0.S3 ジュニア版 特別章(3)　格子とツリーのプログラミング方法

　ここでは、多重ループの応用として、２項や３項の格子やツリーをプログラムします。スプレッドシート式の思考方法やプログラミング法を脱して、まずは表の大枠を確保するのに０を埋めこんで、そのすべてを欠損値に変換することから始めましょう。

０から始めて表の確保する

　例えば、縦９横５（すなわち９行５列）の零行列を作成しておいて、それを欠損値のドット表示に変換することによって、これから数値を埋め込んでいく表の大枠を作ります。

プログラム

```
new;
cls;
m=miss(zeros(5,5),0);
print m;
```

画面表示

```
.    .    .    .    .
.    .    .    .    .
.    .    .    .    .
.    .    .    .    .
.    .    .    .    .
```

プログラム各行の説明

１行目　メモリを初期化する。
２行目　スクリーンをクリアする。
３行目　５行５列の零行列を作成しておき、その行列の要素が０のものを組込み関数 miss によって欠損値のドットに変換する。その結果を変数mに代入する。
４行目　変数mを画面表示する。

なお、３行目はm=zeros(5,5);として変数mに５行５列の零行列を代入しておいて、それを組込み関数missによってm=miss(m,0);とすることによって、変数mの要素のうち０であるものを欠損値のドットに変換した結果をあらためて変数mに代入するのと同じことです。GAUSSには、スプレッドシートのように枠がありません。その代わりに、例えばこのように表がイメージできるようにドットで表しておきましょう。

行列の行と列のインデックス

　プログラムの世界では通常、行はｉ列はｊでその行列のインデックスを表します。私たちもその考え方を極力変えることなく用いて、これからのプログラムを考えることにしましょう。まずは練習として、表の行の１行目に横に1,2,3,4,5と代入してみましょう。つまり

今、上と同じ５行５列の表の大枠を考えているものだとすると、１行目の１行に１列目から順に1,2,3,4,5と５列目まで埋めることになります。

プログラム

```
new;
cls;
m=miss(zeros(5,5),0);
j=1;
do while j<=5;
    m[1,j]=j;
    j=j+1;
endo;
print m;
```

画面表示

```
      1.0000000       2.0000000       3.0000000       4.0000000       5.0000000
              .               .               .               .               .
              .               .               .               .               .
              .               .               .               .               .
              .               .               .               .               .
```

プログラム各行の説明

１行目　メモリを初期化する。
２行目　スクリーンをクリアする。
３行目　５行５列の零行列を作成しておき、その行列の要素が０のものを組込み関数 miss によって欠損値のドットに変換する。その結果を変数mに代入する。
４行目　列のインデックス番号ｊを１に設定しておく。
５行目　ｊが５（すなわち変数 m の列数）以下の間はこの Do ループを行なう。
６行目　変数mの１行目の各ｊ列目にｊを代入していく。
７行目　ｊを１増やしてあらためてｊとする。
８行目　この Do ループの終わりの部分。ここまでを繰り返す。
９行目　変数mを画面表示する。

すなわち、ｊが１の１列目からmの列数まで、変数mの１行目に１列目には１を２列目には２を、これを５列目まで繰り返して、この Do ループを脱して、最終的に変数mを画面表示させるプログラムです。行を表すインデックスはこの場合１で固定されていて、列を表すｊを１からmの列数である５まで１つずつ動かしています。ちょうど

	１列目	２列目	３列目	４列目	５列目
１行目	m[1,1]=1,	m[1,2]=2,	m[1,3]=3,	m[1,4]=4,	m[1,5]=5

となる１から５までの部分をｊと置いたものです。ループで j=1 から５までインデックス番号を動かすことによって、数字を代入すべき場所を動かしているのと同時に、代入すべき数値も（この場合偶然に列番号と一致しているが）j=1 から５まで動かしているのです。

今度は、いままで１行目に固定していた行もインデックス番号を用いて動かしてみる例を考えます。１列目は１行目まで、２列目は２行目まで、これを繰り返し５列目は５行目まで列番号と同じ数を表に代入することをやってみます。

プログラム

```
new;
cls;
m=miss(zeros(5,5),0);
j=1;
do while j<=5;
    i=1;
    do while i<=j;
        m[i,j]=j;
        i=i+1;
    endo;
    j=j+1;
endo;
print m;
```

画面表示

```
     1.0000000     2.0000000     3.0000000     4.0000000     5.0000000
             .     2.0000000     3.0000000     4.0000000     5.0000000
             .             .     3.0000000     4.0000000     5.0000000
             .             .             .     4.0000000     5.0000000
             .             .             .             .     5.0000000
```

プログラム各行の説明

１行目　メモリを初期化する。

２行目　スクリーンをクリアする。

３行目　５行５列の零行列を作成しておき、その行列の要素が０のものを組込み関数 miss

によって欠損値のドットに変換する。その結果を変数mに代入する。
４行目　列のインデックス番号ｊを１に設定しておく。
５行目　ｊが５（すなわち変数 m の列数）以下の間はこの外側の Do ループを行なう。
６行目　行のインデックス番号ｉを１に設定しておく。
７行目　ｉがｊ以下の間はこの内側の Do ループを行なう。
８行目　変数mのｉ行ｊ列目に列番号と同じｊを代入していく。
９行目　ｉを１増やしてあらためてｉとする。
10 行目　この内側のｉに関する Do ループの終わりを表す部分。
11 行目　ｊを１増やしてあらためてｊとする。
12 行目　この外側のｊに関する Do ループの終わりを表す部分。
13 行目　変数mを画面表示する。

少し複雑ですが、内側のｉに関する Do ループが挿入されていて、行を表すインデックス番号のｉもそれぞれの列ｊに対して１ずつまされていると考えてください。ただし、このｉは最大ｊまでしか行かないという意味で do while i<=j;というふうに最大行数５ではなくてｊがきています。第ｊ列では、最大ｊ行目までしか数字は入らないということです。

２項格子（簡単なケース）

ここで、上のｉとｊを動かす２重ループ（各ｊ列に対してｉを動かすループ）と同じものを用いて、今度はｊを代入するのではなくて２項格子にしたがうダイナミックな数を代入してみましょう。今、初期値 S＝100 に対して、水平に左に移動すればｕ=1.1 をかけたもの、そうではなくて下降して左に移動する場合ｄ=1/u をかけたものがくるような、簡単な２項格子モデルを考えよう。すなわち、求めたいのは

S	Su	Su^2	Su^3	Su^4
	Sd	Sud	Su^2d	Su^3d
		Sd^2	Sud^2	Su^2d^2
			Sd^3	Sud^3
				Sd^4

である。これをプログラム化するため、u と d の両方が各項にあらわれるように書き直すと

Su^0d^0	Su^1d^0	Su^2d^0	Su^3d^0	Su^4d^0
	Su^0d^1	Su^1d^1	Su^2d^1	Su^3d^1
		Su^0d^2	Su^1d^2	Su^2d^2
			Su^0d^3	Su^1d^3
				Su^0d^4

となる。ここでは何も u またはｄの項がない部分は $u^0=1$ と $d^0=1$ であることと、u とｄが単独でくるところは１乗と考えてすべてを書きなおしている。乗数の和が１列目は０、２

列目は１、３列目は２、４列目は３、５列目は４にすべての項がなっていることに着目しよう。パターンがわかれば、あとのプログラム化は簡単である。すなわち、第ｊ列目は常にｕまたはｄにかかる乗数がｊ－１の和になっているのである。なお、ｕの乗数はｊ－１から１つずつその列を１つ下にいくごとに減って最終的に０になっている。他方、ｄの乗数は１つずつその列を１つ下にいくごとに増えて最終的にｊ－１になっている。

プログラム

```
new;
cls;
S=100; u=1.1; d=1/u;
m=miss(zeros(5,5),0);
j=1;
do while j<=5;
    i=1;
    do while i<=j;
        m[i,j]=S*u^(j-i)*d^(i-1);
        i=i+1;
    endo;
    j=j+1;
endo;
print m;
```

画面表示

100.00000	110.00000	121.00000	133.10000	146.41000
.	90.909091	100.00000	110.00000	121.00000
.	.	82.644628	90.909091	100.00000
.	.	.	75.131480	82.644628
.	.	.	.	68.301346

プログラム各行の説明

１行目　メモリを初期化する。

２行目　スクリーンをクリアする。

３行目　初期値Ｓは 100、ｕは 1.1 そして d は u の逆数 1/u に設定しておく。

４行目　５行５列の零行列を作成しておき、その行列の要素が０のものを組込み関数 miss によって欠損値のドットに変換する。その結果を変数ｍに代入する。

５行目　列のインデックス番号ｊを１に設定しておく。

６行目　ｊが５（すなわち変数 m の列数）以下の間はこの外側の Do ループを行なう。

７行目　行のインデックス番号ｉを１に設定しておく。

８行目　ｉがｊ以下の間はこの内側の Do ループを行なう。

９行目　変数mのｉ行ｊ列目に$Su^{j-i}d^{i-1}$を代入していく。
10 行目　ｉを１増やしてあらためてｉとする。
11 行目　この内側のｉに関する Do ループの終わりを表す部分。
12 行目　ｊを１増やしてあらためてｊとする。
13 行目　この外側のｊに関する Do ループの終わりを表す部分。
14 行目　変数mを画面表示する。

これまで、変数mのｉ行ｊ列目にはｊを代入していたが、ここでは２項格子の意味のある数値を計算して、パターン化して代入している。つまり、第ｊ列について一番上の u の乗数は１を引いたｊ－１である。その下はｊ－２、これを繰り返す。これをｉ行目の値を見る観点から考えると、ｊ－ｉをｉ＝１から順に１つずつｉを増していることにあたる。他方、ｄの乗数は一番上から順に０から１ずつ増えていく。一番上は１行目であるから、今内側のループでｉを動かしているので、ｉ－１から順に行数が１増えるごとにこの数も増えていく。実際、u の乗数ｊ－ｉとｄの乗数ｉ－１とを合計した数はｊ－１になっている。これはパターンを見たときの各列ｊの乗数の和に等しくなっている。これで、第ｊ列に対して、ｉを１からｊまで動かすときの計算パターンが決まったわけである。これを前のｊを代入するところにもってきて同じ計算をさせれば、求める２項格子の計算の完成である。これを procedure 化して、S=100 のところを０期、その隣の２列目を１期、などとして、０期から n=４期までの 2 項格子を求める procedure を作成してみよう。

プログラム

```
new;
cls;
S=100; u=1.1; d=1/u;
n=4;
call lattice2(S,u,d,n);

proc lattice2(S,u,d,n);
    local m,i,j;
    m=miss(zeros(n+1,n+1),0);
    j=1;
    do while j<=cols(m);
        i=1;
        do while i<=j;
            m[i,j]=S*u^(j-i)*d^(i-1);
            i=i+1;
        endo;
```

```
        j=j+1;
    endo;
    print/rz seqa(0,1,n+1)';
    print m;
    retp(m);
endp;
```

画面表示

0	1	2	3	4
100.00000	110.00000	121.00000	133.10000	146.41000
.	90.909091	100.00000	110.00000	121.00000
.	.	82.644628	90.909091	100.00000
.	.	.	75.131480	82.644628
.	.	.	.	68.301346

２項格子（通常のケース）

これまでは２項格子の簡単なケースであったが、左に平行に行くところを上昇 u、下降するところを下降ｄとして同じことを計算してみよう。結果は同じになるが、今度はかなりインデックスの扱いが複雑になる。ここでは、これまでと同じように第ｊ列目に対して、内側の Do ループのインデックスｉを１から１つずつ増やす方法をとることにしよう。

プログラム

```
new;
cls;
S=100; u=1.1; d=1/u;
n=4;
call lat2(S,u,d,n);

proc lat2(S,u,d,n);
    local m,i,j;
    m=miss(zeros(2*n+1,n+1),0);
    j=1;
    do while j<=cols(m);
        i=1;
        do while i<=2*j-1;
            m[n+1-j+i,j]=S*u^(j-1-(i-1)/2)*d^((i-1)/2);
            i=i+2;
```

```
        endo;
        j=j+1;
    endo;
    print/rz seqa(0,1,n+1)';
    print m;
    retp(m);
endp;
```

画面表示

0	1	2	3	4
.	.	.	.	146.41000
.	.	.	133.10000	.
.	.	121.00000	.	121.00000
.	110.00000	.	110.00000	.
100.00000	.	100.00000	.	100.00000
.	90.909091	.	90.909091	.
.	.	82.644628	.	82.644628
.	.	.	75.131480	.
.	.	.	.	68.301346

３項格子

今度は、上昇は u 倍、水平は１倍、そして下降はｄ倍となる３項格子を考えよう。ただし、ｄは u の逆数に必ず一致するものとする。そうすると、計算結果は

				Su^4
			Su^3	Su^3
		Su^2	Su^2	Su^2
	Su	Su	Su	Su
S	S	S	S	S
	Sd	Sd	Sd	Sd
		Sd^2	Sd^2	Sd^2
			Sd^3	Sd^3
				Sd^4

というふうな並びになるはずである。なぜならば、d=1/u なので du=1 になるからだ。プログラムでは、列ではなくて行ｉに着目して、先に真ん中の同じＳの行を先に埋めておいてから、上半分の u からなる三角形の部分と下半分のｄからなる三角形の部分に分割して、それぞれについて、１乗の u、２乗の u、これを繰り返して各行ごとに代入する。同様にし

て、下の三角形についても１乗のｄ、２乗のｄ、これを繰り返す。

プログラム

```
new;
cls;
S=100; u=1.1; d=1/u;
n=4;
call lat3(S,u,d,n);

proc lat3(S,u,d,n);
    local m,i,j;
    m=miss(zeros(2*n+1,n+1),0);
    j=1;
    do while j<=n+1;
        m[n+1,j]=S*1;
        j=j+1;
    endo;
    i=n;
    do while i>=1;
        j=n+2-i;
        do while j<=n+1;
            m[i,j]=S*u^(n+1-i);
            j=j+1;
        endo;
        i=i-1;
    endo;
    i=n+2;
    do while i<=2*n+1;
        j=i-n;
        do while j<=n+1;
            m[i,j]=S*d^(i-n-1);
            j=j+1;
        endo;
        i=i+1;
    endo;
    print/rz seqa(0,1,n+1)';
    print m;
```

```
    retp(m);
endp;
```

画面表示

0	1	2	3	4
.	.	.	.	146.41000
.	.	.	133.10000	133.10000
.	.	121.00000	121.00000	121.00000
.	110.00000	110.00000	110.00000	110.00000
100.00000	100.00000	100.00000	100.00000	100.00000
.	90.909091	90.909091	90.909091	90.909091
.	.	82.644628	82.644628	82.644628
.	.	.	75.131480	75.131480
.	.	.	.	68.301346

２項ツリー（ＹＥＳとＮＯの判断分岐）

ツリーは、一般に、格子よりもプログラムは複雑になる。まずは、１と０からなるツリーを作成し、１ならＹＥＳを０ならＮＯを割り当てて、その判断分岐のグラフを作ってみよう。ここでは、あとの計算も考えて、先に１と０を割り当てておいて、１ならＹＥＳという文字列を０ならＮＯという文字列になるように変換した結果を表示させることにする。

プログラム

```
new;
cls;
n=2;
call treeYESNO(n);

proc treeYESNO(n);
    local label1,label0,m,onezero,i,j;
    label1={"YES"};
    label0={" NO"};
    m=miss(zeros((2^n)*2-1,n+1),0);
    onezero=1|0;
    j=1;
    m[2^(n+1-j),j]=0;
    j=2;
    do while j<=n+1;
```

```
        i=1;
        do while i<=2^(j-1);
            onezero=onezero | 1 | 0;
            m[2^(n+1-j)+2^(n+2-j)*(i-1),j]=onezero[i];
            i=i+1;
        endo;
        j=j+1;
    endo;
    j=1;
    do while j<=cols(m);
        i=1;
        do while i<=rows(m);
            if m[i,j]==0;
                m[i,j]=label0;
            elseif m[i,j]==1;
                m[i,j]=label1;
            endif;
            i=i+1;
        endo;
        j=j+1;
    endo;
    print/rz seqa(0,1,n+1)';
    print $m;
    retp(m);
endp;
```

画面表示

0	1	2
.	.	YES
.	YES	.
.	.	NO
NO	.	.
.	.	YES
.	NO	.
.	.	NO

２項ツリーモデル

今度は、上昇 YES か下降 NO の分岐から得られる Payoff を考えるモデルを考える。なお、YES と NO の出方は上のものと同じとする。YES ならｐの割合で減るものとする。NO であれば変化はないものとする。これは、かなり複雑なパターン計算を必要とする。

プログラム

```
new;
cls;
S=100; p=0.1; n=2;
call btree(S,p,n);

proc btree(S,p,n);
    local label0,label1,m,m2,m3,onezero,i,j,index;
 /* Yes(1) or No(0) */
    label1={"YES"};
    label0={" NO"};
    m=miss(zeros((2^n)*2-1,n+1),0);
    onezero=1|0;
    j=1;
    m[2^(n+1-j),j]=0;
    j=2;
    do while j<=n+1;
        i=1;
        do while i<=2^(j-1);
            onezero=onezero|1|0;
            m[2^(n+1-j)+2^(n+2-j)*(i-1),j]=onezero[i];
            i=i+1;
        endo;
        j=j+1;
    endo;
    j=1;
    do while j<=cols(m);
        i=1;
        do while i<=rows(m);
            if m[i,j]==0;
                m[i,j]=label0;
            elseif m[i,j]==1;
```

```
                m[i,j]=label1;
            endif;
            i=i+1;
        endo;
        j=j+1;
    endo;
    print/rz seqa(0,1,n+1)';
    print $m;
/* Amount left */
    m2=miss(zeros(2^n,n+1),0);
    m2[.,1]=zeros(2^n,1);
    j=n+1;
    do while j>=2;
        index=ones(2^(n+1-j),1)|zeros(2^(n+1-j),1);
        i=1;
        do while i<=(2^n)/(2^(n+1-j)*2)-1;
            index=index|ones(2^(n+1-j),1)|zeros(2^(n+1-j),1);
            i=i+1;
        endo;
        m2[.,j]=index;
        j=j-1;
    endo;
    m2=S*(1-p)^cumsumc(m2')';
    m3=miss(zeros((2^n)*2-1,n+1),0);
    j=1;
    do while j<=n+1;
        i=1;
        do while i<=2^(j-1);
            m3[2^(n+1-j)+2^(n+2-j)*(i-1),j]=m2[1+2^(n+1-j)*(i-1),j];
            i=i+1;
        endo;
        j=j+1;
    endo;
    print;
    print "Amount left:";
    print/rz seqa(0,1,n+1)';
```

```
    print/rz m3;
    retp(m3);
endp;
```

画面表示

```
                0             1             2

                .             .           YES
                .           YES             .
                .             .            NO
               NO             .             .
                .             .           YES
                .            NO             .
                .             .            NO

Amount left:
                0             1             2

                .             .            81
                .            90             .
                .             .            90
              100             .             .
                .             .            90
                .           100             .
                .             .           100
```

ここで procedure の前半部分はその前の YES と NO のものと同じプログラムである。後半部分は、m2=S*(1-p)^cumsumc(m2')';の行の前に仮に print m2;として m2 の内容を表示してやると何をやっているのかがわかるだろう。すなわち、

```
      0.00000000     1.0000000     1.0000000
      0.00000000     1.0000000    0.00000000
      0.00000000    0.00000000     1.0000000
      0.00000000    0.00000000    0.00000000
```

というような１と０のからなるインデックス行列を作成していることになる。１は YES であり０は NO である。各行がその YES と NO の履歴になる。１行目は 0,1,1 だから NO から始めて YES,YES ということであり、２行目は NO から始めて YES,NO ということである。このように各行を横に見てやるとその YES と NO の出具合がわかる。なお、これを縦

方向、各列で見てやると、１列目はすべて０、２列目は上半分が１下半分が０となっている。次の列は、前列の上半分のそのまた半分が１で、あと残りの半分が０である。さらに前列の下半分のそんまた半分が１で、その残り半分が０である。これを繰り返す。最終列では YES と NO（すなわち１と０）が交互にくることになる。これをプログラムしたものが、procedure 後半の今 print m2;を仮に挿入してみた部分よりも前のところである。

さらに、このインデックス行列の「行ごとの」累積和を計算すると

0.00000000	1.0000000	2.0000000
0.00000000	1.0000000	1.0000000
0.00000000	0.00000000	1.0000000
0.00000000	0.00000000	0.00000000

となる。これは、GAUSS では cumsumc(m2')'によって計算されるものである。つまり、インデックス行列を転置して列ごとの累積和を計算してやり、もとのインデックス行列と同じ形に直すために、その結果自体をさらに転置させているのである。行の横方向に見てやって、０は１が一回もきていないこと、１は１が１回、２は１が２回きていることを表している。最終的に、S*(1-p)^ cumsumc(m2')'としてやって、１からｐだけの割合を除いた残りの部分をこの累積和で累乗したものに初期値 S をかけると、残りの量の行列ができる。これを表示してもよいのだが、最終行を除き同じものが重なって含まれていて、なおかつツリーの形をなしていない。そこで、それを procedure の最後の部分で行列 m3 の中にピックアップし、(2^n)*2-1 行 n+1 列に重なりがないように代入していくのである。

```
j=1;
do while j<=n+1;
    i=1;
    do while i<=2^(j-1);
        m3[2^(n+1-j)+2^(n+2-j)*(i-1),j]=m2[1+2^(n+1-j)*(i-1),j];
        i=i+1;
    endo;
    j=j+1;
endo;
```

上のように、外側のｊループで各ｊ列について見ていく。内側のｉループは上で計算した重なりを含む結果が１行目から始まるため、ｉ＝１から始まっている。第ｊ列のノードの数は、最初が１つ、次が２つ、その次が４つと続くから、2^{j-1} まで数えればよい。重なりを含んだ計算結果 m2 からピックアップして、ツリーの形をなした変数行列 m3 に代入し直すのだが、問題は行のところになる。なお、m2 の最初はいつも１になる。実際、*(i-1)のところが１から回るので、常にその項は０から始まることになり、１＋のところが残り、これでよい。例えば、２列目は１行目と３行目の計２つをピックアップする必要があり、３列目は、この場合(ｎ=2）ｊが３に対して 2^(n+1-j)の累乗のところは０になるので、この部

分は２の０乗で１となる。全体としては 1,2,3,4 と順番に m2 の行をピックアップしていくことになっている。この部分はすべての自然数ｎについて成り立つように書いているのでこのように複雑になっている。これをｊ列ごとに m3[2^(n+1-j)+2^(n+2-j)*(i-1),j]に代入していく。ｎ＝２に対して、１列目は４行目だけ、２列目は２行目と６行目、３列目は１,３,５,７行目だから上の式は一致している。複雑な累乗数になっているが、出発点＋増分の計算をしているだけである。このように最終的にツリーの形に変形することで計算は完成する。途中複雑な計算をしているように見えるが、足し算掛け算そして累乗の数を使って、ｊ列目に対してｎが決まっているときの配列の場所を式で表しているわけである。ツリーのノードとノードの間隔があいているものを作りたい場合や、上から展開していくツリーを作りたい場合でも、どうようにしてパターンがあるかぎり足し算掛け算そして累乗の数でインデックスは作成できるものだと考えて、式を考えてあてはめて確認する作業を行なっているのです。

　このように、格子やツリーは２次元のｉとｊの配列のインデックスを用いてプログラムしていきます。上のやり方は、ほんの１つのやり方にすぎません。パターンがあるかぎり、特にそのパターンが上下対称の場合、必ずｉとｊの配列表現が可能です。２項格子が下から上へ展開していようが、２項ツリーが上から下へと展開していようが、同様にパターンを足し算や掛け算それに累乗の計算でｉとｊの配列で表現してプログラムできます。

発展

⇒ Luenberger　２項格子　5.3b　11.1a
　　３項格子　13.7d　16.3a
　　２重３項格子　16.5a
　　２項ツリー　5.3c　16.6a