岡本安晴　2001.2.3;2001.8

極 値 探 索－１

１次元最適化問題

関数 $y = f(x)$ の最小値（極小値）を与える変数値 $x_{\min}$ の求め方について考える。最大値を与える変数値 x_{maz} は、関数 $y = g(x) = -f(x)$ の最小値を与える変数値として求めることができる。

図１は関数

$$y = 700 + x(x-6)(x-10)^2 \qquad (1)$$

のグラフをプログラム PDrawCurve.dpr によって描いたものである。

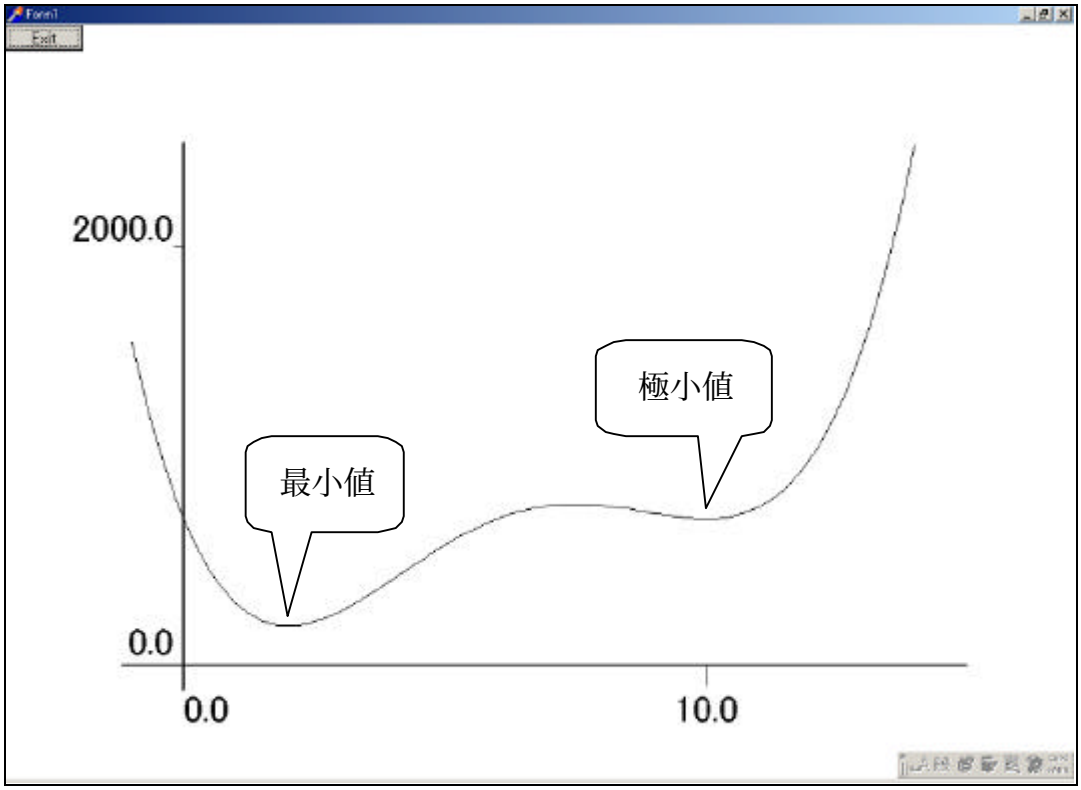

図１　関数 $y = 700 + x(x-6)(x-10)^2$ のグラフ

図１のグラフの場合、最小値の他に極小値がある。このような場合、間違いなく最小値を求めるためには、変数ｘの変域全体にわたって総当たり的に関数値を調べる方法が確実である。Exhaustive Search 法では、変域を細かく区切った分点における関数値を調べることにより、最小値を与える変数値の存在範囲を求めている。この Exhaustive Search 法で得られた存在範囲を基にして、最小値を与える変数値を求めることができる。

Exhaustive Search 法 (1)

関数 $y = f(x)$ の変域が $[Lb,Ub]$ であるとき、区間 $[Lb,Ub]$ を n 等分して関数が最小値をとる区間 $[Lx,Ux]$ を求める。ここで、$(Ux - Lx) \le \frac{2}{n}(Ub - Lb)$ である。区間 $[Lb,Ub]$ を n 等分したときの分点を x_i、$i = 0,\cdots,n$、とおけば

$$x_i = Lx + i(Ux - Lx)/n$$

となる。ただし、x_0 と x_n は区間 $[Lb,Ub]$ の端点である。これらの x_i に対する関数値 $f(x_i)$ を全て調べてその中での最小値を探す。全ての $f(x_i)$ を調べるので、最小値でない極小値を間違って最小値として扱う危険はない。n が十分に大きい値のとき、$i = i0$ のときの関数値 $f(x_{i0})$ が最小であれば、区間 $[Lx,Ux] = [x_{i0-1}, x_{i0+1}]$ において関数が最小値をとると考えられる。ただし、x_{i0} が端点の時は、

$x_{i0} = x_0 = Lb$ の場合は $[Ux, Lx] = [x_0, x_1]$
$x_{i0} = x_n = Ub$ の場合は $[Ux, Lx] = [x_{n-1}, x_n]$

において関数は最小値をとると考えられる。

この exhaustive search 法によって関数が最小値をとる区間 $[Lx,Ux]$ を求める手続きが MinOneDim である。この手続きはユニットファイル UOptOneDim.pas に次のようなヘッダーをもつものとして宣言されている。

```
procedure MinOneDim( L_b, U_b   : extended;
                     n          : Longint;
                     f          : TFunc;
                     var
                       L_x, U_x : extended );
```

第１、第２パラメータで与えられる値を端点とする区間 $[L\_b, U\_b]$ において、第４パラメータとして与えられる関数 f がその最小値をとるものとする。第３パラメータで与えられる数 n で区間 $[L\_b, U\_b]$ を等分して、各分点における関数値を比較することにより、関数が最小値をとる区間を絞り、その区間 $[L\_x, U\_X]$ の端点を L_x および U_x に返す。

先の関数

$$y = f(x) = 700 + x(x-6)(x-10)^2 \qquad (1)$$

の最小値を与える区間を求めるときは、例えば

```
function  f( x : extended ) : extended;
  begin
      f := 700 + x*(x-6)*sqr(x-10);
  end;
```

と関数を宣言しておき、手続き MinOneDim を

```
MinOneDim(-10.0, 20.0, 100, f, L_x, U_x );
```

のように呼出す。

上の場合、関数の最小値が区間[−10.0, 20.0]において調べられる。区間は 100 等分されて、最小値をとる範囲が絞られる。上の手続きの呼び出しによって得られた小区間は

$$[L\_x,\ U\_x] = [1.7,\ 2.3]$$

である。

プログラム PExhaustive.dpr は上の計算を行うものである。手続き MinOneDim はユニットファイル UOptOneDim.pas に宣言されているので、UExhaustve.pas の uses 節で

```
uses  UOptOneDim;
```

とユニット UOptOneDim の使用が宣言されている。ユニット UOptOneDim は、最小値の探索の途中経過を表示するためのフォーム（UOptOneDim.dfm）を生成することができる。ただし、この途中経過を表示するときは、フォームの生成と廃棄を行うところがコメントとしてあるので { } や//をはずす必要がある。また、表示を実行する手続き Display に付いている { } もはずす。

PExhaustive.dpr を実行して表示されるフォーム上の GO ボタンのクリックで計算が始まり、計算結果はフォーム上に表示される（図２）。

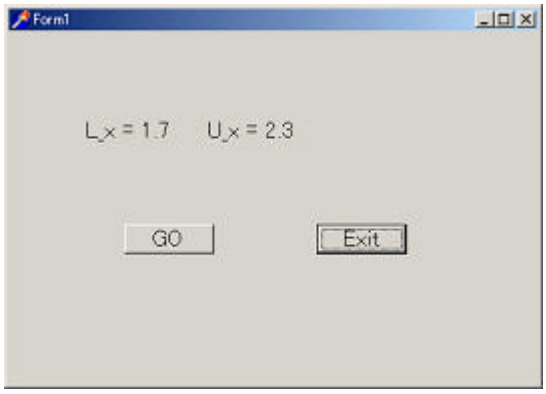

図２　PExhaustive.dpr の実行結果

Golden Section 法 (1)、(2)、(3)

Exhaustive search 法では関数が最小値をとる区間を絞ることはできるが、最小値を与える変数値を決めることはできない。関数が区間[L_b, U_b]で下に凸であるとき、最小値を与える変数値を求める方法の１つに Golden section 法と呼ばれているものがある。

Golden section 法では、関数の最小値が存在する区間[A,B]が与えられたとき（図３、４）、区間内の２点 C,D における関数値を比較して最小値の存在する区間を絞る。

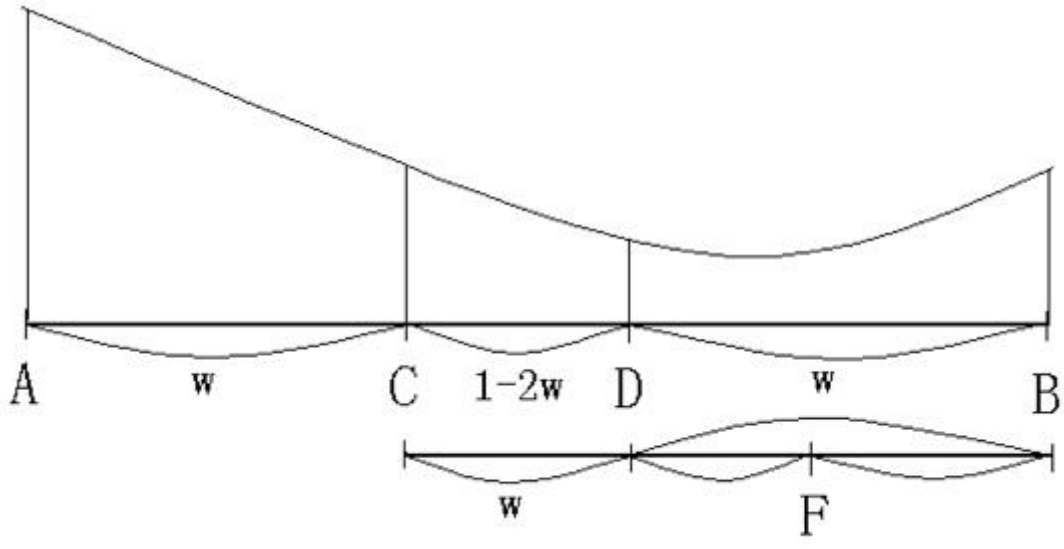

図３ Golden section の比率 w で設定した分点（点 C での関数値より点 D での関数値の方が小さい場合）。w は $\overline{AC}$ の $\overline{AB}$ に対する比率、および $\overline{CD}$ の $\overline{CB}$ に対する比率を表わす。

例えば図３のような場合では、区間$[C,B]$に絞る。これは点Ｄでの関数値が点Ｃでの値より小さい場合であるが、点Ｃでの関数値の方が点Ｄでの値より小さい（図４）ときは区間$[A,D]$に絞る。

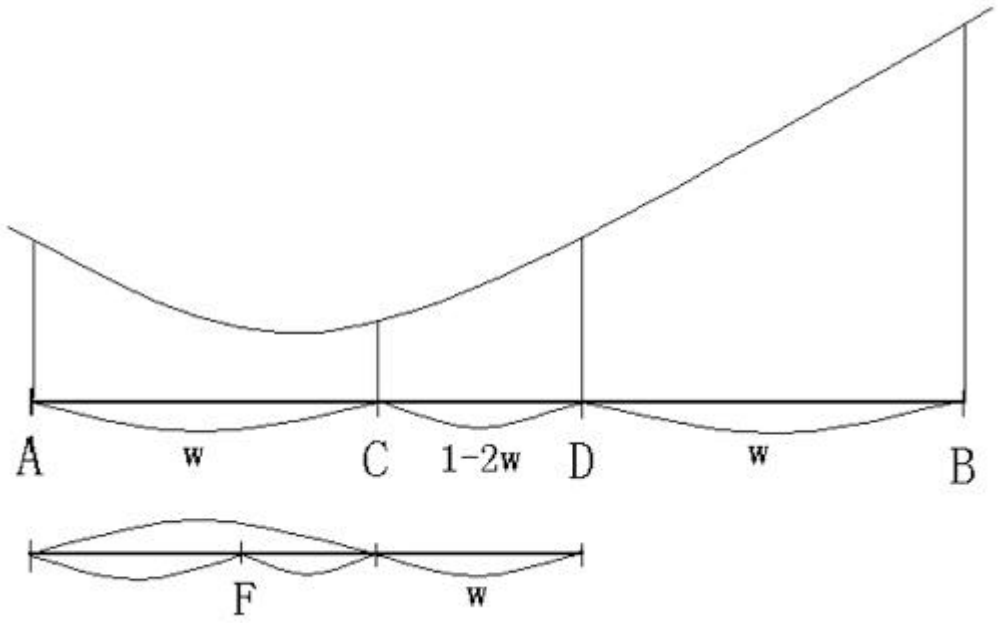

図４ Golden section の比率wで設定した分点（点Ｃでの関数値の方が点Ｄでの関数値より小さい場合)。ここで、$w = \overline{DB}/\overline{AB} = \overline{CD}/\overline{AD}$である。

関数値の計算回数を少なくするために、次のステップでの関数値の比較において、例えば図３の場合なら点Ｄにおける関数値と点Ｆにおける関数値との比較が区間$[A,B]$のときの点Ｃと点Ｄにおける比較に対応するようにする。このために、線分ＡＢ上における点ＣとＤの相対的な位置関係が、線分ＣＢ上における点ＤとＦの相対的な位置関係と同じになるようにする。すなわち、

$$\frac{\overline{AC}}{\overline{AB}} = \frac{\overline{CD}}{\overline{CB}} = w \qquad (2)$$

とする。図３の場合と図４の場合が同じように扱えるようにするため、点ＣとＤは線分ＡＢ上で対象な位置関係になるようにする。すなわち

$$\overline{AC} = \overline{DB}$$

とする。この関係は、線分ＣＢ上の２点ＤとＦについても成り立つようにする（図３の場合）ので

$$\overline{CD} = \overline{FB}$$

とおく。以上の関係と、式（２）を組み合わせて次式を得る。

$$\frac{w}{1} = \frac{1-2w}{1-w}$$

これより

$$w^2 - 3w + 1 = 0 \qquad (3)$$

を得る。

図４の場合も同様にして、

$$\frac{\overline{DB}}{\overline{AB}} = \frac{\overline{CD}}{\overline{AD}}$$

より式（３）が導かれる。

式（３）より、

$$w \approx 0.382$$

となる。

Golden section 法では、この $w \approx 0.382$ の値によって区間の内分点を設定して関数値の比較を行い、最小値の存在する区間を小さくしていく。具体的には、次のような手順になる。

（１）関数 f が最小値をとる区間を $[h0, h3]$ とし、端点での関数値を $v0 = f(h0)$ および $v3 = f(h3)$ とおく。

（２） $h1 = h0 + 0.382(h3 - h0)$ 、 $h2 = h3 - 0.382(h3 - h0)$ とおき、関数値を $v1 = f(h1)$ 、 $v2 = f(h2)$ とおく。

（３）$v1 > v2$ のとき（図３）は区間 $[h1, h3]$ で最小値をとると考えられるので、$h1$、$h2$ を新しく次の $h0$、$h1$ として、$h2 = h1 + 0.382(h3 - h1)$ とおく。この場合、 $f(h0)$、 $f(h1)$ は既に $f(h1)$、 $f(h2)$ として計算されているので、新しい $h2$ の値に対する $f(h2)$ の値の計算だけで済む。

$v1 < v2$ のとき（図４）は区間 $[h0, h2]$ で最小値をとると考えらるので、$h1$、$h2$ を新しく次の $h2$、 $h3$ として、 $h1 = h0 + 0.382(h3 - h0)$ とおく。この場合は新しい $h1$ の値に対する $f(h1)$ の値の計算だけで済む。

（４） $h0$ と $h3$ が十分に近ければ、関数 $f(x)$ は $x = x_{\min} = (h0 + h3)/2$ で最小値をとるとして、最小値を与える変数値 $x_{\min}$ の探索を終了する。

$h0$ と $h3$ が十分に近くないときはステップ（３）に戻る。

上の手順で関数 $f(x)$ の最小値を与える変数値 $x_{\min}$ を求める手続きが MinByGolden である。この手続きもユニットファイル UOptOneDim.pas に宣言されている。関数ヘッダーは次のようになっている。

```
procedure  MinByGolden( L_b, U_b   : extended;
                        f          : TFunc;
                        acc, zero  : extended;
                        var
                          opt_x    : extended );
```

第１，２パラメータには、$x_{\min}$ の存在区間$[L_b, U_b]$の下限と上限の値を設定する。第３パラメータには最小値の探索を行う関数 f を設定し、acc に精度（$h0$ と $h3$がどの程度近い値になれば上の手続きを終了するかという基準）と、zero にゼロとみなす基準値を設定する。求まった $x_{\min}$ の値は第６パラメータ opt_x に返される。

極値探索の精度の限界(3)

acc の値は計算の精度によって決まる。計算は Extended 型で行われているので、Extended 型の精度の平方根が acc に設定する精度の上限の目安となる。これは次のような理由による。

関数 $y = f(x)$ を $x = x_{\min}$ において 2 次式で近似すると

$$f(x) \approx f(x_{\min}) + (x - x_{\min}) f'(x_{\min}) + \frac{(x - x_{\min})^2}{2} f''(x_{\min})$$

となる。ここで、$y = f(x)$ は $x = x_{\min}$ において最小値をとるので

$$f'(x_{\min}) = 0$$

となっている。したがって、

$$f(x) \approx f(x_{\min}) + \frac{(x - x_{\min})^2}{2} f''(x_{\min}) \qquad (4)$$

となる。

いま、

$$x = x_{\min} + d$$

とおくと

$$f(x) = f(x_{\min} + d) \approx f(x_{\min}) + \frac{d^2}{2} f''(x_{\min})$$

となる。したがって、$d^2 f''(x_{\min})/2$ が $f(x_{\min})$ の精度（有効桁数）より小さい値であれば、$f(x_{\min} + d)$ の値は $f(x_{\min})$ の値と変わらないということになる。つまり、変数 x と関数 $f(x)$ および $f''(x)$ の値のオーダーが同じくらいであるときは、目安として $d / x_{\min}$ の値は計算機での精度の平方根ぐらいまでということになる。手続き MinByGolden でのパラメータ acc は、収束判定条件での相対値（$d / x_{\min}$ に対応する値）を指定するものである。Extended 型の有効桁数は１９～２０桁なので、精度の平方根は1.0ｅ－9 ぐらいということになる。したがって、acc の値は1.0ｅ－9 ぐらいが計算における有効桁数を考慮した場合の限界と考えられる。この精度の限界は、式（４）から分かるように、$f(x_{\min})$ と $f''(x_{\min})$ の大きさにも依存しているので、個々の関数によって決まるものである。

関数 $y = 700 + x(x-6)(x-10)^2$ の場合（式（１））について、上のことを調べてみる。この関数は、最小値を $x = 2$ でとる。この関数を $x = 2$ および $x = 5$ の値を中心に相対値 1.0ｅ－11 の大きさで変化させたときの関数値の変化を調べてみる。

関数を

```
function  f( x : extended ) : extended;
  begin
    f := 700 + x*(x-6)*sqr(x-10);
  end;
```

と宣言して、次のように関数値ｆ(ｘ)の変化を調べてみる。

```
x0:=2.0;
for i:=-5 to 5 do
  begin
      x:=x0+i*(1.0e-11);
      v:=f(x);
      writeln(outf,'x = ',x:25:19,
                   '        v = ',v:25:19);
  end;
writeln(outf);
x0:=5.0;
for i:=-5 to 5 do
```

```
begin
    x:=x0+i*(1.0e-11);
    v:=f(x);
    writeln(outf,'x = ',x:25:19,
                 '        v = ',v:25:19);
end;
```

上の実行結果は、リスト１のようになる。

リスト１　変数ｘの変化と関数値の変化。

```
x =     1.9999999999500000000        v =   188.0000000000000000000
x =     1.9999999999600000000        v =   188.0000000000000000000
x =     1.9999999999700000000        v =   188.0000000000000000000
x =     1.9999999999800000000        v =   188.0000000000000000000
x =     1.9999999999900000000        v =   188.0000000000000000000
x =     2.0000000000000000000        v =   188.0000000000000000000
x =     2.0000000000100000000        v =   188.0000000000000000000
x =     2.0000000000200000000        v =   188.0000000000000000000
x =     2.0000000000300000000        v =   188.0000000000000000000
x =     2.0000000000400000000        v =   188.0000000000000000000
x =     2.0000000000500000000        v =   188.0000000000000000000

x =     4.9999999999500000000        v =   574.9999999925000000000
x =     4.9999999999600000000        v =   574.9999999940000000000
x =     4.9999999999700000000        v =   574.9999999955000000000
x =     4.9999999999800000000        v =   574.9999999970000000000
x =     4.9999999999900000000        v =   574.9999999985000000000
x =     5.0000000000000000000        v =   575.0000000000000000000
x =     5.0000000000100000000        v =   575.0000000015000000000
x =     5.0000000000200000000        v =   575.0000000030000000000
x =     5.0000000000300000000        v =   575.0000000045000000000
x =     5.0000000000400000000        v =   575.0000000060000000000
x =     5.0000000000500000000        v =   575.0000000075000000000
```

関数が最小値をとる変数の値ｘ＝２においては、Extended 型の有効桁数（１９～２０桁）の平方根より小さい相対量（1.0ｅ－11）の変化では関数値は変わらない。しかし、同じ相対量（1.0ｅ－11）の変化でも、ｘ＝５を中心とする変化では関数値の方も変化している。

式（１）の関数の最小値を求めるときの変数の変化量は、Extended 型の場合、相対量で 1.0ｅ－11 程度では変数値の変化量は小さすぎるということになる。

リスト１は、プログラム PCkAcc.dpr の実行によって得ることができる。実行開始時のフォーム上のＧＯボタンをクリックすると計算結果を書き出すためのファイル名の設定を求めるダイアログボックスが表示される。このファイルはテキストファイルなので、プログラムの実行終了後、エディタで開いて見ることができる。ダイアログボックスにファイル名を設定して「開く」ボタンをクリックすると計算が始まる。

手続き MinByGolden によって関数の最小値を与える変数値 x_{min} を求めるときは、まず手続き MinOneDim によって x_{min} の存在区間を十分に絞って、その区間内で関数が最小値以外の極小値をとることがないようにする。その区間内で手続き MinByGolden による最小値の探索を行う。すなわち、

```
MinOneDim(-10.0, 20.0, 100, f, L_x, U_x );
```

を実行してから

```
MinByGolden( L_x, U_x, f, 1.0e-9, 1.0e-19, opt_x );
```

によって x_{min} を求める。上の場合、この x_{min} の値は opt_x に返される。

上の方法の例が、プロジェクト PGoldenSection.dpr である。プロジェクト PGoldenSection.dpr を実行して表示されるフォームのＧＯボタンのクリックで計算が始まり、図５のように結果が表示される。

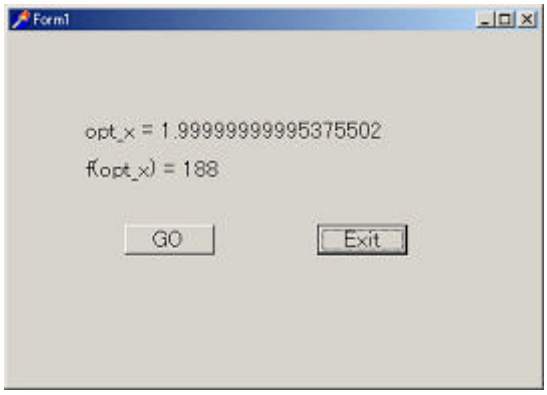

図５　プロジェクト PGoldenSection.dpr の実行結果

Quadratic Interpolation 法 (1)

Golden section 法は、区間内で関数が下に凸であれば、確実に関数が最小値をとる変数値を求めることができる。しかし、区間内で関数を多項式で近似するともっと効率

よく最小値を求めることができる場合がある。これは、根を求めるとき、Bisection 法は確実であるが、より効率の良い方法として関数を接線で近似して根を求める Newton 法があることと似ている。

Quadratic interpolation 法では、関数 $y = f(x)$ を２次関数で近似して最小値を与える変数値を求める。２次関数は、3 点での関数値を与えて求める。

３点、t_A、t_B、t_C、における関数値を f_A、f_B、f_C とする。すなわち、

$$f_A = f(t_A)、\qquad f_B = f(t_B)、\qquad f_C = f(t_C)$$

とおく。さらに、３点、(t_A, f_A)、(t_B, f_B)、(t_C, f_C)、を通る２次関数を

$$y = h(l) = a + b\,l + c\,l^2 \qquad (5)$$

とおく。このとき

$$a = \frac{f_A t_B t_C (t_C - t_B) + f_B t_C t_A (t_A - t_C) + f_C t_A t_B (t_B - t_A)}{(t_A - t_B)(t_B - t_C)(t_C - t_A)}$$
$$b = \frac{f_A (t_B^2 - t_C^2) + f_B (t_C^2 - t_A^2) + f_C (t_A^2 - t_B^2)}{(t_A - t_B)(t_B - t_C)(t_C - t_A)}$$
$$c = -\frac{f_A (t_B - t_C) + f_B (t_C - t_A) + f_C (t_A - t_B)}{(t_A - t_B)(t_B - t_C)(t_C - t_A)}$$

となる。

２次関数（５）は、

$$l_{\min} = -\frac{b}{2c} \qquad (6)$$

のときに最小値

$$h_{\min} = -\frac{b^2}{4c} + a \qquad (7)$$

をとる。

２次関数（５）式による近似を用いると、（６）式で与えられる変数値 $x = l_{\min}$ が最小値を与える値の推定値となる。この推定は、３点の区間 $[t_A, t_C]$ の幅が小さいほどよい結果が期待される。よって、$l_{\min}$ の値を用いて３点を含む区間の幅を小さくすることを考える。関数 $y = f(x)$ の $x = l_{\min}$ における値 $f_{L\_\min} = f(l_{\min})$ と f_B を比較して、区間 $[t_A, t_C]$ を区間 $[t_A, t_B]$ または $[t_B, t_C]$ に絞る。この最小値をとる区間を小さく絞っていく

手続きを繰り返して、$y=f(x)$が最小値をとる変数値$x=x_{\min}$を求める。具体的には次のような手順になる。

（Ⅰ）関数$y=f(x)$が最小値をとる区間を$[t_A,t_C]$とする。この区間は、手続き MinOneDim などによって求めることができる。$t_B=(t_A+t_C)/2$とおき、$f_A=f(t_A)$、$f_B=f(t_B)$、$f_C=f(t_C)$とおく。

（Ⅱ）3点、(t_A,f_A)、(t_B,f_B)、(t_C,f_C)、を通る2次関数を求め、その最小値$h_{\min}$と最小値を与える変数値$l_{\min}$を式（7）と（6）によって求める。

（Ⅲ）（Ⅲａ）$f_{L_\min}=f(l_{\min})$が$h_{\min}$に十分に近い値ならば、$x=l_{\min}$を

関数$y=f(x)$の最小値$f_{L_\min}$を与えるものとして終了する。

（Ⅲｂ）$f_{L_\min}=f(l_{\min})$が$h_{\min}$に十分近い値でないならば、次のステップ（Ⅳ）に進む。

（Ⅳ）（Ⅳａ）$l_{\min}>t_B$のとき

（Ⅳａ－1）$f_{L_\min}<f_B$ならば$t_B<x_{\min}<t_C$と考えて、t_B、$l_{\min}$、t_Cを次のステップでの新しいt_A、t_B、t_Cの値とする。これに合わせて、f_B、$f_{L\_\min}$、f_Cを次のステップでの新しいf_A、f_B、f_Cの値とする。ステップ（Ⅱ）に戻る。

（Ⅳａ－2）$f_{L_\min}>f_B$ならば$t_A<x_{\min}<l_{\min}$と考えて、t_A、t_B、$l_{\min}$を次のステップでの新しいt_A、t_B、t_Cの値とする。これに合わせて、f_A、f_B、$f_{L_\min}$を次のステップでの新しいf_A、f_B、f_Cの値とする。ステップ（Ⅱ）に戻る。

（Ⅳｂ）$l_{\min}<t_B$のとき

（Ⅳｂ－1）$f_{L_\min}<f_B$ならば$t_A<x_{\min}<t_B$と考えて、t_A、$l_{\min}$、t_Bを次のステップでの新しいt_A、t_B、t_Cの値とする。これに合わせて、f_A、$f_{L\_\min}$、f_Bを次のステップでの新しいf_A、

f_B、f_C の値とする。ステップ（Ⅱ）に戻る。

（Ⅳb－2）$f_{L\_\min} > f_B$ ならば $l_{\min} < x_{\min} < t_C$ と考えて、$l_{\min}$、t_B、t_C を次のステップでの新しい t_A、t_B、t_C の値とする。これに合わせて、$f_{L\_\min}$、f_B、f_C を次のステップでの新しい f_A、f_B、f_C の値とする。ステップ（Ⅱ）に戻る。

（Ⅱ）～（Ⅳ）のステップを繰り返して $x_{\min}$ を求めるが、この手順がうまく行かないときは Golden section 法に切り替える。

ステップ（Ⅲ）において $f_{L\_\min} = f(l_{\min}) = h_{\min}$ ならば、点 $(l_{\min}, f_{L\_\min})$ はステップ（Ⅱ）で求めた２次関数上にあるので、このときステップ（Ⅳ）に進んで新しく３点 t_A、t_B、t_C を取り直して２次関数を求め直したとしても前の古い２次関数と同じものになる。したがって、この新しく求めた２次関数の $l_{\min}$ は古いものと同じ値となり、$l_{\min}$ の値は更新されず、３点 t_A、t_B、t_C（このうちの一つが古い $l_{\min}$ である）のいずれかに一致する。このことを考慮して、ステップ（Ⅲ）における収束判定条件として「$f_{L\_\min} = f(l_{\min}) = h_{\min}$」を採用している。

上の方法で $x_{\min}$ を求める手続き MinByQuad もユニットファイル UOptOneDim.pas に用意した。手続きヘッダーは次のようになっている。

```
procedure  MinByQuad( L_b, U_b  : extended;
                      f         : TFunc;
                      accf      : extended;
                      var
                        opt_x   : extended );
```

パラメータは Golden section 法の手続き MinByGolden の場合と同様であるが、ステップ（Ⅲ）での収束判定条件に用いる、関数値 $f_{L\_\min} = f(l_{\min})$ と近似に用いた２次関数の値 $h_{\min}$ の差に関する基準を、精度の基準値 accf として設定するようになっている。

手続き MinByQuad において Quadratic interpolation 法がうまく行かないときは Golden section 法が適用されるが、これらの計算の途中経過は、手続き MinByQuad 内でコメントとしてあるフォーム FOptOneDim の生成などの箇所のコメント記号、{ } や //など、をはずして Memo コンポーネントへの表示や ShowMessage 手続きが実行される

ようにすることによって見ることができる。

手続き MinByQuad による式（１）で与えられる関数の最小値を与える変数値の求め方は、Golden section 法の手続き MinByGolden による方法と同じようになる。MinByGolden の呼び出し

```
MinByGolden( L_x, U_x, f, 1.0e-9, 1.0e-19, opt_x );
```

を、次のように MinByQuad の呼び出し

```
MinByQuad( L_x, U_x, f, 1.0e-17, opt_x );
```

に変更する。収束判定条件に使われる精度は、変数値についてのものではなく、関数値に関するものなので、Extended 型の精度に近い 1.0e-17 を設定している。

この MinByQuad の使用例をプロジェクト PQuadratic.dpr として用意した。上の MinByQuad による計算結果は、図６のように表示される。

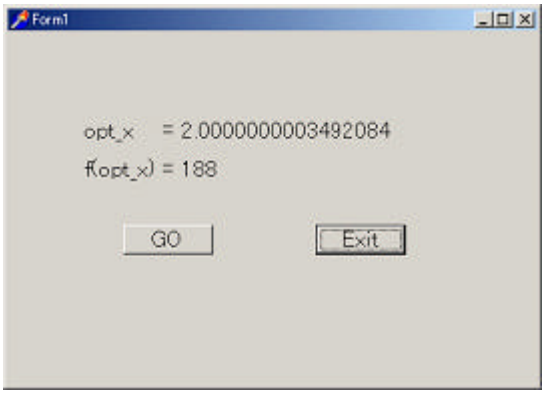

図６　プロジェクト PQuadratic.dpr の実行結果

地震警報の基準

最小値を求める例として、地震の警報を出す基準値の設定について考えてみる。この考え方は、地震警報に限らず、何らかの曖昧さを伴う事態についての判断・決断をモデル化したものである。

地震計の針の振れ具合とか地面の動き方とか、地震の予知に用いられる指標は、これくらいの強さなら確実に地震が起きる、あるいは絶対に地震は起きない、というようなものではないようである。いま、予知で用いられる指標を簡略化して１つの数値 Ix で表わされているとする。この Ix は、地震の心配がないときは小さい値であるが、地震が起きるときは大きな値をとるものとする。しかし、Ix にはノイズが含まれていて、Ix の値はランダムに変動しているものとする。地震の心配がないときでも Ix の値は大

きくなることもあり、地震が起きるのに Ix の値は小さいということもありえるとする。つまり、Ix の値によって確実に地震の予知を行うことができないということである。地震が起きないときの Ix の値は平均０、分散１の正規分布に従い、地震が起きるときの Ix の値は平均 3.5、分散１の正規分布に従っているとする。したがって、Ix の値が -1 などのように小さいときは地震の起きる可能性は小さく、Ix の値が６などのように大きい値のときは地震の起きる可能性は高いと考えられる。しかし、Ix が２ぐらいのときは、地震が起きるのか起きないのかはっきりしない。地震予知の警報を出すためには、基準値 c を設定しておいて、Ix の値が c より大きくなったら警報を出すというように決めておく必要がある。c をどのくらいの値に設定しておくとよいのであろうか？ c の値が小さすぎるときは、本当は地震の起きる可能性はないに等しいが（安全のために）警報を出しておいた、ということが頻繁に生じるようになり迷惑なことになる。逆に c の値が高すぎると、地震が起きるのに警報を出さなかったということになる危険が高くなる。

地震が起きないときに警報を出すと、生活に混乱が生じてそれに伴う損失が予想される。この損失の大きさが何か１つの数値で表わされるものとして、それを V(警報有り| 地震無し) で表わしておく。地震が起きないときに警報を出さない場合は、生活が普通に行われ生産的な活動が営まれる。その生産的な活動の効用も１つの数値 V(警報無し| 地震無し) で表わすが、損失（地震の被害）の方を正の値で表わすと効用（生産的な活動の成果）の方は負の値で表わすことになる。

地震が起きないときに Ix の値が c より小さくなる（警報を出さない）確率を $\Pr ob$(警報無し| 地震無し) で表わすと、c より大きくなる（警報を出す）確率は $1-\Pr ob$(警報無し| 地震無し) となる。したがって、この場合の損失と生産的な活動の成果をそれぞれの確率で重み付けた和である期待値 EV(地震無し) は

$$EV(\text{地震無し}) = \Pr ob(\text{警報無し}|\,\text{地震無し})V(\text{警報無し}|\,\text{地震無し}) + \{1-\Pr ob(\text{警報無し}|\,\text{地震無し})\}V(\text{警報有り}|\,\text{地震無し})$$

となる。

地震が起きる場合も同様に考えて、

$$EV(\text{地震有り}) = \Pr ob(\text{警報無し}|\,\text{地震有り})V(\text{警報無し}|\,\text{地震有り}) + \{1-\Pr ob(\text{警報無し}|\,\text{地震有り})\}V(\text{警報有り}|\,\text{地震有り})$$

となる。地震が起きる場合は、警報があっても被害があるわけであるが、警報が無いときは非常に大きな被害が出ることになる。

Ix の値を考えない場合、すなわち予知のための何らの情報も無いときの地震の起き

る確率を Pr*ob*(地震) で表わすと、警報を出す Ix の基準値を c に決めた場合の全体的な被害（生産的なものは負の値で表わす） $EV(c)$ は、

$$EV(c) = \Pr ov(\text{地震})EV(\text{地震有り}) + (1 - \Pr ov(\text{地震}))EV(\text{地震無し})$$

で表わされる。

$EV(c)$ は c の関数になっている。この $EV(c)$ が最小になるように c を決めるのがよいと考えられる。この c を求めるプログラム例が PExpect.dpr である。$EV(c)$ の最小値を与える c の値を、手続き MinByQuad を用いて Quadratic interpolation 法によって求めている。

PExpect.dpr の実行開始時に表示されるフォームのＧＯボタンをクリックすると、図７のようなフォームが表示される。

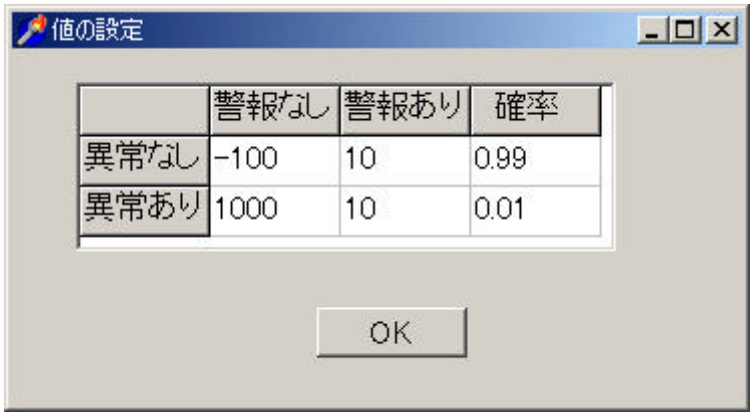

図７　値設定用フォームの表示

「異常なし」の行が地震が起きない場合を表わす。「警報なし」の列に V(警報無し| 地震無し) の値を、「警報あり」の列に V(警報有り| 地震無し) の値を、確率の列に１－Pr*ob*(地震) の値を設定する。確率の値は、和が１にならない値を設定しても、ＯＫボタンのクリック後、設定した値の比率で和が１になるように調整される。図７のフォームでの V(警報無し| 地震無し) の値は、地震も警報も無い場合は生活・経済が正常に営まれて社会的な活動が正の生産である、すなわち、負の被害であると考えてー１００と負の値が設定されている。

「異常あり」は地震が起きる場合の値である。地震が起きるのに警報が無いと大きな被害が生じるとして１０００が設定されている。

確率を設定する列には「異常あり」に対して 0.01 を設定して、地震の起きる（主観的）確率を地震が起きない確率の約 1/100 としている。

図７の設定でＯＫボタンをクリックすると、計算結果が図８のように表示される。

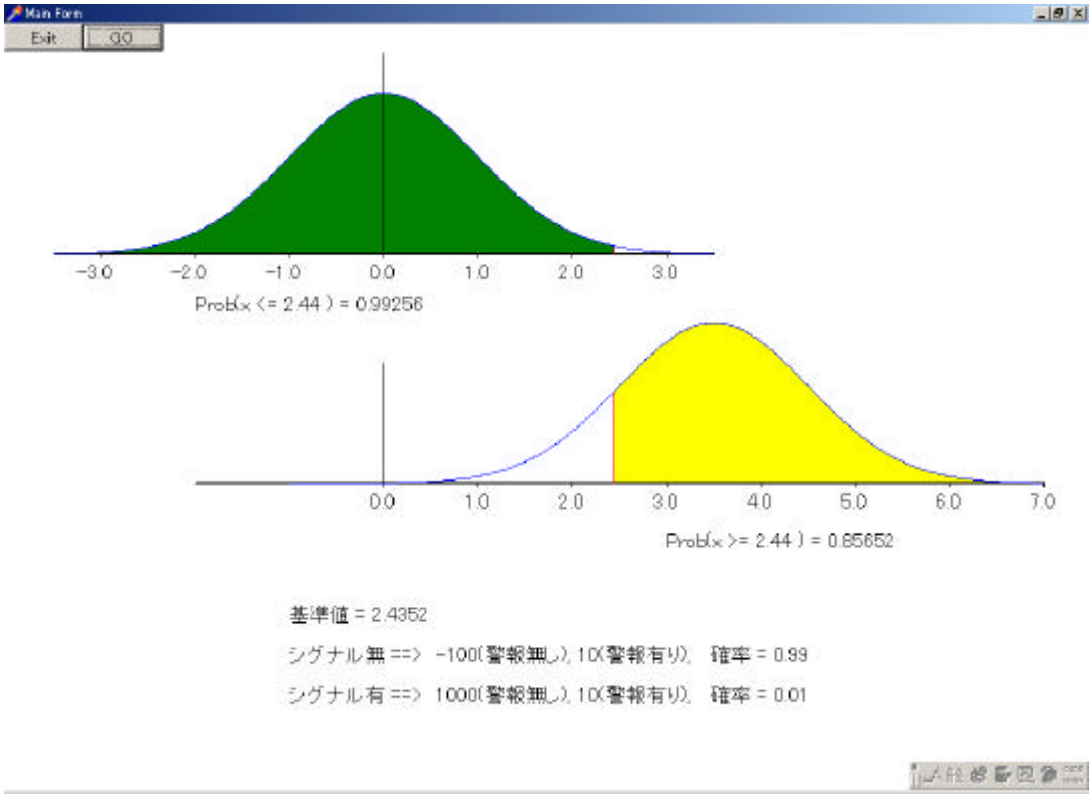

図８　基準値ｃと確率の表示

図８の画面において、上の分布のグラフが地震が起きないときの Ix の分布を、下の分布が地震が起きるときの Ix の分布を表わしている。図８では、それぞれの分布の $c \approx 2.44$ の位置に赤い縦線が引かれている。上の方の地震が起きないときの分布では「$Ix < c$」の領域が緑で塗り潰されている。この領域で表わされる確率が地震が起きないときに（すなわち、正しく）警報を出さない確率 Pr*ob*(警報無し|地震無し) になる。下の地震が起きるときの分布では「$Ix > c$」の領域が黄色で塗り潰されている。この領域の確率は Pr*ob*(警報有り|地震有り) を表わす。図８の画面の下側には、求められた基準値ｃの値と、図７で設定された値が表示されている。

図８のフォームのＧＯボタンをクリックすると、再び図７の値設定用のフォームが表示される。値を設定し直してＧＯボタンをクリックすると、再設定した値での計算結果が表示される。

参 考 文 献

（1）S.S.Rao. Optimization: Theory and Applications (Second Edition). Pp.747, John Wiley & Sons, 1984.

（2）P.E.Gill, W.Murray & M.H.Wright. Practical optimization. Pp.401, Academic Press, 1981.

（3）W.H.Press, B.P.Flannery, S.A.Teukolsky & W.T.Vetterling. Numerical recipes in Pascal. Pp.759, Cambridge University Press, 1989.