FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITY KOMENSKÉHO V BRATISLAVE

Katedra aplikovanej matematiky a štatistiky

Riešenie úloh nelineárneho programovania pomocou metód s ohraničeným krokom

(Diplomová práca)

Martin Mrázek

Diplomový vedúci: Doc.RNDr.Milan Hamala,CSc. Bratislava, 2007

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

Abstrakt

Témou tejto práce je riešenie úloh na voľný extrém metódami s ohraničeným krokom. Je vysvetlená základná myšlienka týchto metód a ukázané možnosti hľadania lokálne ohraničeného kroku v prípade aproximácie kvadratickou funkciou. Využitím kvázinewtonovských formúl na získavanie aproximácie Hessovej matice sú ukázané možnosti transformácie podúlohy na úlohu menších rozmerov. Na konci je realizovaný numerický experiment.

Kľúčové slová: metódy s ohraničeným krokom, nelineárne programovanie

Obsah

Úvod

V tejto práci sa budeme zaoberať minimalizáciou funkcie $f : \mathbb{R}^n \mapsto \mathbb{R}$ na neohraničenej oblasti

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

Úloha sa zväčša rieši niektorou z iteračných metód, ktoré začínajúc štartovacím bodom x_0 sa snažia generovať postupnosť bodov $x_0, x_1, \ldots$ konvergujúcu k optimálnemu riešeniu $\hat{x}$. Táto postupnosť je daná vzorcom $x_{k+1} = x_k + p_k$, kde posun p_k je získaný pomocou nejakého algoritmu.

Väčšinou sa posun p_k získava niektorou z metód s reguláciou dĺžky kroku, t.j. takých, ktoré sa najprv snažia vytýčiť smer s_k a potom hľadajú krok λ_k tak, aby účelová funkcia v smere s_k nadobudla čo najmenšiu hodnotu. Druhý možný spôsob je najprv si zvoliť maximálnu dĺžku kroku Δ_k o ktorý sa môžeme posunúť a následne sa snažiť nájsť posun p_k nepresahujúci zvolenú oblasť. Takéto metódy sa nazývajú metódami s ohraničeným krokom.

Metódy s ohraničeným krokom majú veľmi dobré konvergenčné vlastnosti a vďaka ohraničenosti kroku možno za aproximáciu účelovej funkcie voliť aj nekonvexné funkcie. My sa budeme venovať metódam, v ktorých sa za aproximáciu účelovej funkcie volí kvadratická funkcia

$$f(x) \approx \psi_k(p) = \frac{1}{2} p^T B_k p + g_k^T p$$

V každej iterácii sa potom snažíme hľadať posun p_k ako minimum aproximovanej funkcie $\psi_k(p)$ na ohraničenej oblasti tak, aby platilo $\|p_k\|_2 \leq \Delta_k$.

Práca je členená následovne: v prvej časti si ukážeme základnú ideu metód s ohraničeným krokom. V druhej časti si ukážeme rôzne spôsoby riešenia podúlohy

$$\min_{p \in \mathbb{R}^n} \psi_k(p) = \frac{1}{2} p^T B_k p + g_k^T p$$
$$\|p\|_2 \leq \Delta_k$$

pričom si rozoberieme teoretické vlastnosti presného riešenia. Tieto využijeme v tretej časti práce, kde sa zameriame na možnosť transformácie úlohy na úlohu s podstatne menším rozmerom a navrhneme možný spôsob získavania posunu p_k v tomto podpriestore. Nakoniec si teoretické poznatky overíme na numerickom experimente.

1 Metódy s ohraničeným krokom

1.1 Úloha na voľný extrém

Budeme sa zaoberať úlohou matematického programovania na voľný extrém

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.1}$$

pričom o účelovej funkcii $f(x)$ predpokladáme, že je spojite diferencovateľná a zdola ohraničená.

Väčšina metód riešenia úlohy (1.1) sú iteračné, t.j. začínajúc zo zadaného štartovacieho bodu x_0 generujú postupnosť bodov $\{x_k\}_{k=0}^{\infty}$ danú iteračným vzorcom

$$x_{k+1} = x_k + p_k \tag{1.2}$$

konvergujúcu k optimálnemu riešeniu. Od jednotlivých iterácií sa pritom očakáva pokles hodnoty účelovej funkcie

$$f(x_{k+1}) < f(x_k) \tag{1.3}$$

Vektor korekcie p_k zo vzorca (1.2) možno voliť podľa dvoch schém[4]

1. *s reguláciou dĺžky kroku*, kde v prvej fáze sa zvolí spádový smer s_k a v druhej dĺžka kroku λ_k, posun je potom daný $p_k = \lambda_k s_k$ a iteračný proces vyzerá $x_{k+1} = x_k + \lambda_k s_k$

2. *s reguláciou smeru*, kde v prvej fáze zvolíme dĺžku kroku Δ_k a v druhej hľadáme posun p_k tak, aby platilo $\|p_k\| \leq \Delta_k$, teda proces je daný vzorcom $x_{k+1} = x_k + p_k$

My sa budeme venovať druhej schéme, t.j. spádovými metódami s reguláciou smeru, ktoré sú známe ako metódy s ohraničeným krokom[4](v angličtine *Trust-Region Methods* alebo *Restricted-step Methods*).

Metódy s ohraničeným krokom sú relatívne novšie oproti metódam založeným na 1. schéme. Ich vznik sa datuje ku klasickej Levenberg-Marquardt metóde na riešenie sústavy nelineárnych rovníc[3] $F(x) = 0$, kde $F : \mathbb{R}^n \longmapsto \mathbb{R}^m, m \geq n$. Úlohu

$$\min_{x \in \mathbb{R}^n} \|F(x)\|_2^2 \tag{1.4}$$

rieši iteračne a v jednotlivých iteráciach volí krok

$$d_k = -(J(x_k)J(x_k)^T + \lambda_k I)^{-1} J(x_k)F(x_k) \tag{1.5}$$

kde $J(x_k)$ je Jacobiho matica funkcie $F(x)$ a paramter $\lambda_k \geq 0$ sa v jednotlivých iteráciach vhodne volí tak, aby $\|d_k\|$ bolo dostatočne malé. Táto úloha je ekvivalentná úlohe

$$\min_{d \in \mathbb{R}^n} \|F(x_k) + J(x_k)^T d\|_2^2 \tag{1.6}$$

$$\|d\|_2 \leq \Delta_k \tag{1.7}$$

v ktorej namiesto λ_k volíme priamo požadovanú dĺžku vektora d_k pomocou paramtera Δ_k.

1.2 Metódy s ohraničeným krokom

Metódy s ohraničeným krokom sú založené na lokálnej aproximácii účelovej funkcie $f(x)$ vhodne zvolenou funkciou $\psi(x)$, kde $\psi : \mathbb{R}^n \to \mathbb{R}$, v nejakom okolí Δ_k aktuálneho bodu x_k. Najčastejšie sa za $\psi(x)$ volí kvadratická funkcia. Za najlepšiu aproximáciu $f(x)$ sa v tomto prípade javí použiť Taylorov rozvoj 2.rádu

$$f(x) \approx f_k + g_k^T p + \frac{1}{2} p^T G_k p \tag{1.8}$$

kde $f_k = f(x_k)$, $g_k = \nabla f(x_k)$, $G_k = \nabla^2 f(x_k)$ a $p = x - x_k$ je posun.

V prípade veľkého počtu premenných funkcie $f(x)$ je ale výpočet Hessovej matice G_k príliš drahý, preto sa častejšie používa aproximovaná Hessova matica $B_k = B_k^T \in \mathbb{R}^{n \times n}$ získaná pomocou niektorej z kvázinewtonových formúl. Funkcia $\psi_k(p)$ potom vyzerá následovne

$$\psi_k(p) = \frac{1}{2} p^T B_k p + g_k^T p \tag{1.9}$$

V každej iterácii hľadáme posun p_k z okolia aktuálneho bodu x_k, v ktorom sa predpokladá, že $\psi(p)$ dobre aproximuje účelovú funkciu $f(x)$. Na získanie posunu teda musíme riešiť podúlohu

$$\min_{p \in \mathbb{R}^n} \psi_k(p) = \frac{1}{2} p^T B_k p + g_k^T p \tag{1.10}$$

$$\|p\|_2 \leq \Delta_k \tag{1.11}$$

kde $\Delta_k > 0$ je polomer okolia. Tento polomer v jednotlivých iteráciach meníme podľa tzv. stupňa zhody r_k

$$r_k = \frac{Ared_k}{Pred_k} = \frac{f(x_k) - f(x_k + p_k)}{\psi_k(0) - \psi_k(p_k)} \tag{1.12}$$

ktorý meria kvalitu aproximácie.

Kedže p_k volíme ako riešenie podúlohy (1.10)-(1.11), výraz $Pred_k = \psi_k(0) - \psi_k(p_k)$ je vždy nezáporný. Výraz $Ared = f(x_k) - f(x_k + p_k)$ predstavuje pokles účelovej funkcie vzhľadom na posun p_k a jeho kladnosť závisí od kvality aproximácie a teda môže byť aj záporný.

Stupeň zhody r_k rozhoduje o prijatí, resp. zamietnutí posunu $p_k = p(\Delta_k)$ a o veľkosti oblasti Δ_{k+1}. Zvolíme si preto parameter $\tau_1 \geq 0$ rozhodujúcom o bode x_{k+1}

$$x_{k+1} = \begin{cases} x_k + p_k & \text{ak } r_k \geq \tau_1 \\ x_k & \text{inak} \end{cases} \tag{1.13}$$

a parametre $\tau_2 > 0, 0 < \alpha_1 < 1 < \alpha_2$ určujúce veľkosť oblasti Δ_{k+1}

$$\Delta_{k+1} = \begin{cases} \alpha_1 \Delta_k & \text{ak } r_k < \tau_1 \\ \alpha_2 \Delta_k & \text{ak } r_k \geq \tau_2 \\ \Delta_k & \text{inak} \end{cases} \tag{1.14}$$

ktorých hodnoty si na začiatku algoritmu pevne zvolíme. Od hodnôt týchto paramtetrov závisí kvalita konvergencie celkovej metódy[5].
V ideálnom prípade $r_k \sim 1$, čo znamená že funkcia $\psi_k(x)$ dobre aproximuje účelovú funkciu $f(x)$, preto parameter τ_2 je vhodné voliť blízko 1. Za τ_1 sa javí

najideálnejšie zvoliť $\tau_1 = 0$, t.j. v prípade akéhokoľvek poklesu účelovej funkcie prijať posun. Dôsledkom takejto voľby je však slabšia konvergenčná vlastnosť metódy[2]

$$\liminf_{k\to\infty} \|g_k\|_2 = 0 \tag{1.15}$$

pričom v prípade voľby $\tau_1 > 0$ platí

$$\lim_{k\to\infty} \|g_k\|_2 = 0 \tag{1.16}$$

Zhrnutím predchádzajúcich úvah dostávame základný algoritmus metód s ohraničeným krokom.

Algorithm 1 Klasická metóda s ohraničeným krokom

Vstup: $x_0 \in \mathbb{R}^n, \Delta_0 > 0, \epsilon \geq 0, B_0 = B_0^T \in \mathbb{R}^{n\times n}$
$0 \leq \tau_1 \leq \tau_2, 0 < \tau_2, 0 < \alpha_1 < 1 < \alpha_2, k = 0$
1) Výpočet gradientu g_k, ak $\|g_k\| < \epsilon$ STOP
2) Získanie posunu $\hat{p}_k$ riešením (1.10)-(1.11)
3) Výpočet $r_k = \frac{Ared_k}{Pred_k}$
4) Zmena x_{k+1} podľa (1.13)
5) Zmena Δ_{k+1} podľa (1.14)
6) $k \longleftarrow k+1$ GOTO 1)

1.3 Kvázinewtonovské formuly

Ako bolo spomenuté, v každej iterácii sa budeme snažiť hľadať posun p_k ako minimum kvadratickej funkcie

$$\psi_k(p) = \frac{1}{2} p^T B_k p + g_k^T p$$

kde matica $B_k = B_k^T$ je aproximovaná Hessova matica účelovej funkcie $f(x)$, ktorú zlepšujeme pomocou niektorej z tzv. *kvázinewtonovských formúl*.

Kvázinewtonovské vzorce slúžia na aproximáciu Hessovej matice pomocou prvých parciálnych derivácii funkcie $f(x)$ získaných v jednotlivých iteráciách. Nová matica B_{k+1} sa získa pomocou tzv. *aditívnej korekcie* ako $B_{k+1} = B_k + \Delta B_k$.
O matici B_k pritom predpokladáme splnenie istých podmienok[4]

1. matica B_k je symetrická, t.j. $B_k = B_k^T$, a kladne definitná
2. matica B_k musí spĺňať tzv. *kvázinewtonovskú podmienku*

   $$g_{k+1} - g_k = B_k(x_{k+1} - x_k) \tag{1.17}$$

   kde g_k, resp. g_{k+1} je gradient účelovej funkcie $f(x)$ v príslušných bodoch x_k, resp. x_{k+1}
3. korekčná matica ΔB_k má malú hodnosť

Do triedy kvázinewtonovských formúl patrí tzv. *Broydenova trieda*

$$B_{k+1} = B_k - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} + \frac{y_k y_k^T}{p_k^T y_k} + \theta(p_k^T B_k p_k) w_k w_k^T \tag{1.18}$$

kde $p_k = x_{k+1} - x_k, y_k = g_{k+1} - g_k, w_k = \frac{y_k}{p_k^T y_k} - \frac{B_k p_k}{p_k^T B_k p_k}$ a $\theta \in \mathbb{R}^n$ je tzv. *Broydenov parameter*.
Voľbou parametra θ môžeme získať rôzne kvázinewtonovské formuly, napr.

1. DFP: $\theta = 0$
2. BFGS: $\theta = 1$
3. SR1: $\theta = \frac{p_k^T y_k}{p_k^T y_k - p_k^T B_k p_k}$

2 Metódy riešenia podúloh

Výpočet posunu p_k vyžaduje v každej iterácii riešiť úlohu na viazaný extrém

$$\min_{p\in\mathbb{R}^n} \psi(p) = \frac{1}{2}p^T Bp + g^T p \tag{2.1}$$

$$\|p\|_2 \leq \Delta \tag{2.2}$$

kde $B = B^T \in \mathbb{R}^{n\times n}, p, g \in \mathbb{R}^n, \Delta > 0$. Efektívnosť metód s ohraničeným krokom teda závisí od schopnosti riešiť danú podúlohu. Existujú dva prístupy k získaniu posunu p

1. *Výpočet presného riešenia* - posun $\hat{p}$ nám zaručí maximálny možný pokles, avšak je výpočtovo náročný, čo v prípade veľkého počtu premenných môže byť neúnosné
2. *Získanie približného riešenia* - posun $\hat{p}$ nám zaručí postačujúci pokles, ktorý je síce menej efektívny ako v prípade presného riešenia, avšak aj menej náročný na výpočet

2.1 Presné riešenie

V prípade že v úlohe (2.1)-(2.2) je počet premenných n dostatočne málo, oplatí sa nám hľadať posun p čo najpresnejšie. Vlastnosti opimálneho riešenia popisuje následovná lema[7].

Lema 2.1. *Vektor $\hat{p} \in \mathbb{R}^n$ je globálnym riešením úlohy (2.1)-(2.2) vtedy a len vtedy, ak $\|\hat{p}\|_2 \leq \Delta$ a $\exists\lambda \geq 0$ tak, že platí*

$$(B + \lambda I)\hat{p} = -g \tag{2.3}$$

$$\lambda(\Delta - \|\hat{p}\|_2) = 0 \tag{2.4}$$

$$(B + \lambda I) \succeq 0 \tag{2.5}$$

Dôkaz. Vzťahy (2.3)-(2.4) vyplývajú z nutných podmienok prvého rádu pre riešenie úlohy (2.1)-(2.2). Ostáva nám teda dokázať, že matica $(B + \lambda I)$ je kladne semi-definitná. Predpokladajme, že $\hat{p} \neq 0$. Kedže $\hat{p}$ je riešením úlohy (2.1)-(2.2), musí byť riešením aj úlohy

$$\min_{w\in\mathbb{R}^n} \psi(w)$$
$$\|w\| = \|\hat{p}\|$$

Je jasné, že platí $\psi(w) \geq \psi(\hat{p})$. Využitím (2.3) teda dostávame

$$-\hat{p}^T(B + \lambda I)w + \frac{1}{2}w^T Bw \geq -\hat{p}^T(B + \lambda I)\hat{p} + \frac{1}{2}\hat{p}^T B\hat{p} \tag{2.6}$$

z čoho dostávame

$$\frac{1}{2}(w - \hat{p})^T(B + \lambda I)(w - \hat{p}) \geq \frac{\lambda}{2}(w^T w - \hat{p}^T\hat{p}) = 0 \tag{2.7}$$

pre všetky w pre ktoré platí $\|w\|_2 = \|\hat{p}\|_2$. Kedže $\hat{p} \neq 0$ tak z (2.7) vyplýva že matica $(B + \lambda I)$ je kladne semidefinitná.
V prípade $\hat{p} = 0$ z (2.5) vyplýva $g = 0$, t.j. $\hat{p} = 0$ je riešením úlohy $\min\{\frac{1}{2}w^T Bw : \|w\|_2 \leq \Delta\}$ a teda matica B je kladne semidefinitná. Kedže $\lambda \geq 0$, tak aj $(B + \lambda I)$ musí byť kladne semidefinitná matica. □

Lema 2.1 nám teda bez explicitnej znalosti optimálneho riešenia úlohy (2.1)-(2.2) hovorí o jeho vlastnostiach. Z podmienky (2.4) vyplýva, že aspoň jeden z výrazov λ resp. $(\Delta - \|\hat{p}\|_2)$ musí byť nulový. Teda ak vektor $\hat{p}$ leží vnútri oblasti, musí platiť $\lambda = 0$ z čoho dostávame $B\hat{p} = -g$ a matica B je kladne semidefinitná. Naopak v prípade $\|\hat{p}\|_2 = \Delta$ platí $\lambda \geq 0$ a z (2.3) dostávame

$$\lambda\hat{p} = -B\hat{p} - g = -\nabla\psi(\hat{p})$$

Z výrazu (2.3) možno definovať funkciu

$$p(\lambda) = -(B + \lambda I)^{-1}g \tag{2.8}$$

pre $\lambda \geq 0$ dostatočne veľké aby matica $(B + \lambda I)$ bola kladne semidefinitná. Úlohu (2.1)-(2.2) možno teda previesť na ekvivalentnú úlohu

$$\|p(\lambda)\|_2 = \Delta \tag{2.9}$$

Funkcia $\|p(\lambda)\|_2$ je spojitá a klesajúca, a v prípade $g \neq 0$ platí

$$\lim_{\lambda \to +\infty} \|p(\lambda)\| = 0 \tag{2.10}$$

Z toho vyplýva, že v prípade $g \neq 0$ a $\Delta > 0$ musí mať rovnica (2.9) aspoň jedno riešenie.

Riešenie úlohy (2.1)-(2.2) môžeme previesť na hľadanie takého parametra $\lambda \geq 0$, pre ktorý platí rovnica (2.9). Teoreticky možno na $\|(B + \lambda I)^{-1}g\|_2 = \Delta$ aplikovať *Newtonovu metódu*. Vo všeobecnosti sa odporúča hľadať optimálny parameter $\hat{\lambda}$ riešením ekvivalentnej, avšak numericky stabilnejšej, rovnice

$$\phi(\lambda) = \frac{1}{\|(B + \lambda I)^{-1}g\|_2} - \frac{1}{\Delta} = 0 \tag{2.11}$$

Žiaľ ani riešením (2.11) nemusíme získať dostatočne presné riešenie. Problém môže totiž nastať v prípade, že optimálna hodnota parametra $\hat{\lambda}$ je rovná $\hat{\lambda} = -\lambda_1$, kde λ_1 je najmenšia vlastná hodnota matice B. V tomto prípade je matica $(B + \hat{\lambda} I)$ singulárna, a riešenie (2.11) sa stáva numericky nestabilné, ako to ilustruje následovný príklad[7]

$$\min_{p \in \mathbb{R}^n} \psi(p) = \frac{1}{2} p^T \begin{pmatrix} 1 & 0 \\ 0 & \eta \end{pmatrix} p + p^T \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$\|p\|_2 \leq \Delta$$

kde $\eta \leq 0$ volíme tak, aby platilo $\frac{1}{(1-\eta)^2} \leq \Delta^2$. Riešenie $\hat{p}$ vyzerá následovne $\hat{p}^T = -(\frac{1}{1-\eta}, \theta)$, kde $\frac{1}{(1-\eta)^2} + \theta^2 = \Delta^2$. Pre malú zmenu hodnoty $g_\epsilon^T = (1, \epsilon)$ dostávame riešenia $\hat{p}_\epsilon^T = -(\frac{1}{1+\lambda}, \frac{\epsilon}{\eta+\lambda})$ kde $\frac{1}{(1+\lambda)^2} + \frac{\epsilon^2}{(\eta+\lambda)^2} = \Delta^2$. Zvolením ľubovoľej hodnoty pre θ dostávame odchýlku ϵ čím zmena v $\frac{\|\hat{p} - \hat{p}_\epsilon\|_2}{\|\hat{p}\|} \gg 1$. V prípade $\eta < 0$ musí platiť $\|\hat{p}\|_2 = \Delta$ a s ohľadom na chyby v zaokrúhlovaní môžeme dostať nepresné riešenia.

V prípade, že najmenšia vlastná hodnota matice B je nulová, $\lambda_1 = 0$, riešenie úlohy (2.1)-(2.2) má tvar

$$\hat{p} = -B^{+}g \tag{2.12}$$

kde B^{+} predstavuje *pseudo-inverznú* maticu k matici B. Skutočným problémom je teda prípad $\lambda_1 < 0$. Riešenie $\hat{p}$ úlohy (2.1)-(2.2) potom musí vyzerať následovne

$$\hat{p} = -(B + \hat{\lambda}I)^{+}g + \theta q \tag{2.13}$$

kde $q \in S_1 = \{q \in \mathbb{R}^n : Bq = \lambda_1 q\}, \|q\|_2 = 1$. Teda ak $\lambda_1 < 0$, potom v prípade $\lambda > -\lambda_1$ sa aplikovaním Newtonovej metódy môže stať matica $(B+\lambda^{+}I)$ záporne definitná, takže je potrebné strážiť dolnú hranicu hodnoty λ. Druhou možnosťou je použitie Newtonovej metódy na rovnicu[3]

$$\tilde{\phi}(\lambda) = \phi(\frac{1}{\lambda}) = 0 \tag{2.14}$$

2.2 Približné riešenie

V prípade úloh s veľkým počtom premenných sa stáva hľadanie presného riešenia úlohy (3.1)-(3.2) príliš náročné vzhľadom na nutnosť počítania rozkladu matice $(B-\lambda I)$. Preto sa vyvinuli rôzne metódy na hľadanie približného riešenia, ktoré sú menej výpočtovo náročné oproti presnému riešeniu , avšak vykazujú dostatočný pokles funkcie $\psi(p)$. Medzi tieto patria napríklad *dogleg*, *minimalizácia v 2-rozmernom podpriestore*, *skrátená metóda konjugovaných gradientov* a iné.

Dogleg Metóda dogleg sa snaží aproximovať trajektóriu $\hat{p}(\Delta)$ pomocou dvoch smerov, *Cauchyho smeru najväčšieho spádu* $p_1 = -\frac{g^T g}{g^T B g} g$, kde výraz $\frac{g^T g}{g^T B g}$ predstavuje optimálny krok pre kvadratickú funkciu $\psi(p)$, a *Newtonovho smeru* $p_2 = -B^{-1}g$.

V prípade, že matica B je kladne definitná, dá sa globálne minimum funkcie $\psi(p) = \frac{1}{2}p^T Bp + g^T p$ explicitne vyjadriť (v prípade neohraničeného kroku) vzorcom

$$\hat{p}_2 = -B^{-1}g \tag{2.15}$$

Ak platí $\|\hat{p}_2\|_2 \leq \Delta$ potom je aj riešením úlohy (2.1)-(2.2). V opačnom prípade optimálne riešenie musí ležať na hranici, t.j. $\|\hat{p}\|_2 = \Delta$.

Trajektóriu $\hat{p}(\Delta)$ potom odhadneme ako

$$\bar{p}(\tau) = \begin{cases} \tau p_1 & 0 \leq \tau \leq 1 \\ p_1 + (\tau - 1)(p_2 - p_1) & 1 \leq \tau \leq 2 \end{cases} \tag{2.16}$$

a teda hľadáme parameter $\tau \in \langle 0, 2\rangle$ tak, aby platilo

$$\|p_1 + (\tau - 1)(p_2 - p_1)\|_2 = \Delta \tag{2.17}$$

Skrátená metóda konjugovaných gradientov Nevýhodou *dogleg* metódy je nutnosť výpočtu *Newtonovho smeru* $p_2 = -B^{-1}g$, kde potrebujeme počítať inverznú Hessovu maticu, resp. sústavu rovíc $Bp_2 = -g$, čo zväčša vyžaduje urobiť *Choleského rozklad* matice B. Tejto potreby nás zbavuje práve *metóda združených smerov*.

Najprv uvedieme definíciu združených smerov a vetu o konvergencii metódy v prípade kvadratickej účelovej funkcie, ktorej dôkaz možno nájsť v [4]

Definícia 2.1. *Nech $B = B^T$ je kladne definitná matica typu $n \times n$. Smery $(s_0, s_1, \ldots, s_{n-1})$, t.j. nenulové vektory $s_i \in \mathbb{R}^n$, sa nazývajú B-združené (resp. B-konjugované, B-ortogonálne), ak pre každú dvojicu s_i,s_j $(i \neq j)$ platí*

$$s_i^T B s_j = 0 \tag{2.18}$$

Veta 2.1. *Nech $B = B^T$ je kladne definitná matica typu $n \times n$, smery $(s_0, s_1, \ldots, s_{n-1})$ sú B-združené, $\hat{p} \in \mathbb{R}^n$ je bodom minima kvadratickej funkcie*

$$\psi(p) = \frac{1}{2} p^T B p + g^T p \tag{2.19}$$

a $p_0 \in \mathbb{R}^n$ je ľubovolný štartovací bod iteračného procesu $p_{k+1} = p_k + \lambda_k s_k$, kde λ_k je optimálny krok.
Potom $p_n = \hat{p}$, t.j. iteračný proces určí optimálne riešenie v prípade kvadratickej účelovej funkcie za n krokov.

Na generovanie združených smerov využijeme vzorce Fletchera-Reevesa, ktoré definujú iteračný proces následovne

$$p_{k+1} = p_k + \lambda_k s_k \tag{2.20}$$

$$s_{k+1} = -g_k + \mu_k s_{k-1} \tag{2.21}$$

$$\mu_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \tag{2.22}$$

kde $\lambda_k = \frac{g_k^T g_k}{s_k^T B s_k}$ je optimálny krok, $g_k = \nabla \psi(p_k)$ a $s_0 = -g_0$.

Veta nám hovorí, že v prípade úlohy na voľný extrém s kvadratickou účelovou funkciou metóda združených smerov určí minimum za n krokov. My však riešime úlohu na viazaný extrém. Preto do samotného algoritmu pridáme ešte podmienku ktorá iteračný proces ukončí v prípade, ak $\|p_k\|_2 \geq \Delta$, a optimálne riešenie zvolíme tak, aby platilo $\|\hat{p}\|_2 \leq \Delta$.

Algorithm 2 Skrátená metóda konjugovaných gradientov

Vstup: $p_0 \in \mathbb{R}^n$, $\Delta > 0$, $\epsilon \geq 0$,matica B, vektor g, $k = 0$

1) Cauchyho iterácia: $g_0 = B p_0 + g, s_0 = -g_0, \lambda_0 = \frac{g_0^T g_0}{s_0^T B s_0}$
$p_1 = p_0 + \lambda_0 s_0$

2) Ak $\|g_k\|_2 \leq \epsilon$ STOP

3) Ak $s_k^T B s_k \leq 0$
Nájdi $\tau \geq 0$ aby $\hat{p} = p_k + \tau s_k$ vyhovovalo $\|\hat{p}\|_2 = \Delta$
STOP

4) Výpočet $\alpha_k = \frac{g_k^T g_k}{s_k^T B s_k}$
$p_{k+1} = p_k + \alpha_k s_k$

5) Ak $\|p_{k+1}\|_2 \geq \Delta$
Nájdi $\tau \geq 0$ aby $\hat{p} = p_k + \tau s_k$ vyhovovalo $\|\hat{p}\|_2 = \Delta$
STOP

8) Výpočet $g_{k+1} = B p_k + g$
$\mu_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$
$s_{k+1} = -g_{k+1} + \mu_k s_k$

9) $k \longleftarrow k + 1$, GOTO 2

3 Transformácia do podpriestoru

Ukázali sme doteraz, že metódy s ohraničeným krokom sa v každej iterácii snažia nájsť minimum účelovej funkcie $f(x_k)$ pomocou lokálnej aproximácie kvadratickou funkciou $\psi_k(p)$, pričom posun $\hat{p}_k$ musí byť z nejakého okolia Δ_k, t.j. riešením podúlohy

$$\min_{p\in\mathbb{R}^n} \psi_k(p) = \frac{1}{2}p^T B_k p + g_k^T p \tag{3.1}$$

$$\|p\|_2 \leq \Delta_k \tag{3.2}$$

O riešení $\hat{p}_k$ tejto podúlohy sme zistili, že musí spĺňať isté nutné podmienky optimality, konkrétne musí existovať parameter $\lambda_k \geq 0$ tak, aby platilo

$$(B_k + \lambda_k I)\hat{p}_k = -g_k \tag{3.3}$$

$$\lambda_k(\Delta_k - \|\hat{p}_k\|_2) = 0 \tag{3.4}$$

$$(B_k + \lambda_k I) \succeq 0 \tag{3.5}$$

Pomocou týchto podmienok sme potom navrhli metódu na hľadanie presného riešenia podúlohy (3.1)-(3.2).

Z podmienok však možno usúdiť ešte ďalšie vlastnosti optimálneho posunu $\hat{p}_k$. Ak vezmeme prvú iteráciu, kde matica $B_k = \sigma I$, tak podmienka (3.3) vyzerá následovne

$$(\sigma + \lambda_0)\hat{p}_0 = -g_0 \tag{3.6}$$

a teda vektor $\hat{p}_0$ je násobkom gradientu g_0 funkcie $f(x)$ v bode x_0. O vlastnostiach smerov $p_1, p_2, \ldots$ v prípade, že matica B_k je získaná pomocou niektorej z Broydenových formúl, hovorí následovná lema, ktorej dôkaz možno nájsť v[1]

Lema 3.1. *Nech $B_0 = \sigma I$, $\sigma > 0$ a B_k je k-ta matica získaná pomocou niektorej z Broydenových alebo PSB formúl. Nech $\hat{p}_k$ je riešením úlohy (3.1)-(3.2), $g_k = \nabla f(x_k)$, $\mathcal{G}_k = Span\{g_0, g_1, \ldots, g_k\}$ a iteračný proces je daný vzorcom $x_{k+1} = x_k + p_k$ so štartovacím bodom x_0. Potom pre všetky $k \geq 0$ platí $p_k \in \mathcal{G}_k$.*
Naviac pre ľubovolné $z \in \mathcal{G}_k, u \in \mathcal{G}_k^{\perp}$ platia vzťahy

$$B_k z \in \mathcal{G}_k, B_k u = \sigma u \tag{3.7}$$

Z lemy vyplýva, že štartujúc z počiatočného bodu x_0 s maticou $B_0 = \sigma I$, kde $\sigma > 0$, budú riešenia $\hat{p}_k$ podúlohy (3.1)-(3.2), kde matica B_k je upravovaná pomocou niektorej z Broydenovských formúl, vždy ležať v podpriestore generovanom vektormi $g_0, g_1, \ldots, g_k$. O využití tejto vlastnosti hovorí následovná lema.

Lema 3.2. *Nech S_k je r-rozmerný podpriestor priestoru $\mathbb{R}^n$, $1 \leq r \leq n$. Nech $Z_k \in \mathbb{R}^{n\times r}$ je ortogonálna matica generujúca podpriestor S_k, t.j.*

$$S_k = Span\{Z_k\}, Z_k^T Z_k = I_r \tag{3.8}$$

Predpokladajme, že $g_k \in S_k$ a $B_k = B_k^T$ je matica vyhovujúca vzťahom

$$B_k z \in S_k, \forall z \in S_k \tag{3.9}$$

$$B_k u = \sigma u, \forall u \in S_k^{\perp} \tag{3.10}$$

kde $\sigma > 0$ je skalár. Potom úloha (3.1)-(3.2) je ekvivalentná úlohe

$$\min_{\bar{p}\in R^r} \bar{\psi}_k(\bar{p}) \equiv \frac{1}{2}\bar{p}^T \bar{B}_k \bar{p} + \bar{g}_k^T \bar{p} \tag{3.11}$$

$$\|\bar{p}\|_2 \leq \Delta_k \tag{3.12}$$

kde $\bar{g}_k = Z_k^T g_k$ a $\bar{B}_k = Z_k^T B_k Z_k$. Teda ak $\hat{p}_k$ je riešením úlohy (3.1)-(3.2), potom $\hat{\bar{p}}_k = Z_k^T \hat{p}_k$ je riešením (3.11)-(3.12). Naopak, ak $\hat{\bar{p}}_k$ je riešením úlohy (3.11)-(3.12), potom $\hat{p}_k = Z_k \hat{\bar{p}}_k$ je riešením (3.1)-(3.2).

Dôkaz. Nech $U_k \in \mathbb{R}^{n\times(n-r)}$ je matica taká, že $[U_k, Z_k]$ je ortogonálna matica typu $n \times n$. Potom $\forall p \in \mathbb{R}^n$ existuje jediná dvojica $\bar{p} \in \mathbb{R}^r, u \in \mathbb{R}^{n-r}$ taká, že $p = Z_k\bar{p} + U_k u$, t.j.

$$\psi_k(p) = g_k^T p + \frac{1}{2}p^T B_k p = g_Z^T \bar{p} + g_U^T u + \frac{1}{2}\bar{p}^T Z_k^T B_k Z_k \bar{p} + \frac{1}{2}u^T U_k^T B_k U_k u + \bar{p}^T Z_k^T B_k U_k u \tag{3.13}$$

kde $g_Z = Z_k^T g_k, g_U = U_k^T g_k$. Zo vzťahov $g_k \in Span\{Z_k\}$ a (3.9)-(3.10) dostávame

$$B_k U_k = \sigma U_k, Z_k^T B_k U_k = \sigma Z_k^T U_k = 0, g_U = U_k^T g_k = 0 \tag{3.14}$$

a z (3.13)-(3.14) vyplýva

$$\psi_k(p) = (g_Z^T \bar{p} + \frac{1}{2}\bar{p}^T Z_k^T B_k Z_k \bar{p}) + \frac{1}{2}\sigma u^T u \tag{3.15}$$

Z ortonormality matíc Z_k a U_k platí $\|p\|_2^2 = \|\bar{p}\|_2^2 + \|u\|_2^2$. Dostávame teda, že úloha (3.1)-(3.2) je ekvivalentná úlohe

$$\min_{\bar{p}\in\mathbb{R}^r, u\in\mathbb{R}^{n-r}} (g_Z^T \bar{p} + \frac{1}{2}\bar{p}^T Z_k^T B_k Z_k \bar{p}) + \frac{1}{2}\sigma u^T u \tag{3.16}$$

$$\|\bar{p}\|_2^2 + \|u\|_2^2 \leq \Delta_k^2 \tag{3.17}$$

pričom $p = Z_k\bar{p} + U_k u$. Kedže $\sigma > 0$, potom úloha (3.16)-(3.17) je ekvivalentná úlohe (3.11)-(3.12) pre $u = 0$. Teda úloha (3.1)-(3.2) je ekvivalentná úlohe (3.11)-(3.12), kde $p = Z_k\bar{p}$ a zo vzťahu $Z_k^T Z_k = I_r$ dostávame $\bar{p} = Z_k^T p$. □

Výsledky zhrnieme v následujúcej vete[1]

Veta 3.1. *Nech Z_k je ortogonálna matica generujúca podpriestor $\mathcal{G}_k = Span\{g_0, g_1, \ldots, g_k\}$. Nech matica $B_0 = \sigma I, \sigma > 0$ a B_k je v poradí k-ta matica získaná pomocou niektorej z Broydenovej alebo PSB triedy formúl. Potom pre riešenie $\hat{p}_k$ úlohy (3.1)-(3.2) platí $\hat{p}_k \in \mathcal{G}_k$ a $\hat{p}_k = Z_k\hat{z}_k$, kde $\hat{z}_k$ je riešením úlohy (3.11)-(3.12).*

Ukázali sme teda, že pre riešenie úlohy (3.1)-(3.2) platí $\hat{p}_k \in \mathcal{G}_k$ a teda možno hľadať posun riešením úlohy v podpriestore generovanom vektormi $g_0, g_1, \ldots, g_k$, t.j. ako riešenie úlohy (3.11)-(3.12). Táto transformácia nám môže ušetriť v prípade $k \ll n$ relatívne veľa času, naviac aproximovanú Hessovu maticu možno získavať priamo v podpriestore $\mathcal{G}_k$, o čom hovorí následovná lema[1].

Lema 3.3. *Nech $Z \in \mathbb{R}^{n\times r}$ je ortogonálna matica taká, že $Z^T Z = I_r$. Nech $p_k \in Span\{Z\}$ a matica B_{k+1} je získaná z matice B_k pomocou jednej z Broydenovej alebo PSB formúl využitím vektorov p_k a y_k. Označme $\bar{B}_{k+1} = Z^T B_{k+1} Z, \bar{B}_k = Z^T B_k Z, \bar{p}_k = Z^T p_k, \bar{y}_k = Z^T y_k$.*
Potom matica $\bar{B}_{k+1}$ je získaná z matice $\bar{B}_k$ využitím vektorov $\bar{p}_k$ a $\bar{y}_k$.

Dôkaz. Z predpokladov $p_k \in Span\{Z\}$ a $Z^T Z = I_r$ platí $p_k = ZZ^T p_k$ a

$$p_k^T y_k = (Z^T p_k)^T Z^T y_k = \bar{p}_k^T \bar{y}_k \tag{3.18}$$

$$p_k^T B_k p_k = (Z^T p_k)^T Z^T B_k Z (Z^T p_k) = \bar{p}_k^T \bar{B}_k \bar{p}_k \tag{3.19}$$

$$Z^T B_k p_k = Z^T B_k Z (Z^T p_k) = \bar{B}_k \bar{p}_k \tag{3.20}$$

Využitím vzorcov (3.18)-(3.20) a vynásobením Broydenovej formule

$$B_{k+1} = B_k - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} + \frac{y_k y_k^T}{p_k^T y_k} + \theta_k (p_k^T B_k p_k) w_k w_k^T \tag{3.21}$$

maticou Z^T zľava a Z zprava sa platnosť lemy ľahko overí (pre Broydenovu triedu formúl). □

Ostáva nám navrhnúť algoritmus ktorým by sme získavali transformačnú maticu Z_{r_k}. Predpokladajme, že v k-tej iterácii poznáme maticu Z_{r_k} tvoriacu ortogonálnu bázu r_k-rozmerného priestoru $\mathcal{G}_k = Span\{g_1, \ldots, g_k\}$. Nech $\hat{\bar{p}}_k$ je riešenie transformovanej podúlohy (3.11)-(3.12), potom pre riešenie podúlohy (3.1)-(3.2) $\hat{p}_k$ máme $\hat{p}_k = Z_k \hat{\bar{p}}_k$. Potrebujeme získať maticu Z_{k+1} a hodnoty $\bar{g}_{k+1} = Z_{k+1}^T g_{k+1}$ a $\bar{B}_{k+1} = Z_{k+1}^T B_{k+1} Z_{k+1}$.

Kvôli numerickej stabilite použijeme rozklad[1]

$$g_{k+1} = Z_k u_k + \rho_{k+1} z_{k+1} \tag{3.22}$$

kde $u_k = Z_k^T g_{k+1}, z_{k+1} \perp Span\{Z_k\}, \|z_{k+1}\|_2 = 1, \rho_{k+1} = \|(I - Z_k Z_k^T) g_{k+1}\|_2$. Hodnota ρ_{k+1} rohoduje o tvare matice Z_{k+1}. Za týmto účelom si pevne zvolíme hodnotu $\nu_2 \geq 0$. Ak $\rho_{k+1} \leq \nu_2 \|g_{k+1}\|_2$ potom použijeme "starú"maticu $Z_{k+1} = Z_k$ a položíme

$$\bar{g}_{k+1} = Z_{k+1}^T g_{k+1} = u_k \tag{3.23}$$

$$\tilde{s}_k = Z_{k+1}^T p_k = \bar{p}_k \tag{3.24}$$

$$\tilde{y}_k = Z_{k+1}^T y_k = u_k - \bar{g}_k \tag{3.25}$$

$$\tilde{B}_k = Z_{k+1}^T B_k Z_{k+1} = \bar{B}_k \tag{3.26}$$

V opačnom prípade zvolíme $Z_{k+1} = [Z_k, z_{k+1}], r_k = r_k + 1$ a využitím vzťahov

$$z_{k+1}^T g_{k+1} = \rho_{k+1}, z_{k+1}^T s_k = 0, z_{k+1}^T g_k = 0 \tag{3.27}$$

dostávame

$$\bar{g}_{k+1} = Z_{k+1}^T g_{k+1} = \begin{bmatrix} u_k \\ \rho_{k+1} \end{bmatrix} \tag{3.28}$$

$$\tilde{p}_k = Z_{k+1}^T p_k = \begin{bmatrix} \bar{p}_k \\ 0 \end{bmatrix} \tag{3.29}$$

$$\tilde{y}_k = Z_{k+1}^T y_k = \begin{bmatrix} u_k - \bar{g}_k \\ \rho_{k+1} \end{bmatrix} \tag{3.30}$$

$$\tilde{B}_k = Z_{k+1}^T B_k Z_{k+1} = \begin{bmatrix} \bar{B}_k & 0 \\ 0 & \sigma_k \end{bmatrix} \tag{3.31}$$

S využitím lemy 3.3. môžeme novú aproximáciu Hessovej matice získať pomocou matice $\tilde{B}_k$ a vektorov $\tilde{s}_k, \tilde{y}_k$.

Ako sme už spomínali, voľbou hodnoty $\nu \geq 0$ sa snažíme obísť numerické nástrahy spôsobené zaokrúhlovaním. Rovnaký význam plní aj parameter σ_k vo vzorci (3.31). Za hodnotu ν sa odporúča zvoliť $\nu = 10^{-3}$, zatiaľ čo v prípade hodnoty σ_k máme hneď niekoľko možností[1]

$$\sigma_k^{r0} = 1, \sigma_k^{r1} = \frac{y_0^T y_0}{p_0^T y_0}, \sigma_k^{r2} = \frac{p_k^T y_k}{\|p_k\|_2^2},$$
$$\sigma_k^{r3} = \min_{1 \leq i \leq k} \{\frac{p_i^T y_i}{\|p_i\|_2^2}\}, \sigma_k^{r4} = \frac{y_k^T y_k}{p_k^T y_k},$$
$$\sigma_k^{r5} = \frac{p_0^T y_0}{p_0^T p_0}, \sigma_k^{r6} = \min_{1 \leq i \leq k} \{\frac{y_i^T y_i}{p_i^T y_i}\}$$

Efektívnosť riešenia transformovanej úlohy spočíva hlavne v prípade, ak $r_k \ll n$, t.j. rozmer podúlohy (3.11)-(3.12) je podstatne menší ako rozmer pôvodnej podúlohy (3.1)-(3.2). Preto je vhodné nejakým spôsobom zabrániť zbytočnému narastaniu rozmeru r_k. Toto môžeme zabezpečiť následovným spôsobom[1]. Predpokladajme, že v predchádzajúcej iterácii sme rozšírili maticu $Z_k = [Z_{k-1}, z_k]$ a získali nový posun $\hat{\bar{p}}_k$ riešením (3.11)-(3.12). Zvoľme si parametre μ_1, μ_2 tak, aby platilo $0 < \mu_1 < 0.2 < \mu_2 < 1$. Potom v prípade, že $| \hat{\bar{p}}_k^{r_k} | \leq \mu_1 \|\hat{\bar{p}}_k\|_2$, kde $\hat{\bar{p}}_k^{r_k}$ predstavuje r_k-ty prvok vektora $\hat{\bar{p}}_k$, a $\rho_k \leq \mu_2 \|g_k\|_2$ možno usúdiť, že rozšírenie matice Z_{k-1} o vektor z_k nebolo veľmi významné a v ďalšej iterácii uvažovať maticu $Z_{k+1} = [Z_{k-1}]$.
Naše úvahy môžeme zhrnúť do následovného algoritmu

Algorithm 3 Transformovaná metóda s ohraničeným krokom

Vstup: $x_0 \in \mathbb{R}^n, \Delta_0 > 0, 0 \leq \Delta_{min} \leq 10^{-6}, \epsilon \geq \epsilon_M \geq 0$
$0 \leq \tau_1 \leq \tau_2, 0 < \tau_2, 0 < \alpha_1 < 1 < \alpha_2, k = 0$
$0 \leq \nu < 1, \sigma > 0, 0 < \mu_1 < 0.2 < \mu_2 < 1, \bar{B}_0 = \sigma$
nastavenie $k = 0, r = 1$ 1) Výpočet $f(x_0), g_0 = \nabla f(x_0), \bar{g}_0 = \|g_0\|_2, Z_0 = \left[\frac{g_0}{\|g_0\|_2}\right]$
2) Získanie $\hat{\bar{p}}_k$ riešením (3.11)-(3.12).
Ak $| \bar{\psi}(\hat{\bar{p}}_k) | \leq \epsilon_M$ tak STOP
3) Výpočet $\hat{p}_k = Z_k \hat{\bar{p}}_k, \gamma_k = \frac{f(x_k) - f(x_k + \hat{p}_k}{-\bar{\psi}(\hat{\bar{p}}_k}$
4) Ak $\gamma_k \leq \tau_1$ a $\Delta_k \leq \Delta_{min}$ tak STOP(zlyhanie algoritmu)
5) $x_{k+1} = x_k + \hat{p}_k, \Delta_{k+1} = \begin{cases} \alpha_1 \Delta_k & \text{ak } r_k < \tau_1 \\ \alpha_2 \Delta_k & \text{ak } r_k \geq \tau_2 \\ \Delta_k & \text{inak} \end{cases}$
6) Výpočet $g_{k+1} = \nabla f(x_{k+1})$.
Ak $\|g_{k+1}\|_2 \leq \epsilon$ tak STOP
7) Ak $| \hat{\bar{p}}_k^{r_k} | \leq \mu_1 \|\hat{\bar{p}}_k\|_2$, g_k bolo prijaté v predošlej iterácii a $\rho_k \leq \mu_2 \|g_k\|_2$ zvoľ $Z_k = Z_{k-1}$
8) Získaj (3.22)
9) Ak $\rho_{k+1} > \nu \|g_{k+1}\|_2$ priraď $Z_{k+1} = [Z_k, z_{k+1}], r = r + 1$
inak $Z_{k+1} = Z_k$
10) Vypočítaj $\bar{B}_{k+1}$ pomocou $\tilde{B}_k, \tilde{s}_k, \tilde{y}_k$
$k = k + 1$, GOTO 2

4 Numerické experimenty

Na porovnanie efektívnosti jednotlivých algoritmov sme vytvorili program v jazyku C. Porovnali sme klasickú metódu s ohraničeným krokom s modifikovanou metódou, v ktorej sme využili vlastnosti podpriestoru. Maticu B_k sme upravovali v oboch prípadoch pomocou kvázinewtonovskej formule $BFGS$

$$\text{(BFGS) } B_{k+1} = B_k - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} + \frac{y_k y_k^T}{p_k^T y_k}$$

Na testovanie sme využili prostredie *CUTEr* na príkladoch so stredne veľkým počtom premenných (okolo 1000).

4.1 CUTEr

CUTEr(Constrainded and Unconstrained Testing Environment, revisited) je testovacie prostredie určené na porovnávanie efektívnosti rôznych numerických algoritmov. Ide o vylepšenú verziu pôvodného testovacie prostredia *CUTE*[8] a možno ho získať na adrese *http://hsl.rl.ac.uk/cuter-www/*.

Súčasťou *CUTEr*-u sú rôzne nástroje slúžiace pre meranie efektívnosti jednotlivých optimalizačných balíkov. Je napísaný v jazyku *fortran*, avšak ponúka rozhranie pre vytvorenie nových rutín aj pre jazyky *C/C++* alebo *Matlab*. Jeho súčasťou sú aj predvytvorené rozhrania pre existujúce optimalizačné nástroje ako *loqo, LANCELOT* a iných.

Základný balík samotných príkladov sa dá získať na samotnej domovskej stránke *CUTEr*-u. Každý príklad je 'zabalený' do samostatného súboru s príponou *.SIF*, čo je skratka pre *Standard Imput Format*. Ide o štandardný formát určený pre ukladanie úloh nelineárneho programovania, tak ako formát *MPS* je štandardom pre lineárne úlohy.

Na dekódovanie informácie uloženej v týchto súboroch je potrebné doinštalovať k samotnému *CUTEr*-u balík *SifDec* ktorý je vyvíjaný ako samostatný nástroj. Postup inštalácie jednotlivých programov je popísaný v príslušných dokumentáciách [9], resp.[10].

4.2 Experiment

Experimenty sme previedli na 20 príkladoch s rozmermi medzi 999 a 2000 premenných. Úlohy sme riešili s presnosťou *double*. Klasická metóda s ohraničeným krokom bola realizovaná rovnako ako popisuje *Algoritmus 1* na strane 6 s tým rozdielom, že po získaní posunu $\hat{p_k}$ v kroku 2) sme pridali test $\|p_k\|_2 < \epsilon$ teda samotný algoritmus vyzerá následovne v našom prípade vyzerá následovne

Algorithm 4 Klasická metóda s ohraničeným krokom

Vstup: $x_0 \in \mathbb{R}^n, \Delta_0 > 0, \epsilon \geq 0, B_0 = B_0^T \in \mathbb{R}^{n \times n}$
$0 \leq \tau_1 \leq \tau_2, 0 < \tau_2, 0 < \alpha_1 < 1 < \alpha_2, k = 0$
1) Výpočet gradientu g_k, ak $\|g_k\| < \epsilon$ STOP
2) Získanie posunu $\hat{p}_k$ riešením (1.10)-(1.11)
Ak $\|p_k\|_2 < \epsilon$ tak STOP
3) Výpočet $r_k = \frac{Ared_k}{Pred_k}$
4) Zmena x_{k+1} podľa (1.13)
5) Zmena Δ_{k+1} podľa (1.14)
6) $k \longleftarrow k + 1$ GOTO 1)

Vzhľadom na rozmer úlohy sme ako metódu na získanie posunu p_k v oboch metódach zvolili skrátenú metódu konjugovaných gradientov. Práve kvôli tejto voľbe riešenia podúlohy bolo potrebné pridať spomínaný test v kroku 2). Totiž skrátená metóda konjugovaných gradientov nám dáva iba približné riešenie podúlohy a v prípade že x_k je blízko optimálneho bodu $\hat{x}$ dostávame príliš slabé zlepšenia, čo viedlo k stavom, že aj po ďalších 1000 iteráciách bolo zlepšenie skoro nulové. Pridaním testu na veľkosť p_k sa dá týmto stavom do značnej miery zabrániť.
Modifikovaná metóda s využitím transformácie do podpriestoru bola realizovaná tak ako to popisuje Algoritmus 3 s vynechaním testu 4) a pozmeneným testom v kroku 2) tak ako v prípade klasickej metódy bez použitia transformácie.

Pri testovaní oboch metód sme nastavili maximálny počet iterácií na 2000. Úlohu sme pokladali za úspešne vyriešenú ak $\|g_k\| < \epsilon = 10^{-3}$. Ďalšie hodnoty volené v jednotlivých algoritmoch vyzerali následovne

$$\Delta_0 = 1, \tau_1 = 0.0, \tau_2 = 0.95, \alpha_1 = 0.25, \alpha_2 = 3.5, \nu = 10^{-3}, \mu_1 = 0.1, \mu_2 = 0.3$$

BIGGSB1	1000	NONDIA	1000
CURLY10	1000	NNODQUAR	1000
CURLY20	1000	PENALTY1	1000
CURLY30	1000	POWER	1000
EDENSCH	2000	QUARTC	1000
FREUROTH	1000	SINQUAD	1000
LINVERSE	999	SPARSINE	1000
MSQRTALS	1024	VAREIGVL	999
MSQRTBLS	1024	WOODS	1000

Riešené úlohy

Na začiatok si ukážeme výstup v prípade klasickej metódy s ohraničeným krokom:

Úloha	$\#n$	$\#k$	$\#f$	$\|g\|_2$	$f(x)$	CPU(seconds)
BIGGSB1	1000	2000	2001	0.005862891	0.0038579	2558.58
CURLY10	1000	2000	2001	**4.375568**	-100307.6	483.42
CURLY20	1000	2000	2001	**9.990423**	-100309.5	685.05
CURLY30	1000	2000	2001	**20.41177**	-100310.4	839.16
EDENSCH	2000	116	117	0.0007017997	12003.28	207.09
FREUROTH	1000	2000	2001	**0.03897664**	121469.7	413.98
LINVERSE	999	223	224	0.0009646448	340	52.23
MSQRTALS	1024	2000	2001	**0.01442517**	0.003945274	494.51
MSQRTBLS	1024	2000	2001	**0.01978044**	0.004093463	496.57
NONCVXU2	1000	2000	2001	**0.2345664**	2321.191	482.75
NONDIA	1000	10	11	0.0006568108	1.852891e-10	2.60
NONDQUAR	1000	143	144	0.000989382	0.0001225794	420.20
PENALTY1	1000	82	83	0.0008346289	0.009994345	256.35
POWER	1000	644	645	0.0009619512	1.431674e-06	327.15
QUARTC	1000	280	281	0.0009984431	4.343821e-05	1310.49
SINQUAD	1000	25	26	0.0008996637	-294250.5	6.32
SPARSINE	1000	2000	2001	**0.166625**	0.001132461	526.06
SPMSRTLS	1000	209	210	0.0008177106	3.490781e-06	48.54
VAREIGVL	1000	145	146	0.0009012548	3.754552e-06	35.42
WOODS	1000	2000	2001	**0.005143096**	2.821286e-06	445.64

Z tabuľky vidno, že vo väčšine prípadov bola klasická metóda s ohraničeným krokom schopná nájsť minimum behom prvých 2000 iterácií. Všetky tieto príklady sme vybrali na základe článku *Sensitivity of Trust-Region Algorithms to Their Parameters*[5] od autorov *N.Gould, D.Orban, A.Sartenaer, Ph.L.Toint.* Všetky boli klasifikované ako dostatočne náročné úlohy.

Výstup modifikovanej metódy s ohraničeným krokom využitím transformácie:

Úloha	$\#n$	$\#k$	$\#f$	$\|g\|_2$	$f(x)$	CPU(seconds)
BIGGSB1	1000	2000	2001	0.005449575	0.01196696	644.84
CURLY10	1000	2000	2001	**5.394511**	-100306.6	590.14
CURLY20	1000	2000	2001	**7.270081**	-100308.4	812.64
CURLY30	1000	2000	2001	**7.219919**	-100309.1	949.58
EDENSCH	2000	137	138	0.003231751	12003.28	434.90
FREUROTH	1000	2000	2001	**0.1884448**	121356.5	563.01
LINVERSE	999	370	371	0.0008759012	340	200.38
MSQRTALS	1024	2000	2001	**0.01973053**	0.004371747	601.97
MSQRTBLS	1024	2000	2001	**0.02782957**	0.004627756	605.32
NONCVXU2	1000	2000	2001	**0.2683008**	2322.041	569.39
NONDIA	1000	7	8	0.0002090829	3.755451e-13	0.37
NONDQUAR	1000	102	103	0.0009568876	0.0002844302	48.05
PENALTY1	1000	16	17	**1.550027e+10**	6.086568e+12	1.19
POWER	1000	819	820	0.0006260868	5.130644e-06	379.03
QUARTC	1000	117	118	**0.1602396**	0.03805523	136.01
SINQUAD	1000	36	37	0.0002266703	-294250.5	2.70
SPARSINE	1000	2000	2001	**0.2315292**	0.001570675	692.71
SPMSRTLS	1000	274	275	0.0007488716	2.162306e-06	176.65
VAREIGVL	1000	244	245	0.000997246	1.754596e-06	166.75
WOODS	1000	110	111	**107.5507**	320.2307	10.68

Tu treba poznamenať, že transformácia úloh sa využívala iba do určitej hodnosti transformačnej matice Z_k. Konkrétne v tomto prípade sa riešila úloha v podpriestore iba pokiaľ tento bol menší ako 100. Teda v prípade, že rozmer podpriestoru dosiahol hodnotu 100, tak sa transformácia ďalej nevyužívala.

Z tabuľky vidno, že v tomto prípade riešenie úlohy *PENALTY1* úplne zlyhalo. To isté sa dá povedať aj o aplikovaní na úlohu *WOODS*. Tento jav pripisujeme numerickým chybám spôbených pripisujeme numerickým chybám spôsobených pri transformáciach.

V nasledujúcej tabuľke vidno rozdiely v časovej náročnosti medzi týmito metódami, pričom naľavo je klasická metóda a napravo metóda s využitím transformácie

Úloha	$\|g\|_2$	CPU(seconds)	CPU(seconds)	$\|g\|_2$	Úloha
BIGGSB1	0.005862891	2558.58	644.84	0.005449575	BIGGSB1
CURLY10	4.375568	483.42	590.14	5.394511	CURLY10
CURLY20	9.990423	685.05	812.64	7.270081	CURLY20
CURLY30	20.41177	839.16	949.58	7.219919	CURLY30
EDENSCH	0.0007017997	207.09	434.90	0.003231751	EDENSCH
FREUROTH	0.03897664	413.98	563.01	0.1884448	FREUROTH
LINVERSE	0.0009646448	52.23	200.38	0.0008759012	LINVERSE
MSQRTALS	0.01442517	494.51	601.97	0.01973053	MSQRTALS
MSQRTBLS	0.01978044	496.57	605.32	0.02782957	MSQRTBLS
NONCVXU2	0.2345664	482.75	569.39	0.2683008	NONCVXU2
NONDIA	0.0006568108	2.60	0.37	0.0002090829	NONDIA
NONDQUAR	0.000989382	420.20	48.05	0.0009568876	NONDQUAR
PENALTY1	0.0008346289	256.35	1.19	1.550027e+10	PENALTY1
POWER	0.0009619512	327.15	379.03	0.0006260868	POWER
QUARTC	0.0009984431	1310.49	136.01	0.1602396	QUARTC
SINQUAD	0.0008996637	6.32	2.70	0.0002266703	SINQUAD
SPARSINE	0.166625	526.06	692.71	0.2315292	SPARSINE
SPMSRTLS	0.0008177106	48.54	176.65	0.0007488716	SPMSRTLS
VAREIGVL	0.0009012548	35.42	166.75	0.000997246	VAREIGVL
WOODS	0.005143096	445.64	10.68	107.5507	WOODS

Táto tabuľka poukazuje na značné úskalia v implementácii metódy s využitím transformácie. Hoci v úlohách ktoré boli časovo najnáročnejčie pre klasickú úlohu sa metóda transformácie ukázala byť veľmi opodstatnená(konkrétne úlohy BIGGSB1 a QUARTC), v úlohách ktoré boli menej časovo náročné pre klasickú metódu výrazne zaostávala. Toto sa dá pripísať istej časovej náročnosti vyžadovanej na rozšírenie transformačnej matice a získanie posunu využitím riešenia transformovanej úlohy, čo sa môže stať relatívne drahým v prípade že samotná netransformovaná úloha je rýchlo riešiteľná. V neposlednom rade to však môže poukazovať na nedostatky pri návrhu rutín v programe.
Každopádne by bolo potrebné vykonať experimenty na úlohách väčších rozmerov, aby sa dali povedať jednoznačné závery.

5 Záver

V tejto práci sme sa venovali úlohám s ohraničeným krokom. Využijúc kvadratickú aproximáciu účelovej funkcie sme rozobrali rôzne metódy riešenia podúlohy, ktorej efektívnosť má značný vplyv na rýchlosť a kvalitu konvergencie metód s ohraničeným krokom.

Skúmaním vlastností presného riešenia úlohy s lokálne ohraničeným krokom sme si ukázali možnosti transformácie úlohy do podpriestoru a možnosti riešenia pôvodnej úlohy pomocou úlohy s podstatne menším rozmerom. Rovnako sme si načrtli hlavné numerické úskalia, ktoré táto transformácia prináša a navrhli základné metódy ako sa im vyhnúť.

Na konkrétnom príklade so stredne veľkým počtom premenných sme potom realizovali experiment a ukázali efektivitu a úskalia využitia transformácie v prípade získavania približného riešenia podúlohy skrátenou metódou konjugovaných gradientov.

Bolo by vhodné urobiť ešte testy v prípade využitia presného riešenia podúlohy, čím by sa v prvých iteráciach lepšie využil rozmer transformovanej úlohy.

Referencie

[1] Hong Wang-Zhou, Ya-xiang Yuan: *A subspace Implementation of Quasi-Newton Trust Region Methods for Unconstrained Optimization*, Numerische Mathematik 104(2006) 241-269

[2] Ya-xiang Yuan: *Nonlinear Optimization: Trust Region Algorithms*, Tsinghua University Press, Beijing, 1994,pp.84-102

[3] Ya-xiang Yuan: *A Review of Trust Region Algorithms for Optimization*, Oxford University Press, Oxford, 2000, pp.271-282

[4] M.Hamala: *Študijné texty k prednáškam z Nelineárneho programovania*, rukopis v knižnici FMFI UK

[5] N.Gould, D.Orban, A.Sartenaer, Ph.L.Toint: *Sensitivity of Trust-Region Algorithms to Their Parameters*, 4OR: A Quarterly Journal of Operations Research, Springer Berlin / Heidelberg, Volume 3, Number 3 / September, 2005, pp.227-241

[6] Ya-xiang Yuan: *On the truncated conjugate gradient method*, Report, ICM99003, ICMSEC, Chinese Academy of Sciences. Beijing, China, 1999

[7] D.C.Sorensen: *Newton's Method with a Model Trust Region Modification*, SIAM J. Numerical Analysis, 19(1982), pp.409-426

[8] N.Gould, D.Orban, Ph.L.Toint: *CUTEr (and SifDec), a Constrained and Unconstrained Testing Environment, revisited*, ACM Trans. Math. Softw., 29(2003), pp.373-394

[9] N.Gould, D.Orban, Ph.L.Toint: *General CUTEr documentation*

[10] A.R.Conn, N.Gould, D.Orban, Ph.L.Toint: *The SIF Reference Report (revised version)*

Program

```
/*********************************************************************/
/***       TRUSTREG.H                                              ***/
/*********************************************************************/

#include <stdio.h>

#ifndef TRUSTREG_H
#define TRUSTREG_H

#include "trsubproblem.h"

typedef struct tr_param {
    double tau1;    /* new point acceptation criterion */
    double tau2;    /* delta  resize criterion */
    double a1;      /* reduction ratio criterion for TR */
    double a2;      /* trusted rgion reduction */
    double eps;     /* stopping condition */
    int kwn;        /* type of kwasi-newton formulae
                     *       0 - BFGS
                     *       1 - SR1
                     */
    int subsp;      /* boolean - use subspace transformation?
                     *       0 - NO
                     *       1 - YES
                     */
    int k_max;      /* maximum number of iterations, 0 = infinite */

} TR_PARAM;

typedef struct trustreg {
    int n;         /* number of variables */
    double *x0;    /* starting point x0 */
    double *x;     /* current iterate */
    double *g;     /* current gradient */
    double *s;     /* current step */
    double *x_new; /* new point */
    double *g_new; /* gradient in new point */

    TRS *trs;

    double cur_objval; /* current value of object fction */
    double new_objval; /* new value of object fction */
    double aprval; /* approximation fction value */
    double g_norm;     /* current gradient norm */

    double (*objvalues) ( double*, double* );
double (*objval) ( double* );
void (*objgrad) ( double*, double* );
```

```
void (*solve_trs) ( TRS* );
int (*update_Hessian) ( TRS* );

    TR_PARAM param; /* trust region parameters */
    int iter;        /* number of iterations */
} TRUSTREG;

extern int solvetr( TRUSTREG *tr );
extern int adjust_TR( TRUSTREG *tr );
extern int update_Hessian_BFGS( TRS *trs );
extern int update_Hessian_SR1( TRS *trs );
extern void get_newpoint_values( TRUSTREG *tr );
extern TRUSTREG *inittr( void );
extern void freetr( TRUSTREG *tr );
extern int opentr( TRUSTREG *tr, double *x0 );


#endif

/*************************************************************************/
/***                TRUSTREG.C                                         ***/
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include "trustreg.h"
#include "trsubproblem.h"
#include "myalloc.h"

/*----------------- SOLVETR -----------------*/

/* Trust Region algorithm */
int solvetr( TRUSTREG *tr )
{
  int retval;

  /* main Algorithm */
  retval = 0;
  while( retval == 0 && (tr->param.k_max == 0 || tr->iter < tr->param.k_max)
         && tr->g_norm >= tr->param.eps ) {
    /* solve Trust Region Subproblem */
    tr->solve_trs( tr->trs );
    if( tr->trs->s_norm < tr->trs->eps * tr->trs->eps )
      return -1;

    /* obtain information about new point */
    get_newpoint_values( tr );

    /* trasform to subspace if wanted */
    if( tr->trs->r < tr->param.subsp )
```

```
      transform2subspace( tr->trs, tr->n, tr->g_new );

    /* adjust Trust Region and update Hessian matrix aproximation */
    retval = adjust_TR( tr );

    tr->iter++;
  }

  return retval;
}

/*----------------- ADJUST_TR -----------------*/

/* adjust Trust Region radius, point, etc... */
int adjust_TR( TRUSTREG *tr )
{
  double Pred, Ared;
  double *x_new;
  double r, retval;

  Pred = -1 * tr->aprval;
  Ared = tr->cur_objval - tr->new_objval;

  r = Ared / Pred;

  /* adjust TR and Hessian matrix */
  retval = tr->update_Hessian( tr->trs );
  if( retval != 0 )
    return 1;

  tr->trs->s_norm_prev = tr->trs->s_norm;
  tr->trs->g_norm_prev = tr->g_norm;
  if( r > tr->param.tau1 ) {
    int i;

    x_new = tr->x;
    tr->x = tr->x_new;
    tr->x_new = x_new;

    x_new = tr->g;
    tr->g = tr->g_new;
    tr->g_new = x_new;
    if( tr->trs->r >= tr->param.subsp )
      tr->trs->g = tr->g;
    tr->cur_objval = tr->new_objval;
    tr->g_norm = 0;
    for( i = 0; i < tr->n; i++)
      tr->g_norm += tr->g[i] * tr->g[i];
    tr->g_norm = sqrt( tr->g_norm );
  }
```

```
  if( r < tr->param.tau1 ) {
    tr->trs->delta = tr->param.a1 * tr->trs->delta;
  } else if( r >= tr->param.tau2 )
    tr->trs->delta = tr->param.a2 * tr->trs->delta;
  return 0;
}



/*----------------- UPDATE_HESSIAN -----------------*/

/* Update Hessian using BFGS formulae */
int update_Hessian_BFGS( TRS *trs )
{
  int i, j;
  double c1, c2;
  double *Bs;

  MYMALLOC( Bs, trs->r, double );

  c1 = c2 = 0;
  for( i = 0; i < trs->r; i++ ) {
    Bs[i] = 0;
    for( j = 0; j < trs->r; j++ ) {
      if( j < i )
        Bs[i] += trs->B[i][j] * trs->s[j];
      else
        Bs[i] += trs->B[j][i] * trs->s[j];
    }
    c2 += trs->s[i] * trs->y[i];
    c1 += trs->s[i] * Bs[i];
  }

/* small c1 or c2 => Hessian matrix reinitialization */
  if( c1 != c1 || c2 != c2 ) {
    MYFREE( Bs );
    return 1;
  } else if( fabs(c1) > MYZERO && fabs(c2) > MYZERO ) {
    for( i = 0; i < trs->r; i++ )
      for( j = 0; j <= i; j++ ) {
        trs->B[i][j] -= Bs[i] * Bs[j] / c1 - trs->y[i] * trs->y[j] / c2;
        if( trs->B[i][j] != trs->B[i][j] )
          return 1;
      }
  }

  MYFREE( Bs );
  return 0;
}
```

```
/* Update Hessian using SR1 formulae */
int update_Hessian_SR1( TRS *trs )
{
  int i, j;
  double *y_Bs;
  double c;

  MYMALLOC( y_Bs, trs->r, double );

  /* calculate y - B * s */
  c = 0;
  for( i = 0; i < trs->r; i++ ) {
    y_Bs[i] = trs->y[i];
    for( j = 0; j < trs->r; j++ ) {
      if( j < i )
        y_Bs[i] -= trs->B[i][j] * trs->s[j];
      else
        y_Bs[i] -= trs->B[j][i] * trs->s[j];
    }
    c += y_Bs[i] * trs->s[i];
  }

  if( c != c ) {
    MYFREE( y_Bs );
    return 1;
  } else if( fabs(c) > MYZERO ) {
    for( i = 0; i < trs->r; i++ )
      for( j = 0; j <= i; j++ ) {
        trs->B[i][j] += y_Bs[i] * y_Bs[j] / c;
        if( trs->B[i][j] != trs->B[i][j] )
          return 1;
      }
  }

  MYFREE( y_Bs );
  return 0;
}

/*----------------- GET_NEWPOINT_VALUES -----------------*/

/* compute values for new point candidate */
void get_newpoint_values( TRUSTREG *tr )
{
  double v;
  int i, j;

  /* calculate Aproximation function value */
  if( tr->param.subsp != 0 && tr->trs->r == tr->param.subsp
                           && tr->trs->r < tr->n ) {
```

```
  MYREALLOC( tr->trs->B, tr->n, double* );
}
if( tr->trs->r == tr->n )
  tr->s = tr->trs->s;

tr->aprval = 0.0;
for( i = 0; i < tr->n; i++ ) {
  v = 0;
  if( tr->trs->r < tr->param.subsp
      || (tr->param.subsp != 0 && tr->trs->r == tr->param.subsp) )
    tr->s[i] = 0;
  if( tr->param.subsp != 0 && tr->trs->r == tr->param.subsp
                           && tr->trs->r <= i ) {
    MYMALLOC( tr->trs->B[i], i+1, double );
    for( j = 0; j < i; j++ )
      tr->trs->B[i][j] = 0.0;
    tr->trs->B[i][i] = 1.0;
  }

  for( j = 0; j < tr->trs->r; j++ ) {
    if( i < tr->trs->r ) {
      if( j < i )
        v += tr->trs->B[i][j] * tr->trs->s[j];
      else
        v += tr->trs->B[j][i] * tr->trs->s[j];
    }
    if( tr->trs->r < tr->param.subsp
        || (tr->param.subsp != 0 && tr->trs->r == tr->param.subsp) )
      tr->s[i] += tr->trs->Z[i][j] * tr->trs->s[j];
  }
  if( i < tr->trs->r ) {
    tr->aprval += 0.5 * v * tr->trs->s[i];
    tr->aprval += tr->trs->g[i] * tr->trs->s[i];
  }
  if( tr->param.subsp != 0 && tr->trs->r == tr->param.subsp )
    MYFREE( tr->trs->Z[i] );
/* new point */
  tr->x_new[i] = tr->x[i] + tr->s[i];
}

if( tr->param.subsp !=0 && tr->trs->r == tr->param.subsp ) {
  MYFREE( tr->trs->Z );
  MYFREE( tr->trs->y );
  tr->trs->s = tr->s;
  tr->trs->r = tr->n;
  MYMALLOC( tr->trs->y, tr->n, double );
}

/* objective function value and gradient */
tr->new_objval = tr->objvalues( tr->x_new, tr->g_new );
```

```
  if( tr->trs->r >= tr->param.subsp ) {
    for( i = 0; i < tr->n; i++ )
      tr->trs->y[i] = tr->g_new[i] - tr->g[i];
    if( tr->param.subsp != 0 )
      tr->param.subsp = 0;
  }
}


/*----------------- OTHER_STUFF -----------------*/

/* Init problem */
TRUSTREG *inittr( void )
{
  TRUSTREG *ltr;

  MYMALLOC( ltr, 1, TRUSTREG );
  /* Null tr parameters */
  ltr->n = 0;
  ltr->x0 = NULL;
  ltr->x = NULL;
  ltr->g = NULL;
  ltr->s = NULL;
  ltr->x_new = NULL;
  ltr->g_new = NULL;

  MYMALLOC( ltr->trs, 1, TRS );
  ltr->trs->r = ltr->n;
  ltr->trs->Z = NULL;
  ltr->trs->B = NULL;
  ltr->trs->g = NULL;
  ltr->trs->s = NULL;
  ltr->trs->y = NULL;
  ltr->trs->delta = ltr->trs->eps = ltr->trs->k = 0;
  ltr->trs->prev_accept = ' ';

  ltr->cur_objval = ltr->new_objval = ltr->aprval = ltr->g_norm = 0;

  ltr->objval = NULL;
  ltr->objgrad = NULL;
  ltr->solve_trs = NULL;
  ltr->update_Hessian = NULL;

  ltr->trs->delta = 1.0;
  ltr->param.tau1 = 0.0001;
  ltr->param.tau2 = 0.99;
  ltr->param.a1 = 0.25;
  ltr->param.a2 = 3.5;
  ltr->param.eps = 0.00001;
  ltr->param.kwn = 0; /* use BFGS */
```

```
  ltr->param.subsp = 0; /* don't use subspace tranformation */
  ltr->param.k_max = 1000;
  ltr->iter = 0;

  return ltr;
}


/* Free tr memory */
void freetr( TRUSTREG *tr )
{
  int i;

  if( tr == NULL )
    return;

  if( tr->trs != NULL ) {
    if( tr->trs->B != NULL ) {
      for( i = 0; i < tr->n; i++ ) {
if( i < tr->trs->r )
          MYFREE( tr->trs->B[i] );
if( tr->trs->Z != NULL )
          MYFREE( tr->trs->Z[i] );
      }
      MYFREE( tr->trs->B );
      if( tr->param.subsp > tr->trs->r ) {
        if( tr->trs->Z != NULL )
          MYFREE( tr->trs->Z );
        if( tr->trs->g != NULL )
          MYFREE( tr->trs->g );
        if( tr->trs->s != NULL )
          MYFREE( tr->trs->s );
      }
    }
  }
  MYFREE( tr->trs );

  if( tr->g != NULL )
    MYFREE( tr->g );
  if( tr->x != NULL )
    MYFREE( tr->x );
  if( tr->iter < tr->param.subsp && tr->s != NULL )
    MYFREE( tr->s );
  if( tr->x_new != NULL )
    MYFREE( tr->x_new );
  if( tr->g_new != NULL )
    MYFREE( tr->g_new );
  MYFREE( tr );
}
```

```
/* TR initialization */
int opentr( TRUSTREG *tr, double *x0 )
{
  int i, j;

  if( tr == NULL || tr->n == 0 ) {
    printf("opentr: tr is not correctly defined...\n");
    return 1;
  }

  /* Memory allocation for starting, current and gradient vector */
  tr->x0 = x0;
  MYMALLOC( tr->x, tr->n, double );
  MYMALLOC( tr->g, tr->n, double );
  MYMALLOC( tr->x_new, tr->n, double );
  MYMALLOC( tr->g_new, tr->n, double );

  /* gradient, gradient norm, set up 'iterating' point x */
  tr->cur_objval = tr->objvalues( tr->x0, tr->g );
  tr->g_norm = 0;
  for( i = 0; i < tr->n; i++ ) {
    tr->g_norm += tr->g[i] * tr->g[i];
    tr->x[i] = tr->x0[i];
  }
  tr->g_norm = sqrt( tr->g_norm );
  if( tr->g_norm < tr->param.eps ) {
    printf("Already in stationary point, g_norm = %.7lf\n", tr->g_norm);
    exit(1);
  }

  /* Hessian matrix aproximation initialization */
  if( tr->trs != NULL ) {
    if( tr->param.subsp > 0 ) {
      tr->trs->r = 1;
      /* transformation matrix Z, transformed g */
      MYMALLOC( tr->trs->Z, tr->n, double * );
      MYMALLOC( tr->trs->g, tr->trs->r, double );
      *tr->trs->g = tr->g_norm;
      MYMALLOC( tr->s, tr->n, double );
    } else {
      tr->trs->r = tr->n;
      tr->trs->g = tr->g;
      tr->trs->Z = NULL;
      tr->s = NULL;
    }

    MYMALLOC( tr->trs->B, tr->trs->r, double * );
    for( i = 0; i < tr->n; i++ ) {
      if( i < tr->trs->r )
        MYMALLOC( tr->trs->B[i], tr->trs->r, double );
```

```
      /* Hessian matrix */
      for( j = 0; j < tr->trs->r; j++ ) {
if( i < tr->trs->r && j <= i ) {
          if( i == j )
            tr->trs->B[i][i] = 1.0;
          else
    tr->trs->B[i][j] = 0.0;
}
      }
      /* transformation matrix Z */
      if( tr->param.subsp > 0 ) {
        MYMALLOC( tr->trs->Z[i], tr->trs->r, double );
        tr->trs->Z[i][0] = tr->g[i] / tr->g_norm;
      }
    }
    MYMALLOC( tr->trs->s, tr->trs->r, double );
    MYMALLOC( tr->trs->y, tr->trs->r, double );
  } else
    return 1;

  /* Trust Region Algorithm initializations */
  if( tr->param.kwn == 0 )
    tr->update_Hessian = update_Hessian_BFGS;
  else
    tr->update_Hessian = update_Hessian_SR1;

  tr->solve_trs = truncated_CG;

  return 0;
}

/*************************************************************************/
/***                TRSUBPROBLEM.H                                     ***/
/*************************************************************************/

#include <stdio.h>

#ifndef TRSUBPROBLEM_H
#define TRSUBPROBLEM_H

#define MYZERO 0.00000001

#define MAX(x,y) ( (x) > (y) ? (x) : (y) )
#define MIN(x,y) ( (x) < (y) ? (x) : (y) )
#define ABS(x) ( (x) > 0 ? (x) : -(x) )
#define SGN(x) ( (x) > 0 ? (1.0) : (-1.0) )

typedef struct trs {
int r;
double **Z;    /* subspace transformation matrix */
```

```
    double **B;    /* aproximation of Hessian matrix */
    double *g;     /* current gradient */
    double *s;     /* trial step solving TR subproblem */
    double *y;     /* y = g_new - g */
    double s_norm; /* trial step norm */
    double s_norm_prev;
    double g_norm_prev;
    int k;         /* iterations counter */
double delta;  /* initial size of trusted region */
    double eps;    /* stopping condition */
    double v;      /* subspace transformation condition */
    double mu1;
    double mu2;
    char prev_accept; /* previous acceptation for Z_k... */
    double rho_prev;

} TRS;

extern void truncated_CG( TRS *trs );
extern double get_tau( double *p, double *d, double delta, int n );
extern void transform2subspace( TRS *trs, int n, double *g_new );

#endif

/*********************************************************************/
/***                TRSUBPROBLEM.C                                 ***/
/*********************************************************************/

#include <stdio.h>
#include <math.h>

#include "trsubproblem.h"
#include "myalloc.h"

/*----------------- SOLVE_TRS -----------------*/

/* Truncated Conjugate Gradient method for TRS */
void truncated_CG( TRS *trs )
{
  double *s, *g, *p, *Bs;
  double alfa, mu, g_g, gn_gn, sBs, t;
  int i, j;

  MYMALLOC( s, trs->r, double );
  MYMALLOC( p, trs->r, double );
  MYMALLOC( g, trs->r, double );
  MYMALLOC( Bs, trs->r, double );

  /* Cauchy iteration */
  trs->k = 0;
```

```
sBs = g_g = 0;
for( i = 0; i < trs->r; i++ ) {
  trs->s[i] = 0;
  g_g += trs->g[i] * trs->g[i];

  Bs[i] = 0;
  for( j = 0; j < trs->r; j++ ) {
    if( i == 0 )
      s[j] = -trs->g[j];
    if( j < i )
      Bs[i] += trs->B[i][j] * s[j];
    else
      Bs[i] += trs->B[j][i] * s[j];
  }
  sBs += s[i] * Bs[i];
}

/* test of solution */
if( sBs <= 0 ) {
  trs->s_norm = 0;
  t = get_tau( trs->s, s, trs->delta, trs->r );
  for( i = 0; i < trs->r; i++ ) {
    trs->s[i] += t * s[i];
    trs->s_norm += trs->s[i] * trs->s[i];
  }
  trs->s_norm = sqrt( trs->s_norm );

  MYFREE( g );
  MYFREE( s );
  MYFREE( Bs );
  return;
}

alfa = g_g / sBs;
trs->s_norm = 0;
for( i = 0; i < trs->r; i++ ) {
  p[i] = alfa * s[i];
  trs->s_norm += p[i] * p[i];
}
trs->s_norm = sqrt( trs->s_norm );
trs->k++;

/* start real iterating */
while( trs->s_norm < trs->delta ) {
  /* calculate gradient */
  gn_gn = 0;
  for( i = 0; i < trs->r; i++ ) {
    g[i] = 0;
    for( j = 0; j < trs->r; j++ ) {
      if( j < i )
```

```
          g[i] += trs->B[i][j] * p[j];
        else
          g[i] += trs->B[j][i] * p[j];
      }
      g[i] += trs->g[i];
      gn_gn += g[i] * g[i];
    }

    /* test of end */
    if( sqrt(gn_gn) < trs->eps * trs->eps ) {
      MYFREE( trs->s );
      trs->s = p;
      MYFREE( g );
      MYFREE( Bs );
      return;
    }

    /* get new point */
    mu = gn_gn / g_g;
    sBs = 0;
    for( i = 0; i < trs->r; i++ ) {
      Bs[i] = 0;
      for( j = 0; j < trs->r; j++ ) {
if( i == 0 )
  s[j] = -g[j] + mu * s[j];
        if( j < i )
          Bs[i] += trs->B[i][j] * s[j];
        else
          Bs[i] += trs->B[j][i] * s[j];
      }
      sBs += s[i] * Bs[i];
    }

    /* test of solution */
    if( sBs <= 0 ) {
      trs->s_norm = 0;
      t = get_tau( trs->s, s, trs->delta, trs->r );
      for( i = 0; i < trs->r; i++ ) {
        trs->s[i] += t * s[i];
        trs->s_norm += trs->s[i] * trs->s[i];
      }
      trs->s_norm = sqrt( trs->s_norm );

      MYFREE( g );
      MYFREE( s );
      MYFREE( Bs );
      return;
    }

    alfa = gn_gn / sBs;
```

```
    trs->s_norm = 0;
    for( i = 0; i < trs->r; i++ ) {
      trs->s[i] = p[i];
      p[i] = p[i] + alfa * s[i];
      trs->s_norm += p[i] * p[i];
    }
    trs->s_norm = sqrt( trs->s_norm );
    g_g = gn_gn;

    trs->k++;
  }

  t = get_tau( trs->s, s, trs->delta, trs->r );
  trs->s_norm = 0;
  for( i = 0; i < trs->r; i++ ) {
    trs->s[i] += t * s[i];
    trs->s_norm += trs->s[i] * trs->s[i];
  }
  trs->s_norm = sqrt( trs->s_norm );

  MYFREE( s );
  MYFREE( p );
  MYFREE( g );
  MYFREE( Bs );
}

/*----------------- SUBSPACE_TRANSFORMATION -----------------*/

void transform2subspace( TRS *trs, int n, double *g_new )
{
  double *u, *z, ro;
  double c1, c2, gnorm;
  int i, j, k;

  /* reducing Z_k... */
  if( fabs(trs->s[trs->r-1]) <= trs->mu1 * trs->s_norm_prev
      && trs->prev_accept == 'X'
      && trs->rho_prev < trs->mu2 * trs->g_norm_prev ) {
    trs->r -= 1;
  }

  MYMALLOC( u, trs->r, double );

  /* initialize vectors u and z */
  gnorm = ro = 0.0;
  for( i = 0; i < n; i++ ) {
    c1 = 0.0;
    if( i < trs->r )
      u[i] = 0.0;
    for( j = 0; j < n; j++ ) {
```

```
    if( i < trs->r ) {
      u[i] += trs->Z[j][i] * g_new[j];
    }

  /* calculate ro */
    if( i == j )
      c2 = 1.0;
    else
      c2 = 0.0;
    for( k = 0; k < trs->r; k++ )
      c2 -= trs->Z[i][k] * trs->Z[j][k];
    c1 += c2 * g_new[j];
  }
  ro += c1 * c1;
  gnorm += g_new[i] * g_new[i];
  if( i < trs->r ) {
    trs->y[i] = u[i] - trs->g[i];
    trs->g[i] = u[i];
  }
}

ro = sqrt( ro );
gnorm = sqrt( gnorm );

if( ro < trs->v * gnorm ) {
 trs->prev_accept = ' ';
  /* g_new is in subspace defined by Z, nothing happend */
  MYFREE( u );
  return;
} else {
  trs->rho_prev = ro;
  trs->prev_accept = 'X';
}

/* obtain vector z_new = (g_new - Zu) / ro_new */
MYREALLOC( trs->B, trs->r+1, double * );
MYMALLOC( trs->B[trs->r], trs->r+1, double );
for( i = 0; i < n; i++ ) {
  /* calculate B */
  if( i < trs->r )
    trs->B[trs->r][i] = 0.0;
  /* calculate z */
  MYREALLOC( trs->Z[i], trs->r+1, double );
  z = &trs->Z[i][trs->r];
  *z = 0.0;
  for( j = 0; j < trs->r; j++ )
    *z += trs->Z[i][j] * u[j];
  *z = (g_new[i] - *z) / ro;
}
trs->B[trs->r][trs->r] = 1.0;
```

```
  trs->r++;

  /* update vectors g and s */
  MYREALLOC( trs->g, trs->r, double );
  trs->g[trs->r-1] = ro;
  MYREALLOC( trs->s, trs->r, double );
  trs->s[trs->r-1] = 0;
  MYREALLOC( trs->y, trs->r, double );
  trs->y[trs->r-1] = ro;

  MYFREE( u );
}

/*----------------- OTHER_STUFF -----------------*/

double get_tau( double *p, double *d, double delta, int n )
{
  double a, b, c, D, v;
  double retval;
  int i;

  a = b = c = 0;
  for( i = 0; i < n; i++ ) {
    a += d[i] * d[i];
    b += p[i] * d[i];
    c += p[i] * p[i];
  }
  b *= 2;
  c -= delta * delta;

  D = b * b - 4 * a * c;
  v = sqrt( D ) - b;
  if( v < 0 )
    v *= -1;
  retval = v / (2 * a);

  return retval;
}
```

Makefile:

```
trustregionma.o: trustregionma.c cuter.h libtrustregion.a myalloc.h trustreg.h trsubproblem.h
gcc $(CFLAGS) -c trustregionma.c

trustreg.o: trustreg.c trustreg.h trsubproblem.h myalloc.h
gcc $(CFLAGS) -c trustreg.c

trsubproblem.o: trsubproblem.c trsubproblem.h myalloc.h
gcc $(CFLAGS) -c trsubproblem.c

libtrustregion.a: trustreg.o trsubproblem.o
```

```
ar rcvu libtrustregion.a trustreg.o trsubproblem.o
ranlib libtrustregion.a

clean:
rm -f *.o *.a trustregion
```

trustregionma.c:

```
/* ================================================
 * CUTEr interface for TRUSTREGION package Jul.22, 2007
 *
 *
 *
 *
 *
 * ================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define TRUSTREGION

#ifdef __cplusplus
extern "C" {   /* To prevent C++ compilers from mangling symbols */
#endif

#include "cuter.h"
#include "trustreg.h"
#include "myalloc.h"

/* global data */
typedef struct keyword keyword;
struct keyword {
    char *name;
    int type; /* types: 0 - double, 1 - integer, 2 - boolean */
    double value;
};

#define KW(a,b,c) {a,b,c}
#define KW_NUMBER 12 /* number of keywords in keyword structure */

static keyword keywds[] = {
    KW("delta", 0, 1.0),
    KW("tau1", 0, 0.0001),
    KW("tau2", 0, 0.5),
    KW("a1", 0, 0.25),
    KW("a2", 0, 3.5),
    KW("eps", 0, 0.00001),
    KW("kwn", 1, 0),
    KW("subsp", 1, 0),
    KW("v", 0, 0.001),
    KW("mu1", 0, 0.1),
    KW("mu2", 0, 0.3),
```

```
    KW("k_max", 1, 10000)
};

static int bad_opns = 0;
static int spec = 0;

/* prototypes */
    double objval( double *x );
    void objgrad( double *c, double *x );
    double objvalues( double *x, double *g );
    int read_spc( char *spec_name );
    int set_opns( TRUSTREG *tr );
    keyword *get_kwtype( char *opt );
    void getinfo( integer, integer, doublereal*, doublereal*,
                  doublereal*, doublereal*, logical*, logical*,
                  VarTypes* );

/*
 * Global variables used by auxilliary library in ccuter.c
 */

    integer CUTEr_nvar;        /* number of variables */
    integer CUTEr_ncon;        /* number of constraints */
                               /*   part of the Hessian of the Lagrangian */


/* Counters for number of function and derivative evaluations */

    int count_f = 0;
    int count_g = 0;

/*-------------------------------------------------------------------------*/

/* ============ */
/* Main program */
/* ============ */

int main( void ) {
  TRUSTREG *tr;    /* TRUSTREGION data structure */
  double *x_new;
  int retval;

  char *fname = "OUTSDIF.d"; /* CUTEr data file */
  integer funit = 42;        /* FORTRAN unit number for OUTSDIF.d */
  integer iout = 6;          /* FORTRAN unit number for error output */
  integer ierr;              /* Exit flag from OPEN and CLOSE */

  VarTypes vtypes;

  integer    *indvar = NULL, *indfun = NULL, ncon_dummy, nnzj;
```

```
doublereal *x, *bl, *bu, *dummy1, *dummy2;
doublereal *v = NULL, *cl = NULL, *cu = NULL, *c = NULL, *cjac = NULL;
logical    *equatn = NULL, *linear = NULL;
char       *pname, *vnames, *gnames;
logical     efirst = FALSE_, lfirst = FALSE_, nvfrst = FALSE_, grad;
logical     constrained = FALSE_;

real        calls[7], cpu[2];
integer     nlin = 0, nbnds = 0, neq = 0;
doublereal  dummy;
integer     status;
int         i;

/* Open problem description file OUTSDIF.d */
ierr = 0;
FORTRAN_OPEN( &funit, fname, &ierr );
if( ierr ) {
  printf("Error opening file OUTSDIF.d.\nAborting.\n");
  exit(1);
}

/* Determine problem size */
CDIMEN( &funit, &CUTEr_nvar, &CUTEr_ncon );

/* Determine whether to call constrained or unconstrained tools */
if( CUTEr_ncon ) {
  constrained = TRUE_;
  printf("TRUSTREGION solve only unconstrained problems...\nAborting\n");
  exit(1);
}

/* Seems to be needed for some Solaris C compilers */
ncon_dummy = CUTEr_ncon + 1;

/* Reserve memory for variables, bounds, and multipliers */
/* and call appropriate initialization routine for CUTEr */
MALLOC( x,      CUTEr_nvar, doublereal );
MALLOC( bl,     CUTEr_nvar, doublereal );
MALLOC( bu,     CUTEr_nvar, doublereal );

MALLOC( equatn, 1, logical    );
MALLOC( linear, 1, logical    );
MALLOC( cl, 1, doublereal );
MALLOC( cu, 1, doublereal );
USETUP( &funit, &iout, &CUTEr_nvar, x, bl, bu, &CUTEr_nvar );

/*----------------------------TRUSTREG_START-----------------------------*/

/* now we can start to define the problem for TRUSTREGION */
tr = inittr();
```

```
tr->n = CUTEr_nvar;

bad_opns = read_spc( "TRUSTREGION.SPC" );
if( bad_opns != 0 ) {
  printf("Error in TRUSTREGION.SPC, line %d\n", bad_opns);
  return 0;
}

retval = set_opns( tr );
if( retval != 0 ) {
  printf("TRUSTREG.SPC not correctly defined( check out numbers... ).\n");
  return 0;
}

tr->objvalues = objvalues;

/* TR parameters description */
printf("parameters: \n");
printf("    delta   = %.8lf\n", tr->trs->delta);
printf("    tau1    = %.8lf\n", tr->param.tau1);
printf("    tau2    = %.8lf\n", tr->param.tau2);
printf("    a1      = %.8lf\n", tr->param.a1);
printf("    a2      = %.8lf\n", tr->param.a2);
printf("    eps     = %.8lf\n", tr->param.eps);
if( tr->param.kwn == 0 )
  printf("    kwn     = BFGS\n");
else
  printf("    kwn     = SR1\n");
printf("    subsp   = %d\n", tr->param.subsp);
printf("    k_max   = %d\n\n", tr->param.k_max);

/* SOLVE TR PROBLEM */
retval = opentr( tr, x );
if( retval != 0 ) {
  printf("Error in opentr...\n");
  exit(1);
}

status = solvetr( tr );

/* Obtain basic info on problem */
/* Output some TRUSTREGION info */
printf( "# Iterations\t%-5d\n", tr->iter );
printf( "# Eval f(x) \t%-6d\n", count_f );
printf( "# Eval g(x) \t%-6d\n", count_g );

/*-----------------------TRUSTREG_END-------------------------------------*/
/* Get problem name */
MALLOC( pname, FSTRING_LEN+1, char );
MALLOC( vnames, CUTEr_nvar*(FSTRING_LEN+1), char );
```

```
if( constrained ) {
  MALLOC( gnames, CUTEr_ncon*(FSTRING_LEN+1), char );
  CNAMES( &CUTEr_nvar, &CUTEr_ncon, pname, vnames, gnames );
  FREE( gnames );
} else {
  UNAMES( &CUTEr_nvar, pname, vnames );
}
FREE( vnames );

/* Make sure to null-terminate problem name */
pname[FSTRING_LEN] = '\0';
i = FSTRING_LEN - 1;
while( i-- > 0 && pname[i] == ' ') {
  pname[i] = '\0';
}

equatn[0] = FALSE_;
linear[0] = FALSE_;
getinfo( CUTEr_nvar, 1, bl, bu, cl, cu, equatn,
linear, &vtypes );


/* Get CUTEr statistics */
CREPRT( calls, cpu );

printf("\n\n ********************** CUTEr statistics ************************\n\n");
printf(" Code used               : TRUSTREGION\n");
printf(" Problem                 : %-s\n", pname);
printf(" # variables             = %-10d\n", CUTEr_nvar);
printf(" # constraints           = %-10d\n", CUTEr_ncon);
printf(" # linear constraints    = %-10d\n", vtypes.nlin);
printf(" # equality constraints  = %-10d\n", vtypes.neq);
printf(" # inequality constraints= %-10d\n", vtypes.nineq);
printf(" # bound constraints     = %-10d\n", vtypes.nbnds);
printf(" # objective functions   = %-15.7g\n", calls[0]);
printf(" # objective gradients   = %-15.7g\n", calls[1]);
printf(" # objective Hessians    = %-15.7g\n", calls[2]);
printf(" # Hessian-vector prdct  = %-15.7g\n", calls[3]);
printf(" # constraints functions = %-15.7g\n", calls[4]);
printf(" # constraints gradients = %-15.7g\n", calls[5]);
printf(" # constraints Hessians  = %-15.7g\n", calls[6]);
printf(" Exit code               = %-10d\n", status);
printf(" Final f                 = %-15.7g\n",tr->cur_objval);
printf(" Final g_norm            = %-15.7g\n", tr->g_norm);
printf(" Set up time             = %-10.2f seconds\n", cpu[0]);
printf(" Solve time              = %-10.2f seconds\n", cpu[1]);
printf(" ****************************************************************\n\n");

ierr = 0;
FORTRAN_CLOSE( &funit, &ierr );
```

```
  if( ierr ) {
    printf( "Error closing %s on unit %d.\n", fname, funit );
    printf( "Trying not to abort.\n" );
  }

  /* Free workspace */
  FREE( pname );
  FREE( x ); FREE( bl ); FREE( bu );
  FREE( v ); FREE( cl ); FREE( cu );
  /* Free unneeded arrays */
  FREE( equatn );
  FREE( linear );

  freetr( tr );


  return 0;
}

/* ---------------------------------------------- */
/* Interface-specific functions                   */
/* ---------------------------------------------- */

#ifdef  __FUNCT__
#undef  __FUNCT__
#endif
#define __FUNCT__ "objval"
double objval( double *x ) {
  logical grad = FALSE_;
  doublereal *dummy, f;

  count_f++;

  UFN( &CUTEr_nvar, x, &f );
  return f;
}

/* ---------------------------------------------- */

#ifdef  __FUNCT__
#undef  __FUNCT__
#endif
#define __FUNCT__ "objgrad"
void objgrad( double *c, double *x ) {
  logical grad = TRUE_;
  doublereal fdummy;

  count_g++;

  UGR( &CUTEr_nvar, x, c );
```

```
  return;
}

/* ---------------------------------------------- */

#ifdef  __FUNCT__
#undef  __FUNCT__
#endif
#define __FUNCT__ "objvalues"
double objvalues( double *x, double *g ) {
  logical grad = TRUE_;
  doublereal f;

  count_f++; count_g++;

  UOFG( &CUTEr_nvar, x, &f, g, &grad );
  return f;
}

/* ---------------------------------------------- */

#ifdef  __FUNCT__
#undef  __FUNCT__
#endif
#define __FUNCT__ "read_spc"
int read_spc( char *spec_name )
{
  keyword *kw;
  char *CUTEr_loc;
  char *Spec_loc;
  char specline[80];
  char option[15], val[15], comment[50];
  char *valptr, *s1;
  int i, j;
  FILE *specfile;
  int s;

  spec = 1;

  /* open the specs file */
  specfile = fopen(spec_name, "r");
  if( specfile == NULL ) {
   printf("File %s does not exist... \n", spec_name);
   exit(1);
  }
  fgets( specline, 80, specfile );
  s = sscanf(specline, "%s%s%s", option, val, comment);
  i = 0;

  /* read the specs file by lines */
```

```
  while( s && !feof(specfile) ) {
    i++;

    if( (option[0] >= 'a' && option[0] <= 'z')
|| (option[0] >= 'A' && option[0] <= 'Z') ) {
      kw = get_kwtype( option );
      if( kw == NULL ) { /* bad option */
bad_opns++;
fclose( specfile );
return i;
      }

      switch( kw->type ) {
case 0:
          valptr = val+0;
  kw->value = strtod(s1=valptr, &valptr);
  if( valptr == s1 ) {
    fclose( specfile );
    return i;
  }
  break;
case 1:
  valptr = val+0;
  kw->value = atoi(s1=valptr);
  break;
      }
    } else {
      if( option[0] != '*' ) { /* comment */
fclose( specfile );
return i;
      }
    }
    fgets( specline, 80, specfile );
    s = sscanf(specline, "%s%s%s", option, val, comment);
  }

  fclose(specfile);
  return 0;
}

/* ------------------------------------------------ */

#ifdef  __FUNCT__
#undef  __FUNCT__
#endif
#define __FUNCT__ "get_kwtype"
keyword *get_kwtype( char *opt )
{
  keyword *kw = keywds;
  keyword *kw1;
```

```
  int i, retval;

  for( i = 0; i < KW_NUMBER; i++ ) {
    retval = strcmp( opt, kw[i].name );
    if( retval == 0 ) {
      kw1 = kw+i;
      return kw1;
    }
  }

  return NULL;
}

/* ------------------------------------------------ */

#ifdef  __FUNCT__
#undef  __FUNCT__
#endif
#define __FUNCT__ "set_opns"
int set_opns( TRUSTREG *tr )
{
  keyword *kw = keywds;

  tr->trs->delta = kw->value;
  if( kw->value <= 0 ) /* delta */
    return 3;
  kw++;

  tr->param.tau1 = kw->value;
  if( kw->value < 0 ) /* trusted region extension */
    return 4;
  kw++;

  tr->param.tau2 = kw->value;
  if( kw->value <= 0 ) /* reduction ratio criterion for TR */
    return 5;
  kw++;

  tr->param.a1 = kw->value;
  if( kw->value <= 0 ) /* TR reduction */
    return 6;
  kw++;

  tr->param.a2 = kw->value;
  if( kw->value <= 0 ) /* TR reduction */
    return 7;
  kw++;

  tr->trs->eps = tr->param.eps = kw->value;
  if( kw->value <= 0 ) /* TR reduction */
```

```
    return 8;
  kw++;

  tr->param.kwn = kw->value;
  if( kw->value < 0 || kw->value > 1 ) /* kwn algorithm */
    return 9;
  kw++;

  tr->param.subsp = MIN( kw->value, tr->n );
  if( kw->value < 0 ) /* ratio of subspace transformation */
    return 10;
  kw++;

  tr->trs->v = kw->value;
  if( kw->value < 0 ) /* trasnformation criterion parameter */
    return 11;
  kw++;

  tr->trs->mu1 = kw->value;
  if( kw->value < 0.0 || kw->value >= 0.2 ) /* trasnformation criterion parameter */
    return 12;
  kw++;

  tr->trs->mu2 = kw->value;
  if( kw->value < 0.2 || kw->value >= 1.0 ) /* trasnformation criterion parameter */
    return 13;
  kw++;

 tr->param.k_max = kw->value;
  if( kw->value < 0 ) /* maximum number of iterations */
    return 12;

  return 0;
}

/* ---------------------------------------------- */

void getinfo( integer n, integer m, doublereal *bl, doublereal *bu,
              doublereal *cl, doublereal *cu, logical *equatn, logical *linear,
              VarTypes *vartypes )
{
  int i;

  vartypes->nlin = 0; vartypes->neq = 0; vartypes->nbnds = 0; vartypes->nrange = 0;
  vartypes->nlower = 0; vartypes->nupper = 0; vartypes->nineq = 0;
  vartypes->nineq_lin = 0; vartypes->nineq_nlin = 0;
  vartypes->neq_lin = 0; vartypes->neq_nlin = 0;

  for( i = 0; i < n; i++ )
    if( bl[i] > -CUTE_INF || bu[i] < CUTE_INF ) vartypes->nbnds++;
```

```
  for( i = 0; i < m; i++ ) {
    if( linear[i] ) vartypes->nlin++;
    if( equatn[i] ) {
      vartypes->neq++;
      if( linear[i] )
        vartypes->neq_lin++;
      else
        vartypes->neq_nlin++;
    } else {
     if( cl[i] > -CUTE_INF ) {
       if( cu[i] < CUTE_INF )
         vartypes->nrange++;
       else
         vartypes->nlower++; vartypes->nineq++;
       } else {
         if( cu[i] < CUTE_INF )
           vartypes->nupper++; vartypes->nineq++;
       }
       if( !equatn[i] && linear[i] ) {
         vartypes->nineq_lin++;
       } else {
         vartypes->nineq_nlin++;
       }
     }
    }
  return;
}

#ifdef __cplusplus
}    /* Closing brace for  extern "C"  block */
#endif
```

trustreg:

```
#!/bin/csh -f
#  ( Last modified on 23 Dec 2000 at 17:29:56 )
#
# trustreg: apply TRUSTREGION to a problem and delete the executable after use.
#
#{version}
#

#{args}

#{cmds}

#
# Environment check
#

envcheck
```

```
if( $status != 0 ) exit $status

#
#  define a short acronym for the package to which you wish to make an interface
#

setenv caller trustreg
setenv PAC trustregion

#
#  define the name of the subdirectory of $CUTER/common/src/pkg
#  in which the package lies
#

setenv PACKAGE trustregion

#
#  Check the arguments
#

set PRECISION = "double"
@ decode_problem = 0
@ last=$#argv
@ i=1

while ($i <= $last)
  set opt=$argv[$i]
  if("$opt" == '-s')then
    set PRECISION = "single"
  else if("$opt" == '-decode' ) then
    @ decode_problem = 1
  else if("$opt" == '-h' || "$opt" == '--help' )then
    $MYCUTER/bin/helpmsg
    exit 0
  endif
  @ i++
end

#
#  define the system libraries needed by the package
#  using the format -lrary to include library.a
#

setenv SYSLIBS "-ltrustregion"

#  define the name(s) of the object file(s) (files of the form *.o)
#  and/or shared-object libraries (files of the form lib*.so)
#  and/or libraries (files of the form lib*.a) for the TRUSTREGION package.
#  Object files must lie in
#    $MYCUTER/(precision)/bin
```

```
#  while libraries and/or shared-object libraries must be in
#    $MYCUTER/(precision)/lib
#  where (precision) is either single (when the -s flag is present) or
#  double (when there is no -s flag)
#

setenv PACKOBJ ""

#
#  define the name of the package specification file (if any)
#  (this possibly precision-dependent file must either lie in
#  the current directory or in $MYCUTER/common/src/pkg/$PACKAGE/ )
#

setenv SPECS "TRUSTREGION.SPC"

#  this is where package-dependent commands (using the commands defined
#  in the current system.(your system) file) may be inserted prior to
#  decoding the problem file


#  decode the problem file

#{sifdecode}
if( $decode_problem == 1 ) then
  if( $?MYSIFDEC ) then
     $MYSIFDEC/bin/sifdecode $argv
     if ( $status != 0 ) exit $status
  else
     echo " ${caller} : environment variable MYSIFDEC not set"
     echo "      Either SifDec is not installed or you"
     echo "      should properly set MYSIFDEC"
     exit 7
  endif
endif

#  this is where package-dependent commands (using the commands defined
#  in the current system.(your system) file) may be inserted prior to
#  running the package


#  run the package, removing -decode option if present

if( $decode_problem == 1 ) then
  @ n = ${#argv} - 1
  @ i = 1
  set arguments = ''
  while( $i <= $n )
    if( "$argv[$i]" != '-decode' ) then
      set arguments = ( "$arguments" "$argv[$i]" )
```

```
    endif
    @ i++
  end
else
  set arguments = "$argv"
endif

$MYCUTER/bin/runpackage $arguments -n -c

if ( $status != 0 ) exit $status

#  this is where package-dependent commands (using the commands defined
#  in the current system.(your system) file) may be inserted to clean
#  up after running the package
```

sdtrustreg:

```
#!/bin/csh -f
# sdtrustreg: script to decode a sif file and then run TRUSTREGION on the output
#  ( Last modified on 22 Jul 2007 at 16:31:52 )
#
#{version}
#

#{args}

#{cmds}

#
#  define a short acronym for the package to which you wish to make an interface
#

setenv caller sdtrustreg
setenv PAC trustreg

#
#  Check the arguments
#

set PRECISION = "double"
@ decode_problem = 0
@ last=$#argv
@ i=1

while ($i <= $last)
  set opt=$argv[$i]
  if("$opt" == '-s')then
    set PRECISION = "single"
  else if("$opt" == '-decode' ) then
```

```
    @ decode_problem = 1
  else if("$opt" == '-h' || "$opt" == '--help' )then
    $MYCUTER/bin/helpmsg
    exit 0
  endif
  @ i++
end

if( $decode_problem == 0 ) then
  set arguments = ('-decode' "$argv[1-$last]")
else
  set arguments = ("$argv[1-$last]")
endif

# call main script

${PAC} ${arguments}
```