岡本安晴　2001.1.29；2001.8

積 分 の 計 算[1]

定積分$\int_a^b f(x)dx$、および２重積分$\int_a^b\left(\int_{g(x)}^{h(x)} f(x,y)dy\right)dx$の計算について考える。

シンプソンの公式

定積分$\int_a^b f(x)dx$の値を求める方法として、区間$[a,b]$を２N 等分したときの各分点を隣合う３点ごとにまとめてこれらの３点を通る２次関数で$f(x)$を近似し、それらの２次関数の積分値の和として求めるものがある。この、定積分の値を、被積分関数を近似する２次関数の積分の和として求める方法は、シンプソンの公式と呼ばれている[1)、2)、3)、4)]。

２N 個の分点を、$a=x_0$、x_1、・・・、x_{2N-1}、$x_{2N}=b$、とする。

$$x_i = a + i\cdot(b-a)/2N \quad ; \quad i=0,\cdots,2N$$

となっている。

３分点、x_{2j}、x_{2j+1}、$x_{2(j+1)}$、における関数値を、$y_{2j}=f(x_{2j})$、$y_{2j+1}=f(x_{2j+1})$、$y_{2(j+1)}=f(x_{2(j+1)})$、とおき、３点、(x_{2j},y_{2j})、(x_{2j+1},y_{2j+1})、$(x_{2(j+1)},y_{2(j+1)})$、を通る 2 次関数を$g_j(x)$とおく。このとき、

$$\int_{x_{2j}}^{x_{2(j+1)}} g_j(x) = \frac{h}{3}(y_{2j}+4y_{2j+1}+y_{2(j+1)})$$

となる。ここで、

$$h=(b-a)/(2N)$$

である。

上式は、以下のようにして導くことができる。

まず、２次関数$g_j(x)$を次のようにおく。

---

[1] この解説は、TRY!PC,2000 年６月号「Delphi による数値計算プログラミングのすすめ：第４回積分の計算」の原稿をもとにしたものである。

$$g_j(x) = y_{2j}\frac{(x-x_{2j+1})(x-x_{2(j+1)})}{(x_{2j}-x_{2j+1})(x_{2j}-x_{2(j+1)})}$$

$$+ y_{2j+1}\frac{(x-x_{2j})(x-x_{2(j+1)})}{(x_{2j+1}-x_{2j})(x_{2j+1}-x_{2(j+1)})}$$

$$+ y_{2(j+1)}\frac{(x-x_{2j})(x-x_{2j+1})}{(x_{2(j+1)}-x_{2j})(x_{2(j+1)}-x_{2j+1})}$$

上の$g_j(x)$が３点、(x_{2j}, y_{2j})、(x_{2j+1}, y_{2j+1})、$(x_{2(j+1)}, y_{2(j+1)})$、を通ることは容易に確かめられる。

変数変換

$$x = t + x_{2j+1}$$

を行って$g_j(x)$の積分を計算すると、

$$\int_{x_{2j}}^{x_{2(j+1)}} g_j(x)dx = \int_{-h}^{h} g_j(t+x_{2j+1})dt$$

$$= y_{2j}\int_{-h}^{h}\frac{t\cdot(t-h)}{(-h)\cdot(-2h)}dt + y_{2j+1}\int_{-h}^{h}\frac{(t+h)(t-h)}{h\cdot(-h)}dt$$

$$+ y_{2(j+1)}\int_{-h}^{h}\frac{(t+h)\cdot t}{2h\cdot h}dt$$

$$= y_{2j}\cdot\frac{1}{2h^2}\cdot\left\{\left(\frac{h^3}{3}+\frac{h^3}{3}\right)-h\left(\frac{h^2}{2}-\frac{h^2}{2}\right)\right\}$$

$$- y_{2j+1}\cdot\frac{1}{h^2}\cdot\left\{\left(\frac{h^3}{3}+\frac{h^3}{3}\right)-h^2(h+h)\right\}$$

$$+ y_{2(j+1)}\cdot\frac{1}{2h^2}\cdot\left\{\left(\frac{h^3}{3}+\frac{h^3}{3}\right)+h\left(\frac{h^2}{2}-\frac{h^2}{2}\right)\right\}$$

$$= y_{2j}\cdot\frac{h}{3} - y_{2j+1}\cdot\left(-\frac{4h}{3}\right) + y_{2(j+1)}\cdot\frac{h}{3}$$

$$= \frac{h}{3}(y_{2j} + 4y_{2j+1} + y_{2(j+1)})$$

となる。

被積分関数 $f(x)$ の２次関数 $g_j(x)$ による近似を用いた積分により、

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{j=0}^{N-1}\int_{x_{2j}}^{x_{2(j+1)}} f(x)dx \\ &\approx \sum_{j=0}^{N-1}\int_{x_{2j}}^{x_{2(j+1)}} g_j(x)dx \\ &= \sum_{j=0}^{N-1}\frac{h}{3}(y_{2j}+4y_{2j+1}+y_{2(j+1)}) \\ &= \frac{h}{3}\left(y_0+y_{2N}+4\sum_{j=0}^{N-1}y_{2j+1}+2\sum_{j=1}^{N-1}y_{2j}\right) \qquad (1)\end{aligned}$$

となる。ここで、記号≈は近似式であることを表わす。

上式（１）はシンプソンの公式と呼ばれている。誤差は

$$\int_a^b f(x)dx-\frac{h}{3}\left(y_0+y_{2N}+4\sum_{j=0}^{N-1}y_{2j+1}+2\sum_{j=1}^{N-1}y_{2j}\right)$$

$$=-\frac{(b-a)\cdot h^4}{180}\cdot f^{(4)}(m)$$

となる[2]。ここで、$a<m<b$ である。

関数 $f(x)$ と区間 $[a,b]$ に対して、$f^{(4)}(m)$ の値を無視すると、誤差は

$$h^4=\left(\frac{b-a}{2N}\right)^4=\left(\frac{b-a}{2}\right)^4\cdot\left(\frac{1}{N}\right)^4$$

で決まり、$(1/N)$ の４乗の関数になっている。Ｎを２倍にすると誤差は $(1/2)^4=1/16$、すなわち１６分の１になることが期待される。

このように、分点の数２N によって（１）式による積分の値の精度は変わる。適当な N の値から始めて十分な精度が得られるまで N の値を順番に大きくしていく。（１）式において N を 2 倍にすると始めの分点の関数値 y_j は全て偶数番目の分点の値となり、新しく加えられた分点は奇数番目のものとなる。したがって、N を２倍にする前に求めた関数値を（１）式による再計算で使うことができ、２倍にしたときは奇数番目の関数値のみを計算すればよいことになる。

さらに、（１）式を見ると、$S_1=\sum_{j=0}^{N-1}y_{2j+1}$ は奇数番目の分点の関数値の和であり、

$S_2 = \sum_{j=1}^{N-1} y_{2j}$ は偶数番目の分点の関数値の和になっている。つまり、N を２倍にする前の S_1 と S_2 の値を足せば N を２倍にしたときの S_2 の値となり、N を２倍にしたときの S_1 の値だけを直接関数値の計算によって求めればよいことになる。

上の方法でシンプソンの公式により定積分の値を求める関数を Simpson として用意した。関数 Simpson はユニットファイル UIntegral.pas に宣言されている。

関数 Simpson のヘッダーは

```
function Simpson( a, b : extended;
                  f    : TIntegralFunc;
                  acc, vzero  : extended ) : extended;
```

となっている。第１，２パラメータ、ａとｂ、に積分範囲の下限と上限を設定する。第３パラメータｆは被積分関数を指定する。手続き型 TIntegralFunc はユニット UIntegral において

```
type TIntegralFunc = function( x : extended ) : extended;
```

と宣言されている。被積分関数のヘッダーは、この手続き型に合わせて宣言する。

例えば、

```
function f( a : extended ) : extended;
  begin
          f:=1/a;
  end;
```

とする。ヘッダーに現れる変数の識別子（名前）は、a でも x でも、あるいは他のものでもよい。ヘッダーにおけるパラメタリストの形が一致しておればよい。

上の関数ｆは、$\int_a^b 1/x \cdot dx$ を計算するときの被積分関数を与えるものである。

関数 Simpson の第４，第５パラメータ、acc と vzero、は積分値を求めるときの精度とゼロとみなす基準値を指定するものである。

（１）式においてNの値を２倍にしていくと（１）式の精度が上がるが、そのときの積分値の変化量の割合が acc より小さくなったところで十分な精度の値が得られたとして計算が終了する。また、算出した積分値の絶対値が vzero よ

り小さいときは、求める積分値は0であるとして計算を終了する。

関数Simpsonを用いるときは、ユニットUIntegralの使用を宣言しておいて、被積分関数を上の例のように宣言する。

$$\int_1^{10} 1/x \cdot dx = \log 10$$

の計算のときは

```
v:=Simpson( 1.0, 10.0, f, 1.0e-15, 1.0e-17 );
```

のようにSimpsonを呼出す。

プログラムトPSimpson.dprでは上の計算が行われている。このプロジェクトを実行して表示されるフォーム上のＯＫボタンをクリックすると、図１のように結果が表示される。Simpson によって求めた値と正しい値log10がラベルのCaptionに設定されて表示される。

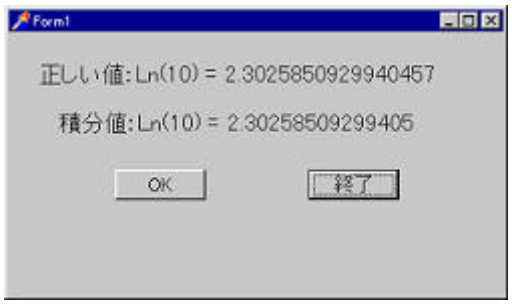

図１　定積分$\int_1^{10} 1/x \cdot dx$の計算

プログラム PSimpson.dpr は、ダウンロードしたファイルの解凍で作成されるフォルダintegralfilesにある。

適応的方法

シンプソンの公式を用いる上の方法では、算出した積分値の精度を上げるためにNの値を２倍にしている。しかし、精度を上げるためには、積分区間$[a,b]$で一様に分点の数を増やす必要はないこともある。２次関数による近似が不十分である領域においてのみ分点の数を増やせばよく、比較的少ない分点で十分によい近似が得られる領域においてまで分点の数を同じように増やすのは計算の無駄になる。適応的方法[2)、5)]では、分点の数を増やす必要のある領域においてのみ分点を増やし、増やす必要のない領域は分点の数を増やさない、ということによって不必要な分点の増加に伴う計算量の増加が避けられている。

Press ら[2)]の適応的方法では、N=1 のときのシンプソンの公式を基にして、積分区間の２分割が以下のように適応的に繰り返されている（図２）。

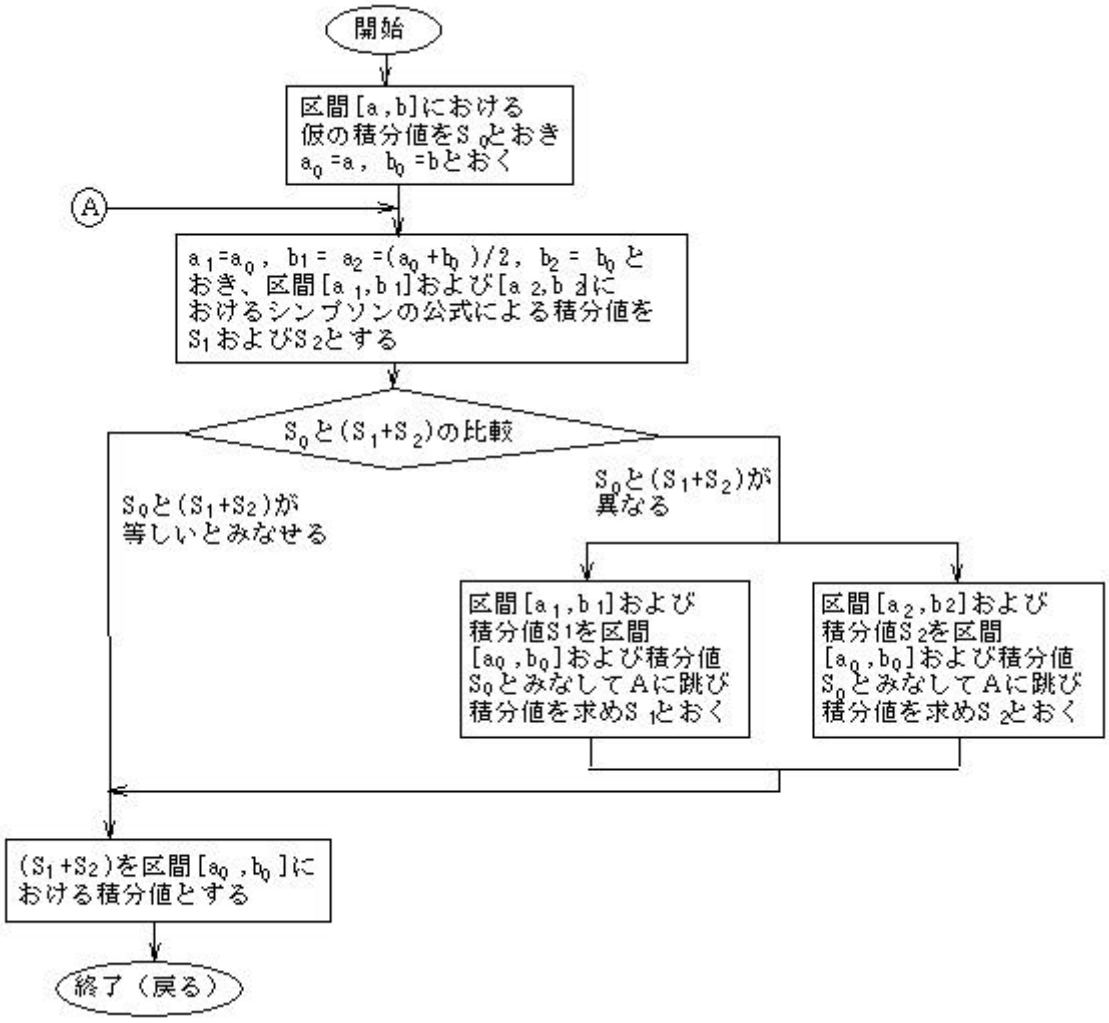

図２　シンプソンの公式（N=1）を用いた適応的方法

（１）区間$[a,b]$における仮の積分値をS_0とする。

（２）$a_0 = a$、$b_0 = b$とおき、区間I_0を$I_0 = [a_0, b_0]$とおく。

（３）$a_1 = a_0$、$b_1 = a_2 = (a_0 + b_0)/2$、$b_2 = b_0$とおき、区間$I_0$を 2 等分して$I_1 = [a_1, b_1]$および$I_2 = [a_2, b_2]$とおく。それぞれの積分区間$I_1$および$I_2$でのシンプソンの公式（Ｎ＝１）による積分値を$S_1$および$S_2$とおく。

（４）$S_1 + S_2$がS_0に十分近ければ$S_1 + S_2$を区間I_0における積分値とする。

十分に近くないときは、I_1およびI_2をそれぞれ$I_0=[a_0,b_0]$とおき、

（３）で算出されたS_1およびS_2をそれぞれのS_0として（３）に戻る。

上の適応的方法によって積分値を求める関数 AdaptiveQ をユニットファイル UIntegral.pas に用意した。関数 AdaptiveQ では、ステップ（１）でのS_0（初期値）を 0.0 とおいて、被積分関数の値の計算を省いている。これは、最初のS_0の値は不正確である可能性が高いので、関数値を求めて計算しても無駄になる可能性があるからである。

AdaptiveQ の関数ヘッダーは、次のように宣言されている。

```
function AdaptiveQ( a, b : extended;
                    f    : TIntegralFunc;
                    acc, vzero  : extended ) : extended;
```

パラメータは、関数 Simpson の場合と同じである。

定積分$\int_0^1 e^x dx$の値を求めるときは、被積分関数を例えば

```
function f( a : extended ) : extended;
  begin
        f := exp(a);
  end;
```

と宣言して、

```
v:=AdaptiveQ(0.0, 1.0, f, 1.0e-9, 1.0e-11 );
```

というように関数 AdaptiveQ を呼出す。

プログラム PAdaptiveQ.dpr では、上のようにして求めた値ｖに１を加えて、

$$v+1=\int_0^1 e^x dx+1=e^1-e^0+1=e$$

とeの値を求めている。この値は、フォーム上に「積分値+１」の値として表示

される（図３）。フォーム上にはeの値も表示される。

プログラム PAdaptiveQ.dpr はフォルダ integralfiles にある。

図３　$\int_0^1 e^x dx + 1 = e$ の計算

関数 AdaptiveQ などユニット UIntegral の関数を呼出したときは、UIntegral のフォームが表示される。このフォームには Memo コンポーネントが貼り付けられていて、計算の途中経過が表示されるようになっている。

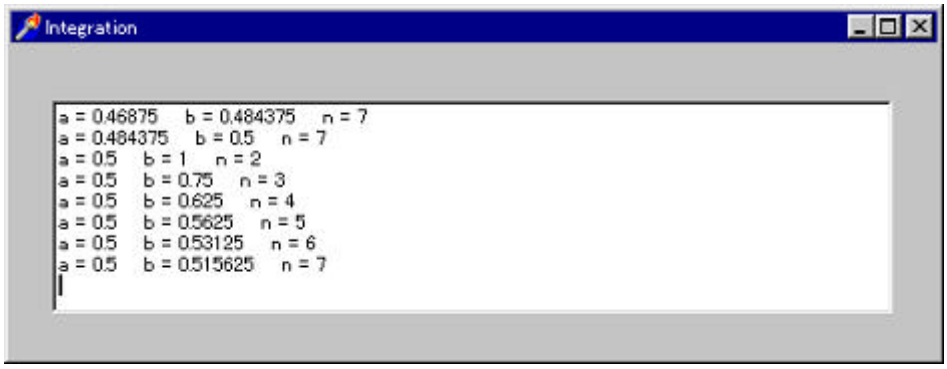

図４　計算の途中経過の表示

図４は、PAdaptiveQ.dpr の実行時に AdaptiveQ を呼出したときのものである。積分区間 $I_0 = [a_0, b_0]$ の下限と上限の値 a_0 と b_0 が、ａとｂの値として表示されている。ｎは再帰的呼び出しの深さを表す。

これらの途中経過の表示はかなりの実行時間を要するので、この表示が行われないようにすると計算時間は随分と速くなる。表示が行われないようにするには、この表示を行う手続き Display の実行の箇所をコメントとする。また表示が必要になったときは、コメント記号//などをはずせば再び表示させることができる。

また、Display を実行しないときは Memo コンポーネントの貼り付けられているフォームも必要ないので、このフォームの生成と廃棄の部分を

```
//  FIntegral:=TFIntegral.Create(application);
//  FIntegral.Visible:=true;
        •
        •
        •
//  FIntegral.Close;
```

というようにコメントとしておくと、関数 AdaptiveQ などの呼び出し時のユニット UIntegral のフォームの表示がなくなる。

ガウス・ルジャンドルの積分公式

シンプソンの公式（１）では、積分区間$[a,b]$の下限と上限での値、$f(a)$と$f(b)$、が用いられている。（半）無限区間での積分を変数変換によって有限区間$[a,b]$での積分に変換したときには、$f(a)$あるいは$f(b)$の値を与えることが難しいことがある。このような積分の計算では上限と下限での関数値を用いない積分公式が便利である。このような公式としてガウス･ルジャンドルの積分公式[1)、2)、4)、6)]

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f(\{(b-a)/2\}x_i + (b+a)/2)$$

がある。ここで、x_iはｎ次のルジャンドルの多項式$P_n(x)$の根、

$-1 < x_1 < \cdots < x_n < 1$であり、$w_i$は次式

$$w_i = \int_{-1}^{1} \frac{P_n(x)}{(x-x_i)P_n'(x_i)}dx$$

で与えられるものである。

ガウス・ルジャンドルの積分公式の誤差は

$$\frac{(n!)^4(b-a)^{2n+1}}{(2n+1)\{(2n)!\}^3} f^{(2n)}(x)$$

と表わされる[(1)]。積分区間$[a,b]$を２分すると、誤差は$(1/2)^{2n+1}$になることが期待される。$n=9$のときは、

$$(1/2)^{2n+1} = (1/2)^{19} \approx 10^{-6}$$

となる。

ユニットファイル UIntegral.pas の関数 Gauss_Legendre は、ｎ＝９の場合のガウス･ルジャンドルの積分公式によって定積分の値を求めるものである。ｎの値は９で十分な場合もあるが、関数と積分区間の組み合わせによってはｎ＝９では十分な精度が得られないことがある。十分な精度が得られないときは、積分区間を２分して、それぞれの区間においてｎ＝９の場合のガウス･ルジャンドルの積分公式によって積分値を求め直すことにする。この積分区間の２分を繰り返すという適応的方法によって所定の精度を得ることにする。このガウス・ルジャンドルの積分公式を基にした適応的方法によって積分の計算を行う関数を AdaptiveGL として UIntegral.pas に宣言した。関数のヘッダーは

```
function AdaptiveGL( a, b     : Extended;
                     f        : TIntegralFunc;
                     w_intvl  : Extended;
                     acc,       //    acc >= 1.0E-17
                     ZeroV
                              : Extended ) : Extended;
```

となっている。ａとｂに積分区間の下限と上限を設定する。ｆは被積分関数である。w_intvl は、適応的方法で積分区間を分割していくときの積分区間の長さの基準値である。積分区間の長さが w_intvl より小さくなるまで分割が進められる。これにより、関数値が最初の積分区間の狭い範囲で非ゼロである場合に、誤って積分値がゼロに収束したと判定されることを防ぐ。acc と ZeroV には、精度とゼロとみなす基準値を設定する。

AdaptiveGL を用いたサンプルプログラム PGL.dpr では、正規分布の累積確率を求めている。正規分布の累積確率

$$\Pr ob(X < z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2p}} \cdot e^{-0.5x^2} dx$$

$$= \frac{1}{\sqrt{2p}} \int_{-\infty}^{z} e^{-0.5x^2} dx$$

の計算のために、定積分 $\int_{-\infty}^{z} e^{-0.5x^2} dx$ の計算が関数 AdaptiveGL によって行われている。積分区間が半無限区間なので、次の変数変換を行っている。

$$x = \log t$$

このとき、

$$\int_{-\infty}^{z} e^{-0.5x^2}\,dx = \int_{0}^{\exp z} \exp(-0.5\times(\log t)^2)\frac{dt}{t} \qquad (2)$$

となる。

PGL.dpr のユニット UGL.pas では、関数宣言を

```
function NormalKernel( x : extended ) : extended;
  begin
        NormalKernel:=exp(-0.5*sqr(x));
  end;

function NormalKLN( t : extended ) : extended;
  begin
        NormalKLN:=NormalKernel(LN(t))/t;
  end;
```

と行って、（2）式の積分を

```
cump:=AdaptiveGL( 0.0, exp(z), NormalKLN, 0.5, 1.0e-9, 1.0e-15 );
```

と求めている。上の場合、w_intvl の値 0.5 は適当に設定したものである。正規分布の場合はこれでよいが、他の関数、例えばベータ分布の場合は極狭い区間に確率密度が集中することがある。このような場合には、確率密度の集中している区間の幅より小さい値を設定する必要がある[4)]。最後のパラメータ ZeroV は、適応的に積分区間を細分して行ったときの各積分区間において積分値がゼロであるかどうかの判定に使う基準値であるが、目安としては

$$ZeroV < \text{積分値の大体の予想値の絶対値})\times acc$$

である。

PGL.dpr を実行して表示されるフォーム上の GO ボタンをクリックすると、図５のような画面になる。画面上の適当な位置をクリックすると横軸上の位置に対応したｚの値が読み込まれ、確率 $Prob(X < z)$ が計算されて表示されるとともに、グラフの $Prob(X < z)$ に対応する領域が緑で塗り潰される。クリックのときの縦方向の位置は無視される。

プログラム PGL.dpr はフォルダ integralfiles にある

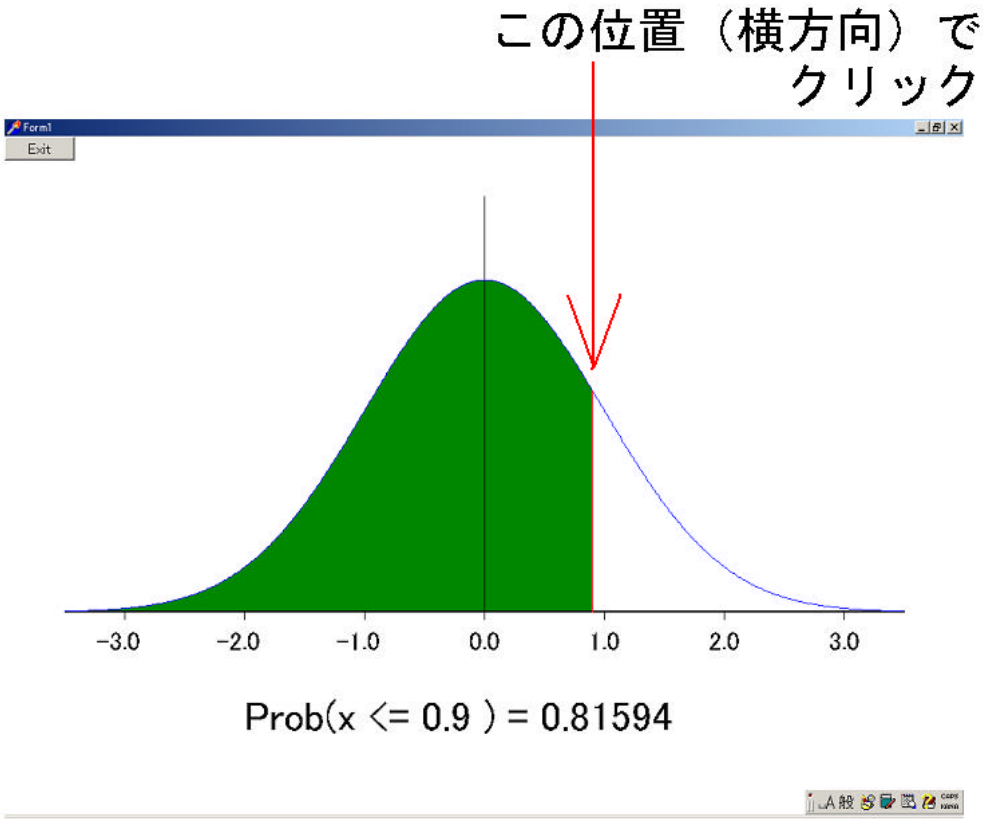

図５　画面のクリックで確率が計算される

２重積分（積分領域が矩形の場合）

ガウス・ルジャンドルの積分公式を 2 重積分

$$\int_a^b \int_c^d f(x,y)dydx \qquad (3)$$

に適用してみる[6)]。

まず、ｙに関する積分にガウス･ルジャンドルの積分公式を適用して

$$\int_c^d f(x,y)dy \approx \frac{d-c}{2}\sum_{j=1}^{n} w_j f(x,\{(d-c)/2\}y_j + (d+c)/2) \qquad (4)$$

を得る。ここで、y_j はガウス・ルジャンドルの積分公式における第ｊ番目の分点であり、w_j はその点に対する重みである。

（３）式と（４）式より

$$\int_a^b\int_c^d f(x,y)dydx \approx \int_a^b\left[\frac{d-c}{2}\sum_{j=1}^{n} w_j f(x,\{(d-c)/2\}y_j+(d+c)/2)\right]dx$$

を得る。上式における x に関する積分にガウス・ルジャンドルの積分公式を適用すると、

$$\int_a^b\left[\frac{d-c}{2}\sum_{j=1}^{n} w_j f(x,\{(d-c)/2\}y_j+(d+c)/2)\right]dx$$

$$\approx \frac{b-a}{2}\sum_{i=1}^{n} w_i\left[\frac{d-c}{2}\sum_{j=1}^{n} w_j f(\{(b-a)/2\}x_i+(b+a)/2,\{(d-c)/2\}y_j+(d+c)/2)\right]$$

$$=\frac{(b-a)}{2}\cdot\frac{(d-c)}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(\{(b-a)/2\}x_i+(b+a)/2,\{(d-c)/2\}y_j+(d+c)/2)$$

となる。

以上より、

$$\int_a^b\int_c^d f(x,y)dydx$$

$$\approx \frac{(b-a)}{2}\cdot\frac{(d-c)}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(\{(b-a)/2\}x_i+(b+a)/2,\{(d-c)/2\}y_j+(d+c)/2) \quad (5)$$

を得る。

２重積分の場合も、設定した精度の条件が満たされるまで積分区間の分割を繰り返す。変数 x と y それぞれの区間を２分するので４つの積分領域に分割されることになる。

上のガウス・ルジャンドルの積分公式を２重積分に適用して（（５）式）適応的に積分値を求める方法で２重積分を計算する関数 AdaptiveGLDbl をユニットファイル UIntegral.pas に用意した。この関数のヘッダーは次のようになっている。

```
function AdaptiveGLDbl( xa, xb, ya, yb,
                        xw_intvl, yw_intvl,
                        acc, zerov  : Extended;
                        f   : TDIntegralFunc ) : Extended;
```

第１，第２パラメータ xa と xb は、被積分関数の第１変数ｘの積分区間[a,b]の下限ａと上限ｂを設定するものである。第３、第４パラメータ ya と yb には、

第２変数ｙの積分区間[*c*,*d*]の下限ｃと上限ｄを設定する。パラメータ xw_intvl と yw_intvl は、それぞれ変数ｘとｙの区間を分割していったときに、それぞれの小区間の長さの最大値の基準を設定する。パラメータ acc は精度、zerov はゼロとみなす計算値を指定するものである。

被積分関数は、最後のパラメータｆに指定する。型 TDInregralFunc は、次のように宣言されている。

```
type  TDIntegralFunc = function( x, y : Extended ) : Extended;
```

関数 AdaptiveGLDbl によって次の２重積分

$$\int_{-1}^{1}\int_{0}^{p/2}(x\sin y - ye^{x})dydx = (1/e - e)p^{2}/8 \qquad (6)$$

を計算するプログラムが PDAdapGL.dpr である。

被積分関数を

```
function CheckFV( x, y : extended ) : extended;
  begin
       CheckFV:=x*sin(y)-y*exp(x);
  end;
```

と宣言して、次のように AdaptiveGLDbl を呼出している。

```
v:=AdaptiveGLDbl( -1.0, 1.0, 0.0,pi/2, 0.5, 0.5,
                  1.0e-15, 1.0e-17, CheckFV );
```

プログラム PDAdapGL.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積分の計算が始まり、図６のように計算結果が表示される。積分の計算結果はフォーム上に積分値として表示されている。フォームには（６）式の右辺の値も正しい値として表示されている。

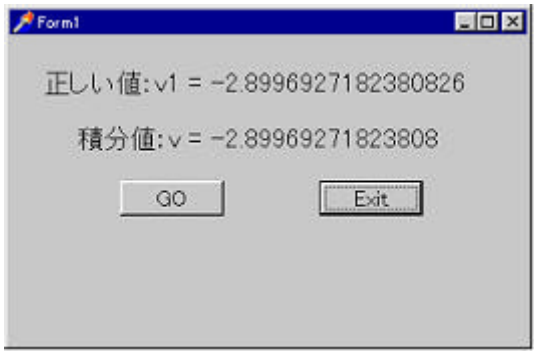

図６　$\int_{-1}^{1}\int_{0}^{p/2}(x\sin y - ye^{x})dydx$ の計算

２重積分（積分区間が矩形でない場合）

関数 AdaptiveGLDbl では、積分の領域が矩形になっている。次に、ｙの積分区間がｘの関数になっている

$$\int_{a}^{b}\int_{r_1(x)}^{r_2(x)} f(x,y)dydx \qquad (7)$$

の形の積分の計算について考える。

（７）式の場合は、

$$g(x)=\int_{r_1(x)}^{r_2(x)} f(x,y)dy \qquad (8)$$

とおいて、（８）式の計算と次の（９）式

$$\int_{a}^{b} g(x)dx \qquad (9)$$

の計算の２段階に分けて行う。（８）式および（９）式の計算は、いずれもガウス・ルジャンドルの積分公式を基とした適応的方法によって行う。（８）式の計算のときにｘを固定するが、このｘの値をオブジェクトのコンポーネントとして管理することにする。この方法によって（７）式の積分を計算するためのクラス型 TDInt を、ユニットファイル UIntegral.pas に宣言した。

クラス型 TDInt では、（８）式でのｘの値を保持するコンポーネント xg の他、被積分関数 $f(x,y)$ を保持する TDIntegralFunc 型のコンポーネント fd、ｙに関する積分区間を与える関数 r_1 および r_2 を保持する TIntegralFunc 型のコンポーネント YLB および YUB、および（８）式の関数 $g(x)$ を与えるメソッド f2 などが宣言されている。

この２重積分の計算を行うためのクラス型 TDInt のオブジェクトは、関数 DblGL の実行によって生成される。DblGL の関数ヘッダーは次のようになって

いる。

```
function DblGL( xa, xb : extended;
                GD : TDIntegralFunc;
                ry1, ry2 : TIntegralFunc;
                xw_intvl, yw_intvl,
                acc,                  //  >= 1.0e-16
                zero  : extended ) : extended;
```

第１、第２パラメータ xa と xb には、第１変数ｘの積分区間$[a,b]$の下限ａと上限ｂを設定する。第３パラメータには被積分関数 $f(x,y)$ を設定する。ry1 と ry2 には、第２変数ｙの積分区間を与える関数 r_1 と r_2 を設定する。xw_intvl および yw_intvl は、それぞれ変数ｘおよびｙの積分区間の分割における長さの最大値を与える基準値を設定する。適応的方法で積分区間が分割されるとき、分割された区間の長さがこの基準値より短くなるまで分割が繰り返される。acc と zero には、精度と計算値をゼロとみなす基準値を設定する。

次の積分

$$\int_0^1\left[\int_0^{\sqrt{1-x^2}}\sqrt{1-x^2-y^2}\,dy\right]dx=\frac{p}{6} \qquad (10)$$

の計算を行っているプログラム PDIntegralVY.dpr では、被積分関数および第２変数ｙの積分区間の下限と上限を与える関数が次のように宣言されている。

```
function CheckFV( x, y : extended ) : extended;
  begin
        CheckFV:=sqrt(1-sqr(x)-sqr(y));
  end;

function ryV1( x : extended ) : extended;
  begin
         ryV1 := 0.0;
  end;

function ryV2( x : extended ) : extended;
  begin
         ryV2 := sqrt(1-sqr(x));
  end;
```

上の宣言に基づいて２重積分（１０）式の値を求めるため、関数 DblGL が以下のように呼出されている。

```
v:=DblGL( 0.0, 1.0, CheckFV, ryV1, ryV2,
                        0.5, 0.5, 1.0e-11, 1.0e-14 );
```

関数 DblGL の実行により、TDInt 型のオブジェクトが生成されて２重積分の計算が行われる。TDInt 型のオブジェクトは自動的に生成・廃棄されるので、関数 DblGL の呼び出しにおいて、このオブジェクトの管理を意識する必要はない。

プログラム PDIntegralVY.dpr を実行して表示されるフォーム上のＧＯボタンをクリックすると上の関数 DblGL による計算が行われ、計算結果がフォーム上のラベルの Caption に設定・表示される。積分値である式（１０）の右辺の値$p/6$もフォーム上のラベルの Caption に正しい値として設定･表示される（図７）。

プログラム PDIntegralVY.dpr は、フォルダ integralfiles にある。

図７ 関数 DblGL による計算例

参 考 文 献

（1）川上一郎「数値計算」、Pp.201、岩波書店、1989.
（2）R.L.Burden and J.D.Faires. *Numerical Analysis, 3rd ed.*, Pp.676, PWS Publishers, 1985.
（3）岡本安晴「Delphi プログラミング入門」、Pp.207、CQ 出版株式会社、1997.
（4）岡本安晴「Delphi で学ぶデータ分析法」、Pp.274、CQ 出版株式会社、1998.
（5）日本数学会編集「岩波数学辞典」、第３版、Pp.1609、岩波書店、1985.
（6）W.H.Press, B.P.Flannery, S.A.Teukolsky and W.T.Vetterling. *Numerical recipes in Pascal*, Pp.759, Cambridge University Press, 1989.