TR－H－054

大規模非線形計画問題に対する逐次線形化法

山川 栄樹
阿部 敦 (奈良先端科技院大)

1994. 2. 18

ATR人間情報通信研究所

〒619-02 京都府相楽郡精華町光台2-2　☎07749-5-1011

ATR Human Information Processing Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-7749-5-1011
Facsimile: +81-7749-5-1008

大規模非線形計画問題に対する逐次線形化法

人間情報通信研究所　第6研究室　　指導者 山川 栄樹
実習生 阿部 敦（奈良先端科技院大）

1　はじめに

現在非線形計画問題も理論的にはかなり整理され、特に最近の並列計算機の進歩により、並列化可能なアルゴリズムを適用してより大きな問題を高速に解くことに注目が集まっている。今回の実習では[1]で提案されているSuccesive Linearization Method - Succesive OverRelaxation(SLM-SOR)、SORと同様にRow-actionアルゴリズムとして分類されるJacobi法を実装したSLM-JacobiOverRelaxation、(SLM-JOR)そして同等の処理をする並列版SLM-JORを実際にプログラムを組んで、それらを比較検討する。

2　問題の定式化 と SLM の基本アルゴリズム

以下のような非線形計画問題を考える。

$$\begin{aligned} NLP \ : \ & min \ \ f(x) \\ s.t. \quad & g_i(x) = 0, \quad i = 1, \cdots, \bar{m} \\ & g_i(x) \leq 0, \quad i = \bar{m}+1, \cdots, m \end{aligned} \tag{1}$$

SLMアルゴリズムの収束性については[1]に詳しく述べられているので、ここではアルゴリズムの概要を紹介するにとどめる。

まず(1)の最適Lagrange乗数をu^*とする時、$r_i \geq |u_i^*|, i = 1, \cdots, m$ならば、次の関数は(1)の正確なペナルティ関数となる。

$$p(x) \equiv f(x) + \sum_{i=1}^{\bar{m}} r_i |g_i(x)| + \sum_{i=\bar{m}+1}^{m} r_i \{g_i(x)\}_+ \tag{2}$$

SLMは(2)の線形近似関数

$$\bar{p}(x,d) \equiv f(x) + \nabla f(x)^T d + \sum_{i=1}^{\bar{m}} r_i |g_i(x) + \nabla g_i(x)^T d| + \sum_{i=\bar{m}+1}^{m} r_i \{g_i(x) + \nabla g_i(x)^T d\}_+ \tag{3}$$

ただし、

$$\{\cdot\}_+ \equiv max\{0, \cdot\} \tag{4}$$

をステップサイズ:d を制限するために付加した凸 2 次項とともに最小化する部分問題 SP

$$min \ \frac{1}{2}\lambda^k d^T C^k d + \bar{p}(x^k, d) \tag{5}$$

を繰り返し解いて原問題の解を得ようとする方法である。

2.1 部分問題 SP について

SLM のアルゴリズム実現する際には、部分問題 (5) と等価な 2 次計画問題に対する Lagrange の双対問題:

$$\begin{aligned} DSP(x, C, \lambda) \ : \ & \min \phi(u) \\ s.t. \quad & -r_i \le u_i \le r_i, \quad i = 1, \cdots, \bar{m}, \\ & 0 \le u_i \le r_i, \quad i = \bar{m}+1, \cdots, m, \end{aligned} \tag{6}$$

を解く。ここに、

$$\begin{aligned} \phi(u) &= \frac{1}{2\lambda}\nabla L^T(x,u)\, C^{-1} \nabla L(x,u) - \sum_{i=1}^{m} u_i b_i \\ \nabla L(x,u) &= \sum_{i=1}^{m} u_i \nabla g_i(x) + \nabla f(x) \\ d &= -\frac{1}{\lambda} C^{-1} \nabla L(x,u) \end{aligned} \tag{7}$$

である。次節に、今回採用したアルゴリズムを書き下す。

2.2 SLM の アルゴリズム

1. $0 < \mu_0 < \mu_1 < \mu_2 < 1, \gamma > 1$ を満たす $\mu_0, \mu_1, \mu_2, \gamma$ と x^1, C^1, λ_1 および $k = 1$ を設定する。

2. 探索幅 d を得るために、 DSP(x^k, C^k, λ_k) を解く

3. ペナルティ関数の実際の変化、ペナルティ関数の一次近似式の変化を次の式に基づいて求める。

$$\Delta p_k \equiv p(x^k) - p(x^k + d^k) \tag{8}$$

$$\Delta\bar{p}_k \equiv p(x^k) - \bar{p}(x^k, d^k) \tag{9}$$

もし、$\Delta\bar{p}_k = 0$ ならば、終了。

4. 実際の変化と　一次近似式から得られた変化の比を次の式を用いて求める。

$$q_k \equiv \Delta p_k / \Delta\bar{p}_k \tag{10}$$

5. q_k の値に従って、x^k, λ^k を更新する。

λ について
$$\lambda_{k+1} = \begin{cases} \gamma \cdot \lambda_k & q_k \leq \mu_1 \text{ のとき} \\ \lambda_k & \mu_1 < q_k \leq \mu_2 \text{ のとき} \\ \lambda_k/\gamma & \mu_2 < q_k \text{ のとき} \end{cases} \tag{11}$$

x^k について
$$x_{k+1} = \begin{cases} x^k & q_k \leq \mu_0 \text{ のとき} \\ x^k + d & \mu_0 < q_k \text{ のとき} \end{cases} \tag{12}$$

C^{k+1} について $\mu_0 < q_k$ ならば、C^{k+1} を修正 BFGS 等 によって、更新する。

6. $k = k + 1$ とし、2 に戻る。

3　DSP の解法

SLM の過程で (6) を 解く手法として、さまざまな方法が 考えられるが、今回は次の 3 種類を適用した。

- 逐次過緩和法 (Succesive OverRelaxation)
- Jacobi OverRelaxation
- 並列版 Jacobi OverRelaxation

ここで SOR は、各反復では特定の双対変数に関してのみ最適化を行うが、反復が進むにつれて、その変数の選択が周期的になるようにして探索を進める方法である。これに対して、JOR は、現在の探索点において、各双対変数に関して独立に最適化を行うことによって得られた探索方向を合成することにより、次の探索点を求める方法と言える。以下に各々のアルゴリズムを紹介する。

3.1 Succesive OverRelaxation のアルゴリズム

1. $0 < \omega < 2$ を満たす ω と制約条件を満たす u^1 を選ぶ。

2. $$\nabla L = \nabla L(x, u^1) \tag{13}$$

3. $i = 1, \cdots, m$ について、順次 $u_i, \nabla L$ を更新する。

(a)

$$u_i^+ = u_i^j - \frac{\omega}{\nabla g_i(x)^T C^{-1} \nabla g_i(x)} \{g_i(x)^T C^{-1} \nabla L - \lambda \cdot g_i(x)\} \tag{14}$$

(b)
$$u_i^{j+1} = \begin{cases} min\{r_i, max(-r_i, u_i^+)\} & ,1 \leq i \leq \bar{m} \text{ のとき} \\ min\{r_i, max(0, u_i^+)\} & ,\bar{m}+1 \leq i \leq m \text{ のとき} \end{cases} \tag{15}$$

(c)
$$\nabla L = \nabla L - u_i^j \cdot g_i(x) + u_i^{j+1} \cdot g_i(x) \tag{16}$$

4. 次の式に従い、d^j と z_i^j を計算する。

$$\begin{aligned} d^j &= -\frac{1}{\lambda} C^{-1} \nabla L \\ z_i^j &= \nabla g_i(x)^T \cdot d^j + g_i(x), \quad i = 1, \cdots, m \end{aligned} \tag{17}$$

$i = 1, \cdots, \bar{m}$ のとき　$(u_i^j = -r_i$　かつ $z_i^j \leq 0)$ または $(u_i^j = r_i$　かつ $z_i^j \geq 0)$
ならば $z_i^j = 0$

$i = \bar{m}+1, \cdots, m$ のとき　$(u_i^j = 0$　かつ $z_i^j \leq 0)$ または $(u_i^j = r_i$　かつ $z_i^j \geq 0)$
ならば $z_i^j = 0$

5. 終了判定

$z_i = 0$ $(i = 1, \cdots, m)$ ならば、終了

それ以外ならば、$j = j + 1$ として 2 に戻る。

ただし、4),5) の収束判定は毎回行なう必要はない。

3.2 Jacobi 法のアルゴリズム

Jacobi OverRelaxation の アルゴリズムは 以下の通りである。

1. $0 < \omega < \bar{\omega}$ を満たす ω と制約条件を満たす u^1 を選ぶ。

2. $i = 1, \cdots, m$ について独立に u_i を更新する。

(a)

$$u_i^+ = u_i^j - \frac{\omega}{\nabla g_i(x)^T C^{-1} \nabla g_i(x)} \{g_i(x)^T C^{-1} \nabla L(x, u^j) - \lambda \cdot g_i(x)\} \quad (18)$$

(b)
$$u_i^{j+1} = \begin{cases} min\{r_i, max(-r_i, u_i^+)\} & ,1 \leq i \leq \bar{m} \text{ のとき} \\ min\{r_i, max(0, u_i^+)\} & ,\bar{m}+1 \leq i \leq m \text{ のとき} \end{cases} \quad (19)$$

3. 次の式に従い、d^j と z_i^j を計算する。

$$\begin{aligned} d^j &= -\frac{1}{\lambda} C^{-1} \nabla L(x, u^{j+1}) \\ z_i^j &= \nabla g_i(x)^T \cdot d^j + g_i(x), i = 1, \cdots, m \end{aligned} \quad (20)$$

$i = 1, \cdots, \bar{m}$ のとき　$(u_i^j = -r_i$　かつ $z_i^j \leq 0)$ または $(u_i^j = r_i$　かつ $z_i^j \geq 0)$
ならば $z_i^j = 0$

$i = \bar{m}+1, \cdots, m$ のとき　$(u_i^j = 0$　かつ $z_i^j \leq 0)$ または $(u_i^j = r_i$　かつ $z_i^j \geq 0)$
ならば $z_i^j = 0$

4. 終了判定

$z_i = 0$ $(i = 1, \cdots, m)$ ならば、終了

それ以外ならば、$j = j + 1$ として、2 に戻る。

ただし、3),4) の収束判定は毎回行なう必要はない。
DSP の制約条件が interval constrain で、かつ、それに対して並列化をおこなった場合、[3] に よると $\bar{\omega}$ は 問題に 依存し それは つぎの通りである。

$$\begin{aligned} \bar{\omega} &= min_{1 \leq i \leq m} \bar{\omega}_i \\ \bar{\omega}_i &= min\{\frac{1}{\theta_i}, \frac{3}{2+\theta_i}\} \end{aligned}$$

$$\theta_i = \frac{2}{\alpha_{ii}} \sum_{j=1, j\neq i}^{m} |\alpha_{ij}|$$

$$\alpha_{ij} = \nabla g_i(x)^T C^{-1} \nabla g_j(x) \tag{21}$$

$$\tag{22}$$

このことから ω の 範囲が $0 \leq \omega \leq \frac{3}{2}$ と 限定される。今回はDSP の制約条件が interval constrain ではないが、ω の値の上限値としてこの値を参照した。

3.3 並列版 Jacobi 法について

並列版 Jacobi 法に関しても 3.2 で 紹介したアルゴリズムは 全く 同じである。ただ、Jacobi 法のアルゴリズムのうち (19),(18) の計算は i について独立して実行できるので、並列処理が可能である。また、SLM に 関しても ペナルティ関数の計算や BFGS 公式等についても 並列処理が可能であるが 今回は時間がなく、実装できなかった。

4 数値計算

今回は Sun Sparc Station 4/370 と Connection Machine CM5 を 用いて 10 変数 8 制約式の 非線形計画問題を ω の 値を 変化させて 表 1 の条件で 計算する。

また、実際には 、次のような 手順に従って 計算時間の計測を行なった。

1. 生成された問題の サイズ に 依存する部分を生成する。
2. コンパイル、実行。(ただし、実行時間計測のために time コマンドを併用。)

プログラムの構成としては、つぎのようになっている。(付録参照)

def.h 定数を定義しているヘッダファイル

auto_ def.h 関数値を代入する関数部分

judge_ tool.h 判定等の関数を記述したファイル

main.c メイン関数を含むファイル

このようにして付録に添付した表 2 から表 7 の結果が得られた。それぞれの表の説明は以下の通りである。

表 2 Sun Sparc Station 4/370 上での SLM-JOR の実行結果 (実行時間と SLM,JOR の各反復数)

表 3 CM5 上での SLM-JOR の実行結果 (実行時間と SLM,JOR の各反復数)

表 4 Sun Sparc Station 4/370 上での SLM-SOR の実行結果 (実行時間と SLM,SOR の各反復数)

表 5 表 2 を元に算出した SLM-JOR の平均反復時間と JOR の実行時間と SLM の実行時間の比

表 6 表 3 を元に算出した並列版 SLM-JOR の平均反復時間と JOR の実行時間と SLM の実行時間の比

表 7 表 4 を元に算出した SLM-SOR の平均反復時間と SOR の実行時間と SLM の実行時間の比

5 考察

今回の実習では、大規模非線形計画問題を解く過程で、主として副問題 SP を解く手段として並列化可能な JOR を用いて、同じ Row-action Algorithm として分類されている SOR と比較検討を行なった。ただ、[6] で 述べられている加速率等は、計算機自体の能力を直接比較できなかったこと、同じソースでも、 CM5 用にコンパイルした場合には、フロントエンドで評価した関数値や勾配の値を各プロセッサに通信する必要が生じること、また、 connection machine 上での プロセッサ割り当てに時間がかかることなどの理由から CM5 用の 方が 実行に時間が かかり 、この規模では 有効性を確かめることができなかった。ただ、指標として前述の表 2 から表 4 で得られたデータから Row-action アルゴリズム の 実行時間 と SLM の 実行時間の 比を算出した結果、表 5 から表 7 のような結果が得られた。これは、 SLM 自体の アルゴリズムは 共通であるので、相対的な Row-action アルゴリズムの性能を比較する目安として算出した。その結果、並列版 JOR(0.1232079),JOR(0.084219),SOR(0.0414799) という順になった。今後の課題としては

1. 今回は問題が密であったが、問題が疎である場合について、並列化がどの程度の効力を発揮するのか、

2. 現在の プログラムは Front end で計算させる部分がまだまだ多く、 Front end と Connection Machine 間での データ通信等に 時間が 多く取られているので、通信時間を 減らすために その並列化されていない部分を いかにして並列計算機で実行すれば良いか、

3. 過緩和係数と 収束回数の解析

また、非線形計画問題と直接の関係はないが、つぎの要因も 重要である。

4. 異機種間での計算結果を どのようにして比較すれば アルゴリズムが 正当に評価できるか

などが 考えられる。

参考文献

[1] Masao Fukushima, Keiichi Takazawa, Shuichi Ohsaki and Toshihide Iabaraki. "Successive Linearization Methods for Large-Scale Nonlinear Programmming Problems".*Japan Journal of Industrial and Applied Mathematics* Vol. 9, No.1 .(1992),pp.117-132

[2] Stavos A.Zenios and Yair Censor. "Massively Parallel Row-Action Algorithms for Some Nonliunear Transportation Problems." *SIAM J.Opt.,* Vol. 1(1991),pp.643-669

[3] Takanobu Sugimoto,Masao Fukushima and Toshihide Ibaraki. "Paralelle Relaxation Meghod for Quadratic Prgramming Problems with Interval Constraints" .*Technical Report 93004, Dept. of Appl. Math.& Phys., Kyoto Univ.*,1993

[4] 山川 栄樹. 大規模非線形計画問題に対する逐次線形化法の適用. 第 28 回 SSOR 予稿集.1993,pp47-52

[5] Yoshihazu Shimazu, Masao Fukushima and Toshihide Ibaraki. "A Succesive Over-Relaxation Method for Quadratic Programming Problems with Interval Constraints" .*J.Oper.Res. Soc. Japan* ,Vol. 36 (1993)

[6] 福嶋 雅夫. 数理計画問題に対する並列アルゴリズム. 第 5 回 RAMP シンポジウム論文集.1993

[7] 茨木 俊秀・福嶋 雅夫 . *FORTRAN 77* 最適化プログラミング .岩波コンピュータサイエンス.1991

6 付録

表 1: 初期値の設定

μ_0	0.1
μ_1	0.25
μ_2	0.75
γ	2.0
λ	2.0

表 2: Sun Sparc Station 4/370 上での SLM-JOR の実行結果

ω の値	system time	user time	実行時間	JOR の反復回数	SLM の反復回数
0.003	3.74	426.09	429.83	380480	30
0.05	0.27	26.17	26.44	22760	30
0.10	0.18	13.09	13.27	11343	30
0.20	0.15	6.52	6.67	5638	30
0.30	0.16	4.21	4.37	3732	30
0.40	0.13	3.20	3.33	2780	30
0.50	0.10	2.54	2.64	2213	30
0.60	0.11	2.11	2.22	1830	30
0.70	0.10	1.85	1.95	1568	30

表 3: CM5 上での並列版 SLM-JOR の実行結果

ω の値	実行時間	JOR の反復回数	SLM の反復回数
0.20	34.601698	5638	30
0.30	23.367738	3732	30
0.40	17.94678	2780	30
0.50	14.365386	2213	30
0.60	11.991102	1830	30
0.68	10.920948	1602	30
0.70	10.520028	1568	30
0.73	9.988242	1483	30
0.74	10.001523	1473	30
0.75	13.668573	1511	30

表 4: Sun Sparc Station 4/370 上での SLM-SOR の 実行結果

ω の値	system time	user time	実行時間	SOR の反復回数	SLM の 反復回数
0.005	1.73	326.77	328.5	284445	23
0.03	5.39	549.73	555.12	474593	23
0.05	0.10	31.64	31.74	27738	23
0.10	0.21	15.45	15.66	13476	23
0.20	1.66	8.91	10.57	6349	23
0.30	0.16	4.58	4.74	3679	23
0.40	0.15	3.34	3.49	2799	23
0.50	0.08	2.48	2.56	2098	23
0.60	0.15	1.89	2.04	1630	23
0.70	0.10	1.55	1.65	1297	23
0.80	0.08	1.25	1.33	1034	23
0.90	0.09	1.05	1.14	836	23
1.00	0.11	0.70	0.81	552	24
1.10	0.05	0.71	0.76	552	24
1.20	0.09	0.58	0.67	427	23
1.30	0.56	0.47	1.03	340	23
1.40	0.07	0.56	0.63	408	24
1.50	0.08	0.64	0.72	497	23
1.60	0.09	0.84	0.93	660	24
1.70	0.08	1.11	1.19	898	25
1.80	0.08	1.68	1.76	1401	24
1.90	0.08	3.41	3.49	2929	24
1.95	0.16	6.80	6.96	5936	25

表 5: SLM-JOR の 平均反復時間

SLM 1回の 反復にかかる時間	0.0120329
JOR 1回の 反復にかかる時間	0.0010134
JOR の実行時間 / SLM の 実行時間	0.084219

表 6: 並列版 SLM-JOR の 平均反復時間

SLM 1回の 反復にかかる時間	0.0457265
JOR 1回の 反復にかかる時間	0.0058939
JOR の実行時間 / SLM の 実行時間	0.1232079

表 7: SLM-SOR の 平均反復時間

SLM 1回の 反復にかかる時間	0.0277604
SOR 1回の 反復にかかる時間	0.0011515
SOR の実行時間 / SLM の 実行時間	0.0414799

```
/*          SLM-JOR   FOR CM2 /5      */
#include<stdio.h>
#include<math.h>
#include<cscomm.h>
#include<cstimer.h>
#include"def.h"


    shape [M+1][N]plane;
static double:plane Sc_inv,Sd,Sc,SNg,oldG_L,SG_L,U,r,denom,z,tmpz,gvalue,oldNg,mask1,mask2;
static double f_eps,tmpp,dcd,lmda,plty,Omega,Ng[M+1][N+1],g[M+1];

#include"judge_tool.h"
#include"./problem.h"
#include"auto_def.h"

int flag(tmpz,x)
 double:plane tmpz,x;
{
double:plane Sdcd,tmp;
    double DCD,Aplty;
    int i;
    with(plane){
            tmp = (tmpz < 0) ? (-tmpz) : tmpz;
            tmp = spread(tmp,0,CMC_combiner_max);
            if(absolute([0][0]tmp) < f_eps){
                Aplty = Ap(x,Sd);
                tmpp = plty - Aplty;
                Sdcd = 0.0;
                Sdcd = Sd*Sc*Sd;
                Sdcd = spread(Sdcd,1,CMC_combiner_add);
                dcd = [1][1]Sdcd;
                DCD = dcd *lmda*DELTA;
                if(tmpp >= DCD){
                        return 1;
                }
                else{
                        f_eps = f_eps*SIGMA;
                        return 0;
                }
            }
            else{
                return 0;
            }
    }
}


void JOR(x)
 double:plane x;
{
 double:plane tmpU,tmpU1,tmpU2,enom;
      int k,j,i;
      with(plane){
              for(k = 1;k < 100000;k++){
                  enom = 0.0;
                  enom = SNg*SG_L*Sc_inv;
                  enom = spread(enom,1,CMC_combiner_add);
                  enom += -lmda*gvalue;
                  [0][0]enom = 0.0;
                  enom = copy_spread(&enom,1,0);
                  tmpU = 0.0;
                  tmpU = U - Omega*enom*denom;
                  tmpU1 = 0.0;
```

```
                tmpU1 = (tmpU > -r) ? tmpU  :(-r);
                tmpU1 = (tmpU1 < r) ? tmpU1 : r;
                tmpU2 = 0.0;
                tmpU2 = (tmpU > 0.0) ? tmpU  :0.0;
                tmpU2 = (tmpU2 < r) ? tmpU2 : r;
                U = 0.0;
                U = mask1*tmpU1 + mask2*tmpU2;
                [0][0]U = 1.0;
                U = copy_spread(&U,1,0);
                SG_L = 0.0;
                SG_L = U*SNg;
                SG_L = spread(SG_L,0,CMC_combiner_add);
                Sd = 0.0;
                Sd = -SG_L*Sc_inv /lmda;
                Sd = copy_spread(&Sd,0,1);
                z = 0.0;
                z = SNg*Sd;
                z = spread(z,1,CMC_combiner_add);
                z += gvalue;
                [0][0]z = 0.0;
                z = copy_spread(&z,1,0);
                tmpz = judge(U,z,r);
                if(flag(tmpz,x) == 1){
                        break;
                }
            }
        }
        printf("JOR Iteration : %d \n",k);
}

int SLM_judge(x)
double:plane x;
{

    if(tmpp <= EPS1*(absolute(p(x)) + 1.0)){
            return 1;
    }
    else{
            return 0;
    }
}

void BFGS()
{
 double:plane y,Sdy,y_t;
      double theta,dy;
      int i;
      with(plane){
            y = 0.0;
            y = SG_L - oldG_L;
            Sdy = 0.0;
            Sdy = Sd*y;
            Sdy = spread(Sdy,1,CMC_combiner_add);
            dy = [0][0]Sdy;
    }
    if(dy >= 0.2*dcd){
            theta = 1.0;
    }
    else{
            theta = 0.8*dcd /(dcd - dy);
    }
    with(plane){
            y_t = theta*y + (1.0 - theta)*Sc*Sd;
            Sdy = 0.0;
```

```
            Sdy = Sd*y_t;
            Sdy  = spread(Sdy,1,CMC_combiner_add);
            dy = [0][0]Sdy;
            Sc = Sc + y_t*y_t/dy - Sc*Sc*Sd*Sd/dcd;
            Sc = (Sc > 0.001) ? Sc : 0.001;
            Sc = (Sc < 100.0) ? Sc : 100.0;

            for(i = 0;i < N;i++){
                [0][i]Sc_inv = 1.0/[0][i]Sc;
            }
            Sc_inv = copy_spread(&Sc_inv,0,0);
    }
}

int UpdateR(r)
 double:plane r;
{
    int tmp,i;
            tmp = 0;
    with(plane){
            r = (-r < U && U < r) ? r : 2.0*r;
            where(-r < U && U < r){
                tmp ++;
            }
            for(i = 0;i < N;i++){
                [0][i]r = 1.0;
            }
            if(tmp == M*N){
                return 1;
            }
            else{
                return 0;
            }
    }
}

void show_value(x,i)
double:plane x;
int i;
{
    int j,i1;
    printf("\n\nf(x) = %15.13f\n",f(x));
    for(j = 0 ; j < N/2; j++){
            printf("\nx%d = %15.13f,x%d = %15.13f\n",2*j,[0][2*j]x,2*j+1,[0][2*j+1]x);
    }
    for(j = 0 ; j < M/2; j++){
            printf("g%d(x) = %15.13f,g%d(x) = %15.13f\n",2*j,g[2*j],2*j+1,g[2*j+1]);
    }
    for(j = 0 ; j <= M; j++){
            for(i1 = 1 ; i1 < N/2; i1++){
                printf("dg%d(x) /d x%d = %12.10f,dg%d(x) /d x%d = %12.10f\n",j,2*i1-1,[j][2*i1-1]SNg,j,2*i1,[j][2*i1]SNg);
            }
    }


    printf("g1(x) = %15.13f,g2(x) = %15.13f\n",g1(x),g2(x));
    printf("dg1(x) /d x1 = %12.10f,dg1(x) /d x2 = %12.10f\n",Ng[1][1],Ng[1][2]);
    printf("dg2(x) /d x1 = %12.10f,dg2(x) /d x2 = %12.10f\n",Ng[2][1],Ng[2][2]);
    printf("u1 = %15.13f,u2 = %15.13f\n",[1][0]U,[2][0]U);
    printf("*************************************************************\n");
    printf("*                      SLM Iteration : %d                   *\n",i);
    printf("*************************************************************\n");
    printf("\n\n");
}
```

```
void start()
{
    printf("*************************************************************\n");
    printf("*                    SLM--JOR  START                        *\n");
    printf("*************************************************************\n");
}


int main(){
    int l,j,slmflag,i,count;
 double:plane  x,newx,tmp,tmp_denom;
    double q;
    lmda = 1.0;
    slmflag = 0;
    f_eps = EPS;
    printf("omega = ");
    scanf("%lf",&Omega);
    printf("\nomega = %f",Omega);
    with(plane){
            CMC_timer_clear(0);
            CMC_timer_start(0);
            Sc = 1.0;
            mask1 = 0.0;
            for(j = 0;j < EM;j++){
                [j][0]mask1 = 1.0;
            }
            mask1 = copy_spread(&mask1,1,0);
            mask2 = Sc - mask1;

            for(i = 0;i < N;i++){
                [0][i]Sc_inv = 1.0/[0][i]Sc;
            }
            Sc_inv = copy_spread(&Sc_inv,0,0);

            [0][0]x = 2.0;
            [0][1]x = 3.0;
            [0][2]x = 5.0;
            [0][3]x = 5.0;
            [0][4]x = 1.0;
            [0][5]x = 2.0;
            [0][6]x = 7.0;
            [0][7]x = 3.0;
            [0][8]x = 6.0;
            [0][9]x = 10.0;
            x = copy_spread(&x,0,0);
            r = 20.0;
            U = 0.0;
            [0][0]r = 1.0;
            [0][0]U = 1.0;
            r = copy_spread(&r,1,0);
            U = copy_spread(&U,1,0);

            Cal_N_L(x);
            for(j = 0;j < N;j++){
                [0][j]SG_L = Ng[0][j+1];
            }
            SG_L = copy_spread(&SG_L,0,0);

            start();
            for(l = 1;l <= 500;l++){

                tmp_denom = 0.0;
                tmp_denom = SNg*SNg*Sc_inv;
```

```
                tmp_denom = spread(tmp_denom,1,CMC_combiner_add);

                denom = 0.0;
                for(j = 1; j <= M ; j++){
                        [j][0]denom = 1.0/[j][0]tmp_denom;
                }
                denom = copy_spread(&denom,1,0);


                JOR(x);
                if(UpdateR(r) ==  1){
                        plty  = p(x);
                }
                else{
                        if(SLM_judge(x) == 1.0){
                            slmflag++;
                        }

                        else{
                        slmflag = 0;
                }
                        if(slmflag == 2){
                            slmflag = 0;
                            break;
                        }
                        q = delta_p(x,Sd)/tmpp;
                        if(q < MYU0 ){
                            lmda = GAMMA*lmda;
                        }
                        else{
                            x = x + Sd;
                            Cal_N_L(x);
                            BFGS();
                            if(q <= MYU1){
                                    lmda = GAMMA*lmda;
                            }
                            if(q > MYU2){
                                    lmda = lmda/GAMMA;
                            }
                            else{
                                    lmda = lmda;
                            }
                        }
                }
                show_value(x,l);
            }
            show_value(x,l);
    }
    CMC_timer_stop(0);
    printf("busy time:%f,elasped time:%f\n",CM_timer_read_cm_busy(0),CM_timer_read_elapsed(0));
    return 0;
}
```

```
/*    main.c   FOR   SLM-JOR    */
#include<stdio.h>
#include<math.h>
#include"def.h"
#include"judge_tool.h"


static double lmda,tmpp,f_eps,dcd,plty,Aplty;
static double d[N+1],u[M+1],c[N+1],r[M+1],G_L[N+1],g[M+1],Ng[M+1][N+1];
double tmpu[M+1],z[M+1],oldG_L[N+1],tmpg[M+1],oldNg[M+1][N+1];
static double denom[M+1];

#include"problem1.h"
#include"auto_def.h"

int flag(tmpz,x)
double tmpz[],x[];
{
    double tmp,DCD;
    int i;
    tmp = absolute(tmpz[1]);
    for(i=2;i<=M;i++){
            if(tmp <= absolute(tmpz[i])){
                tmp = absolute(tmpz[i]);
            }
    }

    if(tmp < f_eps){
                    Aplty = Ap(x,d);

            tmpp = plty - Aplty;
            dcd = 0.0;
            for(i = 1; i <= N;i++){
                dcd = dcd + d[i]*c[i]*d[i];
            }
            DCD = dcd*lmda*DELTA;
            if(tmpp >= DCD){
                return 1;
            }
            else{
                f_eps = f_eps*SIGMA;
                return 0;
            }
    }
    else{
            return 0;
    }
}

void JOR(x)
double x[];
{
    double temp,enom;
    double tmpz[M+1];

    int i1,i2,i3,i4,i5,i6,i7,j1,j2,j3,j4,j5,j6,fflag,k;

    for(k=1;k<=100000;k++){

            for(i4 = 1;i4 <= M;i4++){
                enom = -lmda*g[i4];
                for(j2 = 1;j2 <= N;j2++){
                        enom += Ng[i4][j2]*G_L[j2]/c[j2];
                }
```

```
                tmpu[i4] = u[i4] - Omega*enom /denom[i4];
                if(i4 <= EM){
                        tmpu[i4] = lt_y(r[i4],gt_y(-r[i4],tmpu[i4]));
                }
                else{
                        tmpu[i4] = lt_y(r[i4],gt_y(0.0,tmpu[i4]));
                }
                u[i4] = tmpu[i4];
        }
        for(j3 = 1;j3 <= N;j3++){
                G_L[j3] = Ng[0][j3];
                for(i7 = 1;i7 <= M;i7++){
                        G_L[j3] += u[i7]*Ng[i7][j3];
                }
        }
        for(j4 = 1;j4 <= N;j4++){
                d[j4] = - G_L[j4] /(c[j4]*lmda);
        }

        for(i5 = 1;i5 <= M;i5++){
                z[i5] = g[i5];
                for(j5 = 1;j5 <= N;j5++){
                        z[i5] += Ng[i5][j5]*d[j5];
                }
                tmpz[i5] = judge(u[i5],z[i5],r[i5],i5);
        }
        if(flag(tmpz,x) == 1){
                break;
        }
    }
    printf("JOR Iteration : %d \n",k);
}

int SLM_judge(x)
double x[];
{
    if(tmpp <= EPS1*(absolute(p(x)) + 1.0)){
            return 1;
    }
    else{
            return 0;
    }
}

void BFGS()
{
    double y[N+1],theta,dy,y_t[N+1];
    int i,i1,i2,i3;

    for(i = 1;i <= N;i++){
            y[i] = 0.0;
            y[i] = G_L[i] - oldG_L[i];
    }
    dy = 0.0;
    for(i1 = 1;i1 <= N;i1++){
            dy += d[i1]*y[i1];
    }
    if(dy >= 0.2*dcd){
            theta = 1.0;
    }
    else{
            theta = 0.8*dcd /(dcd - dy);
    }
    for(i2 = 1;i2 <= N;i2++){
```

```
            y_t[i2] = theta*y[i2] + (1.0 - theta)*c[i2]*d[i2];
    }
    dy = 0.0;
    for(i3 = 1;i3 <= N;i3++){
            dy += d[i3]*y_t[i3];
    }
    for(i3 = 1;i3 <= N;i3++){

            c[i3] = c[i3] + sq(y_t[i3])/dy - sq(c[i3])*sq(d[i3])/dcd;
            c[i3] = gt_y(0.001,lt_y(100.0,c[i3]));
    }
}

int UpdateR(r)
double r[];
{
    int tmp,j;
    tmp = 1;
    for(j = 1;j <= M;j++){
            if(absolute(u[j]) < r[j]){
                tmp ++;
            }
            else{
                r[j] = BETA*r[j];
            }
    }
    if(tmp == M){
            return 1;
    }
    else{
            return 0;
    }
}

void Show_value(x,i)
double x[];
int i;
{
    int k;
    printf("F(x) = %15.13f\n",f(x));
   for(k = 1;k <= N; k++){
            printf("\nx[%d] = %15.13f\n",k,x[k]);
    }
   for(k = 1;k <= N; k++){
            printf("g%d(x) = %15.13f\n",k,g[k]);
    }
   for(k = 1;k <= N; k++){
            printf("dg%d(x)/d x1 = %12.10f,dg%d(x)/d x2 = %12.10f\n",k,Ng[k][1],k,Ng[k][2]);
            printf("dg%d(x)/d x3 = %12.10f,dg%d(x)/d x4 = %12.10f\n",k,Ng[k][3],k,Ng[k][4]);
            printf("dg%d(x)/d x5 = %12.10f,dg%d(x)/d x6 = %12.10f\n",k,Ng[k][5],k,Ng[k][6]);
            printf("dg%d(x)/d x7 = %12.10f,dg%d(x)/d x8 = %12.10f\n",k,Ng[k][7],k,Ng[k][8]);
    }
    printf("u1 = %15.13f,u2 = %15.13f\n",u[1],u[2]);
    printf("u3 = %15.13f,u4 = %15.13f\n",u[3],u[4]);
    printf("u5 = %15.13f,u6 = %15.13f\n",u[5],u[6]);
    printf("u7 = %15.13f,u8 = %15.13f\n",u[7],u[8]);
    printf("************************************************************\n");
    printf("*                    SLM Iteration : %d                    *\n",i);
    printf("************************************************************\n");
    printf("\n\n");
}

void start()
{
```

```
    printf("*************************************************************\n");
    printf("*                        SLM-JOR  START                     *\n");
    printf("*************************************************************\n");
}

int main(){
    int i,i1,i2,i3,i4,i5,i6,j1,l,slmflag;
    double x[N+1],q,newx[N+1],tmp[N+1];

    for(i6 = 1 ;i6 <= N;i6++){
            c[i6] = 1.0;
    }
    x[1] = 2.0;
    x[2] = 3.0;
    x[3] = 5.0;
    x[4] = 5.0;
    x[5] = 1.0;
    x[6] = 2.0;
    x[7] = 7.0;
    x[8] = 3.0;
    x[9] = 6.0;
    x[10] = 10.0;


    lmda = 1.0;
    slmflag = 0;
    f_eps = EPS;
    for(i5 = 1;i5 <= M;i5++){
            r[i5] = R1;
            u[i5] =  0.0;
    }
    Cal_N_L(x);

    for(j1 = 1;j1 <= N;j1++){
            G_L[j1] = Ng[0][j1];
    }

    start();
    for(l=1;l <= 500;l++){
            /* ^[$@J,Jl$N^[(J ^[$@7W;;^[(J */
            for(i2 = 1;i2 <= M;i2++){
                denom[i2] = 0.0;
                for(j1 = 1;j1 <= N;j1++){
                        denom[i2] = denom[i2] + sq(Ng[i2][j1])/c[j1];
                }
            }

            JOR(x);

            if(UpdateR(r) == 1){
                plty = p(x);
            }
            else{
                if(SLM_judge(x) == 1){
                        slmflag++;
                }
                else{
                        slmflag = 0;
                }
                if(slmflag == 2){
                        slmflag = 0;
                        break;
                }
```

```
                q = delta_p(x,d)/tmpp;

                if(q < MYU0){
                        lmda = GAMMA*lmda;
                }
                else{
                        for(i2 = 1;i2 <= N;i2++){
                            x[i2] = x[i2] + d[i2];
                        }
                        Cal_N_L(x);
                         BFGS();
                        if(q < MYU1){
                            lmda = GAMMA*lmda;
                        }
                        else if(q > MYU2){
                            lmda = lmda/GAMMA;
                        }
                        else {
                            lmda = lmda;
                        }
                }
            }
            Show_value(x,l);
    }
    Show_value(x,l);
    return 0;
}
```

def.h

```
#define    MYU0   0.1
#define    MYU1   0.25
#define    MYU2   0.75
#define    GAMMA   2.0
#define    EPS      0.00001
#define   sq(x)    ((x)*(x))
#define   tr(x)    ((x)*(x)*(x))
#define   R1  20.0
#define   R2  20.0
#define   Omega 0.9
#define   DELTA 0.1
#define   SIGMA 0.85
#define   EPS1   0.00001
#define   BETA   2.0
/* #define   EM 2 */
#define   EM 0
#define   M 8
#define   N 10
```

auto_def.h

```
void Cal_N_L(x)
double x[];
{
    int i,j1,j,i1;
    for(j = 1;j <= N;j++){
            for(i = 1;i <= M;i++){
                oldNg[i][j] = Ng[i][j];
            }
            oldG_L[j] = G_L[j];
    }
    Ng[0][1] = dif_f_x1(x);
    Ng[0][2] = dif_f_x2(x);
    Ng[0][3] = dif_f_x3(x);
    Ng[0][4] = dif_f_x4(x);
    Ng[0][5] = dif_f_x5(x);
    Ng[0][6] = dif_f_x6(x);
    Ng[0][7] = dif_f_x7(x);
    Ng[0][8] = dif_f_x8(x);
    Ng[0][9] = dif_f_x9(x);
    Ng[0][10] = dif_f_x10(x);
    Ng[1][1] = dif_g1_x1(x);
    Ng[1][2] = dif_g1_x2(x);
    Ng[1][3] = dif_g1_x3(x);
    Ng[1][4] = dif_g1_x4(x);
    Ng[1][5] = dif_g1_x5(x);
    Ng[1][6] = dif_g1_x6(x);
    Ng[1][7] = dif_g1_x7(x);
    Ng[1][8] = dif_g1_x8(x);
    Ng[1][9] = dif_g1_x9(x);
    Ng[1][10] = dif_g1_x10(x);
    Ng[2][1] = dif_g2_x1(x);
    Ng[2][2] = dif_g2_x2(x);
    Ng[2][3] = dif_g2_x3(x);
    Ng[2][4] = dif_g2_x4(x);
    Ng[2][5] = dif_g2_x5(x);
    Ng[2][6] = dif_g2_x6(x);
    Ng[2][7] = dif_g2_x7(x);
    Ng[2][8] = dif_g2_x8(x);
    Ng[2][9] = dif_g2_x9(x);
    Ng[2][10] = dif_g2_x10(x);
    Ng[3][1] = dif_g3_x1(x);
    Ng[3][2] = dif_g3_x2(x);
    Ng[3][3] = dif_g3_x3(x);
    Ng[3][4] = dif_g3_x4(x);
    Ng[3][5] = dif_g3_x5(x);
    Ng[3][6] = dif_g3_x6(x);
    Ng[3][7] = dif_g3_x7(x);
    Ng[3][8] = dif_g3_x8(x);
    Ng[3][9] = dif_g3_x9(x);
    Ng[3][10] = dif_g3_x10(x);
    Ng[4][1] = dif_g4_x1(x);
    Ng[4][2] = dif_g4_x2(x);
    Ng[4][3] = dif_g4_x3(x);
    Ng[4][4] = dif_g4_x4(x);
    Ng[4][5] = dif_g4_x5(x);
    Ng[4][6] = dif_g4_x6(x);
    Ng[4][7] = dif_g4_x7(x);
    Ng[4][8] = dif_g4_x8(x);
    Ng[4][9] = dif_g4_x9(x);
    Ng[4][10] = dif_g4_x10(x);
    Ng[5][1] = dif_g5_x1(x);
    Ng[5][2] = dif_g5_x2(x);
    Ng[5][3] = dif_g5_x3(x);
    Ng[5][4] = dif_g5_x4(x);
```

auto_def.h

```
    Ng[5][5] = dif_g5_x5(x);
    Ng[5][6] = dif_g5_x6(x);
    Ng[5][7] = dif_g5_x7(x);
    Ng[5][8] = dif_g5_x8(x);
    Ng[5][9] = dif_g5_x9(x);
    Ng[5][10] = dif_g5_x10(x);
    Ng[6][1] = dif_g6_x1(x);
    Ng[6][2] = dif_g6_x2(x);
    Ng[6][3] = dif_g6_x3(x);
    Ng[6][4] = dif_g6_x4(x);
    Ng[6][5] = dif_g6_x5(x);
    Ng[6][6] = dif_g6_x6(x);
    Ng[6][7] = dif_g6_x7(x);
    Ng[6][8] = dif_g6_x8(x);
    Ng[6][9] = dif_g6_x9(x);
    Ng[6][10] = dif_g6_x10(x);
    Ng[7][1] = dif_g7_x1(x);
    Ng[7][2] = dif_g7_x2(x);
    Ng[7][3] = dif_g7_x3(x);
    Ng[7][4] = dif_g7_x4(x);
    Ng[7][5] = dif_g7_x5(x);
    Ng[7][6] = dif_g7_x6(x);
    Ng[7][7] = dif_g7_x7(x);
    Ng[7][8] = dif_g7_x8(x);
    Ng[7][9] = dif_g7_x9(x);
    Ng[7][10] = dif_g7_x10(x);
    Ng[8][1] = dif_g8_x1(x);
    Ng[8][2] = dif_g8_x2(x);
    Ng[8][3] = dif_g8_x3(x);
    Ng[8][4] = dif_g8_x4(x);
    Ng[8][5] = dif_g8_x5(x);
    Ng[8][6] = dif_g8_x6(x);
    Ng[8][7] = dif_g8_x7(x);
    Ng[8][8] = dif_g8_x8(x);
    Ng[8][9] = dif_g8_x9(x);
    Ng[8][10] = dif_g8_x10(x);
    g[1] = g1(x);
    g[2] = g2(x);
    g[3] = g3(x);
    g[4] = g4(x);
    g[5] = g5(x);
    g[6] = g6(x);
    g[7] = g7(x);
    g[8] = g8(x);

    for(j1 = 1;j1 <= N;j1++){
            G_L[j1] = Ng[0][j1];
            for(i1 = 1;i1 <= M;i1++){
                G_L[j1] += u[i1]*Ng[i1][j1];

            }

    }
    plty = p(x);


}
```

```
#include<stdio.h>
#include<math.h>
#include"def.h"
#include"judge_tool.h"


static double lmda,tmpp,f_eps,dcd,plty,Aplty;
static double d[N+1],u[M+1],c[N+1],r[M+1],G_L[N+1],g[M+1],Ng[M+1][N+1];
double tmpu[M+1],z[M+1],oldG_L[N+1],tmpg[M+1],oldNg[M+1][N+1];

#include"problem.h"
#include"auto_def.h"


int flag(tmpz,x)
double tmpz[],x[];
{
    double tmp,DCD;
    int i;
    tmp = absolute(tmpz[1]);
    for(i=2;i<=M;i++){
        if(tmp <= absolute(tmpz[i])){
            tmp = absolute(tmpz[i]);
        }
    }

    if(tmp < f_eps){
            Aplty = Ap(x,d);
        tmpp = plty - Aplty;


        dcd = 0.0;
        for(i = 1; i <= N;i++){
            dcd = dcd + d[i]*c[i]*d[i];
        }
        DCD = dcd*lmda*DELTA;

        if(tmpp >= DCD){
            return 1;
        }
        else{
            f_eps = f_eps*SIGMA;
            return 0;
        }
    }
    else{
        return 0;
    }
}


void SOR(x)
double x[];
{
    double temp,enom;
    double tmpz[M+1];

    static double denom[M+1];
    int i1,i2,i3,i4,i5,i6,j1,j2,j3,j4,j5,j6,fflag,k;

    for(i2 = 1;i2 <= M;i2++){
        denom[i2] = 0.0;
        for(j1 = 1;j1 <= N;j1++){
            denom[i2] = denom[i2] + sq(Ng[i2][j1])/c[j1];
```

```
            }
        }

        for(k=1;k<=1000;k++){

            for(i4 = 1;i4 <= M;i4++){
                enom = -lmda*g[i4];
                for(j2 = 1;j2 <= N;j2++){
                    enom = enom + Ng[i4][j2]*G_L[j2]/c[j2];
                }
                tmpu[i4] = u[i4] - Omega*enom/denom[i4];

                  if(i4 <= EM){
                  tmpu[i4] = lt_y(r[i4],gt_y(-r[i4],tmpu[i4]));
                  }
                  else{
                  tmpu[i4] = lt_y(r[i4],gt_y(0.0,tmpu[i4]));
                  }

                for(j3 = 1;j3 <= N;j3++){
                    G_L[j3] += (tmpu[i4] - u[i4])*Ng[i4][j3];
                }
                u[i4] = tmpu[i4];
            }


            for(j4 = 1;j4 <= N;j4++){
                d[j4] = - G_L[j4]/(c[j4]*lmda);
            }

            for(i5 = 1;i5 <= M;i5++){
                z[i5] = g[i5];
                for(j5 = 1;j5 <= N;j5++){
                    z[i5] = z[i5] + Ng[i5][j5]*d[j5];
                }

                tmpz[i5] = judge(u[i5],z[i5],r[i5],i5);
            }
            if(flag(tmpz,x) == 1){
                break;
            }
        }
        printf("SOR Iteration : %d \n",k);
}

int SLM_judge(x)
double x[];
{
        if(tmpp <= EPS1*(absolute(p(x)) + 1.0)){
            return 1;
        }
        else{
            return 0;
        }
}

/*
 void BFGS(void)
 */
void BFGS()
{
        double y[N+1],theta,dy,y_t[N+1];
        int i,i1,i2,i3;
```

```
    for(i = 1;i <= N;i++){
            y[i] = 0.0;
            y[i] = G_L[i] - oldG_L[i];

    }
    dy = 0.0;
    for(i1 = 1;i1 <= N;i1++){
            dy += d[i1]*y[i1];
    }
    if(dy >= 0.2*dcd){
            theta = 1.0;
    }
    else{
            theta = 0.8*dcd/(dcd - dy);
    }

    for(i2 = 1;i2 <= N;i2++){
            y_t[i2] = theta*y[i2] + (1.0 - theta)*c[i2]*d[i2];

    }
    dy = 0.0;
    for(i3 = 1;i3 <= N;i3++){
            dy += d[i3]*y_t[i3];

    }
    for(i3 = 1;i3 <= N;i3++){

            c[i3] = c[i3] + sq(y_t[i3])/dy - sq(c[i3])*sq(d[i3])/dcd;
            c[i3] = gt_y(0.001,lt_y(100.0,c[i3]));
    }
}


/*
 int UpdateR(double r[])
 */

int UpdateR(r)
double r[];
{
    int tmp,j;
    tmp = 1;
    for(j = 1;j <= M;j++){
            if(absolute(u[j]) < r[j]){
                tmp ++;
            }
            else{
                r[j] = BETA*r[j];
            }
    }
    if(tmp == M){
            return 1;
    }
    else{
            return 0;
    }
}

void Show_value(x,i)
double x[];
int i;
{
    printf("\n\nf(x) = %15.13f\n",f(x));
```

```
    printf("*************************************************************\n");
    printf("*                    SLM Iteration : %d                     *\n",i);
    printf("*************************************************************\n");
    printf("\n\n");
}

void start()
{
    printf("*************************************************************\n");
    printf("*                    SLM-SOR  START                         *\n");
    printf("*************************************************************\n");
}



int main(){
    int i,i1,i2,i3,i4,i5,i6,j1,l,slmflag;
    double x[N+1],q,newx[N+1],tmp[N+1];

    for(i6 = 1 ;i6 <= N;i6++){
            c[i6] = 1.0;
            x[i6] = 2.0;

    }

    lmda = 1.0;
    slmflag = 0;
    f_eps = EPS;
    for(i5 = 1;i5 <= M;i5++){
            r[i5] = R1;
            u[i5] =  0.0;
    }
    Cal_N_L(x);

    for(j1 = 1;j1 <= N;j1++){
            G_L[j1] = Ng[0][j1];
    }

    start();
    for(l=1;l <= 500;l++){
            SOR(x);

            if(UpdateR(r) == 1){
                plty = p(x);
            }
            else{

                if(SLM_judge(x) == 1){
                        slmflag++;
                }
                else{
                        slmflag = 0;
                }
                if(slmflag == 2){
                        break;
                }

                q = delta_p(x,d)/tmpp;

                if(q < MYU0){
                        lmda = GAMMA*lmda;
                }
                else{
                        for(i2 = 1;i2 <= N;i2++){
```

```
                    x[i2] = x[i2] + d[i2];
                }
                Cal_N_L(x);
                 BFGS();
                if(q < MYU1){
                    lmda = GAMMA*lmda;
                }
                else if(q > MYU2){
                    lmda = lmda/GAMMA;
                }
                else {
                    lmda = lmda;
                }
            }
        }
    }
    Show_value(x,l);
    return 0;
}
```