X.6 XLISP-STAT to GAUSS の基礎　　ver.0.1

LISP の体系の中ではプログラムは、基本的に、丸括弧(　　)を何重も重ねていくリストと呼ばれる形をとっていきます。その上に統計学の関数が加えられて載っているのが XLISP-STAT です。なお、このベースになるフリーウエアの XLISP は 1988 年のもので、往年の PC9801 シリーズのファンにとっては、15 年も前に 9801 の上に移植された LISP がありましたが、基本的にそのころの仕様そのままです。当然、LISP としても動作可能です。見たこともない人にとっては、慣れるのには時間を要するかもしれませんが、パソコン黎明期に一度さわったことのある人は、実用はともかく、たいへん興味がわくことでしょう。実際に、細かい統計分野のプログラムを書くには適した構造になっています。

四則演算等

XLISP-STAT	GAUSS
(+ 1 2 3 4)	1+2+3+4
(- 10 3 2 1)	10-3-2-1
(/ 2 3)	2/3
(* 1 2 3 4)	1*2*3*4
(* (- 5 3) (+ 1 2))	(5-3)*(1+2)

配列やベクトルの場合には別の方法になりますが、XLISP-STAT の通常の一般的な操作では、簡単な四則演算もすべてリストの丸括弧の中で行ないます。そのリストの先頭に演算子を置きます。もちろん２つの数値の間の計算も可能ですが、最初に演算子を置いて、スペース（必ず必要）の後に３つ以上の数値をスペースで区切って置くことで一括計算できます。上の例では GAUSS でいうところの 1+2+3+4 を一括して計算しています。同様に、引き算もできます。割り算には XLISP-STAT も GAUSS も同様にスラッシュを使いますが出てくる答えは特にフォーマットをしない限り、割りきれないと XLISP-STAT では桁数いっぱいまで計算します。割りきれれば、整数部分だけが表示されます。GAUSS では割りきれても割りきれなくても常にデフォルトの桁数で表示されます。

リスト、ベクトルおよび行列

XLISP-STAT	GAUSS
(list 1 2 3 4)	x={1 2 3 4}; print x;

#(1 2 3 4)	x={1,2,3,4}; print x;
#2A((1 2)(3 4))	x={1 2, 3 4}; print x;

GAUSS ではすべての演算は基本的に行列として扱われ、{1,2,3,4}と表記すると、カンマごとに改行とみなして４×１の行列として解釈されます。すなわち、列ベクトルです。他方、カンマなしで{1 2 3 4}と表記すると１×４の行列、すなわち行ベクトルとなります。{　}括弧なしに x=1;などと表記すると１×１の行列、すなわちスケラーと解釈されます。それに対して XLISP-STAT では、通常の計算はリストによって行なわれることが主になります。これは列であるのか行であるのかはっきりとしませんが、私の解釈では、行として横に並んでいるものをそのまま計算していくスタイルをとります。もっとも、XLISP-STAT にもベクトルや行列の概念は存在していて、丸括弧の前に＃のマークをつけると１列ごとのベクトル、または＃のマークの前に２とか３とか数字がついているとその数だけ次元になります。通常は行の集まりからなる２次元、つまり通常の行列を表すことになります。なお、GAUSS ではセミコロン；のマークは命令ごとにつけ、１行に１つしか命令や定義がない場合にも、例外なく、セミコロンをつけることに注意してください。

行列計算

XLISP-STAT	GAUSS
; Comment	/* Comment */
	a=1;　@ Comment @

XLISP-STAT には行の概念があって、セミコロン；はその行の後にに何を書いても無視されてコメントをつけるのに用いられます。一方、GAUSS は行の概念がなくて、コメントは低レベル言語のように/*　　*/の内側に書きます。また、これを簡易的に@　　@で代用することができますが、これは@のマークは入れ子の構造にすることはできません。

行列の画面表示

XLISP-STAT	GAUSS
(print-matrix '#2A((1 2)(3 4)))	x={1 2, 3 4};
	print x;

XLISP-STAT では通常出力も１行ごとの形をとりますので、行列の場合、非常に見づらくなります。そこで、(print-matrix　　)という形で、その内側に行列にシングルクオートを前につけた形を起きます。（XLSIP-STAT ではリストの前のシングルクオートのマークはスペシャルフォーム(quate　　)と同じ働きをして、その後のリストの内容を計算せずにそのまま画面表示させます。）一方、GAUSS は最初からすべての数値は行列で考えられていますから、通常どおり print 文で扱えば、そのイメージのまま画面表示されます。

行列計算

XLISP-STAT	GAUSS
(def m #2A((1 2)(3 4)))	m={1 2, 3 4};
(transpose m)	m’;
(transpose '#2A((1 2)(3 4)))	
(inverse m)	inv(m);
(determinant m)	det(m);
(diagonal m)	diag(m);
(diagonal '(1 2 3))	v={1,2,3}; diagrv(zeros(3,3),v);

仮に、２×２の行列mが定義されているものとする。そこで、転置、逆行列、行列値、それに対角成分を求めるには、XLISP-STAT では、上のようにフルネームで指定する。また、直接行列を内部で代入するには、'#2A((1 2)(3 4))というようにアポストロフィーをつけて指定する。ただし、diagonal は行列の対角成分を求めるだけでなくて、ベクトルを零行列の対角成分に置きかえることもする。一方、GAUSS では、転置はアポストロフィー、逆行列は inv、行列値は det、対角成分は diag で計算する。なお、ベクトルを対角成分に置きかえることは、GAUSS では若干複雑で、あらかじめ零行列を指定しておき、そこに diagrv を用いて、その第１要素の零行列に第２要素のベクトルを流し込む形をとる。

配列の初期化

XLISP-STAT	GAUSS
(make-array ‘(2 2))	
(make-array '(2 2) :initial-element 0)	zeros(2,2);

なお、行列の初期化は XLISP-STAT では make-array によって行ない、その結果は行列の

すべての要素に NIL が入る。これをゼロにしようと思えば、:initial-element 0 と指定すればよい。この数字は任意である。GAUSS においては、通常、初期化は zeros によってゼロ行列を作成する。

変数定義

XLISP-STAT	GAUSS
(def y (list 5))	y=5;
(def x (list 1 2 3 4))	x={1,2,3,4};
x	print x;
(mean x)	print meanc(x);
(median x)	print median(x);
(standard-deviation x)	print stdc(x);

GAUSS では、変数名とイコールサイン＝を用いて後ろのものを前に定義します。それに対して、XLISP-STAT では(def　　)というマクロを使って変数とリストを結びつけて定義します。読んで字のごとく def つまり define です。(list 1 2 3 4)というものとその前の変数名例えばｘとかｙを結びつけて定義しています。(def　　)のマクロ命令の中に、変数名ｘとその後の(list 1 2 3 4)が入っているものと考えてください。このままでは、ｘと帰ってくるだけなので、変数の中味を表示するには、あらためてｘと入力します。一度定義してしまえば、消されたり変更されたりするまでは保存されていますから、(mean x)や(median x)などの形で各種組込み関数計算をします。ここで注意したいのは、GAUSS では、組込み関数のすべてが「例外なく」行ごとに計算することにあります。それを明確に表して誤解のないように mean や std の後に column の略のｃがついています。もっとも median のようにｃのマークがついていない組込み関数も「すべて」列ごとに計算されます。GAUSS の仕様の GAUSS らしいところはこの「列ごとの計算」にあります。これにより計量計算がデータの並びどおり楽に計算でます。ですから、x={1 2 3 4};などとカンマなしに１行として定義したものを関数計算すると、おのおのの列には１つずつの要素しかありませんから、その１つ１つの計算しかしてくれません。リストやベクトルという考え方と GAUSS の列ごとの計算は、はっきりと区別して考えるべきです。

関数定義

XLISP-STAT	GAUSS
(defun heikin(x)(/(sum x)(length x)))	fn(x)=sumc(x)/rows(x);
	proc heikin(x);
	local y;
	y=sumc(x)/rows(x);
	retp(y);
	endp;
(defun tasu(x y)(+ x y))	fn(x,y)=x+y;

上では、(defun 関数名 (パラメータ))によって「関数名」で「パラメータ」が引数の関数が定義されます。これにより、例えば、その後で(heikin x)というふうに通常の組込み関数と同じように呼び出せます。GAUSS では、これをリストの形式ではなくて、直接的に等号で１行で定義することができます。1 行定義で書く場合には関数に名前をつけることはできませんが、より一般的な proc で関数を表現するならば、local 宣言で内部で使う変数を宣言してから、計算式を書いてその内部で使う変数に代入したものをリターン命令で返す作業して、最後に endp;でそこまでが１つの関数のかたまりであることを示します。他方インプットが複数の場合には、XLISP-STAT ではただ単にパラメータのところをスペースを入れて列挙します。GAUSS の場合にはスペースではなくてカンマで引数を区切ります。

条件分岐

XLISP-STAT	GAUSS
(cond ((> x 0) x)	if x>0; x=x;
((= x 0) 0)	elseif x==0; x=0;
((< x 0) (- x))	elseif x<0; x=-x;
)	endif;

XLISP-STAT では if に相当する条件分は”(cond”から始まって丸括弧で終わります。複数条件がある場合でも、ただ単に関数 cond の後に条件のリストをいくつでも列挙して表記して

いきます。一方、GAUSS では、これは if～endif の一続きの条件クローズで対応させます。なお、複数の条件分岐の場合は elseif 文で続けていきます。ここでは示されていませんが、最後の条件がそれ以外の場合には、ただ単に else;で対応します。ここで注意すべきは、if 文などの最後にもすべてセミコロン；がついて、１文と考えることです。

シークエンス

XLISP-STAT	GAUSS
(iseq 2 10)	seqa(2,1,9)
(rseq 1 2 11)	seqa(1,0.1,11)

XLISP-STAT では始点と終点を示すことでシークエンスを形成します。上の答えは 2 から 10 までの整数となります。一方、GAUSS では(始点,増分,個数)となり、最後の要素は終点ではなく個数であることに注意してください。また、GAUSS では明示的に縦方向に列ベクトルとして結果が表示されます。整数の数列ではない場合には XLISP-STAT では rseq を用います。(rseq 始点　ステップ　個数)となり、GAUSS の seqa(始点,ステップ,個数)と構造的には同じになります。GAUSS では seqa で共通です。１ステップを省略することはできません。どちらも距離とその間に立てるポールの数の関係のように、10 の距離にあるポールの数は１を足した 11 になりますので注意してください。最後の要素は個数です。

繰り返し

XLISP-STAT	GAUSS
	i=1;
(dotimes (i 10)	do while i<=10;
(print i)	print i;
	i=i+1;
)	endo;

XLISP-STAT は、行列の配列と同じように、ここでも ｉ は０から始まる。したがって、上では０から９までの数が表示されることになる。一方、GAUSS では do while 文で書くと

するならば、do 文の外で初期値を設定して、自分で i を管理して i=い+1;で１つずつ増加させる。この場合は１から 10 までが表示される。XLISP-STAT と同じにするには、i=0;とあらかじめ設定しておいて、do while I<=9;としなければ結果は同じにならない。どちらも 10 回分の繰り返しをするので、 i の内容を計算に使わないのであれば特に気にすることはないだろう。GAUSS では if 文に対する endif 文と同様に、do 文は必ず endo 文でしめなければならない。

OLS 回帰

XLISP-STAT	GAUSS
(def x (list 1 2 3 4 5))	x={1,2,3,4,5};
(def y (list 2 4 6 8 9))	y={2,4,6.8.9};
(regression-model x y)	ols(0,y,x);

XLISP-STAT と GAUSS では独立変数と従属変数の順序が逆になっていることに注意してください。どちらも通常の関数として取り扱えます。ただしアウトプットは１変数ではありませんので変数で受けることはできません。

標本抽出

XLISP-STAT	GAUSS
(sample x 3)	
(sample x 100 t)	

XLISP-STAT 特有の関数として、つぼから玉やナンバーを取り出すときに使うような標本抽出の sample というのがあります。上は、ｘというリストから３個だけ非復元で取り出すもの。ｔを最後につけると、復元抽出で取り出すものです。GAUSS にはこの機能は標準になってはいません。基礎的な確率論などで重宝されるものです。

グラフ表示

XLISP-STAT	GAUSS

```
(def y (list 3 8 1 7 3 9 5))        y={3,8,1,7,3,9,5};
(def x (iseq 1 7))                   x=seqa(1,1,7);
                                     library pgraph; graphset;
(plot-lines x y)                     xy(x,y);
(plot-points x y)
```

==========

XLISP-STAT ではグラフは点を順番に結ぶ(plot-lines)と結ばない(plot-points)に分かれています。一方、GAUSS は、グラフ関係はすべて pgraph というライブラリを呼び出して行ないます。また、通常 graphset;としてそのグローバル変数をクリアする作業も行ないます。そのあとに、xy(x,y);とすれば点が順番に結ばれたグラフが描かれます。これを線なしにしたりドットのマークを変更するには、グローバル変数を変更します。

Reference

[1]竹村彰通『共立講座 21 世紀の数学 14　統計』 ISBN4-320-01566-5　定価 2678 円
付属プログラム http://www.e.u-tokyo.ac.jp/~takemura/kyoritsuprogs.html

[2] Luke Tierney 著, 垂水共之他訳『LISP-STAT』ISBN4-320-02805-8 定価 4841 円
Web 上にある英文の翻訳を出版したもの。他に竹村ゼミ訳も Web 上に存在する。

[3] 垂水共之『Lisp-Stat による統計解析入門』ISBN4-320-01635-1 定価 2800 円
付属プログラムおよびサポートページ http://www.f7.ems.okayama-u.ac.jp/~t2/appstat/

[4]Xlisp-Stat Contributed Code
http://www.xlispstat.org/code/