2007 年 12 月 26 日

LINGO10 の衝撃

LINDO JAPAN 代表:新村秀一(成蹊大学教授)

1. LINGO の機能

1.1 LINDO 社の多彩な製品群

LINGO は, Excel のアドイン・ソルバーである What'sBest!と同じく, LINDO Systems Inc.が開発販売する, 線形計画法(LP), 整数計画法(IP), 2 次計画法(QP), 非線形計画法(NLP)のための, 使い勝手が良く強力な数理計画法(MP)の総合的なソルバーである.

LP における Barrier オプション(いわゆる内点法)や, NLP で大域的最適解を求めるオプションも準備している.

どの解法を選ぶかは, ソルバーがモデルを分析し決定するので, ユーザーの負担はない. また, 開発には, これらのソルバーに用いられたCのライブラリー集である LINDO API がある.

LINDO は, LINDO Systems Inc.が最初に開発した LP,IP,QP のソルバーであり, 現在は LINGO に吸収され発売を停止しているので, LINDO API にその記念碑的な名前をとどめている.

1.2 What'sBest!

What'sBest!は, Excel のアドイン・ソフトとしての MP ソルバーである. ABC という3段階の開発手順を提案している. A は Adjustable すなわち最適化によって数値が変わる決定変数をまず決めることを意味している. B は Best Cell すなわち決定変数で目的関数を定義することを意味している. 最後の C は, Constraints すなわち決定変数で, 制約式を記述することを意味する.

しかし, ユーザーが分析したい対象をモデル化するのは難しい. そこで, 種々のタイプのモデルをサンプルモデルとして提供している. これらの雛形モデルを理解し, ちょっと工夫して修正すれば, ユーザー自身が MP モデルを作成し, 分析できることを想定している.

これまでの数理計画法のソフトは, 使い勝手が悪く, 一部専門家にしか受け入れられなかった. What'sBest!や LINGO は, LINDO 以来一貫して追求してきた LINDO Systems Inc.の「**使いやすさ**」と「**大規模モデル作成の容易さ**」という**2 つの相反する機能**を実現した究極のソルバーである.

このため, LINDO は自然表記で分かりやすいが, 大規模な実用システムの開発にモデル作成の工数を必要とする欠点があり, その使命を全うした.

1.3 LINGO

LINGO は, 数理計画法の全ての分野をカバーする専用のソフトである. モデルの記述方法は, 次の 3 通りである.

１） LINDO と同じ自然表記なモデル記述（スカラーモデル）. これは, 初心者にとって理解しやすい. また, LINDO 形式で記述したモデルも分析できる.

２） 集合によるモデル記述. 世の中の事象を, 集合とその属性として捕らえます. 例えば, 配合問題を例にとれば, 11 個の原材料（RAW）という集合に対し, その購入価格（PRICE）, 購入量（BUY）などが属性になる. また, 最終製品の 8 個の品質（PRODUCT）という集合を考えると, 品質の上限値（UPPER）と下限値（LOWER）が属性になる. これらは, 次のように定義される. すなわち, 4 つの 1 次元配列が, 2 つの集合にまとめられている.

```
SET:
RAW:PRICE,BUY;
PRODUCT:UPPER,LOWER;
ENDSET
```

そして, 各原材料ごとの品質成分（SEIBUN）は, 次のような定義で, いわゆる 11 行*8 行の 2 次元配列になる.

```
V（RAW,PRODUCT): SEIBUN；
```

これは, 原始集合の RAW と PRODUCT から作られた派生集合という. 原始集合を 4 つ書き連ねれば, 4 次元配列になる.

そして, これらの属性に対し, DATA 節でデータが定義される.

```
DATA:
PRICE=1, 2, . . . , 11；
UPPER=71, . . . . , 81；
LOWER=50, . . . , 60；
SEIBUN=１１＊８個の値のリスト；
ENDDATA
```

これらに対して, 便利な集合演算が準備されている. これによって数理計画法モデルは, モデルがデータから独立して記述できる.

３） データのオブジェクト･リンク

データを DATA 節で直接記述するのでなく, オブジェクトリンクで, Excel ファイルやテキスト形式のファイルから入力できる.

```
DATA:
PRICE=@OLE()；
UPPER=@OLE()；
LOWER=@OLE()；
```

```
SEIBUN=@OLE();
ENDDATA
```

これによって，データサイズの違いから，モデルが解放される．すなわち，次のLINGO モデルがあれば，単純などんな配合モデルも Excel 上にデータを準備するだけで解くことができる．すなわち，モデルのサイズに影響されなくなった．このような汎用化を通常のプログラムで作成するには，かなり凝ったモデル生成の汎用プログラムを作成する必要がある．

LINDO の欠点は，自然表記で教育上わかりやすいが，それを企業にはいって実際の問題の解決に当たる際,モデル生成のプログラム開発が必要になる．そして，開発工数と費用が発生する．

```
SETS:
 CONTENT: UL,LL;
 RAW: PRICE,PRODUCT;
 MATRIX(RAW,CONTENT):SEIBUN;
ENDSETS
DATA:
 UL=@OLE();
 LL=@OLE();
 PRICE=@OLE();
 SEIBUN=@OLE();
ENDDATA
MIN=@SUM( RAW(i):PRICE(i)*PRODUCT(i));
@FOR( CONTENT(j): @SUM(RAW(i): SEIBUN(i,j)*PRODUCT(i))<=UL(j));
@FOR( CONTENT(j): @SUM(RAW(i): SEIBUN(i,j)*PRODUCT(i))>=LL(j));
@SUM(RAW(i): PRODUCT(i))=1;
```

４）　プログラミング機能

LINGO はさらに，CALC 節で，2 章のサンプルのようにプログラミングできる．すなわち，最適化問題の結果を見て次の最適化問題を逐次的に解くことができる．これによって，個人が SE やプログラマーの助けを借りずに，これまで大規模で複雑な数理計画法のシステムと考えられてきたものが簡単に開発できる．

私自身，MP による判別モデルを WB で開発してきたが，LINGO でより洗練したモデルに整理しようと考えている．

2. 雛形モデルの例

以下は，LINGO10 の SAMPLE ホルダーにある CALC 節を使ったプログラミングの例である．

2.1 バイナリサーチ (LOOPBINS.lg4)

以下は, 集合 X で定義された数列から指定した数値 16 を, CALC 節でバイナリー・サーチで探すプログラムであり, MP モデルではない. すなわち, LINGO は, 一般の連立不等式の解などを求めるのにも利用できる.

```
MODEL:
! Illustrates programming looping
  capabilities of LINGO by doing a
  binary search;
SETS:
  S1: X;
ENDSETS

DATA:
 ! The key for which we will search;
  KEY = 16;
 ! The list (must be in sorted
   increasing order);
  X = 2 7 8 11 16 20 22 32;
ENDDATA

! Do a binary search for key;
CALC:
 ! Set begin and end points of search;
  IB = 1;
  IE = @SIZE( S1);
 ! Loop to find key;
  @WHILE( IB #LE# IE:
   ! Cut bracket in half;
    LOC = @FLOOR((IB + IE)/2);
    @IFC( KEY #EQ# X(LOC):
      @BREAK; ! Do no more loops;
    @ELSE
      @IFC( KEY #LT# X( LOC):
        IE = LOC-1;
      @ELSE
```

```
      IB = LOC+1;
    );
  );
);

@IFC( IB #LE# IE:
 ! Display key's location;
  @PAUSE( 'Key is at position: ', LOC);
@ELSE
 ! Key not in list;
  @STOP( ' Key not on list!!!');
);
ENDCALC
END
```

以下は，出力である．

```
Feasible solution found.
Total solver iterations:                         0

                                  Variable           Value
                                       KEY        16.00000
                                        IB        5.000000
                                        IE        5.000000
                                       LOC        5.000000
                                     X( 1)        2.000000
                                     X( 2)        7.000000
                                     X( 3)        8.000000
                                     X( 4)        11.00000
                                     X( 5)        16.00000
                                     X( 6)        20.00000
                                     X( 7)        22.00000
                                     X( 8)        32.00000
```

2.2 人員配置（LOOPSTAFF.lg4）

1週間の要求を満たす人員配置を行う．第1段階で雇用費用最小化モデルを解く．第2段階で過剰人員の最小化モデル，第3段階で日曜日の過剰人員数を最小化するモデルを解いている．第1段階の最少の雇用費用を維持したまま，日曜の超過人員を最小化している点がポイントである．一つのモデルをサブモデルに分割定義し，それらの組み合わせで異なった3つのモデルを解いて

いる.

```
MODEL:

 ! Staffing example:
    Step 1:
       - Minimize total cost
    Step 2:
       - Fix costs to their level from Step 1
       - Solve to minimize max staff overage
         on any given day, thereby spreading
         excess staff through the week
    Step 3:
       - Fix max overage to it's level from
         Step 2
       - Solve to minimize the number working
         on a Sunday
 ;

 SETS:
    DAY / MON, TUE, WED, THU, FRI, SAT, SUN/ :
       NEED, START, COST, EXCESS;
 ENDSETS

 DATA:
    NEED =  18,  15,  12,  16,  19,  14,  12;
    COST = 200, 200, 200, 200, 200, 200, 200;
 ENDDATA

 SUBMODEL OBJ_COST:
   MIN = TTL_COST;
 ENDSUBMODEL

 SUBMODEL OBJ_MAX_EXCESS:
   MIN = MAX_EXCESS;
 ENDSUBMODEL
```

```
SUBMODEL OBJ_MIN_SUNDAY_STARTS:
  MIN = EXCESS( @INDEX( DAY, SUN));
ENDSUBMODEL

SUBMODEL BASE:
  ! Compute cost;
  TTL_COST = @SUM( DAY( D) : START( D) * COST( D));

  ! The constraints;
  @FOR( DAY( D):
   @SUM( DAY( COUNT)| COUNT #LE# 5:
   START( @WRAP( D - COUNT + 1, @SIZE( DAY)))) -
    EXCESS( D)= NEED( D)
  );

  ! Computes the maximum staff excess;
  @FOR( DAY( D): MAX_EXCESS >= EXCESS( D));

  ! Starts must be integral;
  @FOR( DAY: @GIN( START));

  @BND( 0, TTL_COST, BNDU_COST);
  @BND( 0, MAX_EXCESS, BNDU_MAX_EXCESS);
ENDSUBMODEL

CALC:
  !Run in quiet mode;
  @SET( 'TERSEO', 2);

  !Free up bounds;
  BNDU_COST =  1.E10;
  BNDU_MAX_EXCESS =  1.E10;

  !Solve to minimize cost;
  @SOLVE( OBJ_COST, BASE);
```

```
@WRITE( 'Solution 1 - min cost:', @NEWLINE( 1));
@WRITE( '    Start: ');
@WRITEFOR( DAY: @FORMAT( START, '6.0f'));
@WRITE( @NEWLINE( 1));
@WRITE( '   On Duty: ');
@WRITEFOR( DAY: @FORMAT( NEED + EXCESS, '6.0f'));
@WRITE( @NEWLINE( 1));
@WRITE( '    Excess: ');
@WRITEFOR( DAY: @FORMAT( EXCESS, '6.0f'));
@WRITE( @NEWLINE( 2));

!Fix TTL_COST to optimal value;
BNDU_COST = TTL_COST;

!Re-solve to minimize the max excess staff on any given day
 in order to spread any excess staff throughout the week;
@SOLVE( OBJ_MAX_EXCESS, BASE);

@WRITE( 'Solution 2 - min cost / min max excess:', @NEWLINE( 1));
@WRITE( '    Start: ');
@WRITEFOR( DAY: @FORMAT( START, '6.0f'));
@WRITE( @NEWLINE( 1));
@WRITE( '   On Duty: ');
@WRITEFOR( DAY: @FORMAT( NEED + EXCESS, '6.0f'));
@WRITE( @NEWLINE( 1));
@WRITE( '    Excess: ');
@WRITEFOR( DAY: @FORMAT( EXCESS, '6.0f'));
@WRITE( @NEWLINE( 2));

!Fix max excess to optimal level;
BNDU_MAX_EXCESS = MAX_EXCESS;

!Re-solve to minimize total workers on Sundays;
@SOLVE( OBJ_MIN_SUNDAY_STARTS, BASE);

@WRITE( 'Solution 3 - min cost / min max excess / min Sunday:',
```

```
@NEWLINE( 1));
   @WRITE( '    Start: ');
   @WRITEFOR( DAY: @FORMAT( START, '6.0f'));
   @WRITE( @NEWLINE( 1));
   @WRITE( '   On Duty: ');
   @WRITEFOR( DAY: @FORMAT( NEED + EXCESS, '6.0f'));
   @WRITE( @NEWLINE( 1));
   @WRITE( '   Excess: ');
   @WRITEFOR( DAY: @FORMAT( EXCESS, '6.0f'));
   @WRITE( @NEWLINE( 2));

 ENDCALC

END
```

出力結果は次の通りである．第1段階で，費用最小解を求めている．第2段階で，同じ最小費用で過剰人員の最小化を行っている．第3段階で，第2段階の条件の上に，日曜の過剰人員の最小化を行っている．単価は曜日で変わらないので，過剰人員数は4名で一定である．3つのモデルの日曜の過剰人員数は，2,1,0と減少している．

```
Solution 1 - min cost:
     Start:     7    1    3    4    4    2    1
   On Duty:    18   15   14   16   19   14   14
    Excess:     0    0    2    0    0    0    2

Solution 2 - min cost / min max excess:
     Start:     7    2    1    5    4    2    1
   On Duty:    19   16   13   16   19   14   13
    Excess:     1    1    1    0    0    0    1

Solution 3 - min cost / min max excess / min Sunday:
     Start:     6    4    0    6    4    1    1
   On Duty:    18   16   12   17   20   15   12
    Excess:     0    1    0    1    1    1    0
```

2.3 Portfolio分析(LOOPPORT.lg4)

10段階のリターンレベルで，Portfolio分析を行い，エフィシエント・フロンティア曲線を描く．

```
MODEL:
! Solves the generic Markowitz portfolio
  model in a loop to generate the points
  on the efficient frontier;
SETS:
   ASSET: RATE, UB, X;
   COVMAT( ASSET, ASSET): V;
   POINTS: XRET, YVAR;
ENDSETS

DATA:
! Number of points on the
   efficient frontier graph;
  NPOINTS = 10;
  POINTS = 1..NPOINTS;
! The stocks;
  ASSET = GOOGLE, YAHOO, CISCO;
! Expected growth rate of each asset;
  RATE = 1.3   1.2  1.08;
! Upper bound on investment in each;
  UB   = .75   .75   .75;
! Covariance matrix;
  V    =   3    1   -.5
           1    2   -.4
        -.5   -.4    1;
ENDDATA

! Below are the three objectives we'll use;
SUBMODEL SUB_RET_MAX:
   [OBJ_RET_MAX] MAX = RETURN;
ENDSUBMODEL

SUBMODEL SUB_RET_MIN:
   [OBJ_RET_MIN] MIN = RETURN;
ENDSUBMODEL
```

```
SUBMODEL SUB_MIN_VAR:
   [OBJ_MIN_VAR] MIN =
    @SUM( COVMAT( I, J): V( I, J) * X( I) * X( J));
ENDSUBMODEL

!and the constraints;
SUBMODEL SUB_CONSTRAINTS:
   ! Compute return;
   RETURN = @SUM( ASSET: RATE * X);
   ! Must be fully invested;
   @SUM( ASSET: X) = 1;
   ! Upper bounds on each;
   @FOR( ASSET: @BND( 0, X, UB));
   ! Must achieve target return;
   RETURN >= RET_LIM;
ENDSUBMODEL

CALC:
! Set some parameters;
   ! Reset all params;
   @SET( 'DEFAULT');
   ! Output error messages only;
   @SET( 'TERSEO', 2);
   ! Suppress status window;
   @SET( 'STAWIN', 0);

! Solve to get maximum return;
   RET_LIM = 0;
   @SOLVE( SUB_RET_MAX, SUB_CONSTRAINTS);

! Save maximum return;
   RET_MAX = OBJ_RET_MAX;

! Solve to get minimum return;
   @SOLVE( SUB_RET_MIN, SUB_CONSTRAINTS);
```

```
! Save minimum return;
   RET_MIN = OBJ_RET_MIN;

! Interval between return points;
   INTERVAL =
    ( RET_MAX - RET_MIN) / ( NPOINTS-1);

! Loop over range of possible returns,
  minimizing variance;
   RET_LIM = RET_MIN;
   @FOR( POINTS( I):
      @SOLVE( SUB_MIN_VAR, SUB_CONSTRAINTS);
      XRET( I) = RET_LIM;
      YVAR( I) = OBJ_MIN_VAR;
      RET_LIM = RET_LIM + INTERVAL;
   );

! Display the results;
   @WRITE( '    Return    Variance', @NEWLINE( 1));
   @FOR( POINTS: @WRITE( @FORMAT( XRET, '#12.6G'),
    @FORMAT( YVAR, '#12.6G'), @NEWLINE( 1))
   );
ENDCALC

CALC:

! The remainder of the model graphs the efficient frontier;

   NHASHY = 20;
   NHASHX = 60;
   SCALE = 10;

   V0 = @FLOOR( YVAR( 1) * SCALE);
   V0 = V0 / SCALE;
   V1 = @FLOOR( YVAR( NPOINTS) * SCALE + .5);
   V1 = V1 / SCALE;
```

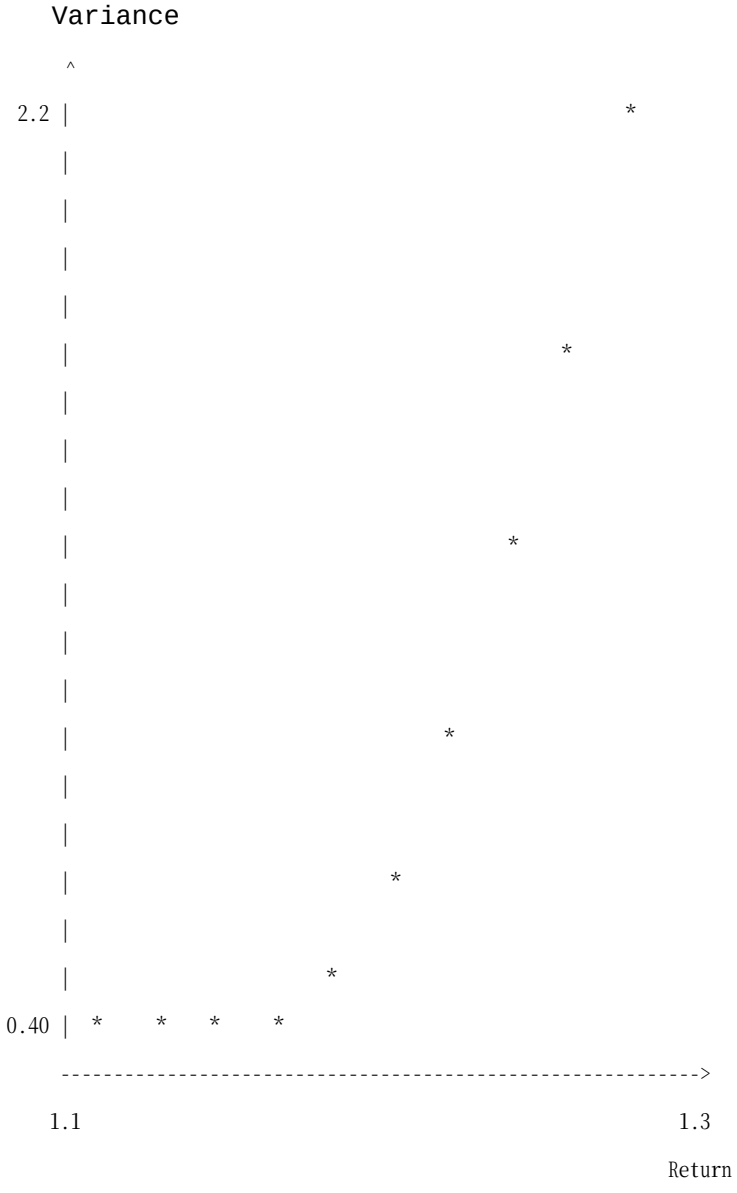
```
   R0 = @FLOOR( RET_MIN * SCALE);
   R0 = R0 / SCALE;
   R1 = @FLOOR( RET_MAX * SCALE + .5);
   R1 = R1 / SCALE;

ENDCALC

SETS:
   HASHX /1..NHASHX/: XAXIS;
   HASHY /1..NHASHY/: YAXIS;
   GRID( HASHY, HASHX): CHECK;
ENDSETS

CALC:
   XWIDTH = ( R1 - R0) / NHASHX;
   XAXIS( 1) = R0 + XWIDTH;
   XAXIS( NHASHX) = R1;

   YHEIGHT = ( V1 - V0) / NHASHY;
   YAXIS( 1) = V0 + YHEIGHT;
   YAXIS( NHASHY) = V1;

   @FOR( HASHY( J) | J #GT# 1 #AND# J #LT# NHASHY:
      YAXIS( J) = YAXIS( J - 1) + YHEIGHT;
   );
   @FOR( HASHX( J) | J #GT# 1 #AND# J #LT# NHASHX:
      XAXIS( J) = XAXIS( J - 1) + XWIDTH;
   );

   @FOR( GRID: CHECK = 0);

   @FOR( POINTS( P):
      J = 1;
      @WHILE( XRET( P) #GT# XAXIS( J): J = J + 1);
      I = 1;
```

```
      @WHILE( YVAR( P) #GT# YAXIS( I): I = I + 1);
      CHECK( I, J) = 1;
   );

   INDENT = 12;
   @WRITE( @NEWLINE( 2), (INDENT-3)*' ', 'Variance', @NEWLINE( 1));
   @WRITE( INDENT*' ', '^', @NEWLINE( 1));

   @FOR( HASHY( II):
      @IFC( II #EQ# 1:
         @WRITE( ( INDENT - 5) * ' ', @FORMAT( V1, '#4.2G'), ' |');
      @ELSE
         @IFC( II #EQ# @SIZE( HASHY):
            @WRITE( ( INDENT - 5) * ' ', @FORMAT( V0, '#4.2G'), ' |');
         @ELSE
            @WRITE( INDENT*' ', '|');
         );
      );
      @FOR( HASHX( JJ):
         @IFC( CHECK( @SIZE( HASHY) - II + 1, JJ):
            @WRITE( '*');
         @ELSE
            @WRITE( ' ');
         );
      );
      @WRITE( @NEWLINE( 1));
   );

   @WRITE( INDENT*' ', NHASHX*'-', '>', @NEWLINE( 1));
   @WRITE( (INDENT-2)*' ', @FORMAT( R0, '4.2G'), ( NHASHX - 4)*' ', R1,
@NEWLINE( 1));
   @WRITE( ( INDENT + NHASHX - 3)*' ',  'Return', @NEWLINE( 3));
ENDCALC
END
```

出力結果は，次の通りである．

```
     Return    Variance
```

1.11000	0.417375
1.12833	0.417375
1.14667	0.418054
1.16500	0.462381
1.18333	0.575957
1.20167	0.758782
1.22000	1.01086
1.23833	1.33218
1.25667	1.72275
1.27500	2.18750



2.4　鉄板や紙ロールの切断(LOOPCUT.lg4)

注文に応じて，鉄板や紙ロールの最適な切断パターンと切断長の決定を行う．ここでは，45フィート幅の鉄板から，34，24，15，10，18フィート幅で，350，100，800，1001，377フィート長の板を，無駄なロスを最小にして自動的に切り分けるパターンを生成する．製造単位期間ごとに注文のデータを作成すれば，無駄が削減できる．製鉄，製紙，ガラス，テキスタイルに適用できる．

```
MODEL:
! Uses Lingo's programming capability to do
  on-the-fly column generation for a
  cutting-stock problem;
SETS:
  PATTERN: COST, X;
  FG: WIDTH, DEM, PRICE, Y, YIELD;
  FXP( FG, PATTERN): NBR;
ENDSETS

DATA:
  PATTERN = 1..20; ! Allow up to 20 patterns;
  RMWIDTH = 45;    ! Raw material width;
  FG =  F34  F24  F15  F10 F18;!Finished goods...;
  WIDTH= 34   24   15   10  18;!their widths...;
  DEM = 350  100  800 1001 377;!and demands;
  BIGM = 999;
ENDDATA

SUBMODEL MASTER_PROB:
  [MSTROBJ] MIN= @SUM( PATTERN( J)| J #LE# NPATS:
   COST( J)*X( J));
  @FOR( FG( I):
   [R_DEM]
    @SUM( PATTERN( J)| J #LE# NPATS:
     NBR( I, J) * X( J)) >= DEM( I);
  );
ENDSUBMODEL

SUBMODEL INTEGER_REQ:
```

```
  @FOR( PATTERN: @GIN( X));
ENDSUBMODEL

SUBMODEL PATTERN_GEN:
  [SUBOBJ] MAX = @SUM( FG( I): PRICE( I)* Y( I));
  @SUM( FG( I): WIDTH( I)*Y( I)) <= RMWIDTH;
  @FOR( FG( I): @GIN(Y( I)));
ENDSUBMODEL

CALC:

  ! Set parameters;
  @SET( 'DEFAULT');
  @SET( 'TERSEO', 2); ! Turn off default output;

  ! Max number of patterns we'll allow;
  MXPATS = @SIZE( PATTERN);
  ! Make first pattern an expensive super pattern;
  COST( 1) = BIGM;
  @FOR( FG( I): NBR( I, 1) = 1);

  ! Loop as long as the reduced cost is
    attractive and there is space;
  NPATS = 1;
  RC = -1;   ! Clearly attractive initially;
  @WHILE( RC #LT# 0 #AND# NPATS #LT# MXPATS:
    ! Solve for best patterns to run among ones
       generated so far;
    @SOLVE( MASTER_PROB);
    ! Copy dual prices to PATTERN_GEN submodel;
    @FOR( FG( I): PRICE( I) = -@DUAL( R_DEM( I)));
    ! Generate the current most attractive pattern;
    @SOLVE( PATTERN_GEN);
    ! Marginal value of current best pattern;
    RC = 1 - SUBOBJ;
    ! Add the pattern to the Master if it is attractive;
```

```
    @IFC( RC #LT# 0:
       NPATS = NPATS + 1;
       @FOR( FG( I): NBR( I, NPATS) = Y( I));
       COST( NPATS) = 1;
        );
  );

  ! Finally solve Master as an IP;
  @SOLVE( MASTER_PROB, INTEGER_REQ);

ENDCALC

! This following calc section displays the
  solution in a tabular format;
CALC:
  ! Compute yield of each FG;
  @FOR( FG( F): YIELD( F) =
   @SUM( PATTERN( J)| J #LE# NPATS:
    NBR( F, J) * X(J))
  );
  ! Compute some stats;
  TOTAL_FT_USED = @SUM( PATTERN: X) * RMWIDTH;
  TOTAL_FT_YIELD = @SUM( FG: YIELD * WIDTH);
  PERC_WASTE =  100 * ( 1 - ( TOTAL_FT_YIELD / TOTAL_FT_USED)) ;
 ! Display the table of patterns and their usage;
  FW = 6;
  @WRITE( @NEWLINE( 1));
  @WRITE( ' Total raws used:    ', @SUM(PATTERN: X) , @NEWLINE( 2),
         ' Total feet yield:   ', TOTAL_FT_YIELD , @NEWLINE( 1),
         ' Total feet used:    ', TOTAL_FT_USED , @NEWLINE( 2),
         ' Percent waste:    ', @FORMAT( PERC_WASTE, '#5.2G'), '%',
@NEWLINE( 1));
  @WRITE( @NEWLINE( 1), 24*' ', 'Pattern:', @NEWLINE( 1));
  @WRITE( '   FG  Demand Yield');
  @FOR( PATTERN( I) | I #LE# NPATS: @WRITE( @FORMAT( I, '6.6G')));
  @WRITE( @NEWLINE( 1));
```

```
 @WRITE( ' ',FW*( NPATS+3)*'=', @NEWLINE( 1));
 @FOR( FG( F):
   @WRITE((FW - @STRLEN( FG( F)))*' ', FG( F), ' ',
     @FORMAT( DEM( F), '6.6G'), @FORMAT( YIELD( F), '6.6G'));
   @FOR( FXP( F, P) | P #LE# NPATS:
     @WRITE( @IF( NBR( F, P) #GT# 0,
     @FORMAT( NBR( F, P), "6.6G"), '    .')));
   @WRITE( @NEWLINE( 1))
 );
 @WRITE( ' ',FW*( NPATS+3)*'=', @NEWLINE( 1));
 @WRITE( 2*FW*' ', ' Usage:');
 @WRITEFOR( PATTERN( P) | P#LE# NPATS: @FORMAT( X( P), '6.6G'));
 @WRITE( @NEWLINE( 1));
ENDCALC

END
```

出力結果の見方は，3列目で15フィート幅のものを3枚，133フィート長で切り取る．幅のロスは0である．F15の行を見ると，3*133+377+25=801になり，１フィートだけ無駄が生じる．

```
 Total raws used:    985

 Total feet yield:   43121
 Total feet used:    44325

 Percent waste:     2.7%

                    Pattern:
   FG  Demand Yield    1    2    3    4    5    6    7    8
 ========================================================================
   F34    350   350    1    .    .    .    1    .    .    .
   F24    100   100    1    .    .    .    .    1    .    .
   F15    800   801    1    .    3    .    .    .    1    1
   F10   1001  1002    1    4    .    .    1    2    1    3
   F18    377   377    1    .    .    2    .    .    1    .
 ========================================================================
            Usage:    0    0   133    0   350   100   377    25
```

2.5　DEA法(LOOPDEA.lg4)

次は，評価技法として有名なDEA法のプログラムである．．

```
! Data Envelopment Analysis of Decision Maker Efficiency ;
! Keywords: DEA, Data Envelopment Analysis;
SETS:
  DMU: !The decisionmaking units;
    SCORE; ! Each decision making unit has a
              score to be computed;
  FACTOR: TW;
! There is a set of factors, input & output;
  DXF( DMU, FACTOR):  F, ! F( I, J) = Jth factor of DMU I;
        W;  ! Weights used to compute DMU I's score;
 ENDSETS
 DATA:
  DMU = BL HW NT OP YK EL;
! Inputs are spending/pupil, % not low income;
! Outputs are Writing score and Science score;
  NINPUTS = 2;  ! The first NINPUTS factors are inputs;
 FACTOR= COST RICH    WRIT SCIN;
!     The inputs,    the outputs;
  F  =  89.39  64.3    25.2   223
        86.25  99      28.2   287
       108.13  99.6    29.4   317
       106.38  96      26.4   291
        62.40  96.2    27.2   295
        47.19  79.9    25.5   222;
  WGTMIN = .0005; ! Min weight applied to every factor;
  BIGM = 999999;  ! Biggest a weight can be;
 ENDDATA
!-----------------------------------------------------------;

! The Model;
! SUBMODEL Dea:
 ! IU = DMU we are currently considering;
! Try to make the score of DMU IU as high as possible;
```

```
  MAX = TSCORE;
    TSCORE = @SUM( FACTOR(J)|J #GT# NINPUTS:
                      F(INOW, J)* TW( J));

! Sum of inputs(denominator) = 1;
   [SUM21] @SUM( FACTOR( J)| J #LE# NINPUTS:
                   F( INOW, J)* TW( J)) = 1;

! Using DMU IU's weights, no DMU can score better than 1
  Note, Numer/Denom <= 1 implies Numer <= Denom;
   @FOR( DMU( K):
   [LE1]    @SUM( FACTOR( J)| J #GT# NINPUTS: F( K, J) * TW( J))
         <= @SUM( FACTOR( J)| J #LE# NINPUTS: F( K, J) * TW( J))
      );

! The weights must be greater than zero;
  @FOR( FACTOR( J): @BND( WGTMIN, TW, BIGM));
!ENDSUB;

CALC:

   @SET( 'TERSEO', 2);
   @SET( 'STAWIN', 0);

   ! Solve the DEA model for every DMU;
   @FOR( DMU( IU):
      INOW = IU;
      @SOLVE();
     ! Store the results;
      SCORE( IU) = TSCORE;
      @FOR( FACTOR( J):
         W(IU,J) = TW( J)
      );
   );
ENDCALC
```

```
DATA:
  @TEXT() = @WRITE( @NEWLINE( 2),
   32*' ','Factor Weight:', @NEWLINE( 1),
   '    DMU    Score', @WRITEFOR( FACTOR( I): '      ', FACTOR( I)),
@NEWLINE( 1)
  );
  @TEXT() = @WRITEFOR( DMU( D):  '     ', DMU( D), '   ',
    @FORMAT( SCORE( D), '6.3f'), '   ',
    @WRITEFOR( FACTOR( I):
      @FORMAT( W( D, I), '8.4f'), '   '
    ),
    @NEWLINE( 1));
ENDDATA
```

出力結果.

```
                          Factor Weight:
    DMU    Score      COST      RICH      WRIT      SCIN
     BL    1.000    0.0005    0.0149    0.0353    0.0005
     HW    0.910    0.0018    0.0086    0.0005    0.0031
     NT    0.962    0.0017    0.0082    0.0005    0.0030
     OP    0.912    0.0017    0.0085    0.0005    0.0031
     YK    1.000    0.0019    0.0092    0.0005    0.0033
     EL    1.000    0.0203    0.0005    0.0349    0.0005
```

2.6 TSP(LOOPTSP.lg4)

次は，米国の12都市のTSP問題を扱っている．しかし，DATA節で直接定義したデータを，例えばExcel上のデータから@OLE()で入力すれば，どのようなTSP問題にも対応できる．この問題では，米国の12都市で考えている．また計算時間のかかるIPでなく，LPで解くため計算時間がかからない．

```
MODEL:
 ! (TSPCUT) Traveling Salesman Problem.  Find the shortest tour that
   visits each city exactly once. Subtour elimination method.
 ! Keywords: TSP, traveling sales person, routing, tour;
 SETS:
  CITY;
  LINK( CITY, CITY):
      DIST,  ! The distance matrix;
```

```
        Y;  ! Y( I, J) = 1 if link I, J is in tour;
  SUBTOUR: TOURSIZE;
  SXC(SUBTOUR,CITY): FLAG;
 ENDSETS
DATA:
  SUBTOUR = 1..20; ! Number subtour cuts we allow;
  CITY = NYK ATL CHI CIN HOU LAX MON PHL PIT STL SND SNF;
   ! Distance matrix for cities of National League of
      US baseball, circa 2004. It need not be symmetric;
  DIST =
     0  864  845  664 1706 2844  396   92  386 1002 2892 3032
   864    0  702  454  842 2396 1196  772  714  554 2363 2679
   845  702    0  324 1093 2136  764  764  459  294 2184 2187
   664  454  324    0 1137 2180  798  572  284  338 2228 2463
  1706  842 1093 1137    0 1616 1857 1614 1421  799 1521 2021
  2844 2396 2136 2180 1616    0 2900 2752 2464 1842   95  405
   396 1196  764  798 1857 2900    0  424  514 1058 2948 2951
    92  772  764  572 1614 2752  424    0  305  910 2800 2951
   386  714  459  284 1421 2464  514  305    0  622 2512 2646
  1002  554  294  338  799 1842 1058  910  622    0 1890 2125
  2892 2363 2184 2228 1521   95 2948 2800 2512 1890    0  500
  3032 2679 2187 2463 2021  405 2951 2951 2646 2125  500    0;
ENDDATA
!-------------------------------------------------------------;
SUBMODEL TSP_CUT:
! Minimize total distance traveled;
 MIN = @SUM( LINK: DIST * Y);

! The Assignment constraints;
  @FOR( CITY( K):
  !  City K must be entered;
    @SUM( CITY( I)| I #NE# K: Y( I, K))= 1;
  !  City K must be departed;
    @SUM( CITY( J)| J #NE# K: Y( K, J))= 1;
  ! Cannot go to yourself;    Y( K, K) = 0;
   );
```

```
! Subtour cuts;
  @FOR( SUBTOUR(t):
!  FLAG(t,i) = 1 if city i is in subtour t;
    @SUM( CITY(I) | FLAG(t,i) #EQ# 1:
      @SUM( CITY(J) | FLAG(t,j) #EQ# 1: Y(i,j))) <= TOURSIZE(t) - 1;
      );
ENDSUBMODEL

CALC:
  @SET( 'TERSEO', 2);
  N = @SIZE( CITY);
  MXCUTS = @SIZE(SUBTOUR);
 ! Initially there are no subtour cuts;
 @FOR( SXC(t,i):
   FLAG(t,i) = 0;
    );

   ICUT = 1;
! Loop over subtour cuts, ICUT;
 QUIT1 = 1;
 @WHILE( QUIT1 :
!  Solve current version;
   @SOLVE( TSP_CUT);
!  Find subtour if any;
   TOURSIZE(ICUT) = 0;
!  Loop over cities KURSTOP to find subtour starting at 1;
   KURSTOP = 1;
   QUIT2 = 1;
   @WHILE( QUIT2:
!    Loop over possible next cities j;
     @FOR(CITY(J):
      @IFC( Y(KURSTOP, J) #GT# .5:
        NEXT1 = J;
      ););!    Next j;
      KURSTOP = NEXT1;
```

```
      TOURSIZE(ICUT) = TOURSIZE(ICUT) + 1;
      FLAG(ICUT,KURSTOP) = 1;
      !@WRITE( ' Next stop= ', CITY(KURSTOP),' Tour size=',TOURSIZE(ICUT),
@NEWLINE( 1));
      @IFC( KURSTOP #EQ# 1:  ! Back home/Completed the subtour?;
        QUIT2 = 0;
      );
         ); ! End loop over cities in subtour;
   ! If subtour is in fact a full tour, or out of space, get out;
    @IFC( TOURSIZE(ICUT) #EQ# N #OR# ICUT #GE# MXCUTS:
       !We are done;
       QUIT1 = 0;
       @FOR( LINK: Y = Y); !Fix Y;
    );
   ! Get ready for next cut;
    ICUT = ICUT + 1;
    );  !End loop over add cuts;
ENDCALC

SETS:
   LINKOPT( LINK) | Y( &1, &2) #GT# .5;
ENDSETS

CALC:
! Give simple report;
  @IFC( TOURSIZE(ICUT-1) #LT# N:
    @WRITE(' Sorry, optimum not found', @NEWLINE(1));
  @ELSE
    KURSTOP = 1;
    CUMDIST = 0;
    K = 1;
    @WRITE( 9*' ', 'Tour:    Distance:', @NEWLINE( 1));
    @WRITE( '    1:    ',CITY( KURSTOP),'  ', 8*' ','0',@NEWLINE(1));
    @FOR( CITY:
!     Get next city;
      @FOR( LINKOPT( I, J) | I #EQ# KURSTOP:
```

```
        NEXT = J;
        CUMDIST = CUMDIST + DIST( I, NEXT);
        K = K + 1;
        @WRITE( @FORMAT( K, '5.0f'),':    ',CITY( NEXT),
         '  ', @FORMAT( CUMDIST, '9.0f'),@NEWLINE(1));
      );
      KURSTOP = NEXT;
    );
  );
ENDCALC
END
```

出力結果.

	Tour:	Distance:
1:	NYK	0
2:	PHL	92
3:	PIT	397
4:	CIN	681
5:	ATL	1135
6:	HOU	1977
7:	SND	3498
8:	LAX	3593
9:	SNF	3998
10:	STL	6123
11:	CHI	6417
12:	MON	7181
13:	NYK	7577

2.7 ゲーム(LOOPTTT.lg4)

```
model:

!Two players (X and Y) "square off" in a familiar child's game;

sets:
  dim /1..3/:;
  board( dim, dim): score, x, y;
! there are 8 winning combinations;
```

```
   winners/w1..w8/:
      x_wins_w,   !=1 if x wins this combo;
      y_wins_w,   !=1 if y wins this combo;
      x_blocks_w, !=1 if x blocks y from winning this combo;
      y_blocks_w, !=1 if y blocks x from winning this combo;
      x_checks_w, !=1 if x checks this combo (i.e., holds two squares to
y's none);
      y_checks_w, !=1 if y checks this combo;
      x_count_w,  !number of squares held by x in this combo;
      y_count_w   !number of squares held by y in this combo;
   ;
   wxb( winners, dim, dim) /
! horizontal winning combos;
     w1,1,1 w1,1,2 w1,1,3
     w2,2,1 w2,2,2 w2,2,3
     w3,3,1 w3,3,2 w3,3,3
! vertical winning combos;
     w4,1,1 w4,2,1 w4,3,1
     w5,1,2 w5,2,2 w5,3,2
     w6,1,3 w6,2,3 w6,3,3
! diagonal winners;
     w7,1,1 w7,2,2 w7,3,3
     w8,1,3 w8,2,2 w8,3,1
   /:;
endsets

calc:
! the value of each square on the grid is set to the number
  of winning combos it appears in;
  @for( board: score = 0);
  @for( wxb( w, i, j): score( i, j) = score( i, j) + 1);
endcalc

! The primary objective is to win, followed by blocking, followed by
checking,
  followed by maximizing the raw score of squares held;
```

```
! Note: set some of the obj coefficients to 0 to allow a player to "beat"
  a handicapped opponent;
max =
   ( 10000 * x_wins + 1000 * x_blocks + 100 * x_checks + x_score) * x_wt
+
   ( 10000 * y_wins + 1000 * y_blocks + 100 * y_checks + y_score) * y_wt
;

! can only have one player on a square;
@for( board: x + y <= 1);

! each player gets one square per turn;
@sum( board: x) = x_turns;
@sum( board: y) = y_turns;

! a player wins if he has three squares in a row;
@for( winners( w): @for( wxb( w, i, j):
   x_wins_w( w) <= x( i, j);
   y_wins_w( w) <= y( i, j);
));
! sum up total wins for each player;
x_wins = @sum( winners: x_wins_w);
y_wins = @sum( winners: y_wins_w);

! count how many squares each player holds in each winning combo;
@for( winners( w):
   x_count_w( w) = @sum( wxb( w, i, j): x( i, j));
   y_count_w( w) = @sum( wxb( w, i, j): y( i, j));
);

! a player blocks if he spoils an opponent's winning combination;
@for( winners:
   ! x blocks y if x holds one square and y holds 2;
   2 * x_blocks_w <= y_count_w;
   y_count_w <= 3 - x_blocks_w;
   x_blocks_w <= x_count_w;
```

```
   x_count_w <= 3 - 2 * x_blocks_w;
   ! y blocks x if y holds one square and x holds 2;
   2 * y_blocks_w <= x_count_w;
   x_count_w <= 3 - y_blocks_w;
   y_blocks_w <= y_count_w;
   y_count_w <= 3 - 2 * y_blocks_w;
);
! sum up total blocks for each player;
x_blocks = @sum( winners: x_blocks_w);
y_blocks = @sum( winners: y_blocks_w);

! a player checks a winning combination if he holds two squares
  versus his opponent's none;
@for( winners:
   ! x checks combo?;
   2 * x_checks_w <= x_count_w;
   x_count_w <= 3 - x_checks_w;
   y_count_w <= 3 - 3 * x_checks_w;
   ! y checks combo?;
   2 * y_checks_w <= y_count_w;
   y_count_w <= 3 - y_checks_w;
   x_count_w <= 3 - 3 * y_checks_w;
);
! sum up total checks for each player;
x_checks = @sum( winners: x_checks_w);
y_checks = @sum( winners: y_checks_w);

! compute the raw score of squares held by each player;
x_score = @sum( board: score * x);
y_score = @sum( board: score * y);

! binaries;
@for( board: @bin( x); @bin( y));
@for( winners: @bin( x_blocks_w); @bin( y_blocks_w));
@for( winners: @bin( x_checks_w); @bin( y_checks_w));
```

```
data:
   ! Displays the layout of the final grid;
   @text() = @write( @newline(1), "Board #", x_turns +
y_turns,":",@newline( 1));
   @text() = @writefor( board( i, j): 3*" ",@if( x, "X", @if ( y, "O", ".")),
     @if( j #eq# @size( dim), @newline( 1), ""));
   @text() = @write( @newline( 1));
   @text() = @write( @if( x_wins, "!!!X WINS!!!", @if( y_wins, "!!!Y WINS!!!",
""")));
   @text() = @write( @if( x_turns + y_turns #eq# @size( board) #and#
     x_wins #lt# 1 #and# y_wins #lt# 1, "!!!CAT...NO WINNER!!!",""));
enddata

init:
   x_wins = 0;
   y_wins = 0;
endinit

calc:

   ! Set patameters;
   @set( 'DEFAULT');
   @set( 'OROUTE', 1);
   @set( 'TERSEO', 2);
   @set( 'WNLINE', 10000);
   @set( 'LINLEN', 150);
   @set( 'STAWIN', 0);

   ! no turns taken yet;
   x_turns = 0;
   y_turns = 0;

   ! let's play;
   @for( board | x_wins #lt# 1 #and# y_wins #lt# 1 #and#
    ( x_turns + y_turns) #lt# @size( board):
```

```
      ! x moves first;
      x_turns = x_turns + 1;
      x_wt = 1; y_wt = 1 - x_wt;
      @solve();

      ! fix all of x's moves;
      @for( board:
        @ifc( x:
          x = 1;
        );
      );

      ! y moves next;
      @ifc( x_wins #lt# 1 #and# x_turns + y_turns #lt# @size( board):
        y_turns = y_turns + 1;
        x_wt = 0; y_wt = 1 - x_wt;
        @solve();

        !fix all y's moves;
        @for( board:
          @ifc( y:
            y = 1;
          );
        );
      );
    );

endcalc
end
```

出力結果

```
model:

!Two players (X and Y) "square off" in a familiar child's game;

sets:
```

```
   dim /1..3/:;
   board( dim, dim): score, x, y;
! there are 8 winning combinations;
   winners/w1..w8/:
      x_wins_w,   !=1 if x wins this combo;
      y_wins_w,   !=1 if y wins this combo;
      x_blocks_w, !=1 if x blocks y from winning this combo;
      y_blocks_w, !=1 if y blocks x from winning this combo;
      x_checks_w, !=1 if x checks this combo (i.e., holds two squares to
y's none);
      y_checks_w, !=1 if y checks this combo;
      x_count_w,  !number of squares held by x in this combo;
      y_count_w   !number of squares held by y in this combo;
   ;
   wxb( winners, dim, dim) /
! horizontal winning combos;
     w1,1,1 w1,1,2 w1,1,3
     w2,2,1 w2,2,2 w2,2,3
     w3,3,1 w3,3,2 w3,3,3
! vertical winning combos;
     w4,1,1 w4,2,1 w4,3,1
     w5,1,2 w5,2,2 w5,3,2
     w6,1,3 w6,2,3 w6,3,3
! diagonal winners;
     w7,1,1 w7,2,2 w7,3,3
     w8,1,3 w8,2,2 w8,3,1
   /:;
endsets

calc:
! the value of each square on the grid is set to the number
  of winning combos it appears in;
  @for( board: score = 0);
  @for( wxb( w, i, j): score( i, j) = score( i, j) + 1);
endcalc
```

```
! The primary objective is to win, followed by blocking, followed by
checking,
  followed by maximizing the raw score of squares held;
! Note: set some of the obj coefficients to 0 to allow a player to "beat"
  a handicapped opponent;
max =
   ( 10000 * x_wins + 1000 * x_blocks + 100 * x_checks + x_score) * x_wt
+
   ( 10000 * y_wins + 1000 * y_blocks + 100 * y_checks + y_score) * y_wt
;

! can only have one player on a square;
@for( board: x + y <= 1);

! each player gets one square per turn;
@sum( board: x) = x_turns;
@sum( board: y) = y_turns;

! a player wins if he has three squares in a row;
@for( winners( w): @for( wxb( w, i, j):
   x_wins_w( w) <= x( i, j);
   y_wins_w( w) <= y( i, j);
));
! sum up total wins for each player;
x_wins = @sum( winners: x_wins_w);
y_wins = @sum( winners: y_wins_w);

! count how many squares each player holds in each winning combo;
@for( winners( w):
   x_count_w( w) = @sum( wxb( w, i, j): x( i, j));
   y_count_w( w) = @sum( wxb( w, i, j): y( i, j));
);

! a player blocks if he spoils an opponent's winning combination;
@for( winners:
   ! x blocks y if x holds one square and y holds 2;
```

```
   2 * x_blocks_w <= y_count_w;
   y_count_w <= 3 - x_blocks_w;
   x_blocks_w <= x_count_w;
   x_count_w <= 3 - 2 * x_blocks_w;
   ! y blocks x if y holds one square and x holds 2;
   2 * y_blocks_w <= x_count_w;
   x_count_w <= 3 - y_blocks_w;
   y_blocks_w <= y_count_w;
   y_count_w <= 3 - 2 * y_blocks_w;
);
! sum up total blocks for each player;
x_blocks = @sum( winners: x_blocks_w);
y_blocks = @sum( winners: y_blocks_w);

! a player checks a winning combination if he holds two squares
  versus his opponent's none;
@for( winners:
   ! x checks combo?;
   2 * x_checks_w <= x_count_w;
   x_count_w <= 3 - x_checks_w;
   y_count_w <= 3 - 3 * x_checks_w;
   ! y checks combo?;
   2 * y_checks_w <= y_count_w;
   y_count_w <= 3 - y_checks_w;
   x_count_w <= 3 - 3 * y_checks_w;
);
! sum up total checks for each player;
x_checks = @sum( winners: x_checks_w);
y_checks = @sum( winners: y_checks_w);

! compute the raw score of squares held by each player;
x_score = @sum( board: score * x);
y_score = @sum( board: score * y);

! binaries;
@for( board: @bin( x); @bin( y));
```

```
@for( winners: @bin( x_blocks_w); @bin( y_blocks_w));
@for( winners: @bin( x_checks_w); @bin( y_checks_w));


data:
   ! Displays the layout of the final grid;
   @text() = @write( @newline(1), "Board #", x_turns +
y_turns,":",@newline( 1));
   @text() = @writefor( board( i, j): 3*" ",@if( x, "X", @if ( y, "O", ".")),
     @if( j #eq# @size( dim), @newline( 1), ""));
   @text() = @write( @newline( 1));
   @text() = @write( @if( x_wins, "!!!X WINS!!!", @if( y_wins, "!!!Y WINS!!!",
""")));
   @text() = @write( @if( x_turns + y_turns #eq# @size( board) #and#
     x_wins #lt# 1 #and# y_wins #lt# 1, "!!!CAT...NO WINNER!!!",""));
enddata

init:
   x_wins = 0;
   y_wins = 0;
endinit

calc:

   ! Set patameters;
   @set( 'DEFAULT');
   @set( 'OROUTE', 1);
   @set( 'TERSEO', 2);
   @set( 'WNLINE', 10000);
   @set( 'LINLEN', 150);
   @set( 'STAWIN', 0);

   ! no turns taken yet;
   x_turns = 0;
   y_turns = 0;
```

```
   ! let's play;
   @for( board | x_wins #lt# 1 #and# y_wins #lt# 1 #and#
    ( x_turns + y_turns) #lt# @size( board):

      ! x moves first;
      x_turns = x_turns + 1;
      x_wt = 1; y_wt = 1 - x_wt;
      @solve();

      ! fix all of x's moves;
      @for( board:
         @ifc( x:
            x = 1;
         );
      );

      ! y moves next;
      @ifc( x_wins #lt# 1 #and# x_turns + y_turns #lt# @size( board):
         y_turns = y_turns + 1;
         x_wt = 0; y_wt = 1 - x_wt;
         @solve();

         !fix all y's moves;
         @for( board:
            @ifc( y:
               y = 1;
            );
         );
      );
   );

endcalc
end
```