A Matlab Implementation to Assist Model Structure Analysis (MSA)

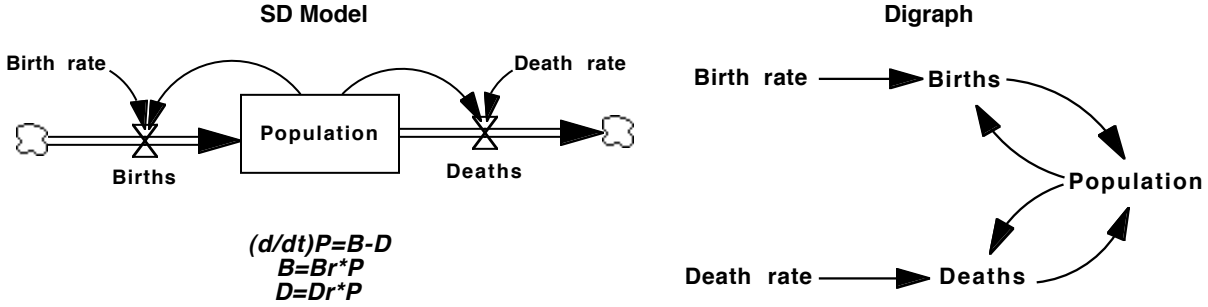
ROGELIO OLIVA
Harvard Business School
Morgan Hall T87
Boston, MA 02163
Ph 617-495-5049 Fx 671-496-5265
Email: roliva@hbs.edu

D-4864-2

Introduction

The MSA package is a set of user-defined functions (M-files) developed for Matlab® to facilitate the analysis of structural complexity of system dynamics models. Full description of the algorithms used in these functions, and the rationale behind their usage is described in Oliva (2003)[1]. The functions use as input the digraph representation of the mode structure. A directed graph or digraph *G* is a pair (*V,E*), where *V*(G) is a finite set of elements, called vertices, and *E(G)* is a binary relation on *V* – a subset of ordered pairs of elements of *V(G)*. The elements of *E(G)* are called edges and constitute the edge set of *G* . We can represent the structure of a system dynamics model as a digraph, where the variables are the vertices and the edges are the relationship "is used in," i.e., there is a directed edge ($u \rightarrow v$) if *u* is used as an argument in the computation of *v*. To facilitate computations, a digraph is often represented as an adjacency matrix.[2] The adjacency matrix representation of a digraph is a square matrix *A* of size |*V*|, where each row (and column) represents a vertex (vertices are numbered 1,2, … |*V*|). The entries in the matrix are restricted to zero and one, where A_{uv}=1 IFF $(u,v) \in E(G)$. The ones in row A_u represent the successor set (*Succ*[*u*]) for vertex *u*, while the ones in column A_u represent the predecessor set (*Pred*[*u*]) for vertex *u*. Figure 1 illustrates the mapping of model structure into a digraph and its corresponding adjacency matrix representation. A separate utility has been developed to automate the generation of an adjacency matrix from a text file containing the model documentation.[3]

Figure 1 Graph representation of system structure



(d/dt)P=B-D
B=Br*P
D=Dr*P

Adjacency matrix

	Br	*B*	*P*	*Dr*	*D*
Br	0	1	0	0	0
B	0	0	1	0	0
P	0	1	0	0	1
Dr	0	0	0	0	1
D	0	0	1	0	0

[1] Oliva, R. 2003. Model structure analysis through graph theory: Partition heuristics and feedback structure decomposition. Harvard Business School Working Paper 04-016. Available at http://www.people.hbs.edu/roliva/research/sd/.

[2] Although system dynamics models normally generate sparse graphs that are represented more efficiently through an adjacency-list (Cormen e TH, Leiserson CE, Rivest RL 1990. *Introduction to Algorithms*. MIT Press. Cambridge, MA), the adjacency-matrix representation will be used throughout this presentation because of the simplifications it allows in the codification of algorithms.

The functions in the MSA package perform a model partition based on data availability, identify the basic elements of a model structure as defined by Warfield[4] –levels, cycles and feedback loops– and identify and hierarchize a model's independent loop set[5]. The package also includes multiple display and plotting functions to facilitate the analysis and visualization of the system's structure. The sequence suggested below illustrates how to use these functions in the context of maximizing the use of data available for calibrating the model.

Functions have been tested with Matlab®[6] versions 5.2 and 6.5 for Macintosh, and versions 5.2 and 5.3 for PC and Unix.[7] At one time the functions were implemented for RLab[8] –a freeware scientific programming environment similar to Matlab® – but I found it much easier to develop the tools for the IO interface of Matlab®. If someone is interested in implementing the MSA package for RLab or compiling the package as a stand-alone application please contact me at roliva@hbs.edu or 617-495-5049.

The rest of this document assumes that the user has some familiarity with the use of Matlab®.

Installing MSA functions

After downloading and decompressing the functions,[9] you should have a folder entitled MSA with a series of text documents with the suffix `.m`[10]. Move the folder to the Matlab directory and start up Matlab. Modify Matlab's current search path to include the MSA folder just installed.

```
»path(path,'/applications/Matlab/MSA')
```
[11]

Verify that Matlab has access to the MSA functions with the `what` command. The command should report the 14 M-files contained in the MSA package:

```
»what  MSA
dat_part            d_pred              param               reach
d_cyles             d_succ              p_loop              structure
d_levels            handles             p_pred
d_loops             loop_h              p_succ
```

[3] Oliva, R. 2003. Vensim® Model to Adjacency Matrix Utility. Harvard Business School. Boston, MA. June 02, 2003. Available at http://www.people.hbs.edu/roliva/research/sd/.
[4] Warfield, J.N. 1989. *Societal Systems: Planning, policy and complexity*. Intersystems Publications. Salinas, CA.
[5] Kampmann, C.E. 1996. *Feedback loop gains and system behavior*. Proceedings of the 1996 Int. System Dynamics Conference. Cambridge, MA. Pg. 260-263.
[6] http://www.mathworks.com
[7] Most of the functions were developed under v4.2. Reinstating v4.2 functionality would only require the unpacking of the subordinated functions included in some M-files.
[8] http://www.eskimo.com/~ians/rlab.html
[9] Functions are available at http://www.people.hbs.edu/roliva/research/sd/
[10] If you have difficulties downloading the .hqx or the .zip archives, you can download one-by-one the 14 functions using the text format. Save the functions in a directory entitled MSA and follow the rest of the steps for installation.
[11] The syntax of the `path` command varies from platform to platform to match the directory description syntax of the operating system. Check `help path` on the Matlab prompt for more details.

A basic description of each function's usage as well as the specification of the required inputs and generated outputs can be obtained through the `help` command at the Matlab prompt:

```
»help  function_name
```

Suggested sequence for Model Structure Analysis

The functions in the MSA package are quite versatile and can be used in different ways to develop an intuition about the structural complexity of a model. The following steps have proven to be a successful sequence to develop such intuition, but in no way are they intended to be the only way to use the MSA functions. The sequence makes use and illustrates the functionality of the 14 functions included in the MSA package. For more information on each function use the `help` command at the Matlab prompt.

1. Load model structure and names

Assign the adjacency matrix describing the system structure and the variable name matrix to variables in the current workspace. MATLAB *.m file generated by the Vensim® Model to Adjacency Matrix Utility contains a simple function, named as the original model, that returns the adjacency matrix and the name list for model variables into a structured array with components .adj and .nms. See the READ_ME file[12] for the Vensim® Model to Adjacency Matrix Utility for instructions on how to generate the *.m files to import them into MATLAB (Make sure that the *.m files containing the structure and the names are in a path where MATLAB can find them).

```
»A=my_model;
```

Each component of the structured array can be reached through sub-indexing. The adjacency matrix (a sparse squared binary matrix with the same number of rows as model variables) is accessible in `A.adj`, and the list of names (character vector) in `A.nms`.

The `spy` command can give you a sense of the density of interconnections in the model structure; a point in the sparcity graph represents a direct interconnection between variables.

```
»spy(A.adj)
```

A traditional equation-by-equation report of the system structure can be generated using the 'display' commands for predecessors and successors. `d_pred` lists all the variables that go into an equation –equivalent to the relationship *uses*– and `d_succ` lists all the equations where a variable is utilized –equivalent to the relationship *used in*.

```
»d_pred(A,VARS)
»d_succ(A,VARS)
```

[12] http://www.people.hbs.edu/roliva/research/sd/mdl2bin_READ_ME.htm

If variable numbers (VARS) are not specified, these functions will list the predecessors and successors for ALL model variables.

It is also possible to identify the model parameters –elements without predecessors– from the adjacency matrix and identify the elements where parameters are being used:

```
»PRM=param(A);
»d_succ(A,PRM);
```

To list of parameters, or access the name of any model variable, it is possible to call directly into the .nms partition of the adjacency matrix.

```
»A.nms(PRM,:)
»A.nms(17,:)
»A.nms([13  16  20],:)
```

2. Identify data availability

Incorporate in a vector the variables for which time series are available:

```
»DATA=[2  8  26  33  45  49]
```

At this stage is possible to visualize the potential impact of data availability on determining other model variables.

```
»p_succ(A,DATA);
```

3. Partition model according to data availability.

The dat_part function partitions a system described as an adjacency binary matrix (ADJ) according to data availability (DATA).

```
»PART=dat_part(A,NMS)
```

The output, PART, is a structure that for each data series available contains the sub-components [y,x,beta,eqs] corresponding to the dependent variable, independent variables, parameters that can be estimated and equations involved in the estimation. Each component of the structure can be reached through the following sub-indexing:

PART.y, PART.x, PART.beta and PART.eqs.

The full specification of equation i can be reached by just specifying the subindex in the PART structure:

```
»PART(i)
```

An individual component can be reached by specifying the subindex in the PART structure (the equation number) AND the component:

```
»PART(1).y
»PART(1).x
»PART(3).beta
»PART(2).eqs
```

In the unlikely case that all parameters are reachable with the existing data, steps 1 through 4 is all that might be needed in terms of structural analysis. Most of the time, however, it will be necessary to develop a strategy on how to do the partial model calibrations so that data and insights from other sections of the model can be used in the process.

4. Generate the reachability matrix for the model.

Although it is not needed for the analysis of the system structure (all the information in it is contained in the ADJ matrix), it gives a clear visual indication of how densely connected the model really is. While SD models normally generate sparse adjacency matrices (variables are affected only by two or three other variables), the reachability matrices of SD models tend to have very high densities due to the number of feedback loops in the structure. The `reach` function returns an array with the same structure as the adjacency matrix – with subcomponents `.adj` (sparse square binary matrix) and `.nms` (character array with variable names).

```
»R=reach(A);
»spy(R.adj);
```

5. Identify elements of model structure: levels, cycles and feedback loops

Run basic function to decompose the structure of the system into levels, identify the Shortest Independent Loop Set (ILS) or the Minimal Shortest Independent Loops Set (MSILS) for the model,[13] and order the model variables in a way to minimize the distance of interactions to the main diagonal of the adjacency matrix.

```
»STR=structure(A);              % For SILS
»STR=structure(A,-1);           % For MSILS
```

The function returns an arrayed structure with five components:

`adj`	(sparse-binary)	a replica of the input matrix.
`nms`	(character vector)	a replica of the input name vector.
`lev`	(sparse-binary)	each row identifies the elements that belong to a level partition.
`cyc`	(sparse-binary)	each row identifies the elements that belong to a cycle.
`ecc`	(sparse)	each row contains the eccentricity of the elements of a cycle.

[13] See Oliva, R. 2003. Model structure analysis through graph theory: Partition heuristics and feedback structure decomposition. Harvard Business School Working Paper 04-016. Available at http://www.people.hbs.edu/roliva/research/sd/.

`fbl`	(array of double)	for every cycle partition, a matrix contains in each row the ordered set of elements the feedback loops that constitute the shortest independent loop set.
`idx`	(structure)	diverse information to facilitate the plotting of the sparsity pattern.

Each component can be reached wit a three-letter subindex in lower case: `STR.adj`, `STR.nms`, `STR.lev`, `STR.cyc`, `STR.ecc`, `STR.fbl`, and `STR.idx`.

The function identifies separately the elements in a cycle (`cyc`). The mapping of each cycle into its corresponding level is reported in the progress output that the function generates, and all elements of the cycle partition are included in their corresponding level (`lev`).

`ecc` is a matrix that has a row for each cycle partition. Each cell contains the eccentricity of the elements belonging to that cycle partition with respect to the cycle partition –the entry for a node not belonging to the cycle partition is 0. Eccentricity of a node is defined (in graph theory) as the longest of the shortest paths between that node and every other node in the graph –this measures how far the node is of its furthest node in the cycle partition. The nodes with the smallest excentricity are said to be in the center of the graph.

`fbl` is an array that contains a matrix for each cycle partition. In these matrices, each row has the ordered set of elements in a feedback loop. Self reference loops are not reported (loops of length 1), and the loop list is sorted by loop length (from shortest to longest). Note that `fbl` contains the SILS or MSILS (depending on selected option) for each cycle partition –each reported loop is linearly independent, and, in the case of the MSILS, there are no redundant loops in the reported set.[14]

`idx` contains a structure with three different components to facilitate displaying the sparsity pattern of the adjacency matrix. The first part (`idx`) contains and index of all the nodes ordered by level, including the cycle elements within the level, and orders elements within a level to minimize the distance of interactions to the main diagonal –bandwidth. The other two elements of `idx` (`tks` and `lbl`) contain information about the level partition to be used by the `p_pred` and `p_succ` functions.

5a. Displaying the model structure

At this stage is possible to see the elements of the model structure with the following 'display' functions:

```
»d_levels(STR,PRT)
»d_cycles(STR,PRT)
```

Each function generates the list of elements that belong to partition `PRT`. If no partition number (`PRT`) is specified when calling these functions, the elements of ALL partitions are listed. The

[14] See Oliva (2003) for formal definition of Shortest Independent Loop Set and Minimal Shortest Independent Loop Set.

d_cycles function also reports node eccentricity within the cycle and sorts the output by node eccentricity (from the center of the graph to its edges).

The function

```
»d_loops(STR,PRT,LPN);
```

displays the variable names of the elements of the LPN feedback loop in the PRT cycle partition. If no loop number (LPN) is provided, the function lists the elements of all the loops in the identified cycle partition. If no cycle partition (PRT) is provided, the function lists the elements of ALL the feedback loops in the model.

Depending on the type of analysis you are doing, you might find it useful to create vectors with the elements of each level, cycle, or feedback loop. The function handles creates a set of handles to the set of partitions included in the structured matrix STR.

```
»H=handles(STR)
```

Handles are provided for levels (H.l01, H.l02 ...) cycles (H.c01, H.c02 ...) and feedback loops. The handles for feedback loops are provided with 3 digits PLL (H.f101, H.f102 ... H.f201, H.f202 ...) where P is a cycle partition and LL the

The display functions d_pred and d_succ –defined above– also function taking in the full structured array matrix or handles as input.

```
»d_pred(STR,VARS);
»d_succ(STR,VARS);
»d_pred(STR,H.f###);
»d_succ(STR,H.l###);
```

Two 'plotting' functions, p_pred and p_succ, further facilitate the exploration of the sparcity pattern of the adjacency and reachability matrices highlighting in a different color the predecessors or successors of a selected group of variables. The following are suggested uses for these functions to visualize the segmentation of the model structure.

```
»p_pred(A,H.l01);
»p_pred(STR,3);
»p_pred(R,H.l##);
»p_succ(R,[4  5  16]);
»p_pred(R,PRM);
»p_succ(STR,PRM);
```

If the input matrix for the plotting functions is the full structural array STR (the output of the structure function), the sparsity plot is sorted by level partitions and the partitions are identified in the plot.

Loops can be traced in the sparcity graph using the function p_loop (note that it only takes as input the full structural array STR):

```
»p_loop(STR,LOOP#)
»p_loop(STR,LOOP#,0)
```
for the sparcity graph not to be sorted by `STR.idx.idx`.

`LOOP#` can either be a loop handle (see handle function), or a direct reference to a loop according to the `PLL` format, where `P` is a digit referring to the cycle partition, and `LL` are two digits identifying the loop within the partition.

The sorted matrix and visual analysis of the sparcity graphs created with `p_pred` and `p_succ` allow the development of a sequence of estimation steps –from high to low level variables– that create new data as confidence is developed in partial sections of the model.

As more parameters are estimated and the estimation process yields reliable values for intermediate variables, these new data series can be incorporated into the data set (step 3) and the model can be re-partitioned with the new data availability.

Unfortunately, most of the complexity in SD models lies in the feedback loops –normally contained in one or two cycle sets– and a times is necessary to explore the relationships among feedback loops to maximize the value of the data available.

6. Identify loop hierarchy

The function `loop_h` develops a loop hierarchy of the loops in a particular cycle partition (`PRT`), based on the *inclusion* relation. A loop **A** is said to be included in loop **B** one if all the elements of **A** are also present in **B**.

```
»LPS=loop_h(STR,   PRT)
```

The result is presented as a level structure matrix `LPS.lev` and the corresponding matrix of inclusions `LPS.str`. The function also generates two plots of the inclusion graph. Because `STR.fbl` is an independent loop set (a great simplification from the total number of feedback loops normally available in a SD model), the loop hierarchy structure does not contain cycle partitions, is relatively flat (2 to 4 levels), and not many 'inclusions' (loops contained in other loops) are found. The inclusion graph, however, can be used to develop an incremental testing sequence vase on building confidence in simple structures first –the inner loops in the model– and moving into more complex and interdependent structures. The first plot o

(Note that neither the length of the loop nor its relative position in the loop hierarchy structure, has anything to do with its relative dominance of the system's behavior).

Appendix A.

Direct access to model structure.

It is possible to access the partitions in the `STR` matrix directly using regular MATLAB commands. The elements of a row `i` in the `lev` or `cyc` matrices can be identified through the following commands:

```
»find(STR.lev(i,:))
»STR.nms(find(STR.lev(i,:)),:)          to see the variable names.
»find(STR.cyc(i,:))
»STR.nms  (find(STR.cyc(i,:)),:)        to see the variable names.
```

The elements of the `i` loop in the `j` cycle partition can be reached through the following command:

```
»nonzeros(STR.fbl(i,:,j))
»STR.nms(nonzeros(STR.fbl(i,:,j)),:)        to see the variable names.
```

The index is useful to visualize the clustering of relationships and the "tightly-coupled sectors of the model".

```
»spy((A(STR.ixd.idx,STR.idx.idx));
»spy((R(STR.ixd.idx,STR.idx.idx));
```

To see the name of a specific variable `i` in the indexed sparsity plot:

```
»STR.nms(STR.idx.idx(i,:))
```

Appendix B.

Example

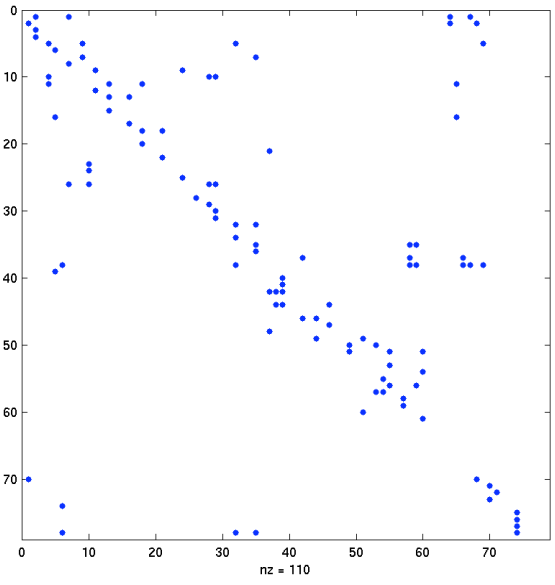
The following sequence of commands illustrates the use of the functions and the output they generate for a sector of the model used by Oliva and Sterman on their study of erosion of service quality[15]. The MODEL.m file containing the adjacency matrix (**.adj**) and a name list (**.nms**) was created using the Vensim® Model to Adjacency Matrix Utility.

The following conventions will be followed for the reminder of the appendix:

a) Commands given to MATLAB will be in bold; "%" indicates the beginning of a comment
b) Function output in regular text; "…" indicates that part of the output has been omitted

First, this information is loaded into the MATALAB workspace's variables **A**.

```
»A=model                    % load  adjacency  matrix
A =
    nms: [78x40 char  ]
    adj: [78x78 sparse]

»spy(A.adj)                 % see  sparsity  pattern  for  A
```

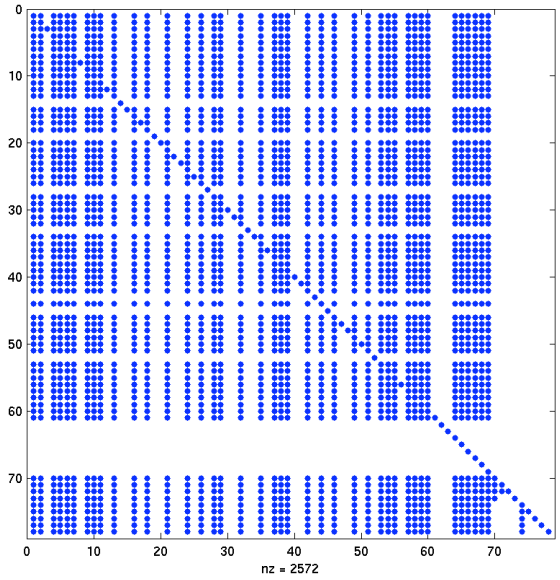


[15] Oliva, R., and J. D. Sterman. "Cutting Corners and Working Overtime: Quality Erosion in the Service Industry." *Management Science* 47, no. 7 (July 2001): 894-914. The model is fully documented and available at: http://www.people.hbs.edu/roliva/research/service/esq.html. The model used for this example is the same model available in the website but without the variables in the quality sector that were not active in the final calibration of the model.

```
»d_pred(A,9)              % id predecesors  for  variable  9
work pressure
     service capacity
     desired service capacity

»d_succ(A,9)              % id successors  for  variable  9
work pressure
     work intensity
     effect of wp on to

»P=param(A)               % id model  parameters
P =
  Columns 1 through 12
     3     8    12    14    15    17    19    20    22    23    25    27
  Columns 13 through 24
    30    31    33    34    36    40    41    43    45    47    48    50
  Columns 25 through 35
    52    56    61    62    63    72    73    75    76    77    78

»A.nms(P,:)               % list  names  of  model  parameters
MIN RESIDENCE T FOR AN ORDER
DESIRED DELIVERY DELAY
BETA
...
a absenteeism
HWE

»d_succ(A,P)              % see  immediate  succesors  of  model  parameters
MIN RESIDENCE T FOR AN ORDER
     order fulfillment
DESIRED DELIVERY DELAY
     desired service capacity
BETA
     work intensity
...
a absenteeism
     absenteeism
HWE
     on office service capacity
     Perceived Labor Productivity
     Desired Labor
     absenteeism

»D=[1  2  5  10  11  35  37  38  49  70  74]   %id  vars.  for  which  data  is  available
D =
     1     2     5    10    11    35    37    38    49    70    74

»A.nms(D,:)               % list  variables  for  which  data  is  available
ans =
Service Backlog
order fulfillment
service capacity
time per order
```

```
work intensity
Desired Labor
turnover rate
total labor
hiring rate
customer orders
absenteeism

»dat_part(A,D)            % execute  the  data  availability   partition
Identifying model parameters ...
...
y:
      time per order
x:
      service capacity
      Service Backlog
beta:
      MIN PROCESSING TPO
      ALPHA
      DESIRED DELIVERY DELAY
      TTDN
      TTUP
eqs:
      time per order
      effect of wp on to
      Desired To
      work pressure
      dto chg
      desired service capacity
      t to adjust dto
...
Parameters not reachable from data available:
      INITIAL FE
      INITIAL FA
      INITIAL DTO
      INITIAL P ELF
      INITIAL EXPERIENCED PERSONNEL
      INITIAL ROOKIES
      INITIAL VACANCIES
      TPO ON DQ GRAPH
      QP ON TO GRAPH
ans =
1x11 struct array with fields:
    y
    x
    beta
    eqs

»R=reach(A)               % generate  the  reachability   matrix
R =
    adj: [78x78 sparse]
    nms: [78x40 char  ]
```

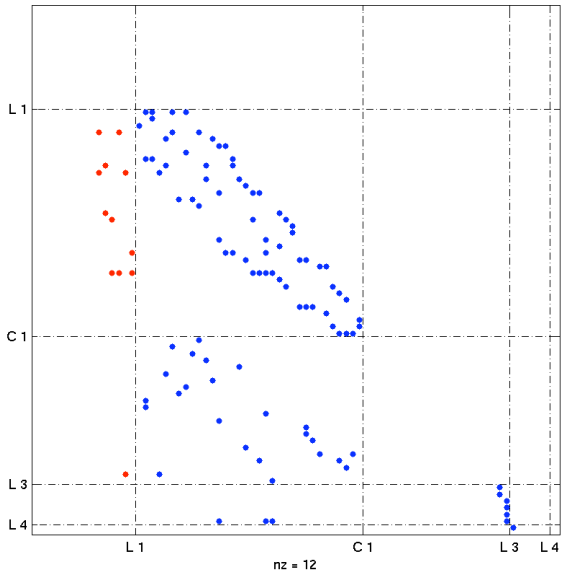
```
»spy(R.adj)                % see sparsity pattern for R
```




```
»S=structure(A)            % identify elements of model structure
Calculating Reachability Matrix ...
Identifying Level Partitions ...
  Cycle 1 belongs to level 2.
  Identifying Loops within Cycle 1 ...
    Reducing Cycle 1 to Independent Loop Set ...
Indexing Variables by Level ...
S =
    adj: [78x78 sparse]
    nms: [78x40 char  ]
    lev: [ 5x78 sparse]
    cyc: [ 1x78 sparse]
    ecc: [ 1x60 sparse]
    fbl: [23x14 double]
    idx: [ 1x1  struct]

»d_cycles(S,1)             % list elements of 1st (only) cycle partition
Cycle   1
  Ecc Variable
    7 Rookies
    8 turnover rate
    8 total labor
...
   14 Desired Labor
   15 Perceived Labor Productivity

»d_levels(S,3)             % list elements of 3rd level partition -vars &
                           parameters that feed into the cycle partition
Level   3
     MIN RESIDENCE T FOR AN ORDER
     DESIRED DELIVERY DELAY
     BETA
```

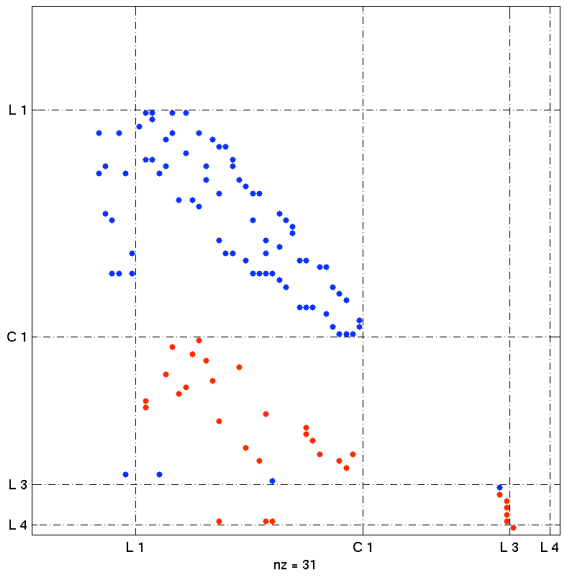
```
...
     customer orders
     absenteeism

»d_loops(S)              % list loops in model (all in first cycle)
Cycle  1
  Loop   1
    Service Backlog
    order fulfillment
  Loop   2
    Desired To
    dto chg
  Loop   3
    Rookies
    experience rate
...
 Loop  23
    Service Backlog
    desired service capacity
    Desired Labor
    replacement rate
    desired hiring
    indicated labor order rate
    labor order rate
    Vacancies
    hiring rate
    Rookies
    effective labor fraction
    service capacity
    potential order fulfillment
    order fulfillment

»d_loops(S,1,16)         % list elements of loop 16 (first cycle partition)
  Loop  16
   Service Backlog
   desired service capacity
   work pressure
   work intensity
   Fatigue E
   effect of fatigue on prod
   service capacity
   potential order fulfillment
   order fulfillment

»H=handles(S)            % create hanldels for model partitions
H =
     l01: [14 19 27 33 43 45 52 62 63 64 65 66 67 68 69]
...
     l05: 72
     c01: [1x34 double]
    f101: [1 2]
...
    f123: [1 7 35 58 57 54 60 51 49 44 39 5 4 2]
```

```
»p_pred(S,H.l01)          % show predecesors to variables in level 01
```

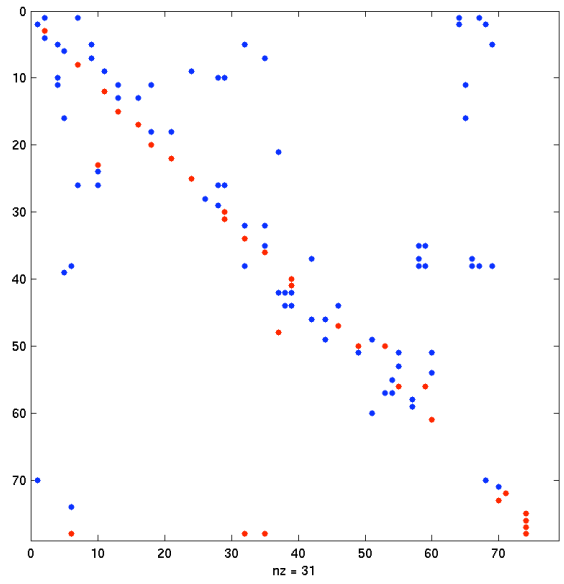



```
»d_pred(S,H.l01)          % list predecesors to variables in level 01
INITIAL FE
INITIAL FA
INITIAL P ELF
...
average productivity of labor
     service capacity
     total labor
```

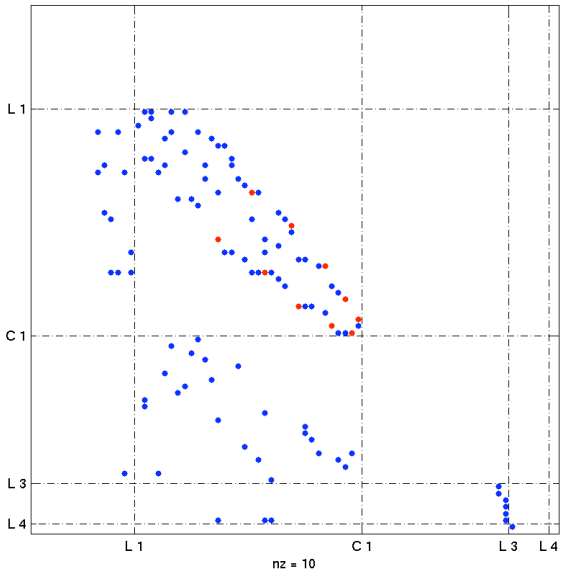
```
»p_succ(S,P)              % show succesors to model parameters
```



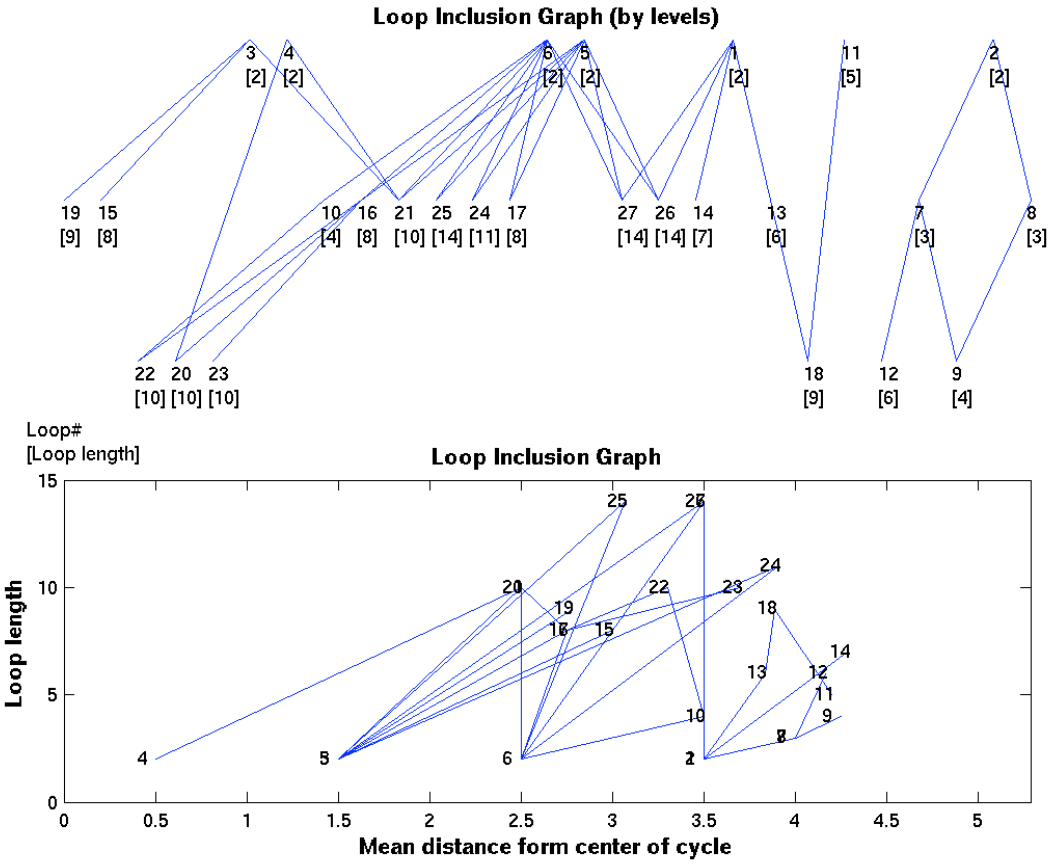
```
»p_succ(A,P)              % same information in original adj. matrix
```




```
»p_loop(S,123)            % show indicence of loop 23 on partition 1
```




```
»L=loop_h(S,1);           % show loop hierarchy structure (cycle 1)
Identifying inclusions within cycle loop set ...
Reducing to Adjacency Matrix ...
Calculating Reachability Matrix ...
Identifying Level Partitions ...
Indexing for plot coordenates ...
L =
    lev: [ 3x23 sparse]
    str: [23x23 sparse]
```


Loop Inclusion Graph (by levels)
3 [2]
4 [2]
6 [2]
5 [2]
1 [2]
11 [5]
2 [2]
19 [9]
15 [8]
10 [4]
16 [8]
21 [10]
25 [14]
24 [11]
17 [8]
27 [14]
26 [14]
14 [7]
13 [6]
7 [3]
8 [3]
22 [10]
20 [10]
23 [10]
18 [9]
12 [6]
9 [4]
Loop#
[Loop length]
Loop Inclusion Graph
Loop length
Mean distance form center of cycle