

A Stochastic Programming extension for FarmDyn

Wolfgang Britz, Institute for Food and Resource Economics, University Bonn, February 2016

Contents

Background.....	1
Changes in the equation structure.....	1
Generating the random variable(s) and the decision tree	3
Introduction of the random variable(s).....	7
Further extensions.....	8
References.....	8

Background

The FarmDyn model comprises already since the first versions the possibility to introduce different states-of-nature for input and output prices and related state-contingent variables in crop production and feeding. However, these states-of-natures are not linked via trees such that there were no true dynamic impacts of the state-contingent decisions. Rather, longer term investment decision needed to be taken such that the available machinery park or manure silo capacity must allow to manage all states-of-natures. In case of machine depreciation based on use, the maximum use in a year and state-of-nature defines the physical depreciation. The long term decision variables (investments, off-farm labor use, herd size) are not stage contingent in that version.

The extension discussed in here introduces scenario trees and renders all variables in the model stage contingent to generate a fully dynamic Stochastic Programming (SP) approach. That is only feasible in conjunction with a tree reduction approach: even if we would only allow for two different states in each year (= decision node), we would end up after twenty years with $2^{10} \sim 1$ Million leaves in the trees. Given the number of variables and equations in any year, the resulting model would be impossible to generate and solve. In the following, we briefly discuss the changes to model structure and the how the decision tree and the related random variable(s) are constructed.

Changes in the equation structure

As mentioned above, all variables in the model are state-contingent with the SP module switched on. They carry hence, as do the equations, an additional index “nCur” which indicates the current node in the decision tree. Equally, the node needs to be linked to the correct year, achieved by a dollar operator and the t_n set, for instance

```

*
*  ---- yearly labour restriction
*
LabTot_(tCur(t),nCur,s) $ t_n(t,nCur) ..

    sum(m, v_labTot(t,nCur,m,s)) =L= p_yearlyLabH(t);

```

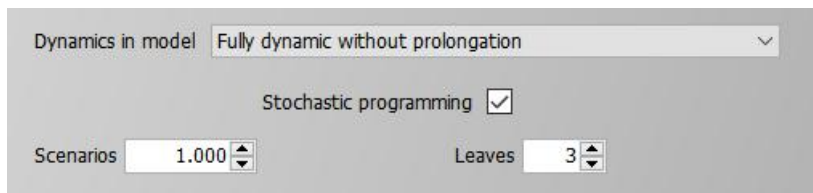
The revised objective function maximizes the probability weighted average of the final liquidity for each final leave in the decision tree; i.e. risk neutral behavior is currently assumed as in the deterministic variant:

```

*
*  OBJE_          ..
*
*      v_obje =e=
$iftheni.compStat not "%dynamics%"=="comparative-static"
*
*      --- accumulated liquidity
*      + household withdrawals
*      + revenues-costs of
*      liquidating farm in last year
*
*      [ sum(t_n("%lastYearCalc%",nCur), v_liquid("%lastYearCalc%",nCur)*p_probN(nCur))
*
*

```

The number of uncompressed scenarios to start with and the desired number of leaves in the final reduced tree used in the model are defined via the interface if the SP module is switched on:



Dynamics in model: Fully dynamic without prolongation

Stochastic programming: ☒

Scenarios: 1.000

Leaves: 3

That information enters the declarations in *"model\templ_decl.gms"*. If the stochastic programming extension is switched off, there is only one node (which is indicated by a blank " ") and the model collapses to a deterministic one:

```

$iftheni.sp not %stochProg%==true
*
*  --- dummy implementation of SP framework
*  there is one universal node, i.e. that is the deterministic version
*

set n "Decision nodes in tree" / " ",nonsense /;
set t_n(t,n) "Link between year and decision node";
t_n(t," ") = YES;

set anc(n,n) "Is the second node the node before first one?";
anc(" "," ") = YES;

set isNodeBefore(n,n) "Is the second node before first one?";
isNodeBefore(" "," ") = YES;

parameter p_probN(n);
p_probN(" ") = 1;

```

The changes in the listing are minimal compared to the previous version without the SP extension: there is a point more in each variable or equation name which indicates the blank common node, e.g.:

```

---- VAR v_objeTS  Gross margin each state of nature

          LOWER          LEVEL          UPPER          MARGINAL

2010..s1    -INF          153800.7949    +INF          .
2011..s1    -INF          149673.2763    +INF          .

```

With the SP extension, information is needed about ancestor nodes and nodes before the current one:

```

$evalglobal nt %lastYear%-%firstYear%+1

$evalGlobal nNode (%nt%-1) * %nOriScen% + 1
*
* --- sets and parameters are population in coeffgen\stochProg.gms
*
set n /n1*n%nNode%/;
set t_n(t,n) "Link between year and decision node";
set anc(n,n) "Is the second node the node before first one?";
set isNodeBefore(n,n) "Is the second node before first one?";
parameter p_probN(n);

$endif.sp

```

In the case,

Generating the random variable(s) and the decision tree

The generation of the decision tree and the related random variable(s) consists of three major steps:

1. **Generation of a predefined number of scenarios** which describe equally probably future developments for the random variables considered
2. **Generating of a reduced decision tree** from the these scenarios where most of the modes are dropped and the remaining nodes receive different probabilities
3. **Defining the symbols in GAMS** according 1. and 2.

As GAMS can become quite slow with complex loops, we implemented the first step in Java.

Currently, only one random variable is generated based on a logarithmic mean-reverting process (to avoid negative outcomes) with pre-defined parameters and an expected mean of unity. The starting price multiplier is also set to unity. The Java program is called from GAMS to pass the information on the number of decision nodes (= simulated time points) and the desired number of scenarios to the program:

```

execute "java -Djava.library.path=..\gui\jars -jar ..\gui\mrpfan.jar %nt% %nOriScen% %scrdir%\mrp.gdx 2>1"
set dummy / price /;

set tnum / t1*t%nt% /;
set tn(tnum,n);
execute_load "%scrdir%\mrp.gdx" p_randUar,isNodeBefore,tn,anc;

```

The Java process stores the generated random developments along with the ancestor matrix in a GDX. The following graphic shows a fan as generated by the Java program for five years and four scenarios. The command root node “1” is on the left, the following nodes 2, 5, 8, 11 and 14 are in the second year. Each second year node has its own set of followers, and all nodes besides 1 are equally probable with a probability of 20%.

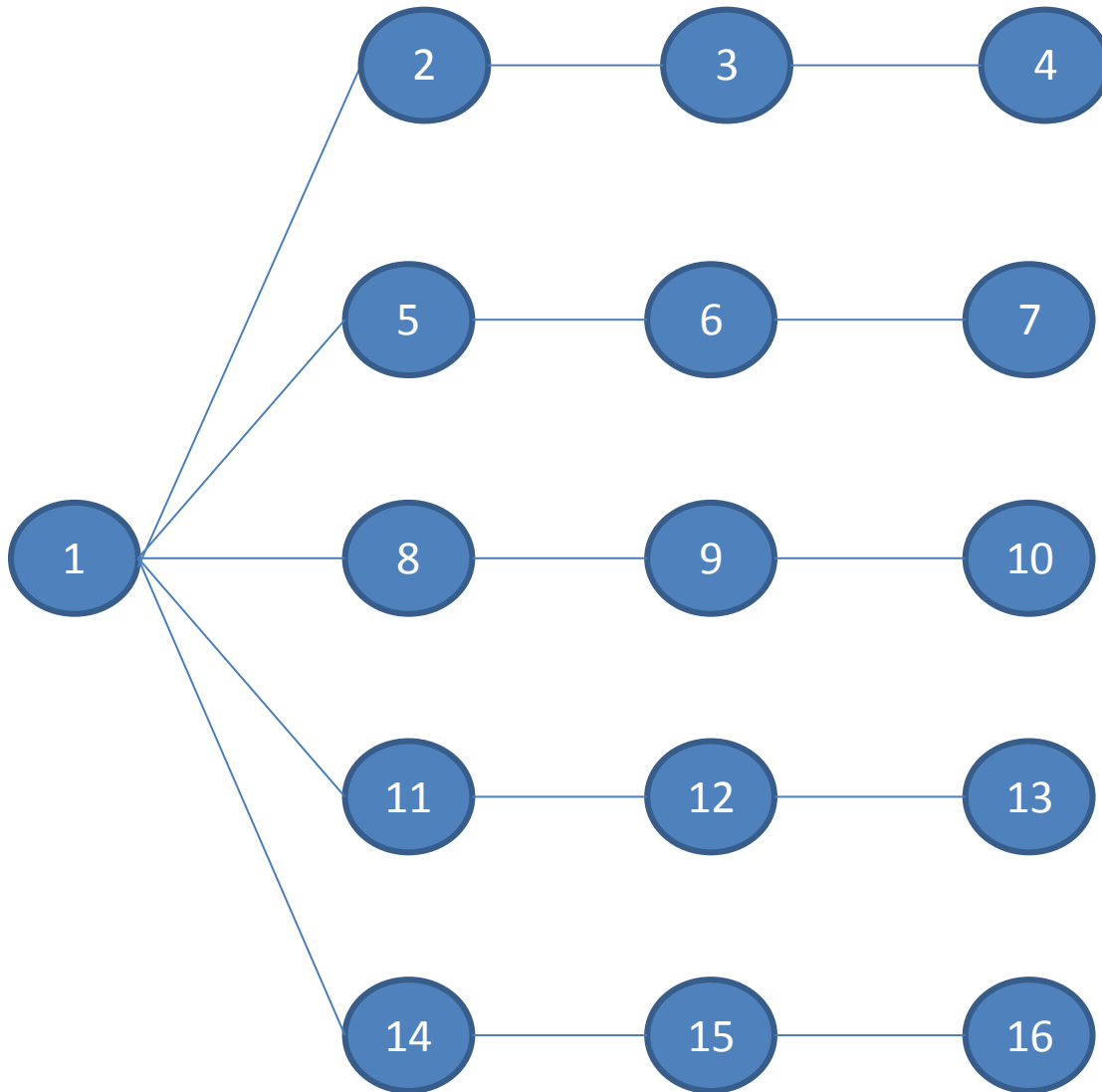


Figure 1. Example of an input tree organized as a fan

Next, the tree must be reduced to avoid the curse of dimensionality, achieved by using the SCENRED2 utility

(<https://www.gams.com/help/index.jsp?topic=%2Fgams.doc%2Ftools%2Fscenred2%2Findex.html>) comprised in GAMS (Heitsch and Römisich 2008, 2009). The algorithm deletes nodes from the tree and adds the probability of dropped node to a neighboring remaining ones.

The example in Figure 2 below depicts a resulting hypothetical tree with 4 final leaves generated from the tree in Figure 1 above. Each scenario starts with the same root for which the information is assumed to be known for certain, i.e. the probability for the root node N1 which falls in the first year is equal to unity. There are two nodes kept in the second year in the example, each depicting possible states of nature with their specific followers and potentially differing in their probabilities.

Node number 8 would carry a probability of 60% as it now represents 3 original nodes while node 11 captures the remaining 40%. The strategy chosen for each of these nodes depends simultaneously on the possible future development beyond that node while being conditioned on the decisions in the root node (which consequently depend on all follow up scenarios).

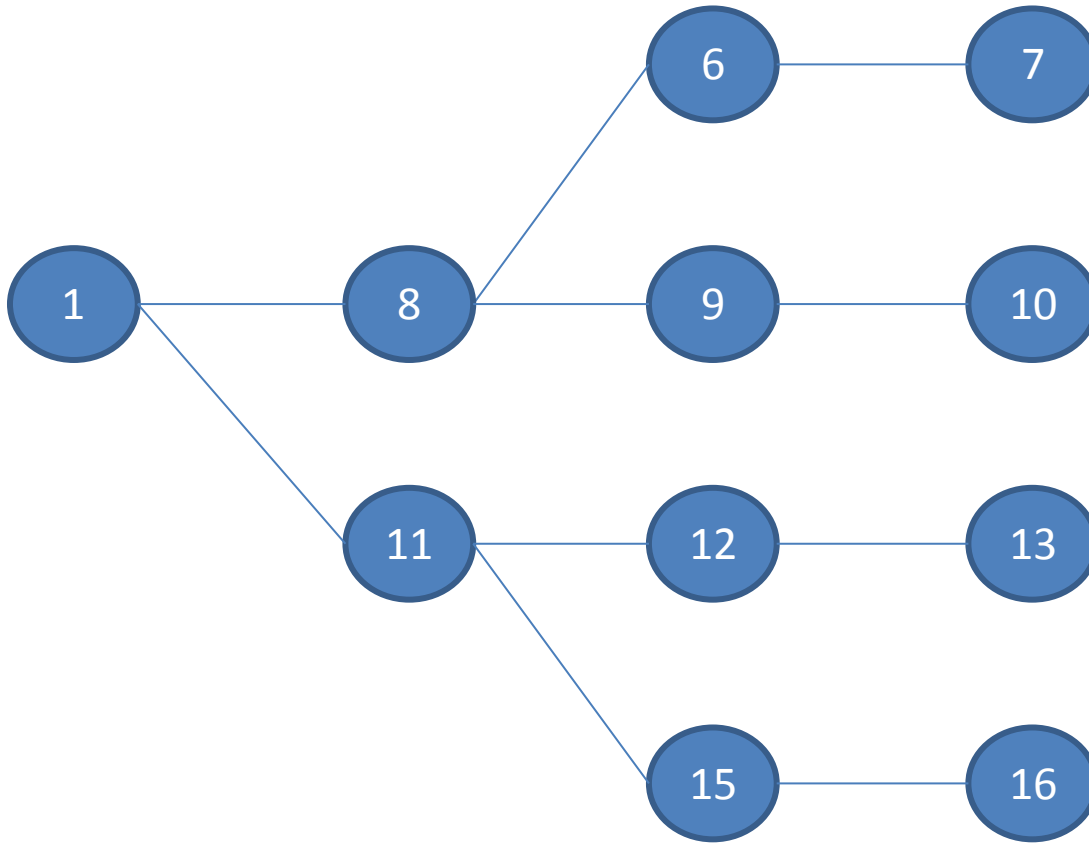


Figure 2. Example of an reduced tree

The example can also help to understand better some core symbols used in the code and relations in the SP extension. The nodes remaining in the reduced tree are stored in the set $nCur$. The set t_n would match the first year with the first node, the second year with the nodes 8 and 11 etc.. For the node 15, the ancestor set anc would be set to $anc("n15", "n11")$ to indicate that node 11 is the node before 15 on the scenario ending with leave 16. $Isbefore("n16", x)$ would be true for $x=16, 15, 11, 1$ and comprises the complete scenario ending with leave 16. The probabilities for nodes 8 and 11 must add up to unity as they relate to the same time point. The same holds for the node set (6,9,12,15) for the third year. It is also obvious that the decision at the root node 1 influences all scenarios, whereas the stage contingent decisions at node 8 influence directly the scenarios ending with the leaves 7 and 10. But it should be mentioned again that the root node reflect all scenarios simultaneously. As such, there is also an indirect influence between all nodes.

Furthermore, in a programming context no backward or forward recursion solution tactic is possible to find the best strategy as the number of strategies is normally not countable (the solution space is bounded, but there exist typically an infinite number of possible solutions). Finding a solution is further complicated by the fact that a larger number of variables has a binary character. That renders it specifically important to find an efficient way to reduce the number of nodes considered.

The scenario reduction algorithm adds the probability of similar, dropped nodes to the probability of an existing, remaining one and assigns a probability accordingly – think about a histogram where the individual observations in each bin are now summarized by the mean of the bin and the bin's probability mass. That allows reducing the number of stage contingent equations and variables to a size which can be handled numerically while still capturing the important moments of the distribution. In the current implementation, we steer the model size by setting exogenously the number of final leaves.

```

$setglobal sr2prefix test
$setglobal treeGen on

$iftheni.runSR2 %treeGen%=on
*
* --- scenario tree construction from fan
*
* $libinclude scenRed2

ScenredParms('sroption')      = 1;
ScenredParms('num_time_steps') = %nt%;
ScenredParms('num_nodes')      = card(n);
ScenredParms('num_random')     = 1;
ScenredParms('num_leaves')     = %nOriScen%;
ScenredParms('visual_red')     = 1;
$libinclude runScenRed2 %sr2Prefix% tree_con n anc p_probN ancRed p_probRed p_randUar

$endif.runSr2

;

execute_load 'sr2%sr2Prefix%_out.gdx' ancRed=red_ancestor,p_probRed=red_prob;

```

Based on the information returned from the scenario reduction utility, the set of active nodes $nCur$ is determined:

```

option kill=nCur;
nCur(n) $ p_probRed(n) = YES;

isNodeBefore(n,n1) $ ( (not nCur(n)) or (not nCur(n1))) = no;
tn(tnum,n) $ (not nCur(n)) = no;
display tn;

t_n(tCur,n) $ sum(tn(tnum,n) $ (tnum.pos eq tCur.pos),1) = YES;
t_n(tBefore,"n1") = YES;

anc(nCur,nCur1) = ancRed(nCur,nCur1);
anc("n1","n1") = YES;
p_probN(n) = p_probRed(n);

```

A little bit trickier is to find in an efficient the way *all* nodes which are before a given node in the same scenario (these are often nodes shares with other scenarios such as the root node). That is achieved by an implicit backward recursion over a year loop:

```

loop(tCur,
  loop(anc(nCur,nCur1),
    isNodeBefore(nCur,nCur2) $ isNodeBefore(nCur1,nCur2) = YES;
  );
);

```

The set *anc* (*nCur*,*nCur1*) indicates that decision node *nCur1* is the node before the node *nCur*, i.e. they belong to the same scenario. That is used in lag and lead operators, e.g.

```
*
* --- if the farm is there in t, it must have been there in t-1
*
  hasFarmOrder_(tCur(t),nCur) $ (tCur(t-1) $ t_n(t,nCur)) ..
      v_hasFarm(t,nCur) =L= sum(t_n(t-1,nCur1) $ anc(nCur,nCur1), v_hasFarm(t-1,nCur1));
..
```

The *isNodeBefore*(*nCur*,*nCur1*) relation depicts all nodes *nCur1* before node *nCur* in the same scenario, including the node *nCur* itself. An example gives:

```
*
* --- steady state: starting herds before the first fully simulated year are equal to that one
*
  v_herdStart(herds,breeds,tBefore,nCur1,m) =e= sum(t_n("%firstYear%",nCur) $ isNodeBefore(nCur,nCur1),
      v_herdStart(herds,breeds,"%firstYear%",nCur,m));
```

The following points are important to remember:

1. Even if the program scales the drawn price changes such that their mean is equal to unity, that cannot guarantee that the model would even without stage contingency perfectly replicated the deterministic version as the timing of the changes is also relevant (discounting, dynamic effects on liquidity etc.).
2. The normal case is that the objective value *increases* when considering stage contingency under risk neutrality. That is due to the effect that profits increase over-proportionally in output prices under profit maximization.
3. The solution time of the model can be expected to increase substantially with the SP extension switched on. MIP models are non-convex and NP-Complete problems. There exist no known polynomial-time algorithm which means that the solution time to optimality increases typically dramatically in the number of considered integers. Even small problems can take quite long to be solved even towards moderate optimality tolerances and not fully optimality. That holds especially if the “economic signal” to choose between one of the two branches of a binary variable is weak, i.e. if the underlying different strategy yield similar objective values. That is unfortunately exactly the case where the SP programming approach is most interesting (if there is one clearly dominating strategy rather independent e.g. of a reasonable range of output prices, considering different future inside that reasonable range is not necessary).

Introduction of the random variable(s)

Currently, the trial implementation allows only considering the milk price as a random variable. Please note here that the notion “random variable” only implies that the variable has an underlying probability distribution, not that it is a decision variable in our problem and hence a variable declared in GAMS. Indeed, that combination is impossible.

As mentioned above, the MRP process simulated in Java generates deviations around unity, i.e. we can multiply the selected mean price level from the interface with the simulated outcomes:

```
*
* --- revenue from sales of animal and crop products in average over states of nature
*      (SON specific price times SON specific production quantities)
*
*      salRev_(tCur(t),nCur,s) $ t_n(t,nCur)    ..
*
*      v_salRev(t,nCur,s)  =e= sum( curProds(prodsYearly),
*                                p_price(prodsYearly,t,s)
$iftheni.sp %stochProg%=true
* ( 1 + (p_randVar("price",nCur)-1) $ sameas(prodsYearly,"milk") )
$endif.sp
* v_saleQuant(prodsYearly,t,nCur,s));
```

Whereas it is technically straightforward to

Further extensions

It is relatively straightforward to extend the current set-up with a value at risk approach to consider risk aversion. Equally, the reporting part should be extended to report some major results for each state of nature.

References

Heitsch, H., and Römisch, W.. 2008. "Scenario Tree Reduction for Multistage Stochastic Programs." Computational Management Science 6 (2): 117–33. doi:10.1007/s10287-008-0087-y.

Heitsch, H., and Römisch, W.. 2009. „Scenario tree reduction for multistage stochastic programs". Computational Management Science, 6:117–133, 2009