MATEMATICKO - FYZIKÁLNÍ FAKULTA

UNIVERSITY KARLOVY V PRAZE

Metoda vnitřního bodu pro rozsáhlé úlohy nelineárního programování

Vedoucí diplomové práce: Ing. Ladislav Lukšan, DrSc.
Studijní zameření: Výpočtová matematika - software

Srpen 1999

Jan Macek

Děkuji Ing. Ladislavu Lukšanovi, DrSc. za cenné rady, připomínky a zapůjčenou literaturu, které mi poskytoval v průběhu vypracování této práce.

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním této práce.

V Praze dne 23.8.1999

....

Jan Macek

Obsah

Úvod

Metody vnitřního bodu jsou dnes jedním z nejvýznamějších a nejsledovanějších témat v oblasti optimalizace. Nebylo tomu však vždy. Ve formě bariérových metod se metody vnitřního bodu hojně používaly zejména v 60. letech pro řešení nelineárně omezených optimalizačních problémů. V 70. letech byly bariérové metody zatlačeny do ústraní nově se objevujícími a zjevně efektivnějšími metodami, jako je třeba sekvenční kvadratické programování (SQP). Jedním z hlavních důvodů ztráty popularity bariérových metod byla špatná podmíněnost matice primární úlohy, která se však později ukázala být téměř neškodnou. Začátkem 80. let byly bariérové metody považovány za uzavřenou kapitolu historie. Vše se však změnilo v roce 1984, kdy Karmarkar představil svou „rychlou“ metodu, která řešila úlohy lineárního programování v polynomiálním čase, a u které bylo v roce 1985 ukázáno formální spojení s klasickými bariérovými metodami. Až do roku 1984 se totiž k řešení lineárních a nelineárních problémů přistupovalo zcela odlišně. Pro řešení lineárních úloh, které ve standardní podobě zapisujeme jako $\min_{x \in R^n} c^T x$ tak, aby platilo $Ax = b$, $x \geq 0$, kde x, b a c jsou vektory a A je matice, se téměř bez vyjímky používala (a dodnes používá) simplexová metoda.

Simplexová metoda, kterou v roce 1947 vynalezl George B. Dantzig, využívá základní vlastnosti prakticky všech lineárních úloh, a to, že řešení takové úlohy se nachází v některém z vrcholů oblasti dané lineárními vazbami $Ax = b$. Tato metoda v praxi vykazuje velmi dobré výsledky, i když pracuje v exponenciálním čase. Je totiž možné sestavit příklad tak, aby simplexová metoda musela prohledat všechny vrcholy oblasti, než najde řešení [10]. Karmarkarova metoda byla na některých úlohách až 50 krát rychlejší než simplexová metoda a na mnoha dalších byla srovnatelně rychlá.

Od roku 1984 se tedy naskýtá jedinečná možnost dívat se na úlohy lineárního i nelineárního programování z jediného úhlu pohledu. Dnes by již nikdo neřekl, že lineární programování je nezávislé a odděleno od nelineárního programování (a samozřejmě opačně). Nedá se však říci, že se pouze převzaly výsledky prací z 60. let a začaly se od 80. let opět používat. Od té doby vývoj metod vnitřního bodu značně pokročil, hlavně v oblasti teorie složitosti, vylepšení algoritmů, lineární algebry a popisu tříd řešitelných problémů. Zejména na téma teorie složitosti bylo sepsáno velké množství prací. Nepostradatelný přínos do této teorie, bez kterého by dnes nebyla tím, čím je, byla bezpochyby práce Nesterova a Nemirovského z konce 80. let, která rozšířila výsledky polynomiální složitosti na širokou třídu konvexních optimalizačních problémů. Jedním z hlavních přínosů této práce byla definice self-concordantních funkcí. Konvexní funkce Φ na konvexní množině $\mathcal{F}^0 \in \mathrm{R}^n$ je κ-selfconcordantní, pokud je třikrát spojitě diferencovatelná a pokud pro všechny $y \in \mathcal{F}^0$ a všechny $h \in \mathrm{R}^n$ platí, že $|\nabla^3 \Phi(y)[h, h, h]| \leq 2\kappa(h^T \nabla^2 \Phi(y) h)^{2/3}$. Self-concordantní s $\kappa = 1$ je například logaritmická bariérová funkce pro lineární a konvexní kvadratické programování. Existence self-concordantní bariérové funkce je úzce spjata s polynomiální složitostí algoritmu. I u algoritmů metod vnitřního bodu však stejně jako u simplexové metody zůstává záhadou, proč v praxi vyžadují daleko menší počet iterací, než je jejich horní polynomiální hranice.

Princip self-concordance umožňuje definovat nové bariérové funkce pro některé druhy konvexního programování, jako je třeba semidefinitní programování (SDP). SDP je možné chápat jako zobecnění lineárního programování, kde v roli proměnných vystupují symetrické $n \times n$ matice X. V SDP se snažíme minimalizovat funkci v proměnné X tak, aby platily lineární vazby a podmínka, že X je pozitivně semidefinitní matice. Problém SDP můžeme tedy zapsat jako $\min_{X \in \mathbb{S}^n} \operatorname{tr}(CX)$ tak, aby platilo $\operatorname{tr}(A_i X) = b_i$ a $X \succeq 0$, kde C a A_i jsou symetrické $n \times n$ matice a b_i jsou vektory.

V této práci se zaměříme na problémy nelineárního nekonvexního programování a jejich řešení pomocí logaritmické bariérové metody.

Kapitola 1

Formulace úlohy

1.1 Nelineární programování

Účel nelineárního programování je stejný jako u lineárního programování, jen objekt zájmu se z lineárních funkcí změní na nelineární. V dalším textu budeme používat následující značení indexů: zavedeme indexové množiny $I = \{1, \ldots, m_I\}$ a $E = \{m_I + 1, \ldots, m_I + m_E\}$. Pod označením $c_I(x)$, resp. $c_E(x)$, budeme myslet vektor o složkách $c_i(x)$, $i \in I$, resp. $c_i(x)$, $i \in E$. Uvažujme problém najít minimum funkce $f(x)$:

$$\min_{x \in \mathrm{R}^{\mathrm{n}}} f(x) \quad \text{tak, aby} \quad c_E(x) = 0,\, c_I(x) \leq 0, \tag{1.1}$$

kde $f : \mathrm{R}^{\mathrm{n}} \to \mathrm{R}$, $c_E : \mathrm{R}^{\mathrm{n}} \to \mathrm{R}^{\mathrm{m_E}}$ a $c_I : \mathrm{R}^{\mathrm{n}} \to \mathrm{R}^{\mathrm{m_I}}$ jsou dvakrát spojitě diferencovatelné funkce a nerovnost $c_I(x) \leq 0$ je míněna po složkách. Podobně jako u lineárního programování lze takto formulovanou úlohu převést na úlohu s rovnostními omezeními. Zavedeme pomocné proměnné $s_I \in \mathrm{R}^{\mathrm{m_I}}$. U těchto proměnných však můžeme vynechávat index I, protože zde nehrozí záměna. Vektor s tedy bude vždy znečit vektor o složkách s_i, $i \in I$. S pomocí pomocných proměnných přeformulujeme úlohu na

$$\min_{x \in \mathrm{R}^{\mathrm{n}}} f(x) \quad \text{tak, aby} \quad c_E(x) = 0,\, c_I(x) + s = 0,\, s \geq 0. \tag{1.2}$$

Výhodou tohoto způsobu je snadné určení přípustnosti vektoru $[x^T, s^T]^T \in \mathrm{R}^{\mathrm{n+m_I}}$ vhodnou volbou vektoru s.

1.2 Základní definice

Definice 1 (*Přípustná oblast pro 1.1*) Pro minimalizační úlohu (1.1) budeme přípustnou oblastí myslet množinu bodů

$$\mathcal{F}_1 = \{x \,|\, c_i(x) = 0,\, i \in E;\, c_i(x) \leq 0,\, i \in I\} \tag{1.3}$$

Pokud bude platit, že bod $x \in \mathcal{F}_1$, potom budeme říkat, že x je přípustný.

Definice 2 (*Přípustná oblast pro 1.2*) Pro minimalizační úlohu (1.2) budeme přípustnou oblastí myslet množinu bodů

$$\mathcal{F}_2 = \{(x, s) \,|\, c_i(x) = 0,\, i \in E;\, c_i(x) + s_i = 0,\, s_i \geq 0;\, i \in I\} \tag{1.4}$$

Lemma 1 *Nechť $x^* \in \mathcal{F}_1$, potom existuje $s \geq 0$ tak, že $(x^*, s) \in \mathcal{F}_2$, a naopak nechť $(x^*, s) \in \mathcal{F}_2$, potom $x^* \in \mathcal{F}_1$.*

Důkaz
K první implikaci: vzhledem k tomu, že pokud $x^* \in \mathcal{F}_1$ je $c_i(x^*) \leq 0$, $i \in I$, stačí brát $s_i = -c_i(x^*)$. Zbytek je zřejmý.

Vzhledem k výsledku lematu 1, budeme přípustnou oblast pro úlohu (1.1) i (1.2) značit $\mathcal{F}$.

Definice 3 (*Striktní přípustnost*) Bod (x^*, s^*) budeme nazývat striktně přípustný, pokud platí, že $s^* > 0$. Striktně přípustnou oblastí budeme nazývat množinu bodů (x, s), které jsou striktně přípustné, a budeme ji značit $\mathcal{F}^0$

Poznámka Definice 3 definuje striktní přípustnost pro formulaci (1.2). Striktní přípustnost pro formulaci (1.1) lze definovat jako $c_i(x^*) < 0$, $i \in I$. Zde je vidět výhoda formulace (1.2).

Definice 4 (*Metoda vnitřního bodu*) Za metodu vnitřního bodu budeme považovat iterativní metodu, která generuje iterace, které jsou striktně přípustné.

Algoritmus 1 (*Newtonova metoda*)

Krok 1	Dáno x_0, $F(x)$ spojitě diferencovatelná funkce; $k \leftarrow 0$
Krok 2	Vyřešit soustavu $F'(x_k)\triangle x = -F(x_k)$ pro proměnnou $\triangle x$
Krok 3	Dosadit $x_{k+1} = x_k + \alpha_k \triangle x$, $0 < \alpha_k \leq 1$, $k = k + 1$, zpět na Krok 2

Předpokládáme, že Newtonova posloupnost $\{x_k\}$ konverguje k bodu x^*, pro který platí $F(x^*) = 0$. Podle [17] je tato metoda skutečně algoritmicky konzistentní (tj. pokud posloupnost $\{x_k\}$ konverguje, potom konverguje k bodu x^*) a je lokálně kvadraticky konvergentní (pokud bude $\alpha_k \to 1$ dostatečně rychle).

Definice 5 (*Aktivní omezení*) Nechť x^* je vázané lokální minimum úlohy (1.2). Omezení $c_i(x)$ budeme nazývat aktivním v bodě x^*, pokud bude platit, že $c_i(x^*) = 0$ (a tedy $s_i = 0$). Omezení nazveme *neaktivní* v bodě x^*, pokud není aktivní. Všechny omezení $c_i(x)$, $i \in E$ jsou samozřejmě aktivní.

1.3 Podmínky optimality

Podmínky optimality jsou nesmírně důležité. Umožňují nejenom rozeznat nalezení řešení, ale zároveň nám dávají návod k nalezení algoritmu pro jeho hledání.

Uvažujme nyní problém (1.2). Budeme vycházet z Lagrangeovy funkce pro tuto formulaci, která má tvar

$$L(x, \lambda, s) = f(x) + \lambda_E^T c_E(x) + \lambda_I^T (c_I(x) + s), \tag{1.5}$$

kde λ_E a λ_I jsou Lagrangeovy multiplikátory. Pro úplnost zavedeme ještě Hessovu matici Lagrangeovy funkce (1.5) vzhledem k x, kterou označíme G, a která je dána

$$G(x, \lambda, s) = \nabla_x^2 L(x, \lambda) = \nabla_x^2 f(x) + \sum_{i \in E} \lambda_i \nabla^2 c_i(x) + \sum_{i \in I} \lambda_i \nabla^2 c_i(x). \tag{1.6}$$

Podmínky pro vázaný extrém funkce vycházejí z Karush-Kuhn-Tuckerovy (KKT) věty. Nutné KKT podmínky (prvního řádu) pro extrém funkce v bodě $[x^*, \lambda^*, s^*]^T$ (1.5) mají tvar

$$\begin{aligned} c_I(x^*) + s^* &= 0,\ s^* \geq 0 && (1.7)\\ c_E(x^*) &= 0 && (1.8)\\ \nabla f(x^*) + A_E(x^*)^T \lambda_E^* + A_I(x^*)^T \lambda_I^* &= 0 \text{ pro nějaké } \lambda_E^* \text{ a } \lambda_I^* && (1.9)\\ \lambda_i^* c_i(x) &= 0,\ i \in I && (1.10)\\ \lambda_i^* &\geq 0,\ i \in I && (1.11) \end{aligned}$$

kde $A_E(x^*)$ resp. $A_I(x^*)$ je Jacobiho matice funkce $c_E(x)$ resp. $c_I(x)$ v bodě x^*. KKT podmínka druhého řádu má tvar

$$N(x^*)^T G(x^*, \lambda^*) N(x^*) \geq 0 \tag{1.12}$$

kde N je báze jádra lineárního zobrazení $\hat{A}$, kde $\hat{A}$ je Jacobiho matice všech omezení, která jsou aktivní v bodě x^* (tj. všechna rovnostní omezení a aktivní nerovnostní omezení). Matici $N^T G N$ se říká *redukovaná Hessova matice.*

Dostačující podmínky pro to, aby x^* byl izolovaný vázaný bod minima (1.2), jsou:

1. gradienty aktivních omezení jsou v bodě x^* lineárně nezávislé
2. platí podmínky (1.7)-(1.10)
3. platí tyto zesílené verze podmínek (1.11) a (1.12)

$$\begin{aligned} \lambda_i^* &> 0 \quad \text{pokud} \quad c_i(x^*) = 0 && (1.13)\\ N(x^*)^T G(x^*, \lambda^*) N(x^*) &> 0 && (1.14) \end{aligned}$$

Nerovnost (1.13) se nazývá *striktní komplementaritou* a platí, pokud jsou všechny Lagrangeovy multiplikátory spojené s aktivními omezeními kladné. Vlastnost striktní komplementarity se často bere jako předpoklad, protože nulovost Lagrangeových multiplikátorů u aktivních omezení způsobuje při výpočtech problémy (viz. 3.1, Postup centrální cesty podél aktivního omezení a 2.1 Určení měrového vektoru). Podmínka (1.14) je ekvivalentní s existencí $\alpha > 0$ takového, že

$$p^T G(x^*, \lambda^*) p \geq \alpha \|p\|^2 \quad \text{pro všechna } p \text{ taková, že} \quad \hat{A}(x^*) p = 0 \tag{1.15}$$

kde $\hat{A}$ je Jacobiho matice aktivních omezení (viz. výše).

1.4 Logaritmická bariérová metoda

Princip bariérové metody spočívá v nahrazení účelové funkce funkcí složenou. Tato složená funkce obsahuje jak původní účelovou funkci, tak i bariérový člen. Bariérový člen je navržen tak, aby měl na hranici striktně přípustné oblasti kladnou singularitu. Logaritmické bariérová metoda, která používá k vytvoření singularity přirozený logaritmus, je iterační metoda, u které se všechny iterace, díky nepřekročitelné singularitě na okraji striktně přípustné oblasti, drží uvnitř této oblasti. Odtud metoda vnitřího bodu. To vše za předpokladu, že se počáteční bod nachází uvnitř přípustné oblasti. Tento požadavek může někdy způsobit problémy, avšak díky formulaci úlohy (1.2) není težké ho splnit.

Uvažujme tedy minimalizační úlohu (1.1). Nejprve nahradíme nerovnosti $c_i(x) \leq 0$, $i \in I$ přidáním pomocných proměnných $s \in \mathrm{R}^{m_I}$ rovnostmi $c_i(x) + s_i = 0$, $s_i \geq 0$, $i \in I$. Tím dostaneme úlohu (1.2). Nyní v duchu bariérových metod nahradíme účelovou funkci $f(x)$ funkcí složenou tak, že k ní přidáme logaritmický bariérový člen. Dostaneme úlohu

$$\min_{x \in \mathrm{R}^n} f(x) - \mu \sum_{i \in I} \ln s_i \quad \text{tak, aby} \quad c_E(x) = 0,\ c_I(x) + s = 0 \tag{1.16}$$

Takto formulovaná úloha má omezení pouze ve tvaru rovnosti. Nerovnost $s \geq 0$ je implicitně dána v $\ln s_i$, neboť přirozený logaritmus jiné než kladné hodnoty s nepřipouští (zahrnuje dokonce striktní přípustnost $s > 0$). Algoritmus musí být tedy navržen tak, aby po celou dobu výpočtu udržoval hodnoty s kladné. Lagrangeova funkce pro (1.16) má tvar

$$L(x, s, \lambda) = f(x) - \mu \sum_{i \in I} \ln s_i + \lambda_E^T c_E(x) + \lambda_I^T (c_I(x) + s) \tag{1.17}$$

Bariérový parametr μ je zde kladné reálné číslo. Jeho význam bude diskutován později, v tomto místě pouze poznamenejme, že pro $\mu = 0$ dostaneme z (1.17) původní Lagrangeovu funkci (1.5). Pro další postup bude užitečné vyjádřit si gradient g a Hessovu matici G této Lagrangeovy funkce vzhledem k proměnné x. Nechť tedy

$$g(x) = \frac{\partial}{\partial x} L(x, s, \lambda) = \nabla f(x) + \sum_{i \in E} \lambda_i \nabla c_i(x) + \sum_{i \in I} \lambda_i \nabla c_i(x) \tag{1.18}$$

$$G(x) = \frac{\partial^2}{\partial x^2} L(x, s, \lambda) = \nabla^2 f(x) + \sum_{i \in E} \lambda_i \nabla^2 c_i(x) + \sum_{i \in I} \lambda_i \nabla^2 c_i(x) \tag{1.19}$$

Nyní již můžeme zformulovat základní rovnice pro výpočet směrového vektoru. Označme $S = \operatorname{diag}(s_1, \ldots, s_{m_I})$, $\Lambda_I = \operatorname{diag}(\lambda_{m_E+1}, \ldots, \lambda_{m_E+m_I})$ a $e = [1, \ldots, 1]$ vektor velikosti m_I, složený ze samých jedniček. Na základě nutných podmínek optimality, tj. nulovosti derivace Lagrangeovy funkce (1.17) podle proměnných x, s, λ_E a λ_I, dostáváme

$$\begin{aligned} g(x) &= 0 && (1.20)\\ -\mu S^{-1}e + \lambda_I &= 0 && (1.21)\\ c_E(x) &= 0 && (1.22)\\ c_I(x) + s &= 0 && (1.23)\end{aligned}$$

Pro výpočet směrového vektoru použijeme Newtonovu metodu. Na základě soustavy rovnic (1.20) - (1.23) dostaneme iterační metodu, kde se v každém kroku řeší soustava lineárních rovnic (tam, kde nebude hrozit nedorozumění, budeme u funkcí vynechávat parametry):

$$\begin{bmatrix} G & 0 & A_E & A_I \\ 0 & \mu S^{-2} & 0 & I \\ A_E^T & 0 & 0 & 0 \\ A_I^T & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle s \\ \triangle \lambda_E \\ \triangle \lambda_I \end{bmatrix} = \begin{bmatrix} -g \\ \mu S^{-1}e - \lambda_I \\ -c_E \\ -(c_I + s) \end{bmatrix} \tag{1.24}$$

Takto formulovaná soustava nám dává tzv. *primární metodu*, protože v matici soustavy vystupují přímo pouze primární proměnné (x a s) a žádné duální proměnné (Lagrangeovy multiplikátory λ_E a λ_I). Hlavní nevýhodou této formulace je to, že prvky matice S se mohou blížit k nule. Díky tomu není matice soustavy (1.24) omezená. Vynásobíme-li rovnici (1.21) maticí S, dostaneme nutné podmínky pro extrém ve tvaru

$$\begin{aligned} g(x) &= 0 && (1.25)\\ -\mu e + S\Lambda_I e &= 0 && (1.26)\\ c_E(x) &= 0 && (1.27)\\ c_I(x) + s &= 0 && (1.28)\end{aligned}$$

Pokud k vyřešení těchto rovnic opět použijeme Newtonovu metodu, dostaneme *primárně-duální* iterační metodu. V této metodě se opět v každém kroku řeší soustava lineárních rovnic, tentokrát ve tvaru

$$\begin{bmatrix} G & 0 & A_E & A_I \\ 0 & \Lambda_I & 0 & S \\ A_E^T & 0 & 0 & 0 \\ A_I^T & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle s \\ \triangle \lambda_E \\ \triangle \lambda_I \end{bmatrix} = \begin{bmatrix} -g \\ \mu e - \Lambda_I e \\ -c_E \\ -(c_I + s) \end{bmatrix} \tag{1.29}$$

Tato soustava je dobře podmíněná (viz. [11] nebo [22]) a matice této soustavy je omezená, není však symetrická. Vzhledem k tomu, že tato metoda a algoritmus v kapitole 2 je určen především pro řešení rozsáhlých úloh, je tato vlastnost zásadním omezením (způsobila by téměř dvakrát tak větší nároky na paměť a výpočet, ale hlavně by si vynutila použití méně efektivních algoritmů). Vynásobením druhé rovnice soustavy (1.29) maticí S^{-1} dostaneme symetrickou soustavu

$$\begin{bmatrix} G & 0 & A_E & A_I \\ 0 & S^{-1}\Lambda_I & 0 & I \\ A_E^T & 0 & 0 & 0 \\ A_I^T & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle s \\ \triangle \lambda_E \\ \triangle \lambda_I \end{bmatrix} = \begin{bmatrix} -g \\ \mu S^{-1} e - \lambda_I \\ -c_E \\ -(c_I + s) \end{bmatrix} \tag{1.30}$$

Tato soustava má podobné nevýhody jako soustava rovnic primární metody (1.24), prvky bloku $S^{-1}\Lambda_I$ mohou růst nade všechny meze. Podle [23] však tato vlastnost nemusí mít pro výpočet katastrofální následky. V kapitole 2 si ukážeme, jak upravit výpočet tak, aby na něj tato vlastnost neměla vliv.

1.5 Centrální cesta

Definice 6 (*Centrální cesta*) Centrální cestou budeme rozumět množinu bodů $v_\mu^* = \left[x_\mu^*, s_\mu^*, \lambda_\mu^*\right]^T$, které jsou řešením soustavy

$$\begin{aligned} g(x) &= 0 \\ -\mu e + S\Lambda_I e &= 0 \\ c_E(x) &= 0 \\ c_I(x) + s &= 0 \end{aligned}$$

Centrální cesta má pro metody vnitřního bodu zásadní význam. Centrální cesta definovaná výše se dá chápat jako křivka, která je parametrizovaná kladným bariérovým parametrem μ. Význam centrální cesty spočívá v tom, že pro $\mu \to 0$ vede tato křivka do bodu, který je řešením původního problému (1.2). Tato vlastnost dává přímo návod na algoritmus - sledovat centrální cestu až k řešení (jedná se o tzv. Path-following metody známé mimo jiné z lineárního programování).

Pokud chceme sledovat centrální cestu, bude důležité vědět, jak daleko od centrální cesty se iterace, kterou právě počítáme, nachází. S centralitou souvisí volba parametru μ. Parametr μ nám totiž udává, jak budou jednotlivé iterace tlačeny od hranice striktně přípustné oblasti. Čím větší bude hodnota μ, tím více se projeví bariérový člen upravené účelové funkce a tím více bude iterace tlačena od hranice striktně přípustné oblasti. Volba parametru μ a výpočet míry centrality jsou podrobněji diskutováy v kapitole 2.

1.6 Hodnotová funkce

Hodnotové funkce mají za úkol ohodnotit krok vypočtený iterační metodou a určit, zda se přiblížil k řešení. Podle toho bude krok náležitě zkrácen nebo úplně zamítnut. Hodnotové funkce pro nelineární programování s rovnostími omezeními byly v posledních letech intenzivně studovány. Pro problém bez vazeb by hodnotová funkce nebyla nic jiného než samotná účelová funkce $f(x)$. Význam hodnotové funkce pro problémy s vazbami spočívá v zajištení jak snížení hodnoty účelové funkce, tak i přiblížení se k přípustné oblasti popř. přiblížení se k centrální cestě. Toto se dá zajistit zkrácením délky kroku vypočteným ze soustavy (1.30) tak, aby došlo k požadovanému snížení hodnoty hodnotové funkce. Pro problém nelineárního programování s rovnostními omezeními ve tvaru

$$\min_{x \in \mathrm{R}^n} f(x) \text{ tak, aby } c(x) = 0$$

se nabízí použít hodnotovou funkci

$$\Phi(x, \sigma) = f(x) + \sigma \|c(x)\|_1, \tag{1.31}$$

kde $\|\cdot\|_1$ značí ℓ_1 normu. Tato hodnotová funkce je „přesná“ ve smyslu, že existuje σ_0 tak, že pro všechna $\sigma \geq \sigma_0$ bude bod, ve kterém má (1.31) minimum zaručeně přípustný a za obecných podmínek zároveň řešení původního problému. Norma ℓ_1 však nemusí být vždy vhodná, protože může působit problémy numerickým algoritmům (nastává Maratosův jev [22], [20]). Vezměme tedy hodnotovou funkci

$$\Phi(x, \sigma) = f(x) + \sigma \|c(x)\|_2^2, \tag{1.32}$$

kde $\|\cdot\|_2$ značí ℓ_2 normu. Tato hodnotová funkce je vlastně pokutová funkce pro nelineární programování s rovnostními vazbami, studovaná Fiacco a McCormickem [6]. Tato hodnotová funkce má sice svoje nevýhody (viz. [20]), ale je všude diferencovatelná a je vhodnější jí použít.

Budeme-li tedy uvažovat hodnotovou funkci (1.32) a použijeme-li ji na problém s nerovnostními vazbami (1.2), dostaneme hodnotovou funkci

$$\Phi(x, s, \sigma, \mu) = f(x) - \mu \sum_{i \in I} \ln s_i + \frac{\sigma}{2} \|c_E(x)\|_2^2 + \frac{\sigma}{2} \|c_I(x) + s\|_2^2. \tag{1.33}$$

To, že tato hodnotová funkce je řádně definována pro problém (1.2) s výběrem směrového vektoru podle soustavy (1.30), je ukázáno např. v [20].

Další možná hodnotová funkce je *KKT residuová funkce*, kterou používá například El-Bakry et al [4], a která je definována jako

$$\Phi(x, s, \lambda_I, \mu) = \frac{1}{2}(\|g\|^2 + \|c(x) + s\|^2 + \|S\Lambda_I e - \mu e\|^2 + \|c_E(x)\|^2) \tag{1.34}$$

Hodnotová funkce (1.34) je vlastně residuum při řešení soustavy (1.25)-(1.28), které je v řešení rovno nule. El-Bakry et al v [4] ukázali, že pro libovolný striktně přístupný bod $v = \left[x^T, s^T, \lambda_E^T, \lambda_I^T\right]^T$ je směr $\triangle v$ vypočtený Newtonovou metodou spádový směr pro residuovou funkci (1.34), pokud volíme parametr μ určitým způsobem. Takže

$$\nabla_v \Phi(v, \mu)^T \triangle v < 0$$

právě když

$$\mu < \frac{(\|g\|^2 + \|c_I(x) + s\|^2 + \|S\Lambda_I e - \mu e\|^2 + \|c_E(x)\|^2)}{s^T \lambda_I}.$$

U hodnotové funkce (1.34) lze ukázat, že za určitých podmínek při a vhodné volbě μ podle předchozí nerovnosti, konverguje algoritmus k bodu, kde $\Phi = 0$. Zevrubnější testování (viz [20]) však ukázalo, že algoritmus často konverguje k bodu lokálního maxima nebo sedlového bodu, kde je splněna nutná podmínka pro extrém, ale není bodem lokálního minima.

Další možností je hodnotová funkce založená na rozšířené Lagrangeově funkci zahrnující pokutový člen pro rovnostní omezení. Taková hodnotová funkce bude vypadat takto:

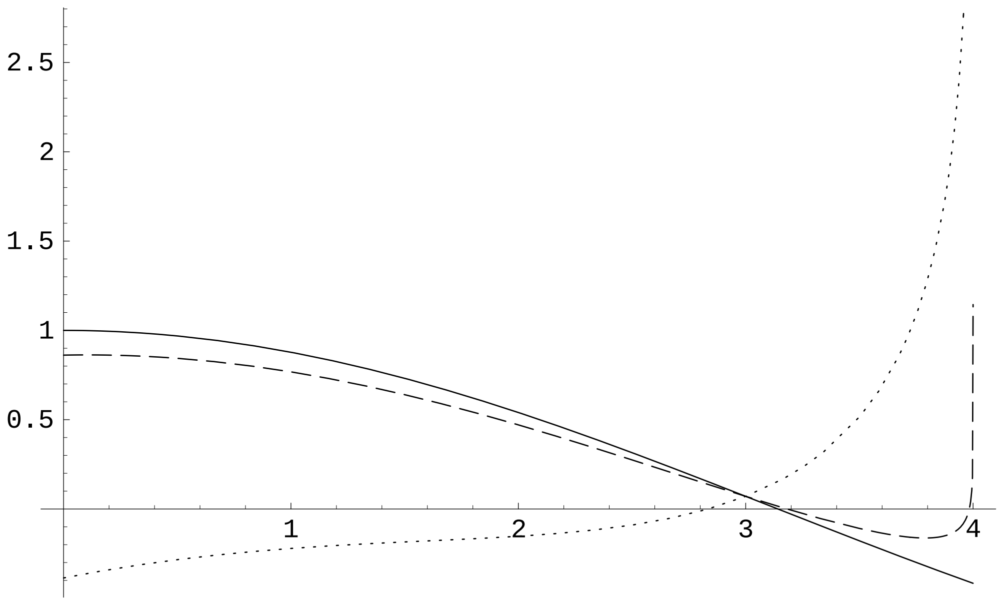
$$\begin{aligned} \Phi(x, s, \lambda_E, \lambda_I \sigma, \mu) &= f(x) - \mu \sum_{i \in I} \ln s_i + \lambda_E^T c_E + \lambda_I^T (c_I + s) + \\ &+ \frac{\sigma}{2} \|c_E(x)\|_2^2 + \frac{\sigma}{2} \|c_I(x) + s\|_2^2. \end{aligned} \tag{1.35}$$

Tento výčet typů hodnotových funkcí samozřejmě není zdaleka úplný. Hodnotové funkce pro metody vnitřního bodu jsou předmětem studia a jejich vhodná (resp. špatná) volba může mít zásadní vliv na výkonnost algoritmu.

1.7 Volba bariérového parametru μ

Zálkadní požadavek na parametr μ je, aby v průběhu výpočtu konvergoval k nule, což nám zajistí, že algoritmus bude konvergovat k řešení původního problému (1.2). Parametr μ má za úkol tlačit jednotlivé iterace od hranice striktně přípustné oblasti. Obrázek 1.1 ilustruje vliv hodnoty bariérového parametru μ na Lagrangeovu funkci (1.17).

Na volbu bariérového parametru má vliv i vzdálenost iterace od centrální cesty. Pokud se nacházíme blízko centrální cesty, je možné bariérový parametr snížit, protože centrální cesta vede k řešení vnitřkem striktně přípustné oblasti. Je-li však vzdálenost od centrální cesty větší, je třeba parametr zvětšit (nebo většinou nezmenšovat), aby se iterace nedostala příliš blízko hranici striktně přípustné oblasti, čímž by se algoritmus mohl neúnosně zpomalit, či dokonce zastavit - bylo by nutné volit nulovou délku kroku.



Obrázek 1.1: Obrázek ukazuje vliv bariérového parametru na funkci s jednoduchým omezením $x \le 4$. Plnou čarou je zakreslena původní funkce $\cos(\frac{1}{2}x)$. Tečkovanou čarou je zakreslena původní funkce s bariérovým členem s parametrem $\mu = 1$ a čárkovanou čarou s parametrem $\mu = 0.1$

Ukážeme si nyní několik způsobů jak volit bariérový parametr. Většina teoretických analýz metod vnitřního bodu volí μ jako

$$\mu = \sigma \frac{s^T \lambda_I}{m_I}, \tag{1.36}$$

kde $0 \le \sigma < 1$ a pravá strana (1.36) je vlastně odhadem centrality iterace. Pokud se iterace budou blížit k řešení, bude se součin $s^T \lambda_I$ blížit k nule, a tedy μ bude splňovat základní požadavek konvergence k nule. El Bakry et al. v [4] ukázali, že pro tuto volbu μ jejich algoritmus konverguje k minimální (v jejich případě nulové) hodnotě hodnotové funkce (1.34).

R. J. Vanderbei a D. F. Shanno používají v [20] k měření vdálenosti od centrální cesty výraz

$$\xi = \frac{\min S \Lambda_I e}{s^T \lambda_I / m_I}.$$

Je zřejmé, že $0 < \xi \le 1$ a $\xi = 1$ právě, když jsou hodnoty $s_i \lambda_i$ konstantní pro všechny $i = 1 \dots m_I$. Následně používají tento způsob určení parametru μ

$$\mu = \gamma \min\left((1-r)\frac{1-\xi}{\xi}, 2\right)^3 \frac{s^T \lambda_I}{m}, \tag{1.37}$$

kde $0 < r < 1$ a γ jsou parametry s výchozími hodnotami $r = 0.95$ a $\gamma = 0.1$.

Kromě již zmiňované možnosti výpočtu μ podle (1.36) navrhli M. Argaez et al. v [1] další dvě možnosti výpočtu μ v závislosti na různých metodách výpočtu vdálenosti od centrální cesty. V následujících chvílích budeme předpokládat, že se výpočet právě nachází v nějaké konkrétní iteraci. Všechny hodnoty x, s, λ_E a λ_I se tedy budou vztahovat k této iteraci. Označíme μ stávající hodnotu bariérového parametru a μ^+ jeho nově spočtenou hodnotu.

1. Zvolíme počáteční hodnotu $\mu = \sigma \frac{s^T \lambda_I}{m_I}$. Spočteme $W = (S\Lambda_I)^{\frac{1}{2}}$.

   pokud $(\|c_E(x)\|^2 + \|c_I(x) + s\|^2 + \|W^{-1}(S\Lambda_I e - \mu e)\|^2) \leq (\frac{1}{2}\mu)$ potom
   $$\mu^+ = 10^{-2}\mu$$
   jinak
   $$\mu^+ = \mu$$

2. Označme
   $$F(x, s, \lambda_E, \lambda_I, \mu) = (\|g\|^2 + \|c_E(x)\|^2 + \|c_I(x) + s\|^2 + \|S\Lambda_I e - \mu e\|^2)^{\frac{1}{2}}$$
   Zvolíme počáteční hodnotu
   $$\mu = \frac{1}{n}\min(\frac{1}{2}, F(x, s, \lambda_E, \lambda_I, \mu_0))F(x, s, \lambda_E, \lambda_I, \mu_0)$$
   pokud $(\|c_E(x)\|^2 + \|c_I(x) + s\|^2 + \|S\Lambda_I e - \mu e\|^2) \leq (0.8\mu)$ potom
   $$\mu^+ = \frac{1}{n}\min(0.5, F(x, s, \lambda_E, \lambda_I, \mu))F(x, s, \lambda_E, \lambda_I, \mu)$$
   jinak
   $$\mu^+ = \mu$$

Poznamenejme, že funkce F není nic jiného než residuová hodnotová funkce Φ z (1.34), která má v řešení hodnotu 0, což zaručuje, že parametr μ půjde k nule. Hodnota μ_0 je nějaká zvolená počáteční hodnota (např. $\mu_0 = 0.5$). Výpočet dle bodu 1 lze provést i pro $W = I$, detaily viz. [1].

Kromě již zmíněných způsobů určování bariérového parametru μ je možný i poněkud obecnější přístup. Bariérovou úlohu (1.16) můžeme také formulovat s více bariérovými parametry, pro každé nerovnostní omezení zvlášť:

$$\min_{x \in \mathrm{R}^n} f(x) - \sum_{i=1}^{m_I} \mu_i \ln s_i \quad \text{tak, aby} \quad c_E(x) = 0,\ c_I(x) + s = 0.$$

Primárně-duální metoda využívající právě zmíněné formulace je popsána např. v [19]. Hlavní rozdíly mezi metodou v této práci a „standardními“ metodami jsou použití více bariérových parametrů a volba různé délky kroku pro primární a pro duální (Lagrangeovy multiplikátory) proměnné.

Výběr vhodné metody výpočtu bariérového parametru je otázkou numerického experimentování a dalšího studia, avšak je zřejmé, že výpočet bariérového parametru může ovlivnit počet iterací nutných k dosažení požadované přesnosti, ale i schopnost algoritmu požadované přesnosti vůbec dosáhnout.

Kapitola 2

Algoritmus

2.1 Určení směrového vektoru

Pro výpočet směrového vektoru, jak již bylo řečeno dříve, použijeme Newtonovu metodu. Budeme tak v každém kroku řešit soustavu (1.30). Z této soustavy můžeme přímo eliminovat pomocné proměnné $\triangle s$. Dostaneme

$$\triangle s = -\Lambda_I^{-1} S \triangle \lambda_I - s + \mu \Lambda_I^{-1} e. \tag{2.1}$$

Po eliminaci $\triangle s$ bude soustava (1.30) vypadat

$$\begin{bmatrix} G & A_E & A_I \\ A_E^T & 0 & 0 \\ A_I^T & 0 & -\Lambda_I^{-1} S \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle \lambda_E \\ \triangle \lambda_I \end{bmatrix} = \begin{bmatrix} -g \\ -c_E \\ -(c_I + \mu \Lambda_I^{-1} e) \end{bmatrix} \tag{2.2}$$

Z nutných podmínek pro extrém (1.26) plyne, že $\lambda_i s_i \sim \mu$, $i \in I$. Pokud $\mu \to 0$ (což je předpoklad, abychom dosáhli řešení původní úlohy), pak je buď $\lambda_i \to 0$ nebo $s_i \to 0$. Je-li splněna pomínka ryzí komplementarity (1.13), nemůže platit současně $\lambda_i \to 0$ a $s_i \to 0$. Rozdělíme teď omezení na aktivní a neaktivní. Dle definice 5 jsou aktivní omezení ty, pro které $s_i \to 0$ a neaktivní v opačném případě. Přeindexujeme omezení a označíme $s = [\hat{s}^T, \check{s}^T]^T$ a $\lambda_I = [\hat{\lambda}_I^T, \check{\lambda}_I^T]^T$ tak, že $\hat{s} \to 0$ a $\check{\lambda}_I \to 0$. Obdobným způsobem označíme matice $\hat{A}_I$ a $\check{A}_I$. Ze soustavy (2.2) můžeme eliminovat neaktivní omezení. Dostaneme

$$\triangle \check{\lambda}_I = \check{S}^{-1} \check{\Lambda}_I \check{A}_I^T \triangle x + \check{S}^{-1} \check{\Lambda}_I (\check{c}_I + \mu \check{\Lambda}_I^{-1} e) \tag{2.3}$$

a ze soustavy (2.2) dostaneme soustavu

$$\begin{bmatrix} \hat{G} & A_E & \hat{A}_I \\ A_E^T & 0 & 0 \\ \hat{A}_I^T & 0 & -\hat{\Lambda}_I^{-1} \hat{S} \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle \lambda_E \\ \triangle \hat{\lambda}_I \end{bmatrix} = \begin{bmatrix} -\hat{g} \\ -c_E \\ -(\hat{c}_I + \mu \hat{\Lambda}_I^{-1} e) \end{bmatrix}, \tag{2.4}$$

kde

$$\begin{aligned}\hat{G} &= G + \breve{A}_I \breve{S}^{-1} \breve{\Lambda}_I \breve{A}_I^T \\ \hat{g} &= g + \breve{A}_I \breve{S}^{-1} \breve{\Lambda}_I \breve{c}_I + \mu \breve{A}_I \breve{S}^{-1} e\end{aligned}$$

Řešení původní soustavy (1.30) se tedy zredukovalo na řešení menší soustavy (2.4). Ostatní proměnné dopočteme podle (2.1) a (2.3). Počítání se soustavou (2.4) má tu výhodu, že matice této soustavy je omezená (prvky $-\hat{\Lambda}_I^{-1}\hat{S}$ konvergují díky striktní komlementaritě k nule).

2.2 Určení délky kroku

Je známo, že u nelineárních problémů může špatný počáteční odhad pro Newtonovu metodu způsobit, že tato metoda nebude konvergovat. U problémů nelineárního programování může špatný počáteční odhad způsobit, že Newtonova metoda bude generovat kroky, které se dostanou mimo striktně přípustnou oblast $\mathcal{F}^0$. Špatný počáteční odhad nemusí být jedinou příčinou příliš dlouhých Newtonových kroků, ale tento jev většinou nastává v prvních iteracích metody, ale i po snížení hodnoty bariérového parametru μ. Aby se dala zajistit správná konvergence Newtonovy metody, je třeba zkrátit délku kroku. K tomu, abychom věděli o kolik zkrátit délku kroku, slouží hodnotové funkce diskutované v (1.6 Hodnotová funkce).

V našem algoritmu budeme používat hodnotovou funkci (1.35), tedy funkci

$$\begin{aligned}\Phi(x, s, \lambda_E, \lambda_I, \sigma, \mu) &= f(x) - \mu \sum_{i \in I} \ln s_i + \lambda_E^T c_E + \lambda_I^T (c_I + s) + \\ &+ \frac{\sigma}{2} \|c_E(x)\|_2^2 + \frac{\sigma}{2} \|c_I(x) + s\|_2^2.\end{aligned}$$

Protože budeme soustavu lineárních rovnic (2.4) řešit iterační metodou sdružených gradientů, budeme předpokládat, že výsledný vektor $\triangle v = \left[\triangle x^T, \triangle\lambda_E^T, \triangle\hat{\lambda}_I^T\right]^T$, získaný iterační metodou řeší soustavu (2.4) s omezenou přesností tj.

$$\begin{bmatrix} \hat{G} & A_E & \hat{A}_I \\ A_E^T & 0 & 0 \\ \hat{A}_I^T & 0 & -\hat{\Lambda}_I^{-1}\hat{S} \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle\lambda_E \\ \triangle\hat{\lambda}_I \end{bmatrix} + \begin{bmatrix} \hat{g} \\ c_E \\ \hat{c}_I + \mu\hat{\Lambda}_I^{-1} e \end{bmatrix} = \begin{bmatrix} h \\ r_E \\ \hat{r}_I \end{bmatrix}. \tag{2.5}$$

To odpovídá řešení původní soustavy (1.29) s přesností

$$\begin{bmatrix} G & 0 & A_E & A_I \\ 0 & \Lambda_I & 0 & S \\ A_E^T & 0 & 0 & 0 \\ A_I^T & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle s \\ \triangle\lambda_E \\ \triangle\lambda_I \end{bmatrix} + \begin{bmatrix} g \\ S\lambda_I - \mu e \\ c_E \\ c_I + s \end{bmatrix} = \begin{bmatrix} h \\ 0 \\ r_E \\ r_I \end{bmatrix}, \tag{2.6}$$

kde $r_I = \left[\hat{r}_I^T, 0\right]^T$.

Nechť je tedy $\triangle v = \left[\triangle x^T, \triangle s^T, \triangle\lambda_E^T, \triangle\lambda_I^T\right]^T$ příslušné řešení soustavy (2.6). Pro výběr délky kroku α zavedeme funkci $P(\alpha)$, která je vlastně upravená hodnotová funkce (1.35)

$$\begin{aligned} P(\alpha) &= \Phi(v + \alpha\triangle v, \sigma, \mu) = \\ &= f(x + \alpha\triangle x) - \mu e^T \ln(S + \alpha\triangle S)e + \\ &+ (\lambda_E + \triangle\lambda_E)^T c_E(x + \alpha\triangle x) + \\ &+ (\lambda_I + \triangle\lambda_I)^T (c_I(x + \alpha\triangle x) + s + \alpha\triangle s) + \\ &+ \frac{\sigma}{2}\|c_E(x + \alpha\triangle x)\|^2 + \\ &+ \frac{\sigma}{2}\|c_I(x + \alpha\triangle x) + s + \alpha\triangle s\|^2. \end{aligned} \tag{2.7}$$

Význam funkce $P(\alpha)$ není třeba příliš komentovat, poznamenejme snad jen, že P dává hodnotu funkce Φ v bodě $v + \alpha\triangle v$. Vyjádřeme si nyní derivaci $P(\alpha)$ podle α v bodě $\alpha = 0$. Dostaneme

$$\begin{aligned} P'(0) &= (\triangle x)^T (\nabla f(x) + A_E(\lambda_E + \triangle\lambda_E) + A_I(\lambda_I + \triangle\lambda_I) - \\ &- \mu e^T S^{-1}\triangle s + \sigma c_E^T A_E^T \triangle x + \sigma(c_I + s)^T A_I^T \triangle x + \\ &+ \sigma(c_I + s)^T \triangle s + (\triangle s)^T(\lambda_I + \triangle\lambda_I)), \end{aligned} \tag{2.8}$$

což není nic jiného než směrová derivace hodnotové funkce Φ ve směru $\triangle v$ v bodě v. Pokud si rozepíšeme jednotlivé rovnice soustavy (2.6) a vyjádříme jimi jednotlivé členy výrazu (2.8), dostaneme

$$\begin{aligned} (\triangle x)^T (\nabla f(x) + A_E(\lambda_E + \triangle\lambda_E) + & \\ + A_I(\lambda_I + \triangle\lambda_I) &= (\triangle x)^T h - (\triangle x)^T G(\triangle x) && (2.9) \\ (\triangle s)^T (\lambda_I + \triangle\lambda_I) - \mu(\triangle s)^T S^{-1} e &= -(\triangle s)^T S^{-1}\Lambda_I(\triangle s) && (2.10) \\ c_E^T A_E^T \triangle x &= c_E^T r_E - c_E^T c_E && (2.11) \\ (c_I + s)^T A_I^T \triangle x + (c_I + s)^T \triangle s &= (c_I + s)^T r_I - (c_I + s)^T (c_I + s) && (2.12) \end{aligned}$$

Když rovnosti (2.9) - (2.12) dosadíme do (2.8) dostaneme, že

$$\begin{aligned} P'(0) &= (\triangle x)^T h + \sigma c_E^T r_E + \sigma(c_I + s)^T r_I - \\ &- (\triangle x)^T G\triangle x - \sigma c_E^T c_E - \sigma(c_I + s)^T (c_I + s) - \\ &- (\triangle s)^T S^{-1}\Lambda_I \triangle s \end{aligned} \tag{2.13}$$

Nyní si dokážeme, že pro vhodnou volbu parametru σ bude směrový vektor spočtený ze soustavy (2.6) spádovým směrem pro hodnotovou funkci (1.35).

Lemma 2 *Nechť* $\underline{G} > 0$ *a*

$$\sigma \geq \frac{2\underline{G}\|\triangle x\|^2 - (\triangle x)^T G \triangle x - (\triangle s)^T S^{-1}\Lambda_I \triangle s}{\|c_E\|^2 + \|c_I + s\|^2} \tag{2.14}$$

a

$$\begin{aligned} (\triangle x)^T h + \sigma c_E^T r_E + \sigma (c_I + s)^T r_I \quad \leq \quad & \frac{1}{2}\Big((\triangle x)^T G \triangle x + \sigma c_E^T c_E + \\ & + \sigma (c_I + s)^T (c_I + s) + (\triangle s)^T S^{-1} \Lambda_I \triangle s\Big) \end{aligned} \tag{2.15}$$

Pak

$$P'(0) \leq -\underline{G}\|\triangle x\|^2$$

Důkaz
Podle (2.14) platí

$$\begin{aligned} 2\underline{G}\|\triangle x\|^2 \quad \leq \quad & (\triangle x)^T G \triangle x + \sigma c_E^T c_E + \sigma (c_I + s)^T (c_I + s) + \\ & + (\triangle s)^T S^{-1} \Lambda_I \triangle s \end{aligned} \tag{2.16}$$

což nám spolu s (2.15) a výrazem (2.13) pro

$$\begin{aligned} z &= (\triangle x)^T G \triangle x + \sigma c_E^T c_E + \sigma (c_I + s)^T (c_I + s) \\ w &= (\triangle s)^T S^{-1} \Lambda_I \triangle s \end{aligned}$$

dává

$$P'(0) \leq \frac{1}{2}(z + w) - (z + w) \leq -\underline{G}\|\triangle x\|^2 \tag{2.17}$$

Existuje tedy hodnota pokutového parametru σ tak, že pokud je soustava lineárních rovnic (2.6) vyřešena dostatečně přesně, je směrová derivace $P'(0)$ dostatečně záporná a je možné snížit hodnotu funkce $P(\alpha)$, a tedy i hodnotové funkce Φ.

Při volbě délky kroku je zároveň nezbytné zajistit, aby proměnné λ_I a s, které mají zůstat kladné, kladné zůstaly. Budeme tedy v každém kroku počítat hodnotu $\alpha_{\max}$ standardním poměrovým testem. Tedy

$$\alpha_{\max} = \min\left(1, \gamma \min_{i \in I, \triangle s_i < 0} \left\{-\frac{s_i}{\triangle s_i}\right\}\right). \tag{2.18}$$

Parametr $\gamma < 1$ má za úkol zajistit, aby některé z pomocných proměnných nebyly rovny nule. Pokud je hodnota $\triangle s_i$ kladná, není nutné maximální délku kroku zkracovat (pomocná proměnná s se zvětšuje). Abychom zabránili přílišnému zkracování délky kroku primárních proměnných x a s, budeme volit různou délku kroku pro primární proměnné (budeme značit α) a pro duální proměnné λ_I (budeme značit $\tilde{\alpha}$). Spočteme tedy ještě maximální délku duálního kroku:

$$\tilde{\alpha}_{\max} = \min\left(1, \gamma \min_{i \in I, \triangle\lambda_i < 0} \left\{-\frac{\lambda_i}{\triangle\lambda_i}\right\}\right). \tag{2.19}$$

Obvyklá hodnota γ se bude pohybovat kolem 0.99. V intervalu $[0, \alpha_{\max}]$ budeme potom hledat hodnotu α, která umožní největší snížení hodnoty funkce Φ. Délku duálního kroku určíme jako

$$\tilde{\alpha} = \min(\alpha, \tilde{\alpha}_{\max}). \tag{2.20}$$

Pokud označíme x,s,λ_I a λ_E současnou hodnotu proměnných a x^+,s^+,λ_I^+ a λ_E^+ jejich novou hodnotu, bude iterace generovat krok

$$\begin{aligned} x_i^+ &= x_i + \alpha\triangle x_i,\ i = 1, \ldots, n \\ s_i^+ &= s_i + \alpha\triangle s_i,\ i \in I \\ \lambda_i^+ &= \lambda_i + \tilde{\alpha}\triangle\lambda_i,\ i \in E \cup I \end{aligned}$$

Konkrétní pravidlo pro výběr délky primárního kroku je otázkou volby. Standardní volbou je výběr délky kroku podle Armijova pravidla (viz. [14]), tj. $\alpha > 0$ je zvoleno jako první člen posloupnosti $\underline{\beta}^j$, $j = 0, 1, \ldots$, $0 < \underline{\beta} < 1$, pro který platí

$$P(\alpha) - P(0) \le \underline{\epsilon}\alpha P'(0),$$

kde $0 < \underline{\epsilon} < 1$ je zvolená konstanta. Pokud platí, že $\alpha_{max} \neq 1$, je nutné volit α jako $\underline{\beta}^j\alpha_{max}$. Pro konstantu $\underline{\beta}$ platí, že čím bude větší, tím delší vhodné kroky bude možno použít, ale počet iterací (Armijova algoritmu) bude o to větší.

2.3 Řešení soustavy lineárních rovnic

Algoritmus popsaný v předchozích kapitolách vyžaduje, aby se v každé iteraci řešila soustava lineárních rovnic (2.4). Tuto soustavu budeme řešit iteračně metodou sdružených gradientů. Tento postup má několik důležitých výhod. Jednou z nich je, že tato metoda nemění v průběhu výpočtu matici soustavy a tudíž odpadají problémy s dodatečným zaplňováním matice soustavy, což by narušilo její řídkou strukturu. U soustav s řídkou maticí je metoda sdružených gradientů také značně efektivní - v porovnání s jinými metodami vyžaduje velmi málo operací. Další výhodou iterační metody je i to, že pokud se v současném kroku metody nacházíme daleko od řešení $x^* \in \mathrm{R}^n$, není potřeba soustavu (2.4) řešit s velkou přesností.

Uvažujme nyní soustavu lineárních rovnic

$$\begin{bmatrix} B & A \\ A^T & -C \end{bmatrix} \begin{bmatrix} d \\ v \end{bmatrix} = - \begin{bmatrix} b \\ c \end{bmatrix} \tag{2.21}$$

s indefinitní maticí

$$K = \begin{bmatrix} B & A \\ A^T & -C \end{bmatrix} \tag{2.22}$$

Připomeňme si nyní stručně algoritmus metody sdružených gradientů. V následujícím algoritmu budeme řešit soustavu $Kx = b$.

Algoritmus CGM (*Metoda sdružených gradientů*)

Krok 1	Dáno x_1; $\beta_0 = 0$, $k \leftarrow 1$
Krok 2	Residuum soustavy: $r_k = Kx_k - b$; Je-li $r_k = 0$ potom $x = x_k$, konec
Krok 3	Spočteme $\beta_{k-1} = -\frac{(Kr_k, s_{k-1})}{(Ks_{k-1}, s_{k-1})} = -\frac{(r_k, Ks_{k-1})}{(Ks_{k-1}, s_{k-1})}$
Krok 4	Spočteme $s_k = r_k + \beta_{k-1} s_{k-1}$
Krok 5	Spočteme $\alpha_k = -\frac{(r_k, s_k)}{(Ks_k, s_k)}$
Krok 6	Krok $x_{k+1} = x_k + \alpha_k s_k$, zpět na krok 2

Pro metodu sdružených gradientů je důležité najít vhodný předpodmiňovač. Hlavním úkolem předpodmiňovče Q je změnit spektrum matice K tak, aby vlastní čísla předpodmíněné matice byly blíže u sebe, což obvykle vede ke zrychlení konvergence metody sdružených gradientů. Ideálně, matice Q by měla být co nejblíze matici K, ale násobení maticí C^{-1} by nemělo být výpočetně náročné. Platí

$$K^{-1} = \begin{bmatrix} B^{-1} - B^{-1}A(A^TB^{-1}A + C)^{-1}A^TB^{-1} & B^{-1}A(A^TB^{-1}A + C)^{-1} \\ (A^TB^{-1}A + C)^{-1}A^TB^{-1} & -(A^TB^{-1}A + C)^{-1} \end{bmatrix} \tag{2.23}$$

Předpodmiňovač budeme hledat ve tvaru

$$Q = \begin{bmatrix} D & A \\ A^T & -C \end{bmatrix}, \tag{2.24}$$

kde D je diagonální matice, která dobře aproximuje matici B (například její diagonála). Platí

$$\begin{aligned} KQ^{-1} &= \begin{bmatrix} B & A \\ A^T & -C \end{bmatrix} \begin{bmatrix} D^{-1} - D^{-1}ARA^TD^{-1} & D^{-1}AR \\ RA^TD^{-1} & -R \end{bmatrix} = \\ &= \begin{bmatrix} I + MP & MA(A^TD^{-1}A + C)^{-1} \\ 0 & I \end{bmatrix}, \end{aligned}$$

kde $M = BD^{-1} - I$, $P = I - A(A^TD^{-1}A + C)^{-1}A^TD^{-1}$ a $R = (A^TD^{-1}A + C)^{-1}$.

My potřebujeme řešit soustavu (2.4). Označení v souladu s (2.21) bude vypadat takto:

$$\begin{aligned} B &= \hat{G} \\ A &= [\, A_E \quad \hat{A}_I \,] \\ C &= \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_I^{-1}\hat{S} \end{bmatrix} \end{aligned}$$

Pokud budou splněny podmínky striktní komplementarity, bude matice C v blízkosti řešení původní úlohy blízká nulové matici. Indefinitní předpodmiňovač pro soustavu (2.21) je odvozen od předpodmiňovače pro soustavu s maticí

$$K = \begin{bmatrix} B & A \\ A^T & 0 \end{bmatrix}, \tag{2.25}$$

který je studován v L. Lukšan a J. Vlček [12], a který vykazuje velmi dobré numerické vlastnosti.

2.4 Pseudokód algoritmu

Nejprve si uvedeme obecný algoritmus bariérové metody, který vychází z věty 2 v kapitole 3. Tento algoritmus v každém kroku hledá pro pevný bariérový parametr μ řešení soustavy (1.25) - (1.28). V okamžiku, kdy toto řešení najde, sníží se bariérový parametr μ a proces se opakuje až do tak dlouho, až jsou splněny požadavky na přesnost řešení.

Algoritmus 2 Nechť x_0 je striktně přípustný výchozí bod

Krok 1 *Inicializace.* $k \leftarrow 0$, μ_0 nechť je kladné
Vnější cyklus:
Krok 2 *Test konvergence.* Pokud x_k vyhovuje podmínkám pro lokální vázaný extrém původního problému, potom konec, x_k je řešení.
Krok 3 *Vnitřní cylkus.* Newtonovou metodou vypočítat $x(\mu_k)$ jakožto řešení KKT rovnic (1.25) - (1.28) pro pevné μ.
Krok 4 *Krok.* $x_{k+1} \leftarrow x(\mu_k)$; určit $\mu_{k+1} < \mu_k$; zpět na krok 2.

Věta 2 vyžaduje, aby bylo v kroku 3 řešení nalezeno přesně, což by způsobovalo značnou neefektivitu při výpočtech. Proto se v praxi vektor $x(\mu_k)$ vypočítává přibližně, jen v několika iteracích minimalizační metody.

Poznámka 1 V kroku 3 algoritmu 1 lze samozřejmě použít i jiné vhodné metody, než je metoda Newtonova, jelikož se jedná o hledání nevázaného lokálního extrému funkce.

Poznámka 2 Tento algoritmus v principu odpovídá původním myšlenkám algoritmů bariérových metod z 60. let.

Dílčí kroky algoritmu z předchozích kapitol můžeme shrnout do následujícího pseudokódu.

Algoritmus 3 Nechť x_0 je přípustný výchozí bod

Krok 1 *Inicializace.* $k \leftarrow 0$, spočítat s_0 podle (2.27) a λ_0 podle (2.28). Zvolit $\mu > 0$ a $\sigma > 0$.

Krok 2 *Test konvergence.* Pokud x_k vyhovuje podmínkám pro lokální vázaný extrém původního problému, potom konec, x_k je řešení.

Krok 3 *Aktivní omezení* Rozdělit omezení na aktivní a neaktivní.

Krok 4 *Směrový vektor.* Metodou sdružených gradientů vypočítat vektor $\triangle v_k = \left[\triangle x_k^T, \triangle s_k^T, \triangle\lambda_k^T\right]^T$, $\lambda^T = \left[\lambda_E^T, \lambda_I^T\right]$, jako nepřesné řešení soustavy lineárních rovnic (2.6). Dopočítat Lagrangeovy multiplikátory neaktivních omezení a pomocných proměnných s_k podle (2.3) a (2.1).

Krok 5 *Parametry.* Spočítat penaltový parametr σ podle lemma 2 tak, aby byla zaručena spádovost směru $\triangle v_k$. Spočítat bariérový parametr μ podle zvolené metody.

Krok 6 *Hodnoty.* Spočítat hodnotu hodnotové funkce (1.35), směrové derivace $P'(0)$ podle (2.13), $\alpha_{\max}$ podle (2.18) a $\tilde{\alpha}_{\max}$ podle (2.19).

Krok 7 *Výběr délky kroku* Podle zvolené metody vypočítat délku primárního kroku $\alpha \in (0, \alpha_{max}]$ a délku duálního kroku $\tilde{\alpha}$ určit podle (2.20).

Krok 8 *Krok.* Provést krok $x_{k+1} = x_k + \alpha x_k$, $s_{k+1} = s_k + \alpha s_k$ a $\lambda_{k+1} = \lambda_k + \alpha\lambda_k$. $k \leftarrow k + 1$. Zpět na krok 2.

2.5 Implementační detaily

2.5.1 Volba počátečních hodnot

Logaritmická bariérová metoda vyžaduje, aby byly všechny iterace (včetně startovacího bodu) striktně přípustné, protože rozšířená Lagrangeova funkce není mimo $\mathcal{F}^0$ řádně definována (pomocné proměnné s by mohly nabývat nekladných hodnot a rozšířená Lagrangeova funkce počítá s jejich přirozeným logaritmem). Startovací bod x je většinou součástí zadání testovacích i praktických problémů, avšak algoritmus metody vnitřního bodu používá i další proměnné λ a s, jejichž počáteční hodnoty se musí určit výpočtem. Pro pomocné proměnné s můžeme, pokud známe počáteční hodnotu $x^{(0)}$, spočítat hodnotu $c_i(x^{(0)})$ a za počáteční hodnotu s brát

$$s_i^{(0)} = -c_i(x^{(0)}),\ i \in I. \tag{2.26}$$

Volba pomocných proměnných podle (2.26) může způsobit v zásadě dva problémy. První je, že pokud není $x^{(0)}$ striktně přípustný bod, volba (2.26) dává pro některé s_i nekladné hodnoty, což není, jak již bylo řečeno, u metod vnitřního bodu povoleno. Druhý problém je, že i kdyby byl bod $x^{(0)}$ striktně přípustný, může ležet na hranici nebo tak blízko hranice striktně přípustné oblasti $\mathcal{F}^0$, že by některé s_i byly příliš blízko nule, a to by mohlo zabránit algoritmu rozumně postupovat k řešení (bylo by nutné značně zkracovat délku kroku). Je tedy nutné, podobně jako u metod vnitřního bodu pro lineární programování, zvolit $\theta > 0$ tak, aby všechny počáteční hodnoty s_i byly alespoň tak kladné jako θ. Je tedy lepší volit pomocné proměnné následovně:

$$s_i^{(0)} = \max(-c_i(x^{(0)}), \theta),\ i \in I, \tag{2.27}$$

kde konstatna θ je volena jako 0.1. Počáteční hodnoty Lagrangeových multiplikátorů λ budeme v našem algoritmu volit takto

$$\lambda_i^{(0)} = \delta,\ i \in I. \tag{2.28}$$

Parametr δ má stejně jako θ výchozí hodnotu 0.1.

2.5.2 Určení aktivních omezení

Podle Definice 5 jsou aktivními omezeními $c_i(x)$, $i \in I$ ty, pro které platí, že s_i konverguje v průběhu výpočtu k nule. Bohužel, množina aktivních omezení není předem známa. Na principu určování aktivních omezení je dokonce založena celá třída optimalizačních metod tzv. Active set methods. Je tedy nutné určovat aktivní omezení v průběhu výpočtu podle nějakého pravidla.

Jednou z možností je určovat aktivní omezení podle toho, zda pomocná proměnná s_i klesne pod určitou předem zvolenou mez. Omezení $c_i(x)$, $i \in I$ tedy označíme za aktivní, pokud bude platit

$$s_i \leq \epsilon,\ \epsilon > 0.$$

Nevýhodou tohoto postupu je, že nebere zřetel na různě škálované úlohy. Pokud bychom např. použili stejnou hodnotu ϵ na úlohu, kde se hodnoty omezení $c_i(x)$ pohybují v milionech a na úlohy, kde se hodnoty omezení pohybují v miliontinách, byly by v prvním případě téměř všechny omezení určeny jako neaktivní (i když by některé z nich třeba ve skutečnosti byly) a v druhém případě by naopak téměř všechny vyšly jako aktivní.

Je tedy vhodnější použít některé jiné metody. Můžeme například využít striktní komplementarity (tu máme jako předpoklad). Tato vlastnost nám říká, že nemůže zároveň nastat situace, kdy v blízkosti řešení x^* bude pro nějakou vazbu $c_i(x)$, $i \in I$ platit $s_i \sim 0$ a zároveň $\lambda_i \sim 0$. Toho můžeme s úspěchem využít při určování aktivních omezení. Jako výchozí hodnoty nastavíme Lagrangeovy multiplikátory λ_I a pomocné proměnné s tak, aby platilo

$$\lambda_i \leq s_i,\ i \in I. \tag{2.29}$$

To odpovídá nastavení výchozích hodnot podle (2.27) a (2.28) z (2.5.1 Volba počátečních hodnot). Jako výchozí budeme považovat všechny omezení za neaktivní. V okamžiku, kdy v průběhu výpočtu narazíme na omezení, pro které jsou pomocné proměnné s „o hodně menší“ než Lagrangeovy multiplikátory λ_I, označíme takové omezení za aktivní. Pojem „o hodně menší“ můžeme zhmotnit například pomocí této nerovnosti:

$$s_i \leq \gamma\lambda_i,\ i \in I, \tag{2.30}$$

kde $0 < \gamma < 1$ je konstanta, jejíž hodnotu zvolíme jako $\gamma = 0.1$. Poznamenejme, že pravidlo (2.30) není zbytečné, protože, zejména v prvních iteracích, se může stát, že pomocné proměnné u některých omezení dočasně mírně klesnou pod hodnotu Lagrangeových multiplikátorů. Algoritmus by měl také mít možnost se opravit. Tím je míněno, aby omezení jednou určené jako aktivní nebyly navždy odsouzeny k tomu být aktivní. To se dá zajistit tak, že se určování aktivních omezení provede vždy pro všechna omezení bez ohledu na to, zda byly před tím určeny jako aktivní, či nikoliv.

2.5.3 Kritéria pro ukončení iteračního procesu

Iterační proces metody vnitřního bodu budeme chtít ukončit v okamžiku, kdy jsme dosáhli požadované přesnosti. Protože neznáme přesné řešení x^*, nemůžeme použít kritérium

$$\|x_k - x^*\| < TOL_x.$$

Pro určení vzdálenosti iterace od řešení tedy budeme muset použít jiný způsob. Budeme vycházet z podmínek optimality (viz. odstavec 1.3). Použijeme zejména vazební podmínky (1.7) a (1.8) a podmínku nulovosti gradientu rozšířené Lagrangeovy funkce (1.9). U vazebních podmínek, které mají tvar

$$\begin{aligned} c_E(x^*) &= 0 \\ c_I(x^*) + s &= 0, \end{aligned}$$

budeme vyžadovat, aby maximální porušení těchto podmínek nepřekročilo určitou mez, tedy aby platilo

$$\begin{aligned} \|c_E(x_k)\|_\infty &\leq TOL_c \\ \|c_I(x_k) + s\|_\infty &\leq TOL_c, \end{aligned}$$

kde $\|\cdot\|_\infty$ značí maximovou normu a TOL_c je zvolená konstanta.

Požadavek nulovosti gradientu Lagrangeovy funkce, který zapisujeme jako

$$g(x^*) = \nabla f(x^*) + A_E(x^*)^T \lambda_E^* + A_I(x^*)^T \lambda_I^*, \tag{2.31}$$

budeme podobně jako u vazebních podmínek kontrolovat nějakou kladnou konstantou TOL_g, tedy

$$\|g(x_k)\| \leq TOL_g. \tag{2.32}$$

Obě konstanty TOL_c a TOL_g mají jako výchozí stanovenou hodnotu 10^{-6}. Iterační proces se zastavuje, pokud jsou splněny obě podmínky, tj. vazební podmínky i podmínka nulovosti gradientu Lagrangeovy funkce.

2.6 Implementace v systému UFO

V této části si detailněji popíšeme, jak je metoda vnitřního bodu realizována v systému UFO. Stručný popis systému UFO lze nalézt v příloze A. Nejprve uvedeme seznam nejdůležitějších šablon, ze kterých se v systému UFO sestaví program v jazyce Fortran 77.

`U2FSI1` Driver metody vnitřního bodu. Obsahuje deklarace všech proměnných specifických pro metodu vnitřního bodu. Zabezpečuje základní vnější cyklus iterační metody včetně testu konvergence. Volá všechny ostatní potřebné podprogramy.

`UDSLI3` Tato šablona vypočítává směrový vektor. Metodou sdružených gradientů řeší soustavu lineárních rovnic (2.4). Obstarává též předpodmínění metody sdružených gradientů. Volá `UCIACT`, `UCINAT` a `UCISEC`. Provádí výpočet směrové derivace - volá `UCIDER`.

`UCIACT` Šablona slouží k určování aktivních omezení. Aktivní omezení se určují podle (2.30).

`UCINAT` Upravuje gradient na pravé straně soustavy g a Hessovu matici G ze soustavy (2.2). Výsedkem je vektor $\hat{g}$ a matice $\hat{G}$ v soustavě (2.4).

`UCISEC` Vypočítává poslední část pravé strany soustavy (2.4), tj. $-\hat{c}_I + \mu\Lambda_I^{-1}e$, a jihovýchodní roh matice soustavy, tj $-\Lambda_I^{-1}\hat{S}$.

`UP1LF5` Vypočítává hodnotu a gradient účelové funkce a hodnoty a gradienty vazbových funkcí. Zároveň počítá maximální odchylku rovnostních vazeb.

`UP0AF5` Šablona počítá hodnotu rozšířené Lagrangeovy funkce.

`UD1TL3` Testuje spádovost vapočteného směrového vektoru a provádí přípravy na výpočet délky kroku.

`UP0BF1` K účelové funkci $f(x)$ přidává logaritmický bariérový člen $\ln s_i$.

`UF1HS2` Numerický výpočet Hessovy matice účelové funkce. Výpočet Hessovy matice se dá zadat i analyticky, ale obvykle se provádí numericky pomocí tohoto podprogramu.

`UCIDER` Provádí výpočet směrové derivace hodnotové funkce.

`UCIDIR` Zde se provádí krok pro pomocné proměnné s a duální proměnné λ_E a λ_I.

`UCIDOT` Vypočítává residuum při řešení soustavy rovnic metodou sdružených gradientů, které slouží pro kontrolu přesnosti vypočteného směrového vektoru.

`UCIDEL` Provádí výpočet přírůstků pomocných proměnných s pro aktivní omezení a přírůstků neaktivních Lagrangeových multiplikátorů λ_I.

`UCIMAX` Provádí poměrový test na přírůstky pomocných proměnných s a Lagrangeových multiplikátorů a vypočítává hodnoty maximální délky primárního a duálního kroku $\alpha_{\max}$ a $\tilde{\alpha}_{\max}$.

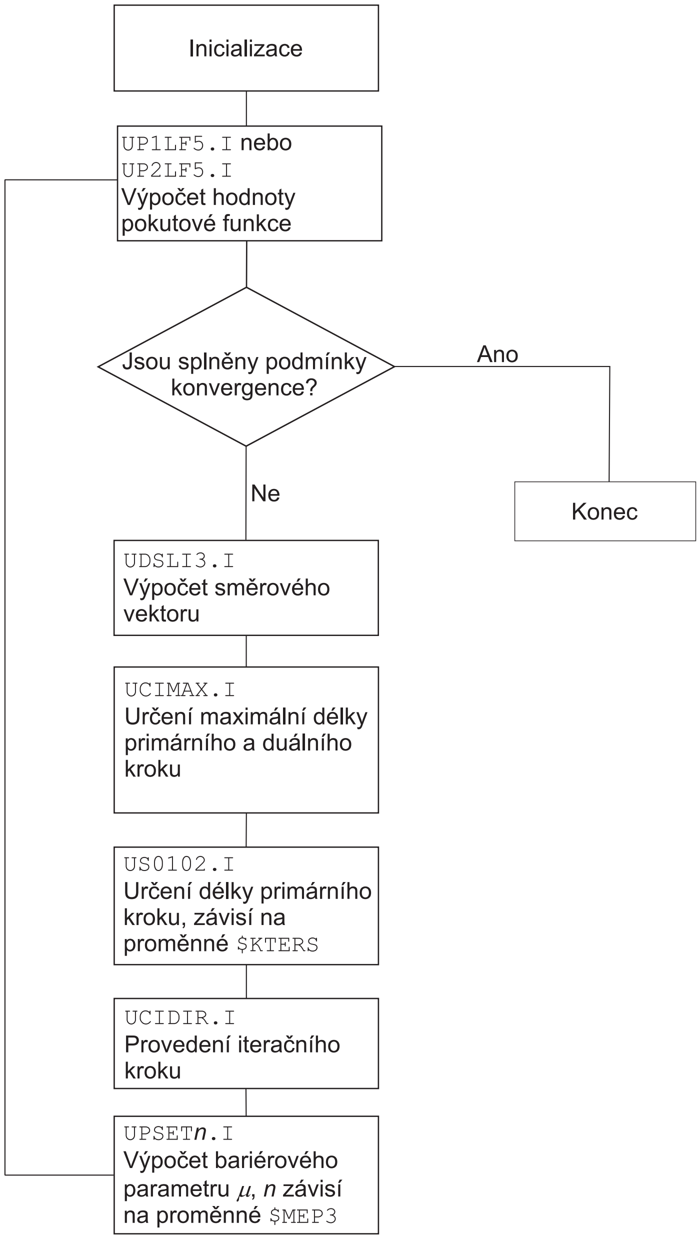
`USOL02` Určuje délku primárního kroku podle zvoleného pravidla (standardně Armijovo pravidlo). Vhodná délka kroku α se hledá v intervalu $(0, \alpha_{\max}]$.

`UCISET` Nastavuje počáteční hodnoty pomocných proměnných s a Lagrangeových multiplikátorů λ_I podle (2.27) a (2.28) na hodnotu minimálně 0.1.

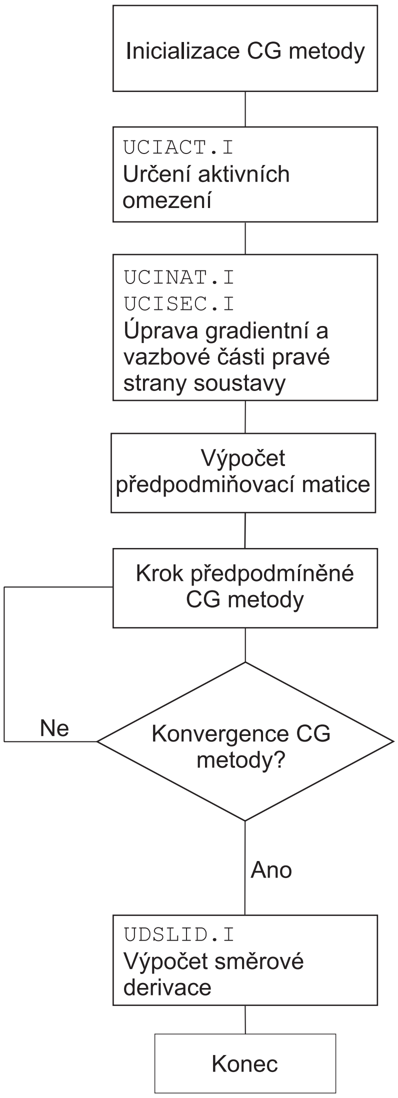
`UPSET`n Různé metody výpočtu bariérového parametru μ. Funkční jsou šablony pro n =`3,5,6,7`. Šablona `UPSET3` odpovídá výpočtu podle (1.37), `UPSET5` odpovídá (1.36), a `UPSET6` a `UPSET7` jsou výpočty podle M. Argaez [1] popisované v (1.7 Volba bariérového parametru μ). Volba n se řídí hodnotou proměnné `$MEP3`

Schéma na obrázku 2.1 popisuje práci šablony `U2FSI1.I`. Protože je tato šablona driverem metody, odpovídá blokové schéma algoritmu 2.

Obrázek 2.2 ukazuje schéma šablony `UDSLI3.I`. Zkratka „CG“ ve schématu znamená sdružené gradienty. Podotkněme, že výsledky volání dalších podprogramů z této šablony (jako je například šablona `UCIACT`, která určuje aktivní omezení) zůstávají přístupné i z kontextu ostatních šablon, které se volají později.



Obrázek 2.1: Blokové schéma šablony `U2FSI1.I`



Obrázek 2.2: Blokové schéma šablony `UDSLI3.I`

2.7 Numerické výsledky

Algoritmus implementovaný v systému UFO byl testován na sadě testovacích úloh převzatých z [13]. Tyto úlohy jsou původně navrženy pro testování algoritmů pro optimalizaci s rovnostními omezeními. Rovnostní omezení však lze předefinovat na nerovnostni, a tak z původních 18 testovacích úloh vziklo 35 úloh pro nerovnostní omezení (některé původní úlohy nelze pro nerovnostní omezení použít). Všechny použité úlohy mají nastavenu dimenzi (počet proměnných) na 100. Počet nelineárních vazeb se liší příklad od příkladu.

Nejprve si ukážeme důležitost předpodmiňovače při používání metody sdružených gradientů pro řešení soustavy lineárních rovnic. Následující tabulka udává výsledky s a bez použití předpodmiňovače pro řešení soustavy rovnic metodou sdružených gradientů:

Problém	NIT[1]	NFV[1]	NFG[1]	NIT[2]	NFV[2]	NFG[2]
1	34	35	204	99	106	596
2	13	13	182	70[4]	1004	994
3	16	17	96	13	13	78
4	21	25	126	66	66	396
5	69	72	690	199	292	1990
6	57	58	798	61	80	854
7	22	22	154	16	16	112
8	370	442	2590	173[3]	252	1211
9	16[3]	132	112	173[3]	252	1211
10	11	11	66	14	14	84
11	40	46	240	19	24	114
12	19	19	133	39	92	273
13	22	22	176	19	45	152
14	24	24	168	28	28	196
15	31	34	186	53	53	318
16	54	62	270	23	23	115
17	51	92	255	105	169	525
18	20	22	100	16	16	80
19	19	44	114	169	693	1020
20	15	15	210	21	21	294
21	22	22	132	29	32	174
22	27	33	162	32	34	192
23	12	12	120	13	13	130
24	147	193	2058	146	305	2058
25	13	13	91	14	14	98
26	331	392	2177	14	14	98
27	85	100	595	77	159	539
28	22	25	132	77	159	539
29	28	36	168	60	105	360
30	20	23	140	18	18	108
31	25	25	200	56	242	392
32	41	41	287	33	37	264
33	21	23	105	83	191	581
34	16	16	80	18	18	90
35	11	12	55	13	13	65
Celkem	1725	2572	15778	2027	4496	15824

[1] s použitím předpomiňovače
[2] bez použitím předpomiňovače
[3] algoritmus selhal při výběru délky kroku
[4] došlo k překročení maximálního počtu vyhodnocení funkční hodnoty

Zajímavý je také poměr počtu vyhodnocení gradientů nelineárních omezení (NCG), kde $\texttt{NCG}^1$=39159 a $\texttt{NCG}^2$=116921. Tento obrovský rozdíl je přímím důsledkem vetšího počtu iterací metody sdružených gradientů, která pracuje mimo jiné s Jacobiho maticí vazbových funkcí.

V následující tabulce je možné pozorovat rozdíl mezi výpočtem, probíhajícím s výběrem délky kroku podle Armijova pravida a výpočtem bez výběru délky délky, tj. s nezkrácenou délkou kroku. Krácení kroku v druhém případě bylo prováděno pouze pro zachování kladnosti pomocných proměnných a Lagrangeových multiplikátorů.

Problém	NIT[1]	NFV[1]	NFG[1]	NIT[2]	NFV[2]	NFG[2]
1	34	35	204	33	33	198
2	13	13	182	46	13	182
3	16	17	96	16	16	96
4	21	25	126	27	27	162
5	69	72	690	82	82	820
6	57	58	798	57	57	798
7	22	22	154	22	22	154
8	370	442	2590	400[4]	401	2807
9	16[3]	132	112	47	47	329
10	11	11	66	11	11	66
11	40	46	240	40	40	240
12	19	19	133	19	19	133
13	22	22	176	22	22	176
14	24	24	168	24	24	168
15	31	34	186	39	39	234
16	54	62	270	48	48	240
17	51	92	255	64	64	320
18	20	20	100	20	20	100
19	19	44	114	17	17	102
20	15	15	210	15	15	210
21	22	22	132	22	22	132
22	27	33	162	27	27	162
23	12	12	120	12	12	120
24	147	193	2058	400[4]	401	5614
25	13	13	91	13	13	91
26	331	392	2177	379	379	2653
27	85	100	595	84	84	588
28	22	25	132	23	23	138
29	28	36	168	55	55	330
30	20	23	140	18	18	126
31	25	25	200	25	25	200
32	41	41	287	41	41	287
33	21	23	105	24	24	120
34	16	16	80	16	16	80
35	11	12	55	19	19	95
Celkem	1055	2669	10920	2174	2176	18271

[1] s výběrem délky kroku podle Armijova ptavidla
[2] bez krácení délky kroku
[3] algoritmus selhal při výběru délky kroku

[4] došlo k překročení maximálního počtu vyhodnocení funkční hodnoty

I zde můžeme porovnat počet vyhodnocení gradientů nelineárních omezení (`NCG`), kde `NCG`[1]`=39159` a `NCG`[2]`=45246`. Je vidět, že výběr délky kroku podle Armijova pravidla s použitou hodnotovou funkcí (1.35) výsledky oproti výpočtu bez krácení kroku o mnoho nezlepší. Ve dvou případech se ale díky krácení kroku podařilo dostatečně snížit počet iterací pod zadanou horní mez. V jednom případě však Armijovo pravidlo selhalo.

Jako poslední si uvedeme výsledky porovnávající jednotlivé metody pro výpočet bariérového parametru μ. V prvním dvousloupci sloupci jsou výsledky s výpočtem μ podle (1.37) - volba `$MEP3=1`; v druhém pro volbu `$MEP3=2` výpočet dle bodu 2 v (1.7 Volba bariérového parametru μ) podle [1]; další dvousloupec používá `$MEP3=3`, tj. výpočet podle (1.36); jako poslední je výpočet podle bodu 1 v (1.7 Volba bariérového parametru μ) podle [1] s `$MEP3=4`.

Problém	NIT[1]	NFV[1]	NIT[2]	NFV[2]	NIT[3]	NFV[3]	NIT[4]	NFV[4]
1	34	35	24[5]	80	34	34	38	43
2	13	13	8[5]	90	122[5]	1003	8[5]	90
3	16	14	15[5]	57	14	14	12	12
4	21	25	125[5]	156	43	45	125[5]	156
5	69	72	29[5]	74	38	43	29[5]	75
6	57	58	18	18	53	53	55	56
7	22	22	17	17	17	17	14	14
8	370	422	145[5]	221	400[7]	401	145[5]	221
9	16[5]	132	178[6]	1005	34	238	178[7]	1005
10	11	11	11	11	11	11	11	11
11	40	46	43[5]	87	50	56	44	46
12	19	19	25	25	18	18	25	25
13	22	22	20	21	22	25	18	19
14	24	24	136[6]	1003	24	24	119[7]	1005
15	31	34	139[6]	1002	34	36	43	54
16	54	62	10[5]	55	84	109	10[5]	55
17	51	92	9[5]	104	62	96	74	640
18	20	20	15[5]	57	20	20	22	22
19	19	44	105[5]	1000	99[6]	1007	129[6]	1003
20	15	15	10[5]	52	14	14	13[5]	81
21	22	22	25	25	24	24	20	20
22	27	33	27[5]	75	36	43	29	34
23	12	12	16	16	13	13	14	14
24	147	193	53	53	57	74	50	50
25	13	13	17	17	34	36	16	16
26	311	392	193	275	400[7]	494	188[5]	312
27	85	100	153[6]	1001	110[6]	1002	93	96
28	22	25	28	50	37	62	20	41
29	28	36	28[5]	76	26	29	24	30
30	20	23	34	41	18	19	31	42
31	25	25	21[5]	64	24	24	26	27
32	41	41	25[5]	68	51	41	25[5]	68
33	21	23	21[5]	93	20	20	22	22
34	16	16	7[5]	49	17	17	7[5]	49
35	11	12	5[5]	72	10	11	5[5]	72
Celkem	1725	2572	1735	7111	2156	5951	1682	5526

[1] `$MEP3=1` [2] `$MEP3=2` [3] `$MEP3=3` [4] `$MEP3=4`
[5] Algoritmus selhal při výběru délky kroku
[6] Algoritmus překročil maximální počet vyhodnocení funkce
[7] Algoritmus překročil maximální počet iterací

Ukazuje se, že výpočet bariérového parametru podle [1], kde obě metody vycházejí z KKT residuové hodnotové funkce, nejsou v kombinaci s hodnotovou funkcí (1.35) tou nešťastnější volbou. Oproti tomu jsou postupy založené na odhadu centrality podle (1.36) velmi účinné.

2.8 Špatná podmíněnost a vliv strojové artitmetiky

Špatná podmíněnost Hessovy matice v soustavě primární metody vnitřního bodu pronásledovala tyto metody po více než dvacet pět let. Pro nelineární programování byla považována za fatální, ale jen zřídka kdy byly její katastrofické důsledky dostatečně zdokumentovány. Je obecně dobře známo, že podmíněnost Hessovy matice původní metody vyvinuté a analyzované Fiacco a McCromickem [6] v 60. letech se zhoršuje, jak se iterace přibližují k řešení (viz. např. [11], [16] nebo [22]). Existuje více způsobů, jak se vyhnout špatně podmíněné matici soustavy (která obsahuje špatně podmíněnou Hessovu matici Lagrangeovy funkce). Jeden z nich je prezentován v 2.1 Určení směrového vektoru, jiný je možné najít v [22]. Mnohé metody však používají běžné řešiče soustav lineárních rovnic přímo na řešení špatně podmíněného systému. Ani při tomto postupu však byly pouze málokdy zaznamenány škodlivé vlivy špatné podmíněnosti. Špatně podmíněné systémy lineárního programování a problémů lineární komlementarity jsou detailně analyzovány např. v [24], [24] nebo [24] a symetrizované primárně-duální systémy nelineárního programování v [7] nebo [8].

Pokud je metoda vnitřního bodu implementována ve strojové aritmetice, může docházet ke třem základním typům chyb:

- chyby vzniklé vlivem ořezávání (často se používá anglický termín „cancellation“) při vyhodnocování funkčních hodnot, gradientů, Hessovy matice a pravé strany soustavy
- zaokrouhlovací chyby vzniklé při vyhodnocování funkčních hodnot, gradientů, Hessovy matice a pravé strany soustavy
- zaokrouhlovací chyby a vliv ořezávání při řešení soustavy lineárních rovnic

Zaokrouhlovací chyby vznikají vždy při vyhodnocování funkce. Funkční hodnotu je nutné zaokrouhlit na nejbližší číslo zobrazitelné ve zvolené strojové aritmetice (v systému UFO se standardně používá Double-precision aritmetika podle normy IEEE). Ořezávání (cancellation) je jev, který nastává při sčítání (a pochopitelně odčítání) ve strojové aritmetice. Pokusíme-li se například sečíst dvě čísla, kde jedno je „velice velké“ a druhé je „velice malé“, může se stát, že se malé číslo v součtu vůbec neprojeví. Pojmy „velice velký“ a „velice malý“ souvisejí s přesností zvolené aritmetiky a klíčový je zde význam tzv. zaokrouhlovací jednotky, někdy také strojového epsilon, které se často značí ϵ_{mach}. Zaokrouhlovací jednotka je definována jako největší číslo zobrazitelné v dané strojové aritmetice, pro které platí

$$1 + \epsilon_{\text{mach}} = 1.$$

Předchozí výraz je příkladem vlivu ořezávání. Zajímavé je, že záleží i na pořadí, ve kterém sčítání probíhá. Patí totiž

$$(\epsilon_{\mathrm{mach}} + \epsilon_{\mathrm{mach}}) + 1 > (1 + \epsilon_{\mathrm{mach}}) + \epsilon_{\mathrm{mach}}.$$

Ve strojové aritmetice tedy nemusí platit asociativita. Operace + v předchozích dvou výrazech je sčítání ve strojové aritmetice. Je zřejmé, že vliv ořezávání při řešení soustavy lineárních rovnic bude tím větší, čím hůře podmíněná bude matice soustavy. Vlivy strojové aritmetiky na metody vnitřního bodu pro nelineární programování podrobněji studovány např. v [27] nebo [24].

Kapitola 3

Teoretické pozadí

3.1 Lokální konvergence bariérových metod

Klíčovým bodem této kapitoly je věta o lokální konvergenci bariérových metod. Tato věta je analogií vět 8 a 10 z Fiacco a McCormic [6] o lokální konvergenci bariérových metod. Tato věta má dva důležité předpoklady. Prvním je požadavek, aby množina $\mathcal{M}^*$ lokálních bodů minima účelové funkce byla neprázdná a kompaktní (v nejjednodušším případě je $\mathcal{M}^*$ jediný bod). Druhou podmínkou je, aby alespoň jeden bod v $\mathcal{M}^*$ ležel v uzávěru striktně přípustné oblasti $\mathcal{F}^0$, což znamená, že je buď striktně přípustný nebo je to hromadný bod nějaké posloupnosti z $\mathcal{F}^0$.

Definice 7 *Izolovaná podmnožina* Nechť $\mathcal{M}$ a $\mathcal{M}^*$ jsou množiny v R^n. Množina $\mathcal{M}^*$ se nazývá izolovaná podmnožina $\mathcal{M}$, jestliže existuje uzavřená množina E taková, že $\mathcal{M}^* \subset \mathrm{int}(E)$ a $E \cap \mathcal{M} = \mathcal{M}^*$.

Jinými slovy množina $\mathcal{M}^*$ je „oddělena“ množinou E od ostatních bodů $\mathcal{M}$. Poznamenejme, že definice je splněna například pro $\mathcal{M}^* = \mathcal{M}$ nebo v nejjednodušším případě pokud je $\mathcal{M}^*$ izolovaný bod v $\mathcal{M}$.

Budeme teď uvažovat problém

$$\min_{x \in \mathrm{R}^n} f(x) \quad \text{tak, aby} \quad c_i(x) \geq 0,\, i = 1 \ldots m. \tag{3.1}$$

Zavedeme logaritmickou bariérovou funkci $B(x, \mu)$ k (3.1)

$$B(x, \mu) = f(x) - \mu \sum_{i=1}^{m} \ln c_i(x), \tag{3.2}$$

kde bariérový parametr μ je vždy kladný. Nechť $\mathcal{F}$ značí přípustnou oblast pro minimalizační problém (3.1) a $\mathcal{F}^0$ značí striktně přípustnou oblast, která je pro (3.1) rovna $\mathcal{F}^0 = \mathrm{int}\mathcal{F}$. Označením f^* budeme mínit minimální hodnotu účelové funkce $f(x)$ pro $x \in \mathcal{F}$.

Poznámka To, zda bereme omezení ve tvaru $c_i(x) \geq 0$ nebo $c_i(x) \leq 0$ nehraje roli, neboť stačí brát místo funkce $c_i(x)$ funkci $-c_i(x)$ a dostaneme opačnou nerovnost.

V následujících větách budeme předpokládat, že funkce $f(x)$ i funkce $c_i(x)$ jsou spojité na množině $\overline{\mathcal{F}^0}$.

Věta 1 *Existence kompaktní nadmnožiny* Nechť $\mathcal{M}$ je množina bodů minima účelové funkce $f(x)$ odpovídající hodnotě f^*. Budeme předpokládat, že $\mathcal{M}$ je neprázdná. Budeme dál předpokládat, že $\mathcal{M}^* \subseteq \mathcal{M}$ je neprázdná kompaktní izolovaná podmnožina $\mathcal{M}$.

Potom existuje kompaktní množina S taková, že $\mathcal{M}^*$ leží v $\text{int}(S) \cap \mathcal{F}$, s tím, že pro každý přípustný bod y v S, který ale neleží v $\mathcal{M}^*$, platí, že $f(y) > f^*$.
Důkaz této věty je možné najít v M. H. Wright [21].

Věta 2 *Lokální konvergence bariérových metod* Nechť $\mathcal{M}$ je množina bodů minima účelové funkce $f(x)$ odpovídající hodnotě f^*. Nechť $\{\mu_k\}$ je klesající posloupnost kladných bariérových parametrů (platí tedy, že $\lim_{k\to 0} \mu_k = 0$). Budeme předpokládat, že

(a) existuje neprázdná kompaktní množina lokálních bodů minima $\mathcal{M}^*$, která je izolovanou podmnožinou $\mathcal{M}$;
(b) alespoň jeden bod $\mathcal{M}^*$ se nachází v uzávěru $\mathcal{F}^0$.

Potom platí následující

(i) existuje kompaktní množina S, taková, že $\mathcal{M}^*$ je její vlastní podmnožina a pro libovolný $\hat{x}$, který je v S, ale není v $\mathcal{M}^*$, platí $f(\hat{x}) > f^*$
(ii) pro všechny dostatečně malá μ_k má $B(x,\mu)$ alespoň jedno nevázané minimum v $\mathcal{F}^0 \cap \text{int}(S)$ a posloupnost globálních nevázaných minim $B(x,\mu)$ v $\mathcal{F}^0 \cap \text{int}(S)$ obsahuje alespoň jednu konvergentní vybranou posloupnost
(iii) nechť $\{x_k\}$ je konvergentní vybraná poslounost globálních nevázaných minim $B(x,\mu)$ v $\mathcal{F}^0 \cap \text{int}(S)$ potom je limita posloupnosti $\{x_k\}$ v $\mathcal{M}^*$
(iv) $\lim\limits_{k\to 0} f(x_k) = f^* = \lim\limits_{k\to 0} B(x_k,\mu)$

Důkaz (podle [21])
(i) plyne přímo z věty 1, která dává existenci vlastní kompaktní nadmnožiny S, v níž všechny body, které jsou zároveň v $\mathcal{M}^*$, jsou globální vázané body minima f. Uvažujme nyní chování bariérové funkce $B(x,\mu)$ v omezené množině $\mathcal{F}^0 \cap S$. Spojitost funkce f a funkcí $\{c_i\}$ implikuje, že $B(x,\mu)$ je spojitá na množině $\mathcal{F}^0 \cap S$. Bariérová funkce $B(x,\mu)$ dosahuje dle [21] Lemma 1 (funkce $B(x,\mu)$ je bariérová funkce mající na hranici $\mathcal{F}$ kladnou singularitu) konečného globálního minima v $\mathcal{F}^0 \cap S$. Nechť y_k je libovolný bod v $\mathcal{F}^0 \cap S$, pro který dosahuje funkce $B(x,\mu_k)$ svého minima.

Posloupnost $\{y_k\}$ je omezená, a má tedy alespoň jeden hromadný bod. Nechť tedy $\hat{x}$ je hromadným bodem $\{y_k\}$. Protože y_k je pro všechny k striktně přípustný bod a množina S je kompaktní, dostaneme, že $\hat{x} \in \mathcal{F} \cap S$, $\hat{x}$ je tedy přípustný.

Chceme dokázat, že $\hat{x}$ leží v množině $\mathcal{M}^*$ vázaných minim f, a že $f(\hat{x}) = f^*$. Důkaz provedeme sporem. Budeme tedy předpokládat, že $\hat{x} \notin \mathcal{M}^*$.

Jelikož je bod $\hat{x}$ přípustný a zároveň v množině S, výsledek (i) nám říká, že $f(\hat{x}) > f^*$. Dále dokážeme, že tato nerovnost implikuje existenci striktně přípustného bodu x_{int} v množině S, takového, že

$$f(\hat{x}) > f(x_{int}). \tag{3.3}$$

Bod x_{int} můžeme najít následujícím způsobem. Z předpokladu (b) věty víme, že alespoň jeden bod z $\mathcal{M}^*$ leží v uzávěru množiny $\mathcal{F}^0$. Nechť tedy x^* je takový bod, což znamená, že x^* buď leží v $\mathcal{F}^0$ nebo je hromadným bodem $\mathcal{F}^0$. Protože je $\mathcal{M}^*$ obsažena v $\operatorname{int}(S)$, je také bod x^* ve vnitřku množiny S.

Pokud je bod x^* sám striktně přípustný, můžeme za x_{int} brát x^*. Pokud x^* není striktně přípustný, znamená to, že x^* je hromadným bodem v $\mathcal{F}^0$, z čehož plyne, že libovolné okolí x^* obsahuje striktně přípustné body (nekonečně mnoho). Dále dostaneme, že libovolné okolí x^* obsahuje body z S. Víme, že f je spojitá funkce, $\hat{x}$ je přípustný a leží v S, $f(\hat{x}) > f(x^*)$ a x^* je globální vázaný bod minima pro všechny přípustné body v S. Musí tedy existovat striktně přípustný bod x_{int} v okolí x^*, pro který platí, že $f(x_{int}) < f(\hat{x})$.

Nechť $\{x_k\}$ je konvergentní vybraná posloupnost $\{y_k\}$, která má limitu $\hat{x}$. Vztah $f(\hat{x}) > f(x_{int})$ nám potom pro dostatečně velké k dává

$$f(x_k) > f(x_{int}). \tag{3.4}$$

Protože x_{int} je v $\mathcal{F}^0 \cap S$, naše definice x_k jako globálního bodu minima funkce $B(x, \mu_k)$ v $\mathcal{F}^0 \cap S$ implikuje nerovnost

$$f(x_k) - \mu_k \sum_{i=1}^{m} \ln c_i(x_k) \leq f(x_{int}) - \mu_k \sum_{i=1}^{m} \ln c_i(x_{int}). \tag{3.5}$$

Striktní přípustnost x_{int} znamená, že bariérový člen výrazu na pravé straně nerovnosti v (3.5) je konečný a

$$\lim_{k \to \infty} B(x_{int}, \mu_k) = f(x_{int}). \tag{3.6}$$

Předpolkádejme nyní, že limita $\hat{x}$ posloupnosti $\{x_k\}$ je také striktně přípustná, tedy $\hat{x} \in \mathcal{F}^0 \cap S$. Potom je bariérový člen (3.5) v x_k konečný pro $k \to \infty$ a

$$\lim_{k \to \infty} B(x_k, \mu_k) = f(\hat{x}). \tag{3.7}$$

Limitním přechodem pro $k \to \infty$ v (3.5) dostaneme nerovnost $f(\hat{x}) \leq f(x_{int})$, a tím i spor s nerovností (3.3).

Předpolkádejme teď, že $\hat{x}$ není striktně přípustný. Přidáním bariérových členů na každou stranu nerovnosti $f(x_k) > f(x_{int})$ dostaneme

$$f(x_{int}) - \mu_k \sum_{i=1}^{m} \ln c_i(x_{int}) < f(x_k) - \mu_k \sum_{i=1}^{m} \ln c_i(x_{int}). \tag{3.8}$$

Použijeme-li ještě nerovnost (3.5), upravíme a vydělíme μ_k, dostaneme

$$f(x_k) - \mu_k \sum_{i=1}^{m} \ln c_i(x_k) < f(x_k) - \mu_k \sum_{i=1}^{m} \ln c_i(x_{int}). \tag{3.9}$$

Z obou stran odečteme $f(x_k)$ a výsledkem je nerovnost

$$-\mu_k \sum_{i=1}^{m} \ln c_i(x_k) < -\mu_k \sum_{i=1}^{m} \ln c_i(x_{int}). \tag{3.10}$$

Striktní přípustnost x_{int} nám opět zaručuje, že suma na pravé straně nerovnosti (3.10) je konečná a má pevnou hodnotu pro všechna k. Předpokládáme, že $\hat{x}$ není striktně přípustný. Výraz $-\ln c_i(x_k)$ se tedy pro $k \to \infty$ blíží k nekonečnu pro alespoň jeden index i. Levá strana nerovnosti (3.10) je tedy shora neomezená a opět dostáváme spor.

V obou případech jsme ukázali, že $f(\hat{x}) = f^*$ a tedy $\hat{x} \in \mathcal{M}^*$. Vzhledem k tomu, že $\hat{x}$ byl vybrán jako libovolný hromadný bod $\{y_k\}$, můžeme říci, že každý hromadný bod posloupnosti globálních minim funkce $B(x, \mu_k)$ ležící v $\mathcal{F}^0 \cap S$ musí být vázaný bod minima účelové funkce $f(x)$ s hodnotou f^*.

Bod (ii) lze dokázat konstatováním, že $\hat{x} \in \mathcal{M}^*$ znamená, že $\hat{x} \in \mathrm{int}S$. Jelikož je $\hat{x} = \lim_{k\to\infty} x_k$, musí platit, že také $x_k \in \mathrm{int}S$ pro dostatečně velká k. Body x_k jsou z definice striktně přípustné. Globální bod minima $\hat{x}$ funkce $B(x, \mu_k)$ je tedy nevázaným bodem minima. Výsledky (ii) a (iii) jsou tedy dokázány.

První výraz v (iv), tj. $\lim_{k\to\infty} f(x_k) = f^*$, plyne z rovnosti $f(\hat{x}) = f^*$. Než budeme dokazovat druhou rovnost v (iv), poznamenejme nejprve, že optimální hodnota bariérové funkce $B(x, \mu_k)$ je v S jednoznačná a je rovna $B(x_k, \mu_k)$.
Budeme rozlišovat dva případy v závislosti na tom, zda bod $x^* = \lim_{k\to\infty} x_k$ je nebo není striktně přípustný.
Pokud je x^* striktně přípustným bodem, zůstane součet logaritmů vazbových funkcí konečný pro $k \to \infty$. Pro tento případ je jasné (protože $\mu_k \to 0$), že $\lim_{k\to\infty} B(x_k, \mu_k) = f^*$.

Uvažujme tedy opačnou možnost, kdy x^* není striktně přípustný bod. Protože hodnota alespoň jedné vazbové funkce konverguje k nule, musí pro všechna dostatečně velká k být bariérový člen $B(x_k, \mu_k)$ kladný. Zkombinujeme-li tuto vlastnost s (3.5), dostaneme nerovnost

$$0 < -\sum_{i=1}^{m} \ln c_i(x_k) \leq -\sum_{i=1}^{m} \ln c_i(x_{k+1}). \tag{3.11}$$

Jeden z důsledků (3.11) je, že pro dostatečně velká k platí

$$B(x_k, \mu_k) > f(x_k). \tag{3.12}$$

Navíc to, že x_{k+1} je bod minima $B(x, \mu_{k+1})$ v S, první nerovnost v (3.11) a nerovnost $\mu_{k+1} \leq \mu_k$ dohromady dávají nerovnosti

$$
\begin{aligned}
f(x_{k+1}) - \mu_{k+1}\sum_{i=1}^{m} \ln c_i(x_{k+1}) &\leq f(x_k) - \mu_{k+1}\sum_{i=1}^{m} \ln c_i(x_k) \\
f(x_k) - \mu_{k+1}\sum_{i=1}^{m} \ln c_i(x_k) &\leq f(x_k) - \mu_k\sum_{i=1}^{m} \ln c_i(x_k),
\end{aligned}
$$

které ukazují, že pro dostatečně velké k je

$$B(x_{k+1}, \mu_{k+1}) \leq B(x_k, \mu_k). \tag{3.13}$$

Z (3.12) a (3.13) plyne, že pro dostatečně velká k, je

$$f^* \leq \ldots \leq B_{k+1} \leq B_k, \tag{3.14}$$

kde B_k je hodnota $B(x_k, \mu_k)$. Posloupnost $\{B_k\}$ hodnot bariérové funkce je nerostoucí a omezená zespoda, a tedy musí shora monotónně konvergovat k limitě. Tuto limitu budeme označovat B^* a platí, že $B^* \geq f^*$.

Cílem je, aby platilo $B^* = f^*$. Budeme předpokládat opak, tedy, že $B^* > f^*$. Pro tento případ budeme definovat kladné reálné číslo $\delta = \frac{1}{2}(B^* - f^*)$. Ze spojitosti f a nerovnosti $f^* < B^*$ plyne, že musí v S existovat okolí bodu x^* tak, že platí

$$f(x) \leq B^* - \delta \tag{3.15}$$

pro všechna x v tomto okolí. Vezměme konkrétní striktně přípustný bod $\overline{x}$ z tohoto okolí. Ze striktní přípustnosti $\overline{x}$ plyne, že hodnota součtu $\sum_{i=1}^{m} \ln c_i(\overline{x})$ je konečná. Protože platí $\mu_k > 0$ a $\mu_k \to 0$, musí existovat kladné celé číslo K takové, aby pro všechna $k \geq K$ bylo

$$-\mu_k \sum_{i=1}^{m} \ln c_i(\overline{x}) < \frac{1}{2}\delta. \tag{3.16}$$

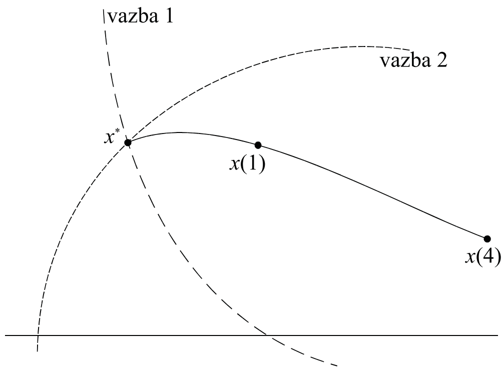
Vzhledem k tomu, že x_k je globální bod minina $B(x, \mu_k)$ v S, víme, že

$$B(x_k, \mu_k) \leq B(\overline{x}, \mu_k) = f(\overline{x}) - \mu_k \sum_{i=1}^{m} \ln c_i(\overline{x}).$$

Pokud použijeme (3.15) a (3.16), výsledkem je

$$B(x_k, \mu_k) < B^* - \delta + \frac{1}{2}\delta = B^* - \frac{1}{2}\delta,$$

což je spor s monotónní konvergencí $\{B_k\}$ k B^* shora. Závěr tedy je, že $B^* = f^*$, což je požadovaný výsledek (iv). $\square$



Obrázek 3.1: Obrázek ukazuje centrální cestu blížící se k řešení odraženě od všech aktivních vazeb

Podle předchozí věty se dá přímo sestavit algoritmus (Algoritmus 2) pro numerický výpočet. Postup je jednoduchý: určit μ_k, spočítat nevázané minimum bariérové funkce $B(x,\mu)$ a toto celé opakovat, až jsou splněny konvergenční podmínky. Tento postup má tu nevýhodu, že je přesný výpočet minima $B(x,\mu)$ příliš náročný. Proto se tento výpočet provádí nepřesně, jen v několika málo iteracích.

Pokud jsou splněny podmínky věty 2 a podmínky striktní komplementarity, platí, že centrální cesta $x(\mu)$ definovaná jako minima funkce $B(x,\mu)$, se k řešení x^* blíží odraženě od všech aktivních vazeb, tzn. nikoliv podél některé z vazeb. Důkaz tohoto tvrzení lze nalézt v H. H. Wright [21].

Tato vlastnost centrální cesty je velice důležitá, protože kdyby se centrální cesta mohla k řešení blížit podél některé z vazeb, bylo by nutné hledat minimum $B(x,\mu)$ velice přesně a volit velice krátké kroky, aby se iterace nedostala mimo striktně přípustnou oblast. To by znamenalo značné zpomalení popř. i zastavení výpočtu. Z důkazu je také zřejmé, že je zde podstatná vlastnost striktní komplementarity, bez níž by se odražení od vazeb nedalo zaručit.

3.2 Základní pojmy z teorie konvergence

Pod označním $\{x_i\}_{i=1}^{\infty}$ budeme myslet posloupnost reálných čísel takovou, že $\lim_{i\to\infty} x_i = x^*$.

Definice 8 (*R-Lineární konvergence*) Nechť posloupnost $\{x_i\}$ konverguje k x^*. Jestliže existuje index k a konstanty $0 < M_k < \infty$ a $0 < q < 1$ tak, že

$$\|x_i - x^*\| \le M_k q^{i-1} \|x_k - x^*\| \qquad \forall i \ge k,$$

tj.

$$\frac{\|x_i - x^*\|}{\|x_k - x^*\|} \le M_k q^{i-1} \qquad \forall i \ge k,$$

řekneme, že posloupnost $\{x_i\}$ konverguje k x^* R-lineárně

Definice 9 (*Q-Lineární konvergence*) Nechť posloupnost $\{x_i\}$ konverguje k x^*. Jestliže existuje index k a konstanta $0 < q < 1$ tak, že

$$\frac{\|x_{i+1} - x^*\|}{\|x_i - x^*\|} \leq q \qquad \forall i \geq k,$$

řekneme, že posloupnost $\{x_i\}$ konverguje k x^* Q-lineárně

Definice 10 (*Q-Superlineární konvergence*) Řekneme, že posloupnost $\{x_i\}$ konverguje k bodu x^* Q-superlineárně, jestliže

$$\lim_{i\to\infty} \frac{\|x_{i+1} - x^*\|}{\|x_i - x^*\|} = 0.$$

Definice 11 (*Q-Kvadratická konvergence*) Nechť posloupnost $\{x_i\}$ konverguje k x^*. Jestliže existuje index k a konstanta $0 < M_k < \infty$ tak, že

$$\frac{\|x_{i+1} - x^*\|}{\|x_i - x^*\|^2} \leq M_k \qquad \forall i \geq k$$

řekneme, že posloupnost $\{x_i\}$ konverguje k x^* Q-kvadraticky.

Platí následující tvrzení:

Věta 3 Nechť posloupnost $\{x_i\}$ konverguje k x^* Q-lineárně (Q-superlineárně), potom koverguje k x^* také R-lineárně (R-superlineárně). (Důkaz viz. [14])

3.3 Lokální superlineární konvergence primárně-duální metody

V této kapitole si ukážeme, jak rychle může primárně-duální metoda v okolí řešení konvergovat. Uvidíme, že tato rychlost závisí zejména na volbě bariérového parametru μ. Přesněji řečeno na rychlosti s jakou se tento parametr blíží k nule, a že lze algoritmus navrhnout tak, aby konvergence byla superlineární. Pro přehlednost můžeme soustavu primárně-duálních rovnic (1.25)-(1.28) zapsat ve tvaru

$$F_\mu(v) := \begin{bmatrix} g(x) \\ -\mu e + S\Lambda_I e \\ c_E(x) \\ c_I(x) + s \end{bmatrix} = 0, \;\; v = [x, s, \lambda_E, \lambda_I]^T. \tag{3.17}$$

Poznamenejme, že Jacobiho matice $F'_\mu(v)$ je nezávislá na bariérovém parametru μ, a proto můžeme index vynechávat a tuto matici značit $F'(v)$. Obdobně budeme také značit $F_0(v) = F(v)$. Vektor v bude znamenat současnou iteraci a v^+ bude značit nově spočtenou iteraci.

Budeme dále přespokládat, že v bodě x^* platí podmínky optimality a že Hessovy matice zúčastněných funkcí jsou lokálně Lipschitzovsky spojité. Tedy dohromady, že pro x^* platí následující podmínky:

P.1 *Stacionarita* Vektor $v^* = [x^*, s^*, \lambda_E^*, \lambda_I^*]^T$ splňuje KKT pomínky (1.7)-(1.11), tj. $F_0(v^*) = 0, \quad \lambda_I^* \geq 0, \quad s^* \geq 0$

P.2 *Regularita* Gradienty aktivních omezení (tj. všech rovnostních a aktivních nerovnostních) jsou v x^* lineárně nezávislé

P.3 *Optimalita* Je splněna podmínka optimality druhého řádu (1.12), tj. $N^T(x^*)G(x^*)N(x^*) > 0$

P.4 *Striktní komplementarita* Pro nerovnostní omezení platí nerovnost $s^* + \lambda_I^* > 0$

P.5 *Hladkost* Hessovy matice $\nabla^2 f(x)$,$\nabla^2 c_i(x)$, $i \in E \cup I$ jsou lokálně Lipschitzovsky spojité v x^*

Použijeme-li značení (3.17), můžeme definavat primárně-duální krok metody jako

$$F'_\mu(v)\triangle v_N = -F_\mu(v), \tag{3.18}$$

což plně odpovídá soustavě rovnic (1.29) pro Newtonovu metodu. Protože budeme soustavu (3.18) řešit iterační metodou sdružených gradientů, řešením bude nepřesný Newtonův krok. Bude tedy

$$F'_\mu(v)\triangle v = -F_\mu(v) + r_\mu, \tag{3.19}$$

kde r_μ je residuový vektor odpovídající pravé straně (2.6). Nepřesný Newtonův krok $\triangle v$ je s přesným Newtonovým krokem $\triangle v_N$ ve vztahu

$$\triangle v = \triangle v_N + \Big(F'(v)\Big)^{-1} r_\mu \tag{3.20}$$

Velikost residuového vektoru nám vlastně udává, s jakou přesností metoda sdružených gradientů řeší soustavu (3.18). Abychom ukázali rychlost konvergence, uvedeme si nejprve několik pomocných tvrzení a potom větu, která nám řekne, za jakých podmínek bude konvergence superlineární.

Lemma 3 Nechť platí podmínky (P.1)-(P.5). Potom existuje okolí $\delta(v^*) = \{v : \|v - v^*\| \leq \delta\}$ takové, že pro všechna $v \in \delta(v^*)$ je Jacobiho matice $F'(v)$ invertibilní a

$$\left\|\Big(F'(v)\Big)^{-1}\right\| \leq M, \tag{3.21}$$

pro nějakou konstantu $M > 0$. Navíc pro všechna $v_1, v_2 \in \delta(v^*)$ platí

$$\|F'(v_1)(v_1 - v_2) - F(v_1) + F(v_2)\| \leq L(\|v_1 - v_2\|^2), \tag{3.22}$$

pro nějakou konstantu $L > 0$.

Následující lemma je přímým důsledkem věty o implicitní funkci. Říká nám, že řešení $v^*(\mu)$ bariérového problému (1.16) je Lipschitzovsky spojitá funkce v μ.

Lemma 4 Nechť platí podmínky (P.1)-(P.5). Potom existuje $\overline{\mu} > 0$ takové, že pro všechna $\mu < \overline{\mu}$ má soustava $F_\mu(v) = 0$ řešení $v^*(\mu) \in \delta(v^*)$, kde $\delta(v^*)$ je definováno v lematu 3. Navíc platí

$$\|v^*(\mu) - v^*\| \leq C\mu, \tag{3.23}$$

kde,

$$C = \max_{v \in \delta(v^*)} \left\| \Big(F'(v)\Big)^{-1} \frac{\partial F_\mu(v)}{\partial \mu} \right\|$$

je kostanta nezávislá na μ.

Lemma 5 Nechť platí podmínky (P.1)-(P.5) a nechť $\delta(v^*)$ je okolí definované v lemmatu 3 a $\overline{\mu}$ je hodnota definovaná v lemmatu 4. Potom pro libovolné $v \in \delta(v^*)$ a $\mu < \overline{\mu}$ je

$$\|v^+ - v^*(\mu)\| \leq ML\|v - v^*(\mu)\|^2 + M\|r_\mu\|, \tag{3.24}$$

kde M a L jsou konstanty definované v (3.21) a (3.22), a r_μ je residuový vektor metody sdružených gradientů definovaný v (3.18).

Lemma 6 Nechť M a $\delta(v^*)$ jsou definována podle lemmatu 3 a $\overline{\mu}$ je definováno podle lemmatu 4. Potom pro všechna v dostatečně blízko k $v^*(\mu)$ a všechna $\mu < \overline{\mu}$ je

$$\|v - v^*(\mu)\| \leq 2M\|F_\mu(v)\|, \quad \|F_\mu(v)\| \leq 2Q\|v - v^*(\mu)\|, \tag{3.25}$$

kde

$$Q := \sup_{v \in \delta(v^*)} \|F'(v)\|. \tag{3.26}$$

Důkazy lemmat 3 až 6 lze nalézt v [5].

Následující věta dává návod, jak postupovat při snižování bariérového parametru a jak kontrolovat velikost residuového vektoru tak, abychom poblíž řešení docílili rychlý řád konvergence.

Věta 4 ([5]) Nechť platí předpoklady (P.1)-(P.5) a nechť v je iterace dostatečně blízko v^* ve které se bariérový parametr sníží z μ na μ^+. Předpokládejme, že residuový vektor v (3.19) splňuje

$$\|r_{\mu^+}\| \leq M_1 \|F_{\mu^+}(v)\|^{1+\alpha}, \tag{3.27}$$

pro nějaké kladné konstanty M_1 a α. Potom, pokud $\mu^+ = o(\|F(z)\|)$ a $\alpha > 0$, bude krok superlineárně konvergentní k v^*; navíc, pokud $\mu^+ = O(\|F(z)\|^2)$ a $\alpha > 1$, bude krok kvadraticky konvergentní.

Důkaz
Platí

$$\begin{aligned} \|F_{\mu^+}(v)\| &= \|F(v) - \mu^+ [0, e, 0, 0]^T \| \\ &\leq \|F(v)\| + \sqrt{m_I}\mu^+ \\ &= \|F(v)\| \left(1 + \frac{\sqrt{m_I}\mu^+}{\|F(v)\|}\right). \end{aligned}$$

Jelikož platí předpoklad, že $\mu^+ = o(\|F(z)\|)$ nebo $\mu^+ = O(\|F(z)\|^2)$, bude

$$\frac{\sqrt{m_I}\mu^+}{\|F(v)\|} \to 0,$$

z čehož můžeme předpokládat, že

$$\frac{\sqrt{m_I}\mu^+}{\|F(v)\|} \leq 1.$$

Potom tedy bude

$$\|F_{\mu^+}(v)\| \leq 2\|F(v)\|. \tag{3.28}$$

Použijeme-li poslední nerovnost, lemma 4 a lemma 5, dostaneme:

$$\begin{aligned} \|v^+ - v^*\| &\leq \|v^+ - v^*(\mu^+)\| + \|v^*(\mu^+) - v^*\| \\ &\leq ML\|v - v^*(\mu^+)\|^2 + M\|r_{\mu^+}\| + C\mu^+ \\ &\leq ML\left(\|v - v^*\| + \|v^* - v^*(\mu^+)\|\right)^2 + MM_1\|F_{\mu^+}(v)\|^{1+\alpha} + C\mu^+ \\ &\leq ML\left(2\|v - v^*\|^2 + 2\|v^* - v^*(\mu^+)\|^2\right) + MM_1\|F_{\mu^+}(v)\|^{1+\alpha} + C\mu^+ \\ &\leq 2ML\|v - v^*\|^2 + 2MLC^2(\mu^+)^2 + 2^{1+\alpha}MM_1\|F(v)\|^{1+\alpha} + C\mu^+ \\ &\leq 2ML\|v - v^*\|^2 + 2^{1+\alpha}MM_1\|F(v)\|^{1+\alpha} + \\ &\quad (2MLC\overline{\mu} + 1)C\mu^+. \end{aligned} \tag{3.29}$$

Druhá nerovnost v (3.25) nám pro $\mu = 0$ dává

$$\|F(v) \leq 2Q\|v - v^*\|. \tag{3.30}$$

Použijeme-li nerovnosti (3.29) a (3.30) dostaneme

$$\begin{aligned}
\frac{\|v^+ - v^*\|}{\|v - v^*\|} &\leq 2ML\|v - v^*\| + 2^{1+\alpha} MM_1 \frac{\|F(v)\|^{1+\alpha}}{\|v - v^*\|} + \\
&\quad (2MLC\overline{\mu} + 1)C\frac{\mu^+}{\|v - v^*\|} \\
&\leq 2ML\|v - v^*\| + (4Q)^{1+\alpha} MM_1 \|v - v^*\|^\alpha + \\
&\quad (2MLC\overline{\mu} + 1)C\frac{\mu^+}{\|v - v^*\|} \\
&\leq 2ML\|v - v^*\| + (4Q)^{1+\alpha} MM_1 \|v - v^*\|^\alpha + \\
&\quad 2QC(2MLC\overline{\mu} + 1)\frac{\mu^+}{\|F(v)\|},
\end{aligned}$$

z čehož vzhledem k podmínce

$$\mu^+ = o(\|F(v)\|), \alpha > 0 \tag{3.31}$$

plyne, že pravá strana je $o(1)$, takže krok je superlineárně konvergentní. Navíc, nerovnost (3.30) a podmínka

$$\mu^+ = O(\|F(v)\|^2), \alpha \geq 1 \tag{3.32}$$

implikují, že pravá strana je $O(\|v - v^*\|)$, takže konvergence bude kvadratická. □

Příloha A

Standardní postupy pro řešení úloh s nerovnostními vazbami

A.1 Logaritmická bariérová metoda vnitřního bodu

Logaritmickou bariérovou metodou vnitřního bodu se zabývá celá tato práce. Poznamenejme pouze, že logaritmická bariéra (kladná singularita na hranici přípustné oblasti) není jedinou možností jak zajistit, aby iterace setrvávaly vždy uvnitř přípustné oblasti (tj. aby byly striktně přípustné). Alternativou k logaritmickým bariérovým metodám jsou např. pokutové bariérové metody viz. např. [2] nebo [3].

A.2 Sekvenční kvadratické programování (SQP)

Metody SQP jsou spolu s metodami vnitřního bodu v současnosti nejpoužívanějšími a nejefektivnějšími metodami pro řešení úloh nelineárního programování

$$\min_{x \in \mathrm{R}^n} f(x) \text{ tak, aby } g_i(x) \geq 0,\ i = 1, \ldots, m. \tag{A.1}$$

Metody SQP řeší v každé iteraci podproblém kvadratického programování (QP). Každý QP podproblém minimalizuje modifikovanou Lagrangeovu funkci problému (A.1) vzhledem k linearizovaným vazbám. SQP metoda se sestává z majoritních a minoritních iterací (odpovídá vnějšímu a vnitřnímu cyklu). Majoritní iterace generují posloupnost bodů (x_k, λ_k), které konvergují k řešení problému (A.1) - vnější cyklus. Každý QP podproblém generuje směrový vektor k další iteraci (tj. od (x_k, λ_k) k (x_{k+1}, λ_{k+1})). Řešení QP podproblému je samo iterační proces, jehož jednotlivé iterace se v kontextu SQP metody označují jako minoritní - vnitřní cylkus.

Modifikovanou Lagrangeovu funkci k (A.1) označíme $L(x, x_k, \lambda_k) = f(x) + \lambda_k^T c_L(x, x_k)$, kde $g_L(x, x_k) = g(x_k) + A(x_k)(x - x_k)$ je linearizace vazbových funkcí

($A(x)$ je Jacobiho matice $g(x)$) a $c_L(x, x_k) = g(x) - g_L(x, x_k)$ je míra odchylky lineárního modelu od nelineárních vazeb. Podproblém kvadratického programování můžeme formulovat jako

$$\min_{x \in \mathrm{R}^n} f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 L(x_k, x_k, \lambda_k)(x - x_k)$$

tak, aby platila linearizace vazbových funkcí

$$g_L(x, x_k) \geq 0.$$

Výhodou takovéto formulace je, že vyhodnocení $A(x_k)$ a $\nabla f(x_k)$ je potřeba provádět pouze jednou za majoritní iteraci. Bližší popis SQP metod je možné najít např. v Gill, Murray a Wright [9].

A.3 Squared slack variables

Uvažujme problém nelineárního programování s nerovnostními vazbami

$$\min_{x \in \mathrm{R}^n} f(x) \text{ tak, aby } g_i(x) \geq 0,\ i = 1, \ldots, m. \tag{A.2}$$

Pro takto formulovanou úlohu platí následující ekvivalence

$$g_i(x) \geq 0 \Leftrightarrow g_i(x) = y_i^2, \tag{A.3}$$

pro nějaké $y_i \in \mathrm{R}^m$. Můžeme tedy úlohu (A.2) přeformulovat na úlohu s rovnostními vazbami:

$$\min_{x \in \mathrm{R}^n} f(x), \text{ tak, aby } g_i(x) = y_i^2,\ i = 1, \ldots, m, \tag{A.4}$$

pro nějaké pomocné proměnné y_i. Protože je úloha (A.4) ekvivalentní s úlohou (A.2) a úloha (A.4) má pouze omezení ve tvaru rovností, lze použít Newtonovu metodu pro rovnostně vázané problémy. V průběhu let bylo nasbíráno mnoho zkušeností s squared-slack metodou. Pokusíme se nyní shrnout základní výhody a nevýhody squared-slack metody používající Newtonovu metodu pro řešení problému (A.2).

Výhody

1. Úloha nemá žádné nerovnostní omezení.

2. Newtonova metoda je algoritmicky konzistentní vzhledem ke KKT podmínkám (blíže viz. [17]).

Nevýhody

1. Dimenze úlohy (počet proměnných) se zvýší.
2. Stupeň nelinearity nerovnostních vazeb se zvýší a vede ke ztrátě konvexity.
3. Použití pomocných proměnných dle (A.3) zapříčiní násobnost řešení a následně výrazné zhoršení globální konvergence Newtonovy metody.

První nevýhoda není zvláště závažná a lze ji obejít speciálními postupy (viz [18]). Druhá nevýhoda je vážná a hodná zájmu. Avšak třetí nevýhoda je pro squared-slack metodu katastrofická a v podstatě znehodnocuje squared-slack metodu do té míry, že je nepoužitelná jako numerický nástroj. Abychom vysvětlili předchozí tvrzení, uvědomme si, že pokud je (x^*, y^*) řešením úlohy (A.4), je i $(x^*, \hat{y^*})$ řešením téže úlohy, kde $\hat{y^*}$ je y^* s libovolnou složkou zaměněnou za její opačnou hodnotu. Jestliže je tedy $y^* \in \mathrm{R}^m$, je možné $\hat{y^*}$ volit 2^m různými způsoby, a tak vznikne $2^m - 1$ dodatečných řešení. Tento exponenciální nárůst počtu řešení skutečně způsobí značnou neefiktivitu Newtonovy metody. Efektivní konvergence Newtonovy metody se totiž omezí pouze na malé okolí každého řešení. Tato omezující vlastnost se dá demonstrovat i numericky. Přes svoje špatné chování našla squared-slack metoda používající Newtonovu metodu uplatnění ve vědeckých výpočtech v posledních několika desetiletích. Dnes se však již téměř nepoužívá.

A.4 Metoda aktivních omezení (Active Set Strategy)

Budeme uvažovat problém nelineárního programování (A.2). V bodě řešení x^* existuje množina aktivních omezení (active set) $E_* = \{i : g_i(x^*) = 0\}$ a množina neaktivních omezení $I_* = \{i : g_i(x^*) > 0\}$. Bod x^* je řešením obou problémů (A.2) i problému

$$\min_{x \in \mathrm{R}^n} f(x), \text{ tak, aby } g_i(x) = 0, \; i \in E_*. \tag{A.5}$$

Pokud bychom tedy předem věděli, která omezení budou aktivní, mohli bychom ignorovat neaktivní omezení a problém považovat za problém s rovnostními omezeními. Metoda aktivních omezení pracuje tak, že v každé fázi iteračního procesu (např. Newtonovy metody) „hádá“, která omezení budou v řešení aktivní. S těmi pracuje jako s rovnostmi a ostatní ignoruje. Modelovým příkladem metody aktivních omezení je simpexová metoda. Většina strategií aktivních omezení pro nelineární programování přímo navazuje na myšlenku simplexové metody. Dnes se však strategie aktivních omezení uplatňuje téměř výhradně v lineárním popř. kvadratickém programování

Výhody

1. Soustava nelineárních rovnic má menší dimenzi než původní problém.

Nevýhody

1. Stejně jako simplexová metoda, metoda aktivnách omezení pro nelineární programování pracuje v exponenciálním čase, tj. v nejhorším možném případě poroste počet nutných iterací exponenciálně s počtem proměnných.
2. Algoritmus, který v každé iteraci „chytře“ hádá množinu aktivních omezení je výpočetně drahý.

Příloha B

UFO

Tato část obsahuje stručný popis základů systému UFO tak, jak je uveden v [15].

B.1 Úvod

Algoritmus popisovaný v této práci byl implementován v systému UFO. Název UFO pochází z „universal functional optimization“. Je to interaktivní modulární systém pro řešení jak středně velkých hustých, tak velkých řídkých optimalizačních problémů. Systém UFO může být využit k následujícím aplikacím:

1. Formulace a řešení konkrétních optimalizačních problémů, které umí systém vyřešit.
2. Příprava speciálních programů (nebo podprogramů), založených na metodách zabudovaných v systému.
3. Návrh a testování nových optimalizačních metod. Systém UFO je vhodný nástroj pro vývoj algoritmů pro optimalizaci funkcí.

Speciální realizace systému UFO ho činí plně přenosným a rozšiřitelným a jeho vývoj stále pokračuje.

B.2 Principy systému UFO

UFO je otevřený softwarový systém pro řešení široké třídy optimalizačních problémů. Řešení optimalizačního problému je rozděleno do čtyř fází. *První fáze* je formulace problému a výběr optimalizační metody. To může proběhnou třemi různými způsoby:

1. Plně dialogový režim: Zadání problému a výběr metody probíhá interaktivně, kdy uživatel zodpovídá dotazy systému.

2. Dávkový režim: Zadání problému a výběr metody se provádí pomocí kontrolního jazyka systému UFO. Vstupní soubor, psaný v kontrolním jazyce, musí být předem připraven a uložen.

3. Kombinovaný režim: Pouze část zadání je zapsána ve vstupním souboru. Zbytek zadání se provede v dialogovém režimu. Toto je zřejmě nejlepší možnost, protože zadání funkcí, které v problému vystupují, se dá provést předem ve vhodném textovém editoru.

Druhá fáze je provedena UFO preprocesorem. Tento presprocesor je napsán v jazyce Fortran 77 a jeho výstupem je kontrolní program opět v jazyce Fortran 77. Tento postup má několik výhod. Fortran 77 (odpovídající normě ANSI) je dostatečně vysoký a přenositelný programovací jazyk. Navíc je tento jazyk vhodný pro numerické výpočty a mnoho podprogramů v tomto jazyce je již k dispozici. Další výhodou je, že program generovaný UFO preprocesorem pouze volá potřebné moduly a jeho vytvoření je velice snadné. Navíc se dají globální deklarace ovlivnit, aby odpovídaly velikosti problému a ušetřily se tím nároky na paměť. Velice snadné je také přidání a úprava implementovaných metod. Není nutné měnit celý systém, stačí upravit nebo přidat příslušné šablony a připravit nové moduly.

Ve *třetí fázi* je kontrolní program přeložen překladačem Fortran 77 a jsou k němu připojeny knihovní moduly.

Ve *čtvrté fázi* je přeložený program spuštěn a výsledky jsou dle požadavků zobrazeny. Uživatel může také dodat do kontrolního programu vlastní podprogramy, které mohou zpracovávat vstupní a výstupní data nestandardními způsoby.

B.3 Popis vybraných makroproměnných

Tato část obsahuje popis některých makroproměnných, které mohou přímo ovlivnit výpočet metodou vnitřního bodu. Kompletní seznam makroproměnných lze nalézt v [15].

`$KTERS` Určuje způsob výběru délky kroku. Pro `$KTERS=1` je délka kroku volena přesně (tj. jako minimum hodnotové funkce). `$KTERS=2` znamená výběr podle Goldsteinovy podmínky. Při `$KTERS=3` resp. `$KTERS=4` délka kroku splňuje Wolfeho resp. silnou Wolfeho podmínku. `$KTERS=5` výchozí volba a znamená volbu délky kroku podle Armijova pravidla. Pro `$KTERS=6` není zkracování délky kroku prováděno.

`$MEP` Tato volba ovlivňuje použití pokutové funkce. Možné hodnoty jsou $MEP=1, $MEP=2 a $MEP=3. Blíže viz. [15].

`$MEP3` Podle hodnoty této proměnné se vybírá metoda pro výpočet bariérového parametru μ. `$MEP3=1`: výpočet šablonou `UPSET3.I` - výpočet podle (1.37),

`$MEP3=3`: výpočet šablonou `UPSET5.I` - výpočet podle (1.36), `$MEP3=2` resp. `$MEP3=4`: výpočet šablonou `UPSET6.I` resp. `UPSET7.I` - výpočet podle [1].

`$MOS1` Určuje způsob kontroly přesnosti metody sdružených gradientů. Pro \$MOS1=0 je kontrola vypnuta a pro \$MOS1=1 je zapnuta a je použita základní volba pokutového parametru.

`$MOS2` Tato proměnná určuje, jak bude metoda sdružených gradientů předpodmíněna. Pro \$MOS2=0 je předpodmínění vypnuto. Pro přehled dalších voleb viz. [1].

`$MOS3` Proměnná \$MOS3 specifikuje zhlazení metody sdružených gradientů. Pro \$MOS3=0 je zhlazování vypnuto a pro \$MOS3=1 je zhlazování zapnuto.

`$TOLG`,`$TOLC` Těmito proměnnými lze nastavit kritéria pro ukončení iteračního procesu. Výchozí hodnota je 10^{-6}. Blíže viz. 2.5.3 Kritéria pro ukončení iteračního procesu.

`$MIT` Maximální počet iterací. Iterační proces se po překročení maximálního počtu iterací zastaví.

`$MFV` Maximální počet vyhodnocení účelové funkce. Iterační proces se po překročení maximálního počtu vyhodnocení zastaví.

`RPF1` Tato proměnná specifikuje výchozí hodnotu pokutového parametru, který se používá při výpočtu hodnoty pokutové funkce.

`RPF3` Tato konstanta reprezentuje bariérový parametr μ. V sekci `CONST` je možné nastavit výchozí hodnotu tohoto parametru.

Příloha C

Zdrojový kód

C.1 Výpisy klíčových částí šablon systému UFO pro metodu vnitřního bodu

Tato část obsahuje pouze vybrané šablony, které mají přímo něco společného s metodou vnitřního bodu. Systém UFO obsahuje na dva tisíce šablon, které implementují prakticky všechny používané metody pro optimalizaci funkcí.

U2FSI1

Jednou z nejdůležitějších šablon pro metodu vnitřního bodu je driver metody `U2FSI1`. Všechny matice používané v kódu jsou v paměti uloženy jako řídké.

```
$INTERFACE U2FSI1                                                99/12/01

  *
  *  Globalni deklarace jsou odstraneny z duvodu uspory mista
  *

  $MCL=NC
   MCL=NC
  $GN='GF'
  $SN='S'
  NDECF=0
  $SUBST('UFSET1')
  $SUBST('UO0FU1')
  $SUBST('UZLIND')
  $IF(NAL>0)
    $MAL=NAL
     MAL=NAL
    $JOB=1
```

```
    $IF(NORMA>0) $SUBST('UANRL3')
  $ENDIF
  $IF(NCL>0)
    $JOB=1
    $IF(NORMC>0) $SUBST('UCNRL3')
    $IF(NORMF>0) $SUBST('UCTRL3')
  $ENDIF
  $IF(MODELF='YES')
    $NUM=5
    $IF(KDF=1) $SUBST('UYPTSH')
  $ELSE
    $NUM=6
    $IF(KDC=1)
      $IF(KDA=1)
        $IF(NUMDER=2) $SUBST('UYPTSH')
      $ENDIF
    $ENDIF
  $ENDIF
  $JOB=NUMBER
  $HF=H
*
*     --------------------
*     SPARSE NEWTON METHOD
*     TEMPLATE : U2FSI1
*     --------------------
*
  $LMD=$10
  ASSIGN $20 TO IMD
  CALL UYPRO1('UXFU',2)
  CALL UYPRO3(0)
  CALL UXVINI(NF,$IX)
  MOT=0
  T=ZERO
  FO=FMIN
  IPOM1=0
  IF (ITERM.NE.0) GO TO $80
$L=$ 10 ; $LABEL
  $KD=1
  $LET='F'
  $SUBST('UP2LF$NUM')
  $IF(TYPE='F')
    CALL UPPRE1(NF,NX,NC,$X,$IX,$XL,$XU,$XSL,$XSU,$CF,$IC,$CL,$CU,$CSL&
,$CSU,FC)
```

```
$REM    FC=CMAX
  $ELSE
    CF(NC+1)=F$LET
  $ENDIF
  IF (TSS(NS).EQ.'UUXX') THEN
  ELSE IF (TSS(NS).EQ.'UXFU') THEN
  ELSE
  $IF(TYPE='L')
    $IF(MEP=2) $SUBST('UP0AF5')
    $SUBST('UP0BF1')
  $ELSEIF(TYPE='G')
  $ENDIF
  $IF(KDS<1)
  $ELSE
    IF (TSS(NS).EQ.'USXX') P=UXUDOT(NF,$G,$S,$IX,-KBF)
  $ENDIF
  ENDIF
  GO TO IMD
$L=$ 20 ; $LABEL
  $IF(TYPE='F')
    KS=1
    FS(1,1)=-ETA9
    FS(1,2)= MAX(ONE,TEN*FC)
  $ENDIF
$L=$ 30 ; $LABEL
  $SUBST('UYTS$(CONSTR)G')
  $SUBST('UO2FN4')
  $SUBST('UYFUT2')
  IF (GMAX.LE.TOLG.AND.CMAX.LE.TOLC) THEN
  ITERM=4
  TXFU='GRAD TOL'
  ENDIF
  IF (ITERM.NE.0) GO TO $80
$L=$ 40 ; $LABEL
  $SUBST('UUSSD1')
  IF (ITERM.NE.0) GO TO $80
  $IF(KDF=0)
    $IF(MCG<>0)
      $IF(MCG1=1)
        $SUBST('UYCP$(CON)S')
      $ENDIF
    $ENDIF
  $ENDIF
```

```
  $IF(KBF>0)
    CALL UXVINF(NF,M,$IX,$JH)
    CALL UXVINF(NF,MC,$IX,$JCG)
    $IF(MODELA='YES') CALL UXVINF(NF,MA,$IX,$JAG)
  $ENDIF
  $SUBST('UYTS$(CONSTR)H')
  $TYP=TYPE
  $IF(TYPE='F') $TYPE='L'
*
*     -----------------------
*     DIRECTION DETERMINATION
*     TEMPLATE : UDS$(TYPE)$(DECOMP)$(NUMBER)
*     -----------------------
*
  $SUBST('UDS$(TYPE)$(DECOMP)$(NUMBER)')
*
*     ------------------------------
*     END OF DIRECTION DETERMINATION
*     ------------------------------
*
  $SUBST('UD1T$(TYPE)3')
  CALL UCIMAX(NF,NX,NC,NCA,$X,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU,$CFD,$IC&
,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$ICA,$S,$GS(NF+1),$XO(NF+1),$GO(NF+1),&
RMAX,RMAXD)
  $IF(KBF>0)
    CALL UXVINE(M,$JH)
    CALL UXVINE(MC,$JCG)
    $IF(MODELA='YES') CALL UXVINE(MA,$JAG)
  $ENDIF
  $IF(TYP='L')
    $IF(MEP=2) $SUBST('UP0AF5')
    $SUBST('UP0BF1')
  $ELSEIF(TYP='G')
  $ENDIF
  IF (ITERM.NE.0) GO TO $80
  IF (IREST.NE.0) GO TO $40
  IF (NIT.EQ.LIT+1) KIT=NIT
$L=$ 50 ; $LABEL
  $SUBST('UYTR$(CONSTR)S')
  CALL UXVCOP(NC+1,$CF,$CFO)
$L=$ 60 ; $LABEL
  ASSIGN $60 TO IMD
  $SUBST('UZLINS')
```

```
  GOTO ($64,$62) ISB+1
$L=$ 62 ; $LABEL
  CALL UXUDIR(NF,R,$S,$XO,$X,$IX,-KBF)
  ISNC=1
  GOTO $10
$L=$ 64 ; $LABEL
  ISNC=2
  $IF(KDA=1) LD=0
  $IF(KDC=1) LD=0
  IF (ITERS.LE.0) THEN
  $IF(KDS<1)
    $SUBST('UYZER0')
  $ELSE
    $SUBST('UYZER1')
  $ENDIF
  CALL UXVCOP(NC+1,$CFO,$CF)
  IF (IDIR.EQ.0) CALL UYRES1(TSXX)
  CALL UYSET1
  GOTO $40
  ENDIF
  CALL UCIDIR(NF,NX,NC,NCA,$XO,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU,$CFD&
,$IC,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$ICA,$S,$GS(NF+1),$XO(NF+1),$GO(NF+1)&
,MIN(R,RMAXD))
  $IF(MEP3=1) $BPAR=3
  $IF(MEP3=2) $BPAR=6
  $IF(MEP3=3) $BPAR=5
  $IF(MEP3=4) $BPAR=7
  $SUBST('UPSET$BPAR')
  IF (KD.GT.LD) THEN
  ASSIGN $70 TO IMD
  GO TO $10
  ENDIF
$L=$ 70 ; $LABEL
  TXFU=TUXX
  TUXX='NEWTON   '
  $SUBST('UYTRED')
  GOTO $30
$L=$ 80 ; $LABEL
  IF(ITERM.LT.0) TXFU=TDXX
  CALL UYEPI1(2)
  $IF(NAL>0)
    $JOB=-1
    $IF(NORMA>0) $SUBST('UANRL3')
```

```
  $ENDIF
  $IF(NCL>0)
    $JOB=-1
    $IF(NORMF>0) $SUBST('UCTRL3')
    $IF(NORMC>0) $SUBST('UCNRL3')
  $ENDIF
  $TYPE=TYP
  $SUBST('UO1FN2')
  $ADD(INIT)
    NXFU='U2FSI1'
  $ENDADD
  $EXIT
```

UDSLI3

```
$INTERFACE UDSLI3                                           99/12/01

  $D='D'
  $XS='XS'
  $GS='GS'
  $XP='XP'
  $CON1='1.0$P-3'
  $CON2='5.0$P-1'
  $CON4='1.0$P-1'
  $CON5='1.0$P-3'
  $CON6='1.0$P 6'
  $POM1='U'
  $POM2=',$$IX,-KBF)'
  $HELP
    MAXIMUM NUMBER OF DENSE COLUMNS:
  $CHECK(MD,10,INT,2)
  $HELP
    MINIMUM NUMBER OF NONZERO ELEMENTS IN DENSE ROWS:
  $CHECK(MDE,50,INT,2)
  $HELP
    STRATEGY FOR PRECISION DETERMINATION:
      1 - SINGLE STRATEGY
      2 - GEOMETRIC DECREASING STRATEGY
      3 - HARMONIC  DECREASING STRATEGY
  $CHECK(MOS,3,'1\2\3',3)
  $IF(MEP=0) $MOS1=0
$REM  $IF(MEP=1) $MOS1=2
```

```
  $HELP
    TERMINATION, MATRIX CORRECTION AND
    CHOICE OF THE PENALTY PARAMETER
      0 - CLASSIC TERMINATION
      1 - DESCENT TERMINATION
      2 - EXTENDED CHOICE OF THE PENALTY PARAMETER
  $CHECK(MOS1,0,'0\1\2',4)
  $HELP
    PRECONDITIONING THE KARUSH-KUHN-TUCKER SYSTEM
           0 - PRECONDITIONING IS SUPPRESSED
     -1 OR 1 - INDEFIFITE PRECONDITIONING BASED ON A DIAGONAL
               APPROXIMATION OF THE HESSIAN MATRIX
    MOS2 > 0 - COMPLETE CHOLESKI DECOMPOSITION
    MOS2 < 0 - INCOMPLETE CHOLESKI DECOMPOSITION
  $CHECK(MOS2,-1,'-1\0\1',3)
  $ADD(REAL,'\$H($MMAX)')
  $ADD(INTEGER,'\$IH($NF+1)\$JH($MMAX)')
$REM  $IF(MEP=1) $MOS3=1
  $HELP
    TYPE OF SMOOTHING
      0 - NO SMOOTHING
      1 - SINGLE SMOOTHING
  $CHECK(MOS3,0,'0\1',4)
  $IF(MOS3=0)
    $XS=GS
    $XP=S
  $ENDIF
  $ADD(REAL,'\$CG($MC)\$CF($NC+1)\$CFD($NC+1)\$G($NF)&
  \$S($NF+$NCA)\$XO($NF+$NCA)\$GO($NF+$NCA)\$XS($NF+$NCA)&
  \$GS($NF+$NCA)\$XP($NF+$NCA)\$D($NF+$NCA)')
  $ADD(INTEGER,'\$ICG($NC+1)\$JCG($MC)\$IX($NX)')
  $ADD(REAL,'\UXVDOT\UX$(POM1)DOT\UCIDOT\UCIPEN')
  $ADD(REAL,'\ALF\RHO\RHO1\RHO2\SIG\SIG1\SIG2\BTR\BTB\RMU\TEMP\TEMP2')
  $ADD(INTEGER,'\I\K\L\MM\MMX\IPOM\KPOM\NOS')
  $ADD(LOGICAL,'\L1')
  $HELP
    METHOD SELECTION:
      1 - BISECTION
      2 - TWO POINT QUADRATIC INTERPOLATION
      3 - THREE POINT QUADRATIC INTERPOLATION
      4 - THREE POINT CUBIC INTERPOLATION
  $CHECK(MES,1,'1\2\3\4',3)
  $IF(MEP=1)
```

```
  $ELSEIF(MEP=2)
    $HELP
      PENALTY COEFFICIENT
    $CHECK(RPF1,'1.5$P 0',REAL,3)
  $ELSEIF(MEP=3)
  $ENDIF
  $DEFAULT='0.9$P 0'
  $IF (MOS=1) $DEFAULT='0.1$P 0'
  $HELP
    MINIMUM PRECISION OF AN INEXACT SOLUTION
  $CHECK(EPS6,DEFAULT,REAL,4)
  $DEFAULT='ETA0*1.0D-3'
$REM  $IF(MEP=3) $DEFAULT='1.0$P-4'
  $HELP
    TOLERANCE FOR POSITIVE DEFINITENESS
  $CHECK(ETA2,DEFAULT,REAL,4)
  $IF(MD>0)
    $BD='BD'
    $ADD(REAL,'\$BD($MD*($MD+1)/2)')
  $ENDIF
  CALL UOD1D1
  IPOM=0
  GNORM=SQRT(UX$(POM1)DOT(NF,$G,$G$POM2)
  CALL UXVNEG(NF,$G,$XS)
*
*     DETERMINATION OF ACTIVE CONSTRAINTS AND CONSTRUCTION OF THE
*     CORRESPONDING KARUSH-KUHN-TUCKER SYSTEM
*
  M=$IH(NF+1)-1
  CALL UCIACT(NF,NX,NC,NCA,$X,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU,$CF,$IC&
,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$ICA,TEMP,INF)
  CALL UCINAT(NF,NX,NC,MC,M,$X,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU,$CF,$IC&
,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$CG,$ICG,$JCG,$XS,$GO,$H,$IH,$JH,RPF3)
  CALL UCISEC(NF,NX,NC,NCA,$X,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU,$CF,$IC&
,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$ICA,$XS(NF+1),$D(NF+1),RPF3)
  CALL UODLI3(NF,NX,NC,NCA,$IX,$IC,$ICA,$XS,RPF2,RPF3,INF)
*
*     CONSTRUCTION OF THE CG PRECONDITIONER
*
  $LII=$90
  $SUBST('UDSLII')
  MM=NF+NCA
  RHO1=UX$(POM1)DOT(NF,$XS,$XS$POM2
```

```
      SIG1=UXVDOT(NCA,$XS(NF+1),$XS(NF+1))
      INITD=MAX(ABS(INITD),1)
      PAR=MIN(EPS6,SQRT(RHO1+SIG1))
      IF (PAR.GT.TEN*$CON1) THEN
      $IF(MOS=1)
      $ELSEIF(MOS=2)
        PAR=MIN(PAR,$CON1**($DBLE(NIT-LIT)/$DBLE(MM)))
      $ELSEIF(MOS=3)
        PAR=MIN(PAR,ONE/$DBLE(NIT-LIT))
      $ENDIF
      ENDIF
      PAR=PAR*PAR
*
*     CG INITIATION
*
      CALL UXVSET(MM,ZERO,$S)
      $IF(MOS3=0)
      $ELSE
        CALL UXVSET(MM,ZERO,$XP)
        CALL UXVCOP(MM,$XS,$GS)
      $ENDIF
      $IF(NOS2=0)
        CALL UXVCOP(MM,$XS,$XO)
      $ENDIF
*
*     CG PRECONDITIONING
*
      $INPF=$XS
      $OUTF=$XO
      $SUBST('UDSLIP')
      RHO=UX$(POM1)DOT(NF,$INPF,$OUTF$POM2
      SIG=UXVDOT(NCA,$INPF(NF+1),$OUTF(NF+1))
      KPOM=0
      MMX=MM+3
*
*    CG ITERATIONS
*
      DO $15 NRED=1,MMX
      IF (RHO+SIG.EQ.ZERO) THEN
      ITERD=-4
      TDXX='CG BREAK'
      CALL UOERR1('UDSLI3',4)
      GO TO $90
```

```
  ENDIF
  CALL UXSSMM(NF,M,$H,$IH,$JH,$XO,$GO)
  CALL UCICMD(NF,NC,NCA,MC,$ICA,$CG,$ICG,$JCG,$XO(NF+1),ONE,$GO,$GO)
  CALL UXVMUL(NCA,$D(NF+1),$XO(NF+1),$GO(NF+1),1)
  CALL UCIRMD(NF,NC,NCA,MC,$ICA,$CG,$ICG,$JCG,$XO,ONE,$GO(NF+1)&
,$GO(NF+1))
  ALF=UX$(POM1)DOT(NF,$XO,$GO$POM2+UXVDOT(NCA,$XO(NF+1),$GO(NF+1))
  IF (ALF.EQ.ZERO) THEN
  ITERD=-5
  TDXX='CG BREAK'
  CALL UOERR1('UDSLI3',5)
  GO TO $90
  ENDIF
*
*     CG STEP
*
  ALF=(RHO+SIG)/ALF
  TDXX='CG STEP'
  CALL UX$(POM1)DIR(NF, ALF,$XO,$XP,$XP$POM2
  CALL UXVDIR(NCA, ALF,$XO(NF+1),$XP(NF+1),$XP(NF+1))
  CALL UX$(POM1)DIR(NF,-ALF,$GO,$XS,$XS$POM2
  CALL UXVDIR(NCA,-ALF,$GO(NF+1),$XS(NF+1),$XS(NF+1))
*
*     CG SMOOTHING
*
  $IF(MOS3=0)
  $ELSE
    RMU=ONE/ETA9
    CALL UX$(POM1)DIF(NF,$GS,$XS,$GO$POM2
    CALL UXVDIF(NCA,$GS(NF+1),$XS(NF+1),$GO(NF+1))
    BTB=UX$(POM1)DOT(NF,$GO,$GO$POM2+UXVDOT(NCA,$GO(NF+1),$GO(NF+1))
    BTR=UX$(POM1)DOT(NF,$GO,$XS$POM2+UXVDOT(NCA,$GO(NF+1),$XS(NF+1))
    RMU=-BTR/MAX(BTB,RMU)
    CALL UX$(POM1)DIR(NF,RMU,$GO,$XS,$GS$POM2
    CALL UXVDIR(NCA,RMU,$GO(NF+1),$XS(NF+1),$GS(NF+1))
    CALL UX$(POM1)DIF(NF,$S,$XP,$GO$POM2
    CALL UXVDIF(NCA,$S(NF+1),$XP(NF+1),$GO(NF+1))
    CALL UX$(POM1)DIR(NF,RMU,$GO,$XP,$S$POM2
    CALL UXVDIR(NCA,RMU,$GO(NF+1),$XP(NF+1),$S(NF+1))
  $ENDIF
  TDXX='CG STEP '
*
*     TERMINATION CRITERION
```

```
*
  L1=NRED.GE.MMX
  IF (L1) TDXX='IT LIMIT'
  SIG2=UXVDOT(NCA,$GS(NF+1),$GS(NF+1))
  IF (SIG2.LE.PAR*SIG1) THEN
  RHO2=UX$(POM1)DOT(NF,$GS,$GS$POM2
  IF (RHO2.LE.PAR*RHO1) L1=.TRUE.
  ENDIF
  IF (KOUT.GT.2) RHO2=UX$(POM1)DOT(NF,$GS,$GS$POM2
*
*     TEST FOR DESCENT DIRECTION
*
  $LIT=$16
  $SUBST('UDSLIT')
*
*     CG PRECONDITIONING
*
  $INPF=XS
  $IF(MOS2=0)
    $OUTF=XS
  $ELSE
    $OUTF=GO
  $ENDIF
  $SUBST('UDSLIP')
  RHO2=UX$(POM1)DOT(NF,$XS,$OUTF$POM2
  SIG2=UXVDOT(NCA,$XS(NF+1),$OUTF(NF+1))
  ALF=(RHO2+SIG2)/(RHO+SIG)
  CALL UX$(POM1)DIR(NF,ALF,$XO,$OUTF,$XO$POM2
  CALL UXVDIR(NCA,ALF,$XO(NF+1),$OUTF(NF+1),$XO(NF+1))
  RHO=RHO2
  SIG=SIG2
$15 CONTINUE
  TDXX='IT LIMIT'
*
*     AN INEXACT SOLUTION IS OBTAINED
*
$16 CONTINUE
*
*     COMPUTATION OF THE DIRECTIONAL DERIVATIVE
*
  $SUBST('UDSLID')
  CALL UODLI4(NF,NX,NC,NCA,$D,$S(NF+1),$XO,$GO,$GS(NF+1))
$90 CONTINUE
```

```
CALL UOD1D2(NF,$G,$S)
$ADD(INIT)
  NDXX='UDSLI3'
$ENDADD
$EXIT
```

UCIACT

Šablona `UCIACT` určuje aktivní omezení.

```
* SUBROUTINE UCIACT             ALL SYSTEMS                 98/12/01
C PORTABILITY : ALL SYSTEMS
C 98/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* DETERMINATION OF ACTIVE CONSTRAINTS.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NX)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NX)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NX)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NX)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CF(NC)  VECTOR CONTAINING VALUES OF THE CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  IO  ICA(NX+NC)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
*  RO  CDOT  SQUARED NORM OF CONSTRAINT VIOLATIONS.
*  IO  INF  CHANGE OF ACTIVE CONSTRAINT INDICATOR. INF=0-NO CHANGE.
*          INF=1-CHANGE OF ACTIVE CONSTRAINTS.
```

```
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*          KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*          CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*          SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UCIACT(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CF,IC,
     & CL,CU,CSL,CSU,CZL,CZU,ICA,CDOT,INF)
      $INCLUDE ('UMCOMM')
      INTEGER NF,NX,NC,NCA,IX(NX),IC(NC),ICA(NX+NC),INF
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & CF(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),CDOT
      INTEGER I,KC,KCA
      $FLOAT CON
      PARAMETER(CON=1.0$P-1)
      INF=0
      CDOT=ZERO
      KCA=NCA
      NCA=0
      IF (KBF.GT.0) THEN
      DO 1 I=1,NX
      IX(I)=ABS(IX(I))
      IF (IX(I).EQ.0) THEN
      ELSE IF (IX(I).EQ.1) THEN
      CDOT=CDOT+(XL(I)-X(I)+XSL(I))**2
      IF (XSL(I).LE.CON*ABS(XZL(I))) THEN
      IX(I)=-1
      NCA=NCA+1
      IF (ICA(NCA).NE.-I) THEN
      ICA(NCA)=-I
      INF=1
      ENDIF
      ENDIF
      ELSE IF (IX(I).EQ.2) THEN
      CDOT=CDOT+(X(I)-XU(I)+XSU(I))**2
      IF (XSU(I).LE.CON*ABS(XZU(I))) THEN
      IX(I)=-2
      NCA=NCA+1
      IF (ICA(NCA).NE.-I) THEN
      ICA(NCA)=-I
      INF=1
```

```
      ENDIF
      ENDIF
      ELSE IF (IX(I).EQ.3.OR.IX(I).EQ.4) THEN
      CDOT=CDOT+(XL(I)-X(I)+XSL(I))**2+(X(I)-XU(I)+XSU(I))**2
      IF (ABS(XSL(I)*XZU(I)).LE.ABS(XSU(I)*XZL(I))) THEN
      IF (XSL(I).LE.CON*ABS(XZL(I))) THEN
      IX(I)=-3
      NCA=NCA+1
      IF (ICA(NCA).NE.-I) THEN
      ICA(NCA)=-I
      INF=1
      ENDIF
      ENDIF
      ELSE
      IF (XSU(I).LE.CON*ABS(XZU(I))) THEN
      IX(I)=-4
      NCA=NCA+1
      IF (ICA(NCA).NE.-I) THEN
      ICA(NCA)=-I
      INF=1
      ENDIF
      ENDIF
      ENDIF
      ELSE IF (IX(I).EQ.5) THEN
      IX(I)=-5
      ENDIF
    1 CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
      DO 2 KC=1,NC
      IC(KC)=ABS(IC(KC))
      IF (IC(KC).EQ.0) THEN
      ELSE IF (IC(KC).EQ.1) THEN
      CDOT=CDOT+(CL(KC)-CF(KC)+CSL(KC))**2
      IF (CSL(KC).LE.CON*ABS(CZL(KC))) THEN
      IC(KC)=-1
      NCA=NCA+1
      IF (ICA(NCA).NE.KC) THEN
      ICA(NCA)=KC
      INF=1
      ENDIF
      ENDIF
      ELSE IF (IC(KC).EQ.2) THEN
```

```
      CDOT=CDOT+(CF(KC)-CU(KC)+CSU(KC))**2
      IF (CSU(KC).LE.CON*ABS(CZU(KC))) THEN
      IC(KC)=-2
      NCA=NCA+1
      IF (ICA(NCA).NE.KC) THEN
      ICA(NCA)=KC
      INF=1
      ENDIF
      ENDIF
      ELSE IF (IC(KC).EQ.3.OR.IC(KC).EQ.4) THEN
      CDOT=CDOT+(CL(KC)-CF(KC)+CSL(KC))**2+(CF(KC)-CU(KC)+CSU(KC))**2
      IF (ABS(CSL(KC)*CZU(KC)).LE.ABS(CSU(KC)*CZL(KC))) THEN
      IF (CSL(KC).LE.CON*ABS(CZL(KC))) THEN
      IC(KC)=-3
      NCA=NCA+1
      IF (ICA(NCA).NE.KC) THEN
      ICA(NCA)=KC
      INF=1
      ENDIF
      ENDIF
      ELSE
      IF (CSU(KC).LE.CON*ABS(CZU(KC))) THEN
      IC(KC)=-4
      NCA=NCA+1
      IF (ICA(NCA).NE.KC) THEN
      ICA(NCA)=KC
      INF=1
      ENDIF
      ENDIF
      ENDIF
      ELSE IF (IC(KC).EQ.5) THEN
      CDOT=CDOT+CF(KC)**2
      IC(KC)=-5
      NCA=NCA+1
      IF (ICA(NCA).NE.KC) THEN
      ICA(NCA)=KC
      INF=1
      ENDIF
      ENDIF
    2 CONTINUE
      ENDIF
      IF (NCA.NE.KCA) INF=1
      RETURN
```

```
      END
```

UCINAT

```
* SUBROUTINE UCINAT             ALL SYSTEMS                   99/12/01
C PORTABILITY : ALL SYSTEMS
C 99/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* GRADIENT PART OF RIGHT HAND SIDE VECTOR.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CF(NC)  VECTOR CONTAINING VALUES OF THE CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  II  CG(MC)  ELEMENTS OF THE CONSTRAINT JACOBIAN MATRIX.
*  II  ICG(NC+1)  POSITIONS OF THE FIRST ROWS ELEMENTS IN THE SPARSE
*         JACOBIAN STRUCTURE.
*  II  JCG(MC)  COLUMN INDICES OF ELEMENTS IN THE SPARSE JACOBIAN
*         STRUCTURE.
*  RU  G(NF)  RIGHT HAND SIDE VECTOR.
*  RA  GO(NF)  AUXILIARY ARRAY.
*  RU  H(M)  SPARSE HESSIAN MATRIX OF THE LAGRANGIAN FUNCTION.
*  II  IH(NF+1) POINTERS OF DIAGONAL ELEMENTS OF THE SPARSE HESSIAN
*         MATRIX.
*  II  JH(M)  COLUMN INDICES OF NONZERO ELEMENTS OF THE SPARSE HESSIAN
```

```
*         MATRIX.
*  RI  RPF3  BARRIER COEFFICIENT.
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*         KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UCINAT(NF,NX,NC,MC,M,X,IX,XL,XU,XSL,XSU,XZL,XZU,CF,IC,
     & CL,CU,CSL,CSU,CZL,CZU,CG,ICG,JCG,G,GO,H,IH,JH,RPF3)
      $INCLUDE ('UMCOMM')
      INTEGER NF,NX,NC,MC,M,IX(NX),IC(NC),ICG(NC+1),JCG(MC),IH(NF+1),
     & JH(M)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & CF(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),CG(MC),
     & G(NF),GO(NF),H(M),RPF3
      INTEGER I,K,KK,KC
      $FLOAT TEMP,TEMP1
      IF (KBF.GT.0) THEN
      DO 10 I=1,NX
      KK=IX(I)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      H(IH(I))=H(IH(I))+XZL(I)/XSL(I)
      G(I)=G(I)+(RPF3+XZL(I)*(XL(I)-X(I)))/XSL(I)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      H(IH(I))=H(IH(I))+XZU(I)/XSU(I)
      G(I)=G(I)-(RPF3+XZU(I)*(X(I)-XU(I)))/XSU(I)
      ENDIF
   10 CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
      DO 20 KC=1,NC
      KK=IC(KC)
      TEMP=ZERO
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP=CZL(KC)/CSL(KC)
      TEMP1= (RPF3+CZL(KC)*(CL(KC)-CF(KC)))/CSL(KC)
      IF (TEMP.NE.ZERO) THEN
      K=ICG(KC)
      CALL UALNG3(NF,NC,MC,KC,X,CG,ICG,JCG,GO,ISNC,0)
```

```
      CALL UASSH3(NF,NC,MC,M,H,IH,JH,ICG,JCG,GO,KC,TEMP)
      CALL UXWDIR(ICG(KC+1)-K,NF,JCG(K),TEMP1,CG(K),G,G,1)
      ENDIF
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      TEMP=CZU(KC)/CSU(KC)
      TEMP1=-(RPF3+CZU(KC)*(CF(KC)-CU(KC)))/CSU(KC)
      IF (TEMP.NE.ZERO) THEN
      K=ICG(KC)
      CALL UALNG3(NF,NC,MC,KC,X,CG,ICG,JCG,GO,ISNC,0)
      CALL UASSH3(NF,NC,MC,M,H,IH,JH,ICG,JCG,GO,KC,TEMP)
      CALL UXWDIR(ICG(KC+1)-K,NF,JCG(K),TEMP1,CG(K),G,G,1)
      ENDIF
      ENDIF
   20 CONTINUE
      ENDIF
      RETURN
      END
```

UCISEC

```
* SUBROUTINE UCISEC             ALL SYSTEMS                 99/12/01
C PORTABILITY : ALL SYSTEMS
C 99/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* ACTIVE CONSTRAINT PART OF RIGHT HAND SIDE VECTOR.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CF(NF)  VECTOR CONTAINING VALUES OF THE CONSTRAINT FUNCTIONS.
```

```
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  IO  ICA(NCA)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
*  RO  CP(NCA)  RIGHT HAND SIDE VECTOR.
*  RO  D(NCA)  DIAGONAL SOUTH-EAST CORNER.
*  RI  RPF3  BARRIER COEFFICIENT.
*
      SUBROUTINE UCISEC(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CF,IC,
     & CL,CU,CSL,CSU,CZL,CZU,ICA,CP,D,RPF3)
      INTEGER NF,NX,NC,NCA,IX(NX),IC(NC),ICA(NCA)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),CF(NC),
     & CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),CP(NCA),D(NCA),
     & RPF3
      INTEGER I,K
      DO 1 I=1,NCA
      K=ICA(I)
      IF (K.LT.0) THEN
      K=-K
      IF (IX(K).EQ.-1.OR.IX(K).EQ.-3) THEN
      CP(I)=XL(K)-X(K)+RPF3/XZL(K)
      D(I)=-XSL(K)/XZL(K)
      ELSE IF (IX(K).EQ.-2.OR.IX(K).EQ.-4) THEN
      CP(I)=XU(K)-X(K)-RPF3/XZU(K)
      D(I)=-XSU(K)/XZU(K)
      ENDIF
      ELSE
      IF (IC(K).EQ.-1.OR.IC(K).EQ.-3) THEN
      CP(I)=CL(K)-CF(K)+RPF3/CZL(K)
      D(I)=-CSL(K)/CZL(K)
      ELSE IF (IC(K).EQ.-2.OR.IC(K).EQ.-4) THEN
      CP(I)=CU(K)-CF(K)-RPF3/CZU(K)
      D(I)=-CSU(K)/CZU(K)
      ELSE IF (IC(K).EQ.-5) THEN
      CP(I)=-CF(K)
      D(I)=0.0$P 0
      ENDIF
      ENDIF
    1 CONTINUE
```

```
    RETURN
    END
```

UP1LF5

```
$INTERFACE UP1LF5                                                    99/12/01

  $SUBST('UZCSET')
  $HELP
    TYPE OF EXTREMUM:
       0 - MINIMUM
       1 - MAXIMUM
  $CHECK(IEXT,0,'0\1',1)
  $HELP
    LOWER BOUND FOR FUNCTION VALUE
  $CHECK(FMIN,'-$ETA9',REAL,2)
  $ADD(REAL,'\$X($NF)\$XL($NX)\$XU($NX)\$XSL($NX)\$XSU($NX)\$XZL($NX)&
  \$XZU($NX)\$GF($NF)\$CL($NC)\$CU($NC)\$CSL($NC)\$CSU($NC)\$CZL($NC)&
  \$CZU($NC)\$GC($NF)\$G($NF)')
$REM  $ADD(REAL,'\FF\FC\F\CMAX')
  $ADD(INTEGER,'\$IX($NX)\$IC($NC)\$ICG($NC+1)\$JCG($MC)')
$REM  $ADD(INTEGER,'\NF\NX\NC\MC\KC')
$CHPOM='$$'
$IF(.FALSE.)
*IF(.TRUE.)
*CHPOM='*'
*OPTION(CHDIR='$')
$ENDIF
*
*     ---------------------------
*     PENALTY FUNCTION DEFINITION
*     ---------------------------
*
$OPTION(CHDIR=CHPOM)
 $00  CALL UP1LF5(NF,NX,NC,MC,KC,$X,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU&
,$GF,$IC,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$GC,$ICG,$JCG,$G,FF,FC,F,CMAX)
      GOTO ($90,$10,$20,$50,$60) ISB+1
 $10  CONTINUE
      $MTYPE=FMODF
      $SUBST('UZFMOD')
      GO TO $00
 $20  CONTINUE
```

```
      $IF(GMODF='G')
        $MTYPE='G'
        $SUBST('UZFMOD')
        GO TO $00
      $ELSEIF(FMODF='F')
        $HELP
          METHOD FOR DETERMINATION OF DIFFERENCE USED FOR
          NUMERICAL COMPUTATION OF GRADIENT:
            0 - CONSTANT DIFFERENCE
            1 - OPTIMAL DIFFERENCE COMPUTED FROM
                HESSIAN MATRIX APPROXIMATION
            2 - OPTIMAL DIFFERENCE COMPUTED FROM
                NUMERICALLY ESTIMATED SECOND ORDER
                DERIVATIVES
        $CHECK(MCG,2,'0\1\2',3)
        $IF(MCG=0)
          $SUBST('UFOG$(CON)1')
        $ELSE
          $SUBST('UFOG$(CON)2')
        $ENDIF
        GOTO ($40,$30) ISB+1
$30   CONTINUE
        $MTYPE=FMODF
        $SUBST('UZFMOD')
        GO TO $20
$40   CONTINUE
        GO TO $00
      $ELSE
        IF (LD.LT.0) GO TO $00
        $MTYPE=FMODF
        $SUBST('UZFMOD')
        GO TO $00
      $ENDIF
$50   CONTINUE
      $IF(SAVC>-1)
        IF(LDC.GE.0) THEN
        FC=$CF(KC)
        GO TO $00
        ENDIF
      $ENDIF
      $IF(NCL>0)
        IF(KC.LE.NCL) THEN
        $SUBST('UCLNF3')
```

```
      $IF(KBC>0) $SUBST('UCTRF1')
      $IF(ISNC>0)
        $SUBST('UCSVF1')
      $ENDIF
      GO TO $00
      ENDIF
    $ENDIF
    $IF(LCD<0)
      $IF(NCL<NC)
        $MTYPE=FMODC
        $SUBST('UZCMOD')
        $IF(NORMC>0) $SUBST('UCNRF1')
        $IF(KBC>0) $SUBST('UCTRF1')
        $IF(ISNC>0)
          $SUBST('UCSVF1')
        $ENDIF
        $IF(FMODC<>'F')
          $IF(NORMC>0) $SUBST('UCNRG1')
          $IF(ISNC>1)
            $SUBST('UCSVG3')
          $ENDIF
        $ENDIF
        GO TO $00
      $ENDIF
    $ELSE
      $GO TO $00
    $ENDIF
$60 CONTINUE
    $IF(SAVC>0)
      IF(LDC.GE.1) THEN
      $SUBST('UCLNG3')
      GO TO $00
      ENDIF
    $ENDIF
    $IF(NCL>0)
      IF(KC.LE.NCL) THEN
      $SUBST('UCLNG3')
      GO TO $00
      ENDIF
    $ENDIF
    $IF(LCD<1)
      $IF(NCL<NC)
        $IF(GMODC='G')
```

```
            $MTYPE='G'
            $SUBST('UZCMOD')
            $IF(NORMC>0) $SUBST('UCNRG1')
            $IF(ISNC>1)
              $SUBST('UCSVG3')
            $ENDIF
            GOTO $00
          $ELSEIF(FMODC='F')
            $IF(ISNC>0)
              IF (LD.GE.0) FC=$CF(KC)
            $ENDIF
 $65  CONTINUE
            $SUBST('UC0G$(CON)3')
            GOTO ($80,$70) ISB+1
 $70  CONTINUE
            $MTYPE=FMODC
            $SUBST('UZCMOD')
            $IF(NORMC>0) $SUBST('UCNRF1')
            $IF(KBC>0) $SUBST('UCTRF1')
            GO TO $65
 $80  CONTINUE
            $IF(ISNC>1)
              $SUBST('UCSVG3')
            $ENDIF
            GO TO $00
          $ELSE
            IF (LD.LT.0) GOTO $00
            $MTYPE=FMODC
            $SUBST('UZCMOD')
            $IF(NORMC>0) $SUBST('UCNRG1')
            $IF(ISNC>1)
              $SUBST('UCSVG3')
            $ENDIF
            GO TO $00
          $ENDIF
        $ENDIF
      $ELSE
        $GO TO $00
      $ENDIF
 $90  CONTINUE
$CHPOM='$$'
$IF(.FALSE.)
*IF(.TRUE.)
```

```
*CHPOM='*'
*OPTION(CHDIR='$')
$ENDIF
*
*     ----------------------------------
*     END OF PENALTY FUNCTION DEFINITION
*     ----------------------------------
*
$OPTION(CHDIR=CHPOM)
      $ADD(INIT)
        NPFX='UP1LF5'
      $ENDADD
      $EXIT

* IMPLEMENTATION

* SUBROUTINE UP1LF5             ALL SYSTEMS                 99/12/01
C PORTABILITY : ALL SYSTEMS
C 99/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* COMPUTATION OF THE VALUE AND THE GRADIENT OF THE OBJECTIVE FUNCTION.
* COMPUTATION OF THE VALUES AND THE GRADIENTS OF THE CONSTRAINT
* FUNCTIONS.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  MC  NUMBER OF NONZERO ELEMENTS OF THE SPARSE JACOBIAN MATRIX.
*  IU  KC  INDEX OF THE CONSTRAINT.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NX)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NX)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NX)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NX)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NX)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  GF(NF)  GRADIENT OF THE MODEL FUNCTION.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
```

```
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  RI  GC(NF)  GRADIENT OF THE SELECTED FUNCTION.
*  II  ICG(NC+1)  POSITIONS OF THE FIRST ROWS ELEMENTS IN THE SPARSE
*         JACOBIAN STRUCTURE.
*  II  JCG(MC)  COLUMN INDICES OF ELEMENTS IN THE SPARSE JACOBIAN
*         STRUCTURE.
*  RO  G(NF)  GRADIENT OF THE LAGRANGIAN FUNCTION.
*  RI  FF  VALUE OF THE MODEL FUNCTION.
*  RO  FC  VALUE OF THE PENALTY TERM.
*  RO  F  VALUE OF THE OBJECTIVE FUNCTION.
*  RO  CMAX  MAXIMUM CONSTRAINT VIOLATION.
*
* COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*         KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*  II  KD  DEGREE OF REQUIRED DERIVATIVES.
*  II  LD  DEGREE OF PREVIOUSLY COMPUTED DERIVATIVES.
*  II  IEXT  TYPE OF EXTREMUM. IEXT=0-MINIMUM. IEXT=1-MAXIMUM.
*
* STATUS VARIABLES :
*  NS,ISP,TSS
*
* SUBPROGRAMS USED :
*  S   UYPRO1  PROLOGUE.
*  S   UYEPI1  EPILOGUE.
*  S   UYSET0  STATUS DEFINITION.
*  S   UXVCOP  COPYING OF A VECTOR.
*  S   UXVNEG  COPYING OF A VECTOR WITH CHANGE OF THE SIGN.
*
      SUBROUTINE UP1LF5(NF,NX,NC,MC,KC,X,IX,XL,XU,XSL,XSU,XZL,XZU,GF,
     & IC,CL,CU,CSL,CSU,CZL,CZU,GC,ICG,JCG,G,FF,FC,F,CMAX)
      $INCLUDE ('UMCOMM')
      INTEGER NF,NX,NC,MC,KC,IX(NX),IC(NC),ICG(NC+1),JCG(MC)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & GF(NF),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),GC(NF),
     & G(NF),FF,FC,F,CMAX
      INTEGER I,J,K,KK,L,JP
```

```
      $FLOAT TEMP,CON
      PARAMETER(CON=1.0$P-1)
      SAVE KK
      IF (TSS(NS).NE.'UFXX') CALL UYPRO1('UFXX',-1)
      GO TO (1,2,4,10,12) ISP(NS)
    1 CONTINUE
      IF (KD.LE.LD) GO TO 15
      IF (LD.GE.0) GO TO 3
      CALL UYSET0(1,2)
      RETURN
    2 CONTINUE
      IF (IEXT.LE.0) THEN
      F= FF
      ELSE
      F=-FF
      ENDIF
      CMAX=ZERO
    3 IF (KD.LT.1) GO TO 5
      IF (LD.GE.1) GO TO 5
      CALL UYSET0(2,3)
      RETURN
    4 CONTINUE
      IF (IEXT.LE.0) THEN
      CALL UXVCOP(NF,GF,G)
      ELSE
      CALL UXVNEG(NF,GF,G)
      ENDIF
    5 IF (KD.GE.0.AND.LD.LT.0) FF=F
C      IF (KD.GE.1.AND.LD.LT.1) CALL UXVCOP(NF,G,GF)
      IF (KBF.LE.0) GO TO 8
      IF (LD.LT.0) THEN
      DO 6 I=1,NX
      KK=ABS(IX(I))
      IF (KK.EQ.1.OR.KK.EQ.3.OR.KK.EQ.4) THEN
      IF (NIT.LE.LIT) THEN
      XSL(I)=MAX(CON,X(I)-XL(I))
      XZL(I)=CON
      ENDIF
      TEMP=XL(I)-X(I)
      F=F+XZL(I)*(TEMP+XSL(I))
      CMAX=MAX(CMAX,TEMP)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.3.OR.KK.EQ.4) THEN
```

```
      IF (NIT.LE.LIT) THEN
      XSU(I)=MAX(CON,XU(I)-X(I))
      XZU(I)=CON
      ENDIF
      TEMP=X(I)-XU(I)
      F=F+XZU(I)*(TEMP+XSU(I))
      CMAX=MAX(CMAX,TEMP)
      ENDIF
    6 CONTINUE
      ENDIF
      IF (KD.GE.1.AND.LD.LT.1) THEN
      DO 7 I=1,NX
      KK=ABS(IX(I))
      IF (KK.EQ.1.OR.KK.EQ.3.OR.KK.EQ.4) THEN
      G(I)=G(I)-XZL(I)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.3.OR.KK.EQ.4) THEN
      G(I)=G(I)+XZU(I)
      ENDIF
    7 CONTINUE
      ENDIF
    8 IF (KBC.LE.0) GO TO 15
      KC=1
    9 IF (KC.GT.NC) GO TO 15
      KK=ABS(IC(KC))
      IF (KK.EQ.0) GO TO 14
      IF (LD.GE.0) GO TO 11
      CALL UYSET0(3,4)
      RETURN
   10 CONTINUE
      IF (KK.EQ.1.OR.KK.EQ.3.OR.KK.EQ.4) THEN
      IF (NIT.LE.LIT) THEN
      CSL(KC)=MAX(CON,FC-CL(KC))
      CZL(KC)=CON
      ENDIF
      TEMP=CL(KC)-FC
      F=F+CZL(KC)*(TEMP+CSL(KC))
      CMAX=MAX(CMAX,TEMP)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.3.OR.KK.EQ.4) THEN
      IF (NIT.LE.LIT) THEN
      CSU(KC)=MAX(CON,CU(KC)-FC)
      CZU(KC)=CON
```

```
      ENDIF
      TEMP=FC-CU(KC)
      F=F+CZU(KC)*(TEMP+CSU(KC))
      CMAX=MAX(CMAX,TEMP)
      ENDIF
      IF (KK.EQ.5) THEN
      IF (NIT.LE.LIT) THEN
      CZL(KC)=ZERO
      ENDIF
      F=F+CZL(KC)*FC
      CMAX=MAX(CMAX,ABS(FC))
      ENDIF
   11 IF (KD.LT.1) GO TO 14
      IF (LD.GE.1) GO TO 14
      CALL UYSET0(4,5)
      RETURN
   12 CONTINUE
      IF (KK.EQ.3.OR.KK.EQ.4) THEN
      TEMP= CZU(KC)-CZL(KC)
      ELSE IF (KK.EQ.1) THEN
      TEMP=-CZL(KC)
      ELSE IF (KK.EQ.2) THEN
      TEMP= CZU(KC)
      ELSE IF (KK.EQ.5) THEN
      TEMP= CZL(KC)
      ENDIF
      K=ICG(KC)
      L=ICG(KC+1)-K
      DO 13 J=1,L
      JP=JCG(K)
      G(JP)=G(JP)+TEMP*GC(JP)
      K=K+1
   13 CONTINUE
   14 KC=KC+1
      GO TO 9
   15 CONTINUE
   16 CALL UYEPI1(-1)
      LD=KD
      RETURN
      END
```

UP0LF5

```
$INTERFACE UP0AF5                                                    99/12/01

 CALL UP0AF5(NF,NX,NC,NCA,$X,$IX,$XL,$XU,$XSL,$XSU,$XZL,$XZU,$CF,
&$CFD,$IC,$CL,$CU,$CSL,$CSU,$CZL,$CZU,$ICA,$S,$GS(NF+1),$XO(NF+1),
&$GO(NF+1),R,RPF2,FC,F)
$ADD(REAL,'\$X($NF)\$XL($NX)\$XU($NX)\$XSL($NX)\$XSU($NX)\$XZL($NX)&
\$XZU($NX)\$CF($NC+1)\$CFD($NC)\$CL($NC)\$CU($NC)\$CSL($NC)\$CSU($NC)&
\$CZL($NC)\$CZU($NC)\S($NF+$NCA)\GS($NF+$NCA)\XO($NF+$NCA)&
\GO($NF+$NCA)')
$ADD(INTEGER,'\$IX($NX)\$IC($NC)\$ICA($NCA)')

$EXIT

* IMPLEMENTATION

* SUBROUTINE UP0AF5             ALL SYSTEMS                  99/12/01
C PORTABILITY : ALL SYSTEMS
C 99/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* COMPUTATION OF VALUE OF THE AUGMENTED LAGRANGIAN FUNCTION.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CF(NC+1)  VECTOR CONTAINING VALUES OF CONSTRAINT FUNCTIONS.
*  RI  CFD(NC)  VECTOR CONTAINING INCREMENTS OF CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
```

```
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  II  ICA(NCA)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
*  II  S(NCA)  VECTOR CONTAINING NEWTON INCREMENTS.
*  RI  GS(NCA)  VECTOR CONTAINING ACTIVE SLACK INCREMENTS.
*  RI  XO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR LOWER INACTIVE
*         MULTIPLIERS.
*  RI  GO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR UPPER INACTIVE
*         MULTIPLIERS.
*  RI  R  STEPSIZE  PARAMETER.
*  RI  RPF2  PENALTY COEFFICIENT.
*  RO  FC  VALUE OF THE PENALTY TERM.
*  RO  F  VALUE OF THE PENALTY FUNCTION.
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*        KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UP0AF5(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CF,CFD,
     & IC,CL,CU,CSL,CSU,CZL,CZU,ICA,S,GS,XO,GO,R,RPF2,FC,F)
      $INCLUDE ('UMCOMM')
      INTEGER NF,NX,NC,NCA,IX(NX),IC(NC),ICA(NCA)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & CF(NC+1),CFD(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),
     & S(NF+NCA),GS(NCA),XO(NF+NC),GO(NF+NC),R,RR,RPF2,FC,F
      INTEGER I,K,KK
      $FLOAT TEMP
      RR=ONE-R
      FC=0.0$P 0
      DO 1 I=1,NCA
      K=ICA(I)
      IF (K.LT.0) THEN
      K=-K
      KK=-IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      TEMP=XL(K)-X(K)+XSL(K)+R*GS(I)
      FC=FC+(XZL(K)+S(NF+I)+HALF*RPF2*TEMP)*TEMP
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      TEMP=X(K)-XU(K)+XSU(K)+R*GS(I)
      FC=FC+(XZU(K)+S(NF+I)+HALF*RPF2*TEMP)*TEMP
```

```
      ENDIF
      ELSE
      KK=-IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      TEMP=CL(K)-CF(K)+CSL(K)+R*GS(I)
      FC=FC+(CZL(K)+S(NF+I)+HALF*RPF2*TEMP)*TEMP
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      TEMP=CF(K)-CU(K)+CSU(K)+R*GS(I)
      FC=FC+(CZU(K)+S(NF+I)+HALF*RPF2*TEMP)*TEMP
      ELSE IF (KK.EQ.5) THEN
      FC=FC+(CZL(K)+S(NF+I)+HALF*RPF2*CF(K))*CF(K)
      ENDIF
      ENDIF
    1 CONTINUE
      IF (KBF.GT.0) THEN
      DO 2 K=1,NX
      KK=IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP=RR*(XSL(K)+XL(K)-X(K))+R*S(K)
      FC=FC+(XZL(K)+XO(K)+HALF*RPF2*TEMP)*TEMP
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      TEMP=RR*(XSU(K)-XU(K)+X(K))-R*S(K)
      FC=FC+(XZU(K)+GO(K)+HALF*RPF2*TEMP)*TEMP
      ENDIF
    2 CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
      DO 3 K=1,NC
      KK=IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP=RR*(CSL(K)+CL(K))-CF(K)+R*CFD(K)
      FC=FC+(CZL(K)+XO(NX+K)+HALF*RPF2*TEMP)*TEMP
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      TEMP=RR*(CSU(K)-CU(K))+CF(K)-R*CFD(K)
      FC=FC+(CZU(K)+GO(NX+K)+HALF*RPF2*TEMP)*TEMP
      ENDIF
    3 CONTINUE
      ENDIF
      F=CF(NC+1)+FC
      RETURN
      END
```

UCIMAX

```
* SUBROUTINE UCIMAX                ALL SYSTEMS                99/12/01
C PORTABILITY : ALL SYSTEMS
C 99/12/01 MA : ORIGINAL VERSION
*
* PURPOSE :
* DETRMINATION OF THE MAXIMUM STEPSIZE.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CFD(NC)  VECTOR CONTAINING VALUES OF THE CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  II  ICA(NX+NC)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
*  II  S(NF+NCA)  DIRECTION VECTOR.
*  RI  GS(NCA)  VECTOR CONTAINING ACTIVE SLACK INCREMENTS.
*  RI  XO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR LOWER INACTIVE
*         MULTIPLIERS.
*  RI  GO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR UPPER INACTIVE
*         MULTIPLIERS.
*  RU  RMAX  MAXIMUM VALUE OF THE PRIMAL STEPSIZE PARAMETER.
*  RU  RMAXD  MAXIMUM VALUE OF THE DUAL STEPSIZE PARAMETER.
*
*  COMMON DATA :
```

```
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*          KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*          CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*          SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UCIMAX(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CFD,IC,
     & CL,CU,CSL,CSU,CZL,CZU,ICA,S,GS,XO,GO,RMAX,RMAXD)
      $INCLUDE('UMCOMM')
      INTEGER NF,NX,NC,NCA,IX(NF),IC(NC),ICA(NCA)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & CFD(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),S(NF+NCA),
     & GS(NCA),XO(NX+NC),GO(NX+NC),RMAX,RMAXD
      INTEGER I,K,KK
      $FLOAT TEMP,CON
      PARAMETER(CON=0.99$P 0)
      RMAXD=RMAX
      DO 1 I=1,NCA
      K=ICA(I)
      IF (K.LT.0) THEN
      K=-K
      KK=-IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      IF (GS(I).LT.ZERO) RMAX=MIN(RMAX,-XSL(K)/GS(I))
      IF (S(NF+I).LT.ZERO) RMAXD=MIN(RMAXD,-XZL(K)/S(NF+I))
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      IF (GS(I).LT.ZERO) RMAX=MIN(RMAX,-XSU(K)/GS(I))
      IF (S(NF+I).LT.ZERO) RMAXD=MIN(RMAXD,-XZU(K)/S(NF+I))
      ENDIF
      ELSE
      KK=-IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      IF (GS(I).LT.ZERO) RMAX=MIN(RMAX,-CSL(K)/GS(I))
      IF (S(NF+I).LT.ZERO) RMAXD=MIN(RMAXD,-CZL(K)/S(NF+I))
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      IF (GS(I).LT.ZERO) RMAX=MIN(RMAX,-CSU(K)/GS(I))
      IF (S(NF+I).LT.ZERO) RMAXD=MIN(RMAXD,-CZU(K)/S(NF+I))
      ENDIF
      ENDIF
    1 CONTINUE
      IF (KBF.GT.0) THEN
      DO 2 K=1,NX
      KK=IX(K)
```

```
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP=X(K)+S(K)-XL(K)-XSL(K)
      IF (TEMP.LT.ZERO) RMAX=MIN(RMAX,-XSL(K)/TEMP)
      TEMP=XO(K)
      IF (TEMP.LT.ZERO) RMAXD=MIN(RMAXD,-XZL(K)/TEMP)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      TEMP=XU(K)-X(K)-S(K)-XSU(K)
      IF (TEMP.LT.ZERO) RMAX=MIN(RMAX,-XSU(K)/TEMP)
      TEMP=GO(K)
      IF (TEMP.LT.ZERO) RMAXD=MIN(RMAXD,-XZU(K)/TEMP)
      ENDIF
    2 CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
      DO 3 K=1,NC
      KK=IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP=CFD(K)-CL(K)-CSL(K)
      IF (TEMP.LT.ZERO) RMAX=MIN(RMAX,-CSL(K)/TEMP)
      TEMP=XO(NX+K)
      IF (TEMP.LT.ZERO) RMAXD=MIN(RMAXD,-CZL(K)/TEMP)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      TEMP=CU(K)-CFD(K)-CSU(K)
      IF (TEMP.LT.ZERO) RMAX=MIN(RMAX,-CSU(K)/TEMP)
      TEMP=GO(NX+K)
      IF (TEMP.LT.ZERO) RMAXD=MIN(RMAXD,-CZU(K)/TEMP)
      ENDIF
    3 CONTINUE
      ENDIF
      RMAX=CON*RMAX
      RMAXD=CON*RMAXD
      RETURN
      END
```

UCIDEL

```
* SUBROUTINE UCIDEL             ALL SYSTEMS                    99/12/01
C PORTABILITY : ALL SYSTEMS
C 99/12/01 LU : ORIGINAL VERSION
*
```

```
* PURPOSE :
* DETERMINATION OF ACTIVE AND INACTIVE INCREMENTS.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CF(NC+1)  VECTOR CONTAINING VALUES OF THE CONSTRAINT FUNCTIONS.
*  RI  CFD(NC)  VECTOR CONTAINING INCREMENTS OF CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  II  ICA(NCA)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
*  RI  S(NF+NCA)  VECTOR CONTAINING NEWTON INCREMENTS.
*  RI  D(NCA)  DIAGONAL SOUTH-EAST CORNER.
*  RO  GS(NCA)  VECTOR CONTAINING ACTIVE SLACK INCREMENTS.
*  RO  XO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR LOWER INACTIVE
*         MULTIPLIERS.
*  RO  GO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR UPPER INACTIVE
*         MULTIPLIERS.
*  RI  RPF3  BARRIER COEFFICIENT.
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*         KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UCIDEL(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CF,CFD,
```

```
 & IC,CL,CU,CSL,CSU,CZL,CZU,ICA,S,D,GS,XO,GO,RPF3)
  $INCLUDE ('UMCOMM')
  INTEGER NF,NX,NC,NCA,IX(NX),IC(NC),ICA(NCA)
  $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
 & CF(NC),CFD(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),
 & S(NF+NCA),D(NCA),GS(NCA),XO(NX+NC),GO(NX+NC),RPF3
  INTEGER I,K,KK
  DO 1 I=1,NCA
  K=ICA(I)
  IF (K.LT.0) THEN
  K=-K
  KK=-IX(K)
  IF (KK.EQ.1.OR.KK.EQ.3) THEN
  S(NF+I)=-S(NF+I)
  GS(I)=D(I)*S(NF+I)-XSL(K)+RPF3/XZL(K)
  ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
  GS(I)=D(I)*S(NF+I)-XSU(K)+RPF3/XZU(K)
  ENDIF
  ELSE
  KK=-IC(K)
  IF (KK.EQ.1.OR.KK.EQ.3) THEN
  S(NF+I)=-S(NF+I)
  GS(I)=D(I)*S(NF+I)-CSL(K)+RPF3/CZL(K)
  ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
  GS(I)=D(I)*S(NF+I)-CSU(K)+RPF3/CZU(K)
  ENDIF
  ENDIF
1 CONTINUE
  IF (KBF.GT.0) THEN
  DO 2 K=1,NX
  KK=IX(K)
  IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
  XO(K)=(RPF3+XZL(K)*(XL(K)-X(K)-S(K)))/XSL(K)
  ENDIF
  IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
  GO(K)=(RPF3+XZU(K)*(X(K)+S(K)-XU(K)))/XSU(K)
  ENDIF
2 CONTINUE
  ENDIF
  IF (KBC.GT.0) THEN
  DO 3 K=1,NC
  KK=IC(K)
  IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
```

```
      XO(NX+K)=(RPF3+CZL(K)*(CL(K)-CF(K)-CFD(K)))/CSL(K)
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      GO(NX+K)=(RPF3+CZU(K)*(CF(K)+CFD(K)-CU(K)))/CSU(K)
      ENDIF
    3 CONTINUE
      ENDIF
      RETURN
      END
```

UCIDER

```
* SUBROUTINE UCIDER             ALL SYSTEMS                 98/12/01
C PORTABILITY : ALL SYSTEMS
C 98/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* DETERMINATION OF THE DIRECTIONAL DERIVATIVE.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CF(NC+1)  VECTOR CONTAINING VALUES OF THE CONSTRAINT FUNCTIONS.
*  RI  CFD(NC)  VECTOR CONTAINING INCREMENTS OF CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  II  ICA(NCA)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
```

```
*  II  S(NF+NCA)  VECTOR CONTAINING NEWTON INCREMENTS.
*  RI  GS(NCA)  VECTOR CONTAINING ACTIVE SLACK INCREMENTS.
*  RI  XO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR LOWER INACTIVE
*         MULTIPLIERS.
*  RI  GO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR UPPER INACTIVE
*         MULTIPLIERS.
*  RI  STEPSIZE  PARAMETER.
*  RI  RPF3  BARRIER COEFFICIENT.
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*         KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UCIDER(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CF,CFD,
     & IC,CL,CU,CSL,CSU,CZL,CZU,ICA,S,GS,XO,GO,RPF2,RPF3,P)
      $INCLUDE ('UMCOMM')
      INTEGER NF,NX,NC,NCA,IX(NX),IC(NC),ICA(NCA)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & CF(NC),CFD(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),
     & S(NF+NCA),GS(NCA),XO(NX+NC),GO(NX+NC),RPF2,RPF3,P
      INTEGER I,K,KK
      $FLOAT TEMP,TEMP1,TEMP2
      DO 1 I=1,NCA
      TEMP=GS(I)
      K=ICA(I)
      IF (K.LT.0) THEN
      K=-K
      KK=-IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      TEMP1=S(NF+I)
      TEMP2=XL(K)-X(K)+XSL(K)
      P=P-RPF3*TEMP/XSL(K)+TEMP*(XZL(K)+TEMP1)-S(K)*(TEMP1+RPF2*TEMP2)+
     & RPF2*TEMP*TEMP2
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      TEMP1=S(NF+I)
      TEMP2=X(K)-XU(K)+XSU(K)
      P=P-RPF3*TEMP/XSU(K)+TEMP*(XZU(K)+TEMP1)+S(K)*(TEMP1+RPF2*TEMP2)+
     & RPF2*TEMP*TEMP2
      ENDIF
      ELSE
```

```
      KK=-IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      TEMP1=S(NF+I)
      TEMP2=CL(K)-CF(K)+CSL(K)
      P=P-RPF3*TEMP/CSL(K)+TEMP*(CZL(K)+TEMP1)-CFD(K)*(TEMP1+RPF2*TEMP2)+
     & RPF2*TEMP*TEMP2
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      TEMP1=S(NF+I)
      TEMP2=CF(K)-CU(K)+CSU(K)
      P=P-RPF3*TEMP/CSU(K)+TEMP*(CZU(K)+TEMP1)+CFD(K)*(TEMP1+RPF2*TEMP2)+
     & RPF2*TEMP*TEMP2
      ELSE IF (KK.EQ.5) THEN
      P=P+CFD(K)*(S(NF+I)+RPF2*CF(K))
      ENDIF
      ENDIF
    1 CONTINUE
      IF (KBF.GT.0) THEN
      DO 2 K=1,NX
      KK=IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP1=XO(K)
      TEMP2=XL(K)-X(K)+XSL(K)
      TEMP=-(TEMP2-S(K))
      P=P-RPF3*TEMP/XSL(K)+TEMP*(XZL(K)+TEMP1)-S(K)*TEMP1-RPF2*TEMP2**2
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      TEMP1=GO(K)
      TEMP2=X(K)-XU(K)+XSU(K)
      TEMP=-(TEMP2+S(K))
      P=P-RPF3*TEMP/XSU(K)+TEMP*(XZU(K)+TEMP1)+S(K)*TEMP1-RPF2*TEMP2**2
      ENDIF
    2 CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
      DO 3 K=1,NC
      KK=IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      TEMP1=XO(NX+K)
      TEMP2=CL(K)-CF(K)+CSL(K)
      TEMP=-(TEMP2-CFD(K))
      P=P-RPF3*TEMP/CSL(K)+TEMP*(CZL(K)+TEMP1)-CFD(K)*TEMP1-
     & RPF2*TEMP2**2
     ENDIF
```

```
 IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
  TEMP1=GO(NX+K)
  TEMP2=CF(K)-CU(K)+CSU(K)
  TEMP=-(TEMP2+CFD(K))
  P=P-RPF3*TEMP/CSU(K)+TEMP*(CZU(K)+TEMP1)+CFD(K)*TEMP1-
 & RPF2*TEMP2**2
  ENDIF
3 CONTINUE
  ENDIF
  RETURN
  END
```

UCIDIR

```
* SUBROUTINE UCIDIR             ALL SYSTEMS                  98/12/01
C PORTABILITY : ALL SYSTEMS
C 98/12/01 LU : ORIGINAL VERSION
*
* PURPOSE :
* STEP FOR LAGRANGIAN MULTIPLIERS.
*
* PARAMETERS :
*  II  NF  NUMBER OF VARIABLES.
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  NCA  NUMBER OF ACTIVE CONSTRAINTS.
*  II  X(NF)  VECTOR OF VARIABLES.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XL(NF)  VECTOR CONTAINING LOWER BOUNDS FOR VARIABLES.
*  RI  XU(NF)  VECTOR CONTAINING UPPER BOUNDS FOR VARIABLES.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  RI  CFD(NC)  VECTOR CONTAINING INCREMENTS OF CONSTRAINT FUNCTIONS.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CL(NC)  VECTOR CONTAINING LOWER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CU(NC)  VECTOR CONTAINING UPPER BOUNDS FOR CONSTRAINT FUNCTIONS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
```

```
*  II  ICA(NCA)  VECTOR CONTAINING INDICES OF ACTIVE CONSTRAINTS.
*  II  S(NF+NCA)  VECTOR CONTAINING NEWTON INCREMENTS.
*  RI  GS(NCA)  VECTOR CONTAINING ACTIVE SLACK INCREMENTS.
*  RI  XO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR LOWER INACTIVE
*         MULTIPLIERS.
*  RI  GO(NX+NC)  VECTOR CONTAINING INCREMENTS FOR UPPER INACTIVE
*         MULTIPLIERS.
*  RI  R  STEPSIZE  PARAMETER.
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*         KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UCIDIR(NF,NX,NC,NCA,X,IX,XL,XU,XSL,XSU,XZL,XZU,CFD,
     & IC,CL,CU,CSL,CSU,CZL,CZU,ICA,S,GS,XO,GO,R)
      $INCLUDE ('UMCOMM')
      INTEGER NF,NX,NC,NCA,IX(NX),IC(NC),ICA(NCA)
      $FLOAT X(NF),XL(NX),XU(NX),XSL(NX),XSU(NX),XZL(NX),XZU(NX),
     & CFD(NC),CL(NC),CU(NC),CSL(NC),CSU(NC),CZL(NC),CZU(NC),S(NF+NCA),
     & GS(NCA),XO(NF+NC),GO(NF+NC),R,RR
      INTEGER I,K,KK
      RR=ONE-R
      DO 1 I=1,NCA
      K=ICA(I)
      IF (K.LT.0) THEN
      K=-K
      KK=-IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      XZL(K)=XZL(K)+R*S(NF+I)
      XSL(K)=XSL(K)+R*GS(I)
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
      XZU(K)=XZU(K)+R*S(NF+I)
      XSU(K)=XSU(K)+R*GS(I)
      ENDIF
      ELSE
      KK=-IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3) THEN
      CZL(K)=CZL(K)+R*S(NF+I)
      CSL(K)=CSL(K)+R*GS(I)
      ELSE IF (KK.EQ.2.OR.KK.EQ.4) THEN
```

```
      CZU(K)=CZU(K)+R*S(NF+I)
      CSU(K)=CSU(K)+R*GS(I)
      ELSE IF (KK.EQ.5) THEN
      CZL(K)=CZL(K)+R*S(NF+I)
      ENDIF
      ENDIF
    1 CONTINUE
      IF (KBF.GT.0) THEN
      DO 2 K=1,NX
      KK=IX(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      XZL(K)=XZL(K)+R*XO(K)
      XSL(K)=RR*XSL(K)+R*(X(K)+S(K)-XL(K))
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      XZU(K)=XZU(K)+R*GO(K)
      XSU(K)=RR*XSU(K)+R*(XU(K)-X(K)-S(K))
      ENDIF
    2 CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
      DO 3 K=1,NC
      KK=IC(K)
      IF (KK.EQ.1.OR.KK.EQ.3.OR.ABS(KK).EQ.4) THEN
      CZL(K)=CZL(K)+R*XO(NX+K)
      CSL(K)=RR*CSL(K)+R*(CFD(K)-CL(K))
      ENDIF
      IF (KK.EQ.2.OR.KK.EQ.4.OR.ABS(KK).EQ.3) THEN
      CZU(K)=CZU(K)+R*GO(NX+K)
      CSU(K)=RR*CSU(K)+R*(CU(K)-CFD(K))
      ENDIF
    3 CONTINUE
      ENDIF
      RETURN
      END
```

UPSET3

Šablona UPSET3 počítá hodnotu bariérového parametru μ podle (1.37).

```
$INTERFACE UPSET3                                               99/01/23
```

```
  CALL UPSET3(NX,NC,$IX,$XSL,$XSU,$XZL,$XZU,$IC,$CSL,$CSU,$CZL,$CZU&
,$ETA9,RPF3)
 $EXIT

* IMPLEMENTATION

* SUBROUTINE UPSET3             ALL SYSTEMS                 98/12/01
C PORTABILITY : ALL SYSTEMS
C 98/12/01 MA : ORIGINAL VERSION
*
* PURPOSE :
* BARRIER PARAMETER DETERMINATION (R.J.VANDERBEI AND F.SHANNO
* ALGORITHM).
*
* PARAMETERS :
*  II  NX  NUMBER OF BOX CONSTRAINTS.
*  II  NC  NUMBER OF GENERAL CONSTRAINTS.
*  II  IX(NF)  VECTOR CONTAINING TYPES OF BOUNDS.
*  RI  XSL(NX)  VECTOR CONTAINING LOVER SLACKS FOR VARIABLES.
*  RI  XSU(NX)  VECTOR CONTAINING UPPER SLACKS FOR VARIABLES.
*  RI  XZL(NF)  VECTOR CONTAINING LOVER MULTIPLIERS FOR VARIABLES.
*  RI  XZU(NF)  VECTOR CONTAINING UPPER MULTIPLIERS FOR VARIABLES.
*  II  IC(NC)  VECTOR CONTAINING TYPES OF CONSTRAINTS.
*  RI  CSL(NC)  VECTOR CONTAINING LOVER SLACKS FOR CONSTRAINTS.
*  RI  CSU(NC)  VECTOR CONTAINING UPPER SLACKS FOR CONSTRAINTS.
*  RI  CZL(NC)  VECTOR CONTAINING LOVER MULTIPLIERS FOR CONSTRAINTS.
*  RI  CZU(NC)  VECTOR CONTAINING UPPER MULTIPLIERS FOR CONSTRAINTS.
*  RI  ETA9  MAXIMUM MAXHINE NUMBER.
*  RI  RPF3  BARRIER COEFFICIENT.
*
*  COMMON DATA :
*  II  KBF  SPECIFICATION OF SIMPLE BOUNDS. KBF=0-NO SIMPLE BOUNDS.
*         KBF=1-ONE SIDED SIMPLE BOUNDS. KBF=2=TWO SIDED SIMPLE BOUNDS.
*  II  KBC  SPECIFICATION OF LINEAR CONSTRAINTS. KBC=0-NO LINEAR
*         CONSTRAINTS. KBC=1-ONE SIDED LINEAR CONSTRAINTS. KBC=2=TWO
*         SIDED LINEAR CONSTRAINTS.
*
      SUBROUTINE UPSET3(NX,NC,IX,XSL,XSU,XZL,XZU,IC,CSL,CSU,CZL,CZU,
     & ETA9,RPF3)
      $INCLUDE ('UMCOMM')
      INTEGER NX,NC,IX(NX),IC(NC)
      $FLOAT XSL(NX),XSU(NX),XZL(NX),XZU(NX),CSL(NC),CSU(NC),CZL(NC),
     & CZU(NC),ETA9,RPF3
```

```
      $FLOAT STEPL,SCALEF,CDOTMIN,CDOT1,SXI
      INTEGER I,K,IIC
      PARAMETER(STEPL=9.5$P-1,SCALEF=1.0$P-1)
      IIC=0
      CDOT1=ZERO
      CDOTMIN=ETA9
      IF (KBF.GT.0) THEN
        DO 2 I=1,NX
          K=ABS(IX(I))
          IF (K.EQ.1.OR.K.EQ.3.OR.K.EQ.4) THEN
            SXI=XSL(I)*ABS(XZL(I))
            IF (SXI.LT.CDOTMIN) THEN
              CDOTMIN=SXI
            ENDIF
            CDOT1=CDOT1+SXI
            IIC=IIC+1
          ENDIF
          IF (K.EQ.2.OR.K.EQ.3.OR.K.EQ.4) THEN
            SXI=XSU(I)*ABS(XZU(I))
            IF (SXI.LT.CDOTMIN) THEN
              CDOTMIN=SXI
            ENDIF
            CDOT1=CDOT1+SXI
            IIC=IIC+1
          ENDIF
    2   CONTINUE
      ENDIF
      IF (KBC.GT.0) THEN
        DO 3 I=1,NC
          K=ABS(IC(I))
          IF (K.EQ.1.OR.K.EQ.3.OR.K.EQ.4) THEN
            SXI=CSL(I)*ABS(CZL(I))
            IF (SXI.LT.CDOTMIN) THEN
              CDOTMIN=SXI
            ENDIF
            CDOT1=CDOT1+SXI
            IIC=IIC+1
          ENDIF
          IF (K.EQ.2.OR.K.EQ.3.OR.K.EQ.4) THEN
            SXI=CSU(I)*ABS(CZU(I))
            IF (SXI.LT.CDOTMIN) THEN
              CDOTMIN=SXI
            ENDIF
```

```
            CDOT1=CDOT1+SXI
            IIC=IIC+1
          ENDIF
    3   CONTINUE
      ENDIF
      IF (IIC.EQ.0) THEN
        RPF3=ZERO
        RETURN
      ENDIF
      CDOT1=CDOT1/$DBLE(IIC)
      SXI=CDOTMIN/CDOT1
      WRITE(2,*) '-- A --',CDOT1,CDOTMIN,SXI
      CDOTMIN=MIN((ONE-STEPL)*(ONE-SXI)/SXI,TWO)
C      CDOTMIN=ONE
      RPF3=SCALEF*CDOT1*CDOTMIN**3
      WRITE(2,*) '-- B --',SCALEF,CDOTMIN,RPF3
      END
```

C.2 Příklad zadání úlohy

TEST20i1.UFO

```
$SET(INPUT)
  CALL EIUS20(NF,M,X,IH,JH,FMIN,XMAX,NEXT,IEXT,IERR)
  CALL EINS20(NF,NC,NCL,MC,ICG,JCG,NEXT)
  CALL UXVINS(NC,1,IC)
  CALL UXVSET(NC,ZERO,CL)
  IF (NEXT.EQ.6) XMAX=1.0$P 1
  IF (NEXT.EQ.8) GO TO $$ENDTEST
  IF (IERR.NE.0) GO TO $$ENDTEST
$ENDSET
$SET(FMODELF)
  CALL EFFU20(NF,X,FF,NEXT)
$ENDSET
$SET(FMODELC)
  CALL ECFU20(NF,KC,X,FC,NEXT)
$ENDSET
$SET(GMODELF)
  CALL EFGU20(NF,X,GF,NEXT)
$ENDSET
```

```
$SET(GMODELC)
  CALL ECGU20(NF,KC,X,GC,NEXT)
$ENDSET
$SET(CONST)
  IF (ABS(MOS2).EQ.0) THEN
  IF (NEXT.EQ. 1) RPF1=1.0$P-2
  IF (NEXT.EQ. 4) RPF1=1.0$P-1
  IF (NEXT.EQ. 7) RPF1=1.0$P-1
  IF (NEXT.EQ. 9) RPF1=1.0$P-2
  IF (NEXT.EQ.12) RPF1=1.0$P 2
  ELSE IF (ABS(MOS2).EQ.1) THEN
  RPF1=0.0$P 0
  IF (NEXT.EQ. 1) RPF1=2.0$P-1
  IF (NEXT.EQ. 6) RPF1=1.0$P 4
  IF (NEXT.EQ. 7) RPF1=2.0$P-1
  IF (NEXT.EQ.11) RPF1=1.0$P-4
  IF (NEXT.EQ.12) RPF1=1.0$P-4
  IF (NEXT.EQ.13) RPF1=2.0$P 0
  ENDIF
$ENDSET
$NF=100
$M=1500
$MMAX=5000
$NC=100
$NCL=0
$MC=800
$ME=100
$KBC=1
$MIT=200
$MFV=1000
$KOUT=0
$LOUT=1
$MOUT=1
$TOLX='1.0$P-16'
$TOLG='1.0$P-6'
$TOLC='1.0$P-6'
$MODEL='FF'
$JACC='S'
$HESF='S'
$FORM='SI'
$CLASS='MN'
$TYPE='L'
$DECOMP='I'
```

```
$NUMBER=3
$MEP=2
$MEP1=1
$MEP2=1
$MEP3=1
$MOS1=0
$MOS2=1
$MOS3=0
$COLLECTION='YES'
$NEXT=18
$BATCH

$STANDARD
```

Literatura

[1] M. Argaez, R. A. Tapia, L. Velazquez: *Numerical Comparisons of Path-Following Strategies for a Basic Interior-Point Method for Nonlinear Programming*, Center for Research on Parallel Computation, Rice Univesity, Houston, CRPC-TR97777-S (1998)

[2] Marc G. Breitfeld, David F. Shanno: *A Globally Convergent Penalty-Barrier Algorithm for Nonlinear Programming and its Computational Performance*, RUTCOR, Rutgers Center for Operations Research, Rutgers University, New Brunswick, New Jersey, RRR 17-93 (1993)

[3] Marc G. Breitfeld, David F. Shanno: *Computational Experience with Penalty-Barrier Methods for Nonlinear Programming*, RUTCOR, Rutgers Center for Operations Research, Rutgers University, New Brunswick, New Jersey, RRR 17-93 (1993)

[4] A. El-Bakry, R. Tapia, T. Tsuchiya, Y. Zhang: *On the formulation and theory of Newton Interior-point method for nonlinear programming*, Journal of Optimization Theory and Applications (89:507-541, 1996)

[5] R. H. Byrd, G. Liu, J. Nocedal: *On the Local Behavior of an Interior Point Method for Nonlinear Programming*, Report OTC 98/02 (1998)

[6] A. V. Fiacco, G. P. McCormick: *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, Research Analysis Corporation, McLean Virginia (1968)

[7] A. Forsgren, P. E. Gill: *Primal-dual interior methods for nonconvex nonlinear programming*, Report NA 96-3, Departement of Mathematics, University of Carolina (1996)

[8] A. Forsgren, P. E. Gill, J. R. Shinnerl: *Stability of symmetric ill-conditioned systems arising in iterior methods for constrained optimization*, SIAM Journal of Matrix Analalysis and Applications vol. 17 (1996)

[9] P. E. Gill, W. Murray, M. H. Wright: *Practical Optimization*, Academic Press, London, New York (1981)

[10] V. Klee, G. J. Minty: *How good is the simplex algorithm?*, Inequalities, O. Shisha, ed., Academic Press, New York (1972)

[11] F. A. Lootsma: *Hessian matrices of penalty functions for solving constrained optimization problems*, Philips Res. Repts. 24 (1969)

[12] L. Lukšan, J. Vlček: *Indefinitely Preconditioned Inexact Newton Method for Large Sparse Equality Constrained Nonlinear Programing Problems*, Numerical Linear Algebra with Applications vol. 5 number 3 (1998)

[13] L. Lukšan, J. Vlček: *Subroutines for Testing Large Sparse and Partially Separable Unconstrained and Eqality Constrained Optimization Problems*, Technical Report 767, Ústav Informatiky a Výpočetní Techniky AVČR (1999)

[14] L. Lukšan: *Numerical Methods for Unconstrained Optimization*, Ústav Informatiky a Výpočetní Techniky AVČR (1996)

[15] L. Lukšan, M. Tůma, M. Šiska, J. Vlček, N. Ramešová: *Interactive System for Universal Functional Optimization*, Technical Report V-738, Ústav Informatiky a Výpočetní Techniky AVČR (1997)

[16] W. Murray: *Analytical expressions for the eigenvalues and eigenvectors of the Hessian matrices of barrier and penalty functions*, Journal of Optimization Theory and Applications vol. 7 (1971)

[17] Richart A. Tapia: *On the Fundamental Role of Interior-Point Methology in Constrained Optimization*, Center for Research on Parallel Computation, Rice Univesity, Houston, CRPC-TR97730 (1997)

[18] Richart A. Tapia: *On the role of slack variables in quasi-Newton methods for constrained optimization*, Numerical Optimization of Dynamic Systems, L. C. W. Dixon and D. P. Szego, eds., North–Holland, Amsterdam (1980)

[19] Thomas Urban, André L. Tits, Craig T. Lawrence: *A primal-dual interior-point method for nonconvex optimization with multiple logarithmic barrier parameters and with strong convergence properties*, University of Maryland, College Park (1998)

[20] Robert J. Vanderbei, David F. Shanno: *An Interior Point Algorithm for Nonconvex Nonlinear Programming*, Statistics and Operation Research, Princeton University (SOR-97-12)

[21] Margaret H. Wright: *Interior methods for constrained optimization*, Acta Numerica (1991)

[22] Margaret H. Wright: *Some properties of the Hessian of then logarithmic barrier function*, Mathematical Programming 67 (1994)

[23] Margaret H. Wright: *Ill-Conditioning and Computational Error in Interior Methods for Nonlinear Programming*, Technical Report 97-4-04, Computing Sciences Research Center, Bell Laboratories (1997)

[24] S. J. Wright: *Stability of linear equation solvers in interior-point methods*, SIAM Journal of Matrix Analalysis and Applications vol. 16 (1995)

[25] S. J. Wright: *Stability of augmented system factorizations in interior-point methods*, SIAM Journal of Matrix Analalysis and Applications vol. 18 (1997)

[26] S. J. Wright: *Modified Cholesky factorizations in interor-point algorithms for linear programming*, Technical Report ANL/MCS-P600-0595, Argonne National Laboratory, Argonne, Illinois (1996)

[27] S. J. Wright: *Effects of finite-precision arithmetic on interior-point methods for monlinear programming*, Technical Report ANL/MCS-P705-0198, Argonne National Laboratory, Argonne, Illinois (1998)

[28] Yinyu Ye: *Interior Point Algorithms - Theory and Analysis*, A Wiley-Interscience Publication (1997)