UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

DIPLOMOVÁ PRÁCE

Řešení úlohy kvadratického programování pomocí metod vnitřních bodů

Vedoucí diplomové práce:
RNDr. Horymír Netuka, Ph.D.
Rok odevzdání: 2010

Vypracovala:
Bc. Karolína Macková
AME, II. ročník

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracovala samostatně pod vedením pana RNDr. Horymíra Netuky, Ph.D. s použitím uvedené literatury.

V Olomouci dne 8. dubna 2010

Poděkování

Na tomto místě bych chtěla poděkovat především svému vedoucímu diplomové práce panu RNDr. Horymíru Netukovi, Ph.D., že měl se mnou dostatek trpělivosti, aby mi pomohl dovést tuto práci ke zdárnému konci. Také bych ráda poděkovala své rodině a přátelům, že mě po celou dobu studia podporovali.

Obsah

Úvod

Cílem této práce je seznámení s metodami vnitřních bodů. Zaměříme se jak na část teoretickou, tak praktickou a budeme se zabývat úlohou kvadratického programování, speciálně úlohou konvexní. V praktické části využijeme především možnost řešení v matematickém programu Matlab a programovacím jazyce Fortran 77.

Metody vnitřních bodů jsou v současné době velmi populární oblastí optimalizace. Základem byl Karmarkarův článek z roku 1984, ve kterém byla poprvé uvedena projektivní metoda vnitřního bodu. Na rozdíl od simplexové metody, která je známá od 40. let 20. století a která má exponenciální složitost, dosahuje tato metoda pouze polynomiální složitosti a má také dobré výsledky v praxi. Metody vnitřních bodů jsou velmi efektivní při řešení rozsáhlých problémů. Nejprve se aplikovaly na úlohu lineárního programování a později na úlohu kvadratického programování. Největší zájem teoretiků vzbuzuje primárně-duální metoda vnitřních bodů, které se věnujeme v naší práci.

V první kapitole uvádíme základní definice z oblasti optimalizace a matematické analýzy. Jde například o pojmy: gradient, Hesián, globální minimum, atd..

Druhá kapitola popisuje úlohu kvadratického programování, a to jak s rovnostními, tak s nerovnostními podmínkami. Uvádíme zde také základní tvar úlohy nelineárního programování a Karush-Kuhn-Tuckerovy podmínky.

Ve třetí kapitole se zabýváme řešením úlohy kvadratického programování pomocí primárně-duální metody, která je jednou z metod vnitřních bodů. Třetí kapitola je rozdělena do několika podkapitol, ve kterých se zabýváme jednotlivými částmi výpočtu např.: startovacím bodem, délkou kroku α, ukončením iteračního procesu, atd..

Čtvrtá kapitola popisuje řešení úlohy kvadratického programování pomocí metody logaritmické bariérové funkce. Jelikož po všech úpravách získáme stejnou soustavu jako v předchozí metodě, odkážeme se po odvození na kapitolu třetí.

V páté kapitole se zaměříme na metody řešení soustav rovnic vznikajících při použití metody vnitřních bodů. Uvádíme zde metodu Schurova komplementu.

Tyto soustavy je však možné řešit i pomocí klasické Gaussovy eliminační metody s částečným výběrem prvku.

Šestá kapitola obsahuje řešené příklady, u kterých jsou uvedeny počty iterací potřebných k dosažení řešení při požadované přesnosti a při použití různých metod výpočtu. Ve dvou podkapitolách dále uvádíme stručný popis programového zpracování algoritmu v programu Matlab a základní příkazy programovacího jazyka Fortran 77.

V přílohách jsou uvedena programová zpracování algoritmu pro řešení úlohy kvadratického programování s podmínkami ve tvaru rovností a nerovností pomocí primárně-duální metody v programu Matlab a v programu vytvořeném programovacím jazykem Fortran 77.

Vypracované programy jsou přiloženy na CD.

1. Základní definice

V této kapitole se krátce seznámíme s pojmy, které budeme používat při řešení optimalizačních problémů.

Definice 1. *Kruhové okolí bodu x o poloměru ϵ* je definováno jako

$$B(x, \epsilon) = \{y \in R^n : 0 \leq ||x - y||_2 < \epsilon\}.$$

Definice 2. Řekneme, že bod $\hat{x}$ je bodem *lokálního minima* úlohy, jestliže $\hat{x} \in S$ a existuje $\epsilon > 0$ takové, že platí

$$f(\hat{x}) \leq f(x) \quad \forall x \in S \cap B(\hat{x}, \epsilon).$$

Definice 3. Řekneme, že bod $\hat{x}$ je bodem *globálního minima* úlohy, jestliže $\hat{x} \in S$ a platí

$$f(\hat{x}) \leq f(x) \quad \forall x \in S.$$

Definice 4. Množina $X \subseteq R^n$ se nazývá *konvexní*, jestliže platí

$$\lambda x + (1 - \lambda)y \in X \quad \forall x, y \in X, \lambda \in (0, 1).$$

Definice 5. *Funkce* $f : X \to R$ se nazývá *konvexní*, jestliže

1. X *je konvexní*
2. $f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \quad \forall x, y \in X, x \neq y, \lambda \in (0, 1).$

Definice 6. *Funkce* $f : X \to R$ se nazývá *ryze konvexní*, jestliže

1. X *je konvexní*
2. $f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda)f(y) \quad \forall x, y \in X, x \neq y, \lambda \in (0, 1).$

Definice 7. Nechť má funkce f v bodě x parciální derivace 1. řádu. *Gradientem funkce* f v bodě x nazýváme vektor

$$\bigtriangledown f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} (x).$$

Definice 8. Nechť má funkce f v bodě x parciální derivace 2. řádu. *Hesiánem funkce f* nazýváme symetrickou matici

$$H(x) = (h_{ij})_{i,j=1}^{n}(x) \quad , \; kde \; h_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Definice 9. *Symetrická pozitivně definitní matice* je taková symetrická čtvercová matice, jejíž vlastní čísla jsou větší než nula.

Definice 10. *Symetrická pozitivně semidefinitní matice* je taková symetrická čtvercová matice, jejíž vlastní čísla jsou větší nebo rovna nule.

Definice 11. *Normu vektoru x* můžeme vyjádřit jako

$$||x||_\infty = \max_{1 \leq i \leq n} |x_i|.$$

Tato norma se nazývá *krychlová norma vektoru x.*

Definice 12. *Normu vektoru x* můžeme vyjádřit jako

$$||x||_2 = \Big(\sum_{i=1}^{n} x_i^2 \Big)^{\frac{1}{2}}.$$

Tato norma se nazývá *eukleidovská norma vektoru x.*

Definice 13. Mějme funkce $f_i(x_1, \ldots, x_n)$ pro $i = 1, \ldots, m$, které mají parciální derivace $\frac{\partial f_i}{\partial x_k}$. Pak *Jacobiho matice* je definována jako

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

2. Optimalizace s podmínkami

Základní úlohu optimalizace s podmínkami můžeme zapsat následovně

$$\begin{aligned} &\textit{minimalizovat} \quad f(x) \\ &\textit{za podmínek} \quad g_i(x) \geq 0, \quad i \in I \\ &\qquad\qquad\quad h_j(x) = 0, \quad j \in J, \end{aligned} \tag{1}$$

kde f, g_i a h_j jsou funkce a I a J jsou konečné množiny indexů. Funkce f se nazývá *účelová funkce*, g_i, $i \in I$, jsou *nerovnostní omezení* a h_j, $j \in J$, jsou *rovnostní omezení*. Při optimalizaci s podmínkami hledáme vázané extrémy, tj. body účelové funkce, které jsou minimální v oblasti omezené jistými podmínkami.

Úlohou nelineárního programování nazveme úlohu, ve které je alespoň jedna z funkcí f, g_i a h_j nelineární.

Zavedeme doplňkový vektor $y \geq 0$ a dostaneme úlohu

$$\begin{aligned} &\textit{minimalizovat} \quad f(x) \\ &\textit{za podmínek} \quad g_i(x) - y_i = 0, \quad i \in I \\ &\qquad\qquad\quad h_j(x) = 0, \quad j \in J \\ &\qquad\qquad\quad y \geq 0. \end{aligned} \tag{2}$$

Definice 14. *Přípustná množina* úlohy (2)

$$S = \{(x, y) | g_i(x) - y_i = 0, \quad y_i \geq 0, \quad i \in I; \quad h_j(x) = 0, \quad j \in J\}.$$

Definice 15. *Striktně přípustná množina* úlohy (2)

$$S^0 = \{(x, y) | g_i(x) - y_i = 0, \quad y_i > 0, \quad i \in I; \quad h_j(x) = 0, \quad j \in J\}.$$

Definice 16. *Aktivní množina* $A(x^0)$ obsahuje indexy rovnostních omezení z J a indexy nerovnostních omezení i, pro které $g_i(x^0) = 0$, tj.

$$A(x^0) = J \cup \{i \in I | g_i(x^0) = 0\}.$$

Definice 17. *Lagrangeovou funkcí* příslušející úloze (1) rozumíme funkci

$$\mathcal{L}(x, \lambda, \nu) = f(x) - \sum_{i \in I} \lambda_i g_i(x) - \sum_{j \in J} \nu_j h_j(x) \tag{3}$$

$$= f(x) - g(x)^T \lambda - h(x)^T \nu. \tag{4}$$

Definice 18. *LICQ-kvalifikační podmínky lineární nezávislosti*
Gradienty podmínek aktivních v bodě x^* jsou lineárně nezávislé, tj. $\bigtriangledown g_i(x^*)$, $i \in I = \{i : g_i(x^*) = 0\}$ a $\bigtriangledown h_j(x^*), j = 1, \ldots, l$ jsou lineárně nezávislé.

Věta 1. *Nutné podmínky 1. řádu*
Předpokládejme, že x^* je lokální minimum úlohy, kde funkce f, g_i a h_j jsou spojitě diferencovatelné a platí LICQ v x^*. Pak existují vektory Lagrangeových multiplikátorů λ^* a ν^* takové, že platí následující podmínky v (x^*, λ^*, ν^*)

$$\bigtriangledown_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \tag{5}$$

$$g_i(x^*) \geq 0, \quad \forall i \in I \tag{6}$$

$$h_j(x^*) = 0, \quad \forall j \in J \tag{7}$$

$$\lambda_i^* \geq 0, \quad \forall i \in I \tag{8}$$

$$\lambda_i^* g_i(x^*) = 0, \quad \forall i \in I. \tag{9}$$

Důkaz: Viz. [8].

Tyto podmínky se nazývají *Karush-Kuhn-Tuckerovy podmínky* nebo také *KKT podmínky*. Podmínky $\lambda_i^* g_i(x^*) = 0$, $\forall i \in I$, jsou *podmínky komplementarity*.

Definice 19. Vektory λ a ν se nazývají *Lagrangeovy multiplikátory* příslušející odpovídajícím omezujícím podmínkám.

Definice 20. Úloha

$$\begin{aligned} \textit{minimalizovat} \quad & f(x) \\ \textit{za podmínek} \quad & g_i(x) \geq 0, \quad i \in I \\ & h_j(x) = 0, \quad j \in J \\ & x \in X \end{aligned}$$

se nazývá *úlohou konvexního programování*, jestliže $f(x)$ je konvexní funkce, $g_i(x)$, $i \in I$, jsou konkávní funkce, $h_j(x)$, $j \in J$, jsou lineární funkce a X je otevřená konvexní množina.

Důsledek 1. KKT podmínky jsou pro úlohu konvexního programování nutné i postačující.

Poznámka 1. *Vlastnosti úlohy konvexního programování*

1. každé lokálně optimální řešení je globálně optimální,
2. je-li množina optimálních řešení neprázdná, je konvexní,
3. je-li $f(x)$ ryze konvexní, má úloha nejvýše jedno optimální řešení.

2.1. Kvadratické programování

Optimalizační problém, ve kterém je účelová funkce kvadratická a omezující podmínky jsou lineární, se nazývá *kvadratické programování*. Úlohu kvadratického programování můžeme zapsat jako:

$$\textit{minimalizovat} \quad f(x) = \frac{1}{2}x^T Gx + x^T c \tag{10}$$

$$\textit{za podmínek} \quad a_i^T x \geq b_i \quad i \in I = \{1, \ldots, m\} \tag{11}$$

$$e_j^T x = f_j \quad j \in J = \{1, \ldots, l\}, \tag{12}$$

kde G je symetrická $n \times n$ matice. Jestliže matice G je pozitivně semidefinitní, pak se jedná o úlohu *konvexního* kvadratického programování a řešení problému je podobné lineárnímu programování. Je-li matice G pozitivně definitní, pak jde o úlohu *ryze konvexního* kvadratického programování. *Nekonvexním* kvadratickým programováním se nebudeme zabývat. Úloha kvadratického programování patří do teorie nelineárního programování, jelikož účelová funkce je kvadratická.

2.1.1. Úloha kvadratického programování s podmínkami ve tvaru rovností

Uvažujme úlohu

$$\textit{minimalizovat} \quad f(x) = \frac{1}{2}x^T Gx + x^T c$$

$$\textit{za podmínek} \quad e_j^T x = f_j \quad j \in J = \{1, \ldots, l\}.$$

Tuto úlohu můžeme zapsat maticově jako:

$$\textit{minimalizovat} \quad f(x) = \frac{1}{2}x^T Gx + x^T c \tag{13}$$

$$\textit{za podmínek} \quad Ex = f, \tag{14}$$

kde E je Jacobiho matice typu $l \times n$ ($l \leq n$), jejíž řádky jsou e_j^T, $j \in J$, a $f \in R^l$ je vektor, jehož složky jsou f_j, $j \in J$.

Sestrojíme Lagrangeovu funkci příslušející úloze (13)-(14)

$$\mathcal{L}(x,\nu) = \frac{1}{2}x^T Gx + x^T c - (Ex - f)^T \nu.$$

Nutné podmínky prvního řádu mají tvar

$$\begin{aligned} Gx + c - E^T\nu &= 0 \\ Ex - f &= 0. \end{aligned}$$

Maticově je můžeme zapsat následovně

$$\begin{bmatrix} G & -E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} -c \\ f \end{bmatrix}, \tag{15}$$

kde ν je vektor *Lagrangeových multiplikátorů*. Matice (15) se nazývá *Karush-Kuhn-Tuckerova matice*.

2.1.2. Úloha kvadratického programování s podmínkami ve tvaru nerovností

Uvažujme následující úlohu

$$\begin{aligned} \textit{minimalizovat} \quad & f(x) = \frac{1}{2}x^T Gx + x^T c \\ \textit{za podmínek} \quad & a_i^T x \geq b_i, \quad i \in I = \{1, \ldots, m\}. \end{aligned}$$

Maticově ji můžeme zapsat následovně:

$$\textit{minimalizovat} \quad f(x) = \frac{1}{2}x^T Gx + x^T c \tag{16}$$

$$\textit{za podmínek} \quad Ax \geq b, \tag{17}$$

kde matice A je typu $m \times n$ a vektor $b \in R^m$.

Zavedeme-li doplňkový vektor $y \geq 0$, dostaneme úlohu

$$\begin{aligned} \textit{minimalizovat} \quad & f(x) = \frac{1}{2}x^T Gx + x^T c \\ \textit{za podmínek} \quad & Ax - y = b \\ & y \geq 0. \end{aligned} \tag{18}$$

Sestrojíme Lagrangeovu funkci příslušející úloze (18)

$$\mathcal{L}(x, y, \lambda) = \frac{1}{2} x^T G x + x^T c - \sum_{i \in I} \lambda_i (a_i^T x - b_i - y_i) \tag{19}$$

$$= \frac{1}{2} x^T G x + x^T c - (Ax - b - y)^T \lambda. \tag{20}$$

Nyní napíšeme KKT podmínky pro tuto úlohu

$$Gx + c - A^T \lambda = 0 \tag{21}$$

$$Ax - b - y = 0 \tag{22}$$

$$y_i \lambda_i = 0, \quad i = 1, \ldots, m \tag{23}$$

$$(y, \lambda) \geq 0. \tag{24}$$

Jak jsme uvedli v předchozím textu, jestliže G je pozitivně semidefinitní (popř. pozitivně definitní), pak jsou tyto KKT podmínky nejen nutné, ale také postačující.

KKT podmínky (21)-(23) můžeme přepsat následovně

$$F(x, y, \lambda) = \begin{bmatrix} Gx + c - A^T \lambda \\ Ax - b - y \\ Y \Lambda e \end{bmatrix} = 0, \tag{25}$$

kde $Y = diag(y_1, \ldots, y_m)$, $\Lambda = diag(\lambda_1, \ldots, \lambda_m)$ a $e = (1, \ldots, 1)^T$.

Definice 21. *Striktně přípustná množina* úlohy (18)

$$\mathcal{F}^0 = \{(x, y, \lambda) | Ax - b - y = 0, Gx + c - A^T \lambda = 0, (y, \lambda) > 0\}. \tag{26}$$

3. Řešení úlohy kvadratického programování pomocí primárně-duální metody

Primárně-duální metoda počítá řešení KKT podmínek pomocí Newtonovy metody, kterou aplikujeme na soustavu (21)-(23). Délka kroku α musí být určena tak, aby platila podmínka $(y, \lambda) > 0$. Proto se tato metoda nazývá *metoda vnitřních bodů*.

Nejprve se zaměříme na řešení úlohy kvadratického programování s podmínkami ve tvaru nerovností a později uvedeme rozšíření na úlohu kvadratického programování s podmínkami ve tvaru rovností a nerovností.

V Newtonově metodě řešíme soustavu $F = 0$ linearizací okolo aktuálního bodu a směr $(\triangle x, \triangle y, \triangle \lambda)$ získáme řešením lineární soustavy

$$J(x, y, \lambda) \begin{pmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \end{pmatrix} = -F(x, y, \lambda), \tag{27}$$

kde $J(x, y, \lambda)$ je Jacobiho matice funkce F.
Dostáváme tedy

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e \end{bmatrix}, \tag{28}$$

kde

$$r_d = Gx + c - A^T \lambda \tag{29}$$

$$r_p = Ax - b - y. \tag{30}$$

Další iteraci získáme jako

$$(x^+, y^+, \lambda^+) = (x, y, \lambda) + \alpha(\triangle x, \triangle y, \triangle \lambda), \tag{31}$$

kde $\alpha \in (0, 1]$ je zvolená tak, aby byla zachována nerovnost $(y^+, \lambda^+) > 0$.

Délka kroku v metodě s čistě Newtonovým směrem je často velmi malá a zlepšení směrem k optimálnímu řešení je tak velmi pomalé.

Proto primárně-duální metoda upravuje základní postup Newtonovy metody následovně:

- upravuje směr hledání směrem dovnitř nezáporného kvadrantu $(y, \lambda) \geq 0$, což umožní delší krok v tomto směru, než dojde k porušení podmínek $(y, \lambda) > 0$,
- zabraňuje komponentám (y, λ), aby se přiblížily k hranici kvadrantu.

3.1. Metoda sledování cesty

Centrální cestou rozumíne křivku, která je parametrizována kladným parametrem τ, tj. $\tau > 0$.

Definice 22. *Centrální cesta* je množina bodů $C = \{(x_\tau, y_\tau, \lambda_\tau) | \tau > 0\}$, které získáme řešením soustavy

$$Gx + c - A^T\lambda = 0 \tag{32}$$

$$Ax - b - y = 0 \tag{33}$$

$$y_i\lambda_i = \tau, \quad \forall i \in I \tag{34}$$

$$(y, \lambda) > 0 \tag{35}$$

nebo pomocí funkce F

$$F(x_\tau, y_\tau, \lambda_\tau) = \begin{pmatrix} 0 \\ 0 \\ \tau e \end{pmatrix}$$

$$(y_\tau, \lambda_\tau) > 0.$$

Tyto podmínky se liší od KKT podmínek pouze veličinou τ. Místo splnění podmínky komplementarity požadujeme, aby součin $y_i\lambda_i$ měl stejnou hodnotu pro všechny indexy i. Parametr τ je roven součinu $\sigma\mu$, kde $\sigma \in (0, 1)$ je *centrující parametr*, který spočítáme podle vztahu

$$\sigma = \gamma \min\left((1 - r)\frac{1 - \epsilon}{\epsilon}, 2\right)^3, \tag{36}$$

kde $0 < r < 1$ a γ jsou parametry s hodnotami $r = 0.95$ a $\gamma = 0.1$. Vzdálenost od centrální cesty ϵ vypočítáme jako

$$\epsilon = \frac{\min Y\Lambda e}{y^T\lambda/m}. \tag{37}$$

Z předchozího vztahu vidíme, že $0 < \epsilon \leq 1$ a také to, že $\epsilon = 1$ právě tehdy, když hodnoty $y_i\lambda_i$ jsou stejné pro všechna i.

Míra duality μ je definována následovně

$$\mu = \frac{1}{m} \sum_{i=1}^{m} y_i \lambda_i = \frac{y^T \lambda}{m}. \tag{38}$$

Aplikujeme-li Newtonovu metodu na soustavu (32)-(34) a položíme-li $\tau = \sigma\mu$, dostáváme soustavu lineárních rovnic

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e + \sigma \mu e \end{bmatrix}, \tag{39}$$

kde

$$r_d = Gx + c - A^T \lambda \tag{40}$$

$$r_p = Ax - b - y. \tag{41}$$

V úlohách, ve kterých máme k dispozici striktně přípustný počáteční bod, jsou výrazy r_d a r_p rovny nule.

Řešení této úlohy pro kladné hodnoty σ a μ definuje *centrální cestu*. Centrální cesta vede k řešení úlohy, pokud $\tau = \sigma\mu$ konverguje k 0. Parametr τ udává, jak budou jednotlivé iterace posouvány od hranice striktně přípustné oblasti. Čím větší bude hodnota parametru τ, tím více bude iterace posouvána od hranice striktně přípustné oblasti. Algoritmus pro sledování cesty se anglicky nazývá *path following algorithm*.

Jestliže se nacházíme blízko centrální cesty, můžeme parametr τ snížit, protože centrální cesta vede k řešení vnitřkem striktně přípustné oblasti. Naopak, jsme-li daleko od centrální cesty, je vhodné parametr τ zvětšit, abychom se nedostali příliš blízko k hranici striktně přípustné oblasti, což by mohlo způsobit zpomalení, případně zastavení algoritmu.

Pokud se iterace budou blížit k řešení, bude se součin $y^T\lambda$ blížit k nule a parametr τ bude konvergovat k nule.

Centrální cesta vede k řešení podél směru, který udržuje hodnoty y a λ kladné a snižuje hodnotu $y_i\lambda_i, i = 1, \ldots, m$, k nule ve stejném poměru.

Algoritmy sledování centrální cesty omezují iterace na blízkém okolí centrální cesty C a sledují C k řešení úlohy, tj. metody sledování cesty omezují délku kroku tak, aby iterace zůstaly v určitém okolí centrální cesty a nepřiblížily se k hranici kvadrantu $(y, \lambda) > 0$.

3.1.1. Metoda sledování cesty s dlouhým krokem

Jedná se o metodu, která omezuje iterace na okolí

$$\mathcal{N}_{-\infty}(\gamma) = \{(x, y, \lambda) \in \mathcal{F}^0 | y_i \lambda_i \geq \gamma\mu, i = 1, \ldots, m\}, \tag{42}$$

kde parametr $\gamma \in (0, 1)$. Často se parametr γ volí jako $\gamma = 10^{-3}$. Pro bod ležící v $\mathcal{N}_{-\infty}(\gamma)$ musí platit, že každý ze součinů $y_i\lambda_i$ je alespoň malým násobkem hodnoty μ.

Pro tuto metodu je typické, že její širší okolí umožňuje dlouhý krok a rychlejší posun k optimálnímu řešení.

Algoritmus sledování cesty s dlouhým krokem

Dáno $\gamma \in (0, 1), 0 < \sigma_{min} < \sigma_{max} < 1$ a $(x^0, y^0, \lambda^0) \in \mathcal{N}_{-\infty}(\gamma)$;

for $k = 0, 1, 2, ...$

Vybereme $\sigma_k \in [\sigma_{min}, \sigma_{max}]$, vypočítáme $\mu_k = \frac{y^{kT}\lambda^k}{m}$ a vyřešíme soustavu (39);

Poté vypočítáme nové iterace

$$(x^{k+1}, y^{k+1}, \lambda^{k+1}) = (x^k, y^k, \lambda^k) + \alpha(\triangle x^k, \triangle y^k, \triangle \lambda^k),$$

kde $\alpha \in (0, 1]$ je největší délka kroku splňující $(x^{k+1}, y^{k+1}, \lambda^{k+1}) \in \mathcal{N}_{-\infty}(\gamma)$;

end.

V mnoha případech nemáme k dispozici striktně přípustný startovací bod, tudíž musíme použít tzv. nepřípustnou metodu vnitřních bodů.

V případě nepřípustné metody sledování cesty s dlouhým krokem jde o rozšíření okolí $\mathcal{N}_{-\infty}(\gamma)$ na okolí $\mathcal{N}_{-\infty}(\gamma, \beta)$, které je definováno následovně

$$\mathcal{N}_{-\infty}(\gamma, \beta) = \{(x, y, \lambda) | \quad ||(r_d, r_p)|| \leq \frac{||(r_d^0, r_p^0)||}{\mu_0}\beta\mu,$$
$$y_i\lambda_i \geq \gamma\mu, i = 1, \ldots, m, (y, \lambda) > 0\},$$

kde $\gamma \in (0, 1)$ a $\beta \geq 1$ jsou dané parametry a (r_d^0, r_p^0) a μ_0 jsou hodnoty reziduí, resp. míry duality pro startovací bod (x^0, y^0, λ^0).

Algoritmus sledování cesty s dlouhým krokem v případě nepřípustného startovacího bodu

Dáno $\gamma \in (0,1)$, $\beta \geq 1$, $0 < \sigma_{min} < \sigma_{max} \leq 0.5$ a (x^0, y^0, λ^0) tak, že $(y^0, \lambda^0) > 0$;

for $k = 0, 1, 2, ...$

Vybereme $\sigma_k \in [\sigma_{min}, \sigma_{max}]$, vypočítáme $\mu_k = \frac{y^{kT}\lambda^k}{m}$ a vyřešíme soustavu (39);

Poté vypočítáme nové iterace

$$(x^{k+1}, y^{k+1}, \lambda^{k+1}) = (x^k, y^k, \lambda^k) + \alpha(\triangle x^k, \triangle y^k, \triangle \lambda^k),$$

kde $\alpha \in (0,1]$ je největší délka kroku taková, že $(x^{k+1}, y^{k+1}, \lambda^{k+1}) \in \mathcal{N}_{-\infty}(\gamma, \beta)$ a platí Armijova podmínka

$$\mu_{k+1} \leq (1 - 0.01\alpha)\mu_k;$$

end.

Tuto problematiku jsme převzali z [12] a následně prokonzultovali s vedoucím diplomové práce.

Parametr σ_k můžeme počítat podle vztahu (36).

3.1.2. Metoda sledování cesty s krátkým krokem

Tato metoda omezuje iterace na okolí

$$\mathcal{N}_2(\varphi) = \{(x, y, \lambda) \in \mathcal{F}^0 | \quad ||Y\Lambda e - \mu e||_2 \leq \varphi\mu\}, \tag{43}$$

kde parametr $\varphi \in [0, 1)$. Často se parametr φ volí jako $\varphi = 0.4$. Toto okolí omezuje délky kroků tak, aby rozdíl $y_i\lambda_i - \mu$ nebyl příliš velký pro žádné i, tj. aby se žádná složka vektoru (y, λ) nepřiblížila k hranici nezáporného kvadrantu $(y, \lambda) \geq 0$ dřív, než je μ dostatečně malé. Délka kroku je konstantní, tj. $\alpha = 1$.

Pro tuto metodu je typické velmi pomalé přiblížení k optimálnímu řešení.

Algoritmus sledování cesty s krátkým krokem

Máme dáno $\varphi \in [0, 1), \sigma = 1 - 0.4/\sqrt{m}$ a $(x^0, y^0, \lambda^0) \in \mathcal{N}_2(\varphi)$;

for $k = 0, 1, 2, ...$

Řešíme soustavu (39) pro $(\triangle x^k, \triangle y^k, \triangle \lambda^k)$;

Poté vypočítáme nové iterace

$$(x^{k+1}, y^{k+1}, \lambda^{k+1}) = (x^k, y^k, \lambda^k) + (\triangle x^k, \triangle y^k, \triangle \lambda^k);$$

end.

Podrobněji je tato problematika popsána v [12].

3.2. Určení délky kroku α

V této kapitole se budeme zabývat volbou délek kroků: zda volit odlišné délky kroků α^{pri} a α^{dual} užívané pro primární proměnné x, y a duální proměnné λ, ν nebo stejné délky kroků $\alpha = \alpha^{pri} = \alpha^{dual}$. Definujeme-li nové iterace jako

$$(x^+, y^+) = (x, y) + \alpha^{pri}(\triangle x, \triangle y)$$
$$\lambda^+ = \lambda + \alpha^{dual} \triangle \lambda,$$

kde α^{pri} a α^{dual} jsou takové délky kroků, které zajistí, že $(y^+, \lambda^+) > 0$, pak nová rezidua splňují následující vztahy

$$r_d^+ = (1 - \alpha^{dual})r_d + (\alpha^{pri} - \alpha^{dual})G \triangle x$$
$$r_p^+ = (1 - \alpha^{pri})r_p,$$

které lze odvodit z

$$G \triangle x - A^T \triangle \lambda = -r_d = -(Gx + c - A^T\lambda)$$
$$A \triangle x - \triangle y = -r_p = -(Ax - b - y).$$

Jestliže tedy $\alpha = \alpha^{pri} = \alpha^{dual}$, pak obě rezidua zmenšíme lineárně pro všechna $\alpha \in (0, 1)$. Pro určité volby různých délek kroků α^{pri} a α^{dual} se reziduum r_d^+ může zvýšit, což může způsobit divergenci.

Možnou volbou je užít stejné délky kroků a položit $\alpha = min(\alpha_\gamma^{pri}, \alpha_\gamma^{dual})$, kde

$$\alpha_\gamma^{pri} = max\{\alpha \in (0, 1] : y + \alpha \triangle y \geq (1 - \gamma)y\}, \tag{44}$$
$$\alpha_\gamma^{dual} = max\{\alpha \in (0, 1] : \lambda + \alpha \triangle \lambda \geq (1 - \gamma)\lambda\}, \tag{45}$$

kde $\gamma \in (0, 1)$ kontroluje, jak daleko ustoupíme od maximálního kroku, pro který jsou podmínky $y + \alpha \triangle y \geq 0$ a $\lambda + \alpha \triangle \lambda \geq 0$ splněny. Pokud γ konverguje během výpočtu řešení k 1, je možné dosáhnout rychlejší konvergence.

Jeden způsob volby nestejných délek kroků je zvolit $(\alpha^{pri}, \alpha^{dual})$ tak, abychom minimalizovali míru optimality

$$M(x^+, y^+, \lambda^+) = ||Gx^+ + c - A^T\lambda^+||_2^2 + ||Ax^+ - b - y^+||_2^2 + (y^+)^T\lambda^+$$

za podmínek $0 \leq \alpha^{pri} \leq \alpha_\gamma^{pri}$ a $0 \leq \alpha^{dual} \leq \alpha_\gamma^{dual}$.

Délky kroků tedy volíme tak, aby platila podmínka

$$M(x^+, y^+, \lambda^+) \leq M(x, y, \lambda).$$

Viz. [7], [8].

Při odlišných délkách kroků můžeme dosáhnout rychlejší konvergence. Různé délky kroků volíme také proto, aby nedošlo k přílišnému zkracování délek kroků.

Při volbě délek kroků je nutné zajistit, aby proměnné y a λ, které mají zůstat kladné, kladné zůstaly. Spočítáme maximální délky kroků, které můžeme brát pro proměnné y a λ bez porušení nezápornosti, a pak vezmeme délky kroků menší než je toto maximum, ale ne větší než 1. Nyní si nadefinujeme největší délky kroků, pro které je $y + \alpha \triangle y \geq 0$ a $\lambda + \alpha \triangle \lambda \geq 0$.

$$\alpha_{max}^{pri} = \min_{i \in I: \triangle y_i < 0} \left\{ -\frac{y_i}{\triangle y_i} \right\}$$

$$\alpha_{max}^{dual} = \min_{i \in I: \triangle \lambda_i < 0} \left\{ -\frac{\lambda_i}{\triangle \lambda_i} \right\}$$

Délky kroků pro primární a duální proměnné pak vypočítáme jako

$$\alpha_\gamma^{pri} = \min(1, \gamma \alpha_{max}^{pri}) \tag{46}$$

$$\alpha_\gamma^{dual} = \min(1, \gamma \alpha_{max}^{dual}), \tag{47}$$

kde $\gamma \in (0, 1)$ má zajistit, aby některé z pomocných proměnných nebyly rovny nule. Obvykle se hodnota γ pohybuje kolem 0.99.

3.3. Startovací bod

Volba startovacího bodu je v metodách vnitřních bodů velmi důležitá. Pokud použijeme dobrý startovací bod, může být schopnost a rychlost přiblížení k optimálnímu řešení mnohem větší než při užití tzv. horkého startu. Počáteční bod x^0 je většinou součástí zadání. Počáteční bod y^0 spočítáme při dosazení bodu x^0 do podmínky $Ax - b - y = 0$, tj.

$$y^0 = Ax^0 - b. \tag{48}$$

Může se stát, že bod x^0 leží na hranici striktně přípustné oblasti nebo tak blízko hranice, že některé hodnoty y_i^0 by mohly být příliš blízko nule, což by mohlo způsobit problémy. Musíme tedy zvolit hodnotu $\theta > 0$ tak, aby počáteční hodnoty y_i^0 měly alespoň hodnotu θ, tj.

$$y_i^0 = \max(a_i^T x^0 - b_i, \theta), \quad \forall i \in I, \tag{49}$$

kde $\theta = 0.1$. Startovací hodnoty Lagrangeových multiplikátorů λ^0 můžeme volit mnoha způsoby. Jeden z nich je následující

$$\lambda^0 = \delta e, \tag{50}$$

kde $\delta = 0.1$ a $e = (1, \ldots, 1)^T$.

Další způsob, jak volit startovací body, je

$$x^0 = \bar{x} \tag{51}$$

$$y_i^0 = \max(1, |\bar{y}_i + \triangle y_i^{aff}|) \tag{52}$$

$$\lambda_i^0 = \max(1, |\bar{\lambda}_i + \triangle \lambda_i^{aff}|), \tag{53}$$

kde $(\bar{x}, \bar{y}, \bar{\lambda})$ je libovolný bod a $(\triangle x^{aff}, \triangle y^{aff}, \triangle \lambda^{aff})$ je Newtonův krok. V našem případě byl bod $\bar{x}$ součástí zadání a body $\bar{y}$ a $\bar{\lambda}$ jsme volili následovně: $\bar{y} = (1, \ldots, 1)^T$, $\bar{\lambda} = (1, \ldots, 1)^T$.

3.4. Kritérium pro ukončení výpočtu

Iterační proces ukončíme v okamžiku, kdy dosáhneme požadované přesnosti. Nemůžeme použít kritérium

$$||x^k - x^*|| < tol,$$

jelikož přesné řešení x^* neznáme.

Použijeme tedy vazební podmínky a podmínku, že gradient Lagrangeovy funkce má být roven nule. Budeme požadovat, aby maximální porušení podmínek

$$Ax - b - y = 0$$

nepřekročilo určitou hranici, tj.

$$||Ax - b - y|| \leq tol, \tag{54}$$

kde $||.||$ je nějaká vektorová norma a tol je zvolená hranice.

Připomeňme si Lagrangeovu funkci

$$\mathcal{L}(x, y, \lambda) = \frac{1}{2}x^T Gx + x^T c - (Ax - b - y)^T \lambda.$$

Vypočítáme si gradient této funkce

$$\bigtriangledown_x \mathcal{L}(x, y, \lambda) = Gx + c - A^T \lambda$$

a postupujeme stejně jako u vazebních podmínek

$$|| \bigtriangledown_x \mathcal{L}(x, y, \lambda)|| \leq tol. \tag{55}$$

Konstantu tol volíme na základě přesnosti, se kterou chceme počítat. Iterační proces se zastaví, pokud jsou splněny podmínky (54) a (55) zároveň.

My jsme použili k ukončení iteračního procesu tři podmínky

$$y^T \lambda < tol \tag{56}$$

$$||Gx + c - A^T \lambda||_2 < tol \tag{57}$$

$$||Ax - b - y||_2 < tol, \tag{58}$$

kde tol je zvolená na základě přesnosti, se kterou chceme počítat. Iterační proces se ukončí, pokud jsou splněny podmínky (56), (57) a (58) zároveň. My jsme si zvolili $tol = 10^{-7}$.

Existuje ještě mnoho způsobů, jak ukončit iterační proces.

3.5. Praktická primárně-duální metoda

Nejvíce používaná primárně-duální metoda je založena na Mehrotrově algoritmu typu prediktor-korektor. Tento algoritmus nevyžaduje přípustný startovací bod.

Hlavním rysem praktického algoritmu je užití korektorových kroků, které kompenzují chybu udělanou při Newtonově kroku v modelování rovnosti $y_i\lambda_i = 0$, $i \in I$. Nejprve spočítáme Newtonův směr $(\triangle x^{aff}, \triangle y^{aff}, \triangle \lambda^{aff})$ a to tak, že klademe $\sigma = 0$ v soustavě (39), tj.

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \triangle x^{aff} \\ \triangle y^{aff} \\ \triangle \lambda^{aff} \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e \end{bmatrix}, \tag{59}$$

kde

$$r_d = Gx + c - A^T\lambda \tag{60}$$

$$r_p = Ax - b - y. \tag{61}$$

Jestliže použijeme plný krok v tomto směru, získáme

$$\begin{aligned}(y_i + \triangle y_i^{aff})(\lambda_i + \triangle \lambda_i^{aff}) &= y_i\lambda_i + y_i \triangle \lambda_i^{aff} + \lambda_i \triangle y_i^{aff} + \triangle y_i^{aff} \triangle \lambda_i^{aff} \\ &= \triangle y_i^{aff} \triangle \lambda_i^{aff}.\end{aligned}$$

Nová hodnota $y_i\lambda_i$ je $\triangle y_i^{aff} \triangle \lambda_i^{aff}$. Abychom získali krok $(\triangle x^{cor}, \triangle y^{cor}, \triangle \lambda^{cor})$, který se pokusí o snížení odchylky, vyřešíme následující soustavu

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \triangle x^{cor} \\ \triangle y^{cor} \\ \triangle \lambda^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ - \triangle \Lambda^{aff} \triangle Y^{aff} e \end{bmatrix}. \tag{62}$$

V mnoha případech kombinace kroků

$$(\triangle x^{aff}, \triangle y^{aff}, \triangle \lambda^{aff}) + (\triangle x^{cor}, \triangle y^{cor}, \triangle \lambda^{cor})$$

lépe zmenšuje míru duality než pouze krok $(\triangle x^{aff}, \triangle y^{aff}, \triangle \lambda^{aff})$.

Jestliže Newtonův krok podstatně zmenšuje míru duality, není potřeba centrování, a proto je vhodná menší hodnota σ. V opačném případě volíme větší hodnoty σ, které zaručí, že další iterace bude více centrovaná a bude možné provádět delší kroky z této nové iterace.

Dále spočítáme délky kroků podél Newtonova směru $(\triangle x^{aff}, \triangle y^{aff}, \triangle \lambda^{aff})$, tj.

$$\alpha_{aff}^{pri} = \min\Big(1, \min_{i \in I: \triangle y_i^{aff} < 0} \Big\{ - \frac{y_i}{\triangle y_i^{aff}} \Big\}\Big), \tag{63}$$

$$\alpha_{aff}^{dual} = \min\Big(1, \min_{i \in I: \triangle \lambda_i^{aff} < 0} \Big\{ - \frac{\lambda_i}{\triangle \lambda_i^{aff}} \Big\}\Big), \tag{64}$$

$$\alpha_{aff} = \min(\alpha_{aff}^{pri}, \alpha_{aff}^{dual}). \tag{65}$$

Nyní spočítáme míru duality μ

$$\mu = \frac{y^T \lambda}{m} \tag{66}$$

a definujeme si μ_{aff}

$$\mu_{aff} = (y + \alpha_{aff} \triangle y^{aff})^T (\lambda + \alpha_{aff} \triangle \lambda^{aff})/m. \tag{67}$$

Centrující parametr σ spočítáme podle vztahu

$$\sigma = \Big(\frac{\mu_{aff}}{\mu}\Big)^3. \tag{68}$$

Celkový krok je získán při řešení soustavy

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e - \triangle \Lambda^{aff} \triangle Y^{aff} e + \sigma \mu e \end{bmatrix}. \tag{69}$$

Řešíme tedy dvě soustavy lineárních rovnic. Koeficienty obou matic jsou stejné. Faktorizaci matice stačí počítat pouze jednou, tudíž mezní náklady na řešení druhého systému jsou relativně malé.

Nyní si uvedeme Mehrotrův algoritmus pro úlohu konvexního kvadratického programování s užitím stejných délek kroků.

Algoritmus 1

Vypočítáme (x^0, y^0, λ^0) tak, aby $(y^0, \lambda^0) > 0$;

for $k = 0, 1, 2, ...$

Položíme $(x, y, \lambda) = (x^k, y^k, \lambda^k)$ a vyřešíme soustavu (59),

dostaneme $(\triangle x^{aff}, \triangle y^{aff}, \triangle \lambda^{aff})$;

Vypočítáme $\mu = \frac{y^T \lambda}{m}$;

Vypočítáme α_{aff} podle (63), (64) a (65);

Vypočítáme $\mu_{aff} = (y + \alpha_{aff} \triangle y^{aff})^T (\lambda + \alpha_{aff} \triangle \lambda^{aff})/m$;

Vypočítáme centrující parametr $\sigma = (\mu_{aff}/\mu)^3$;

Vyřešíme soustavu rovnic (69) a dostaneme $(\triangle x, \triangle y, \triangle \lambda)$;

Délky kroků $\alpha^{pri}_{\gamma,k}$ a $\alpha^{dual}_{\gamma,k}$ vypočítáme podle (46) a (47);

Vybereme $\alpha_k = \min(\alpha^{pri}_{\gamma,k}, \alpha^{dual}_{\gamma,k})$;

Další iteraci získáme jako

$$(x^{k+1}, y^{k+1}, \lambda^{k+1}) = (x^k, y^k, \lambda^k) + \alpha_k(\triangle x, \triangle y, \triangle \lambda);$$

end.

3.6. Rozšíření primárně-duální metody na řešení úlohy kvadratického programování s podmínkami ve tvaru rovností a nerovností

Rozšíření metody na úlohy s rovnostmi a nerovnostmi je velmi snadné.

Uvažujme úlohu

$$\begin{aligned} \textit{minimalizovat} \quad & f(x) = \frac{1}{2}x^T Gx + x^T c \\ \textit{za podmínek} \quad & Ax \geq b \\ & Ex = f. \end{aligned} \tag{70}$$

Zavedeme doplňkový vektor $y \geq 0$ a sestrojíme Lagrangeovu funkci. Lagrangeova funkce pro úlohu s rovnostmi a nerovnostmi bude mít následující tvar

$$\mathcal{L}(x, y, \lambda, \nu) = \frac{1}{2}x^T Gx + x^T c - (Ax - b - y)^T \lambda - (Ex - f)^T \nu. \tag{71}$$

Sestrojíme KKT podmínky

$$\begin{aligned} Gx + c - A^T\lambda - E^T\nu &= 0 \\ Ax - b - y &= 0 \\ Ex - f &= 0 \\ y_i\lambda_i &= 0, \quad i = 1, \ldots, m \\ (y, \lambda) &\geq 0. \end{aligned}$$

Centrální cestu získáme řešením soustavy

$$Gx + c - A^T\lambda - E^T\nu = 0 \tag{72}$$

$$Ax - b - y = 0 \tag{73}$$

$$Ex - f = 0 \tag{74}$$

$$y_i\lambda_i = \tau, \quad i = 1, \ldots, m \tag{75}$$

$$(y, \lambda) > 0, \tag{76}$$

kde $\tau = \sigma\mu$.

Po aplikaci Newtonovy metody na soustavu (72)-(75) dostaneme soustavu rovnic

$$\begin{bmatrix} G & 0 & -A^T & -E^T \\ A & -I & 0 & 0 \\ E & 0 & 0 & 0 \\ 0 & \Lambda & Y & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \\ \triangle \nu \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -r_r \\ -\Lambda Y e + \sigma \mu e \end{bmatrix}, \tag{77}$$

kde

$$\begin{aligned} r_d &= Gx + c - A^T\lambda - E^T\nu \\ r_p &= Ax - b - y \\ r_r &= Ex - f. \end{aligned}$$

Další postup je stejný jako v úloze s podmínkami ve tvaru nerovností.

Počáteční hodnoty Lagrangeových multiplikátorů ν^0 můžeme volit mnoha způsoby. Ve výpočtech jsme používali buď

$$\nu^0 = \delta e, \tag{78}$$

kde $\delta = 0.1$ a $e = (1, \ldots, 1)^T$ nebo

$$\nu^0 = \bar{\nu}, \tag{79}$$

kde $\bar{\nu}$ je libovolný bod. My jsme volili $\bar{\nu} = (1, \ldots, 1)^T$.

Nové iterace vypočítáme podle vztahů

$$(x^+, y^+) = (x, y) + \alpha^{pri}(\triangle x, \triangle y) \tag{80}$$

$$(\lambda^+, \nu^+) = (\lambda, \nu) + \alpha^{dual}(\triangle \lambda, \triangle \nu), \tag{81}$$

kde α^{pri} je délka kroku pro primární proměnné x, y a α^{dual} je délka kroku pro duální proměnné λ, ν.

K podmínkám na ukončení iteračního procesu musíme ještě přidat podmínku

$$||Ex - f||_2 < tol, \tag{82}$$

kde tol volíme jako 10^{-7}.

4. Řešení úlohy kvadratického programování pomocí metody logaritmické bariérové funkce

V této kapitole si popíšeme druhou interpretaci metod vnitřních bodů-metodu logaritmické bariérové funkce.

Metoda logaritmické bariérové funkce spočívá v nahrazení účelové funkce funkcí složenou. Složená funkce obsahuje původní účelovou funkci a bariérový člen.

Účelovou funkci $f(x)$ nahradíme složenou účelovou funkcí, tj. dostaneme úlohu

$$\begin{aligned} \textit{minimalizovat} \quad & f(x) - \tau \sum_{i \in I} \ln\, y_i && (83) \\ \textit{za podmínek} \quad & g_i(x) - y_i = 0 \quad i \in I && (84) \\ & h_j(x) = 0 \quad j \in J. && (85) \end{aligned}$$

Úloha má pouze omezení ve tvaru rovnosti, protože nerovnost $y_i \geq 0$ je zahrnuta v $\ln\, y_i$, neboť v argumentu přirozeného logaritmu mohou být jen kladné hodnoty (tj. y_i je dokonce větší než 0).

Pro úlohu kvadratického programování dostáváme tvar

$$\begin{aligned} \textit{minimalizovat} \quad & \frac{1}{2} x^T G x + x^T c - \tau \sum_{i \in I} \ln y_i \\ \textit{za podmínek} \quad & a_i^T x - b_i - y_i = 0 \quad i \in I \\ & e_j^T x - f_j = 0 \quad j \in J. \end{aligned} \tag{86}$$

Sestrojíme Lagrangeovu funkci úlohy (86)

$$\mathcal{L}(x, y, \lambda, \nu) = \frac{1}{2} x^T G x + x^T c - \tau \sum_{i \in I} \ln y_i - (Ax - b - y)^T \lambda - (Ex - f)^T \nu,$$

kde parametr τ je kladné číslo.

KKT podmínky úlohy (86) si přepíšeme do následujícího tvaru

$$Gx + c - A^T\lambda - E^T\nu = 0 \tag{87}$$

$$Ax - b - y = 0 \tag{88}$$

$$Ex - f = 0 \tag{89}$$

$$-\tau Y^{-1}e + \lambda = 0 \tag{90}$$

$$(y, \lambda) \geq 0. \tag{91}$$

Na soustavu (87)-(90) aplikujeme Newtonovu metodu a získáme soustavu lineárních rovnic

$$\begin{bmatrix} G & 0 & -A^T & -E^T \\ A & -I & 0 & 0 \\ E & 0 & 0 & 0 \\ 0 & -\tau Y^{-2} & I & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \\ \triangle \nu \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -r_r \\ \tau Y^{-1}e - \lambda \end{bmatrix}, \tag{92}$$

kde

$$r_d = Gx + c - A^T\lambda - E^T\nu$$

$$r_p = Ax - b - y$$

$$r_r = Ex - f.$$

Dostaneme tzv. *primární metodu*.

Vynásobíme-li rovnici (90) maticí Y, dostaneme KKT podmínky ve tvaru

$$Gx + c - A^T\lambda - E^T\nu = 0$$

$$Ax - b - y = 0$$

$$Ex - f = 0$$

$$-\tau e + Y\Lambda e = 0$$

$$(y, \lambda) \geq 0.$$

Opět použijeme Newtonovu metodu a dostaneme tzv. *primárně-duální metodu* ve tvaru

$$\begin{bmatrix} G & 0 & -A^T & -E^T \\ A & -I & 0 & 0 \\ E & 0 & 0 & 0 \\ 0 & \Lambda & Y & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \\ \triangle \nu \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -r_r \\ \tau e - Y\Lambda e \end{bmatrix}, \tag{93}$$

kde

$$\begin{aligned} r_d &= Gx + c - A^T\lambda - E^T\nu \\ r_p &= Ax - b - y \\ r_r &= Ex - f. \end{aligned}$$

Centrální cesta je množina bodů $C = \{(x_\tau, y_\tau, \lambda_\tau, \nu_\tau) | \tau > 0\}$, které jsou řešením soustavy

$$\begin{aligned} Gx + c - A^T\lambda - E^T\nu &= 0 \\ Ax - b - y &= 0 \\ Ex - f &= 0 \\ Y\Lambda e &= \tau e \\ (y, \lambda) &> 0, \end{aligned}$$

kde parametr $\tau = \sigma\mu$ a jeho výpočet je popsán ve třetí kapitole.

Jak vidíme, po všech úpravách jsme došli ke stejné soustavě jako v kapitole číslo 3.

4.1. Lokální konvergence metody logaritmické bariérové funkce

Uvažujme následující problém

$$\textit{minimalizovat} \quad f(x) \tag{94}$$

$$\textit{za podmínek} \quad g_i(x) \geq 0, i \in I. \tag{95}$$

Označme si $B(x, \tau)$ logaritmickou bariérovou funkci tvaru

$$B(x, \tau) = f(x) - \tau \sum_{i=1}^{m} \ln g_i(x),$$

kde τ je kladný bariérový parametr. Symbolem $\mathcal{F}$ označme přípustnou oblast úlohy (94)-(95) a symbolem $\mathcal{F}^0$ striktně přípustnou oblast úlohy (94)-(95). Minimální hodnotu účelové funkce $f(x)$ pro $x \in \mathcal{F}$ označíme f^*.

Věta 2. *(Existence kompaktní uzavřené množiny)*
Uvažujme problém (94)-(95). Nechť $\mathcal{N}$ označuje množinu všech lokálních vázaných minim s hodnotou účelové funkce f^* a předpokládejme, že f^* má být vybrána tak, že $\mathcal{N}$ je neprázdná. Předpokládejme dále, že množina $\mathcal{N}^* \subseteq \mathcal{N}$ je neprázdná kompaktní izolovaná podmnožina z $\mathcal{N}$. Pak existuje kompaktní množina S taková, že $\mathcal{N}^*$ leží v $int(S) \cap \mathcal{F}$ a $f(y) > f^*$ pro každý přípustný bod y, který leží v S, ale neleží v $\mathcal{N}^*$. Každý bod x^*, který leží v $\mathcal{N}^*$, má tu vlastnost, že $f(x^*) = f^* = \min f(x)$ pro všechna $x \in S \cap \mathcal{F}$.

Důkaz: Viz. [4].

Věta 3. *(Lokální konvergence bariérových metod)*
Uvažujme problém (94)-(95), kde f a g jsou spojité funkce. Nechť $\mathcal{F}$ označuje přípustnou oblast, nechť $\mathcal{N}$ označuje množinu bodů minima účelové funkce $f(x)$ odpovídající f^* a předpokládejme, že $\mathcal{N}$ je neprázdná. Nechť $\{\tau_k\}$ je ryze klesající posloupnost kladných bariérových parametrů taková, že $\lim_{k\to\infty} \tau_k = 0$. Předpokládáme, že

(a) existuje neprázdná kompaktní množina lokálních minim $\mathcal{N}^*$, která je izolovanou podmnožinou $\mathcal{N}$;

(b) alespoň jeden bod $\mathcal{N}^*$ se nachází v uzávěru $\mathcal{F}^0$.

Pak platí následující tvrzení

(i) existuje kompaktní množina S taková, že $\mathcal{N}^* \subset int(S)$ a taková, že pro každý přípustný bod $\bar{x}$, který leží v S, ale neleží v $\mathcal{N}^*$, je $f(\bar{x}) > f^*$;

(ii) pro všechna dostatečně malá τ_k je nevázané minimum y_k bariérové funkce $B(x, \tau_k)$ v $\mathcal{F}^0 \cap int(S)$ a platí

$$B(y_k, \tau_k) = \min\{B(x, \tau_k) : x \in \mathcal{F}^0 \cap S\}.$$

Tedy $B(y_k, \tau_k)$ je nejmenší hodnota $B(x, \tau_k)$ pro každé $x \in \mathcal{F}^0 \cap S$;

(iii) každá posloupnost těchto nevázaných minim $\{y_k\}$ funkce $B(x, \tau_k)$ má alespoň jednu konvergentní podposloupnost;

(iv) bod x_∞ každé konvergentní podposlounosti $\{x_k\}$ vybrané z neomezených minim $\{y_k\}$ definovaný v (ii) leží v $\mathcal{N}^*$;

(v) pro konvergentní podposloupnosti $\{x_k\}$ z (iv) je

$$\lim_{k\to\infty} f(x_k) = f^* = \lim_{k\to\infty} B(x_k, \tau_k).$$

Důkaz: Viz. [4].

5. Řešení soustav rovnic vznikajících při použití metody vnitřních bodů

Soustavu lineárních rovnic, kterou dostaneme po aplikaci Newtonovy metody, můžeme řešit mnoha způsoby. Výběr nejvhodnější metody je předmětem numerického experimentování.

První z metod, kterou si uvedeme, je *metoda Schurova komplementu (I.)*. Uvažujme soustavu tvaru

$$R\begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} A & B \\ B^T & D \end{pmatrix}\begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}. \tag{96}$$

Výsledné vektory r a s získáme jako řešení následující soustavy

$$\begin{pmatrix} r \\ s \end{pmatrix} = R^{-1}\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A^{-1} - A^{-1}BF & -A^{-1}BH^{-1} \\ F & H^{-1} \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix},$$

kde

$$H = D - B^T A^{-1} B$$
$$F = -H^{-1} B^T A^{-1}.$$

Vidíme, že

$$r = A^{-1}u - A^{-1}BFu - A^{-1}BH^{-1}v$$
$$s = Fu + H^{-1}v.$$

Označme si

$$z_1 = H^{-1}v \Longrightarrow Hz_1 = v$$
$$z_2 = Fu = -H^{-1}B^T A^{-1}u = -H^{-1}w \Longrightarrow Hz_2 = -w,$$

kde

$$w = B^T A^{-1} u.$$

Po výpočtu dosadíme z_1 a z_2 do vztahů pro výpočet r a s, tj.

$$r = A^{-1}u - A^{-1}Bz_2 - A^{-1}Bz_1$$
$$s = z_2 + z_1.$$

Tento způsob výpočtu je vhodný pro rozsáhlé úlohy. Při aplikaci metody Schurova komplementu je velmi výhodné použít Choleského metodu.

Viz. [13].

My potřebujeme najít řešení soustavy

$$\begin{bmatrix} G & 0 & -A^T & -E^T \\ A & -I & 0 & 0 \\ E & 0 & 0 & 0 \\ 0 & \Lambda & Y & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \\ \triangle \lambda \\ \triangle \nu \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -r_r \\ -\Lambda Y e + \sigma \mu e \end{bmatrix}, \tag{97}$$

kde

$$r_d = Gx + c - A^T\lambda - E^T\nu$$
$$r_p = Ax - b - y$$
$$r_r = Ex - f,$$

tudíž musíme tuto soustavu ještě upravit.

Ze soustavy (97) si přímo vyjádříme doplňkový vektor y. Dostaneme

$$\triangle y = -y - \Lambda^{-1}Y \triangle \lambda + \sigma\mu\Lambda^{-1}e.$$

Dosadíme $\triangle y$ do soustavy (97), tj.

$$A \triangle x + \Lambda^{-1}Y \triangle \lambda = -r_p - y + \sigma\mu\Lambda^{-1}e.$$

Nová soustava má tedy tvar

$$\begin{bmatrix} G & -A^T & -E^T \\ A & \Lambda^{-1}Y & 0 \\ E & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle \lambda \\ \triangle \nu \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p - y + \sigma\mu\Lambda^{-1}e \\ -r_r \end{bmatrix}, \tag{98}$$

kde

$$r_d = Gx + c - A^T\lambda - E^T\nu$$
$$r_p = Ax - b - y$$
$$r_r = Ex - f.$$

Po úpravě získáme soustavu

$$\begin{bmatrix} -G & A^T & E^T \\ A & \Lambda^{-1}Y & 0 \\ E & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle\lambda \\ \triangle\nu \end{bmatrix} = \begin{bmatrix} r_d \\ -Ax + b + \sigma\mu\Lambda^{-1}e \\ -r_r \end{bmatrix}, \tag{99}$$

kde

$$r_d = Gx + c - A^T\lambda - E^T\nu$$
$$r_r = Ex - f.$$

Při aplikaci metody Schurova komplementu na soustavu (99) vidíme, že

$$A = -G$$
$$B = \begin{pmatrix} A^T & E^T \end{pmatrix}$$
$$D = \begin{pmatrix} \Lambda^{-1}Y & 0 \\ 0 & 0 \end{pmatrix}$$
$$u = r_d$$
$$v = \begin{pmatrix} -Ax + b + \sigma\mu\Lambda^{-1}e \\ -r_r \end{pmatrix}$$
$$r = \triangle x$$
$$s = \begin{pmatrix} \triangle\lambda \\ \triangle\nu \end{pmatrix}.$$

Vektor $\triangle y$ vypočítáme podle vztahu

$$\triangle y = -y - \Lambda^{-1}Y \triangle \lambda + \sigma\mu\Lambda^{-1}e. \tag{100}$$

Tento postup můžeme použít v případě, že je matice G pozitivně definitní.

V případě, že je matice G pozitivně semidefinitní, použijeme následující postup.
Soustavu (99) přepíšeme do následujícího tvaru

$$\begin{bmatrix} G & -A^T & -E^T \\ -A & -\Lambda^{-1}Y & 0 \\ -E & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle \lambda \\ \triangle \nu \end{bmatrix} = \begin{bmatrix} -r_d \\ Ax - b - \sigma\mu\Lambda^{-1}e \\ r_r \end{bmatrix}.$$

V této soustavě si označíme $B = (-A^T, -E^T)$, $r = \triangle x$, $u = -r_d$,

$$v = \begin{pmatrix} Ax - b - \sigma\mu\Lambda^{-1}e \\ r_r \end{pmatrix}, s = \begin{pmatrix} \triangle\lambda \\ \triangle\nu \end{pmatrix}, D = \begin{pmatrix} -\Lambda^{-1}Y & 0 \\ 0 & 0 \end{pmatrix}.$$

Uvažujme matici $G = M - N$, kde $N = \sigma I$, $\sigma > 0$. Zavedeme si nový iterační proces

$$\begin{pmatrix} M & B \\ B^T & D \end{pmatrix} \begin{pmatrix} r^{k+1} \\ s^{k+1} \end{pmatrix} = \begin{pmatrix} N & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} r^k \\ s^k \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix},$$

tj. řešíme soustavu

$$\begin{pmatrix} M & B \\ B^T & D \end{pmatrix} \begin{pmatrix} r^{k+1} \\ s^{k+1} \end{pmatrix} = \begin{pmatrix} Nr^k + u \\ v \end{pmatrix}.$$

Nyní je matice $M = G + N$ pozitivně definitní a na soustavu můžeme aplikovat metodu Schurova komplementu.

Viz. [7].

Druhou metodou, kterou při výpočtu používáme, je *Gaussova eliminační metoda s částečným výběrem prvku (II.)*, kterou aplikujeme na soustavu (97).

Třetí metodou ($III.$), kterou jsme použili k řešení soustavy lineárních rovnic (97) v programu Matlab, je příkaz

```
x=A\b.
```

6. Řešené příklady

Tato kapitola obsahuje příklady, které jsou řešeny pomocí programů přiložených na CD. U jednotlivých příkladů uvádíme optimální řešení, funkční hodnoty v řešení, vektory Lagrangeových multiplikátorů a počty iterací při užití programového zpracování *Algoritmu 1* v programu Matlab a v programovacím jazyce Fortran 77 a při použití různých metod na řešení soustav rovnic. Pro srovnání zde uvádíme počty iterací při užití programového zpracování algoritmu nepřípustné metody sledování cesty s dlouhým krokem a počty iterací při užití různých délek kroků pro primární proměnné x, y a duální proměnné λ, ν v *Algoritmu 1* v programu Matlab a v programovacím jazyce Fortran 77. V posledních dvou metodách řešíme soustavy rovnic metodou Schurova komplementu. Startovací body volíme ve všech metodách podle (51), (52), (53) a (79). Pro srovnání uvádíme počet iterací při užití programového zpracování *Algoritmu 1* v Matlabu při volbě startovacího bodu podle (49), (50), (51) a (78).

Příklad 1:

$$f(x) = \frac{1}{2}x^T \underbrace{\begin{pmatrix} 4 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}}_{G} x + x^T \underbrace{\begin{pmatrix} -8 \\ -6 \\ -6 \end{pmatrix}}_{c}$$

$$x_1 + x_2 + x_3 = 3$$
$$x_1, x_2, x_3 \geq 0$$

Tj.

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, E = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}^T, f = 3$$

Matice G je pozitivně semidefinitní.

a)

$$x^0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \textit{ tj. } x^0 \textit{ je přípustný}$$

Řešení:

$$x^* = (0.5, 1.25, 1.25)^T$$
$$\lambda^* = (0.6641, 0.01, 0.01)^T.10^{-8}$$
$$\nu^* = -6$$
$$f(x^*) = -18.5$$

Počet iterací: Matlab

Metoda	$I.$	$II.$	$III.$
stejné kroky	5	5	5
start. bod (49), (50), (51) a (78)	4	–	–
různé kroky	5	–	–

Počet iterací: Fortran 77

Metoda	$I.$	$II.$
stejné kroky	5	5
různé kroky	5	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
6	6

b)

$$x^0 = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}, \textit{ tj. } x^0 \textit{ je nepřípustný}$$

Řešení:

$$x^* = (0.5, 1.25, 1.25)^T$$
$$\lambda^* = (0.6641, 0.01, 0.01)^T.10^{-8}$$
$$\nu^* = -6$$
$$f(x^*) = -18.5$$

Počet iterací: Matlab

Metoda	*I.*	*II.*	*III.*
stejné kroky	5	5	5
start. bod (49), (50), (51) a (78)	4	–	–
různé kroky	5	–	–

Počet iterací: Fortran 77

Metoda	*I.*	*II.*
stejné kroky	5	5
různé kroky	5	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
6	6

Příklad 2:

$$f(x) = x_1^2 + x_2^2 - 6x_1 - 4x_2 + 13$$

$$x_1 + x_2 \leq 3$$
$$x_1, x_2 \geq 0$$

$$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Tj.

$$A = \begin{pmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, b = \begin{pmatrix} -3 \\ 0 \\ 0 \end{pmatrix}, G = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, c = \begin{pmatrix} -6 \\ -4 \end{pmatrix}$$

Řešení:

$$x^* = (2, 1)^T$$
$$\lambda^* = (2, 0.18.10^{-8}, 0.53.10^{-8})^T$$
$$f(x^*) = 2$$

Počet iterací: Matlab

Metoda	*I.*	*II.*	*III.*
stejné kroky	5	5	5
start. bod (49), (50) a (51)	6	–	–
různé kroky	5	–	–

Počet iterací: Fortran 77

Metoda	*I.*	*II.*
stejné kroky	5	5
různé kroky	5	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
6	6

Příklad 3:

$$f(x) = x_1^2 - x_1x_2 + x_2^2 - 3x_1$$

$$-x_1 - x_2 \geq -2$$

$$x_1, x_2 \geq 0$$

$$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Tj.

$$A = \begin{pmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, b = \begin{pmatrix} -2 \\ 0 \\ 0 \end{pmatrix}, G = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, c = \begin{pmatrix} -3 \\ 0 \end{pmatrix}$$

Řešení:

$$x^* = (1.5, 0.5)^T$$

$$\lambda^* = (0.5, 0.53.10^{-9}, 0.31.10^{-8})^T$$

$$f(x^*) = -2.75$$

Počet iterací: Matlab

Metoda	$I.$	$II.$	$III.$
stejné kroky	5	5	5
start. bod (49), (50) a (51)	5	–	–
různé kroky	5	–	–

Počet iterací: Fortran 77

Metoda	$I.$	$II.$
stejné kroky	5	5
různé kroky	5	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
6	6

Příklad 4:

$$f(x) = -2x_1 + 0.5x_1^2 - 6x_2 - x_1x_2 + x_2^2$$

$$3x_1 + x_2 \leq 25$$
$$-x_1 + 2x_2 \leq 10$$
$$x_1 + 2x_2 \leq 15$$
$$x_1, x_2 \geq 0$$

$$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Tj.

$$A = \begin{pmatrix} -3 & -1 \\ 1 & -2 \\ -1 & -2 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, b = \begin{pmatrix} -25 \\ -10 \\ -15 \\ 0 \\ 0 \end{pmatrix}, G = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}, c = \begin{pmatrix} -2 \\ -6 \end{pmatrix}$$

Řešení:

$$x^* = (5.6, 4.7)^T$$
$$\lambda^* = (0.6.10^{-10}, 0.66.10^{-11}, 1.1, 0.21.10^{-10}, 0.61.10^{-10})^T$$
$$f(x^*) = -27.95$$

Počet iterací: Matlab

Metoda	$I.$	$II.$	$III.$
stejné kroky	6	6	6
start. bod (49), (50) a (51)	7	–	–
různé kroky	5	–	–

Počet iterací: Fortran 77

Metoda	$I.$	$II.$
stejné kroky	6	6
různé kroky	5	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
7	7

Příklad 5:

$$f(x) = x_1^2 + x_1x_2 + 2x_2^2 + 2x_3^2 + 2x_2x_3 + 4x_1 + 6x_2 + 12x_3$$

$$x_1 + x_2 + x_3 \geq 6$$
$$-x_1 - x_2 + 2x_3 \geq 2$$
$$x_1, x_3 \geq 0$$

$$x^0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Tj.

$$A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & -1 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, b = \begin{pmatrix} 6 \\ 2 \\ 0 \\ 0 \end{pmatrix}, G = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 4 & 2 \\ 0 & 2 & 4 \end{pmatrix}, c = \begin{pmatrix} 4 \\ 6 \\ 12 \end{pmatrix}$$

Řešení:

$$\begin{aligned} x^* &= (4.3333, -1, 2.6667)^T \\ \lambda^* &= (14.6667, 3, 0.63.10^{-9}, 0.82.10^{-9})^T \\ f(x^*) &= 68.6667 \end{aligned}$$

Počet iterací: Matlab

Metoda	*I.*	*II.*	*III.*
stejné kroky	6	6	6
start. bod (49), (50) a (51)	8	–	–
různé kroky	6	–	–

Počet iterací: Fortran 77

Metoda	*I.*	*II.*
stejné kroky	6	6
různé kroky	6	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
8	8

Příklad 6: Markowitzův model - sestavení portfolia z n akcií na jedno investiční období s předpokladem averze rozhodovatele k riziku

$c_j \ldots$ výnosy jednotlivých akcií(náhodné veličiny)

$e_j \ldots$ střední hodnoty výnosů

$g_j^2 \ldots$ rozptyly výnosů

$g_j \ldots$ směrodatná odchylka, riziko

$x_j \ldots$ podíl akcie na portfoliu, $x_j \in [0, 1]$

Markowitzův model:

$$\begin{aligned} \textit{minimalizovat} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} g_{ij}x_i x_j \\ \textit{za podmínek} \quad & \sum_{j=1}^{n} e_j x_j = \tau \\ & \sum_{j=1}^{n} x_j = 1 \\ & x_j \geq 0, \quad j = 1, \ldots, n, \end{aligned}$$

kde τ označuje požadovanou výši očekávaného výnosu portfolia.

Jde tedy o následující úlohu

$$\begin{aligned} \textit{minimalizovat} \quad & f(x) = x^T G x \\ \textit{za podmínek} \quad & Ex = f \\ & x \geq 0 \end{aligned}$$

$$x^0 = (1, \ldots, 1)^T$$

Máme dána data

$$A = \begin{pmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, E^T = \begin{pmatrix} 0.0093 & 1 \\ 0.0741 & 1 \\ 0.1919 & 1 \\ 0.1865 & 1 \\ 0.0676 & 1 \\ 0.0016 & 1 \\ 0.1178 & 1 \\ 0.0674 & 1 \end{pmatrix}, f = \begin{pmatrix} 0.16 \\ 1 \end{pmatrix}, c = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$G = \begin{pmatrix} 0.1756 & 0.0641 & 0.1462 & 0.0093 & 0.0057 & -0.0531 & -0.0632 & -0.0068 \\ 0.0641 & 0.2177 & 0.1041 & 0.0808 & 0.0596 & 0.0179 & -0.0275 & 0.0898 \\ 0.1462 & 0.1041 & 0.3556 & -0.0134 & 0.0133 & -0.0116 & 0.0640 & 0.0056 \\ 0.0093 & 0.0808 & -0.0134 & 0.3189 & -0.0520 & -0.0452 & -0.0348 & 0.0752 \\ 0.0057 & 0.0596 & 0.0133 & -0.0520 & 0.0768 & 0.0355 & -0.0071 & -0.0004 \\ -0.0531 & 0.0179 & -0.0116 & -0.0452 & 0.0355 & 0.0859 & 0.0695 & 0.0060 \\ -0.0632 & -0.0275 & 0.0640 & -0.0348 & -0.0071 & 0.0695 & 0.1787 & 0.0053 \\ -0.0068 & 0.0898 & 0.0056 & 0.0752 & -0.0004 & 0.0060 & 0.0053 & 0.1619 \end{pmatrix}$$

Řešení:

$$x^* = (0, 0, 0.2896, 0.3892, 0.1195, 0, 0.2017, 0)^T$$

$$\lambda^* = (0.1985, 0.1310, 0.14.10^{-7}, 0.11.10^{-7}, 0.1.10^{-6}, 0.1403, 0.26.10^{-7}, 0.0815)^T$$

$$\nu^* = (1.9454, -0.1488)^T$$

$$f(x^*) = 0.0812$$

Počet iterací: Matlab

Metoda	*I.*	*II.*	*III.*
stejné kroky	6	6	6
start. bod (49), (50), (51) a (78)	6	–	–
různé kroky	6	–	–

Počet iterací: Fortran 77

Metoda	*I.*	*II.*
stejné kroky	6	6
různé kroky	6	–

Počet iterací: Nepřípustná metoda sledování cesty s dlouhým krokem

Matlab	Fortran 77
8	8

Počty iterací se při užití různých metod k řešení soustav rovnic neliší.

V nepřípustné metodě sledování cesty s dlouhým krokem jsme volili parametr $\beta = 5$. Zvolíme-li parametr β jinak, mohou se počty iterací lišit.

Při volbě délky kroku α v metodě s dlouhým krokem jsme použili tzv. zpětné vyhledávání, tj. vybrali jsme první hodnotu α z posloupnosti α, $\rho\alpha$, $\rho^2\alpha$, ... tak, aby nová iterace padla do okolí $\mathcal{N}_{-\infty}(\gamma, \beta)$. Parametr $\rho \in (0, 1)$, nejčastěji se volí jako $\rho = 0.9$.

Volba startovacího bodu je předmětem experimentování. V příkladech jsme použili dvě různé metody na volbu startovacího bodu. Z tabulek je zřejmé, že se počty iterací mohou lišit.

Nepřípustná metoda sledování cesty s dlouhým krokem vyžaduje větší počet iterací než metoda prediktor-korektor.

Při volbě různých délek kroků se v některých příkladech počet iterací snížil.

6.1. Popis programového zpracování algoritmu v programu Matlab

Základem algoritmu je cyklus, ve kterém se provádí jednotlivé iterace. Nejprve začneme tím, že zadáme všechna data, která máme dána, tj. matice G, A, E, vektory c, b, f, a zvolíme maximální počet iterací $maxit$. Matice a vektory načteme pomocí příkazů *load* z textových souborů, které jsme si vytvořili. Data do souborů $*.txt$ zadáváme následovně:

```
4  0  0
0  1 -1
0 -1  1.
```

Dále naprogramujeme cyklus pomocí příkazu $for\ k = 1, maxit\ \ldots\ end$, do něhož vložíme jednotlivé příkazy. V cyklu vytvoříme podmínku pomocí příkazu if, při jejímž splnění se celý cyklus ukončí. Na závěr programu uvedeme příkazy, které zajistí výpis hodnot jednotlivých proměnných.

6.2. Základní příkazy programovacího jazyka Fortran 77

Jazyk Fortran je jedním z nejstarších programovacích jazyků. Je to jazyk, který byl a je určen především pro vědecko-technické výpočty.

Uvedeme si pro ukázku některé základní příkazy, které jsme použili v programovém zpracování algoritmů pro výpočet.

Program se většinou skládá z hlavního programu a z libovolného počtu vnějších procedur a programových jednotek. Hlavní program začíná popisem PROGRAM ih, kde ih je identifikátor a slouží k pojmenování hlavního programu. Podprogram začíná popisem SUBROUTINE jm $[(sfp)]$, jm je identifikátor podprogramu. Volání podprogramu se provádí pomocí příkazu CALL jm $[(ssp)]$, kde jm je identifikátor volaného podprogramu a ssp je seznam skutečných parametrů oddělených čárkami.

Na začátku programu je nutné určit typ proměnných, tj. určit, zda jsou reálné, celočíselné, znakové nebo zda počítáme s dvojitou přesností.

Příkaz OPEN ([UNIT =] j, FILE = zv), kde zv je znakový výraz označující vstupní a výstupní jednotku nebo soubor.

Příkazy READ (j, n) s a WRITE (j, n) s, kde j je celé kladné číslo bez znaménka označující vstupní a výstupní jednotku nebo znak *. Symbol n je návěští popisu FORMAT nebo znak *, který označuje volné formátování.

Příkaz cyklu má tvar DO n [,]$i = v_1$, v_2[,v_3], kde n je návěští posledního příkazu cyklu, i je parametr cyklu a v_1, v_2, v_3 jsou výrazy typu INTEGER nebo REAL, které označují dolní mez resp. horní mez parametru cyklu resp. krok. Příkaz DO ukončíme příkazem END DO.

Podmíněný příkaz je určen k podmíněnému provádění příkazů nebo skupin příkazů. To znamená, že provedení těchto příkazů je vázáno na splnění určité podmínky. My jsme použili příkaz

IF (v) skupina příkazů,

kde v je logický výraz.

Příkaz STOP ukončí činnost programu.

Příkaz END musí být uveden právě jednou v každé programové jednotce jako

její poslední příkaz.

Nyní si ještě ukážeme zadávání dat. Ve složce Output vytvoříme textový soubor *DATA.dt.txt* a následně odstraníme příponu *.txt*. Data do souboru zadáváme podle toho, jak budeme data číst. V našem případě zadáváme data po řádcích. Pro představu uvádíme příklad:
4.,0.,0.,0.,1.,-1.,0.,-1.,1.
Musíme provést následující opravu. V okně programu klikneme na *Project* a dále pak na *Project Options*. Poté opravíme položku *Result file* tak, aby obsahovala $*.dt$.

Ve výpočtech mohou nastat chyby. Jsou to například: zaokrouhlovací chyby nebo chyby způsobené prací a manipulací s maticemi.

Závěr

Cílem této práce bylo seznámení s řešením úlohy kvadratického programování pomocí metod vnitřních bodů. V práci jsme popsali dvě interpretace metod vnitřních bodů a dále jsme se také zabývali různými metodami sledování cesty. Na řešení úloh jsme vytvořili programy v programovacím jazyce Fortran 77 a také jsme řešili problematiku v matematickém programu Matlab.

Metody vnitřních bodů jsou metody, které vyžadují menší počet nákladnějších kroků a jsou velmi účinné pro velké optimalizační problémy. Metody vnitřních bodů jsou méně vhodné při užití tzv. „horkého startu". Různé úpravy algoritmů mohou vést ke změně počtu iterací a k větší přesnosti řešení. Jde například o volbu různých délek kroků pro primární a duální proměnné.

V práci je uvedena metoda k řešení soustav rovnic vznikajících při použití metody vnitřních bodů-metoda Schurova komplementu. Pro autorku práce je tato metoda nová. Při řešení rozsáhlých problémů je užití této metody velmi vhodné. Při výpočtu pomocí metody Schurova komplementu jsme používali Choleského metodu.

Odlišnosti mezi programovacím jazykem Fortran 77 a programem Matlab jsou dosti velké. Program Matlab má v sobě již zabudované některé operace a nemusíme se jimi tedy zabývat, zatímco v jazyce Fortran 77 musíme tyto operace naprogramovat. Rozdíl je především v práci s vektory a maticemi. Jelikož jsme v programu vytvořeném programovacím jazykem Fortran 77 počítali s dvojitou přesností, tak se počty iterací při výpočtu v našem programu a v programu Matlab neliší.

Při psaní diplomové práce jsem získala mnoho nových zkušeností s programováním a také jsem se seznámila s částí optimalizace, kterou jsem dříve nestudovala.

Literatura

[1] Benda, J., Černá, R.: Základy programování ve Fortranu, vydalo ČVUT, Praha, 1995.

[2] Bonnans, J. F., Gilbert, J. Ch., Lemaréchal, C., Sagastizábal, C. A.: Numerical Optimization, Springer, USA, 2006.

[3] Fletcher, R.: Practical Methods of Optimization, Wiley, 2000.

[4] Forsgren, A., Gill, P. E., Wright, M. H.: Interior Methods for Nonlinear Optimization, SIAM, 2002.

[5] Hřebíček, J. a kolektiv: Programovací jazyk Fortran 77 a vědeckotechnické výpočty, Academia, 1989.

[6] Macek, J.: Diplomová práce: Metoda vnitřního bodu pro rozsáhlé úlohy nelineárního programování, Matematicko-fyzikální fakulta Univerzity Karlovy v Praze, 1999.

[7] Netuka, H.: Soukromé sdělení.

[8] Nocedal, J., Wright, S. J.: Numerical Optimization, Springer, USA, 2006.

[9] Potra, F. A., Wright, S. J.: Interior-point methods. Journal of Computational and Applied Mathematics, 124, 281-302, 2000.

[10] Roos, C., Terlaky, T., Vial, J. P.: Interior Point Methods for Linear Optimization, Springer, USA, 2005.

[11] Vicher, M.: Fortran, vydala Univerzita Jana Evangelisty Purkyně v Ústí nad Labem, 2003.

[12] Wright, S. J.: Primal-Dual Interior-Point Methods, SIAM, Philadelphia, 1997.

[13] Ženčák, P.: Soukromé sdělení.

Příloha 1

Uvádíme zde programové zpracování algoritmu pro řešení úlohy kvadratického programování s podmínkami ve tvaru rovností a nerovností pomocí primárně-duální metody v programu Matlab. Soustavy rovnic řešíme pomocí metody Schurova komplementu s využitím Choleského metody a délky kroků volíme stejné pro primární a duální proměnné. Jde o rozšíření *Algoritmu 1* na úlohy s rovnostmi a nerovnostmi. Následující program je určen k řešení *Příkladu 6*.

Tento program je spolu s dalšími přiložen na CD.

```
function primdualschurpr6
load G.txt
load A.txt
load E.txt
load b.txt
load c.txt
load f.txt
load x0.txt
G=2.*G;
gama=0.99;
[GL,nav]=rozklad(G);
[m,n]=size(A);
[h,n]=size(E);
lam0=ones(m,1);
ni0=ones(h,1);
y0=ones(m,1);
Li=zeros(m);
for i=1:m
    Li(i,i)=1/lam0(i);
end;
Y0=diag(y0);
Q=Li*Y0;
nav=2;
u=G*x0+c-A'*lam0-E'*ni0;
rp=A*x0-b-y0;
rr=E*x0-f;
v=[-rp-y0;-rr];
C=[A' E'];
D=[Q zeros(m,h);zeros(h,m) zeros(h)];
for i=1:(m+h)
    [isc(:,i)]=choleski(GL,C(:,i));
end
ctisc=C'*isc;
```

```
P=D+ctisc;
w=isc'*u;
[L,nav]=rozklad(P);
if(nav==0)[z1]=choleski(L,v);
          [z2]=choleski(L,w);
else [z1]=aff(P,v);
     [z2]=aff(P,w);
end
z2=z2+z1;
lam0aff=z2(1:m);
y0aff=-y0-Li*Y0*lam0aff;
y=max(ones(m,1),abs(y0+y0aff));
lam=max(ones(m,1),abs(lam0+lam0aff));
ni=ni0;
x=x0;
u=G*x+c-A'*lam-E'*ni;
rp=A*x-b-y;
rr=E*x-f;
maxit=100;
for k=1:maxit
nav=2;
for i=1:m
    Li(i,i)=1/lam(i);
end
Y=diag(y);
Q=Li*Y;
v=[-rp-y;-rr];
D=[Q zeros(m,h);zeros(h,m) zeros(h)];
P=D+ctisc;
w=isc'*u;
[L,nav]=rozklad(P);
if(nav==0)[z1]=choleski(L,v);
          [z23]=choleski(L,w);
else [z1]=aff(P,v);
     [z23]=aff(P,w);
end
z2=z23+z1;
lamaff=z2(1:m);
yaff=-y-Li*Y*lamaff;
mi=(y'*lam)/m;
alfap=Inf;
for i=1:m
    if(yaff(i)<0) mpri=-y(i)/yaff(i);
```

```
        if(mpri<alfap) alfap=mpri;
        end
    end
end
alfad=Inf;
for i=1:m
    if(lamaff(i)<0) mdual=-lam(i)/lamaff(i);
        if(mdual<alfad) alfad=mdual;
        end
    end
end
alfaaff=min([1;alfap;alfad]);
miaff=((y+alfaaff.*yaff)'*(lam+alfaaff.*lamaff))/m;
sigma=(miaff/mi)^3;
v=[-rp-y-Li*diag(lamaff)*diag(yaff)*ones(m,1)+...
    sigma*mi*Li*ones(m,1);-rr];
if(nav==0)[z1]=choleski(L,v);
else [z1]=aff(P,v);
end
[gu]=choleski(GL,u);
dx=-gu+isc*z23+isc*z1;
z2=z23+z1;
dlam=z2(1:m);
dy=-y-Li*Y*dlam-Li*diag(lamaff)*diag(yaff)*ones(m,1)+...
    sigma*mi*Li*ones(m,1);
dni=z2(m+1:m+h);
alfapr=Inf;
for i=1:m
    if(dy(i)<0) mpri=-y(i)/dy(i);
        if(mpri<alfapr) alfapr=mpri;
        end
    end
end
alfapri=min(1,gama*alfapr);
alfadu=Inf;
for i=1:m
    if(dlam(i)<0) mdual=-lam(i)/dlam(i);
        if(mdual<alfadu) alfadu=mdual;
        end
    end
end
alfadual=min(1,gama*alfadu);
alfa=min(alfapri,alfadual);
```

```
x=x+alfa.*dx;
y=y+alfa.*dy;
lam=lam+alfa.*dlam;
ni=ni+alfa.*dni;
u=G*x+c-A'*lam-E'*ni;
rp=A*x-b-y;
rr=E*x-f;
p1=y'*lam;
if((p1<1e-7)&(norm(u)<1e-7)&(norm(rp)<1e-7)&...
        (norm(rr)<1e-7))break;end;
end
y
disp('Reseni:');
x
disp('Lagrangeovy multiplikátory:');
lam
ni
disp('Pocet kroku:');
k
```

Pomocí této funkce vypočítáme rozklad LL^T matice M, kde L je dolní trojúhelníková matice. Aby bylo možné rozklad provést, musí být M symetrická pozitivně definitní matice.

```
function [L,nav]=rozklad(M)
[n,n]=size(M);
L=zeros(n);
nav=2;
for i=1:n
    for j=i:n
        r=0;
        for k=1:i-1
            r=r+L(i,k)^2;
        end
        m=M(i,i)-r;
        if(m<=0)nav=1;return;end;
        L(i,i)=sqrt(m);
        nav=0;
        s=0;
        for k=1:(i-1)
        s=s+L(i,k)*L(j,k);
        end
        L(j,i)=(M(j,i)-s)/L(i,i);
        end
```

```
end
end
```

Pokud máme rozklad LL^T, můžeme pomocí následující funkce najít řešení soustavy $LL^T x = p$.

```
function [x]=choleski(L,p)
n=length(p);
for i=1:n
    t=0;
    for j=1:i-1
        t=t+L(i,j)*y(j);
    end
    y(i)=(p(i)-t)/L(i,i);
end
LT=L';
for i=n:-1:1
    u=0;
    for j=(i+1):n
        u=u+L(j,i)*x(j);
    end
    x(i)=(y(i)-u)/LT(i,i);
end
x=x';
end
```

V případech, kdy nelze použít Choleského rozklad, používáme na řešení soustav rovnic Gaussovu metodu s částečným výběrem prvku.

```
function[q]=aff(M,p)
%Gaussova metoda
[s,s] = size(M);
T = [M,p];

for i = 1:s
    max = i;
    for r = (i+1):s
        if abs(T(r,i)) > abs(T(max,i))
            max = r;
        end
    end
    if(max ~= i)
        t = T(i,:);
        T(i,:) = T(max,:);
```

```
        T(max,:) = t;
    end
    D(i,i) = 1;
    for j = (i+1):s
        D(j,i) = T(j,i)./T(i,i);
        for l = (i+1):(s+1)
            T(j,l) = T(j,l) - D(j,i) .* T(i,l);
        end
        for l = 1:i
            T(j,l) = 0;
        end
    end
end
%M je dolní trojúhelníková matice
%T je horní trojúhelníková matice(rozšířená s pravou stranou)
for i = s:(-1):1
    q(i) = T(i,s+1);
    for j = s:(-1):(i+1)
        q(i) = q(i) - q(j).*T(i,j);
    end
    q(i) = q(i) ./ T(i,i);
end
q=q';
end
```

Příloha 2

Uvádíme zde programové zpracování algoritmu pro řešení úlohy kvadratického programování s podmínkami ve tvaru rovností a nerovností pomocí primárně-duální metody v programu vytvořeném programovacím jazykem Fortran 77. Soustavy rovnic řešíme pomocí metody Schurova komplementu s využitím Choleského metody a délky kroků volíme stejné pro primární a duální proměnné. Jde o rozšíření *Algoritmu 1* na úlohy s rovnostmi a nerovnostmi. Následující program je určen k řešení *Příkladu 6*.

Tento program je spolu s dalšími přiložen na CD.

```
      PROGRAM PRIMDUALSCHURPR6
      INTEGER N,U,HH,V,I,J,K,NPD,TI
C     MATICE G JE TYPU VxV
C     MATICE A JE TYPU HHxV
C     MATICE E JE TYPU UxV
      PARAMETER(U=2,HH=8,V=8,N=V+HH+U)
      DOUBLE PRECISION LAM(HH),YAFF(HH),YMIAFF(HH),YL(HH),DDD(V),
     1 LL(HH+U,HH+U),P1(V),LAMMIAFF(HH),DLAM(HH),R(V),SSSS(HH),
     2 IDDLAMAFF(HH,HH),GICCZ1(V),DY(HH),DX(V),DDY(HH,HH),D(HH),
     3 DNI(U),NI(U),P(HH+U),A(HH,V),X(V),B(HH),C(V),E(U,V),Y(HH),
     4 LP(HH+U,HH+U),F(U),LAMAFF(HH),EN(V),AT(V,HH),RRR(U),CS(V),
     5 IGC(V,HH+U),ET(V,U),CC(V,HH+U),DD(HH+U,HH+U),P2(HH),P3(U),
     6 IDDLAM(HH,HH),Z23(HH+U),UU(V),PP(HH+U,HH+U),CG(HH+U,HH+U),
     7 SP(HH,HH),DLAMAFF(HH,HH),CT(HH+U,V),Z2(HH+U),W(HH+U),GU(V),
     8 GICCZ2(V),Z1(HH+U),L(V,V),TIGC(HH+U,V),G(V,V),SS(HH),GIC(V)
      DOUBLE PRECISION ALFAPRI,ALFADUAL,MIAFF,YLAM,MI,ALFA,M,MM,KM,
     1 SIGMA,P11,P22,P33,ALFAPAFF,ALFADAFF,YLAMMIAFF,ALFAAFF
      PARAMETER(EPS=1E-7,MAXIT=100,GAMA=0.99)
      OPEN(1,FILE='DATA6.DT')
      OPEN(2,FILE='VYSL6.DT')
      READ(1,*)((G(I,J),J=1,V),I=1,V),(C(I),I=1,V),(X(I),I=1,V),
     1 ((A(I,J),J=1,V),I=1,HH),(B(I),I=1,HH),
     2 ((E(I,J),J=1,V),I=1,U),(F(I),I=1,U)
      DO 1 I=1,V
      DO 2 J=1,V
      G(I,J)=2*G(I,J)
2     END DO
1     END DO
      DO 3 I=1,HH
      Y(I)=1.
      LAM(I)=1.
3     END DO
```

```
      DO 4 I=1,U
      NI(I)=1.
4     END DO
      CALL TRANS(A,AT,HH,V)
      CALL TRANS(E,ET,U,V)
      CALL VYPMATL(G,L,V)
      DO 5 I=1,V
      DO 6 J=1,HH
      CC(I,J)=AT(I,J)
6     END DO
      DO 7 J=1,U
      CC(I,HH+J)=ET(I,J)
7     END DO
5     END DO
      DO 8 I=1,HH+U
      DO 9 J=1,HH+U
      DD(I,J)=0.
9     END DO
8     END DO
      CALL INV(LAM,IDDLAM,HH)
      CALL MATXVEC(IDDLAM,Y,YL,HH,HH)
      DO 10 I=1,HH
      DD(I,I)=YL(I)
10    END DO
      DO 11 I=1,HH+U
      DO 12 J=1,V
      DO 13 K=1,V
      CS(K)=CC(K,I)
13    END DO
      CALL CHOLMET(L,CS,GIC,V)
      IGC(J,I)=GIC(J)
12    END DO
11    END DO
      CALL TRANS(CC,CT,V,HH+U)
      CALL MATXMAT(CT,IGC,CG,HH+U,V,HH+U)
      DO 14 I=1,HH+U
      DO 15 J=1,HH+U
      PP(I,J)=DD(I,J)+CG(I,J)
15    END DO
14    END DO
      CALL MATXVEC(G,X,DDD,V,V)
      CALL MATXVEC(AT,LAM,R,V,HH)
      CALL MATXVEC(ET,NI,EN,V,U)
```

```
      DO 16 I=1,V
      UU(I)=DDD(I)+C(I)-R(I)-EN(I)
16    END DO
      CALL MATXVEC(A,X,D,HH,V)
      DO 17 I=1,HH
      P(I)=-(D(I)-B(I))
17    END DO
      CALL MATXVEC(E,X,RRR,U,V)
      DO 18 I=1,U
      P(HH+I)=-(RRR(I)-F(I))
18    END DO
      CALL TRANS(IGC,TIGC,V,HH+U)
      CALL MATXVEC(TIGC,UU,W,HH+U,V)
C     RESENI SOUSTAVY
      DO 19 I=1,HH+U
      DO 20 J=1,HH+U
      LP(I,J)=0.
20    END DO
19    END DO
      DO 21 I=1,HH+U
      DO 22 J=I,HH+U
      MM=0.
      DO 23 K=1,I-1
      MM=MM+LP(I,K)**2
23    END DO
      M=PP(I,I)-MM
      IF(M.LE.0.)GO TO 600
      LP(I,I)=SQRT(M)
      KM=0.
      DO 24 K=1,I-1
      KM=KM+LP(I,K)*LP(J,K)
24    END DO
      LP(J,I)=(PP(J,I)-KM)/LP(I,I)
22    END DO
21    END DO
      CALL CHOLMET(LP,P,Z1,HH+U)
      CALL CHOLMET(LP,W,Z2,HH+U)
      GO TO 700
600   CONTINUE
      CALL AFF(PP,P,Z1,HH+U)
      CALL AFF(PP,W,Z2,HH+U)
700   CONTINUE
      CALL CHOLMET(L,UU,GU,V)
```

```
      DO 25 I=1,HH+U
      Z2(I)=Z2(I)+Z1(I)
25    END DO
      DO 26 I=1,HH
      LAMAFF(I)=Z2(I)
26    END DO
      CALL DIAG(Y,DDY,HH)
      CALL INV(LAM,IDDLAM,HH)
      CALL MATXMAT(IDDLAM,DDY,SP,HH,HH,HH)
      CALL MATXVEC(SP,LAMAFF,SS,HH,HH)
      DO 27 I=1,HH
      YAFF(I)=-Y(I)-SS(I)
27    END DO
      DO 28 I=1,HH
      Y(I)=MAX(1,ABS(Y(I)+YAFF(I)))
28    END DO
      DO 29 I=1,HH
      LAM(I)=MAX(1,ABS(LAM(I)+LAMAFF(I)))
29    END DO
      CALL MATXVEC(AT,LAM,R,V,HH)
      DO 30 I=1,V
      P1(I)=DDD(I)+C(I)-R(I)-EN(I)
30    END DO
      DO 31 I=1,HH
      P2(I)=D(I)-B(I)-Y(I)
31    END DO
      DO 32 I=1,U
      P3(I)=RRR(I)-F(I)
32    END DO
C     ZACATEK CYKLU
      DO 100 TI=1,MAXIT
      NPD=2.
      CALL INV(LAM,IDDLAM,HH)
      CALL MATXVEC(IDDLAM,Y,YL,HH,HH)
      DO 33 I=1,HH
      DD(I,I)=YL(I)
33    END DO
      DO 34 I=1,HH+U
      DO 35 J=1,HH+U
      PP(I,J)=DD(I,J)+CG(I,J)
35    END DO
34    END DO
      DO 36 I=1,V
```

```
      UU(I)=P1(I)
36    END DO
      DO 37 I=1,HH
      P(I)=-P2(I)-Y(I)
37    END DO
      DO 38 I=1,U
      P(HH+I)=-P3(I)
38    END DO
      CALL MATXVEC(TIGC,UU,W,HH+U,V)
C     RESENI SOUSTAVY
      DO 39 I=1,HH+U
      DO 40 J=1,HH+U
      LL(I,J)=0.
40    END DO
39    END DO
      DO 41 I=1,HH+U
      DO 42 J=I,HH+U
      MM=0.
      DO 43 K=1,I-1
      MM=MM+LL(I,K)**2
43    END DO
      M=PP(I,I)-MM
      IF(M.LE.0.)GO TO 800
      LL(I,I)=SQRT(M)
      KM=0.
      DO 44 K=1,I-1
      KM=KM+LL(I,K)*LL(J,K)
44    END DO
      LL(J,I)=(PP(J,I)-KM)/LL(I,I)
42    END DO
41    END DO
      CALL CHOLMET(LL,P,Z1,HH+U)
      CALL CHOLMET(LL,W,Z23,HH+U)
      GO TO 900
800   CONTINUE
      NPD=1.
      CALL AFF(PP,P,Z1,HH+U)
      CALL AFF(PP,W,Z23,HH+U)
900   CONTINUE
      CALL CHOLMET(L,UU,GU,V)
      CALL MATXVEC(IGC,Z23,GICCZ2,V,HH+U)
      DO 45 I=1,HH+U
      Z2(I)=Z23(I)+Z1(I)
```

```
45    END DO
      DO 46 I=1,HH
      LAMAFF(I)=Z2(I)
46    END DO
      CALL DIAG(Y,DDY,HH)
      CALL MATXMAT(IDDLAM,DDY,SP,HH,HH,HH)
      CALL MATXVEC(SP,LAMAFF,SS,HH,HH)
      DO 47 I=1,HH
      YAFF(I)=-Y(I)-SS(I)
47    END DO
      CALL PODKRIT(Y,YAFF,ALFAPAFF,HH)
      CALL PODKRIT(LAM,LAMAFF,ALFADAFF,HH)
      ALFAAFF=MIN(1.,ALFAPAFF,ALFADAFF)
      CALL SKSOUC(Y,LAM,YLAM,HH)
      MI=(YLAM)/HH
      DO 48 I=1,HH
      YMIAFF(I)=Y(I)+ALFAAFF*YAFF(I)
48    END DO
      DO 49 I=1,HH
      LAMMIAFF(I)=LAM(I)+ALFAAFF*LAMAFF(I)
49    END DO
      CALL SKSOUC(YMIAFF,LAMMIAFF,YLAMMIAFF,HH)
      MIAFF=(YLAMMIAFF)/HH
      SIGMA=(MIAFF/MI)**3
      CALL DIAG(LAMAFF,DLAMAFF,HH)
      CALL MATXMAT(IDDLAM,DLAMAFF,IDDLAMAFF,HH,HH,HH)
      CALL MATXVEC(IDDLAMAFF,YAFF,SS,HH,HH)
      DO 50 I=1,HH
      P(I)=-(D(I)-B(I))+SIGMA*MI*IDDLAM(I,I)-SS(I)
50    END DO
C     RESENI SOUSTAVY
      IF(NPD==1.)CALL AFF(PP,P,Z1,HH+U)
      IF(NPD==2.)CALL CHOLMET(LL,P,Z1,HH+U)
      CALL MATXVEC(IGC,Z1,GICCZ1,V,HH+U)
      DO 51 I=1,V
      DX(I)=-GU(I)+GICCZ2(I)+GICCZ1(I)
51    END DO
      DO 52 I=1,HH+U
      Z2(I)=Z23(I)+Z1(I)
52    END DO
      DO 53 I=1,HH
      DLAM(I)=Z2(I)
53    END DO
```

```
      DO 54 I=1,U
      DNI(I)=Z2(HH+I)
54    END DO
      CALL MATXMAT(IDDLAM,DDY,SP,HH,HH,HH)
      CALL MATXVEC(SP,DLAM,SSSS,HH,HH)
      DO 55 I=1,HH
      DY(I)=-Y(I)-SSSS(I)-SS(I)+SIGMA*MI*IDDLAM(I,I)
55    END DO
      CALL PODKRIT(Y,DY,ALFAPRI,HH)
      ALFAPRI=MIN(1.,GAMA*ALFAPRI)
      CALL PODKRIT(LAM,DLAM,ALFADUAL,HH)
      ALFADUAL=MIN(1.,GAMA*ALFADUAL)
      ALFA=MIN(ALFAPRI,ALFADUAL)
      DO 56 I=1,V
      X(I)=X(I)+ALFA*DX(I)
56    END DO
      DO 57 I=1,HH
      Y(I)=Y(I)+ALFA*DY(I)
57    END DO
      DO 58 I=1,HH
      LAM(I)=LAM(I)+ALFA*DLAM(I)
58    END DO
      DO 59 I=1,U
      NI(I)=NI(I)+ALFA*DNI(I)
59    END DO
C     PODMINKA PRO UKONCENI CYKLU
      CALL SKSOUC(Y,LAM,YLAM,HH)
      CALL MATXVEC(G,X,DDD,V,V)
      CALL MATXVEC(AT,LAM,R,V,HH)
      CALL MATXVEC(ET,NI,EN,V,U)
      DO 60 I=1,V
      P1(I)=DDD(I)+C(I)-R(I)-EN(I)
60    END DO
      CALL MATXVEC(A,X,D,HH,V)
      DO 61 I=1,HH
      P2(I)=D(I)-B(I)-Y(I)
61    END DO
      CALL MATXVEC(E,X,RRR,U,V)
      DO 62 I=1,U
      P3(I)=RRR(I)-F(I)
62    END DO
      CALL NORM(P1,P11,V)
      CALL NORM(P2,P22,HH)
```

```
      CALL NORM(P3,P33,U)
      IF((P11.LT.EPS).AND.(P22.LT.EPS).AND.(P33.LT.EPS).AND.
     2     (YLAM.LT.EPS))THEN
           WRITE(2,*)'X=',(X(I),I=1,V),'Y=',(Y(I),I=1,HH),'LAM',
     1     (LAM(I),I=1,HH),'NI=',(NI(I),I=1,U),'K=',TI,
     2      'Konec vypoctu'
                STOP
                ENDIF
100   END DO
      END
```

V hlavním programu používáme několik podprogramů. Prvním z nich je podprogram na součin matice a vektoru.

```
      SUBROUTINE MATXVEC(MAT,VEC,MATVEC,H,HH)
      INTEGER H,HH,I,J
      DOUBLE PRECISION MAT(H,HH),VEC(HH),MATVEC(H)
      DO 1 I=1,H
      MATVEC(I)=0.
      DO 2 J=1,HH
      MATVEC(I)=MATVEC(I)+MAT(I,J)*VEC(J)
2     END DO
1     END DO
      END
```

Tento podprogram nám spočítá skalární součin vektorů.

```
      SUBROUTINE SKSOUC(VEC1,VEC2,SKSOU,H)
      INTEGER H,I,J
      DOUBLE PRECISION VEC1(H),VEC2(H),SKSOU
      SKSOU=0.
      DO 1 I=1,H
      SKSOU=SKSOU+VEC1(I)*VEC2(I)
1     END DO
      END
```

K výpočtu ukončovacího kritéria potřebujeme znát eukleidovskou normu vektoru. Vytvořili jsme si tedy podprogram s názvem *NORM*.

```
      SUBROUTINE NORM(VEC,NVEC,H)
      INTEGER H
      DOUBLE PRECISION VEC(H),NVEC,SKS
      CALL SKSOUC(VEC,VEC,SKS,H)
      NVEC=SQRT(SKS)
      END
```

K výpočtu inverze diagonální matice slouží následující podprogram.

```
      SUBROUTINE INV(VEC,MAT,H)
      INTEGER H,I,J
      DOUBLE PRECISION VEC(H),MAT(H,H)
      DO 1 I=1,H
      DO 2 J=1,H
      MAT(I,J)=0.
2     END DO
      MAT(I,I)=1/(VEC(I))
1     END DO
      END
```

Jelikož při výpočtu potřebujeme transpozice matic, vytvořili jsme si podprogram s názvem *TRANS*.

```
      SUBROUTINE TRANS(MAT,TMAT,H,HH)
      INTEGER H,HH,I,J
      DOUBLE PRECISION MAT(H,HH),TMAT(HH,H)
      DO 1 I=1,H
      DO 2 J=1,HH
      TMAT(J,I)=MAT(I,J)
2     END DO
1     END DO
      END
```

Díky tomuto podprogramu vytvoříme diagonální matici z vektoru.

```
      SUBROUTINE DIAG(VEC,MAT,H)
      INTEGER H,I,J
      DOUBLE PRECISION VEC(H),MAT(H,H)
      DO 1 I=1,H
      DO 2 J=1,H
      MAT(I,J)=0.
2     END DO
      MAT(I,I)=VEC(I)
1     END DO
      END
```

K výpočtu součinu dvou matic použijeme podprogram s názvem *MATXMAT*.

```
      SUBROUTINE MATXMAT(MAT1,MAT2,MXM,H,HH,HHH)
      INTEGER H,HH,HHH,J,K,L
      DOUBLE PRECISION MAT1(H,HH),MAT2(HH,HHH),MXM(H,HHH)
      DO 1 J=1,H
      DO 2 K=1,HHH
      MXM(J,K)=0.
      DO 3 L=1,HH
      MXM(J,K)=MXM(J,K)+MAT1(J,L)*MAT2(L,K)
3     END DO
2     END DO
1     END DO
      END
```

Následující podprogram používáme k výpočtu délek kroků.

```
      SUBROUTINE PODKRIT(VEC,DVEC,MINMVEC,H)
      INTEGER H,I
      DOUBLE PRECISION VEC(H),DVEC(H),MVEC,MINMVEC
      MINMVEC=1E+15
      MVEC=1E+15
      DO 1 I=1,H
      IF (DVEC(I).LT.0.) MVEC=-VEC(I)/DVEC(I)
      IF (MVEC.LT.MINMVEC) MINMVEC=MVEC
1     END DO
      END
```

Soustavy lineárních rovnic můžeme řešit Gaussovou metodou s částečným výběrem prvku.

```
      SUBROUTINE AFF(M,P,Q,N)
      INTEGER N,I,J,L,MX
      DOUBLE PRECISION M(N,N),P(N),T(N,N+1),TT(N+1),Q(N),D(N,N)
      DO 1 I=1,N
      DO 2 J=1,N
      T(I,J)=M(I,J)
2     END DO
1     END DO
      DO 3 I=1,N
      T(I,N+1)=P(I)
3     END DO
      DO 4 I=1,N
      MX=I
      DO 5 J=I+1,N
```

```
      IF(ABS(T(J,I)).GT.ABS(T(MX,I))) MX=J
5     END DO
      IF(MX==I) GO TO 30
      DO 6 J=1,N+1
      TT(J)=T(I,J)
6     END DO
      DO 7 J=1,N+1
      T(I,J)=T(MX,J)
7     END DO
      DO 8 J=1,N+1
      T(MX,J)=TT(J)
8     END DO
30    CONTINUE
      D(I,I)=1.
      DO 9 J=I+1,N
      D(J,I)=T(J,I)/T(I,I)
      DO 10 L=(I+1),(N+1)
      T(J,L)=T(J,L)-D(J,I)*T(I,L)
10    END DO
      DO 11 L=1,I
      T(J,L)=0.
11    END DO
9     END DO
4     END DO
      DO 12 I=N,1,-1
      Q(I)=T(I,N+1)
      DO 13 J=N,I+1,-1
      Q(I)=Q(I)-Q(J)*T(I,J)
13    END DO
      Q(I)=Q(I)/T(I,I)
12    END DO
      END
```

K výpočtu rozkladu LL^T, kde L je dolní trojúhelníková matice, použijeme následující podprogram.

```
      SUBROUTINE VYPMATL(MAT,L,N)
      INTEGER I,J,K,N
      DOUBLE PRECISION MAT(N,N),L(N,N)
      DOUBLE PRECISION R,S,M
C     VYPOCET DOLNI TROJUHELNIKOVE MATICE
      DO 1 J=1,N
      DO 2 I=1,N
      L(I,J)=0.
```

```
2     END DO
1     END DO
      DO 3 I=1,N
      DO 4 J=I,N
      R=0.
      DO 5 K=1,I-1
      R=R+L(I,K)**2
5     END DO
      M=MAT(I,I)-R
      L(I,I)=SQRT(M)
      S=0.
      DO 6 K=1,I-1
      S=S+L(I,K)*L(J,K)
6     END DO
      L(J,I)=(MAT(J,I)-S)/L(I,I)
4     END DO
3     END DO
      END
```

Choleského rozklad LL^T využijeme k řešení soustav rovnic.

```
      SUBROUTINE CHOLMET(L,B,X,N)
      INTEGER I,J,N
      DOUBLE PRECISION B(N),Y(N),X(N),L(N,N),LTR(N,N)
      DOUBLE PRECISION T,U
C     VYPOCET VEKTORU Y
      DO 1 I=1,N
      T=0.
      DO 2 J=1,I-1
      T=T+L(I,J)*Y(J)
2     END DO
      Y(I)=(B(I)-T)/L(I,I)
1     END DO
      CALL TRANS(L,LTR,N,N)
C     VYPOCET VEKTORU X
      DO 3 I=N,1,-1
      U=0.
      DO 4 J=I+1,N
      U=U+L(J,I)*X(J)
4     END DO
      X(I)=(Y(I)-U)/LTR(I,I)
3     END DO
      END
```