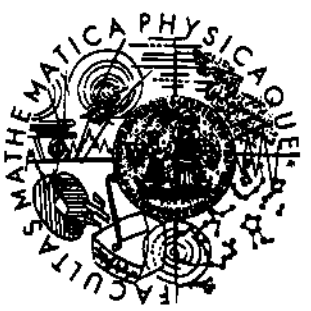
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jan Fiala

Předpodmíněné metody Krylovova typu v optimalizačních algoritmech

Katedra numerické matematiky
Vedoucí diplomové práce: RNDr. Michal Kočvara, DrSc.
Studijní program: Matematika
Studijní obor: Výpočtová matematika
Studijní plán: Výpočtová matematika – software

Chtěl bych především poděkovat vedoucímu své diplomové práce panu RNDr. Michalu Kočvarovi za jeho cenné rady a připomínky, které mi při psaní velmi pomohly.
Dále bych chtěl poděkovat své rodině za její podporu a svatou trpělivost.

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

V Praze dne 5. dubna 2004 Jan Fiala

Obsah

Název práce: Předpodmíněné metody Krylovova typu v optimalizačních algoritmech
Autor: Jan Fiala
Katedra: Katedra numerické matematiky
Vedoucí diplomové práce: RNDr. Michal Kočvara, DrSc.
e-mail vedoucího: kocvara@utia.cas.cz
Abstrakt: Jedním z možných způsobů řešení obecných úloh podmíněné optimalizace je převést je na sekvence úloh nepodmíněné optimalizace. Vzniká tak potřeba spolehlivého a efektivního způsobu, jak problémy z této sekvence řešit. Zpravidla se využívá nějaká metoda Newtonova typu, často s použitím řídkého Choleského rozkladu. V praxi se však na některých problémech ukazuje, že tento způsob není vždy zcela vhodný a v takových případech je lepší nahradit Choleského rozklad či Newtonovu metodu nějakým vhodným iteračním algoritmem. Takovými algoritmy se budeme v této práci zabývat. Půjde o metody Krylovova typu, především o metody CGM a QMR a jejich předpodmínění. Tyto metody jsou aplikovány na úlohy nepodmíněné optimalizace, které vznikají při řešení problémů nelineárního a semidefinitního programování (NLP-SDP) pomocí zobecněné metody rozšířeného lagrangiánu. Konkrétní implementace je provedena v rámci programu PENNON.
Klíčová slova: nelineární programování, předpodmíněné sdružené gradienty, Newtonovy metody

Title: Preconditioned Krylov Subspace Methods in Optimization Algorithms
Author: Jan Fiala
Department: Department of Numerical Mathematics
Supervisor: RNDr. Michal Kočvara, DrSc.
Supervisor's e-mail adress: kocvara@utia.cas.cz
Abstract: One of the possible ways of solving general problems of constrained optimization is to convert them to a sequence of unconstrained problems. Then the need arises to solve unconstrained optimization problem reliably and efficiently. For this, Newton methods are usually applied, often in combination with sparse Cholesky decomposition. In practice, however, this approach may not be optimal in some cases and a suitable iterative algorithm may be preferred. The aim of this work is to use iterative algorithms with preconditioned Krylov subspace methods, like CGM and QMR, to solve unconstrained problems originating in the Augmented Lagrangian method applied to nonlinear and semidefinite programming (NLP-SDP). The specific implementation was carried out in PENNON software package.
Keywords: Nonlinear Programming, Preconditioned Conjugate Gradients, Newton Methods

Kapitola 1

Úvod

V řadě vědních oborů, ať už v matematice, fyzice, ekonomii, chemii, ale i v jiných, vznikají v dnešní době problémy, jejichž matematická formulace vede na úlohu tzv. *podmíněné optimalizace*, tj. na úlohu hledání minima funkce $f(x)$ na množině omezené nějakými podmínkami. Označme tuto množinu

$$\begin{aligned} \Omega := \{x \in \mathbb{R}^n \mid g_i(x) \leq 0, \qquad & i = 1, 2, \ldots, m_g \\ h_i(x) = 0, \qquad & i = 1, 2, \ldots, m_h\}, \end{aligned} \tag{1.1}$$

kde funkce $f, g_i, h_i : \mathbb{R}^n \to \mathbb{R}$ jsou třídy $\mathcal{C}^2$. Tyto problémy můžeme dělit podle několika hledisek:

- velikosti (dimenze)
- hustoty / řídkosti datových struktur
- konvexnosti / nekonvexnosti funkcí f, g_i, h_i
- typu funkcí (lineární, kvadratické, ...).

V některých případech jsou známy spolehlivé algoritmy pro jejich řešení (např. *lineární, konvexní kvadratické programování*), v ostatních nezbývá než použít dostatečně obecnou metodu.

Jedna třída takových metod převádí hledání řešení podmíněné optimalizace na sekvenci řešení úloh *nepodmíněné optimalizace*, tj. na úlohy minimalizace funkce na $\mathbb{R}^n$ bez dalších omezení. Myšlenka spočívá v nahrazení funkce $f(x)$ pozměněnou funkcí $\mathcal{F}(x)$ definovanou na celém $\mathbb{R}^n$, která v sobě bude zahrnovat kromě původní účelové funkce $f(x)$ navíc i nezáporné penalizační členy. Jejich hodnota se bude výrazně zvyšovat při porušení některé z podmínek g_i, h_i. Tím se docílí toho, že hledané minimum $\mathcal{F}(x)$ na $\mathbb{R}^n$ nemůže být „příliš daleko“ od množiny Ω a nalezené řešení bude vhodnou aproximací řešení původní úlohy. Z těchto metod jmenujme především *penalizační, barierové metody a metodu rozšířeného lagrangiánu (augmented Lagrangian method)*.

Tato práce je zaměřena na řešení úloh nepodmíněné optimalizace vznikající při použití zobecněné metody rozšířeného lagrangiánu. Poznamenejme, že charakter

úloh nepodmíněné optimalizace se sice příliš neliší při použití jiných metod (penalizační, barierové), ale uplatněním zobecněné metody rozšířeného lagrangiánu vznikají jednak lépe podmíněné problémy a jednak je tato metoda již implementována v programu PENNON, čehož využijeme při testování nových metod.

Zpravidla se úloha nepodmíněné optimalizace řeší nějakou vhodnou metodou Newtonova typu. V nich je potřeba často řešit soustavu s maticí druhých derivací (hessiánem) pozměněné funkce $\mathcal{F}$, k čemuž se většinou využívá *Choleského rozkladu* ([19]). Lze očekávat, že se tato metoda bude chovat velmi dobře jak při řešení malých úloh, tak i větších s řídkými datovými strukturami, pokud u větších použijeme *řídkého Choleského rozkladu* ([18]). Přesto můžeme v praxi narazit na úlohy, kde by bylo vhodné nahradit Choleského rozklad či Newtonovu metodu nějakým iteračním algoritmem. To bude hlavním předmětem této práce. O zlepšení lze uvažovat především v těchto případech:

- *Hessián funkce $\mathcal{F}$ není dostupný, nebo je příliš drahý na výpočet.* V mnoha praktických úlohách není k dispozici analytický tvar druhých derivací, v takových případech se používá metoda *automatického diferencování* ([19]). Ta někdy může být (např. v nelineárním SDP) velmi časově náročná, proto je nutné použít nějakou iterační metodu prvního řádu.

- *Choleského rozklad (velkého) řídkého hessiánu vede na plnou matici.* U větších úloh je nutné pro zachování co možná nejmenší paměťové a časové náročnosti provést před samotným Choleského rozkladem některý z algoritmů pro přečíslování (permutaci prvků matice), viz [15]. Ne vždy jsou však tyto algoritmy úspěšné. To bylo v praxi pozorováno například na úlohách s tzv. „plným sloupcem", tj. na úlohách, ve kterých se nějaká proměnná objevuje ve většině omezeních g_i. Pak může i přes přečíslování vést Choleského rozklad velmi řídké matice na matici plnou, což ve svém důsledku znamená výrazně vyšší paměťovou i časovou náročnost, nebo dokonce kolaps algoritmu na nedostatek paměti.

- *Choleského rozklad hustého hessiánu je zbytečně časově/paměťově náročný.* Časová náročnost rozkladu je řádu $O(n^3)$ a paměťová řádu $O(n^2)$. To je v některých dobře podmíněných úlohách „předražené", protože soustavu stačí řešit jen přibližně a mnohdy požadované přesnosti dosáhneme během několika málo kroků iteračního algoritmu.

Cílem práce je otestovat a zhodnotit alternativní metody pro řešení úloh nepodmíněné optimalizace, které vznikají z úloh podmíněné optimalizace použitím zobecněné metody rozšířeného lagrangiánu a jí podobným. Především jsme se zaměřili na takové metody nepodmíněné optimalizace, které by umožnily lépe řešit úlohy působící problémy ve většině podobných případů používané Newtonově metodě. Jak je uvedeno výše, jde obvykle o úlohy, ve kterých není vhodné, či dokonce možné použít Choleského rozklad, nebo nelze efektivním způsobem sestavit matici druhých derivací.

Pro konvexní případy byla jako základ algoritmu zvolena metoda spádových směrů, ve které hledáme nový směr poklesu jako přibližné řešení Newtonovy rovnice.

Zde byly použity metody Krylovova typu, konkrétně metoda sdružených gradientů a metoda QMR. Pro nekonvexní případy byla nahrazena metoda spádových směrů metodou s lokálně omezeným krokem, v našem případě verzí metody Steihaug-Tointa. Ve většině úloh by nebylo rozumné použít jmenované metody bez nějakého vhodného předpodmiňovače. V této práci jich bylo testováno několik. Kromě paměťově i časově nenáročných, jako je diagonální, SSOR či SGS předpodmiňovač, byly testovány i propracovanější. Jednak metoda L-BFGS upravená pro předpodmiňování podle [17] jako zástupce předpodmiňovačů, které by byly méně paměťově náročné, zato náročnější na výpočetní čas, a metoda AINV podle [3, 4] jako příklad neúplného rozkladu matice.

V úlohách, ve kterých není přímo dostupná matice druhých derivací, jsme využili toho, že v uvedených metodách obvykle není potřeba znát přímo celý hessián, ale jen jeho násobek s vektorem. Ten totiž lze poměrně snadno aproximovat pomocí konečných diferencí.

Konkrétní implementace a testování probíhaly v rámci programu PENNON. Jde o program určený pro řešení úloh nelineárního a semidefinitního programování. Užívá se v něm zobecněná metoda rozšířeného lagrangiánu a vznikající úlohy nepodmíněné optimalizace se řeší pomocí modifikované Newtonovy metody. Snahou bylo otestovat metody na co možná nejširším spektru problémů, k čemuž bylo využito úloh z volně dostupných knihoven problémů jako SDPLIB a CUTE Collection.

Práce je rozdělena do 6 částí. Nejdříve bude rekapitulováno základní značení a definice, které jsou společné pro celou práci. V kapitole 3 jsou detailněji popsány základní metody nepodmíněné optimalizace a myšlenka aproximace hessiánu a v kapitole 4 použité předpodmiňovače. Předposlední kapitola je věnována programu PENNON a zde užité modifikaci metody rozšířeného lagrangiánu. V poslední kapitole jsou uvedeny numerické výsledky, implementační poznámky a závěry.

Kapitola 2

Značení a základní definice

Pro sjednocení terminologie uvedeme v této kapitole použité značení a zopakujeme několik základních definic, které budeme dále využívat.

2.1 Funkce

Označení 2.1. Nechť f je dostatečně hladká funkce z $\mathbb{R}^n$ do $\mathbb{R}$. Její *gradient* v bodě $x \in \mathbb{R}^n$ je vektor

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)^T$$

a její *hessián* je symetrická $n \times n$ matice

$$\nabla^2 f(x) = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{i,j=1,\ldots,n}.$$

Definice 2.2. Řekneme, že množina $\Omega \subset \mathbb{R}^n$ je *konvexní*, jestliže pro všechna $x, y \in \Omega$ a $0 < \lambda < 1$ platí

$$\lambda x + (1 - \lambda) y \in \Omega.$$

Definice 2.3. Nechť $\Omega \subset \mathbb{R}^n$ je konvexní množina. O funkci $f : \Omega \to \mathbb{R}$ řekneme, že je

(i) *konvexní* na Ω, jestliže pro všechna $x, y \in \Omega$ a $0 < \lambda < 1$ platí

$$f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y),$$

(ii) *ryze konvexní* na Ω, jestliže předcházející nerovnost platí ostře.

2.2 Matice

Označení 2.4. Prostor všech reálných symetrických matic dimenze m budeme značit $\mathbb{S}^m$.

Definice 2.5. Mějme matici $A \in \mathbb{S}^m$. Řekneme, že je

(i) *pozitivně definitní* ($A \succ 0$), jestliže $\forall x \in \mathbb{R}^m . x \neq 0 : x^T Ax > 0$,

(ii) *pozitivně semidefinitní* ($A \succeq 0$), jestliže $\forall x \in \mathbb{R}^m : x^T Ax \geq 0$,

(iii) *negativně definitní* ($A \prec 0$), jestliže $\forall x \in \mathbb{R}^m, x \neq 0 : x^T Ax < 0$,

(iv) *negativně semidefinitní* ($A \preceq 0$), jestliže $\forall x \in \mathbb{R}^m : x^T Ax \leq 0$.

Poznámka 2.1. Nechť A, B jsou prvky prostoru $\mathbb{S}^m$. Na prostoru $\mathbb{S}^m$ můžeme zavést *částečné uspořádání* a *skalární součin* následovně:

$$\begin{aligned} A \preceq B \quad &\Leftrightarrow \quad (A - B) \preceq 0, \\ \langle A, B \rangle_{\mathbb{S}^m} &= \operatorname{tr}(AB), \end{aligned}$$

kde $\operatorname{tr}(X)$ pro $X \in \mathbb{R}^{m \times m}$ značí stopu matice X.

Definice 2.6. Mějme $r \in \mathbb{R}^m$ a $A \in \mathbb{S}^m$. Potom *Krylovovým prostorem stupně j generovaným maticí A a vektorem r*, označeným $\mathcal{K}_j(r, A)$, budeme rozumět

$$\mathcal{K}_j(r, A) = \mathcal{L}\{r, Ar, A^2 r, \ldots, A^{j-1} r\}.$$

Definice 2.7. Nechť A je regulární matice z $\mathbb{S}^m$, pak jejím *číslem podmíněnosti* $\kappa(A)$ nazveme

$$\kappa(A) = \|A\| \, \|A^{-1}\|.$$

Označení 2.8. Nechť $A \in \mathbb{S}^m, A \succeq 0$ má Choleského rozklad $A = LL^T$ a nechť $L = U\Sigma V^T$ značí SVD rozklad matice L. Potom $X \in \mathbb{S}^m, X = U\Sigma U^T$ označíme jako $A^{1/2}$.

Poznámka 2.2. Označení $A^{1/2}$ z definice 2.8 lze zdůvodnit následovně:

$$A = LL^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma^2 U^T = (U\Sigma U^T)(U\Sigma U^T) = X^2.$$

2.3 Maticové funkce

Definice 2.9 (Maticová funkce). Mějme $\varphi : \mathbb{R} \to \mathbb{R}$ a $A \in \mathbb{S}^m$. Nechť $A = S^T \Lambda S$, kde $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$, značí rozklad vlastních čísel matice A. Potom *maticová funkce* $\Phi : \mathbb{S}^m \to \mathbb{S}^m$ generovaná φ je definovaná

$$\Phi : A \mapsto S^T \begin{pmatrix} \varphi(\lambda_1) & 0 & \cdots & 0 \\ 0 & \varphi(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \varphi(\lambda_m) \end{pmatrix} S.$$

Definice 2.10. Maticovou funkci $\Phi : \mathbb{S}^m \to \mathbb{S}^m$ nazveme

(i) *monotonní*, jestliže pro každé $A, B \in \mathbb{S}^m$ platí

$$A \succeq B \;\Rightarrow\; \Phi(A) \succeq \Phi(B),$$

(ii) *konvexní*, jestliže pro všechna $A, B \in \mathbb{S}^m$ a $0 < \lambda < 1$ platí

$$\Phi(\lambda A + (1-\lambda)B) \preceq \lambda\Phi(A) + (1-\lambda)\Phi(B),$$

(iii) *ryze konvexní*, jestliže je předcházející nerovnost splněna ostře.

Definice 2.11. Nechť $\mathcal{A} : \mathbb{R}^n \to \mathbb{S}^m$ a $\Phi : \mathbb{S}^m \to \mathbb{S}^m$ jsou dva maticové operátory. Pro $B \in \mathbb{S}^m$ označíme $D_{\mathcal{A}}\Phi(\mathcal{A}(x); B)$ *derivaci* Φ *v bodě* $\mathcal{A}(x)$ (pro pevné x) *ve směru* B, tj.:

$$D_{\mathcal{A}}\Phi(\mathcal{A}(x); B) = \lim_{t \to 0} \frac{\Phi(\mathcal{A}(x) + tB) - \Phi(\mathcal{A}(x))}{t}.$$

2.4 Optimalizační úloha

Definice 2.12. Mějme $\Omega \subset \mathbb{R}^n$ a $f : \mathbb{R}^n \to \mathbb{R}$. Problém

$$\text{najdi } x^* \in \Omega \text{ tak, aby } f(x) \geq f(x^*) \quad \forall x \in \Omega \tag{2.1}$$

nazveme *optimalizační úlohou*. Jestliže $\Omega = \mathbb{R}^n$, budeme mluvit o úloze *nepodmíněné optimalizace*, jestliže $\Omega \subsetneq \mathbb{R}^n$, tak o úloze *podmíněné optimalizace*. Pokud množina Ω a funkce f, g_i, h_i jsou konvexní, nazveme (2.1) *konvexní* optimalizační úlohou, v opačném případě budeme hovořit o *nekonvexní* optimalizační úloze.

Definice 2.13. Řekneme, že

(i) x^* je *globální řešení* (2.1), jestliže $f(x) \geq f(x^*)\ \forall x \in \Omega$,

(ii) $\tilde{x}$ je *lokální řešení* (2.1), jestliže $\exists O_\epsilon(\tilde{x}) : f(x) \geq f(\tilde{x})\ \forall x \in \Omega \cap O_\epsilon(\tilde{x})$.

Definice 2.14. Mějme optimalizační úlohu (2.1), kde Ω označuje množinu (1.1). Nechť jsou funkce $f, h_i, g_j : \mathbb{R}^n \to \mathbb{R}$ třídy $\mathcal{C}^2$. Pak funkci

$$L : \mathbb{R}^n \times \mathbb{R}^{m_h} \times \mathbb{R}_+^{m_g} \to \mathbb{R}$$

$$L(x, u, v) := f(x) + \sum_{i=1}^{m_h} u_i h_i(x) + \sum_{j=1}^{m_g} v_j g_j(x)$$

nazveme *Lagrangeovou funkcí (lagrangiánem)* problému (2.1).

Na závěr této kapitoly uveďme ještě dvě základní tvrzení, důkazy lze nalézt například v [19].

Věta 2.1. *Nechť $\Omega \neq \emptyset$ je konvexní a $f : \Omega \to \mathbb{R}$ je (ryze) konvexní na Ω. Nechť $\tilde{x} \in \Omega$ je lokální řešení (2.1). Potom $\tilde{x}$ je (jednoznačné) globální řešení (2.1).*

Věta 2.2 (Nutné podmínky optimality 1. řádu). *Nechť x^* je lokální řešení úlohy (2.1). Nechť pro množinu $\mathcal{A}(x^*) = \{j \mid 1 \leq j \leq m_g, g_j(x^*) = 0\}$ platí, že vektory*

$$\nabla h_i(x^*) \quad pro \quad i = 1, \ldots, m_h, \quad \nabla g_j(x^*) \quad pro \quad j \in \mathcal{A}(x^*)$$

jsou lineárně nezávislé. Pak existuje $u^* \in \mathbb{R}^{m_h}$ a $v^* \in \mathbb{R}^{m_g}_+$ takové, že bod (x^*, u^*, v^*) splňuje:

(i) $h_i(x^*) = 0$ *pro* $i = 1, 2, \ldots, m_h$, $g_j(x^*) \leq 0$ *pro* $j = 1, 2, \ldots, m_g$,

(ii) $v^* \geq 0$, $v_j^* g_j(x^*) = 0$ *pro* $j = 1, 2, \ldots, m_g$,

(iii) $\nabla_x L(x^*, u^*, v^*) = 0$.

Kapitola 3

Metody nepodmíněné optimalizace

V úvodní kapitole jsme naznačili základní myšlenku, jak převést úlohy podmíněné optimalizace na posloupnost úloh nepodmíněné optimalizace. Tato kapitola bude věnována algoritmům pro řešení jedné úlohy z této posloupnosti.

Mějme funkci $F : \mathbb{R}^n \to \mathbb{R}$ třídy $\mathcal{C}^2$, naši úlohu můžeme zformulovat jako

$$\min F(x) \qquad \text{přes všechna} \quad x \in \mathbb{R}^n. \tag{3.1}$$

Na její řešení použijeme některou z iteračních metod. Přibližné řešení v i-tém kroku označíme jako x^i. To prohlásíme za řešení úlohy, splní-li požadované konvergenční kritérium, zpravidla použijeme

$$\| \nabla F(x^i) \| \leq \varepsilon,$$

kde $\varepsilon > 0$ je předem daná konstanta.

Pro zjednodušení zavedeme následující značení:

$$F^i = F(x^i), \quad g^i = \nabla F(x^i), \quad H^i = \nabla^2 F(x^i),$$

přičemž indexy i budeme vynechávat v místech, kde bude dostatečně zřejmé, o jakou iteraci se jedná.

3.1 Metoda spádových směrů

Metoda *spádových směrů (line-search method)* patří mezi základní optimalizační metody pro nelineární funkce. Její podstatu můžeme shrnout do následujícího obecného algoritmu:

Algoritmus 3.1 (Metoda spádových směrů).
Dáno x^0.
Pro $i = 0, 1, 2, \ldots$

(i) Najdi směr poklesu s^i pro funkci F v bodě x^i.

(ii) Zvol „vhodnou" délku kroku α^i.

(iii) Polož $x^{i+1} = x^i + \alpha^i s^i$.

(iv) Zkontroluj konvergenční kritérium.

Nabízí se dvě přirozené otázky – jak efektivně volit směr poklesu s^i a délku kroku α^i. Při výběru délky kroku jde o úlohu jednorozměrné minimalizace funkce $F(x^i + \alpha s^i)$ pro $\alpha > 0$. Existenci takového α nám zajišťuje požadavek, aby s^i byl směrem poklesu funkce F v bodě x^i. Bylo by zbytečné a příliš časově náročné řešit tento problém přesně. Existuje řada podmínek na délku kroku, při jejichž splnění máme zaručen[1] dostatečný pokles účelové funkce pro důkaz globální konvergence metody spádových směrů. Jmenujme například *Goldsteinovu* nebo *silnou a slabou Wolfeho podmínku*, bližší podrobnosti lze najít v [5, 19].

Je celá řada strategií, jak volit směr poklesu s^i. Za dvě nejznámější lze považovat

- $s^i = -g^i$ *metoda největšího spádu,*
- $s^i = -(H^i)^{-1} g^i$ *Newtonova metoda.*[2] (3.2)

My se soustředíme na odlišný způsob volby směru s^i – budeme ho hledat jen jako přibližné řešení soustavy

$$H^i s = -g^i \tag{3.3}$$

pomocí metod *CGM* a *QMR*.

Detailnější informace o metodě spádových směrů lze najít např. v knize [19].

3.1.1 Metody Newtonova typu

Víme-li, že funkce F je konvexní, je volba směru s^i podle (3.2) plně ospravedlněna. Máme totiž zaručenou pozitivní definitnost hessiánu H v každém bodě a tím i to, že takto vybraný směr je vždy spádový, což lehce dokážeme:

$$(g^i)^T s^i = (g^i)^T \left(-(H^i)^{-1} g^i\right) = -(g^i)^T (H^i)^{-1} g^i < 0\,.$$

Pro výpočet s^i se většinou používá Choleského, popř. řídkého Choleského rozkladu, algoritmus (3.1) tak můžeme přepsat do následujícího tvaru.

[1] „Vhodná" délka kroku nestačí, je potřeba požadovat i některou podmínku na dostatečně dobrý směr.

[2] Zde se dopouštíme drobné nepřesnosti – při této volbě směru máme jen pro konvexní úlohy zaručeno, že půjde opravdu o směr poklesu, nicméně pro nekonvexní úlohy lze zavést *modifikovanou Newtonovu metodu* s podobným výběrem směru. Dohromady budeme Newtonovu a modifikovanou Newtonovu metodu označovat jako *metody Newtonova typu*. Blíže v kapitole 3.1.1.

Algoritmus 3.2 (Newtonova metoda s omezeným krokem).
Dáno x^0.
Pro $i = 0, 1, 2, \ldots$

(i) Pomocí Choleského rozkladu vyřeš soustavu $H^i s^i = -g^i$ pro s^i:

- Rozlož $H^i = LL^T$
- Vyřeš soustavu $Ly = -g^i$ pro y
- Vyřeš soustavu $L^T s^i = y$ pro s^i

(ii) Zvol „vhodnou" délku kroku α^i

(iii) Polož $x^{i+1} = x^i + \alpha^i s^i$

(iv) Zkontroluj konvergenční kritérium.

V obecném případě, kde může být funkce F nekonvexní, se pochopitelně může stát, že hessián H nebude pozitivně definitní a Choleského rozklad zhavaruje. V takovém případě se využije následující modifikace. Nahradíme původní matici soustavy H novou (již pozitivně definitní) maticí H^*:

$$H^* = H + \omega I,$$

kde I značí jednotkovou matici a $\omega \in \mathbb{R}$ splňuje podmínku $\omega > -\lambda_{min}$, kde λ_{min} je nejmenší vlastní číslo hessiánu H. Tato metoda se nazývá *modifikovaná Newtonova metoda*, viz [14, 19].

Zbývá otázka, jak levně odhadnout λ_{min}. K tomu lze použít následující heuristiku. Pokaždé, ať s konvexní či nekonvexní funkcí F, se pokusíme rozložit matici H Choleského rozkladem. Pokud se to podaří, pokračujeme v algoritmu 3.2 běžným způsobem. V opačném případě položíme $\omega = \omega_0$, kde $\omega_0 > 0$ je předem pevně zvolený parametr, a provedeme druhý pokus s maticí $H^* = H + \omega I$. Pokud ani tentokrát neuspějeme, zvětšíme ω dvojnásobně a celý postup opakujeme tak dlouho, dokud se matice nepodaří rozložit. Jestliže se matice H^* podařila rozložit už při volbě ω_0, byl zvolen parametr ω zbytečně velký a budeme ho podobným způsobem snižovat. Tím máme zaručeno, že $\omega \in [-\lambda_{min}, -2\lambda_{min}]$.

Obecně lze říci, že Newtonova, resp. modifikovaná Newtonova metoda dosahuje při řešení úloh (3.1) velmi dobrých výsledků. Navíc díky tomu, že je již implementována v programu PENNON, v rámci kterého budeme testovat další metody, nám poslouží jako dobré srovnávací kritérium při vyhodnocování kvality metod.

3.1.2 Metoda sdružených gradientů

Metoda *sdružených gradientů (conjugate gradient method, CGM)* se řadí mezi nejznámější iterační metody pro řešení soustav lineárních rovnic se symetrickou pozitivně definitní maticí. My tuto metodu použijeme pro přibližné řešení soustavy (3.3). Pro zaručení pozitivní definitnosti je nutné omezit se jen na konvexní funkce F.

Předpokládejme, že jsme v i-té iteraci algoritmu 3.1 a hledáme směr poklesu s^i. Iterační posloupnost přibližných řešení s^i označíme s^i_j pro $j = 0, 1, 2, \ldots$ Algoritmus CGM nastartujeme typicky s $s^i_0 = 0$ a iterace přerušíme při splnění kritéria

$$\|r_j\| \,/\, \|g^i\| \leq \varepsilon,$$

kde $r_j = Hs^i_j + g$ značí reziduum v kroku j a $\varepsilon > 0$ je pevně zvolený parametr, zpravidla nastavený na 0,01. V dalším budeme indexy i pro přehlednost vynechávat.

V této práci uvedeme jen základy metody CGM, znění konvergenčních vět a samotný algoritmus. Podrobnosti lze najít v [1, 19].

Metoda CGM je založena na následující větě, pomocí které převedeme řešení soustavy na minimalizaci kvadratického funkcionálu:

Tvrzení 3.1. *Nechť $H \in \mathbb{S}^n$, $H \succeq 0$ a nechť $\phi(s) = \frac{1}{2}s^T Hs + g^T s$. Potom jsou následující dva problémy ekvivalentní:*

(i) $Hs = -g$

(ii) $\min_{s \in \mathbb{R}^n} \phi(s)$.

V každé iteraci metody najdeme nové přibližné řešení s_{j+1} jako minimum funkcionálu $\phi(s)$ na stále se rozšiřující množině $s_0 + \mathcal{K}_{j+1}(r_0, H)$. Díky speciální volbě báze pro Krylovův prostor $\mathcal{K}_{j+1}(r_0, H)$ postačí hledat s_{j+1} jako minimum $\phi(s)$ v jednom směru, směru p_j, tj. minimalizujeme $\phi(s_j + \alpha p_j)$ pro α. Vektory $p_0, p_1, \ldots, p_j$ tvoří bázi pro $\mathcal{K}_{j+1}(r_0, H)$ a generujeme je průběžně tak, aby splňovaly podmínku:

$$p_k^T H p_j = 0 \qquad \text{pro } k = 0, 1, \ldots, j-1,$$

tuto podmínku nazveme podmínka *sdružených směrů*, též *H-ortogonalita.*

Celkový algoritmus lze zapsat takto:

Algoritmus 3.3 (CGM).
Dáno s_0. Polož $r_0 = Hs_0 + g, p_0 = -r_0$.
Pro $j = 0, 1, 2, \ldots, n$

(i) $\alpha_j = \dfrac{r_j^T r_j}{p_j^T H p_j}$

(ii) $s_{j+1} = s_j + \alpha_j p_j$

(iii) $r_{j+1} = r_j + \alpha_j H p_j$

(iv) $\beta_{j+1} = \dfrac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$

(v) $p_{j+1} = -r_{j+1} + \beta_{j+1} p_j$

(vi) Zkontroluj konvergenční kritérium.

Konvergenční teorie metody CGM

Zde uvedeme jen dvě základní konvergenční věty. Podrobnosti o chování metody a důkazy zde citovaných vět lze najít v [16].

Věta 3.2. *Jestliže má matice $H \in \mathbb{S}^n$ jen $r \leq n$ různých vlastních čísel, skončí metoda CGM (v přesné aritmetice) po nejvýše r krocích.*

Věta 3.3. *Nechť s_j je j-tá aproximace, kterou získáme metodou CGM, a nechť s^* značí přesné řešení soustavy (3.3). Pak platí*

$$\|s_j - s^*\|_H \leq 2\left(\frac{\sqrt{\kappa(H)}-1}{\sqrt{\kappa(H)}+1}\right)^j \|s_0 - s^*\|_H,$$

kde $\kappa(H)$ značí číslo podmíněnosti matice H.

Předpodmínění metody

Z konvergenčních vět je vidět, že rozložení vlastních čísel matice H velmi ovlivňuje chování metody CGM. Vzniká tak přirozená otázka, zda by nešlo toto rozložení nějak zlepšit, a tím zrychlit konvergenci metody CGM.

Mějme matici $M \in \mathbb{S}^n, M \succeq 0$. Pak můžeme nahradit soustavu (3.3) novou soustavou:

$$M^{-\frac{1}{2}} H M^{-\frac{1}{2}} M^{\frac{1}{2}} s = -M^{-\frac{1}{2}} g. \tag{3.4}$$

Nová matice soustavy $\tilde{H} = M^{-\frac{1}{2}} H M^{-\frac{1}{2}}$ zůstane symetrická pozitivně definitní a pokud bylo M zvoleno vhodně, zlepší se číslo podmíněnosti a rozložení vlastních čísel. Aplikací metody CGM na soustavu (3.4) a drobnou úpravou získáme následující algoritmus:

Algoritmus 3.4 (PCGM).
Dáno s_0. Polož $r_0 = Hs_0 + g, y_0 = M^{-1}r_0, p_0 = -y_0$.
Pro $j = 0, 1, 2, \ldots$

(i) $\alpha_j = \dfrac{r_j^T y_j}{p_j^T H p_j}$

(ii) $s_{j+1} = s_j + \alpha_j p_j$

(iii) $r_{j+1} = r_j + \alpha_j H p_j$

(iv) $y_{j+1} = M^{-1} r_{j+1}$

(v) $\beta_{j+1} = \dfrac{r_{j+1}^T y_{j+1}}{r_j^T y_j}$

(vi) $p_{j+1} = -y_{j+1} + \beta_{j+1} p_j$

(vii) Zkontroluj konvergenční kritérium.

3.1.3 Lanzcosův proces

Lanzcosův proces je, podobně jako H-ortogonální směry u metody sdružených gradientů, další z možností, jak generovat bázi pro Krylovův prostor. Tento způsob může být vhodný v řadě algoritmů a teorií. Nás bude zajímat využití Lanczosova procesu v metodách pro řešení soustav lineárních rovnic, konkrétně v metodě QMR, o které se zmíníme vzápětí.

Existuje poměrně velký počet modifikací Lanzcosova procesu. Jde především o různá rozšíření a vylepšení původního algoritmu, například rozšíření pro nesymetrické matice, strategii nazvanou *look-ahead* pro omezení havárií metody a numerickou stabilizaci označovanou jako *coupled two-term recurrences*. Nejdříve odvodíme základní verzi pro symetrické matice, o některých vylepšeních se zmíníme později.

Místo prve jmenovaných H-ortogonálních vektorů budeme sestavovat bázi pro $\mathcal{K}_j(r_0, H)$ z *ortonormálních* vektorů; označíme je v_j pro $j = 1, 2, 3, \ldots$ Naše požadavky shrňme do následujících dvou bodů. Nechť platí pro všechna j a k:

$$\mathcal{K}_j(r_0, H) = \mathcal{L}\{v_1, v_2, \ldots, v_j\}, \tag{3.5}$$

$$v_j^T v_k = \begin{cases} 0 & \text{pro } j \neq k, \\ 1 & \text{pro } j = k. \end{cases} \tag{3.6}$$

Na jejich základě se pokusíme odvodit algoritmus jak bázi generovat. Okamžitě z (3.5) plyne, jak můžeme zvolit první vektor báze v_1:

$$v_1 = r_0/\rho_1, \quad \rho_1 = \|r_0\|. \tag{3.7}$$

Lze rovněž lehce ukázat, že každý další vektor báze by měl splňovat podmínku

$$v_{j+1} \in \mathcal{L}\{v_1, v_2, \ldots, v_j, Hv_j\}. \tag{3.8}$$

Hledejme vektor $\tilde{v}_{j+1}$, který by vyhovoval (3.5) a místo (3.6) požadujme jen ortogonalitu ke všem předchozím vektorům, tj. $v_i^T \tilde{v}_{j+1} = 0$ pro $i = 1, 2, \ldots, j$. Na základě (3.8) volme tvar $\tilde{v}_{j+1}$ takto:

$$\tilde{v}_{j+1} = Hv_j + \sum_{k=1}^{j} \lambda_k v_k.$$

Z požadované ortogonality vůči v_i pro $i = 1, 2, \ldots, j$ máme

$$0 = v_i^T \tilde{v}_{j+1} = v_i^T H v_j + \sum_{k=1}^{j} \lambda_k v_i^T v_k = v_i^T H v_j + \lambda_i = (Hv_i)^T v_j + \lambda_i$$

a odtud s uvážením, že $Hv_i \in \mathcal{K}_{i+1}(r_0, H) = \mathcal{L}\{v_1, v_2, \ldots, v_{i+1}\}$ a že v_j je ortonormální vůči ostatním v_i, zjistíme

$$\tilde{v}_{j+1} = Hv_j + \lambda_j v_j + \lambda_{j-1} v_{j-1}. \tag{3.9}$$

Označme $\beta_{j+1} = \|\tilde{v}_{j+1}\|$: pak můžeme nový vektor do báze Krylovova podprostoru $\mathcal{K}_{j+1}(r_0, H)$ volit jako $v_{j+1} = \tilde{v}_{j+1}/\beta_{j+1}$. Přepsáním (3.9) získáváme

$$\tilde{v}_{j+1} = \beta_{j+1} v_{j+1} = H v_j + \lambda_j v_j + \lambda_{j-1} v_{j-1}$$

a odtud vynásobením celého vztahu v_{j+1} můžeme β_{j+1} vyjádřit jako

$$\beta_{j+1} v_{j+1}^T v_{j+1} = v_{j+1}^T H v_j + \lambda_j v_{j+1}^T v_j + \lambda_{j-1} v_{j+1}^T v_{j-1} = v_{j+1}^T H v_j.$$

Přeznačíme-li $\alpha_j := -\lambda_j = v_j^T H v_j$, dostáváme s přihlédnutím na dříve vypočtený tvar $\beta_j = -\lambda_{j-1}$ finální podobu tříčlenné rekurence pro vektory v_j pro $j = 1, 2, \ldots$ (zaveďme $v_0 := 0$):

$$\beta_{j+1} v_{j+1} = H v_j - \alpha_j v_j - \beta_j v_{j-1} \tag{3.10}$$

Pro větší přehlednost přepišme (3.10) do maticového tvaru

$$H V_j = V_j T_j + [0 \cdots 0 \; \beta_{j+1} v_{j+1}], \tag{3.11}$$

kde $V_j = [v_1 \; v_2 \; \cdots \; v_j] \in \mathbb{R}^{n \times j}$ a $T_j \in \mathbb{R}^{j \times j}$ je třídiagonální matice tvaru

$$T_j = \begin{pmatrix} \alpha_1 & \beta_2 & & \cdots & 0 \\ \beta_2 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \alpha_{j-1} & \beta_j \\ 0 & \cdots & & \beta_j & \alpha_j \end{pmatrix}.$$

Někdy též budeme používat

$$H V_j = V_{j+1} T_j^+, \tag{3.12}$$

kde $T_j^+ \in \mathbb{R}^{j+1 \times j}$ značí rozšířený tvar matice T_j

$$T_j^+ = \begin{pmatrix} T_j \\ \varrho_j^T \end{pmatrix}, \quad \text{kde} \quad \varrho_j = [0 \cdots 0 \; \beta_{j+1}]^T \in \mathbb{R}^j.$$

Lanczosův proces pro nesymetrické matice

V případě, že je matice soustavy nesymetrická[3], zvolíme podobnou strategii jako v metodě *BiCGM (Biconjugate gradient method)* – budeme generovat báze pro dva Krylovovy podprostory, jeden generovaný maticí H a vektorem r_0, druhý maticí H^T a vektor vybereme libovolně.

Volme vektor v_1 stejně jako v (3.7) a vektor $w_1 \in \mathbb{R}^n$ libovolně tak, aby splňoval $\|w_1\| = 1$. Lanczosův proces bude generovat posloupnosti vektorů v_i, w_i

$$\|v_i\| = \|w_i\| = 1 \quad \text{pro} \quad i = 1, 2, \ldots$$

[3]V naší situaci má smysl o tomto případu hovořit při použití nesymetrického předpodmiňovače.

takové, že platí

$$\mathcal{K}_j(r_0, H) = \mathcal{L}\{v_1, v_2, \ldots, v_j\} \tag{3.13}$$

$$\mathcal{K}_j(w_1, H^T) = \mathcal{L}\{w_1, w_2, \ldots, w_j\} \tag{3.14}$$

a navíc že jsou navzájem biortogonální, tj. splňují

$$W_j^T V_j = D_j, \tag{3.15}$$

kde D_j je diagonální matice a W_j a V_j jsou matice zavedené podobně jako v předchozí části, tj.

$$D_j = \operatorname{diag}(\delta_1, \delta_2, \ldots, \delta_j), \quad \text{kde} \quad \delta_i = w_j^T v_j, \tag{3.16}$$

$$W_j = [\, w_1 \; w_2 \; \cdots \; w_j] \in \mathbb{R}^{n \times j} \quad \text{a} \quad V_j = [\, v_1 \; v_2 \; \cdots \; v_j\,] \in \mathbb{R}^{n \times j}. \tag{3.17}$$

Obdobně jako v části pro symetrické matice lze odvodit vztahy

$$\begin{aligned}
\tilde{v}_{j+1} &= H v_j - \mu_j v_j - \nu_j v_{j-1}, \\
\rho_{j+1} &= \|\tilde{v}_{j+1}\|, \quad v_{j+1} = \tilde{v}_{j+1}/\rho_{j+1}, \\
\tilde{w}_{j+1} &= H^T w_j - \mu_j w_j - (\nu_j \rho_j/\xi_j) w_{j-1}, \\
\xi_{j+1} &= \|\tilde{w}_{j+1}\|, \quad w_{j+1} = \tilde{w}_{j+1}/\xi_{j+1}, \\
\text{kde} \quad \mu_j &= w_j^T H v_j / w_j^T v_j, \quad \nu_j = \xi_j w_j^T v_j / w_{j-1}^T v_{j-1},
\end{aligned}$$

a zapsat je do maticového tvaru

$$\begin{aligned}
H V_j &= V_{j+1} T_j^+, \\
H^T W_j &= W_{j+1} \Gamma_{j+1}^{-1} T_j^+ \Gamma_{j+1},
\end{aligned} \tag{3.18}$$

kde $\Gamma_j \in \mathbb{R}^{j \times j}$ značí

$$\Gamma_j = \operatorname{diag}(\gamma_1, \gamma_2, \ldots, \gamma_j), \quad \text{kde} \quad \gamma_i = \begin{cases} 1 & \text{pro } i = 1, \\ \gamma_{i-1}\rho_i/\xi_i & \text{pro } i > 1, \end{cases}$$

a $T_j^+ \in \mathbb{R}^{j+1 \times j}$ je opět třídiagonální matice tvaru

$$T_j^+ = \begin{pmatrix}
\mu_1 & \nu_2 & & \cdots & 0 \\
\rho_2 & \mu_2 & \ddots & & \vdots \\
 & \ddots & \ddots & \ddots & \\
 & & \ddots & \mu_{j-1} & \nu_j \\
\vdots & & & \rho_j & \mu_j \\
0 & \cdots & & 0 & \rho_{j+1}
\end{pmatrix}.$$

Na závěr této části poznamenejme, že všechny mimodiagonální prvky matice H_j^+ jsou nenulové. Odtud plyne, že hodnost matice H_j^+ je právě j, což se nám bude velmi hodit při odvozování metody QMR.

Lanczosův proces se strategií „look-ahead"

Z rekurentních vztahů z předchozí části můžeme pozorovat, že je třeba v každé iteraci dělit výrazem $w_j^T v_j$. Pokud by nastalo

$$w_j^T v_j = 0, \qquad \text{případně jen} \qquad w_j^T v_j \approx 0, \tag{3.19}$$

vynutí se tím ukončení algoritmu. Poznamenejme, že nás znepokojuje jen případ (3.19) za podmínky

$$v_j \not\approx 0 \qquad \text{a zároveň} \qquad v_j \not\approx 0.$$

V opačném případě bychom našli H, resp. H^T invariantní podprostor a Lanczosova metoda by pro něj vygenerovala bázi.

Strategie look-ahead je účinnou technikou, jak předcházet podobným případům. Myšlenka spočívá v uvolnění podmínky (3.15). Místo diagonální matice bude D_j jen blokově diagonální s l_j čtvercovými bloky velikostí h_i, $(i = 1, 2, \ldots, l_j)$, přesněji to můžeme zapsat

$$D_j = \operatorname{diag}(D^{(1)}, D^{(2)}, \ldots, D^{(l_j)}), \quad \text{kde} \quad D^{(i)} = (W^{(i)})^T V^{(i)}.$$

Matice $V^{(i)}$ a $W^{(i)}$ pro $i = 1, 2, \ldots, l_j$ značí rozdělení V_j a W_j do bloků velikostí h_i, tj.

$$V_j = [\, V^{(1)}\ V^{(2)}\ \ldots\ V^{(l_j)} \,] \quad \text{a podobně} \quad W_j = [\, W^{(1)}\ W^{(2)}\ \ldots\ W^{(l_j)} \,].$$

Matice T_j tak také nebude třídiagonální, ale blokově třídiagonální. V praxi se ukazuje, že ve skutečnosti není „pravých" look-ahead kroků (to jsou ty s velikostí $h_i > 1$) potřeba mnoho a většinou tak metoda degeneruje na původní případ.

Bližší informace o Lanczosově metodě lze najít v [8, 9, 12].

3.1.4 Metoda QMR

Metoda *QMR (quasi-minimal residual method)* spadá opět do třídy metod tzv. *Krylovova typu*, tj. metod, které hledají řešení v j-té iteraci na množině

$$s_0 + \mathcal{K}_j(r_0, H). \tag{3.20}$$

Algoritmy těchto metod můžeme zpravidla rozdělit na dvě části – nalezení báze pro Krylovův prostor $\mathcal{K}_j(r_0, H)$ a „výběr" iterace. V případě metody QMR se pro generování báze používá Lanczosův proces, případně nějaká jeho modifikace. Výběr iterace z množiny (3.20) objasníme nyní.

Nechť V_j značí v souladu s předchozí částí vygenerovanou bázi Krylovova prostoru. Hledání iterace na (3.20) převedeme na hledání $z_j \in \mathbb{R}^j$ pro

$$s_j = s_0 + V_j z_j. \tag{3.21}$$

Reziduum v j-tém kroku lze s využitím (3.21) zapsat ve tvaru

$$r_j = H s_j + g = H V_j z_j + H s_0 + g = H V_j z_j + r_0.$$

Uvažme navíc speciální volbu v_1 v (3.7) a třídiagonalizaci matice (3.12), resp. (3.18) při použití nesymetrického Lanczosova procesu. Dostáváme

$$\begin{aligned} r_j &= HV_j z_j + \rho_1 v_1 = V_{j+1} T_j^+ z_j + \rho_1 v_1 = V_{j+1} \left(T_j^+ z_j + f_{j+1} \right) \\ &= V_{j+1} \Omega_{j+1}^{-1} \left[\Omega_{j+1} \left(T_j^+ z_j + f_{j+1} \right) \right]. \end{aligned} \tag{3.22}$$

kde $f_{j+1} = [\rho_1\ 0\ \cdots\ 0] \in \mathbb{R}^{j+1}$ a $\Omega_{j+1} \in \mathbb{R}^{(j+1)\times(j+1)}$ je váhová matice

$$\Omega_{j+1} = \operatorname{diag}(\omega_1, \omega_2, \ldots, \omega_{j+1}), \quad \omega_j > 0, \quad j = 1, 2, \ldots, j+1.$$

Jak tedy najít $z_j \in \mathbb{R}^j$? Požadavek na minimalizaci rezidua, tj.

$$\left\| V_{j+1} \left(T_j^+ z_j + f_{j+1} \right) \right\| = \min_{z \in \mathbb{R}^j} \left\| V_{j+1} \left(T_j^+ z + f_{j+1} \right) \right\|.$$

by byl velmi neekonomický. Vedl by totiž ke složitosti řádu $O(nj^2)$. Místo toho budeme požadovat po z_j slabší podmínku, jen *quasi*-minimum rezidua, tj.

$$\left\| \Omega_{j+1} \left(T_j^+ z_j + f_{j+1} \right) \right\| = \min_{z \in \mathbb{R}^j} \left\| \Omega_{j+1} \left(T_j^+ z + f_{j+1} \right) \right\|. \tag{3.23}$$

Tento problém se řeší pomocí QR rozkladu matice $\Omega_{j+1} T_j^+$. Je tak zaručena levná aktualizace při kroku $z_j \to z_{j+1}$. Detailní algoritmus pro řešení (3.23) lze nalézt v [8]. Připomeňme, že díky plné hodnosti matice T_j^+ má problém (3.23) vždy jednoznačné řešení.

Metoda QMR je velmi úzce spjata s metodou BiCGM, avšak na rozdíl od ní má díky své podmínce výrazně hladší průběh reziduí. To ostatně uvidíme v kapitole numerických výsledků.

Předpodmínění metody QMR

Podobně jako v metodě CGM je i v této metodě naprosto klíčové využít v hůře podmíněných příkladech nějakého vhodného předpodmiňovače. Předpokládejme, že matice $M \in \mathbb{R}^{n\times n}$ je regulární a v nějakém smyslu dobrá aproximace matice H. Uvažujme rozklad

$$M = M_1 M_2.$$

Potom můžeme místo soustavy (3.3) se startem v bodě s_0 vyřešit předpodmíněnou soustavu

$$\bar{H} y = -\bar{g}, \quad \text{kde} \quad \bar{H} = M_1^{-1} H M_2^{-1}, \quad \bar{g} = M_1^{-1} (H s_0 + g), \quad y = M_2 (s - s_0),$$

a přetransformovat řešení z nové soustavy zpět pomocí

$$s_n = s_0 + M_2^{-1} y_n, \quad r_n = M_1 \bar{r}_n.$$

Toto je obecně použitelný postup.

Implementace

Bližší informace lze nalézt v [8, 9, 10, 11]. V praxi jsme se zabývali dvěma různými variantami. První z nich byla profesionální implementace metody QMR ve Fortranu 77. Jednalo se o variantu s look-aheadem pro symetrické matice z balíku QMRPACK, [11]. Ten je volně dostupný na

`http://www.netlib.org/linalg/qmrpack.tgz`.

Druhou variantou byla vlastní implementace podle návrhu v [2]. Tato verze následuje:

Algoritmus 3.5 (QMR pro sym. matici i předpodmiňovač).
Dáno s_0. Polož $r_0 = Hs_0 + g, \tilde{v}_1 = -r_0, y = M_1^{-1}\tilde{v}_1, \rho_1 = \|y\|_2$,
$\gamma_0 = \epsilon_0 = 1, \theta_0 = 0, \eta_0 = -1, p_0 = d_0 = t_0 = 0$.
Pro $j = 1, 2, 3, \ldots$

(i) $v_j = \tilde{v}_j/\rho_j, y = y/\rho_j, \tilde{y} = M_2^{-1}y$

(ii) $p_j = \tilde{y} - (\rho_j/\epsilon_{j-1})p_{j-1}$

(iii) $\tilde{p} = Hp_j, \epsilon_j = p_j^T\tilde{p}$, pokud $\epsilon_j = 0$ stop

(iv) $\tilde{v}_{j+1} = \tilde{p} - \epsilon_j v_j$

(v) $y = M_1^{-1}\tilde{v}_{j+1}, \rho_{j+1} = \|y\|_2$, pokud $\rho_j = 0$ stop

(vi) $\theta_j = \rho_{j+1}/(\gamma_{j-1}|\epsilon_j|), \gamma_j = 1/\sqrt{1+\theta_j^2}$, pokud $\gamma_j = 0$ stop

(vii) $\eta_j = -\eta_{j-1}\rho_j\gamma_j^2/(\epsilon_j\gamma_{j-1}^2)$

(viii) $d_j = \eta_j p_j + (\theta_{j-1}\gamma_j)^2 d_{j-1}, t_j = \eta_j\tilde{p} + (\theta_{j-1}\gamma_j)^2 t_{j-1}$

(ix) $s_j = s_{j-1} + d_j, r_j = r_{j-1} + t_j$

(x) Zkontroluj konvergenční kritérium.

3.2 Metoda s lokálně omezeným krokem

Na rozdíl od metody spádových směrů, kde jsme nejdříve určili směr poklesu a pak jsme podél tohoto směru hledali vhodnou délku kroku, se v metodě *s lokálně omezeným krokem (trust region method, TRM)* postupuje „naopak".

V každém iteračním bodě x^i sestavíme na základě lokálních informací o funkci F v bodě x^i *model, modelovou funkci* – funkci, která by měla co nejlépe vystihovat vlastnosti původní funkce F na okolí bodu x^i. Zároveň stanovíme oblast, označíme ji $\mathcal{B}^i$, ve které budeme předpokládat dostatečnou podobnost modelu a původní úče-

lové funkce.[4] Poté najdeme na modelu m^i v oblasti $\mathcal{B}^i$ přibližné minimum, což můžeme zapsat jako

$$\min m^i(x^i + s) \qquad \text{na množině } \{s \in \mathbb{R}^n \mid x^i + s \in \mathcal{B}^i\}; \tag{3.24}$$

označme ho $x^i + s^i$. Pokud v tomto bodě nenastane předpokládaný pokles funkce F, znamená to, že oblast $\mathcal{B}^i$ je příliš velká, model a skutečná funkce si moc neodpovídají, a je tedy potřeba oblast v příštím kroku zmenšit. V opačném případě se posuneme do nového bodu ($x^{i+1} = x^i + s^i$) a oblast můžeme dle uvážení ještě rozšířit.

Jako modelovou funkci v bodě x^i budeme používat Taylorův rozvoj 2. řádu:

$$m^i(x^i + s) = F^i + (g^i)^T s + \frac{1}{2}\, s^T H^i s. \tag{3.25}$$

Pro měření velikosti oblasti zavedeme *M-normu*:

$$\|s\|_M^2 = s^T M s,$$

kde $M \in \mathbb{S}^n, M \succeq 0$ je nějaká snadno invertovatelná aproximace matice H, a oblast $\mathcal{B}^i$ definujme jako:

$$\mathcal{B}^i = \left\{x^i + s \mid \ \|s\|_M \leq \Delta^i\right\},$$

pro nějaké $\Delta^i > 0$.

Použití M-normy namísto běžné euklidovské normy upraví tvar oblasti $\mathcal{B}^i$, aby vyhovoval lépe geometrii příkladu. Jde vlastně o přeškálování měřítek – ostatně jako v případě předpodmínění metod pro řešení soustav lineárních rovnic.

Kromě otázek, jak měřit úspěšnost modelu či jak měnit velikost oblasti $\mathcal{B}^i$, na něž nalezneme odpověď v sestaveném algortimu, se nabízí ještě jedna, patrně nejpalčivější otázka – jak vyřešit problém (3.24).

Velmi podrobně je metoda s lokálně omezenou metrikou popsána v [7], včetně řady algoritmů, jak minimalizaci modelu provést. Zde se zmíníme jen o dvou z nich, metodě Cauchyho bodu a metodě Steihauga-Tointa. Z ostatních způsobů připomeňme velmi často užívané řešení; jde o přesné řešení minima modelu (3.24) pomocí Choleského rozkladu, viz [7].

Nyní můžeme sestavit algoritmus TRM.

[4] Můžeme říct, že „důvěřujeme" shodě modelu a původní funkce na této oblasti. Odtud pochází název metody – trust-region.

Algoritmus 3.6 (TR metoda).
Dáno x^0, Δ^0. Zvol konstanty $\eta_1, \eta_2, \gamma_1, \gamma_2$ tak, aby platilo:

$$0 < \eta_1 \leq \eta_2 < 1 \qquad \text{a} \qquad 0 < \gamma_1 < 1 \leq \gamma_2.$$

Prováděj pro $i = 0, 1, 2, \ldots$, dokud není splněno konvergenční kriterium:

(i) Definice modelu

- Urči předpodmiňovač M^i
- $m^i(x^i + s) := F^i + (g^i)^T s + \frac{1}{2}\, s^T H^i s$

(ii) Výpočet minima modelu a ověření kvality nového bodu

- $s^i \approx \arg\min\{m^i(x^i + s) \mid s \in \mathbb{R}^n, \|s\|_{M^i} \leq \Delta^i\}$
- $\rho^i = \dfrac{F(x^i) - F(x^i + s^i)}{m^i(x^i) - m^i(x^i + s^i)}$

(iii) Aktualizace polohy a velikosti oblasti

- $x^{i+1} = \begin{cases} x^i + s^i & \text{pokud} \quad \rho^i \geq \eta_1 \\ x^i & \text{jinak} \end{cases}$
- $\Delta^{i+1} = \begin{cases} \gamma_2 \Delta^i & \rho^i \geq \eta_2 \\ \Delta^i & \text{pokud} \quad \rho^i \in [\eta_1, \eta_2) \\ \gamma_1 \Delta^i & \rho^i < \eta_1 \end{cases}$

3.2.1 Cauchyho bod

Jedna z možností, jak minimalizovat alespoň přibližně problém (3.24), je použít tzv. *Cauchyho bod*. Je to bod, který dostaneme minimalizací modelu (3.25) na oblasti $\mathcal{B}^i$ ve směru záporného gradientu. Při použití euklidovské normy místo M-normy může být definován Cauchyho bod následovně:

$$x_c^{i+1} = x^i + s_c^i, \quad \text{kde } s_c^i = -\gamma g^i,$$
$$\gamma = \begin{cases} \Delta^i & \text{pro } (g^i)^T H^i g^i \leq 0, \\ \min\left(\Delta^i, \frac{(g^i)^T g^i}{(g^i)^T H^i g^i}\right) & \text{jinak.} \end{cases} \tag{3.26}$$

Je zřejmé, že použitím Cauchyho bodu získáme metodu největšího spádu s omezeným krokem.

Cauchyho bod není ani tak moc důležitý v praxi (vede obyčejně k pomalé konvergenci), jako spíš z teoretického hlediska. Je totiž dokázáno ([7]), že při volbě minima modelu alespoň tak dobrého, jako je Cauchyho bod, máme zaručenou globální konvergenci algoritmu 3.6.

3.2.2 Steihaug-Toint – metoda pro minimalizaci modelu

Metoda *Steihauga-Tointa* je další způsob, jak nalézt přibližné řešení problému (3.24). Myšlenka spočívá v použití metody sdružených gradientů. Tu ovšem nelze aplikovat v takové podobě, jakou jsme zavedli v kapitole 3.1.2. Tam jsme předpokládali, že metoda CGM (resp. PCGM) bude sloužit pro nalezení minima ryze konvexního kvadratického funkcionálu na neomezené oblasti, zatímco v úloze (3.24) může být funkcionál obecně nekonvexní a problém řešíme jen na oblasti $\mathcal{B}^i$.

Co se tedy může stát? Možnosti jsou následující:

- Metoda proběhne bez komplikací a nalezne přibližné minimum $x^i + s^i$ modelu m^i uvnitř oblasti $\mathcal{B}^i$.
- Metoda v některé iteraci narazí na záporné zakřivení modelu ($p_j^T H^i p_j \leq 0$ pro nějaké j) a zhavaruje.
- Metoda během některé iterace překročí hranici oblasti $\mathcal{B}^i$.

Prvním případ je ideální, problém (3.24) je vyřešen přesně. Ve druhém případě jsme narazili na takový směr p_j, ve kterém je funkcionál m^i zdola neomezený na přímce $s_j + \alpha p_j$. Je proto rozumné posunout se až na hranici oblasti $\mathcal{B}^i$, tj. volit takové $\tilde{\alpha}$, pro které platí:

$$\|s_j + \tilde{\alpha} p_j\|_M = \Delta^i. \tag{3.27}$$

Před ošetřením posledního případu uveďme následující tvrzení, důkaz lze najít v [7].

Tvrzení 3.4. *Předpokládejme, že je algoritmus metody PCGM aplikován na minimalizaci modelu $m^i(x^i + s)$, se startem $s_0 = 0$. Dále předpokládejme, že pro $0 \leq j \leq N$ je splněno $p_j^T H^i p_j > 0$. Potom platí:*

$$\|s_j\|_M < \|s_{j+1}\|_M,$$

pro $0 \leq j \leq N - 1$.

Z uvedeného je vidět, že pokud metoda sdružených gradientů v nějaké iteraci opustí oblast $\mathcal{B}^i$, už se do ní nikdy nevrátí. Jako dobré řešení se proto jeví opět posun bodu s_k až na hranici oblasti ve směru p_k, stejně jako v předchozím případě.

Algoritmus formulujme takto:

Algoritmus 3.7 (Steihaug-Toint).
Polož $s_0 = 0, r_0 = g, y_0 = M^{-1}r_0, p_0 = -y_0$.
Pro $j = 0, 1, 2, \ldots$

(i) $\kappa_j = p_j^T H p_j$

(ii) Pokud $\kappa_j \leq 0$, najdi $\tilde{\alpha}$ jako řešení rovnice (3.27).
Polož $s_{j+1} = s_j + \tilde{\alpha} p_j$ a skonči.

(iii) $\alpha_j = r_j^T y_j / \kappa_j$

(iv) Pokud $\|s_j + \alpha_j p_j\|_M \geq \Delta^i$, najdi $\tilde{\alpha}$ jako řešení rovnice (3.27).
Polož $s_{j+1} = s_j + \tilde{\alpha} p_j$ a skonči.

(v) $s_{j+1} = s_j + \alpha_j p_j$

(vi) $r_{j+1} = r_j + \alpha_j H p_j$

(vii) $y_{j+1} = M^{-1} r_{j+1}$

(viii) $\beta_{j+1} = r_{j+1}^T y_{j+1} / r_j^T y_j$

(ix) $p_{j+1} = -y_{j+1} + \beta_{j+1} p_j$

(x) Zkontroluj konvergenční kritérium.

Pro vyřešení rovnice v bodech (ii) a (iv) rozepišme vztah (3.27):

$$(\Delta^i)^2 = \|s_j + \tilde{\alpha} p_j\|_M^2 = \|s_j\|_M^2 + 2\tilde{\alpha} s_j^T M p_j + \alpha^2 \|p_j\|_M^2 .$$

Vidíme, že se jedná o obyčejnou kvadratickou rovnici pro $\tilde{\alpha}$, což nám dává řešení:

$$\tilde{\alpha}_{1,2} = \frac{-s_j^T M p_j \pm \sqrt{(s_j^T M p_j)^2 + \|p_j\|_M^2 (\Delta^2 - \|s_j\|_M^2)}}{\|p_j\|_M^2} ,$$

přičemž jako $\tilde{\alpha}$ vybereme kladný kořen – nabízí totiž nižší hodnotu modelu m^i.

V souvislosti s algoritmem 3.7 můžeme narazit ještě na jeden problém. Často se v algoritmu setkáváme s potřebou vypočítat M-normu. To může být poměrně problematické, pokud matici M neznáme explicitně a máme jen funkci pro násobení matice M^{-1} vektorem. V takovém případě si pomůžeme následujícími rekurencemi:

$$\begin{aligned} s_j^T M p_j &= \beta_j (s_{j-1}^T M p_{j-1} + \alpha_{j-1} \|p_{j-1}\|_M^2) , \\ \|p_j\|_M^2 &= r_j^T y_j + \beta_j^2 \|p_{j-1}\|_M^2 , \end{aligned}$$

kde pro $j = 0$ volme

$$\begin{aligned} s_0^T M p_0 &= 0, \\ \|s_0\|_M^2 &= 0, \\ \|p_0\|_M &= \sqrt{p_0^T M p_0} = \sqrt{-p_0^T M y_0} = \sqrt{-p_0^T r_0} . \end{aligned}$$

Na závěr zdůrazněme, že jako první směr (p_0) volíme záporný gradient (resp. jeho přeškálování maticí M při použití předpodmínění), což znamená, že kdyby se metoda zastavila již v počáteční iteraci ($j = 0$), dostaneme Cauchyho bod. Navíc platí, že v každé další iteraci se hodnota modelu snižuje. Odtud dostáváme globální konvergenci metody Steihauga-Tointa.

3.3 Výpočet přibližného hessiánu

V úvodní kapitole jsme se zmínili o problémech, ve kterých je sestavení přesného hessiánu velmi časově náročné, a proto by mělo smysl uvažovat o nějaké jeho aproximaci i za cenu horších konvergenčních vlastností.

Vyjdeme z toho, že se ve většině algoritmů používá hessián jen při násobení s vektorem a není potřeba ho znát celý. V takových případech nahradíme $\nabla^2 F(x)z$ aproximací konečnými diferencemi:

$$\nabla^2 F(x)z \approx \frac{\nabla F(x + \delta z) - \nabla F(x)}{\delta}, \tag{3.28}$$

kde volba $\delta > 0$ je naprosto klíčová. V přesné aritmetice by bylo pochopitelně nejlepší volit δ co nejmenší. V konečné aritmetice se setkáváme s problémem, aby bylo δ „měřitelné" ve srovnáním se členy x a z. Používáme proto podobný přístup jako v [17] a volíme

$$\delta = (1 + \|x\|_2)\sqrt{\epsilon_M},$$

kde ϵ_M značí zaokrouhlovací chybu použitého počítače. Bližší informace k této problematice lze nalézt například v [19].

Kapitola 4

Předpodmiňovače

V předchozí kapitole jsme se zmínili o předpodmiňovačích v souvislosti s metodami CGM a QMR. Byly to pro nás jen „vhodné matice", které zlepšily číslo podmíněnosti a rozložení vlastních čísel matice soustavy. Mluvili jsme například o tom, jak se předpodmiňovače použijí, ale o konkrétních vlastnostech či volbě jsme pomlčeli. Tyto otázky budou hlavním předmětem této kapitoly.

Co by tedy předpodmiňovač měl splňovat? Ideální předpodmiňovač by měl dobře aproximovat inverzi k matici soustavy, měl by být snadno invertovatelný (ve smyslu, aby soustava $Mz = y$ byla rychle řešitelná) a neměl by mít velké nároky na paměť. Takový předpodmiňovač však bohužel neexistuje. Některé předpodmiňovače jsou sice obecně doporučované, ale žádný není zcela univerzální a použitelný při každé příležitosti. Vždy proti sobě bude bojovat účinnost na jedné straně a na straně druhé nároky, které můžeme ještě konkretizovat:

- paměťové nároky,
- časové nároky na sestavení,
- časové nároky na použití.

V této práci jsme se pokusili otestovat zástupce z každé třídy. Nejdříve si představíme celkově nenáročné předpodmiňovače – diagonální, SSOR a SGS. V dalších částech uvedeme L-BFGS a na závěr zástupce z třídy neúplných rozkladů – metodu AINV. Opět si budeme všímat i toho, zda lze použít aproximaci hessiánu zmíněnou v předchozí kapitole.

V dalším textu budeme značit předpodmiňovač jako matici M a budeme předpokládat, že slouží k předpodmínění soustavy (3.3).

4.1 Diagonální předpodmiňovač

Diagonální, někdy též označovaný jako *Jacobiho* předpodmiňovač patří mezi nejjednodušší předpodmiňovače, které lze najít. Jako M se volí diagonála matice soustavy H. Nelze tedy od něj sice čekat žádné zázračné zlepšení, ale někdy může výrazně pomoci, a navíc bývá díky svým prakticky nulovým paměťovým nákladům,

rychlosti a jednoduché implementaci používán velmi často, a to i třeba jako součást jiného předpodmiňovače. Bohužel vyžaduje explicitní znalost (diagonály) matice H, a tím znemožňuje použít aproximaci pomocí konečných diferencí.

4.2 SGS a SSOR

Předpodmiňovače *SGS (symetrický Gaussův-Seidelův)* a *SSOR (symetrický superrelaxační)* jsou příklady předpodmiňovačů, které se vyvinuly z iteračních metod pro řešení soustav lineárních rovnic. Představme si takovou iterační metodu zapsanou ve třetím normálním tvaru pro řešení soustavy $Hx = y$:

$$W_H(x^m - x^{m+1}) = Hx^m - y.$$

Při volbě $x^m = 0$ získáme:

$$-W_H x^{m+1} = -y \quad \Rightarrow \quad x^{m+1} = W_H^{-1} y.$$

Odtud dostáváme pěkný návod, jak rychle počítat řešení soustavy s maticí W_H a pravou stranou y - stačí udělat jeden krok této metody se startem v bodě 0. Proto se jako předpodmiňovač M pro matici H volí právě matice W_H.

Touto volbou máme zajištěnou minimální paměťovou náročnost a snadnou invertovatelnost. Zbývá ještě vhodně zvolit iterační metodu, aby matice W_H splňovala alespoň částečně poslední podmínku na předpodmiňovače – aby byla v nějakém smyslu dobrou aproximací inverzní matici soustavy. V praxi se osvědčily pro tento účel především metody SGS a SSOR.[1]

Mějme následující rozklad matice H:

$$H = D + L + L^T,$$

kde D je diagonální a L je spodní trojúhelníková matice. Potom můžeme matici W_H metody SSOR zapsat jako

$$M = W_H = \frac{1}{2-\omega}\left(\frac{1}{\omega}D + L\right)\left(\frac{1}{\omega}D\right)^{-1}\left(\frac{1}{\omega}D + L\right)^T,$$

případně jeden krok metody přepsat ve složkovém zápise:

Pro $i = 1, 2, \ldots, n$

$$x_i^{k+1/2} = \omega\left(\frac{y_i}{h_{ii}} - \frac{1}{h_{ii}}\left(\sum_{j<i} h_{ij}x_j^{k+1/2} + \sum_{j>i} h_{ij}x_j^k\right)\right) + (1-\omega)\,x_i^k$$

Pro $i = n, n-1, \ldots, 1$

$$x_i^{k+1} = \omega\left(\frac{y_i}{h_{ii}} - \frac{1}{h_{ii}}\left(\sum_{j<i} h_{ij}x_j^{k+1/2} + \sum_{j>i} h_{ij}x_j^{k+1}\right)\right) + (1-\omega)\,x_i^{k+1/2},$$

[1]Pro úplnost dodejme, že Jacobiho iterační metoda vede na diagonální předpodmiňovač.

kde $\omega \in (0,2)$ se nazývá *superrelaxační parametr*. Ten do velké míry ovlivňuje chování metody SSOR. Pro odhad optimální hodnoty ω existují numerické algoritmy, ale pro naše účely by to bylo zbytečně časově náročné, volí se proto pevně. Speciální volba $\omega = 1$ nám dává metodu (předpodmiňovač) SGS. Pro úplnost zapišme její matici W_H:

$$M = W_H = (D + L)\, D^{-1}\, (D + L)^T .$$

Je vidět, že ani tyto předpodmiňovače nejsou použitelné spolu s aproximací matice H uvedenou v předchozí kapitole.

Bližší podrobnosti lze najít například v [1, 2].

4.3 L-BFGS

Uveďme nejdříve stručné základy metody *BFGS*, protože na její myšlence je předpodmiňovač *L-BFGS* založen.

Metoda *BFGS*[2] patří mezi metody pro řešení úloh nepodmíněné optimalizace, do skupiny tzv. *metod s proměnnou metrikou (quasi-Newton methods)*. Předpokládejme, že minimalizujeme funkci $F(x)$ pro $x \in \mathbb{R}^n$, a označme stejně jako v kapitole 3 jednotlivé iterace x^i, funkční hodnotu, gradient a hessián funkce F v bodě x^i jako F^i, g^i a H^i.

Základní myšlenka metody BFGS spočívá v použití vhodné aproximace inverze hessiánu funkce F. Označme takovou aproximaci v bodě x^i jako W^i, tj. $W^i \approx (H^i)^{-1}$ a stanovme pro ni následující požadavky:

(i) $W^i \in \mathbb{S}^n, W^i \succ 0$,

(ii) W^i je počítána rekurzivně podle vzorce $W^i = W^{i-1} + Q^{i-1}$, kde Q^{i-1} je matice s hodností maximálně dva,

(iii) W^i splňuje tzv. *quasi-Newtonovu (sečnou) rovnici*

$$W^i y^{i-1} = t^{i-1}, \tag{4.1}$$

kde $t^{i-1} = x^i - x^{i-1}$ a $y^{i-1} = g^i - g^{i-1}$,

(iv) W^i je „nejbližší" matice k W^{i-1} splňující (i) a (iii) v tom smyslu, že řeší úlohu

$$\min_W \|W - W^{i-1}\| \qquad \text{za podmínky} \quad W = W^T,\ W y^{i-1} = t^{i-1} .$$

Poznamenejme, že podmínka (iii) vychází z následující přirozené úvahy. Definujme kvadratický model funkce F v bodě x^i pomocí matice W^i:

$$m^i(x^i + d) := F^i + (g^i)^T d + \frac{1}{2} d^T (W^i)^{-1} d .$$

[2]Název BFGS je sestaven z počátečních písmen jmen autorů: Broyden, Fletcher, Goldfarb, Shanno.

A požadujme, aby platila v bodech x^{i-1} a x^i rovnost gradientu funkce F a modelu m^i. Odtud získáváme rovnosti:

$$\begin{aligned}\nabla m^i(x^i + d) &= g^i + (W^i)^{-1}d\,,\\ \nabla m^i(x^i) &= g^i\,,\\ \nabla m^i(x^{i-1}) &= g^i + (W^i)^{-1}(x^{i-1} - x^i) = g^{i-1},\end{aligned}$$

z nichž přímo plyne (4.1).

Konkrétní tvar formulí pro metodu BFGS bychom získali z vlastností (i) (iv) speciální volbou normy ve (iv). Tím se zde zabývat nebudeme, podrobnosti lze najít např. v [19]. Výsledný vztah pro matice W^i je

$$W^{i+1} = \left(I - \frac{t^i(y^i)^T}{(y^i)^T t^i}\right) W^i \left(I - \frac{y^i(t^i)^T}{(y^i)^T t^i}\right) + \frac{t^i(t^i)^T}{(y^i)^T t^i}\,. \tag{4.2}$$

Pro úplnost dodejme, že pomocí Shermanovy-Morrisonovy formule (opět viz [19]) lze odvodit podobný vztah pro inverzi W^i, tj. aproximace hessiánu H^i, nikoli jeho inverze. Označme tuto aproximaci B^i. Dostali bychom:

$$B^{i+1} = B^i + \frac{y^i(y^i)^T}{(y^i)^T t^i} - \frac{B^i t^i (t^i)^T B^i}{(t^i)^T B^i t^i}\,.$$

Myšlenka použít nějakým způsobem aproximaci inverze hessiánu (4.2) pro předpodmínění naší soustavy (3.3) je sice možná zajímavá, leč zatím předčasná. Uvedené vztahy mají bohužel jednu špatnou vlastnost. Je vidět, že matice W^i může být už po prvním kroku obecně plná. To nám prakticky znemožňuje použít (4.2) v této podobě pro úlohy větší dimenze. Řešením je metoda *L-BFGS (Limited Memory BFGS)*.

Přepišme rovnost (4.2) do následujícího tvaru:

$$W^{i+1} = V_i^T W^i V_i + \rho_i t^i (t^i)^T\,, \tag{4.3}$$

kde

$$\rho_i = \frac{1}{(y^i)^T t^i}\,, \qquad V_i = I - \rho_i y^i (t^i)^T\,.$$

Několikanásobným použitím (4.3) pro $i = k-m, \ldots, k-1$ dostaneme:

$$\begin{aligned}W^k = {} & (V_{k-1}^T \cdots V_{k-m}^T) W_0^{k-m} (V_{k-m} \cdots V_{k-1})\\ & + \rho_{k-m}(V_{k-1}^T \cdots V_{k-m+1}^T) t^{k-m}(t^{k-m})^T (V_{k-m+1} \cdots V_{k-1})\\ & + \rho_{k-m+1}(V_{k-1}^T \cdots V_{k-m+2}^T) t^{k-m+1}(t^{k-m+1})^T (V_{k-m+2} \cdots V_{k-1})\\ & \vdots\\ & + \rho_{k-1} t^{k-1}(t^{k-1})^T\,.\end{aligned} \tag{4.4}$$

Myšlenka metody L-BFGS spočívá v tom, že místo toho, aby se ukládal explicitní tvar matice W^k, uloží se do paměti pouze sada vektorů $\{t^i, y^i\}$ pro $i = k-m, \ldots, k-1$ a místo matice W_0^{k-m} se zvolí nějaká její jednoduchá aproximace, například

$$W_0^{k-m} = \frac{(t^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}}\, I\,.$$

Tím je zaručeno, že k uchování aktuální podoby matice W^k bude stačit jen $2m$ vektorů. Pokud bude potřeba násobit matici W^k nějakým vektorem d, využije se vztahu (4.4), z něhož lze odvodit následující algoritmus. Jeho složitost je $4mn$ násobení.

Algoritmus 4.1 (L-BFGS násobení).
Dána sada vektorů $\{t^i, y^i\}$ pro $i = k-m, \ldots, k-1$ a vektor d,
algoritmus spočte $r = W^k d$

(i) Polož $q = d$

(ii) Pro $i = k-1, k-2, \ldots, k-m$

$$\alpha_i = \frac{(t^i)^T q}{(y^i)^T t^i}, \qquad q = q - \alpha_i y^i$$

(iii) Polož $r = W_0^{k-m} q$

(iv) Pro $i = k-m, k-m+1, \ldots, k-1$

$$\beta = \frac{(y^i)^T r}{(y^i)^T t^i}, \qquad r = r + s^i(\alpha_i - \beta)$$

Předpodmínění pomocí L-BFGS

Základní rozdíl mezi L-BFGS předpodmiňovačem a ostatními je ten, že L-BFGS je přímo navržen pro řešení posloupnosti soustav lineárních rovnic. Myšlenka spočívá v tom, že se během výpočtu řešení jednoho systému současně generuje předpodmiňovač pro systém další.

Konkrétně řečeno, první systém se řeší nepředpodmíněný a během jeho výpočtu se ukládají vybrané dvojice vektorů $\{t^i, y^i\}$, ze kterých se generuje předpodmiňovač pro další soustavu rovnic. Druhá soustava se už řeší s předpodmíněním L-BFGS složeného z těchto vektorů. V průběhu jejího řešení se opět vybírají vektory do nového předpodmiňovače a tak dále. Předpokládá se totiž, že si systémy rovnic v jednotlivých krocích budou poměrně podobné, a tak předpodmiňovač generovaný na základě informací z minulé soustavy bude dostatečně dobrý i pro soustavu následující.

Poslední otázkou je, jakým způsobem vybírat dvojice vektorů $\{t^i, y^i\}$. Existují dva základní přístupy. Buď ukládat vždy posledních m vektorů, nebo volit vektory rovnoměrně v celém průběhu výpočtu soustavy. Druhá strategie je sice o něco komplikovanější, ale dosahuje lepších výsledků. Problém je totiž v tom, že nikdy není předem známo, v kolika iteracích bude řešení soustavy rovnic nalezeno. Následující algoritmus se tento problém snaží vyřešit a vybírá dvojice vektorů tak rovnoměrně, „jak to jen jde“.

Algoritmus 4.2 (Vzorkovací algoritmus pro L-BFGS).
Dáno sudé číslo m, polož $cyklus = 1$.
Pro $i = 0, 1, 2, \ldots$

Inicializace
Pokud $i < m$

- Ulož $\{t^i, y^i\}$ do $\mathcal{P}$

Mazání/vkládání
Pokud lze i vyjádřit jako $i = (\frac{m}{2} + l - 1)2^{cyklus}$ pro nějaké $l \in \{1, 2, \ldots, \frac{m}{2}\}$ potom

- Spočítej index dvojice na smazání jako $k = (2l - 1)2^{cyklus-1}$
- Odstraň $\{t^k, y^k\}$ z $\mathcal{P}$
- Ulož $\{t^i, y^i\}$ do $\mathcal{P}$
- Pokud $l = \frac{m}{2}$, pak $cyklus = cyklus + 1$

S ohledem na to, že metoda BFGS je navržena jako gradientní metoda, tak ani předpodmiňovač L-BFGS nepotřebuje znát prvky hessiánu. Je proto ideální v případech, kdy hessián není dostupný.

Bližší informace o metodách BFGS a L-BFGS lze nalézt v [19], o použití metody L-BFGS jako předpodmiňovač je popsáno v [17].

4.4 AINV

Předpodmiňovače založené na neúplném rozkladu matice by se daly obecně označit za poměrně účinné, ale náročné na paměť i čas potřebný k jejich vygenerování. V této kapitole si představíme ne úplně typického zástupce této třídy předpodmiňovačů. Zatímco většina z nich je založena na neúplném Choleského rozkladu, předpodmiňovač *AINV (Aproximate INVerse)* je postaven na H-ortogonalizaci (H značí opět matici soustavy (3.3)). V dalším předpokládejme, že H je symetrická a pozitivně definitní.

Mezi základní vlastnosti předpodmiňovače AINV patří, že neúplný rozklad matice existuje vždy a že není potřeba znát předem rozložení nenulových prvků řídké matice – AINV si ho totiž generuje sám v průběhu výpočtu rozkladu.

Použitím AINVu na matici H získáme přibližný rozklad její inverze:

$$M = ZD^{-1}Z^T \approx H^{-1}, \tag{4.5}$$

kde Z je horní trojúhelníková matice s jedničkami na diagonále a D je diagonální matice. Jak ukážeme později, Z je řídká aproximace matice L^{-T}, kde L je dolní trojúhelníková matice v rozkladu $H = LDL^T$.

Matici Z získáme v n krocích H-ortogonalizací jednotkové matice. Její podobu v k-tém kroku označme $Z^{(k)}$ a její sloupce jako vektory $z_i^{(k)}$. Na počátku položme

$Z^{(0)} = I$. H-ortogonalizaci provedeme pomocí Gramm-Schmidtova procesu.

Předpokládejme, že jsme v k-tém kroku algoritmu AINV a už máme H-ortogonalizované vektory $z_i^{(k-1)}$ pro $i = 1, 2, \ldots, k-1$, tj.

$$(z_i^{(k-1)})^T H z_j^{(k-1)} = 0, \quad \text{pro } 1 \leq i \leq k-1,\ 1 \leq j \leq n,\ i \neq j. \tag{4.6}$$

a hledáme vektory $z_i^{(k)}$, aby rovnost (4.6) platila i pro k. Pro $i = 1, 2, \ldots, k$ položíme $z_i^{(k)} = z_i^{(k-1)}$ a vektory pro $i > k$ budeme H-ortogonalizovat vůči $z_k^{(k)}$, tj. najdeme takové α pro vztah

$$z_i^{(k)} = z_i^{(k-1)} - \alpha z_k^{(k)}, \tag{4.7}$$

aby platila podmínka (4.6). Vynásobením $(z_k^{(k)})^T H$ zleva obdržíme

$$0 = (z_k^{(k)})^T H z_i^{(k)} = (z_k^{(k)})^T H z_i^{(k-1)} - \alpha (z_k^{(k)})^T H z_k^{(k)}$$

a odtud získáme

$$\alpha = \frac{(z_k^{(k)})^T H z_i^{(k-1)}}{(z_k^{(k)})^T H z_k^{(k)}}.$$

Díky volbě $Z^{(0)}$ jako jednotkové matice máme zaručeno, že vzniklá matice $Z := Z^{(n)}$ bude horní trojúhelníková s jedničkami na diagonále. Podmínku H-ortogonality můžeme přepsat do maticového tvaru:

$$Z^T H Z = D,$$

kde D značí diagonální matici s prvky $d_{ii} = (z_i)^T H z_i$. Jednoduchou úpravou dostaneme (4.5) s rovností:

$$H = Z^{-T} D Z^{-1}$$
$$H^{-1} = Z D^{-1} Z^T.$$

Pokud bychom takto provedli H-ortogonalizaci, dostali bychom přesný rozklad matice H^{-1}. To by byla obecně plná matice, což by pro naše účely bylo příliš paměťově i časově náročné. Myšlenkou proto je stanovit nějaký test na „dostatečnou" velikost prvků matice Z a jen s těmi prvky dále počítat. Vznikla by tak řídká matice, která by při správné volbě takového testu mohla velmi dobře aproximovat původní matici. O konkrétním způsobu volby testu se budeme ještě bavit.

Celkově můžeme algoritmus zapsat takto:

Algoritmus 4.3 (AINV).
Označení: $Z = [z_1, z_2, \ldots, z_n]$, $D = \mathrm{diag}\,(p_1, p_2, \ldots, p_n)$
Inicializace: $z_i = e_i$ $(1 \leq i \leq n)$
Pro $i = 1, 2, \ldots, n$

(i) $v = Az_i$

(ii) $p_i = v^T z_i$

(iii) Pro $j = i + 1, \ldots, n$

(a) $z_j = z_j - (v^T z_j / p_i)\, z_i$

(b) Proveď test na velikost složek

Testování velikosti prvků matice Z

Z povahy algoritmu je vidět, že právě test na velikost prvků a implementace řídkých datových struktur je klíčem k úspěchu, či neúspěchu předpodmiňovače. Na jedné straně stojí požadavek na minimální zaplnění matice Z (to ovlivňuje nejen velikost paměti, ale i celkovou rychlost generování a používání předpodmiňovače) a na druhé straně bojuje účinnost předpodmiňovače za co největší počet nenulových prvků. Bohužel v článcích o této metodě ([3, 4]) je tomuto tématu věnováno minimum. Lze se proto jen domnívat a zkoušet různé strategie.

Jednou možností je nastavit kritérium na velikost pevně (v citovaných článcích se mluví o 0.1) a testovat vektory, lépe řečeno jejich složky, hned během jejich výpočtu, tj. v průběhu H-ortogonalizace. Tato strategie vede na poměrně malé zaplnění matice, bohužel chyba vnesená odebráním daného prvku se výrazně akumuluje, což celkově zhoršuje chování předpodmiňovače.

Druhým možným řešením je testovat velikost prvků vektoru $z_k^{(k)}$ až potom, co budou vypočítány všechny vektory $z_i^{(k)}$ pro $i = k + 1, \ldots, n$. Tím se sice chyba akumulovat nebude, ale při generovaní předpodmiňovače bude potřeba výrazně více paměti i času.

Dalšími možnostmi by mohlo být nějakým způsobem adaptovat kritérium na základě aktuálního zaplnění, případně uvedené způsoby kombinovat.

Možná vylepšení metody

Poznamenejme, že k zlepšení celkového chování algoritmu by mohlo přispět použití diagonálního předpodmiňovače nebo nějakého algoritmu pro přečíslování (permutaci prvků matice). V rámci této práce jsme zkoušeli modifikovat AINV jen s použitím diagonálního předpodmiňovače.

Další informace lze najít v [3, 4].

Kapitola 5

Program PENNON

Program PENNON (PENalty method for NONlinear optimization) byl vytvořen na univerzitě v Erlangen [13]. Jedná se o program (jak název napovídá) pro řešení problémů nelineárního a semidefinitního programování založený na zobecněné metodě rozšířeného lagrangiánu.

5.1 Definice problému, základní algoritmus

Upřesněme nejdříve, jakou úlohou se budeme zabývat. Místo podmíněné optimalizace zmíněné v úvodní kapitole uvažujme obecnější problém:

Definice 5.1 (NLP-SDP). Nechť f, g_i jsou funkce z $\mathcal{C}^2(\mathbb{R}^n; \mathbb{R})$ pro každé i a nechť $\mathcal{A} : \mathbb{R}^n \to \mathbb{S}^{m_A}$ je maticový operátor. Pak úlohu

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{za podmínek} \qquad g_i(x) \leq 0, \qquad i = 1, 2, \ldots, m_g \tag{5.1}$$

$$\mathcal{A}(x) \preceq 0. \tag{5.2}$$

budeme nazývat úlohou *nelineárního a semidefinitního programování (NLP-SDP).*[1]

Základní myšlenka algoritmu je založena na vhodné volbě funkcí penalizujících nerovnosti (5.1) a (5.2). Zvolme penalizační funkce $\varphi_g, \varphi_A : \mathbb{R} \to \mathbb{R}$ s následujícími vlastnostmi:

(i) $\varphi \in \mathcal{C}^2$ je ryze konvexní a rostoucí

(ii) φ je definována na $(-\infty, b)$, kde $0 < b \leq \infty$

(iii) $\varphi(0) = 0$

(iv) $\varphi'(0) = 1$

[1] Podmínky ve tvaru rovnosti $h_i = 0$ byly záměrně pro zjednodušení vypuštěny. Dají se ekvivalentně nahradit dvěma podmínkami ve tvaru nerovností: $h_i \leq 0, -h_i \leq 0$.

(v) $\exists C$ tak. že $\lim_{p \searrow 0} \varphi'(\sigma/p) \leq C$ pro každé $\sigma < 0$

(vi) $\lim_{t \to b} \varphi'(t) = \infty$

(vii) $\lim_{t \to -\infty} \varphi'(t) = 0.$

Pomocí φ_A můžeme nyní definovat penalizační funkci $\Phi_P : \mathbb{S}^{m_A} \to \mathbb{S}^{m_A}$ pro porušení podmínky (5.2):

$$\Phi_P : A \mapsto S^T \begin{pmatrix} P\varphi_A(\frac{\lambda_1}{P}) & 0 & \cdots & 0 \\ 0 & P\varphi_A(\frac{\lambda_2}{P}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P\varphi_A(\frac{\lambda_{m_A}}{P}) \end{pmatrix} S,$$

kde $A = S^T \Lambda S$, $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{m_A})$, značí rozklad matice A na vlastní čísla a vektory a $P \in \mathbb{R}^+$ je penalizační parametr.

Pro funkce φ_g a φ_A splňující (i) (vii) a pro $p_i, P \in \mathbb{R}^+$ lze podmínky (5.1) a (5.2) ekvivalentně přepsat ve tvaru

$$\begin{aligned} g_i(x) \leq 0 \quad &\Leftrightarrow \quad p_i \varphi_g(g_i(x)/p_i) \leq 0, \qquad i = 1, 2, \ldots, m_g \\ \mathcal{A}(x) \preceq 0 \quad &\Leftrightarrow \quad \Phi_P(\mathcal{A}(x)) \preceq 0 \end{aligned}$$

a úlohu (NLP-SDP) jako

$$\left.\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} f(x) & \\ \text{za podmínek} & p_i \varphi_g(g_i(x)/p_i) \leq 0, \quad i = 1, 2, \ldots, m_g \\ & \Phi_P(\mathcal{A}(x)) \preceq 0. \end{array}\right\} \qquad \text{(NLP-SDP')}$$

Lagrangián nové úlohy (NLP-SDP') můžeme považovat za rozšířený lagrangián původní (NLP-SDP):

$$\mathcal{F}(x, u, U, p, P) = f(x) + \sum_{i=1}^{m_g} u_i p_i \varphi_g(g_i(x)/p_i) + \langle U, \Phi_P(\mathcal{A}(x)) \rangle_{\mathbb{S}^{m_A}}$$

kde $u \in \mathbb{R}^{m_g}$ a $U \in \mathbb{S}^{m_A}$ jsou Lagrangeovy multiplikátory a $p, P > 0$ jsou penalizační parametry.

Základní algoritmus programu PENNON formulujme takto:

Algoritmus 5.1 (Základní algoritmus programu PENNON).
Dáno x^0 a U^0, polož $p_i^0 > 0$ pro $i = 1, 2, \ldots, m_g$ a $P^0 > 0$.
Pro $k = 0, 1, 2, \ldots$

(i) Řešení úlohy nepodmíněné optimalizace

- $x^{k+1} = \arg\min_{x \in R^n} \mathcal{F}(x, u^k, U^k, p^k, P^k)$

(ii) Aktualizace Lagrangeových multiplikátorů

- $u_i^{k+1} = u_i^k \varphi_q'(g_i(x^{k+1})/p_i^k), \qquad i = 1, 2, \ldots, m_g$
- $U^{k+1} = D_{\mathcal{A}} \Phi_P(\mathcal{A}(x); U^k)$

(iii) Aktualizace penalizačních parametrů

- $p_i^{k+1} < p_i^k, \qquad i = 1, 2, \ldots, m_g$
- $P^{k+1} < P^k$

(iv) Kontrola konvergenčního kritéria

Smysl bodu (ii) lze vysvětlit takto: Jestliže x^{k+1} padne mimo přípustnou oblast (např. $g_j(x^{k+1}) > 0$), vzroste zvětšením Lagrangeova multiplikátoru u_j^k vliv podmínky g_j a ta se v příštím kroku poruší výrazně méně, nebo dokonce vůbec. Připomeňme vlastnosti (i) a (iv) funkce φ_q, z nichž plyne

$$\varphi_q'(g_j(x^{k+1}/p_j^k) > 1 \qquad \text{pro} \quad g_j(x^{k+1}) > 0.$$

Z toho dostáváme $u_j^{k+1} > u_j^k$.

V každé iteraci algoritmu 5.1 se začne hledat řešení nepodmíněné optimalizace (bod (i)) startem z bodu x^k a hledá se tak dlouho, dokud není splněno konvergenční kritérium

$$\| \nabla_x \mathcal{F}(x, u^k, U^k, p^k, P^k) \| \leq \alpha^k,$$

kde α^k je parametr, který se adaptuje v průběhu výpočtu. Typicky se po několika prvních iteracích ustálí na hodnotě 0,01. Poznamenejme, že ve stávající verzi programu je použita Newtonova (resp. modifikovaná Newtonova) metoda tak, jak je představena v kapitole 3.1.1.

Konvergenční kritéria použitá v bodu (iv) algoritmu 5.1 jsou následující:

$$\frac{|f(x^k) - \mathcal{F}(x^k, u^k, U^k, p, P)|}{1 + |f(x^k)|} < \epsilon,$$

$$\frac{|f(x^k) - f(x^{k-1})|}{1 + |f(x^k)|} < \epsilon,$$

kde ϵ je typicky nastaveno na 10^{-7}. Navíc se sleduje splnění podmínek (5.1) a (5.2). Druhotně je ještě kontrolováno kritérium:

$$\|\nabla_x \mathcal{F}(x^k, u^k, U^k, p, P)\| < \epsilon_2 \,.$$

Podrobnější informace o programu PENNON lze najít v [13, 14] či na webových stránkách projektu

`http://www2.am.uni-erlangen.de/~kocvara/pennon/`,

podrobnosti o metodě rozšířeného lagrangiánu v [19] a důkaz globální konvergence algoritmu v pracích [6, 20].

Kapitola 6

Numerické výsledky

V této kapitole si uvedeme výsledky numerických testů metod a předpodmiňovačů diskutovaných v minulých kapitolách. Ty byly implementovány do programu PENNON a v jeho rámci testovány na úlohách lineárního semidefinitního programování, konvexního a nekonvexního nelineárního programování. Konkrétní úlohy byly získány z knihoven volně dostupných na internetu:

`ftp://plato.asu.edu/pub/ampl_files/qpdata_ampl/`
`http://infohost.nmt.edu/~sdplib/`
`http://www2.am.uni-erlangen.de/~kocvara/pennon/problems.html`

V jednotlivých částech této kapitoly provedeme srovnání různých modifikací nově implementovaných metod mezi sebou a s původní verzí programu. Zde uvedeme jen část dosažených výsledků, jejich detailnějších verzi lze nalézt v příloze.

Testování probíhalo na počítačích s procesory AMD Athlon1800+ a Intel Pentium 4 2,8 GHz na platformě MS Windows v prostředí Cygwin, viz

`http://www.cygwin.com/` .

Implementace algoritmů byla provedena v jazyce C.

6.1 Značení

V tabulkách výsledků budeme používat následující značení.

- **n** dimenze úlohy
- **m** počet omezení v úloze
- **t** celkový čas výpočtu úlohy v sekundách, označení **TLM** značí překročení stanoveného časového limitu (typicky 20 minut)
- **Pn** počet iterací algoritmu 5.1, též počet úloh nepodmíněné optimalizace, na kterou se převedl řešený problém; značka **LIM** označuje dosažení maximálního povoleného počtu iterací (typicky 100)
- **Nw** celkový počet iterací Newtonova algoritmu, resp. spádových směrů, resp. TRM

Cg	celkový počet iterací metody CGM, resp. QMR
t/Nw	průměrná doba potřebná k výpočtu jedné Nw iterace
Cg/Nw	průměrný počet Cg iterací na jeden Nw krok

Označení metod:

ORIG	původní verze programu PENNON (využívající modifikovanou Newtonovu metodu a řídkého Choleského rozklad)
CGM	metoda sdružených gradientů
ACGM	metoda CGM s využitím přibližného výpočtu hessiánu
TRST	metoda trust-region – Steihaug-Toint
QMR	metoda QMR – modifikace pro symetrickou matici i předpodmiňovač
QMRP	metoda QMR implementovaná v QMRPacku

Označení předpodmiňovačů:

D	diagonální předpodmínění
Bm	L-BFGS s maximálně **m** dvojicemi vektorů generujícími předpodmiňovač
GS	symetrický Gaussův-Seidelův předpodmiňovač
SS	symetrický SOR předpodmiňovač
Acn	AINV, kde c=D/N (diagonální/normální modifikace), n=1/3/4 (kritérium na vypouštění prvků nastaveno na 0,1/0,95/0,99)

Při testování bylo nastaveno konvergenční kritérium na 10^{-7} pro úlohy NLP a na 10^{-4} pro SDP. Pokud řešení úlohy NLP splňuje všechna potřebná konvergenční kritéria pro ukončení algoritmu 5.1 (viz kapitola 5) až na požadavek dostatečně malé hodnoty gradientu rozšířeného lagrangiánu, program PENNON řešení označí jako *suboptimální* řešení. To v tabulkách výsledků budeme signalizovat znakem *.

6.2 Úlohy semidefinitního programování

6.2.1 Teoretická složitost výpočtu

Zkusme nyní odhadnout výpočetní náročnost jednotlivých kroků prováděných během jedné iterace Newtonovy metody v původním algoritmu. Omezíme se jen na úlohy lineárního SDP, tj. úlohy SDP s maticí $\mathcal{A}(x)$ z (5.2) ve tvaru $\mathcal{A}(x) = \sum_i x_i A_i$. Připomeňme, že v původní verzi programu PENNON je pro řešení Newtonovy soustavy využito (řídkého) Choleského rozkladu.

Jedna iterace se skládá z kroků:

- Výpočet hessiánu
  - $O(m_A^3 n + m_A^2 n^2)$ pro husté datové matice A_i
  - $O(m_A^2 n + K^2 n^2)$ pro řídké datové matice A_i, kde K značí maximální počet nenulových prvků v datových maticích A_i přes všechna i

- Výpočet gradientu rozšířeného lagrangiánu
  - $O(m_A^3)$ pro husté matice
  - $O(m_A^2 \kappa)$ pro řídké matice, kde κ je počet nenulových prvků v rozkladu $\mathcal{A}$
- Řešení Newtonovy soustavy pomocí Choleského rozkladu hessiánu
  - $O(n^3)$ pro husté matice
  - $O(n^p)$ pro řídké matice, kde $2 \leq p \leq 3$

kde m_A je zavedeno v definici 5.1. V našem případě nahrazujeme Choleského rozklad metodou CGM (QMR). Pokud by úloha byla špatně podmíněná, vedla by metoda CGM také na složitost $O(n^3)$ či horší a nahrazení Choleského rozkladu by se projevilo spíš negativně (větší nestabilitou). Ovšem v případě, že by byla soustava podmíněna dobře, pak by metoda CGM potřebovala jen několik málo iterací a rozdíl by se projevil výraznějším způsobem. To ovšem jen za předpokladu, že ostatní výpočty během iterace (výpočet hessiánu a gradientu) jsou nenáročné oproti řešení Newtonovy soustavy. Tedy jen v případě, kdy m_A je malé oproti n a v případě velkých řídkých úloh, ve kterých je matice $\mathcal{A}$ řídká, ale hessián je hustý.

6.2.2 Výsledky pro lineární SDP

V následujících tabulkách je možné vidět porovnání metody CGM a originálního programu s Choleského rozkladem. Podle předpokladu nastává zlepšení jen u úloh **theta** – mají výrazný rozdíl m a n a nejsou příliš špatně podmíněné. Pro ně se chová metoda CGM velmi dobře. To vidíme i v porovnání v další tabulce na průměrném počtu Cg iterací v jednom Nw kroku. Zdůrazněme, že zlepšení v úloze **theta6** je více než pětinásobné.

			ORIG			CGM			
	n	m	Pn	Nw	t	Pn	Nw	Cg	t
theta1	104	50	13	45	0	13	48	271	0
theta2	498	100	12	47	3	12	50	339	2
theta3	1106	150	12	52	24	12	63	367	9
theta4	1949	200	12	57	121	12	64	421	33
theta5	3028	250	12	61	446	12	62	412	97
theta6	4375	300	12	52	1115	13	59	420	204
thetaG11	2401	801	21	99	480	20	133	1133	256
control1	21	15	10	45	0	10	51	1085	0
control2	66	30	12	66	0	12	109	11986	1
control3	136	45	11	63	1	11	68	9553	2
control4	231	60	11	110	5	11	108	33837	14
control5	351	75	11	88	16	11	94	45118	46
control6	496	90	11	120	66	10	108	71735	155
control7	666	105	15	132	150	14	103	61737	257
control8	861	120	14	166	359	14	104	89948	562
control9	1081	135	16	198	759	16	161	155509	1535
control10	1326	150	16	235	1463				TLM

Na druhé straně je vidět opačný extrém – špatně podmíněné příklady control způsobují pomalou konvergenci metody CGM, a tím i velmi drahé Nw iterace.

			ORIG		CGM		
	n	m	t	t/Nw	t	t/Nw	Cg/Nw
theta1	104	50	0	0,00	0	0,00	5,6
theta2	498	100	3	0,06	2	0,04	6,8
theta3	1106	150	24	0,46	9	0,14	5,8
theta4	1949	200	121	2,12	33	0,52	6,6
theta5	3028	250	446	7,31	97	1,56	6,6
theta6	4375	300	1115	21,44	204	3,46	7,1
thetaG11	2401	801	480	4,85	256	1,92	8,5
control1	21	15	0	0,00	0	0,00	21,3
control2	66	30	0	0,00	1	0,01	110,0
control3	136	45	1	0,02	2	0,03	140,5
control4	231	60	5	0,05	14	0,13	313,3
control5	351	75	16	0,18	46	0,49	480,0
control6	496	90	66	0,55	155	1,44	664,2
control7	666	105	150	1,14	257	2,50	599,4
control8	861	120	359	2,16	562	5,40	864,9
control9	1081	135	759	3,83	1535	9,53	965,9

V níže uvedené tabulce můžeme sledovat chování úloh s převážně řídkými maticemi A_i a hustým hessiánem. Podle předpokladu se výhoda užití metody CGM (případně QMR) namísto Choleského rozkladu reflektuje až u úloh s vyšší dimenzí (například trto5).

			ORIG		QMR D		
	n	m	t	t/Nw	t	t/Nw	Cg/Nw
buck2	144	237	5	0,06	4	0,06	7
buck3	544	1185	247	1,24	239	1,28	33
buck4	1200	2545	757	5,82	815	5,91	54
buck5	3280	6801	20291	77,15	16273	65,62	80
trto2	144	235	3	0,03	3	0,04	7
trto3	544	865	50	0,53	57	0,51	20
trto4	1200	1873	475	3,30	517	3,08	30
trto5	3280	5040	11939	63,84	3168	23,64	20
vibra2	144	237	6	0,06	5	0,06	8
vibra3	544	1185	223	1,04	149	1,04	17
vibra4	1200	2545	632	5,50	914	6,30	17
vibra5	3280	6801	10848	63,44	18439	54,88	77
shmup2	200	1281	211	2,48	199	2,80	5
shmup3	420	1641	465	4,51	368	4,60	6
shmup4	800	4961	3001	27,28	2455	27,58	7
spmup5	1800	11141	4745	139,56	4564	138,30	3

Následující tabulka ukazuje srovnání několika různých předpomiňovačů na vybraném vzorku úloh. Jak se ostatně dalo předpokládat, žádný z nich nijak výrazně nevyniká. U každého můžeme narazit na úlohy, na kterých nebude dobře fungovat. Obecně by se asi dala doporučit jako základ pro algoritmus metoda QMR.

	CGM		CGM D		CGM B16		CGM GS		QMR		QMR D	
	Cg	t	Cg	t	Cg	t	Cg	t	Cg	t	Cg	t
theta5	412	97	824	113	332	94	1340	190	393	100	794	117
theta6	420	204	573	225	334	196	248	213	303	201	522	221
control3	9553	2	8398	1	4813	1	5393	2	7047	1	6979	1
control6	71735	155	77827	183	40273	105	44830	181	44521	114	53141	146
arch2	1467	9	575	8	1083	9	367	9	1170	9	527	9
arch4	1512	9	629	9	983	9	383	9	1212	9	566	9
buck2	3252	2	785	3	3160	3	521	3	2646	3	661	3
shmup2	11971	123	536	115	17093	119	496	102	8706	105	478	104
trto3	31693	123	2897	34	27924	128	1215	34	28187	122	2329	39
truss7	3755	0	2607	1	3090	0	1335	0	3248	0	1704	0
truss8	17050	66	4132	26	16487	66	1905	26	14140	61	3596	27
vibra2	5254	4	1175	4	5886	5	1339	4	4067	3	995	3

6.2.3 Ukázka konkrétních úloh

Na níže uvedených obrázcích můžeme vidět chování metody CGM a QMR na úlohách control3 a theta2. Lépe řečeno na první (v prvním sloupečku) a poslední (ve druhém sloupečku) úloze nepodmíněné optimalizace ze sekvence, na kterou se převedly problémy control3 a theta2. Poznamenejme, že dimenze úlohy control3 je 136 a úlohy theta2 je 498.

V prvním řádku je uvedeno rozložení vlastních čísel (v logaritmickém měřítku). Ve druhém řádku máme (opět logaritmický) graf vývoje rezidua

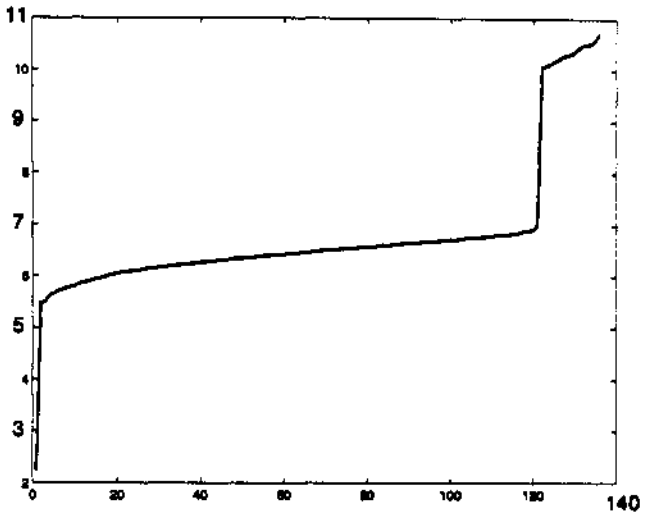
$$\frac{\|Hs+g\|}{\|g\|}$$

v jednotlivých iteracích metody CGM a ve třetím řádku totéž pro metodu QMR.

Jsou zde vidět dvě důležité věci. Jednak výrazně zhoršující se podmíněnost matic sekvence. A jednak rozdílné chování metod CGM a QMR. Výrazně hladší průběh reziduí metody QMR je dán tím, že metoda QMR „quasi-minimalizuje“ reziduum, zatímco metoda CGM minimalizuje kvadratický funkcionál $\frac{1}{2}s^THs+g$.

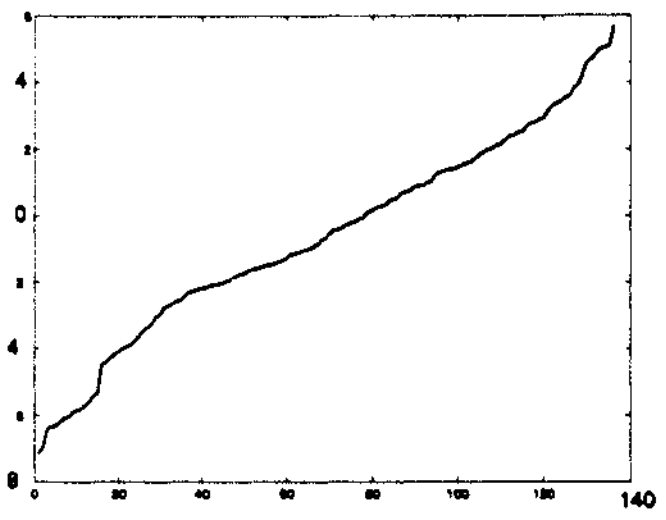
CONTROL3

první příklad sekvence



$\kappa_1 = 3,1 \cdot 10^8$

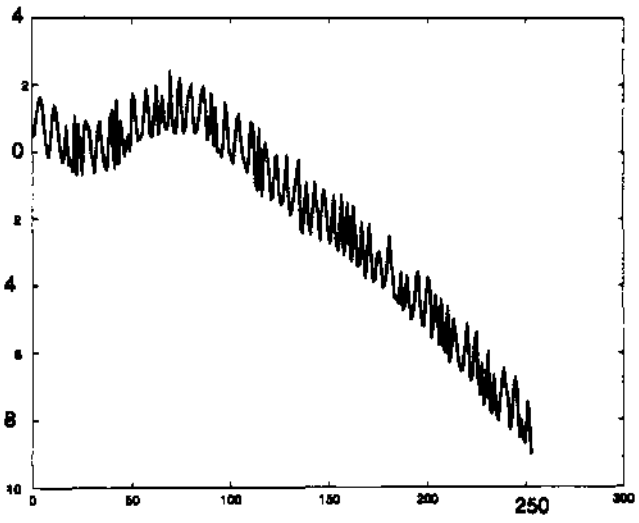
poslední příklad sekvence

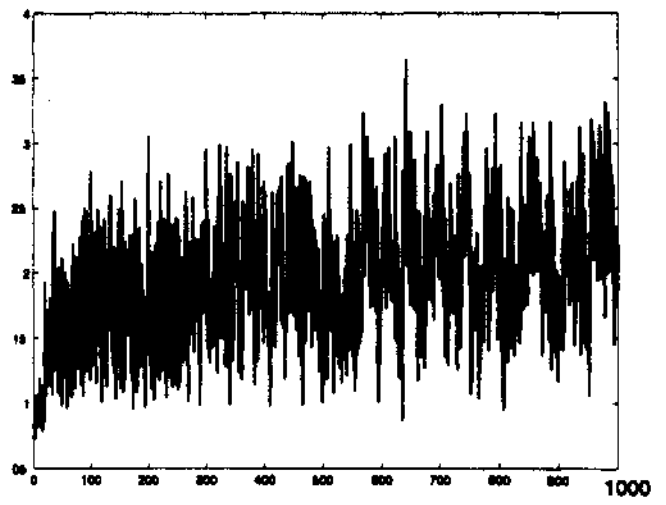


$\kappa_N = 7,3 \cdot 10^{12}$

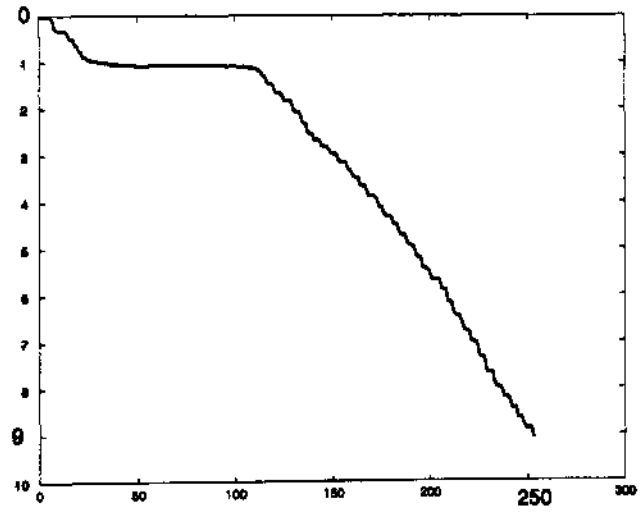
rozložení vlastní čísel

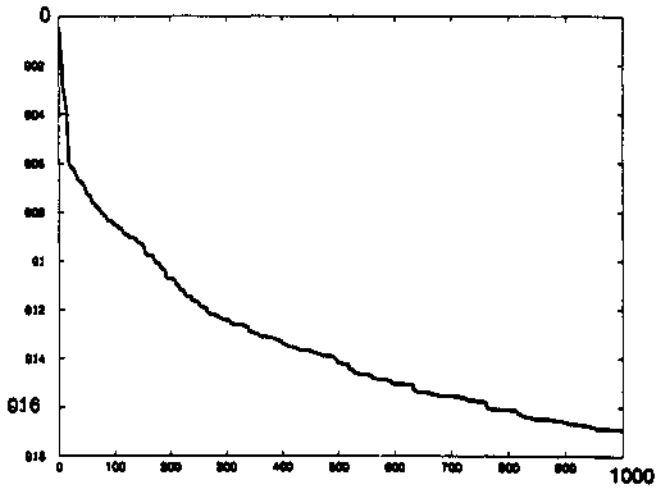
řešení pomocí metody CGM





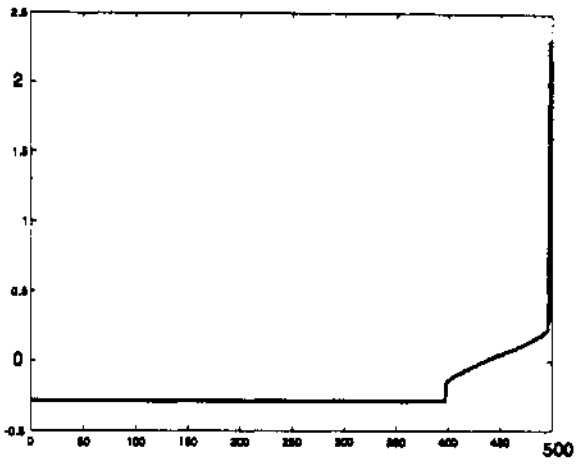
řešení pomocí metody QMR



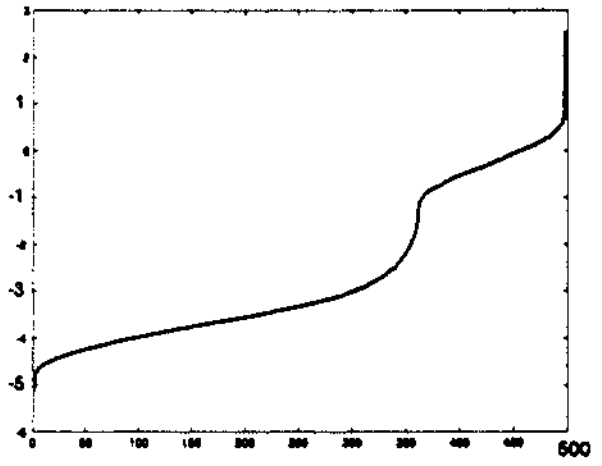


THETA2

první příklad sekvence



poslední příklad sekvence

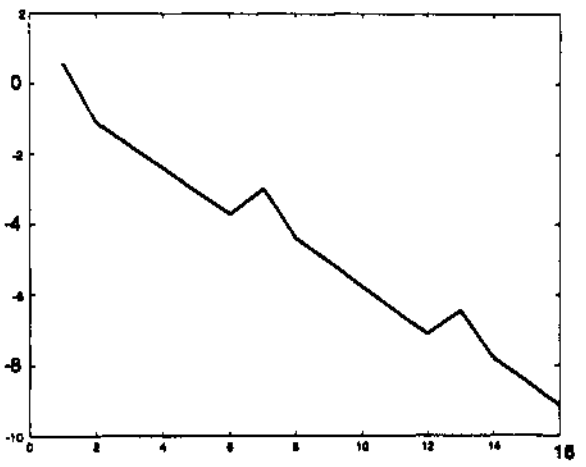


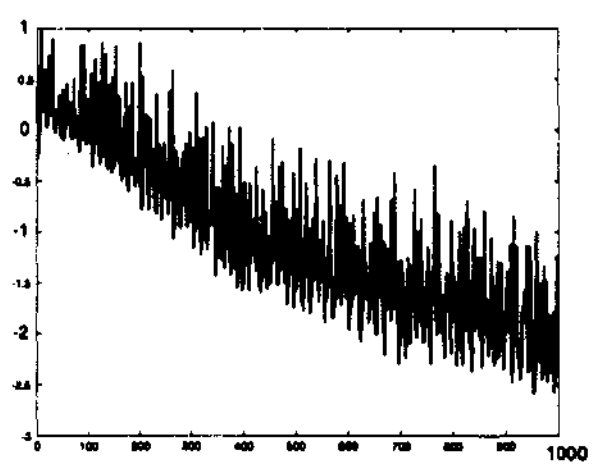
rozložení vlastní čísel

$\kappa_1 = 394$

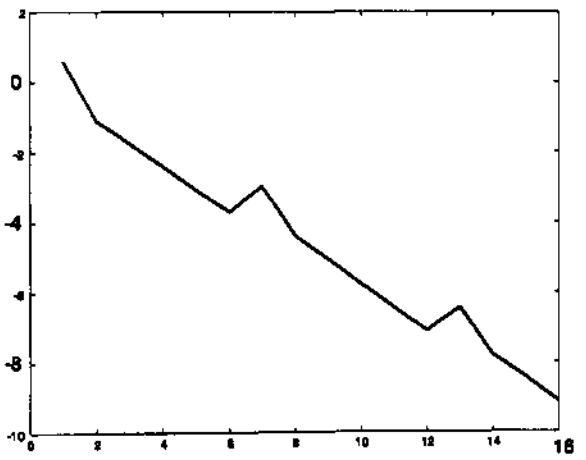
$\kappa_N = 4,9 \cdot 10^7$

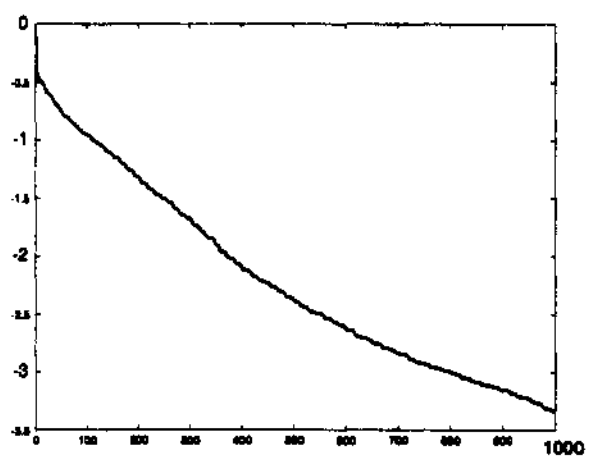
řešení pomocí metody CGM



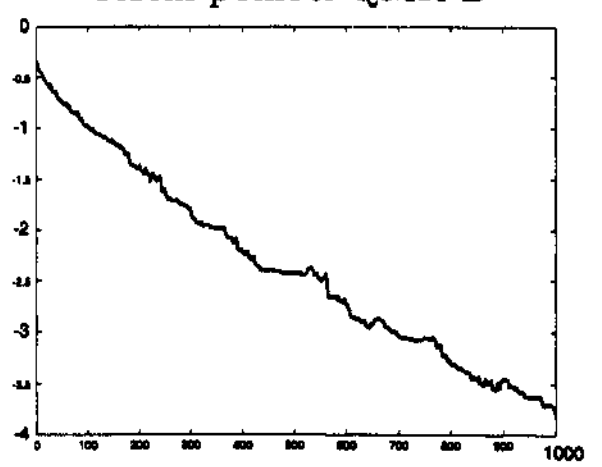


řešení pomocí metody QMR





řešení pomocí QMR D



6.2.4 Použití přibližného hessiánu v úlohách SDP

Motivací pro nás budiž úloha uvedená v [14]. Jedná se o úlohu ze strukturální optimalizace; patří do třídy úloh nelineárního SDP. V takových je teoretická složitost výpočtu hessiánu řádu $O(K * n^3)$.

Výpočet úlohy o velikosti 400 elementů trval na počítači Intel Pentium 4 2,4 GHz přes 8 hodin. To by znamenalo, že pro úlohu o 1000 elementech by výpočet trval přibližně 130 hodin. Naprostá většina tohoto času by byla strávená výpočtem hessiánu, proto se nalezením jeho vhodné aproximace zabýváme.

V této práci jsme vyzkoušeli na úlohách lineárního semidefinitního programování aproximaci pomocí konečných diferencí, výsledky jsou vidět v následující tabulce. Chování by se dalo označit za velmi dobré. U řady úloh aproximace způsobila jen minimálním zvýšení počtu Cg iterací. Navíc se v některých uvedených úlohách dá pozorovat efekt, že i přes zvýšení celkového počtu Cg iterací se metoda výrazně zrychlí. To je způsobeno tím, že výpočet hessiánu je dražší než výpočet gradientů. Pokud tedy nebyl přírůstek Cg iterací velký, celý výpočet se může zrychlit.

	CGM			ACGM			ACGM B16		
	Nw	Cg	t	Nw	Cg	t	Nw	Cg	t
control3	68	9553	3	80	18442	15	65	8429	7
control4	108	33837	15	858	406212	1005	124	42680	112
gpp124-2	35	188	1	35	200	4	35	163	3
gpp250-2	38	221	12	38	235	28	39	215	26
gpp250-3	35	208	11	35	229	27	35	141	19
gpp500-3	38	177	76	38	195	185	38	151	156
mcp250-2	46	302	4	46	326	12	46	180	8
mcp250-3	45	236	10	45	244	30	45	163	23
mcp500-2	51	423	20	51	465	91	50	234	53
mcp500-3	48	250	62	49	268	192	49	160	134
theta2	50	339	4	50	448	4	51	360	3
theta3	63	367	18	62	437	12	63	400	11
theta4	64	421	62	64	520	33	64	560	36
theta5	62	412	200	63	529	63	63	487	62
theta6	59	420	448	59	546	110	58	539	111
truss6	115	5979	1	217	21208	25	133	16916	23
truss7	129	3755	0			LIM	157	8636	8
truss8	108	17050	66	725	80802	801	86	28695	272
vibra2	104	5254	4			TLM	706	190067	1141

6.3 Úlohy konvexního nelineárního programování

Zatímco jsme v úlohách lineárního semidefinitního programování pozorovali zlepšení poměrně často, v úlohách nelineárního programování je tomu přesně naopak. Zlepšení nastává u úloh, o kterých jsme se zmiňovali v úvodní kapitole. Tedy v takových úlohách, ve kterých vede řídký hessián na hustý Choleského faktor, a v úlohách, které jsou velmi dobře podmíněné, a tak stačí jen málo Cg iterací.

První jmenovanou možnost lze sledovat na úlohách **hues-mod** a **huestis**. V nich byly paměťové nároky při rozkladu hessiánu natolik velké, že došlo k havárii původní metody (viz tabulka – „mem. err"). V takovém případě jsou jen dvě možná řešení – použít nějakou iterační metodu místo Choleského rozkladu (jako jsme to provedli v této práci) nebo převést problematickou úlohu do jiného tvaru.

Případ dobře podmíněné metody můžeme sledovat na úloze **cvxqp2_l** (a podobných), průměrný počet 533 iterací během jednoho Nw kroku (u dimenze 10 000) je velmi slušný výsledek.

V jiných případech zlepšení vůči původnímu algoritmu nenastává. Na druhou stranu se není čemu divit, pokud uvážíme, že Choleského rozklad je přizpůsoben právě na takový typ úloh (např. **cont***).

Uvádíme tabulky vybraných výsledků.

			ORIG			CGM				
	n	m	Pn	Nw	t	Pn	Nw	Cg		t
aug2d	20200	10000	9	10	5	11	11	1365		13
aug3d	3873	1000	5	6	1	9	6	128	*	0
cont-050	2597	2401	15	24	2	17	22	11423		12
cont-200	40397	39601	23	34	71					TLM
cvxqp1_m	1000	500	14	25	1	13	24	13136	*	2
cvxqp2_m	1000	250	12	22	1	12	22	3714	*	1
cvxqp3_m	1000	750	18	37	4	18	53	63368	*	11
cvxqp1_l	10000	5000	20	28	582	19	63	411654	*	1546
cvxqp2_l	10000	2500	12	27	190	19	33	18253	*	80
cvxqp3_l	10000	7500	14	25	503	15	32	69057	*	307
hues-mod	10000	2	mem. err			10	47	2275		12
huestis	10000	2	mem. err			10	47	2280		11
qbandm	401	246	10	37	0	17	310	251876	*	32
qship04s	1291	241	13	43	3	16	57	46853	*	86
qship12s	1996	417	14	46	5	14	67	92946		193
qship08l	3149	520	14	46	30	15	70	90557	*	556
qship12l	4224	687	14	48	33	16	87	263543		1877

			ORIG		CGM		
	n	m	t	t/Nw	t	t/Nw	Cg/Nw
aug2d	20200	10000	5	0,50	13	1,18	124
aug3d	3873	1000	1	0,17	0	0,00	21
cont-050	2597	2401	2	0,08	12	0,55	519
cont-200	40397	39601	71	2,09	TLM		
cvxqp1_m	1000	500	1	0,04	2	0,08	547
cvxqp2_m	1000	250	1	0,05	1	0,05	169
cvxqp3_m	1000	750	4	0,11	11	0,21	1196
cvxqp1_l	10000	5000	582	20,79	1546	24,54	6534
cvxqp2_l	10000	2500	190	7,04	80	2,42	553
cvxqp3_l	10000	7500	503	20,12	307	9,59	2158
hues-mod	10000	2	mem. err		12	0,26	48
huestis	10000	2	mem. err		11	0,23	49
qbandm	401	246	0	0,00	32	0,10	813
qship04s	1291	241	3	0,07	86	1,51	822
qship12s	1996	417	5	0,11	193	2,88	1387
qship08l	3149	520	30	0,65	556	7,94	1294
qship12l	4224	687	33	0,69	1877	21,57	3029

Dále uvádíme tabulku, ve které je porovnáno několik předpodmiňovačů. Ze srovnání je vidět, že opět nelze žádný předpodmiňovač zcela jednoznačně doporučit. Jako vcelku dobrý se jeví CGM SGS. Bohužel je poněkud drahý na výpočet, takže aby se jeho efekt skutečně projevil i na celkovém čase, musí být snížení iterací poměrně razantní. Trochu překvapující je chování metody QMR, na základě výsledků u úloh SDP by se daly očekávat vyrovnanější výkony.

	CGM		QMR		CGM Diag		QMR Diag		CGM SGS	
	Cg	t	Cg	t	Cg	t	Cg	t	Cg	t
aug2d	1365	13	1321	18	1318	15	1057	15	527	11
aug3d	128	0	121	0	122	1	117	1	81	1
cont-050	11423	12	10969	16	10177	13	9972	15	3768	8
cvxqp1_m	13136	2	9001	2	8418	3	4310	2	4367	2
cvxqp2_m	3714	1	2429	2	2592	1	2177	1	1557	1
cvxqp3_m	63368	11	65671	14	26076	5	17241	5	11422	5
cvxqp1_l	411654	1546	370886	2318	138110	732	404897	2447	65404	660
cvxqp2_l	18253	80	6272	44	8507	53	5005	41	4731	53
cvxqp3_l	69057	307	92645	605	13079	85	11714	93	6561	84
hues-mod	2275	12	1823	13	65	2	65	2	65	2
huestis	2280	11	1940	14	65	2	65	2	65	2
qbandm	251876	32	22565	3	38413	5	18917	3	24695	5
qshell	334343	314	87668	98	318510	299	105715	118	228567	437
qship04s	46853	86	62849	126	35296	67	23644	48	25368	111
qship12s	92946	193	83996	200	69634	154	65965	157	83502	413
qship08l	90557	556	118218	777	88747	571	48190	327	72030	1104

Na závěr části věnované konvexním úlohám nelineárního programování srovnejme chování různých nastavení téhož předpodmiňovače. Pro předpodmiňovač L-BFGS je jediným parametrem počet dvojic, ze kterých se L-BFGS generuje. Z tohoto vzorku úloh bohužel nelze učinit žádný významnější závěr.

	CGM		BFGS(8)		BFGS(16)		BFGS(32)	
	Cg	t	Cg	t	Cg	t	Cg	t
cont-050	11423	12	10066	22	9588	35	6254	40
cvxqp1_m	13136	2	14130	6	18896	17	13844	27
cvxqp2_m	3714	1	3679	2	7180	7	6777	14
cvxqp3_m	63368	11	64005	33	62760	60	55381	112
qbandm	251876	32	607179	112	365428	116	391246	248
qpcblend	6322	0	7989	1	6101	0	4110	0
qship04s	46853	86	30295	69	33913	98	33313	140
qship12s	92946	193	53438	153	60353	242	52644	321
qship08l	90557	556	64318	504	77911	715	58681	758
qship12l	263543	1877	165030	1533	124123	1470	97599	1569

Předpodmiňovač AINV je pro nás trochu zklamáním. U předpodmiňovačů založených na neúplném rozkladu lze čekat, že budou poměrně robusní a budou mít dobré výsledky i u špatně podmíněných matic. V případě AINVu zůstává až příliš nedořešených otázek – v jaké fázi vypouštět prvky matice, zda volit nějakou strategii pro úpravu velikosti vypouštěných prvků, zda nějakým způsobem omezovat paměť ... To jsou všechno otázky, na které není zatím jednoznačná odpověď. Z toho, co jsme zkoušeli v rámci této práce, se jako nejvhodnější jevila volba hodně vysokého vypouštěcího kritéria. I to však mělo jen nejisté úspěchy.

CGM	CGM		AINV(D1)		AINV(N1)		AINV(D3)		AINV(N3)	
	Cg	t	Cg	t	Cg	t	Cg	t	Cg	t
cont-050	11423	12	8659	83	8562	68	10320	21	10409	21
cvxqp1_m	13136	2	4618	13	7490	30	8194	4	7235	5
cvxqp2_m	3714	1	2751	9	4376	16	2803	3	2382	3
cvxqp3_m	63368	11	9839	17	10820	41	26923	9	21051	10
hues-mod	2275	12	65	185			65	184	65	145
huestis	2280	11	65	187			65	184	65	148
qpcblend	6322	0	5841	1	5030	0	4486	0	3912	0

QMR	CGM		AINV(D1)		AINV(N1)		AINV(D3)		AINV(N3)	
	Cg	t	Cg	t	Cg	t	Cg	t	Cg	t
cont-050	11423	12	6215	77	5993	56	10182	25	10165	25
cvxqp1_m	13136	2	3235	13	4839	27	4307	3	5075	4
cvxqp2_m	3714	1	1823	10	5432	24	2292	3	1875	3
cvxqp3_m	63368	11	7073	19	7240	40	17763	9	15463	10
hues-mod	2275	12					65	184	65	152
huestis	2280	11					65	184	65	147
qpcblend	6322	0	4011	0	4864	0	3094	0	3889	0

6.4 Úlohy nekonvexního nelineárního programování

Hlavní obtíží při testování nekonvexních úloh je fakt, že může existovat několik lokálních minim a nijak nelze ovlivnit to, k jakému z nich mají metody konvergovat. Vzniká tak problém neporovnatelnosti výsledků. Navíc výsledky modifikované Newtonovy metody a metody s lokálně omezeným krokem jsou založeny na velmi odlišném přístupu, takže nelze čekat například ani řádově stejný počet Nw kroků. Pro ilustraci zde uvádíme několik výsledků a „srovnání" různých předpodmiňovačů. Zjednodušeně se dá říci, že pokud metoda alespoň nějak konverguje k numerickému řešení, konverguje docela dobře a v řádově srovnatelném čase jako původní algoritmus.

			original			TR CGM SGS			
problem	n	m	Pn	Nw	t	Pn	Nw	Cg	t
lukvli7	50000	4	11	22	21	10	76	69	29
lukvli9	250000	6	6	15	35	5	24	27	32
lukvli12	49997	37497	15	25	23	15	171	584	87
lukvli13	49997	33330	17	30	23	15	109	228	54
lukvli16	49997	37497	19	37	27	19	130	213	56
lukvli16_1	249997	187497	20	39	144	20	134	214	287

	CGM0		CGM_diag		CGM_SGS	
problem	Cg	t	Cg	t	Cg	t
lukvli7	2667	66	245	91	69	29
lukvli9	1880	494	26	32	27	32
lukvli12	1585	111	1190	99	584	87
lukvli13	414	46	294	45	228	54
lukvli16	563	53	415	57	213	56
lukvli16_1	780	342	351	276	214	287

Kapitola 7

Závěr

Na závěr této práce shrňme a zhodnoťme stručně její výsledky. Hlavním cílem bylo zejména otestovat a porovnat několik alternativních přístupů pro řešení úloh nepodmíněné optimalizace, které vznikají z úloh podmíněné optimalizace použitím zobecněné metody rozšířeného lagrangiánu. Ukázalo se, že iterační metody jsou úspěšnou alternativou na řešení zejména těch úloh, které působící problémy ve většině podobných případů používané Newtonově metodě.

Konkrétně jsme vyzkoušeli metodu spádových směrů v kombinaci s metodami CGM a QMR pro konvexní případy a metodou Steihaga-Tointa pro nekonvexní případy. Pro zlepšení chování metod byly vyzkoušeny různé předpodmiňovače. Na základě numerických výsledků (kapitola 6) nemůžeme jednoznačně žádný předpodmiňovač považovat za dostatečně univerzální, nicméně v řadě případů je použití nepředpodmíněných variant iteračních metod dostatečně dobrou volbou.

Literatura

[1] Axelsson O. (1966): Iterative Solution Methods. Cambridge University Press, New York.

[2] Barrett R., Berry M., Chan T. F., Demmel J., Donato J., Dongarra J., Eijkhout V., Pozo R., Romine C., Van der Vorst H. (1994): Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, Philadelphia. K dispozici též volně na `ftp://ftp.netlib.org/templates/templates.ps`.

[3] Benzi M., Cullum J. K., Tůma M. (2000): Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal Scientific Computing* **22**, 1318–1332.

[4] Benzi M., Tůma M. (2001): A robust incomplete factorization preconditioner for positive definite matrices. *Numerical Linear Algebra with Applications* **99**, 1–20.

[5] Bonnans J.F., Gilbert J.C., Lemaréchal C., Sagastizábal, C.A. (2003): Numerical Optimization Theoretical and Practical Aspects Series. Springer, Heidelberg.

[6] Breitfeld M. G., Shanno D. F. (1994): A globally convergent penalty-barrier algorithm for nonlinear programming. *Rutcor Reasearch Report* **12-94**.

[7] Conn A. R., Gould N. I. M., Toint P. L. (2000): Trust-region methods. SIAM, Philadelphia.

[8] Freund R. W., Nachtigal N. M. (1991): QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik* **60**, 315–339.

[9] Freund R. W., Nachtigal N. M. (1994): An Implementation of the QMR Method Based on Coupled Two-Term Recurrences. *SIAM Journal on Scientific Computing* **15**, 313–337.

[10] Freund R. W., Nachtigal N. M. (1995): Software for Simplified Lanczos and QMR Algorithms. *Applied Numerical Mathematics* **19**, 319–341.

[11] Freund R. W., Nachtigal N. M. (1996): QMRPACK a Package of QMR Algorithms. *ACM Transactions on Mathematical Software* **22**, 46–77.

[12] Golub G. H., Van Loan Ch. F. (1996): Matrix Computation. The Johns Hopkins University Press, Baltimore and London.

[13] Kočvara M., Stingl M. (2003): PENNON - A Code for Convex Nonlinear and Semidefinite Programming. *Optimization Methods and Software* **18(3)**, 317 333.

[14] Kočvara M., Stingl M. (2004): Solving nonconvex SDP problems of structural optimization with stability control. Preprint 300, Institut für Angewandte Mathematik, Universität Erlangen.

[15] Liu J. W.-H. (1985): Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software* **11**, 141 153.

[16] Luenberger D. G. (1973): Introduction to linear and nonlinear programming. Addison-Wesley, New York.

[17] Morales J. L., Nocedal J. (2000): Automatic preconditioning by limited memory quasi-Newton updating. *SIAM Journal on Optimization* **10(4)**, 1079 1096.

[18] Ng E., Peyton B. W. (1993): Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM Journal Scientific Computing* **14**, 1034 1056.

[19] Nocedal J., Wright S. J. (1999): Numerical Optimization. Springer, New-York.

[20] Stingl M. (2004): On the Solution of Nonlinear Semidefinite Programs by Augmented Lagrangian Methods. Doktorská disertační práce, Institut für Angewandte Mathematik, Universität Erlangen.

Numerické výsledky - lineární semidefinitní programování

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM Pn	CGM Nw	CGM Cg	CGM t	CGM D Pn	CGM D Nw	CGM D Cg	CGM D t	CGM GS Pn	CGM GS Nw	CGM GS Cg	CGM GS t	QMR Pn	QMR Nw	QMR Cg	QMR t	QMR D Pn	QMR D Nw	QMR D Cg	QMR D t
theta1	104	50	13	45	0	13	48	271	0	13	48	273	0	13	48	152	1	13	49	250	0	13	48	254	0
theta2	498	100	12	47	3	12	50	339	2	12	51	389	2	12	49	175	2	12	51	300	2	12	51	324	2
theta3	1106	150	12	52	24	12	63	367	9	12	60	421	10	12	58	187	9	13	65	334	10	12	61	479	11
theta4	1949	200	12	57	121	12	64	421	33	12	61	562	34	12	58	237	31	12	65	384	32	12	61	591	36
theta5	3028	250	12	61	446	12	62	412	97	12	61	824	113	12	67	1340	190	12	69	393	100	12	65	794	117
theta6	4375	300	12	52	1115	13	59	420	204	13	61	573	225	13	61	248	213	13	60	303	201	13	60	522	221
thetaG11	2401	801	21	99	480	20	133	1133	256	20	130	1392	259				T15min	20	138	1086	264	20	143	2090	309
control1	21	15	10	45	0	10	51	1085	0	11	53	1137	0	11	53	839	0	11	52	1075	0	11	53	1109	0
control2	66	30	12	66	0	12	109	11986	1	12	101	10694	1	12	78	5470	1	12	121	12688	0	12	103	9621	0
control3	136	45	11	63	1	11	68	9553	2	11	73	8398	1	11	74	5393	2	11	68	7047	1	11	72	6979	1
control4	231	60	11	110	5	11	108	33837	14	11	121	32771	14	11	118	22786	15	11	109	24114	11	11	117	23669	11
control5	351	75	11	88	16	11	94	45118	46	11	109	40989	45	11	107	27119	51	11	92	33309	38	11	111	35222	43
control6	496	90	11	120	66	10	108	71735	155	11	146	77827	183	11	128	44830	181	10	103	44521	114	10	139	53141	146
control7	666	105	15	132	150	14	103	61737	257	14	131	60592	285	14	132	43016	329	13	99	41656	206	14	122	45403	240
control8	861	120	14	166	359	14	104	89948	562	14	132	78154	575	14	135	47352	612	14	105	55792	432	14	130	57166	489
control9	1081	135	16	198	759	16	161	155509	1535	16	189	145037	1577				T35min	16	184	82837	1159	15	212	89745	1300
control10	1326	150	16	235	1463				T35min				T35min				T35min				T35min				T35min

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM B8 Pn	CGM B8 Nw	CGM B8 Cg	CGM B8 t	CGM B16 Pn	CGM B16 Nw	CGM B16 Cg	CGM B16 t	CGM B32 Pn	CGM B32 Nw	CGM B32 Cg	CGM B32 t	QMR AD3 Pn	QMR AD3 Nw	QMR AD3 Cg	QMR AD3 t	QMR AN3 Pn	QMR AN3 Nw	QMR AN3 Cg	QMR AN3 t
theta1	104	50	13	45	0					13	50	238	0	13	50	245	0								
theta2	498	100	12	47	3					12	51	286	2	12	51	307	3								
theta3	1106	150	12	52	24					12	63	290	10	12	63	294	10								
theta4	1949	200	12	57	121					12	63	346	31	12	63	340	30								
theta5	3028	250	12	61	446					12	67	332	94	12	67	343	95								
theta6	4375	300	12	52	1115					13	59	334	196	13	59	335	197								
thetaG11	2401	801	21	99	480																				
control1	21	15	10	45	0	10	50	820	0	10	49	588	0	10	51	582	0	11	54	1154	0	11	51	1034	0
control2	66	30	12	66	0	12	81	5238	0	12	98	6013	0	12	77	3315	0	11	103	8962	1	12	99	10237	0
control3	136	45	11	63	1	11	70	6814	1	11	62	4813	1	11	63	4665	1	11	67	6373	2	11	69	7818	2
control4	231	60	11	110	5	11	96	24714	12	11	107	27328	14	11	103	22757	14	11	109	24041	12	11	130	30572	15
control5	351	75	11	88	16	11	75	25525	31	11	81	29225	36	11	78	22398	32	11	107	42786	54	11	118	45023	58
control6	496	90	11	120	66	11	92	51124	121	11	84	40273	105	11	87	38046	106	10	186	127239	331	10	190	122191	328
control7	666	105	15	132	150					14	106	50044	242	14	94	38826	208								
control8	861	120	14	166	359					14	96	62757	455	14	100	59777	467								
control9	1081	135	16	198	759					16	128	89208	1031	16	136	81288	1037								
control10	1326	150	16	235	1463								T35min				T35min								

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM AD3 Pn	CGM AD3 Nw	CGM AD3 Cg	CGM AD3 t	QMR AD4 Pn	QMR AD4 Nw	QMR AD4 Cg	QMR AD4 t	QMR AN4 Pn	QMR AN4 Nw	QMR AN4 Cg	QMR AN4 t	CGM AN3 Pn	CGM AN3 Nw	CGM AN3 Cg	CGM AN3 t
theta1	104	50	13	45	0	13	48	257	1	13	48	252	0	13	48	245	1	13	49	271	0
theta2	498	100	12	47	3	12	51	378	2	12	51	324	3	12	51	324	3	12	51	383	2
theta3	1106	150	12	52	24	12	60	443	14	12	61	479	14	12	61	482	13	12	60	462	13
theta4	1949	200	12	57	121	12	61	577	45	12	61	592	48	12	61	587	44	12	61	560	45
theta5	3028	250	12	61	446	12	63	788	141	12	65	783	150	12	65	794	138	12	61	771	134
theta6	4375	300	12	52	1115	13	61	575	302	13	60	522	308	13	60	521	285	13	61	573	298
thetaG11	2401	801	21	99	480																
control1	21	15	10	45	0	11	53	1111	0	11	53	1114	0	11	51	1079	0	10	52	1090	0
control2	66	30	12	66	0	12	85	8082	1	11	92	7050	0	13	118	12721	1	12	105	11615	0
control3	136	45	11	63	1	11	73	8912	2	11	68	6399	2	11	69	7525	2	11	76	10192	2
control4	231	60	11	110	5	11	128	36517	16	11	117	23632	12	11	119	29916	14	11	138	40081	18
control5	351	75	11	88	16	11	133	62256	73	11	108	37073	45	11	108	43646	54	11	128	55653	67
control6	496	90	11	120	66	10	193	133266	336	10	132	59646	160	10	193	132379	357	10	226	162970	419
control7	666	105	15	132	150	14	158	101597	478	14	126	54407	278	14	147	97327	461	14	184	114954	542
control8	861	120	14	166	359	14	141	92735	678	14	133	63712	532	14	135	66262	570	14	137	84105	643
control9	1081	135	16	198	759	16	197	154344	1714	16	233	105595	1507	15	200	111337	1482	15	190	154277	1751
control10	1326	150	16	235	1463				T35min				T25min				T35min				T35min

Numerické výsledky - lineární semidefinitní programování

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM Pn	CGM Nw	CGM Cg	CGM t	CGM D Pn	CGM D Nw	CGM D Cg	CGM D t	CGM B16 Pn	CGM B16 Nw	CGM B16 Cg	CGM B16 t	CGM GS Pn	CGM GS Nw	CGM GS Cg	CGM GS t	QMR Pn	QMR Nw	QMR Cg	QMR t	QMR D Pn	QMR D Nw	QMR D Cg	QMR D t	QMR AN4 Pn	QMR AN4 Nw	QMR AN4 Cg	QMR AN4 t
arch0	174	335	12	68	7				LIM	12	74	690	7	12	130	12862	36	12	75	405	7				LIM	12	79	693	7	12	78	684	8
arch2	174	335	12	59	7	12	70	1467	9	12	71	575	8	12	72	1083	9	12	71	367	9	12	69	1170	9	12	70	527	9	12	69	519	9
arch4	174	335	12	69	9	12	74	1512	9	12	75	629	9	12	74	983	9	12	76	383	9	12	72	1212	9	12	74	566	9	12	74	569	9
arch8	174	335	1	1	1				LIM				LIM				LIM				LIM				LIM				LIM				LIM
buck1	36	85	12	45	1	12	44	832	0	12	46	463	0	12	45	603	0	12	45	303	0	12	46	800	0	12	45	437	0	12	45	472	0
buck2	144	337	16	80	2	16	81	3252	2	16	84	785	3	16	82	3160	3	16	85	521	3	16	84	2646	3	16	84	661	3	16	82	652	3
buck3	544	1185	22	200	129				T20min	22	225	16692	175				T20min	22	228	13966	231	22	273	163787	642	22	233	7336	152	22	232	39643	314
gpp100	101	100	11	34	0	11	32	146	1	11	33	133	1	11	32	112	1	11	34	111	1	11	32	135	1	11	33	134	1	11	33	134	0
gpp124-1	125	124	14	32	2	14	40	282	2	14	40	237	1	14	40	221	1	14	40	225	1	14	40	239	1	14	40	222	1	14	40	223	2
gpp124-2	125	124	11	28	1	11	35	188	1	11	35	166	2	11	35	166	1	11	35	154	2	11	35	172	1	11	35	162	1	11	35	163	1
gpp124-3	125	124	10	31	1	9	32	149	2	9	32	134	1	9	33	139	1	9	32	111	2	9	32	136	2	9	32	131	1	9	32	131	1
gpp124-4	125	124	9	34	1	8	32	194	1	8	32	135	1	8	32	140	1	8	32	125	1	8	32	148	1	8	32	137	1	8	32	137	1
gpp250-1	250	250	14	39	12	14	47	298	14	14	47	284	13	14	47	209	13	14	47	262	14	14	46	276	13	14	47	264	14	14	47	263	14
gpp250-2	250	250	11	35	11	12	38	221	12	12	39	182	12	12	38	198	12	12	39	160	12	12	38	199	11	12	39	168	12	12	39	166	12
gpp250-3	250	250	10	35	11	10	35	208	11	10	36	164	10	10	35	169	10	10	36	148	11	10	35	158	10	10	36	156	10	10	36	157	11
gpp250-4	250	250	7	30	8	8	34	156	9	8	34	126	9	8	34	125	10	8	34	125	10	8	34	129	10	8	34	127	10	8	34	127	10
gpp500-1	501	500	15	52	105	15	52	393	105	15	52	374	105	15	52	279	106	15	52	408	107	15	52	359	105	15	52	344	105	15	52	347	110
gpp500-2	501	500	11	48	88	12	44	276	88	12	44	228	88	12	44	236	90	12	44	230	91	12	44	231	89	12	44	215	89	12	44	215	92
gpp500-3	501	500	10	45	84	10	38	177	76	10	38	163	76	10	38	142	76	10	38	162	77	10	38	165	77	10	38	157	77	10	38	157	79
gpp500-4	501	500	10	47	86	9	36	143	72	9	36	148	71	9	36	120	71	9	36	152	72	9	36	137	71	9	36	143	71	9	36	143	74
maxG11	800	800	19	59	56	18	51	567	41	18	50	457	41	18	50	295	39	18	50	263	42	18	51	475	41	18	50	382	40	18	50	380	43
maxG32	2000	2000	18	57	887	18	55	607	702	18	54	457	690	18	58	337	730	18	54	274	696	18	54	509	732	18	54	380	715	18	54	381	699
maxG51	1000	1000	18	73	629	17	72	478	594	17	72	326	583	17	73	381	598	17	71	499	595	17	72	430	579	17	72	309	588	17	72	309	629
mcp100	100	100	15	43	0	14	43	251	0	14	41	157	0	14	41	155	0	14	41	142	0	14	43	242	0	14	41	153	1	14	41	153	1
mcp124-1	124	124	16	49	0	16	42	266	1	16	42	198	1	16	42	176	0	16	42	202	0	16	42	251	1	16	42	187	1	16	42	186	1
mcp124-2	124	124	16	48	1	14	42	258	1	14	42	159	1	14	42	158	0	14	42	139	1	14	42	245	1	14	42	152	1	14	42	152	1
mcp124-3	124	124	15	49	1	14	42	230	2	14	42	185	1	14	42	161	1	14	42	158	1	14	43	228	2	14	43	186	1	14	43	186	2
mcp124-4	124	124	15	49	1	14	45	211	2	14	45	176	1	14	46	146	1	14	46	186	1	14	45	201	1	14	45	170	2	14	45	170	2
mcp250-1	250	250	17	48	3	16	47	306	2	16	47	222	2	16	46	191	2	16	47	222	2	16	47	283	3	16	47	211	2	16	47	209	3
mcp250-2	250	250	15	47	4	15	46	302	4	15	46	175	4	15	46	176	3	15	46	176	3	15	46	277	4	15	46	166	3	15	46	166	4
mcp250-3	250	250	15	48	11	14	45	236	10	14	46	175	11	14	45	162	10	14	47	174	10	14	45	224	11	14	46	167	10	14	46	167	11
mcp250-4	250	250	15	46	11	14	48	217	11	14	48	165	11	14	48	145	11	14	48	186	11	14	48	208	11	14	48	156	11	14	48	156	11
mcp500-1	500	500	17	53	14	16	53	361	12	16	52	255	12	16	51	231	11	16	54	286	14	16	50	323	11	16	52	231	11	16	52	230	12
mcp500-2	500	500	15	61	26	15	51	423	20	15	50	208	20	15	50	230	20	15	50	215	21	15	51	381	20	15	50	196	19	15	50	196	20
mcp500-3	500	500	15	50	72	15	48	250	62	15	48	180	59	15	49	159	63	15	48	213	62	15	49	246	62	15	49	177	62	15	49	177	63
mcp500-4	500	500	15	53	82	14	50	224	74	14	50	162	73	14	51	162	74	14	50	215	75	14	50	215	74	14	50	156	74	14	50	156	78
qap5	136	26	15	55	0	13	47	449	1	13	45	653	0	13	45	317	1	13	47	446	0	14	56	489	0	13	47	641	0	13	51	721	0
qap6	229	37	14	42	1	13	51	1181	1	13	47	762	0	13	47	1120	1	13	44	464	1	12	49	619	0	14	50	766	0	12	45	575	1
qap7	358	50	13	41	2	13	53	1232	3	13	50	756	2	13	52	1261	4	13	48	505	3	17	108	3702	8	14	54	686	2	13	50	681	3
qap8	529	65	15	68	8	13	52	1003	6	13	48	536	4	13	48	731	5	13	46	493	7	15	64	1691	9	13	47	458	4	13	48	364	5
qap9	748	82	13	53	16	13	57	1679	16	13	47	607	9	13	52	1942	18	13	49	537	13	13	55	887	12	13	49	573	9	13	48	407	12
shmup1	16	113	16	57	1	15	59	448	1	15	60	249	0	15	62	419	0	15	60	247	1	15	60	449	1	15	59	234	1	15	61	291	0
shmup2	200	1281	16	85	126	16	89	11971	123	16	86	536	115	16	96	17093	119	16	84	496	102	16	84	6706	105	16	86	478	104	16	86	495	105
ss30	132	426	13	44	22	12	53	373	26	12	52	222	26	12	52	223	26	12	52	215	26	12	52	347	26	12	52	214	26	12	52	213	27
thetaG11	2401	801	21	99	606	20	133	1133	461	20	130	1392	466	20	119	1750	457				T20mn	20	138	1086	472	20	143	2090	542				T20mn
thetaG51	6910	1001			T20min				T20mn				T20mn								T20mn				T20mn				T20mn				T20mn
trto1	36	61	11	54	1	11	58	779	0	11	59	212	0	11	60	573	0	11	59	175	0	11	58	748	0	11	58	207	0	11	58	213	0
trto2	144	235	12	93	2	12	96	4120	2	12	96	659	2	12	115	9028	3	12	104	558	1	12	100	3113	2	12	92	603	1	12	94	659	1
trto3	544	865	13	94	28	13	124	31693	123	13	114	2897	34	13	139	27924	128	13	112	1215	34	13	159	28187	122	13	141	2329	38	13	129	2156	38
trto4	1200	1873	14	143	282				T20mn	14	144	3958	241				T20mn	14	152	1750	261				T20mn	14	172	4132	294	14	177	3815	321
truss1	6	13	5	21	0	5	23	112	0	5	21	99	0	7	22	74	0	5	23	73	0	5	23	115	0	7	23	112	0	9	24	111	0
truss2	58	133	11	43	0	11	59	871	0	11	52	857	0	11	52	749	0	11	49	431	1	11	62	818	0	11	53	745	0	11	51	875	0
truss3	27	31	13	37	0	13	48	901	0	13	42	552	0	14	44	521	0	13	43	342	0	14	49	837	0	13	40	509	0	13	41	436	0
truss4	12	19	7	22	0	8	23	175	0	6	23	177	0	8	23	114	0	8	23	112	0	8	23	170	0	8	23	174	0	6	23	179	0
truss5	208	331	13	54	2	13	84	9458	3	13	62	3096	2	13	77	10096	4	13	61	1476	2	13	84	5744	3	13	75	3197	2	13	63	2263	2
truss6	172	451	7	77	1	7	115	5979	1	7	94	3713	1	7	128	7015	2	7	88	1583	1	7	112	4880	2	7	88	2085	1	7	87	1888	0
truss7	86	301	7	91	0	5	129	3755	0	7	123	2607	1	7	139	3090	0	6	119	1335	0	5	127	3248	0	7	100	1704	0	7	102	1539	0
truss8	496	628	14	64	17	14	106	17050	66	14	72	4132	26	14	88	16487	66	14	70	1905	26	15	112	14140	61	14	82	3586	27	14	72	2676	52
vibra1	36	85	15	70	0	15	74	1347	1	15	73	523	0	15	72	1207	0	15	75	405	0	15	75	1304	0	15	72	422	0	15	72	440	0
vibra2	144	337	18	95	3	18	104	5254	4	18	106	1175	4	18	125	5886	5	18	107	1339	4	18	101	4067	3	18	108	995	3	18	107	1049	4
vibra3	544	1185	21	215	126	20	282	151532	642	21	259	20084	224	21	403	242436	1081	21	228	7074	191	21	235	79735	384	21	231	5382	137	21	197	4335	123
vibra4	1200	2545	21	115	404				T20mn	21	274	7881	962				T20mn	21	190	4168	713				T20mn	21	179	3617	638	21	180	3755	643

Numerické výsledky - lineární semidefinitní programování s přibližným hessiánem

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM Pn	CGM Nw	CGM Cg	CGM t	ACGM Pn	ACGM Nw	ACGM Cg	ACGM t	CGM B16 Pn	CGM B16 Nw	CGM B16 Cg	CGM B16 t	ACGM B16 Pn	ACGM B16 Nw	ACGM B16 Cg	ACGM B16 t	ACGM B32 Pn	ACGM B32 Nw	ACGM B32 Cg	ACGM B32 t
control3	136	45	11	63	J	11	68	9553	3	11	80	18442	15	11	62	4813	J	11	65	8429	7	11	63	7129	7
control4	231	60	11	110	J	11	108	33837	15	12	858	406212	1005	11	107	27328	J	11	124	42680	112	11	93	27354	76
control5	351	75	11	88	J	11	94	45118	89				T20min	11	81	29225	J	11	81	34388	172	11	75	30048	156
gpp124-2	125	124	11	28	1	11	35	188	1	11	35	200	4	11	35	166	1	11	35	163	3	11	35	163	3
gpp124-3	125	124	10	31	1	9	32	149	2	9	32	156	3	9	33	139	1	9	33	142	2	9	33	142	2
gpp250-2	250	250	11	35	11	12	38	221	12	12	38	235	28	12	38	198	12	12	39	215	26	12	39	212	27
gpp250-3	250	250	10	35	11	10	35	208	11	10	35	229	27	10	35	169	10	10	35	141	19	10	35	141	20
gpp500-2	501	500	11	48	88	12	44	276	88	12	44	296	254	12	44	236	90	12	44	245	223	12	44	232	220
gpp500-3	501	500	10	45	84	10	38	177	76	10	38	195	185	10	38	142	76	10	38	151	156	10	38	151	153
mcp124-2	124	124	16	48	1	14	42	258	1	14	42	272	2	14	42	158	0	14	42	159	1	14	42	159	1
mcp124-3	124	124	15	49	1	14	42	230	2	14	42	235	3	14	42	161	1	14	42	163	3	14	42	163	3
mcp250-2	250	250	15	47	4	15	46	302	4	15	46	326	12	15	46	176	3	15	46	180	8	15	46	180	8
mcp250-3	250	250	15	48	11	14	45	236	10	14	45	244	30	14	45	162	10	14	45	163	23	14	45	163	23
mcp500-2	500	500	15	61	26	15	51	423	20	15	51	465	91	15	50	230	20	15	50	234	53	15	50	223	52
mcp500-3	500	500	15	50	72	15	48	250	62	15	49	268	192	15	49	159	63	15	49	160	134	15	49	160	133
theta2	498	100	12	47	J	12	50	339	4	12	50	448	4	12	51	286	J	12	51	360	3	12	51	333	3
theta3	1106	150	12	52	J	12	63	367	18	12	62	437	12	12	63	290	J	12	63	400	11	12	63	421	13
theta4	1949	200	12	57	J	12	64	421	62	12	64	520	33	12	63	346	J	12	64	560	36	12	63	377	25
theta5	3028	250	12	61	J	12	62	412	200	12	63	529	63	12	67	332	J	12	63	487	62	12	63	421	54
theta6	4375	300	12	52	J	13	59	420	448	13	59	546	110	13	59	334	J	13	58	539	111	13	58	421	91
trto2	144	235	12	93	2	12	96	4120	2				T20min	12	115	9028	3	12	145	18288	61	12	138	17988	61
truss5	208	331	13	54	2	13	84	9458	3	12	151	40939	111	13	77	10096	4	13	85	19699	56	13	80	15573	46
truss6	172	451	7	77	1	7	115	5979	1	7	217	21208	25	7	128	7015	2	7	133	16916	23	7	158	20973	34
truss7	86	301	7	91	0	5	129	3755	0				LIM	7	139	3090	0	7	157	8636	8	7	144	5769	6
truss8	496	628	14	64	17	14	108	17050	66	16	725	80802	801	14	88	16487	66	14	86	28695	272	14	92	33670	331
vibra2	144	337	18	95	3	18	104	5254	4				T20min	18	125	5886	5	18	708	190067	1141				T20min

Poznámka: J ... spouštěno na jiném počítači, časové výsledky se nedaji porovnávat

Numerické výsledky - konvexní nelineární programování

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM Pn	CGM Nw	CGM Cg	CGM t	QMR Pn	QMR Nw	QMR Cg	QMR t	QMRP Pn	QMRP Nw	QMRP Cg	QMRP t	CGM D Pn	CGM D Nw	CGM D Cg	CGM D t	QMR D Pn	QMR D Nw	QMR D Cg	QMR D t	QMRP D Pn	QMRP D Nw	QMRP D Cg	QMRP D t	CGM SS(w=1.5) Pn	CGM SS(w=1.5) Nw	CGM SS(w=1.5) Cg	CGM SS(w=1.5) t	CGM GS Pn	CGM GS Nw	CGM GS Cg	CGM GS t
aug2d	20200	10000	9	10	5	11	11	1365	13	12	11	1321	18	12	11	1329	23	13	12	1318	15	12	11	1057	15	13	13	1214	23	12	11	724	15	12	11	527	11
aug3d	3873	1000	5	6	1	9	6	128 *	0	6	5	121 *	0	6	5	121 *	1	11	6	122 *	1	7	6	117 *	1	7	6	126 *	1	9	6	146 *	1	9	6	81 *	1
cvxqp1_m	1000	500	14	25	1	13	24	13136 *	2	14	28	9001 *	2	15	28	11970 *	4	16	28	8418 *	3	14	25	4310 *	2	14	28	4976 *	3	16	28	6274 *	2	16	28	4367 *	2
cvxqp2_m	1000	250	12	22	1	12	22	3714 *	1	12	21	2429 *	2	12	21	2411 *	2	15	25	2592 *	1	16	26	2177 *	1	18	31	2712 *	2	16	25	2337 *	2	16	26	1557 *	1
cvxqp3_m	1000	750	18	37	4	18	53	63368 *	11	21	74	65671 *	14	18	53	44025 *	13	19	52	26076	5	20	57	17241 *	5	18	55	26329	9	18	49	14974	5	19	49	11422	5
primalc1	230	9	10	19	0	18	42	1918 *	0	18	48	2614	1	18	40	2003	1	13	32	1988	0	15	32	1754	1	14	34	2672 *	2	11	392	184262 *	103	11	138	62961 *	29
primalc2	231	7	11	25	0	20	61	2124	1	18	71	2425	1	22	81	2404	2	13	41	1092	0	13	39	1033	1	17	48	1673	1	16	230	106500 *	59	15	81	33478 *	15
qbandm	401	246	10	37	0	17	310	251876 *	32	13	33	22565 *	3	15	35	25161 *	7	13	57	38413 *	5	12	39	18917	3	10	40	16374 *	3	11	46	27597 *	6	13	48	24695 *	5
qpcblend	83	71	34	55	0	33	46	6322 *	0	37	39	4447 *	0	36	43	4940 *	1	34	39	4202 *	0	23	31	2230 *	0	26	37	3156 *	0	34	39	3329 *	0	42	43	2917 *	0
qship04s	1291	241	13	43	3	16	57	46853 *	86	15	84	62849 *	126	17	75	76948 *	160	17	57	35296 *	67	15	67	23644	48	15	62	42535 *	90	14	55	39894 *	172	14	56	25368 *	111
qship12s	1996	417	14	46	5	14	67	92946	193	14	81	83996	200	14	69	96380	238	15	64	69634	154	14	71	65965 *	157	14	67	71412 *	176	13	65	71828	349	13	75	83502 *	413
cvxqp1_l	10000	5000	20	28	582	19	63	411654 *	1546	22	101	370886 *	2318					19	53	138110	732	25	125	404897 *	2447									20	53	65404 *	660
cvxqp2_l	10000	2500	12	27	190	19	33	18253 *	80	12	25	6272 *	44					19	32	8507 *	53	19	32	5005 *	41									18	30	4731 *	53
cvxqp3_l	10000	7500	14	25	503	15	32	69057 *	307	22	66	92645 *	605					13	29	13079 *	85	17	45	11714 *	93									13	29	6561 *	84
qship08l	3149	520	14	46	30	15	70	90557 *	556	20	102	118218 *	777					15	70	88747 *	571	13	82	48190	327									15	74	72030	1104
qship12l	4224	687	14	48	33	16	87	263543	1877																												
cont-050	2597	2401	15	24	2	17	22	11423	12	20	25	10969	16					17	22	10177	13	22	26	9972	15									17	22	3768	8
cont-200	40397	39601	23	34	71	15		79915 x T45m		15		45391 x T45m						15		77208 x T45m		14		17821 x T45m										15		26700 x T45m	
qshell	1476	487	8	122	6	6	138	334343	314	6	62	87668	98					6	134	318510	299	6	67	105715	118									6	114	226567	437
qsierra	2016	1135	13	70	12	16	124	300021 *	460	18	96	121963 *	220					17	171	539139 *	866	15	111	141336 *	255									13	125	232251	854
hues-mod	10000	2	memory err			10	47	2275	12	10	43	1823	13					10	24	65	2	10	24	65	2									10	24	65	2
huestis	10000	2	memory err			10	47	2280	11	10	43	1940	14					10	24	65	2	10	24	65	2									10	24	65	2

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM B8 Pn	CGM B8 Nw	CGM B8 Cg	CGM B8 t	CGM B16 Pn	CGM B16 Nw	CGM B16 Cg	CGM B16 t	CGM B32 Pn	CGM B32 Nw	CGM B32 Cg	CGM B32 t	CGM AD1 Pn	CGM AD1 Nw	CGM AD1 Cg	CGM AD1 t	CGM AD3 Pn	CGM AD3 Nw	CGM AD3 Cg	CGM AD3 t	CGM AN1 Pn	CGM AN1 Nw	CGM AN1 Cg	CGM AN1 t	CGM AN3 Pn	CGM AN3 Nw	CGM AN3 Cg	CGM AN3 t
cvxqp1_m	1000	500	14	25	1	15	25	14130 *	6	16	28	18896 *	17	15	25	13844 *	27	16	28	4618 *	13	16	28	8194 *	4	16	28	7490 *	30	16	28	7235 *	5
cvxqp2_m	1000	250	12	22	1	12	21	3679 *	2	18	26	7180 *	7	18	27	6777 *	14	12	22	2751 *	9	18	27	2803 *	3	12	22	4376 *	16	12	22	2382 *	3
cvxqp3_m	1000	750	16	37	4	19	54	64005 *	33	19	52	62760 *	60	19	49	55381 *	112	18	46	9839	17	19	51	26923	9	18	46	10820 *	41	19	49	21051 *	10
primalc1	230	9	10	19	0	14	43	2185	1	13	30	1651	1	12	30	1579	1	13	728	340880 *	719	13	356	160325 *	173			x					
primalc2	231	7	11	25	0	14	43	1702 *	1	14	40	1258	1	13	36	915 *	1	17	488	228785 *	490					14	398	185763 *	426	16	1368	x 654545 *	935
qbandm	401	246	10	37	0	20	742	607179 *	112	19	450	365428 *	116	21	481	391246 *	248													14	53	34135 *	8
qpcblend	83	71	34	55	0	42	57	7989 *	1	33	47	6101 *	0	37	42	4110 *	0	46	50	5841 *	1	34	40	4486 *	0	36	43	5030 *	0	30	38	3912 *	0
qship04s	1291	241	13	43	3	14	52	30295 *	69	14	54	33913 *	98	14	54	33313 *	140																
qship12s	1996	417	14	46	5	14	57	53438	153	13	59	60353	242	14	57	52644	321																
qship08l	3149	520	14	46	30	15	63	64318 *	504	17	67	77911 *	715	13	58	58681 *	758																
qship12l	4224	687	14	48	33	20	76	165030 *	1533	17	78	124123 *	1470	16	66	97599 *	1569																
cont-050	2597	2401	15	24	2	17	22	10066	22	17	22	9588	35	17	22	6254	40	17	22	8659	83	17	22	10320	21	17	22	8562	68	17	22	10409	21
cont-200	40397	39601	23	34	71					10		14809 x T30m																					
hues-mod	10000	2	memory err			10	56	7518	76	10	58	9290	140	10	56	7996	194	10	24	65	185	10	24	65	184					10	24	65	145
huestis	10000	2	memory err			10	54	6396	64	10	58	7814	114	10	54	8144	197	10	24	65	187	10	24	65	184					10	24	65	148

problem	n	m	ORIG Pn	ORIG Nw	ORIG t	CGM AD1 Pn	CGM AD1 Nw	CGM AD1 Cg	CGM AD1 t	CGM AD3 Pn	CGM AD3 Nw	CGM AD3 Cg	CGM AD3 t	CGM AN1 Pn	CGM AN1 Nw	CGM AN1 Cg	CGM AN1 t	CGM AN3 Pn	CGM AN3 Nw	CGM AN3 Cg	CGM AN3 t	QMR AD1 Pn	QMR AD1 Nw	QMR AD1 Cg	QMR AD1 t	QMR AD3 Pn	QMR AD3 Nw	QMR AD3 Cg	QMR AD3 t	QMR AN1 Pn	QMR AN1 Nw	QMR AN1 Cg	QMR AN1 t	QMR AN3 Pn	QMR AN3 Nw	QMR AN3 Cg	QMR AN3 t
cvxqp1_m	1000	500	14	25	1	16	28	4618 *	13	16	28	6194 *	4	16	28	7490 *	30	16	28	7235 *	5	16	28	3235 *	13	14	25	4307 *	3	16	28	4839 *	27	16	27	5075 *	4
cvxqp2_m	1000	250	12	22	1	12	22	2751 *	9	18	27	2803 *	3	12	22	4376 *	16	12	22	2382 *	3	12	22	1823 *	10	16	26	2292 *	3	19	28	5432 *	24	12	22	1875 *	3
cvxqp3_m	1000	750	18	37	4	18	46	9839	17	19	51	26923	9	18	46	10820 *	41	19	49	21051 *	10	19	51	7073	19	21	58	17783 *	9	20	51	7240 *	40	21	56	15463 *	10
primalc1	230	9	10	19	0	13	728	340880 *	719	13	356	160325 *	173			x								x		17	1113	523595 *	571			x		18	2369	1137117 *	1708
primalc2	231	7	11	25	0	17	488	228785 *	490					14	398	185763 *	426	16	1368	654545 *	935	20	667	316492 *	674	16	183	31899 *	16	16	202	90589 *	183	13	997	473800 *	649
qbandm	401	246	10	37	0													14	53	34135 *	8					13	44	20585	4					14	45	20685	7
qpcblend	83	71	34	55	0	46	50	5841 *	1	34	40	4486 *	0	36	43	5030 *	0	30	38	3912 *	0	34	41	4011 *	0	26	35	3094 *	0	38	42	4864 *	0	43	43	3899 *	0
qship04s	1291	241	13	43	3																																
qship12s	1996	417	14	46	5																																
qship08l	3149	520	14	46	30																																
qship12l	4224	687	14	48	33																																
cont-050	2597	2401	15	24	2	17	22	8659	83	17	22	10320	21	17	22	8562	68	17	22	10409	21	17	22	6215	77	22	26	10182	25	17	22	5993	56	22	26	10165	25
cont-200	40397	39601	23	34	71																																
hues-mod	10000	2	memory err			10	24	65	185	10	24	65	184					10	24	65	145					10	24	65	184					10	24	65	152
huestis	10000	2	memory err			10	24	65	187	10	24	65	184					10	24	65	148					10	24	65	184					10	24	65	147

Numerické výsledky - nekonvexní nelineární programování

			original			TRST				TRST D				TRST GS			
problem	n	m	Pn	Nw	t	Pn	Nw	Cg	t	Pn	Nw	Cg	t	Pn	Nw	Cg	t
ex1	12482	6241	16	31	11				x T20min	50	6573	10974	x LIM	23	208	38462	568
ex3	6962	3481	18	53	9				x T20min	21	1744	40393	* 228	23	1907	12519	* 187
lukvli11	49997	33330	16	27	30	15	87	358	51	15	72	252	47	15	68	119	45
lukvli12	49997	37497	15	25	23	15	213	1585	111	15	206	1190	99	15	171	584	87
lukvli13	49997	33330	17	30	23	15	93	414	46	15	95	294	45	15	109	228	54
lukvli14	49997	33330	9	40	33				x T20min				x T20min				x T20min
lukvli15	49997	37497	17	89	62	19	157	1466	91	19	170	897	87	18	182	660	93
lukvli16	49997	37497	19	37	27	19	112	563	53	19	134	415	57	19	130	213	56
lukvli17	49997	37497	15	40	32	14	301	2906	143	16	688	813	146	16	1049	1493	253
lukvli18	49997	37497	19	36	26	19	121	465	52	19	418	1959	184	19	134	189	54
lukvli3	50000	2	10	15	13	10	39	117	17	9	43	73	* 18	8	34	31	* 14
lukvli5	50000	49996	15	128	293				x T20min				x T20min				x T20min
lukvli7	50000	4	11	22	21	10	78	2667	66	10	491	245	91	10	76	69	29
lukvli8	50000	49998	29	93	144				x T20min				x T20min				x T20min
lukvli9	250000	6	6	15	35	5	455	1880	494	5	26	26	32	5	24	27	32
steering	32000	25600	16	32	31				x T20min				x T20min				x T20min
lukvli16_l	249997	187497	20	39	144	20	155	780	342	20	123	351	276	20	134	214	287
lukvli17_l	249997	187497	16	38	147	16	291	2212	* 668	16	1066	1398	1043	17	984	746	* 1030
lukvli3_l	250000	2	11	20	91	9	37	99	* 88	9	43	71	106	11	48	38	* 113