情報工学実験１数理計画法

学生番号 4617043 神保光洋

2018 年 11 月 3 日

1　実験目的

本実験では非線形計画問題に対する基本的な解法である最急降下法、ニュートン法、準ニュートン法の実装とさまざまな問題への適用を通し、アルゴリズムの特性の把握の大切さを学ぶことを目的とする

2　アルゴリズムと数学的理論についての概要

2.1　無制約最小化問題に対する基礎理論

無制約最小化問題とは、n 変数関数 f: R^2 -¿ R に対して定義される以下の方法である。最小化 $\{f(x)|x = (x_1, x_2, x_3, ..., x_n)\} \in R^n$ 略して $\min x \in R^n f(x)$

2.1.1　諸定義

■関数の勾配ベクトル、ヘッセ行列、テイラー展開　n 変数関数 $f : R^n \to R$ に対し、勾配ベクトル $\nabla f(x)$ とヘッセ行列 $\nabla^2 f(x)$ はそれぞれ以下のように定義されるベクトルと行列である。

$$\nabla f(x) = (\ 2\ )\ 2$$

勾配ベクトルとヘッセ行列はそれぞれ一変数関数 $f(x)$ の微分係数と 2 階微分係数 $f'(x)$ を n 変数関数にしたものである。なお、微分できない一変数関数もあるのと同様に、勾配ベクトルやヘッセ行列が定義できない関数 (f(x) = $\max(x^T x, 0)$) になることに注意する。$f(x)$ が 2 階微分可能であれば、勾配ベクトルとヘッセ行列を用いて $x = a$ において、以下の式で $f(x)$ を 2 次近似することができる ($x = a$ に置ける周りのテイラー展開を 2 次の項まで求めたもの)。
$f(a + d) = f(a) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(a) d +$ 残差
行列の正定値性 $n \times n$ 対称行列 X が半正定値であるとは

$$d^T X d \geq 0 \forall d \in R^n$$

$n \times n$ 対称行列 X が正定値であるとは
$d^T X d \geq 0 \forall d \in R^n$,$d \neq 0$
$n \times n$ 対称行列 X が半正定値 (正定値) であることと、X の固有値が全て非負 (正) であることは同値である。

2.1.2 最適解と最適性条件

■最適解　問題 $\min x \in R^n f(x)$ において、目的関数 f を最小にする $x^* \in R^n$, すなわち

$$f(x^*)leqf(x)\forall \in R^n$$

を満たす x^* を最適解と呼ぶ。最適解は次に定義する局所的最適解と区別するために大域的最適解と呼ぶことも多い。

■局所最適解　x^* を中心とする (小さな) 球の中で目的関数 f を最小にする $x^* \in R^n$:
ある $\epsilon > 0$ が存在して

$f(x^*) \leq f(x)\ \forall x \in R^n, ||x^* - x|| < \epsilon$

を局所的最適解と呼ぶ。大域的最適解は局所的最適解になっているが、逆は成り立たない

一般に非線形最適化問題において大域的最適解を求めることは大変難しく、多くの場合は局所的最適解を求めることを目指す。局所最適解については以下の最適性条件が成り立つ。

■最適性条件

最適性の必要条件　$x^* \in R^n$ を局所的最適解とする。

- f が連続的微分可能 $\rightarrow \nabla f(x^*) = 0$ (1 次の必要条件)。
- f が 2 階連続的微分可能 $\rightarrow \nabla^2 f(x^*)$ 半正定値行列 (2 次の必要条件)。

2 **次の十分条件**　関数 f が 2 階連続的微分可能とする。$x^* \in R^n$ が以下の条件を満たすならば局所的最適解である。

$\nabla f(x^*) = 0$
$\nabla^2 f(x^*)$ が正定値行列

なお、1 次の必要条件 $\nabla f(^*) = 0$ を満たす x^* を停留点と呼ぶ。

■凸関数　関数 $f : R^n \rightarrow R$ が以下の条件を満たす時、凸関数であるという。

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \forall x, y \in R^n$$

f が 2 階連続的微分可能であるとき、以下の事実が成り立つ。

f が凸関数 $\iff \nabla^2 f(x)$ が半正定値行列 $\forall x \in R^n$
さらに、凸関数は次の「局所最適性＝大域的最適性」を満たす。

f が凸関数のとき、x^* が f の局所的最適解 $\Rightarrow x^*$ は f の大域的最適解これより、微分可能な凸関数に対しては、$\nabla f(x) = 0$ を満たす x^* を求めれば、その x^* は最適解であることが保証される。このように、大域的最適解が必ず求まる関数のクラスとして、凸関数は重要なクラスである

2.1.3 反復法

適当な初期点 $x_0 \in R^n$ からスタートし、以下の更新式で次々と点 $x_1, x_2, \ldots$ を生成するアルゴリズムを反復法という。

$$x_{k+1} = x_k + \alpha_k d_k$$

この式に置ける d_k を探索方向、α_k をステップ幅という。探索方向は次で定義する降下方向になっているものを選び、ステップ幅は直線探索を利用して求めることが多い。

■降下方向 $x \in R^n$ における降下方向とは以下の条件を満たす $d \in R^n$ であり、目的関数値が減少する方向である。$\nabla f(x)^T d < 0$

■最急降下方向 $\nabla f(x) \neq 0$ であれば、$-\nabla f(x)$ は自明な降下方向である。これを最急降下方向という。

■直線探索を用いた反復法

ステップ 0: 初期点 x_0 を選び、$k := 0$ とする
ステップ 1: 停止条件が満たされていれば終了する
ステップ 2: 探索方向 d_k を定める
ステップ 3: 直線探索を用いてステップ幅 α_k を定める
ステップ 4: $x_{k+1} := x_k + \alpha_k d_k, k := k + 1$ としてステップ 1 に戻る

停止条件は、勾配ベクトルの大きさ $||\nabla f(x_k)||$ が十分小さくなったことなどに設定することが多い。

2.1.4 直線探索

点 $x_k \in R^n$ と x_k における降下方向 d_k が与えられたときに可能ならばステップ幅 α を

$$f(x_k + \alpha_k d_k) = \min\{f(x_k + \alpha d_k) | \alpha > 0\}$$

となるように選びたい。関数 f が凸 2 次関数の場合は陽な値として正確に α_k が求まるが、そうでなければこの問題は非常に難しい (それ自体が一変数の最小化問題)。多くの場合は次のアルミホ条件を満たすように α_k を求める。

アルミホ条件 $0 < \xi < 1$ を満たす定数 ξ に対して

$$f(x_k + \alpha d_k) \leq f(x_k) + \xi \alpha \nabla f(x_k)^T d_k$$

アルミホ条件に対する直線探索 (x_k, d_k は所与とする)

ステップ 0: パラメータ $0 < \xi < 1, 0 < \tau < 1$ を選び、$\alpha := 1$ とする
ステップ 1: $f(x_k + \alpha d_k) \leq f(x_k) + \xi \alpha \nabla f(x_k)^T d_k$
ステップ 2: $\alpha := \tau \alpha$ としてステップ 1 に戻る

なお、アルミホ条件 $\nabla f(x_k + \alpha d_k)$ に関する条件を付加した次のウルフ条件も実用的に多く用いられている。

ウルフ条件　$0 < \xi_1 < \xi_2 < 1$ を満たす定数 ξ_1, ξ_2 に対して

$$f(x_k + \alpha d_k) \leq f(x_k) + \xi_1 \alpha \nabla f(x_k)^T d_k$$
$$\xi_2 \nabla f(x_k)^T d_k \leq \nabla f(x_k + \alpha d_k)^T d_k$$

2.2　最急降下法

最急降下法とは、反復法において、x_k における探索方向として最急降下方向 $-\nabla f(x_k)$ を用いる方法である。

最急降下法

ステップ 0:　初期点 x_0 を選び、$k := 0$ とする。また、ξ を十分に小さい正数に設定する
ステップ 1:　$||\nabla f(x_k)|| < \xi$ が満たされていれば終了する (停留点が見つかった)
ステップ 2:　探索方向 $d_k = -\nabla f(x_k)$ とする
ステップ 3:　アルミホ条件による直線探索を用いてステップ幅 α を定める
ステップ 4:　$x_{k+1} := x_k + \alpha_k d_k, k := k + 1$ としてステップ 1 に戻る

最急降下法の大域的収束性　関数 $f(x)$ に関するいくつかの仮定の下ではウルフ条件も用いた最急降下法は任意の初期点に対して以下の式が成り立つ定数 $0 < c < 1$ が存在する。

$$||x_k - x|| \leq ||x_k - x^*||$$

この式では一見、収束が遅いようには見えないが、x^* 付近では $||x_k - x^*||$ が非常に微小のため、最急降下法が収束するまでに要する反復数は非常に多い。

2.3　ニュートン法

ニュートン法は $f(x)$ の 2 次までのテイラー展開を最小化することを繰り返す方法である。

$$q(d) = f(x_k + d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2 f(x_k) d$$

として $\nabla q(d) = \nabla f(x_k) + \nabla^2 f(x_k) d = 0$ を満たす d に対して $x_{k+1} = x_k + d$ とする方法である。方向 d はニュートン方向と呼ばれ、以下のニュートン方程式を解くことで得られる。

$$\nabla^2 f(x_k) d = -\nabla f(x_k)$$

アルゴリズムとして、ニュートン法は以下のように記述できる。

ニュートン法

ステップ 0:　初期点 x_0 を選び、$k := 0$ とする。また、ξ を十分に小さい正の数に設定する
ステップ 1:　$||\nabla f(x_k)|| < \xi$ が満たされていれば終了する
ステップ 2:　方程式 $\nabla^2 d = -\nabla f(x_k)$ を解いて探索方向 d_k を求める
ステップ 3:　$x_{k+1} := x_k + d_k, k := k_1$ としてステップ 1 に戻る

ニュートン法の局所的２次収束性　初期点 x_0 を x^* の十分近くに取ると、ニュートン法で生成される点列 x_k は収束し、その収束先を x^* とすると以下の式が成り立つ定数 $c \geq 0$ が存在する。

$$||x_{k+1} - x^*|| \leq c||x_k - x^*||^2$$

1 次収束と異なり、上の 2 次収束は非常に速い。一方で、初期点の取り方が悪ければニュートン法は収束しなかったり、局所的最適解でない停留点に収束したりすることもしばしばである。その欠点を補う方法が準ニュートン法である。

2.4　準ニュートン法

先の述べたように、ニュートン法は時に局所的最適解でない点を求めるが、その理由は x_k におけるヘッセ行列 $\nabla^2 f(x_k)$ が必ずしも正値道でないからである (そのために方向 d_k が $f(x)$ の降下方向でないことがある)。

準ニュートン法はニュートン方程式におけるヘッセ行列 $\nabla^2 f(x_k)$ を正定値行列 B_k で近似することで探索方向が必ず降下方向になるようにするのである。探索方向 d_k は

$$B_k d = -\nabla f(x_k)$$

の解 d とし、行列 B_k が f の情報をもつように B_{k+1} に更新する。B_k の更新には様々な公式が提案されているが、本実験では良い性能をもつことが知られている、BFGS 公式を用いる。

準ニュートン法 (BFGS **公式**)

ニュートン法

ステップ 0:　初期点 x_0 を選び、B_0 を単位行列にとり、$k := 0$ とする。また、ξ を十分に小さい正の数に設定する

ステップ 1:　$||\nabla f(x_k)|| < \xi$ が満たされていれば終了する

ステップ 3:　方程式 $B_k f(x_k) d = -\nabla f(x_k)$ を解いて探索方向 d_k を求める

ステップ 4:　アルミホ条件による直線探索を用いてステップ幅 α_k を定める

ステップ 5:　$x_{k+1} := x_k + \alpha_k d_k$ とする

ステップ 6:　以下の BFGS 公式を用いて B_k を更新して B_{k+1} を作り、$k := k + 1$ としてステップ 1 に戻る

BFGS **公式**

$$B_{k+1} = B_k - \frac{B_k s_k (B_k s_k) T}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k}, s_k = x_{k+1} - x_k, y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

詳細は述べないが、BFGS 公式の準ニュートン法はいくつかの家庭のもとでは大域的収束性をもち、収束スピードはニュートン法に劣るものの、最急降下法より速いことが知られている。

3　結果

3.1　関数 $f(x) = \frac{1}{2}x^T Qx + c^T x + \exp((x_1 - x_2)^2)$ に関して

2 つのデータ (Q, c および初期点の組 2 つ) を選び、最急降下法、ニュートン法、準ニュートン法の反復回数と目的関数のグラフを作成して比較する

$$Q = \begin{pmatrix} 9.0 & 4.0 \\ 4.0 & 2.0 \end{pmatrix}$$

$$c = \begin{pmatrix} -2.0 \\ 6.0 \end{pmatrix}$$

をとった結果は以下のようになる。

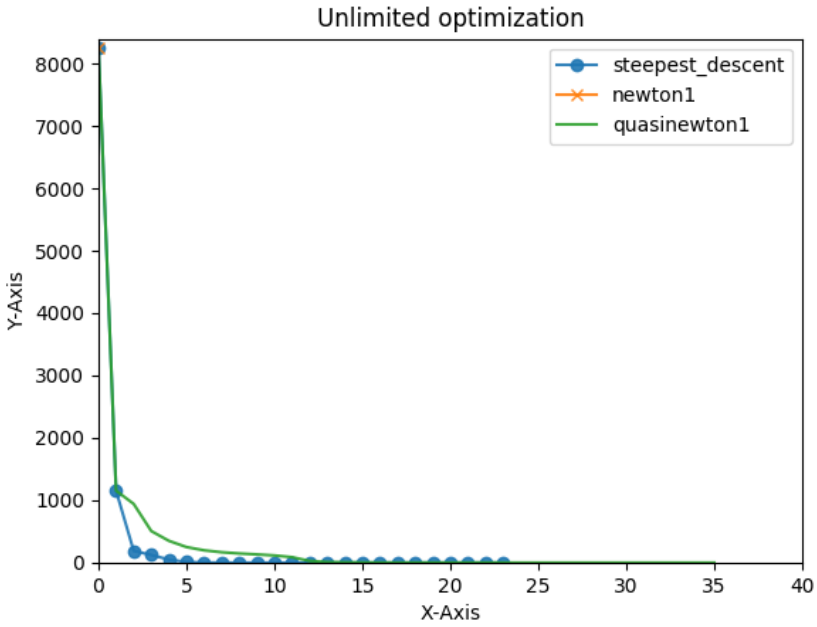

図 1　探索アルゴリズム比較

次に 1 つのデータを選び、勾配ベクトルの大きさがある程度小さくなったあと、最急降下法が生成する点列の 2 次元平面におけるプロットを作成する

Q, c の値は上と同じとする。
結果は以下のようになる探索終了から 10 遷移分をプロットしている

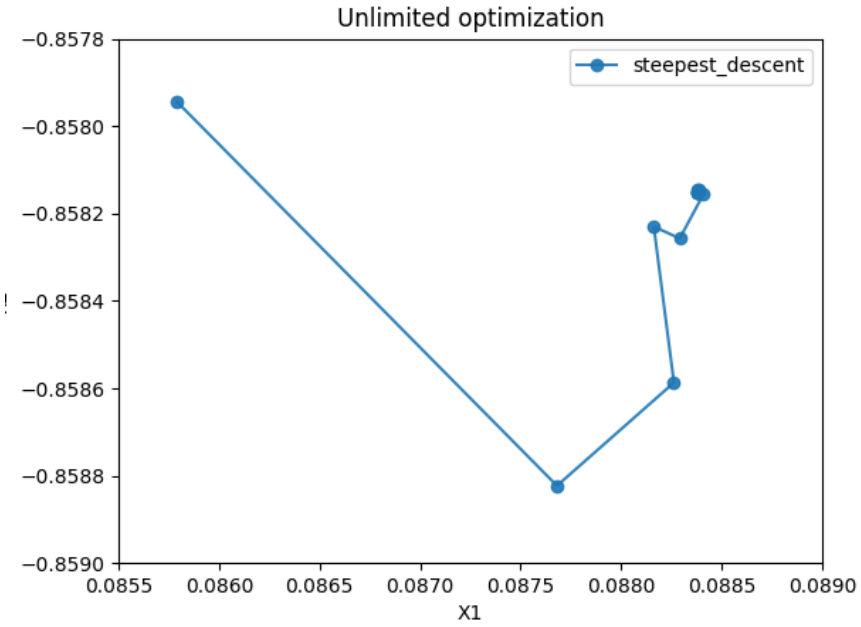

図 2　最急降下法が生成した二次元配列

3.1.1　関数が凸関数であるかの吟味

ある区間で定義された実数値関数 f で、区間内の任意の 2 点 x , y と開区間 $(0, 1)$ 内の任意の t に対して

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

を満たすものをいう。ここで

図 1 から f(x) の値は降下を続け、また図 2 から (x_1, x_2) はジグザグと進みながらもその移動距離を小さくしているすなわち $\nabla f(x)$ が収束していっていることがわかり関数 f(x) は凸関数であると言える

3.2　関数 $f(x) = \Sigma_{i=0}^{4}\Sigma_{j=0}^{4}a_{ij}x_1^i x_2^j$ に関して

3.2.1　ニュートン法、準ニュートン法の収束について

結果よりニュートン法の収束が速いと言える。準ニュートン法は BFGS 公式によりヘッセ行列 $\nabla^2 f(x)$ を近似している。これによりヘッセ行列 $\nabla^2 f(x)$ に逆行列が存在しないことによりこれ以上探索方向を生成できなくなってしまうというニュートン法の短所を補っているこのからも準ニュートン法はニュートン法の派生であって純粋なニュートン法にはかなわないと推測できた。

3.2.2　ニュートン法、準ニュートン法が生成した点列の収束先はどのような点になっているか

ニュートン法においてはニュートン法の局所的 2 次収束性により $||x_{k+1} - x^*|| \leq \Upsilon||x_k - x^*||$ が成り立つと言える。ただし Υ は定数である。

またニュートン法、準ニュートン法の収束先については結果より凸関数であればその頂点部であると考察される。

4　まとめ

今回の実験を通して無制約最小化問題の理論をより実践的な理解をすることができたと言える。ニュートン法、準ニュートン法、そして最急降下法それぞれの式の意味が今回の実験結果を通して白日のもととなったのだ。自分は以前東大の GPU 貸し出しの応募に受かり lstm で株式予想や翻訳、cnn で画像認識などをやったことがあったが最小化問題として共通するところがあり勉強になったと言える。

参考文献

[1] 工学基礎 最適化とその応用・矢部博

5　コード

コードは以下のようになった

ソースコード 1　steepest_descent.cpp

```
// steepest_descent.cpp : R \[ A v P [ V ^^cc^^83 G g | C g ‘
//

#include "stdafx.h"
#include<stdio.h>
#include<iostream>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#define eps 0.000001

using namespace std;

double M[3][3];
double c[3];

void cmp_dfx(double x[], double dfx[], double M[3][3], double c[]);

/* _x ^^c9^^82 ^^da^^93 I ^^d6^^90 l v Z */
double comp_val(double x[], double M[3][3], double c[]);
void show(double g[], double f_x, int k);
```

```
/* _x , ^^c9^^82    ^^c4^^92 T    d   ^^c9^^91^^ce^^82    A   ~ z T  C g = x    z x N g
double line_search(double x[], double d[], double alpha, double M[3][3], double c[], dou

int _tmain(int argc, _TCHAR* argv[])
{
        int i, j, k;
        //int cont;
        double alpha;
        //double val, norm, expval;
        // double newx[3];
        double x[3],  d[3], g[3];
        char fname[128];
        errno_t error;
        FILE *infile;


        printf("input filename:");
        fgets(fname, sizeof(fname), stdin);
        fname[strlen(fname) - 1] = '\0';
        fflush(stdin);

        if (error = fopen_s(&infile, fname, "r") != 0){ printf(" t @ C        ^^dc^^82
        else {
                /* t @ C      f [ ^   ^^c7^^82^^dd^^8 d    */
                for (i = 1; i <= 2; i++){
                        for (j = 1; j <= 2; j++){ fscanf_s(infile, " %lf", &M[i][j]); }
                }
                for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &c[i]); }
                for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &x[i]); }
                fclose(infile);

                /*     v    F e   ^^ec^^90^^ac*/
                k = 0;
                cmp_dfx(x, g, M, c);

                cout << (pow(g[1], 2) + pow(g[2], 2)) << endl;
```

```
                while (sqrt(pow(g[1], 2) + pow(g[2], 2)) >= eps) {
                        cout << sqrt(pow(g[1], 2) + pow(g[2], 2)) << endl;
                        //cout << k << " " << d[1] << ", " << d[2] << endl;
                        show(g, comp_val(x, M, c), k);
                        d[1] = (-1) * g[1];
                        d[2] = (-1) * g[2];//  d ^^cc^^8cvZ
                        alpha = line_search(x, d, 1, M, c, g);
                        x[1] = x[1] + alpha * d[1];
                        x[2] = x[2] + alpha * d[2];
                        cmp_dfx(x, g, M, c); // g ^^cc^^8dXV
                        k = k + 1;
                        if (k == 1000) {
                                break;
                        }
                }
        //      std::cout << alpha << std::endl;
        }

        return 0;
}

/*  _x ^^c9^^82      ^^da^^93I ^^d6^^90  l   v Z  */
double comp_val(double x[], double M[3][3], double c[])
{
        double fx = (M[1][1] * pow(x[1], 2) + 2 * M[1][2] * x[1] * x[2] + M[2][2] * pow(
        return fx;
}

/*  _x,  ^^c9^^82    ^^c4^^92T    d   ^^c9^^91^^ce^^82    A  ~ z T  Cg = x   z x N g
double line_search(double x[], double d[], double alpha, double M[3][3], double c[], dou
{
        // std::cout << alpha << std::endl;
        //cout << "alpha" << alpha << endl;
        double xk[3];
        xk[1] = x[1] + alpha * d[1];
        xk[2] = x[2] + alpha * d[2];
        double kusi = 0.25;
```

```
        double tau = 0.5;
        double left;
        double right;
        left = comp_val(xk, M, c);
        right = comp_val(x, M, c) + kusi * alpha * (g[1] * d[1] + g[2] * d[2]);
        if (left <= right) {
                return alpha;
        }
        else {
                return line_search(x, d, alpha * tau, M, c, g);
        }
}

void cmp_dfx(double x[], double dfx[], double M[3][3], double c[]) {
        dfx[1] = (M[1][1] * x[1] + M[1][2] * x[2]) + c[1] + 2 * exp(pow(x[1] - x[2], 2))
        dfx[2] = (M[2][2] * x[2] + M[1][2] * x[1]) + c[2] + 2 * exp(pow(x[1] - x[2], 2))
        return;
}

void show(double g[], double f_x, int k) {
        cout << "k = " << k << endl;
        cout << "gT = " << "(" << g[1] << ", " << g[2] << ")" << endl;
        cout << "f(x) = " << f_x << endl;
        cout << endl;
        return;
}
```

ソースコード 2 newton1.cpp

```
// newton1.cpp : R \[ A v P [ V ^^cc^^83 G g | C g ‘ ^^dc^^82
//

#include "stdafx.h"
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include <iostream>
```

```
#define  eps 0.00000001

using namespace std;

double comp_val(double x[], double M[3][3], double c[]);
void show(double g[], double f_x, int k);
void make_d(double d[], double M[3][3], double x[], double c[], double g[]);
void cmp_dfx(double x[], double g[], double M[3][3], double c[]);
double cmp_complex(double x[]);
double cmp_base(double M[3][3], double x[]);

int _tmain(int argc, _TCHAR* argv[])
{
        double M[3][3];
        double c[3];
        int i, j, k;
        double x[3];
        char fname[128];
        errno_t error;
        FILE *infile;

        // ori
        double d[3];
        double g[3];

        printf("input filename:");
        fgets(fname, sizeof(fname), stdin);
        fname[strlen(fname) - 1] = '\0';
        fflush(stdin);

        if (error = fopen_s(&infile, fname, "r") != 0){ printf(" t @ C        ^^dc^^82
        else{
                /* t @ C      f [ ^   ^^c7^^82^^dd^^8 d    */
                for (i = 1; i <= 2; i++){
                        for (j = 1; j <= 2; j++){ fscanf_s(infile, " %lf", &M[i][j]); }
                }
                for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &c[i]); }
```

```
                for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &x[i]); }
                fclose(infile);

                /*      v      F e    ^^ec^^90^^ac*/
                k = 0;
                cmp_dfx(x, g, M, c);

                while (sqrt(pow(g[1], 2) + pow(g[2], 2)) >= eps) {
                    show(g, comp_val(x, M, c), k);

                        if (k == 1000) {
                                break;
                        }

                        // d ^^cc^^8cvZ
                        if (abs(cmp_base(M, x)) < eps) {
                                cout << " t s ^^f1^^90^^b6^^90   ^^c5^^82   ^^dc^^82     "
                                return 1;
                        }
                        else {
                                make_d(d, M, x, c, g);
                        }

                        // x ^^cc^^8dXV
                        x[1] = x[1] + d[1];
                        x[2] = x[2] + d[2];

                        // g ^^cc^^8dXV
                        cmp_dfx(x, g, M, c);
                        k = k + 1;
                }
        }
        return 0;
}

double cmp_complex(double x[]) {
        return (1 + 2 * pow( (x[1] - x[2]), 2))  *  exp(pow((x[1] - x[2]), 2));
```

```
}

double dx1dx1(double M[3][3], double x[]) {
        return M[1][1] + 2 * cmp_complex(x);
}

double dx1dx2(double M[3][3], double x[]) {
        return M[1][2] - 2 * cmp_complex(x);
}

double dx2dx2(double M[3][3], double x[]) {
        return M[2][2] + 2 * cmp_complex(x);
}

double cmp_base(double M[3][3], double x[]) {
        return dx1dx2(M, x) * dx1dx1(M, x) - dx2dx2(M, x) * dx1dx2(M, x);
}

void make_d(double d[], double M[3][3], double x[], double c[], double g[]) {
        d[1] = (-1) *         (dx1dx2(M, x) * g[1] - dx1dx2(M, x) * g[2]) / cmp_base(M, x
        d[2] = (-1) * ((-1) * dx2dx2(M, x) * g[1] + dx1dx1(M, x) * g[2]) / cmp_base(M, x
        return;
}

/*  _x ^^c9^^82      ^^da^^93 I ^^d6^^90  l   v Z  */
double comp_val(double x[], double M[3][3], double c[]) {
        return (M[1][1] * pow(x[1], 2) + 2 * M[1][2] * x[1] * x[2] + M[2][2] * pow(x[2],
}

/*  _x ,  ^^c9^^82    ^^c4^^92 T     d   ^^c9^^91^^ce^^82    A   ~ z T  C g = x    z x N g


void cmp_dfx(double x[], double g[], double M[3][3], double c[]) {
        g[1] = (M[1][1] * x[1] + M[1][2] * x[2]) + c[1] + 2 * exp(pow(x[1] - x[2], 2)) *
        g[2] = (M[2][2] * x[2] + M[1][2] * x[1]) + c[2] - 2 * exp(pow(x[1] - x[2], 2)) *
        return;
}
```

```
void show(double g[], double f_x, int k) {
        cout << "k = " << k << endl;
        cout << "gT = " << "(" << g[1] << ", " << g[2] << ")" << endl;
        cout << "f(x) = " << f_x << endl;
        cout << endl;
        return;
}
```

ソースコード 3 newton2.cpp

```
// newton2.cpp : R \[ A v P [ V ^^cc^^83 G g | C g ‘ ^^dc^^82
//

#include "stdafx.h"
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include <iostream>

using namespace std;

#define  eps 0.00000001

double comp_val(double x[], double M[3][3], double c[], double A[5][3]);
void show(double g[], double f_x, int k);
void make_d(double d[], double M[3][3], double x[], double c[], double g[], double A[5][
void cmp_dfx(double x[], double g[], double c[], double A[5][3]);
double line_search(double x[], double d[], double alpha, double M[3][3], double c[], dou
double cmp_base(double x[], double A[5][3]);

int _tmain(int argc, _TCHAR* argv[])
{
        double A[5][3];
        int i, j, k;
        double x[3];
        char fname[128];
```

```
errno_t error;
FILE *infile;

//Original
double alpha;
double d[3];
double g[3];
double M[3][3];
double c[3];

printf("input filename:");
fgets(fname, sizeof(fname), stdin);
fname[strlen(fname) - 1] = '\0';
fflush(stdin);

if (error = fopen_s(&infile, fname, "r") != 0){ printf(" t @ C       ^^dc^^82
else {
        /* t @ C      f [ ^   ^^c7^^82^^dd^^8 d    */
        for (i = 0; i<5; i++){
                for (j = 0; j<3; j++){
                        fscanf_s(infile, " %lf", &A[i][j]);
                }
        }
        for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &x[i]); }
        fclose(infile);

        /*     v     F e   ^^ec^^90^^ac*/
        // k
        k = 0;

        // d
        cmp_dfx(x, g, c, A);

        while (sqrt(pow(g[1], 2) + pow(g[2], 2)) >= eps) {

                show(g, comp_val(x, M, c, A), k);
```

```
                        if (k == 1000) {
                                break;
                        }


                        // d ^^cc^^8dXV
                        if (cmp_base(x, A) < eps) {
                                cout << " t s ^^f1^^90^^b6^^90  ^^c5^^82  ^^dc^^82    "
                                return 1;
                        }
                        else {
                                make_d(d, M, x, c, g, A);
                        }

                        // x ^^cc^^8dXV
                        x[1] = x[1] + d[1];
                        x[2] = x[2] + d[2];

                        // g ^^cc^^8dXV
                        cmp_dfx(x, g, c, A);

                        // k ^^cc^^8dXV
                        k = k + 1;
                }
        }

        return 0;
}


double dx1dx1(double x[], double A[5][3]) {
        return 12 * A[4][0] * pow(x[1], 2) + 6 * A[3][0] * x[1] + 2 * A[2][0];
}

double dx1dx2(double x[], double A[5][3]) {
        return 2 * A[0][2];
}
```

```
double dx2dx2(double x[], double A[5][3]) {
        return A[1][1];
}

double cmp_base(double x[], double A[5][3]) {
        return dx1dx1(x, A) * dx1dx2(x, A) - dx2dx2(x, A) * dx1dx2(x, A);
}

void make_d(double d[], double M[3][3], double x[], double c[], double g[], double A[5][
        d[1] = (       dx1dx2(x, A) * g[1] - dx1dx2(x, A) * g[2]) / cmp_base(x, A) * (-
        d[2] = ((-1) * dx2dx2(x, A) * g[1] + dx1dx1(x, A) * g[2]) / cmp_base(x, A) * (-
        return;
}

/*  _x ^^c9^^82      ^^da^^93 I ^^d6^^90  l   v Z  */
double comp_val(double x[], double M[3][3], double c[], double A[5][3])
{
        double fx = A[4][0] * pow(x[1], 4) + A[3][0] * pow(x[1], 3) + A[2][0] * pow(x[1]
                        A[1][0] * x[1] + A[1][1] * x[1] * x[2] + A[0][2] * pow(x[2], 2)
        //double fx = (M[1][1] * pow(x[1], 2) + 2 * M[1][2] * x[1] * x[2] + M[2][2] * po
        return fx;
}

/*  _x ,  ^^c9^^82    ^^c4^^92 T    d   ^^c9^^91^^ce^^82    A   ~ z T  C g = x   z x N g

void cmp_dfx(double x[], double g[], double c[], double A[5][3]) {
        g[1] = 4 * A[4][0] * pow(x[1], 3) + 3 * A[3][0] * pow(x[1], 2) + 2 * A[2][0] * x
        g[2] = A[1][1] * x[1] + 2 * A[0][2] + A[0][1];
        //dfx[1] = (M[1][1] * x[1] + M[1][2] * x[2]) + c[1] + 2 * exp(pow(x[1] - x[2], 2
        //dfx[2] = (M[2][2] * x[2] + M[1][2] * x[1]) + c[2] + 2 * exp(pow(x[1] - x[2], 2
        return;
}

void show(double g[], double f_x, int k) {
        cout << "k = " << k << endl;
        cout << "gT = " << "(" << g[1] << ", " << g[2] << ")" << endl;
```

```
        cout << "f(x) = " << f_x << endl;
        cout << endl;
        return;
}
```

ソースコード 4 quasinewton1.cpp

```
// quasinewton1.cpp :  R  \ [   A v  P [ V   ^^cc^^83 G  g   | C  g     `   ^^dc
//

#include "stdafx.h"
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include <iostream>

#define  eps 0.00000001

using namespace std;

double M[3][3];
double c[3];

/*  _x ^^c9^^82     ^^da^^93 I ^^d6^^90  l   v Z  */
//double comp_val(double x[]);
double cmp_val(double x[], double M[3][3], double c[]);

/* _x ,  ^^c9^^82   ^^c4^^92 T    d  ^^c9^^91^^ce^^82   A  ~ z T  C g = x   z x N g
//double line_search(double x[], double g[], double d[]);
double line_search(double t, double x[], double d[], double alpha, double M[3][3], doubl



void show(double g[], double f_x, int k);
void init_B(double B[3][3]);
void cmp_g(double x[], double g[], double M[3][3], double c[]);
void cmp_pg(double pg[], double p[]);
void cmp_px(double px[], double x[]);
```

```
void cmp_B(double B[3][3], double x[], double px[], double g[], double pg[], double M[3]
void cmp_d(double d[], double g[], double B[3][3]);

int _tmain(int argc, _TCHAR* argv[])
{
        int i, j, k;
        double x[3];
        char fname[128];
        errno_t error;
        FILE *infile;

        // original
        double alpha = 1;
        double B[3][3];
        double g[3];
        double d[3];

        double px[3];
        double pg[3];

        printf("input filename:");
        fgets(fname, sizeof(fname), stdin);
        fname[strlen(fname) - 1] = '\0';
        fflush(stdin);

        if (error = fopen_s(&infile, fname, "r") != 0){ printf(" t @ C        ^^dc^^82
        else {
                /* t @ C      f [ ^   ^^c7^^82^^dd^^8d    */
                for (i = 1; i <= 2; i++){
                        for (j = 1; j <= 2; j++){ fscanf_s(infile, " %lf", &M[i][j]); }
                }
                for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &c[i]); }
                for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &x[i]); }
                fclose(infile);

                /*     v    F e   ^^ec^^90^^ac*/
                init_B(B);
```

```
// k, g
k = 0;
cmp_g(x, g, M, c);


while (sqrt(pow(g[1], 2) + pow(g[2], 2)) >= eps) {
        show(g, cmp_val(x, M, c), k);

        if (k == 1000) {
                break;
        }

        // g  ^^cc^^8dXV
        cmp_g(x, g, M, c);

        // d  ^^cc^^8dXV
        if (k == 0) {
                cmp_d(d, g, B);
        }
        else {
                cmp_B(B, x, px, g, pg, M, c);
                cmp_d(d, g, B);
        }

        alpha = line_search(0, x, d, 1, M, c, g);

        // pg, px  ^^cc^^8dXV
        cmp_pg(pg, g);
        cmp_px(px, x);

        // x ^^cc^^8dXV
        x[1] = x[1] + alpha * d[1];
        x[2] = x[2] + alpha * d[2];

        // k  ^^cc^^8dXV
        k = k + 1;
```

```
                }
                cout << " v Z I     ^^dc^^82    B " << endl;
        }
        return 0;
}

/*  _x ^^c9^^82       ^^da^^93 I ^^d6^^90  l    v Z  */
double cmp_val(double x[], double M[3][3], double c[]) {
        return (M[1][1] * pow(x[1], 2) + 2 * M[1][2] * x[1] * x[2] + M[2][2] * pow(x[2],
}

void init_B(double B[3][3]) {
        B[1][1] = 1;
        B[1][2] = 0;
        B[2][1] = 0;
        B[2][2] = 1;
}

/* _x ,  ^^c9^^82     ^^c4^^92 T      d   ^^c9^^91^^ce^^82     A   ~ z T  C g = x    z x N g
double line_search(double t, double x[], double d[], double alpha, double M[3][3], doubl
{
        double xk[3];
        xk[1] = x[1] + alpha * d[1];
        xk[2] = x[2] + alpha * d[2];
        double kusi = 0.25;
        double tau = 0.5;
        double left;
        double right;
        left = cmp_val(xk, M, c);
        right = cmp_val(x, M, c) + kusi * alpha * (g[1] * d[1] + g[2] * d[2]);
        if (left <= right || t >= 1000) {
                return alpha;
        }
        else {
                return line_search(t + 1, x, d, alpha * tau, M, c, g);
        }
}
```

```
void cmp_g(double x[], double g[], double M[3][3], double c[]) {
        g[1] = (M[1][1] * x[1] + M[1][2] * x[2]) + c[1] + 2 * exp(pow(x[1] - x[2], 2)) *
        g[2] = (M[2][2] * x[2] + M[1][2] * x[1]) + c[2] - 2 * exp(pow(x[1] - x[2], 2)) *
}

void cmp_pg(double pg[], double g[]) {
        pg[1] = g[1];
        pg[2] = g[2];
}

void cmp_px(double px[], double x[]) {
        px[1] = x[1];
        px[2] = x[2];
}

void cmp_y(double y[], double g[], double pg[]) {
        y[1] = g[1] - pg[1];
        y[2] = g[2] - pg[2];
}

void  cmp_s(double s[], double x[], double px[]) {
        s[1] = x[1] - px[1];
        s[2] = x[2] - px[2];
}

void cmp_Bs(double Bs[], double B[3][3], double s[]) {
        Bs[1] = B[1][1] * s[1] + B[1][2] * s[2];
        Bs[2] = B[2][1] * s[1] + B[2][2] * s[2];
}

double cmp_right(double y[], double s[]) {
        return (pow(y[1], 2) + pow(y[2], 2)) / (s[1] * y[1] + s[2] * y[2]);
}

double cmp_mid(double Bs[], double s[]) {
        return (pow(Bs[1], 2) + pow(Bs[2], 2)) / (s[1] * Bs[1] + s[2] * Bs[2]);
```

```
}

void cmp_B(double B[3][3], double x[], double px[], double g[], double pg[], double M[3]
        double Bs[3];
        double s[3];
        double y[3];

        double mid;
        double right;

        cmp_s(s, x, px);
        cmp_y(y, g, pg);
        cmp_Bs(Bs, B, s);

        mid   = cmp_mid(Bs, s);
        right = cmp_right(y, s);

        B[1][1] = B[1][1] - mid + right;
        B[2][2] = B[2][2] - mid + right;
}

double cmp_BT_base(double B[3][3]) {
        return B[1][1] * B[2][2] - B[1][2] * B[2][1];
}

void cmp_BT(double BT[3][3], double B[3][3]) {
        double tmpB[3][3];
        tmpB[1][1] = B[1][1];
        tmpB[1][2] = B[1][2];
        tmpB[2][1] = B[2][1];
        tmpB[2][2] = B[2][2];

        BT[1][1] = tmpB[2][2] / cmp_BT_base(tmpB);
        BT[2][2] = tmpB[1][1] / cmp_BT_base(tmpB);
        BT[1][2] = 0;
        BT[2][1] = 0;
        cout << BT[1][1] << BT[1][2] << endl;
```

```
        cout << BT[2][1] << BT[2][2] << endl;
}

void cmp_d(double d[], double g[],  double B[3][3]) {
        double BT[3][3];
        cmp_BT(BT, B);

        d[1] = (-1) * (BT[1][1] * g[1] + BT[1][2] * g[2]);
        d[2] = (-1) * (BT[2][1] * g[1] + BT[2][2] * g[2]);
}

void show(double g[], double f_x, int k) {
        cout << "k = "    << k   << endl;
        cout << "gT = "   << "(" << g[1] << ", " << g[2] << ")" << endl;
        cout << "f(x) = " << f_x << endl;
        cout << endl;
}
```

ソースコード 5　quasinewton2.cpp

```
// quasinewton2.cpp :  R  \ [    A v  P [ V    ^^cc^^83 G  g    | C  g      `    ^^dc
//

#include "stdafx.h"
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include <iostream>

#define  eps 0.00000001

using namespace std;

double A[5][3];

// original
double B[3][3];
```

```
/*  _x ^^c9^^82      ^^da^^93 I ^^d6^^90  l   v Z  */
double cmp_val(double x[], double A[5][3]);

/*  _x ,  ^^c9^^82    ^^c4^^92 T     d   ^^c9^^91^^ce^^82    A   ~ z T  C g = x   z x N g
double line_search(double t, double x[], double d[], double alpha, double g[], double A[

// original
void init_B(double B[3][3]);
void cmp_g(double x[], double g[], double A[5][3]);
void show(double g[], double f_x, int k);
double cmp_val(double x[], double A[5][3]);
void cmp_d(double d[], double g[], double B[3][3]);
double line_search(double t, double x[], double d[], double alpha, double g[], double A[
void cmp_B(double B[3][3], double x[], double px[], double g[], double pg[]);
void cmp_pg(double pg[], double g[]);
void cmp_px(double px[], double x[]);

int _tmain(int argc, _TCHAR* argv[])
{
        int i, j, k;
        int cont;
        double x[3];
        char fname[128];
        errno_t error;
        FILE *infile;

        // original
        double alpha = 1;
        double g[3];
        double d[3];
        double pg[3];
        double px[3];


        printf("input filename:");
        fgets(fname, sizeof(fname), stdin);
```

```
fname[strlen(fname) - 1] = '\0';
fflush(stdin);

if (error = fopen_s(&infile, fname, "r") != 0){ printf(" t @ C        ^^dc^^82
else {
        /* t @ C      f [ ^   ^^c7^^82^^dd^^8 d    */
        for (i = 0; i < 5; i++){
                for (j = 0; j < 3; j++){
                        fscanf_s(infile, " %lf", &A[i][j]);
                }
        }
        for (i = 1; i <= 2; i++){ fscanf_s(infile, " %lf", &x[i]); }
        fclose(infile);

        /*     v     F e   ^^ec^^90^^ac*/
        //B
        init_B(B);

        // k, g
        k = 0;
        cmp_g(x, g, A);

        while (sqrt(pow(g[1], 2) + pow(g[2], 2)) >= eps) {
                show(g, cmp_val(x, A), k);

                if (k >= 1000) {
                        break;
                }

                // g  X V
                cmp_g(x, g, A);

                // d  X V
                if (k == 0) {
                        cmp_d(d, g, B);
                }
                else {
```

```
                cmp_B(B, x, px, g, pg);
                cmp_d(d, g, B);
            }

            alpha = line_search(0, x, d, 1, g, A);

            // pg, px  X V
            cmp_pg(pg, g);
            cmp_px(px, x);

            // x  X V
            x[1] = x[1] + alpha * d[1];
            x[2] = x[2] + alpha * d[2];

            // k  X V
            k = k + 1;
        }
        cout << " v Z I     ^^dc^^82    B " << endl;
    }
    return 0;
}

void init_B(double B[3][3]) {
    B[1][1] = 1;
    B[1][2] = 0;
    B[2][1] = 0;
    B[2][2] = 1;
}

void cmp_g(double x[], double g[], double A[5][3]) {
    g[1] = 4 * A[4][0] * pow(x[1], 3) +
           3 * A[3][0] * pow(x[1], 2) +
           2 * A[2][0] * x[1]         +
           A[1][0]                    +
           A[1][1] * x[2];
    g[2] = A[1][1] * x[1] + 2 * A[0][2] + A[0][1];
}
```

```
void show(double g[], double f_x, int k) {
        cout << "k = " << k << endl;
        cout << "gT = " << "(" << g[1] << ", " << g[2] << ")" << endl;
        cout << "f(x) = " << f_x << endl;
        cout << endl;
        return;
}

/*  _x ^^c9^^82      ^^da^^93 I ^^d6^^90  l   v Z  */
double cmp_val(double x[], double A[5][3])
{
        return A[4][0] * pow(x[1], 4) +
                    A[3][0] * pow(x[1], 3) +
                    A[2][0] * pow(x[1], 2) +
                    A[1][0] * x[1]         +
                    A[1][1] * x[1] * x[2]  +
                    A[0][2] * pow(x[2], 2) +
                    A[0][1] * x[2];
}

double cmp_BT_base(double B[3][3]) {
        return B[1][1] * B[2][2] - B[1][2] * B[2][1];
}

void cmp_BT(double BT[3][3], double B[3][3]) {
        double tmpB[3][3];
        tmpB[1][1] = B[1][1];
        tmpB[1][2] = B[1][2];
        tmpB[2][1] = B[2][1];
        tmpB[2][2] = B[2][2];

        BT[1][1] = tmpB[2][2] / cmp_BT_base(tmpB);
        BT[2][2] = tmpB[1][1] / cmp_BT_base(tmpB);
        BT[1][2] = 0;
        BT[2][1] = 0;
        cout << BT[1][1] << BT[1][2] << endl;
```

```
        cout << BT[2][1] << BT[2][2] << endl;
}

void cmp_d(double d[], double g[], double B[3][3]) {
        double BT[3][3];
        cmp_BT(BT, B);

        d[1] = (-1) * (BT[1][1] * g[1] + BT[1][2] * g[2]);
        d[2] = (-1) * (BT[2][1] * g[1] + BT[2][2] * g[2]);
}

void cmp_pg(double pg[], double g[]) {
        pg[1] = g[1];
        pg[2] = g[2];
}

void cmp_px(double px[], double x[]) {
        px[1] = x[1];
        px[2] = x[2];
}

/* _x, ^^c9^^82    ^^c4^^92 T    d   ^^c9^^91^^ce^^82    A   ~ z T  C g = x    z x N g
double line_search(double t, double x[], double d[], double alpha, double g[], double A[
        double xk[3];
        double kusi = 0.25;
        double tau = 0.5;
        double left;
        double right;

        xk[1] = x[1] + alpha * d[1];
        xk[2] = x[2] + alpha * d[2];

        left  = cmp_val(xk, A);
        right = cmp_val(x, A) + kusi * alpha * (g[1] * d[1] + g[2] * d[2]);

        if (left <= right || t >= 1000) {
                return alpha;
```

```
        }
        else {
                return line_search(t + 1, x, d, alpha * tau, g, A);
        }
}

void cmp_y(double y[], double g[], double pg[]) {
        y[1] = g[1] - pg[1];
        y[2] = g[2] - pg[2];
}

void  cmp_s(double s[], double x[], double px[]) {
        s[1] = x[1] - px[1];
        s[2] = x[2] - px[2];
}

void cmp_Bs(double Bs[], double B[3][3], double s[]) {
        Bs[1] = B[1][1] * s[1] + B[1][2] * s[2];
        Bs[2] = B[2][1] * s[1] + B[2][2] * s[2];
}

double cmp_right(double y[], double s[]) {
        return (pow(y[1], 2) + pow(y[2], 2)) / (s[1] * y[1] + s[2] * y[2]);
}

double cmp_mid(double Bs[], double s[]) {
        return (pow(Bs[1], 2) + pow(Bs[2], 2)) / (s[1] * Bs[1] + s[2] * Bs[2]);
}

void cmp_B(double B[3][3], double x[], double px[], double g[], double pg[]) {
        double Bs[3];
        double s[3];
        double y[3];

        double mid;
        double right;
```

```
        cmp_s(s, x, px);
        cmp_y(y, g, pg);
        cmp_Bs(Bs, B, s);

        mid = cmp_mid(Bs, s);
        right = cmp_right(y, s);

        B[1][1] = B[1][1] - mid + right;
        B[2][2] = B[2][2] - mid + right;
}
```