 - 4 \\ 2x_2 - 2 \end{pmatrix},$$

gradienty omezujících podmínek

$$\nabla h(x) = \begin{pmatrix} 1 \\ -2 \end{pmatrix}^T, \nabla g(x) = \begin{pmatrix} 0.5x_1 \\ 2x_2 \end{pmatrix},$$

hessián Lagrangeovy funkce

$$\nabla^2_{xx} L(x, \lambda, \mu) = \nabla^2 f(x) + \lambda \nabla^2 g(x) + \mu \nabla^2 h(x) =$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} + \lambda \begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix} + \mu \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

V bodě x^0 *budou*

$$\nabla^2_{xx} L(x^0, \lambda^0, \mu^0) = \begin{pmatrix} 2.5 & 0 \\ 0 & 2 \end{pmatrix}, \nabla f(x^0) = \begin{pmatrix} -8 \\ -6 \end{pmatrix}, \nabla h(x^0) = \begin{pmatrix} 1 \\ -2 \end{pmatrix},$$

$$h(x^0) = -5, \nabla g(x^0) = \begin{pmatrix} -1 \\ -4 \end{pmatrix}, g(x^0) = 4.$$

Pomocná úloha pak bude mít tvar

$$\begin{cases} \textit{minimalizovat } \frac{1}{2} p^T \begin{pmatrix} 2.5 & 0 \\ 0 & 4 \end{pmatrix} p + \begin{pmatrix} -8 \\ -6 \end{pmatrix}^T p \\ \textit{za podmínky} \quad \begin{pmatrix} 1 \\ -2 \end{pmatrix}^T p + 3 = 0 \\ \qquad\qquad\quad \begin{pmatrix} -1 \\ -4 \end{pmatrix}^T p + 4 \leq 0 \end{cases}$$

Postup výpočtů

$k = 0$:

$p^0 = (2.2857, 2.6429)^T, \quad x^1 = (0.2857, 0.6429)^T, \mu^1 = 3.2857, \lambda^1 = 1$

$k = 1$:

$p^1 = (0.7208, 0.3604)^T, \quad x^2 = (1.0065, 1.0032)^T, \mu^2 = 4.6827, \lambda^2 = 2.6074$

$k = 2$:

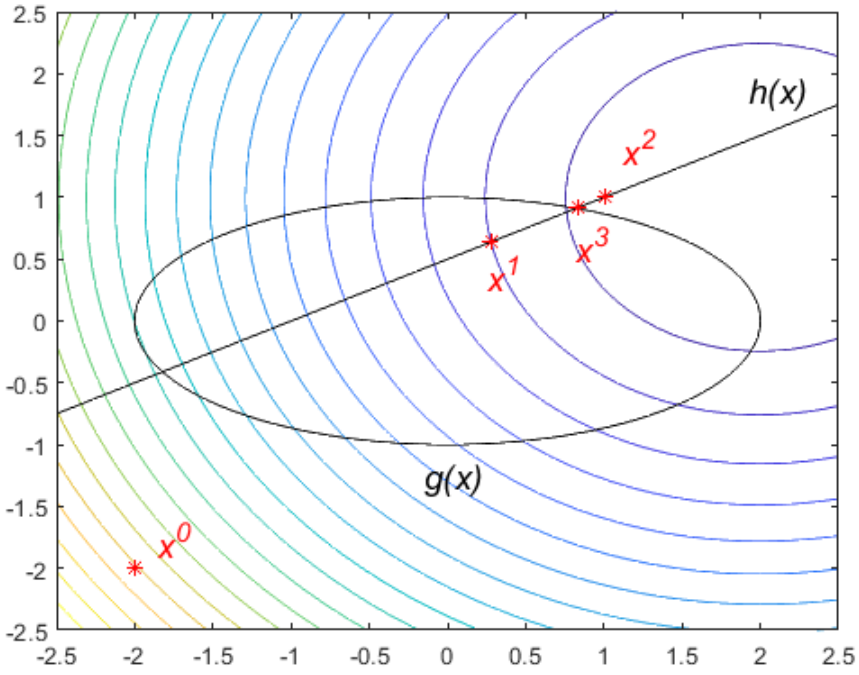
$p^2 = (-0.1724, -0.0862)^T, \quad x^3 = (0.8341, 0.9170)^T,\ \mu^3 = 6.2825,\ \lambda^3 = 4.5088$

$k = 3$:

$p^3 = (-0.0111, -0.0056)^T, \quad x^4 = (0.8229, 0.9115)^T,\ \mu^4 = 7.8825,\ \lambda^4 = 6.3775$

Graficky je úloha znázorněna na obrázku 3.5. Při použití naprogramované funkce ***SQP_obecna*** *v MATLABU získáme výsledek ve tvaru*

```
[x0,lambda,mi,k]=SQP_obecna([-2;-2],1,@(x)[2*x(1)-4;2*x(2)-2],
@(x)[2 0;0 2],@(x)x(1)-2*x(2)+1,@(x)[1;-2],@(x)[0 0;0 0],1,
@(x)0.25*x(1)^2+x(2)^2-1,@(x)[0.5*x(1);2*x(2)],@(x)[0.5 0;0 2])

x0 =
    0.8229
    0.9114

lambda =
    8.2243

mi =
    9.4770

k =
     5
```



Obrázek 3.5: Úloha (3.5)

Kapitola 4

Rozšíření základního algoritmu

Jelikož třetí kapitola nám dává pouze základní algoritmus, existují nástroje pro jeho upravení. Z tohoto důvodu je v této kapitole uvedena aproximace hessiánu a úprava kroku. V tomto případě jsem čerpal zejména z [1] a [2].

4.1. Aproximace hessiánu

Jedním z problémů, které provází použití základní SQP metody je vyčíslení hessiánu v každé iteraci. Velmi často se tedy přistupuje k použití aproximace. Jelikož se jedná o iterační proces máme dvě možnosti, jak hessián aproximovat.

1. Pevná aproximace

   V principu se jedná o jednoduchou možnost nahrazení hessiánu, pevně stanovenou aproximací. Aproximaci určíme pouze na začátku a po celou dobu iteračního procesu zůstane stejná. V případě složitějších příkladů, ale není vhodná.

2. Aproximace s aktualizacemi

   V tomto případě je do iteračího procesu zadán počáteční odhad hessiánu. V každé iteraci aktualizujeme matici, která aproximuje hessián. V případě správného postupu bude tvar matice konvergovat ke skutečnému hessiánu.

V případě SQP je vhodné použít kvazinewtonovské aproximace. Tato metoda aproximace navíc může vyřešit i případnou nedefinitnost původního hessiánu.

SQP metoda nahrazuje Lagrangeovu funkci v bodě x^k kvadratickou částí Taylorova rozvoje. V případě kvazinewtonovské aproximace bude funkce ve stejném bodě nahrazena upravenou kvadratickou aproximací q_k, která bude mít tvar

$$q_k(p) = f(x^k) + (g^k)^T p + \frac{1}{2} p^T B_k p.$$

Základním požadavkem na matici B_k je aby s dostatečnou přesností aproximovala hessián. Jelikož jde o aproximaci může mít matice B_k i lepší vlastnosti jako je například definitnost. Druhým krokem aproximace je určit novou matici B_{k+1}, která bude stále vhodně aproximovat hessián. Nová matice se bude volit tak, aby platila kvazinewtonovská podmínka která je tvaru

$$B_{k+1}(x^{k+1} - x^k) = g^{k+1} - g^k$$

Pro přehlednost se podmínka uvádí ve tvaru

$$B_{k+1} s^k = y^k$$

kde

$$s^k = x^{k+1} - x^k,$$
$$y^k = g^{k+1} - g^k.$$

Další požadavek, který můžeme na matice B_k mít je zachování pozitivní definitnosti, tuto vlastnost zaručuje splnění tzv. křivostní podmínky, která má tvar

$$(s^k)^T (y^k) > 0$$

Aby bylo možné určit matici B_{k+1} jednoznačně, je třeba ještě dalších podmínek, jednou z nich je, že se B_{k+1} od B_k bude lišit pouze o aktualizaci nízké hodnosti,tj. $B_{k+1} = B_k + U_k(s^k, y^k, B_k)$, která bude záviset na původní matici B_k a vektorech s^k, y^k.

Formule BFGS

Nejpoužívanější kvazinewtonovskou formulí pro aproximaci hessiánu je BFGS, která spadá do skupiny aproximací s aktualizacemi hodnosti 2. Formule pro

výpočet matice B_{k+1} má tvar

$$B_{k+1} = B_k + \frac{y^k (y^k)^T}{(y^k)^T s^k} - \frac{B_k s^k (s^k)^T B_k}{(s^k)^T B_k s^k}. \tag{4.1}$$

Tato formule zachovává pozitivní definitnost za přepokladu, že platí křivostní podmínka.

Věta 4.1. *[2, Věta 9.5] Za předpokladu, že platí podmínka*

$$(s^k)^T y^k > 0,$$

formule BFGS zachovává pozitivní definitnost.

Formule BFGS je speciální právě ve schopnosti snadno zachovávat pozitivní definitnost, což je vlastnost, která je u mnoha metod nelineárního programování žádaná.

Poznámka 2. *Metoda SQP bude za jistých předpokladů konvergovat k přesnému řešení i v případě aproximace hessiánu pomocí BFGS formule, jak bude uvedeno v poslední kapitole.*

4.1.1. Algoritmus SQP doplněný o BFGS

Základní algoritmus 4.2 doplníme o formuli BFGS, čímž usnadníme výpočty díky tomu, že aproximujeme hessián Lagrangeovy funkce pomocí matice B_k.

Algoritmus 4.1. *[1] Zadáme počáteční aproximace $B^0, x^0, \lambda^0, \mu^0, \lambda \geq 0$, toleranci tol a položíme $k = 0$*

1. *Vypočítáme $\nabla h(x^k)$a $\nabla f(x^k)$.*

2. *Vyřešíme úlohu kvadratického programování*

$$\begin{cases} \text{minimalizovat } \frac{1}{2} p^T B^k p + \nabla_x f(x^k)^T p \\ \text{za podmínky} \quad \nabla g(x^k) p + g(x^k) \leq 0, \\ \qquad\qquad\quad \nabla h(x^k) p + h(x^k) = 0, \end{cases}$$

a získáme řešení p^k a odpovídající hodnoty multiplikátorů $\Delta\lambda^k, \Delta\mu^k$.

3. Provede se test ukončovacího kritéria

   (a) v případě, že je splněno, položíme $x^* = x^k$, $\lambda^* = \lambda^k$ *a* $\mu^* = \mu^k$ *a ukončíme algoritmus*

   (b) v případě, že není splněno, položíme $x^{k+1} = x^k + p^k$, $\lambda^{k+1} = \lambda^k + \Delta\lambda^k$, $\mu^{k+1} = \mu^k + \Delta\mu^k$, $k = k + 1$, *pomocí formule BFGS* (4.1) *vypočítáme novou matici* B^{k+1} *a přejdeme zpět na krok 1.*

Zavedením metody BFGS do základního algoritmu 3.1 se zbavíme nutnosti počítat Lagrangeovův hessián, který u složitějších úloh může být velmi náročný na výpočet. Ale je třeba navíc ještě zadat i počáteční aproximaci hessiánu B_0, tato matice musí být symetrická a pozitivně definitní.
Algoritmus doplněný o aproximaci hessiánu jsem naprogramoval v MATLABU do funkce s názvem **SQP_BFGS**.

```
function [x0,lambda,mi,k]=SQP_BFGS(x0,B,mi,Gf,Ro,GRo,lambda,No,
GNo,tol)
%Vstupy
%x0 - počáteční bod
%mi - počáteční hodnota multiplikátoru, pro rovnostní omezení
%Gf - gradient funkce
%Ro - rovnostní omezení
%GRo - gradient rovnostních omezení
%lambda - počáteční hodnota multiplikátoru, pro nerovnostní omezení
%No - nerovnostní omezení
%GNo - gradient nerovnostních omezení
%tol - tolerance ukončovacího kritéria

%Výstupy
%x0 - bod řešení dané úlohy
%lambda - odpovídající hodnota multiplikátoru pro nerovnostní omezení
%mi - odpovídající hodnota multiplikátoru pro rovnostní omezení
```

```
%k - počet iterací

imax=100;
k=0;
if (nargin<10)
   tol=1e-6;
end
if (nargin<9)
    while k<=imax
    [p,p_mi]=nulprostor(B,-Gf(x0),GRo(x0).',-Ro(x0));
    if norm(p)<tol
        lambda=[];
        break
    else
    x=x0+p;
    mi=mi+p_mi;

    y=Gf(x)+(GRo(x)*mi)-(Gf(x0)+(GRo(x0)*mi));
    B=B+((y*(y)')/((y)'*p))-((B*p*(p)'*B)/((p)'*B*p));

    x0=x;
    if k==imax
    warning('Dosaženo maximálního počtu iterací')
    else
    k=k+1;
    end
    end
    end

else
```

```
    while k<=imax
    p0=linsolve([GRo(x0).';GNo(x0).'],[-Ro(x0);-No(x0)]);
    [p,p_lambda,p_mi]=MetodaAktMnoziny(p0,B,-Gf(x0),GNo(x0).',
    -No(x0),GRo(x0).',-Ro(x0));

    if norm(p)<tol
        break
    else
    x=x0+p;
    lambda=lambda+p_lambda;
    mi=mi+p_mi;

    y=Gf(x)+(GRo(x)*mi)+(GNo(x)*lambda)-(Gf(x0)+(GRo(x0)*mi)
    +(GNo(x0)*lambda));
    B=B+((y*(y)')/((y)'*p))-((B*p*(p)'*B)/((p)'*B*p));

    x0=x;
    if k==imax
    warning('Dosaženo maximálního počtu iterací')
    else
    k=k+1;
    end
    end
    end
end
end
```

Příklad 4.1. *[6] Použití upraveného algoritmu na příklad 3.3, na kterém jsme demonstrovali základní algoritmus pro rovnostní omezení.*

$$\begin{cases} \textit{minimalizovat } (x_1-2)^4+(x_1-2x_2)^2 \\ \textit{za podmínky} \quad x_1^2-x_2=0 \end{cases}$$

Jako počáteční aproximace je použito $x_0 = (2,2)^T$ *a* $\mu^0 = 1$. *Narozdíl od základního algoritmu zadáme prvotní aproximaci hessiánu, která bude mít tvar*

$$B_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Díky této změně bude dílčí úloha mít jiný tvar

$$\begin{cases} \textit{minimalizovat } \frac{1}{2} p^T \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} p + \begin{pmatrix} -4 \\ 8 \end{pmatrix}^T p \\ \\ \textit{za podmínky} \quad \begin{pmatrix} 4 \\ -1 \end{pmatrix} p + 2 = 0 \end{cases}$$

Pomocnou úlohu vypočítáme pomocí metody nulového prostoru, průběh výpočtů bude totožný s původním příkladem, jediný rozdílem bude dopočítávání matice B_k *pomocí formule BFGS* (4.1). *První iterace bude mít výsledek*

$k = 0: \quad p = (-2.1176, -6.4706)^T, \ \Delta\mu = 1.5294, \quad x^1 = (-0.176, -4.4706)^T,$
$\mu^1 = 2.5294$

Nyní když máme nový bod je třeba aktualizovat matici B_0 *na matici* B_1. *Použitím formule BFGS dostaneme*

$$B_1 = \begin{pmatrix} 2.2109 & 2.4561 \\ 2.4561 & 5.8871 \end{pmatrix}$$

Další iterace budou

$k = 1: \quad p = (5.2367, 3.2523)^T, \ \Delta\mu = -3.2860, \quad x^2 = (5.1191, -1.2183)^T,$
$\mu^2 = -0.7566$

$k = 2: \quad p = (-0.7378, 19.8691)^T, \ \Delta\mu = -11.5555, \quad x^3 = (4.3812, 18.6508)^T,$
$\mu^3 = -12.3121$

$k = 3: \quad p = (-2.1561, -18.3487)^T, \ \Delta\mu = -1.0717, \quad x^4 = (2.2251, 0.3021)^T,$
$\mu^4 = -13.3838$

$k = 4: \quad p = (-0.8264, 0.9713)^T, \ \Delta\mu = 8.9892, \quad x^5 = (1.3987, 1.2733)^T,$
$\mu^5 = -4.3946$

$\vdots$

$k = 12: \quad p = (0,0)^T,\ \Delta\mu = 3.3707, \quad x^{13} = (0.9456, 0.8941)^T,$
$\mu^{13} = 16.7330$

Při použití naprogramované funkce ***SQP_BFGS*** *v MATLABU získáme výsledek ve tvaru*

```
[x0,lambda,mi,k]=SQP_BFGS([2;2],[1 0;0 1],1,@(x)[4*((x(1)-2)^3)+
2*(x(1)-2*x(2));-4*x(1)+8*x(2)],@(x)(x(1)^2)-x(2),@(x)[2*x(1);-1])

x0 =
    0.9456
    0.8941

lambda =
     []

mi =
   25.6331

k =
    13
```

Pokud na uvedeném příkladu porovnáme použití základního algoritmu a upraveného algoritmu, můžeme vidět, že základní algoritmus počítá rychleji, ale jsme nuceni v každém kroku počítat hodnotu Lagrangeova hessiánu. Upravený algoritmus naopak počítá pomaleji, ale není třeba počítat Lagrangeův hessián. Podmínky, na jejichž základě upravený Algoritmus 4.1 bude konvergovat, budou uvedeny v poslední kapitole v konvergenční větě.

4.2. Úprava kroku

V rámci algoritmů hledajících minimum zadané funkce se často řeší otázka volby délky kroku α. Metody, které jsou založeny na hledání směru p, ve kterém

funkci minimalizujeme, vždy tuto délku kroku využívají. V rámci algoritmů můžeme mít více možností, jak délku kroku určit a to

1. Pevně zvolená délka - délka kroku je v každé iteraci stejná a v průběhu algoritmu se nemění. Například Newtonovy metody nebo základní algoritmus pro metodu SQP užívají délka kroku $\alpha = 1$,

2. Přesný výpočet délky kroku - v průběhu výpočtu máme přímo vzorec, kterým délku kroku určíme, například Metoda aktivní množiny, v kapitole 2.2.1.

3. Nepřesná volba délky kroku - často se objevuje podmínka, aby délka kroku dostatečně snižovala funkční hodnotu dané funkce.

V základním algoritmu metody SQP je délka kroku $\alpha = 1$. V tomto bodě nám, ale může vzniknout konfrontace mezi dostatečnou redukcí funkční hodnoty minimalizované funkce a uspokojením omezení. Právě merit funkce (také hodnotové nebo pomocné funkce), které budou definovány později, jsou navrženy tak, aby mezi těmito cíli udrželi rovnováhu a podle toho řídily algoritmus. Proto krok $\alpha = 1$ budeme akceptovat pouze povede-li k dostatečnému snížení hodnoty merit funkce $\Phi(x; \theta)$.
V neomezené optimalizaci je minimalizovaná funkce přirozenou volbou pro merit funkci. Tuto volbu můžeme použít i u podmíněné optimalizace, ale pouze v případě, že výchozí bod a všechny následné iterace splňují omezující podmínky. Některé metody podmíněné optimalizace nám ale dovolují v některých iteracích omezení porušit, a v takovém případě použití minimalizované funkce jako merit funkce není vhodné. Nejčastěji používaný tvar merit funkce je

$$\Phi(x; \theta) = f(x) + \theta \sum_{i}^{r} |h_i(x)| + \theta \sum_{j}^{m} [g_j(x)]^{-}, \tag{4.2}$$

kde $[x]^{-} = max\{0, -x\}$ a $\theta > 0$ je penalizační parametr merit funkce.
Další používanou merit funkcí je Fletcherův rozšířený Lagrangián (anglicky Flet-

cher's augmented Lagrangian). K dispozici máme tvar pouze pro úlohy s rovnostními omezeními

$$\Phi(x;\theta) = f(x) - \mu(x)^T h(x) + \frac{1}{2}\theta \left\|h(x)\right\|^2, \tag{4.3}$$

kde $\theta > 0$ a platí

$$\mu(x) = \left(A(x)A(x)^T\right)^{-1} A(x)\nabla f(x).$$

Matice $A(x)$ označuje Jakobián omezujících podmínek. Oba typy merit funkce, které jsou zde uvedeny označujeme jako exaktní neboli přesné, jelikož řešení úlohy nelineáního programování (1.1) je minimem merit funkce (4.2) i (4.3).

Definice 4.1. *[1, Definice 15.1] Merit funkce $\Phi(x;\theta)$ se nazývá exaktní, pokud existuje skalár $\theta^* > 0$ takový, že pro jakékoliv $\theta > \theta^*$ je jakékoliv lokální řešení úlohy nelineárního programování (1.1) lokálním minimem merit funkce $\Phi(x;\theta)$.*

Ve zbytku tohoto textu se budeme zabývat pouze úlohami s rovnostními omezeními.

Poznámka 3. *[1] Merit funkce (4.2) je exaktní pokud*

$$\theta^* = max\left\{|\mu_i^*|, i \in J\right\}.$$

Když se podíváme na tvar jednotlivých funkcí můžeme říct, že merit funkce (4.2) je snazší na vyčíslení a proto se častěji využívá v optimalizačních algoritmech. Hlavním problémem, který se může v případě použití této funkce objevit je tzv. Maratos efekt. Tento efekt se projeví v případě, že algoritmus zamítne krok, který by nás posunul dostatečně blízko k hledanému řešení úlohy, jelikož v případě merit funkce půjde o navýšení její funkční hodnoty. Jednou z možností jak eliminovat tento efekt je použití Fletcherova rozšířeného Lagrangiánu (4.3). Tento tvar je odolný vůči tomuto efektu, ale narozdíl od jednoduššího tvaru (4.2) je tento tvar náročnější pro vyčíslení, navíc je třeba v každém kroku vypočítat $\lambda(x)$.

Pro použití v metodě SQP je třeba vyřešit za jakých podmínek budeme délku kroku α upravovat. Celá podmínka bude tvaru

$$\Phi(x^k + \alpha_k p^k; \theta^k) > \Phi(x^k; \theta^k) + \eta\alpha D\Phi(x^k; p^k),$$

kde $\eta \in (0, 1)$ je daný parametr a $D\Phi(x^k; p^k)$ je směrová defivace merit funkce ve směru p^k. Merit funkce (4.2) není hladká, obsahuje absolutní hodnotu a funkci $[x]^-$, není tedy diferencovatelná. Nebrání nám to však ve výpočtu směrové derivace ve směru p_k.

Věta 4.2. *[1, Věta 18.2] Nechť p^k a $\Delta\mu$ jsou generované metodou SQP pro úlohy s rovnostními omezeními. Směrová derivace merit funkce Φ (4.2) ve směru p^k splňuje*

$$D(\Phi(x^k; \theta); p^k) = \nabla f(x^k)^T p^k - \theta\|h(x^k)\|_1.$$

Zůstává nám otázka, jak zvolit hodnotu parametru θ. V literatuře se můžeme setkat s více způsoby, jak hodnotu pametru zvolit, často se volí na základě Lagrangeových multiplikátorů. Obecně lze říct, že ve směru p_k hodnota funkce Φ, klesá pokud hessián Lagrangeovy funkce $\nabla^2_{xx}L$ je pozitivně definitní a

$$\theta > \|\mu_{k+1}\|_\infty.$$

Existují i efektivnější alternativy jak volit hodnotu parametru θ. Tyto volby, ale závisí na volbě dalších parametrů a bylo by třeba hlubší analýzy pro posouzení, která alternativa je pro danou úlohu vhodnější.

4.2.1. Algoritmus SQP s merit funkcí

Algoritmus vychází ze základního algoritmu pro obecné úlohy 4.2 a je doplněn o testování merit funkce.

Algoritmus 4.2. *[1] Zadáme počáteční aproximace $x^0, \lambda^0, \mu^0, \lambda \geq 0$, toleranci tol a položíme $k = 0$*

1. *Vypočítáme $\nabla^2_{xx}L(x^k, \lambda^k, \mu^k), \nabla h(x^k)$ a $\nabla f(x^k)$*

2. *Vyřešíme úlohu kvadratického programování*

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}p^T\nabla^2_{xx}L(x^k,\lambda^k,\mu^k)^Tp + \nabla_x f(x^k)^Tp \\ \textit{za podmínky} \quad \nabla g(x^k)p + g(x^k) \leq 0, \\ \qquad\qquad\quad \nabla h(x^k)p + h(x^k) = 0. \end{cases}$$

a získáme řešení p^k *a odpovídající hodnoty multiplikátorů* $\Delta\lambda^k, \Delta\mu^k$

3. *Provede se test ukončovacího kritéria,*

   (a) *v případě, že není splněno, otestujeme zda je krok* $\alpha_k = 1$ *vhodný pomocí merit funkce, kde* $\eta \in (0, 0.5)$

   $$\Phi(x^k + \alpha_k p^k; \theta^k) > \Phi(x^k; \theta^k) + \eta\alpha D\Phi(x^k; p^k).$$

   Pokud bude platit podmínka
   $x^{k+1} = x^k + \tau\alpha_k p^k$, $\lambda^{k+1} = \lambda^k + \tau\alpha_k\Delta\lambda^k$, $\mu^{k+1} = \mu^k + \tau\alpha_k\Delta\mu^k$,
   kde $\tau \in (0, 1)$.
   Pokud podmínka splněna není
   $x^{k+1} = x^k + \alpha_k p^k$, $\lambda^{k+1} = \lambda^k + \alpha_k\Delta\lambda^k$, $\mu^{k+1} = \mu^k + \alpha_k\Delta\mu^k$.
   Nakonec $k = k + 1$.

   (b) *v případě, že je splněno, položíme* $x^* = x^k$, $\lambda^* = \lambda^k$ *a* $\mu^* = \mu^k$ *a ukončíme algoritmus.*

V MATLABU jsem naprogramoval funkci **SQPE_BFGS_M**. Tato funkce je určena pro výpočet úloh pouze s rovnostními omezeními, pro usnadněji je použita aproximace hessiánu pomocí formule BFGS a testování vhodnosti použitého kroku pomocí merit funkce (4.2). V tomto algoritmu navíc musíme zadat funkci f a dva parametry potřebné pro merit funkci η a τ.

```
function [x0,mi,k]=SQPE_BFGS_M(x0,B,mi,f,Gf,Ro,GRo,tol,eta,tau)
%Vstupy
%x0 - počáteční bod
%mi - počáteční hodnota multiplikátoru, pro rovnostní omezení
```

```
%Gf - gradient funkce
%Ro - rovnostní omezení
%GRo - gradient rovnostních omezení
%tol - tolerance ukončovacího kritéria
%eta,tau - parametry merit funkce

%Výstupy
%x0 - bod řešení dané úlohy
%mi - odpovídající hodnota multiplikátoru pro rovnostní omezení
%k - počet iterací

imax=100;
k=0;
if (nargin<8)
   tol=1e-6;
end
if (nargin<9)
   eta=0.5;
   else
    if eta>0.5 || eta<=0
        error('Eta musí být z intervalu (0,0.5)')
    end
end
if (nargin<10)
   tau=0.9;
else
    if tau>=1 || tau<=0
        error('Tau musí být z intervalu (0,1)')
    end
end
```

```
while k<=imax
    [p,p_mi]=nulprostor(B,-Gf(x0),GRo(x0).',-Ro(x0));
    if norm(p)<tol
        break
    else
    theta=max(abs(mi))+1;
    if f(x0+p)+theta*(abs(Ro(x0+p)))>f(x0)+theta*(abs(Ro(x0)))
    +eta*(Gf(x0).'*p-theta*norm(Ro(x0),1))
    alfa=tau;
    else
        alfa=1;
    end
    x=x0+alfa.*p;
    mi=mi+alfa.*p_mi;

    y=Gf(x)+(GRo(x)*mi)-(Gf(x0)+(GRo(x0)*mi));
    B=B+((y*(y)')/((y)'*p))-((B*p*(p)'*B)/((p)'*B*p));
    x0=x;
    if k==imax
        warning('Dosaženo maximálního počtu iterací')
        break
    else
    k=k+1;
    end
    end
end
end
```

Uvedený algoritmus vyzkoušíme na příkladě 3.3, abychom mohli porovnat výsledky.

Příklad 4.2. *[6] Máme úlohu*

$$\begin{cases} \textit{minimalizovat } (x_1-2)^4+(x_1-2x_2)^2 \\ \textit{za podmínky} \quad x_1^2-x_2=0 \end{cases}$$

Jako počáteční aproximace je použito $x_0=(2,2)^T$ *a* $\mu^0=1$. *Pro usnadnění použijeme aproximaci hessiánu pomocí formule BFGS a zadáme počáteční aproximaci*

$$B_0=\begin{pmatrix}1&0\\0&1\end{pmatrix}.$$

Pomocná úloha bude mít stejný tvar a to

$$\begin{cases} \textit{minimalizovat } \frac{1}{2}p^T\begin{pmatrix}1&0\\0&1\end{pmatrix}p+\begin{pmatrix}-4\\8\end{pmatrix}^T p \\ \textit{za podmínky} \quad \begin{pmatrix}4\\-1\end{pmatrix}p+2=0 \end{cases}$$

Pomocnou úlohu vypočítáme pomocí metody nulového prostoru a výsledek bude:
$k=0: \quad p^0=(-2.1176,\,-6.4706)^T,\ \Delta\mu=1.5294.$
V této fázi dopočítáme hodnotu merit funkcí, pro $\theta^0=2$ *a krok* $\alpha_0=1$,

$$\Phi(x^0+\alpha_0p^0;\theta^0)=f(x^0+\alpha_0p^0)+\theta^0|h_1(x^0+\alpha_0p^0)|=106.9336,$$

$$\Phi(x^0;\theta^0)=8,$$

$$D(\Phi(x^k;\theta);p_k)=-47.2941.$$

Otestujeme podmínku $\Phi(x^0+\alpha_0p^0;\theta^0)>\Phi(x^0;\theta^0)+\eta\alpha D\Phi(x^0;p^0)$, *můžeme říct že podmínka platí a proto upravíme hodnotu kroku* $\alpha_0=\tau\alpha_0$, *kde* $\tau\in(0,\,1)$. *V našem případě bude* $\alpha_0=0.95$ *a dopočítáme nový bod* $x^1=(-0.0118,\,-4.1471)^T$ *a odpovídající hodnotu multiplikátoru* $\mu_1=2.3765$. *Další iterace budou ve tvaru*
$k=1: \quad p^1=(2.2044,\,4.2473)^T,\ \Delta\mu=-3.7306,\ x^2=(2.0585,\,-0.2559)^T,$
$\mu_2=-0.9811,$
$k=2: \quad p^2=(-0.6017,\,1.8187)^T,\ \Delta\mu=-1.5289,\ x^3=(1.4989,\,1.7091)^T,$
$\mu_3=-2.3571,$

$\vdots$

$k = 14: \quad p^{14} = (0,\, 0)^T,\ \Delta\mu = 3.3707,\ x^{15} = (0.9456,\, 0.8941)^T,$

$\mu_{15} = 34.0313.$

Při použití naprogramované funkce ***SQPE_BFGS_M*** *v MATLABU získáme výsledek ve tvaru*

```
[x0,mi,k]=SQPE_BFGS_M([2;2],[1 0;0 1],1,@(x)(x(1)-2)^4+(x(1)-2*x(2))^2,
@(x)[4*((x(1)-2)^3)+2*(x(1)-2*x(2));-4*x(1)+8*x(2)],@(x)(x(1)^2)-x(2),
@(x)[2*x(1);-1])

x0 =
    0.9456
    0.8941

mi =
   34.0313

k =
    15
```

Porovnáme-li tento příklad s příkladem 4.1, použití merit funkce nám algoritmus zpomalilo. V průběhu algoritmu je velký problém s použitím parametrů η a τ. Máme k dispozici pouze intervaly, ze kterých máme tyto parametry určit, ale konkrétní volba může být problematická. Z tohoto důvodu by bylo třeba hlubší analýzy pro nalezení vhodné kombinace parametrů pro konkrétní úlohu, abychom mohli zrychlit konvergenci algoritmu.

Kapitola 5

Konvergence

Jak je několikrát v práci zmíněno, tato kapitola je zaměřena na konvergenci SQP metody. Zabývat se zde budeme pouze úlohami s omezeními tvaru rovností. Obecně u iteračních algoritmů nám vyvstává otázka za jakých podmínek bude generovaná posloupnost $\{x^k\}$ blížit k přesnému řešení a právě odpovědí jsou konvergenční věty. Konvergenční věty jsou převzaty z [1]. V každém případě chceme, aby algoritmy zkonvergovali k přesnému řešení, což nemusí být obecně zaručeno. V našem případě je pro zavedení konvergenční věty nutné uvést předpoklady

(p1) Nechť v bodě řešení x^* úlohy nelineárního programování s omezeními tvaru rovností s odpovídajícími Lagrangeovými multiplikátory μ^* má $A(x)$, Jakobián omezujících podmínek, plnou řádkovou hodnost a hessián Lagrangeovy funkce $\nabla^2_{xx} L(x^*, \mu^*, \lambda^*)$ je pozitivně definitní na kritickém kuželu.

(p2) Posloupnosti $\{B_k\}$ a $\{B_k^{-1}\}$ jsou omezeny, to znamená, že existuje konstanta β taková, že

$$\|B_k\| \leq \beta, \quad \|B_k^{-1}\| \leq \beta, \quad \forall k.$$

Věta 5.1. *[1, Věta 18.4] Pokud platí předpoklad (p1) a že f a h jsou dvakrát diferencovatelné s Lipschitzovsky spojitými druhými derivacemi, v okolí bodu (x^*, μ^*). Pak pokud (x^0, μ^0) je dostatečné blízko (x^*, μ^*). Dvojice (x^k, μ^k) generované Algoritmem 3.1 konvergují kvadraticky k (x^*, μ^*).*

Důkaz této konvergenční věty plyne přímo z principu odvození SQP metody pomocí Newtonovy metody pro systém nelineárních rovnic. Jak můžeme vidět v případě splnění předpokladů je konvergence závislá převážně na poloze počátečního bodu.

Druhou verzí SQP algoritmu je nahrazení hessiánu lagrangeovy funkce kvazi-newtonovou aproximací B_k. V tomto případě máme k dispozici dvě konvergenční věty. První z nich požaduje platnost pouze prvního předpokladu (p1) a že B_k je pouze obecnou kvazi-newtonovou aproximací splňující níže uvedenou limitní podmínku.

Věta 5.2. *[1, Věta 18.5] Předpokládejme, že platí předpoklad (p1) a posloupnost* $\{x^k\}$ *generovaná Algoritmem 4.1, s kvazi-newtonovskou aproximací hessiánu* B_k*, konverguje k* x^**. Potom* $\{x^k\}$ *konverguje superlineárně tehdy a jen tehdy když matice* B_k *splňuje*

$$\lim_{k\to\infty} \frac{\|P_k(B_k - \nabla^2_{xx}L(x^*,\mu^*,\lambda^*))(x_{k+1}-x_k)\|}{\|x_{k+1}-x_k\|} = 0, \tag{5.1}$$

kde P_k *je matice projekce na nulový prostor* $A(x^k)$*,*
$P_k = I - A(x^k)^T\left[A(x^k)A(x^k)^T\right]^{-1}A(x^k)$.

Druhá konvergenční věta předpokládá splnění obou předpokladů (p1) a (p2) a použití BFGS formule pro aktualizaci matice B_k. Její splnění je závislé nejen na poloze počátečního bodu, ale i na počáteční aproximaci matice B_0.

Věta 5.3. *[1, Věta 18.6] Předpokládejme, že* $\nabla^2_{xx}L(x^*,\mu^*,\lambda^*)$ *a* B_0 *jsou symetrické a pozitivně definitní, a jsou splněny předpoklady (p1) a (p2). Pokud* $\|x^0-x^*\|$ *a* $\|B_0 - \nabla^2_{xx}L(x^*,\mu^*,\lambda^*))\|$ *jsou dostatečně malé, pak posloupnost* $\{x^k\}$ *generovaná Algoritmem 4.1 s BFGS aproximací* B_k *definovanou (4.1) splňuje limitu (5.1). Tedy posloupnost* $\{x^k\}$ *konverguje superlineárně k* x^**.*

Z jednotlivých konvergenčních vět můžeme udělat závěr, že konvergence je závislá hlavně na počátečních aproximacích. Je třeba, aby se co nejvíce blížili skutečným hodnotám.

Uvedené konvergenční věty pojednávají nejen o konvergenci posloupnosti iterací k přesnému řešení, ale také o rychlosti konvergence. V prvním případě se jedná o konvergenci kvadratickou, metoda SQP je metoda druhého řádu. Pokud bychom chteli řád konvergence pozorovat empiricky, můžeme vyjít ze vztahu pro řád konvergence.

Definice 5.1. *[9, Definice 10.1] Řekneme, že iterační metoda $x^{k+1} = x^k + \alpha p$ je metoda řádu q, pro $q \geq 1$, $q \in \mathbb{N}$ jestliže*

$$\lim_{k\to\infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|^q} = c \neq 0,\ c \in \mathbb{R}$$

za předpokladu $\lim_{k\to\infty} x^k = x^*$.

Pro praktické využití provedeme odvození. Pro velké k můžeme psát

$$\frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|^q} \approx c \approx \frac{\|x^{k+2} - x^*\|}{\|x^{k+1} - x^*\|^q}.$$

Lehkou úpravou získáme vztah pro přibližný výpočet koeficientu q

$$q \approx \frac{ln\left(\frac{\|x^{k+2}-x^*\|}{\|x^{k+1}-x^*\|}\right)}{ln\left(\frac{\|x^{k+1}-x^*\|}{\|x^{k}-x^*\|}\right)}.$$

Řád konvergence budeme pozorovat na příkladu 3.4, u kterého známe přesné řešení.

Příklad 5.1. *Pro příklad 3.4 jsme metodou SQP vygenerovali následující posloupnost iterací*

$$x^0 = (2,\ 2)^T$$
$$x^1 = (1.25,\ 1)^T$$
$$x^2 = (1.025,\ 1)^T$$
$$x^3 = (1.0003,\ 1)^T$$
$$x^4 = (1,\ 1)^T$$

Bod řešení je $x^* = (1,\, 1)^T$. *Postupně dosadíme do vztahu pro koeficient* q.

$$k = 0: \quad q \approx \frac{ln\frac{\|x^2 - x^*\|}{\|x^1 - x^*\|}}{ln\frac{\|x^1 - x^*\|}{\|x^0 - x^*\|}} = 1.1604$$

$$k = 1: \quad q \approx \frac{ln\frac{\|x^3 - x^*\|}{\|x^2 - x^*\|}}{ln\frac{\|x^2 - x^*\|}{\|x^1 - x^*\|}} = 1.8775$$

$$k = 2: \quad q \approx \frac{ln\frac{\|x^4 - x^*\|}{\|x^3 - x^*\|}}{ln\frac{\|x^3 - x^*\|}{\|x^2 - x^*\|}} = 1.9917$$

Vzhledem k tomu, že ve čtvrté iteraci dostáváme přesné řešení, nemůžeme vyčíslit více iterací pro koeficient q. *Přesto můžeme pozorovat že poslední hodnota* q *je přibližně rovna* 2, *proto můžeme říct, že algoritmus konverguje kvadraticky, jak uvádí konvergenční věta 5.1.*

Závěr

Cílem práce bylo nastudovat Sekvencionální kvadratické programování, jenž se v rámci mého studia neprobíralo a sestavit kódy v MATLABU. SQP metoda je jednou z nejúčinějších metod nelineárního programování pro řešení obecných úloh nelineárního programování. Tato metoda je nejvhodnější pro úlohy nelineárního programování s rovnostními omezeními. Stejně tak je možné ji použít i pro obecné úlohy s oběma typy omezujících podmínek. Jedná se o zajímavou metodu pro řešení úloh nelineárního programování, jejíž idea zní jednoduše, ale prakticky tak snadná není.

Pro porozumění této problematiky je třeba si projít základy nelineárního programování a kvadratické programování, které je nutné pro řešení pomocných úloh metody SQP. Obě rozebrané metody, metodu nulového prostoru a metodu aktivní množiny, jsem demonstroval na příkladech a doplnil o kódy v MATLABU. Kdy metodu aktivní množiny jsem vyzkoušel, jak na kvadratických úlohách s omezeními tvaru nerovnostní, tak i na obecné kvadratické úloze.

Při studiu metody SQP jsem se seznámil se dvěma přístupy, jak je možné pohlížet na odvození základního algoritmu. První přístup nám vysvětluje důvod použití lagrangeovy funkce a druhý přístup základní podmínky konvergence. Po odvození základního algoritmu pro úlohy nelineárního programování s omezeními tvaru rovností a obecné úlohy úlohy nelineárního programování, bylo možné je vyzkoušet na příkladech a pokud je to možné, tyto úlohy jsou vykresleny na obrázcích.

Následně jsem vyzkoušel modifikace základního algoritmu. Prvním krokem byla kvazi-newtonovská aproximace hessiánu a druhým testování vhodnosti kroku

za použití merit funkce. Tyto úpravy jsem otestoval na příkladech, které byly představeny u základních algoritmů, aby bylo možné tyto postupy porovnat. Všechny naprogramované kódy v MATLABU jsou k dispozici na přiloženém CD.

V poslední kapitole jsem se seznámil s konvergenčními větami pro základní algoritmy a pro algoritmus s formulí BFGS. Předpoklady pro tyto věty jsou známe i z jiných metod, hlavně Newtonovy metody, která je stejně závislá na poloze počátečního bodu.

Literatura

[1] NOCEDAL, Jorge, WRIGHT, J., Stephen: *Numerical Optimization*. Springer, New York, 1999.

[2] MACHALOVÁ, Jitka, NETUKA, Horymír: *Numerické metody nepodmíněné optimalizace*. UPOL, Olomouc, 2013.

[3] MACHALOVÁ, Jitka, NETUKA, Horymír: *Nelineární programování: Teorie a metody*. UPOL, Olomouc, 2013.

[4] BURDEN, L., Richard, FAIRES, J., Douglas: *Numerical Analysis*. Brooks/Cole, Boston, 2011.

[5] SCHITTKOWSKI, Klaus: *Test Examples for Nonlinear Programming Codes*. University of Bayreuth, Bayreuth, 2009.

[6] BAZARAA, S., Mokhtar, SHERALI, D., Hanif, SHETTY, M.,C.: *Nonlinear programming: Theory and Algorithms*. Wiley, 2006.

[7] COTTLE, W., Richard, THAPA, N., Mukund: *Linear and Nonlinear Optimization*. Springer, 2017

[8] HenryWang – Domovská stránka [online]. [cit. 2022-28-03]. Dostupné z: https://henrywang.nl/another-quadratic-programming-example-with-r/

[9] MACHALOVÁ, Jitka: *Základy numerických metod*. UPOL, Olomouc, 2014.