Stochastic Programming (SP) with EMP

1 Introduction

This chapter describes the stochastic programming (SP) extension of GAMS. Based on a regular deterministic (core) model we can build a stochastic model by defining model parameters to be uncertain, so that GAMS replaces these by random variables. (Note that these are not variables in the sense of mathematical optimization - one could also talk about random parameters instead.) The distribution of those random variables is controlled by the user.

The next section demonstrates this using a simple example before Section 3.1 explains how to define random variables in detail. The Section 3.3 describes the reading of the solution.

2 Simple example

Here is a simple example were a newsboy wants to maximize his profit. He buys newspapers from a distributor (`X`) and sells it to customers (`S`). Newspapers which he does not sell are kept in his inventory (`I`). For every customer, that wants a newspaper but does not get one (`L`) a penalty has to be paid.

$$\begin{array}{ll} \max & vS - cX - hI - pL \\ \text{s.t.} & d = S + L \\ & I = X - S \\ & I, L, S, X \geq 0 \end{array}$$

In GAMS the model can be formulated like this:

```
Scalar  c       Purchase costs per unit                              / 30 /
        p       Penalty shortage cost per unit unsatisfied demand    /  5 /
        h       Holding cost per unit leftover                       / 10 /
        v       Revenue per unit sold                                / 60 /
*       Random parameters
        d       Demand                                               / 63 /;

Variable Z Profit;
Positive Variables
         X Units bought
         I Inventory
         L Lost sales
         S Units sold;

Equations Row1, Row2, Profit;

* demand = UnitsSold + LostSales
Row1.. d =e= S + L;

* Inventory = UnitsBought - UnitsSold
Row2.. I =e= X - S;
```

```
22
23 * Profit, to be maximized;
24 Profit..  Z =e= v*S - c*X - h*I - p*L;
25
26 Model nb / all /;
27
28 solve nb max z use lp;
```

Since there is no uncertainty in this model, the optimal solution is obvious: The newsboy buys as much newspapers as are demanded and sells them to the customers. The inventory is 0 and no demand stays unsatisfied. Now we want to make the demand (d) uncertain. The idea is that the newsboy buys newspaper at the beginning of the day without knowing the demand. Later, after his decision was done, the demand gets revealed, so that it becomes clear if there is any unsatisfied demand in case he did not buy enough or if he bought more newspapers than he can sell, so that the overhead needs to be stored in the inventory. In this case we are talking about a model with two stages: In stage 1 a decision has to be made without knowing the future. Then, at the beginning of stage 2, the outcome of the uncertain event is revealed so that other decisions in that stage can be made to react on the new situation. To define this model in GAMS we basically have to specify two additional things: The distribution of the random variable and the stages of all random variables, variables and equations. We do this by writing a text file %emp.info%:

```
file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
$onput
randvar d discrete 0.7 45 0.2 40 0.1 50
stage 2 I L S d
stage 2 Row1 Row2
$offput
putclose emp;
```

First, we define that parameter d becomes a random variable (randvar). It has a discrete distribution: with probability 0.7 it takes a value of 45, with probability 0.2 it takes a value of 40, and with probability 0.1 it takes a value of 50. The other lines in that file list all variables and equations in stage 2, the ones that are not listed here are assumed to be in stage 1. The objective variable and equation are automatically assigned to the highest stage mentioned (2 in this example). All keywords which can be used in this file are explained in Section 3.1.

The last thing that needs to be defined is where the results for each scenario should be stored:

```
Set scen        Scenarios / s1*s3 /;
Parameter
    s_d(scen)   Demand realization by scenario
    s_x(scen)   Units bought by scenario
    s_s(scen)   Units sold by scenario;

Set dict / scen .scenario.''
           d    .randvar .s_d
           s    .level   .s_s
           x    .level   .s_x /;

solve nb max z use emp scenario dict;
```

The size of the set scen defines the maximal number of scenarios we are willing to store results for. The three dimensional set dict contains mapping information between symbols in the model (in the first position) and symbols to store solution information (in the third position), and the type of storing (in the second position). An exception to this rule is the tuple with label scenario in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. In this example we want to store the realization for each scenario for the random variable d in the parameter s_d and the levels of the variables s and x in the parameters s_s and s_x respectively. A detailed description how this set works can be found in Section 3.3.

Finally, the solve statement needs to be extended by scenario dict to indicate that a stochastic problem should be solved.

It is worth noting that the size of the set `scen` does not have to match the number of scenarios actually generated in the solution process. In this example, if we set the size of `scen` to 2 by replacing the first line in the above code segment by

```
Set scen        Scenarios / s1*s2 /;
```

then the results of the first two scenarios will be stored in the parameters `s_d, s_x` and `s_s`, and the results of the third scenarios will not be stored. On the other hand, if the size of `scen` is bigger than the number of scenarios generated, then in the parameters (e.g. `s_d`) the positions of the excessive elements of `scen` will be empty.

In the case of the random variable coming from a parametric distribution, e.g. Normal, Poisson, etc., scenarios will be generated via a sampling procedure implemented in the solver. In this example, we can specify the random parameter `d` to be a normal random variable with mean 45 and standard deviation 10, by changing the `randvar` line in the emp.info file to:

```
randvar d normal 45 10
```

Currently, only the LINDO solver has implemented the sampling procedure for parametric distributions. The user could control the number of sampled scenarios by setting any of the following LINDO/SP options in the lindo.opt file:

```
STOC_NSAMPLE_PER_STAGE      - list of sample sizes per stage (starting at stage 2)
STOC_NSAMPLE_SPAR           - common sample size per stochastic parameter
STOC_NSAMPLE_STAGE          - common sample size per stage
```

For example, we could insert the following three lines before the `solve` statement:

```
option emp = lindo;
$echo STOC_NSAMPLE_STAGE = 100 > lindo.opt
nb.optfile = 1;
```

The first line tells GAMS to solve the `emp` modeltype using the LINDO solver, the second line writes the desired option to the lindo.opt file, which indicates the solver to generate 100 samples per stage, and the third line informs GAMS to use the solver option file (i.e. lindo.opt).

If parametric distributions are involved in the model but none of the above three options is set, then LINDO will generate 6 samples by default. Other than the number of samples, many sampling details are not currently customizable by the users.

3 Features and Usage

3.1 Modeling Uncertainty

The following keywords can be used in the emp.info file to describe the uncertainty of a problem:

chance: This defines individual or joint chance constraints (CC) using the following syntax:

```
chance equ {equ} [holds] minRatio [weight|varName]
```

This way one defines that a single constraint `equ` (individual CC) or a set of constraints (joint CC) does only have to hold for a certain ratio ($0 \leq$ `minRatio` ≤ 1) of the possible outcomes. The keyword `holds` is optional and does not affect the solver. If `weight` is defined, the violation of a CC gets penalized in the objective (`weight` $*$ `violationRatio`). Alternatively, the violation can be multiplied by an existing variable if this is defined by `varName`.

correlation: One can define a correlation between a pair of random variables like this:

```
correlation rv rv val
```

`Rv` is a random variable which needs to be specified using the `randvar` keyword and `val` defines the desired correlation ($-1 \leq$ `val` ≤ 1).

Distribution	Par 1	Par 2	Par 3
Beta	shape 1	shape 2	
Cauchy	location	scale	
Chi_Square	deg. of freedom		
Exponential	lambda		
F	deg. of freedom 1	deg. of freedom 2	
Gamma	shape	scale	
Gumbel	location	scale	
Laplace	mean	scale	
Logistic	location	scale	
LogNormal	mean	std dev	
Normal	mean	std dev	
Pareto	scale	shape	
StudentT	deg. of freedom		
Triangular	low	mid	high
Uniform	low	high	
Weibull	shape	scale	
Binomial	n	p	
Geometric	p		
Hyper_Geometric	total	good	trials
Logarithmic	p-factor		
Negative_Binomial	failures	p	
Poisson	lambda		

Table 29.1: Parametric distributions

jrandvar: `Jrandvar` can be used to define discrete random variables and their joint distribution:

```
jrandvar rv rv {rv} prob val val {val} {prob val val {val}}
```

At least two random variables `rv` are defined and the outcome of those is coupled. All random variables `rv` need to be in the same stage. The probability of the outcomes is defined by `prob` and the corresponding realization for each random variable by `val`.

randvar: This defines both discrete and parametric random variables:

```
randvar rv discrete prob val {prob val}
```

The distribution of discrete random variables is defined by pairs of the probability `prob` of an outcome and the corresponding realization `val`.

```
randvar rv distr par {par}
```

A list of all supported parametric distributions can be found in table 29.1. All possible values for `distr` and the related parameters `par` are listed there.

stage: Random variables (`rv`), equations (`equ`) and variables (`var`) are assigned to non-default stages like this:

```
stage stageNo rv | equ | var {rv | equ | var}
```

`StageNo` defines the stage number. The default stage for all random variables, equations and variables not mentioned with the `stage` keyword is 1, except for the objective variable and objective equation. The default for these is the highest stage mentioned.

3.2 Solver Configuration

At the moment three GAMS solvers can be used to solve SP models in the way described in this document: DE, DECIS and LINDO. Further information about these solvers can be found in the corresponding solver manuals.

	DE	DECIS	LINDO
chance	√		√
correlation			√
cVaR	√		
expectedValue	√		√
jrandVar	√	√	√
randVar (discrete)	√	√	√
randVar (parametric)			√

Table 29.2: Solver Capabilities

Certain keywords mentioned in Section 3.1 are not supported by all of these three solvers:

The SP options available for the DECIS and LINDO solvers are documented in the DECIS and LINDO/LINDOGlobal manual.

3.3 Output Extraction

After solving an SP model only the expected value of the solution can be accessed via the regular `.L` and `.M` fields. As described in Section 2, additional parameters have to be defined to store the results for the different scenarios solved. These are the things which can be stored by this approach:

`level`:	Stores the levels of a scenario solution of variable or equation
`marginal`:	Stores the marginals of a scenario solution of variable or equation
`randvar`:	Stores the realization of a random variable
`opt`:	Stores the probability of each scenario

In the example above we can use this:

```
Set scen        Scenarios / s1*s6 /;
Parameter
    s_x(scen)    Units bought by scenario
    srep(scen,*) Scenario probability     / #scen.prob 0/;

Set dict / scen .scenario.''
           x    .level   .s_x
           ''   .opt     .srep /;
```

The size of the set `scen` defines the number of scenarios we are willing to store results for. `X` is a variable for which we want to access the `level` and `s_x` is the parameter the `levels` of `x` are stored in. Note that `s_x` needs to have the same indices as `x` plus the additional index `scen` in the first position. In the parameter `srep` we store the probabilities of the different scenarios solved.

4 Case studies from the GAMS EMP library

This section demonstrates how the various EMP/SP features are being used in real-life models from the EMP library.

4.1 Long range forest planning – A multistage model (stocfor3)

This model features: how to write a multistage model in GAMS and how to specify the multistage info in the emp.info file. (Note: Only DE and LINDO but not DECIS support problems with more than two stages.)

While the main part of the model is self-explanatory, we will only examine the section that generates the emp.info file. The code segment is listed as follows.

```
1  file emp / '%emp.info%' /; put emp '* problem %gams.i%' /;
2  $onput
3  randvar f('t2') discrete .1736 .00000 .0299 .20268 .5128 .06258 .2837 .08612
4  randvar f('t3') discrete .1736 .00000 .0299 .20268 .5128 .06258 .2837 .08612
5  randvar f('t4') discrete .1736 .00000 .0299 .20268 .5128 .06258 .2837 .08612
6  randvar f('t5') discrete .6912 .00000 .3088 .20268
7  randvar f('t6') discrete .6912 .00000 .3088 .20268
8  randvar f('t7') discrete .6912 .00000 .3088 .20268
9  $offput
10 loop(t$(ord(t)>1),
11    put / 'stage ' ord(t):2:0 z(t) slack(t-1) defyield(t) defchglo(t-1) defchgup(t-1);
12    loop(k, put x(t,k) s(t,k) rf(t,k) defbnd(t,k) defbal(t-1,k)));
13 emp.pc=0; loop(t$(ord(t)>1), put / 'stage ' ord(t):2:0 ' f(' t.tl:0 ')');
14 putclose;
```

The segment between the pair $onput and $offput will appear in the emp.info file in exactly the same form as it does in the code. This segment declares and characterizes all the random parameters involved in the model one by one . Taking line 6 for example, it says that the parameter f('t5') is a discrete random variable which takes the value 0.00000 with probability 0.6912 and takes the value 0.20268 with probability 0.3088.

The loop from line 10 to line 13 enumerates for each stage (beginning from stage 2) the variables, equations and random parameters involved in the stage. For example, for the second stage, it starts with the keyword "stage 2", followed by a complete enumeration of the variables (e.g. z('t2') and slack('t1'), etc.) and equations (e.g. defyield('t2') and defchglo('t1'), etc.) involved in this this stage, with the items separated by space.

The generated file can be found in the 225a (or 225b, 225c, etc.) folder in the current project directory, in the file named "empinfo.dat", provided the model has been executed with the command-line option "keep = 1".

4.2 A simple model with chance constraints (simplechance)

This model features: how to write a model with chance constraints.

This model compares the effect of enforcing individual chance constraints and joint chance constraints. In particular,

```
chance E1 0.6
chance E2 0.6
```

specifies independently that the constraint E1 must hold with a probability of 60% and that the constraint E2 must also hold with a probability of 60%. Since there are altogether 12 scenarios and all have the same probability, the above statements require that each constraint must hold for at least $\lceil 12 * 0.6 \rceil = 8$ scenarios. After running the model, we can verify that this requirement has been enforced by comparing the constraints' level outputs, i.e. e1_l and e2_l, with their respective right-hand side values, i.e. 7 and 12.

By using the option "–JorI = J" in the command line, we can experiment with joint chance constraints by changing the specification to

```
chance E1 E2 0.6
```

This statement means that at least 60% of all scenarios must satisfy both E1 and E2 at the same time.

4.3 Production planning – A model with correlated random variables (prodsp3)

This model features: how to express correlations between random parameters in the emp.info file.

Instead of explaining the code, we directly look at the generated emp.info file for a clearer view of the EMP/SP syntax. We extract some representative lines, as follows.

```
1  randvar  h('work-1')  normal  6000 100
2  randvar  t('work-1','class-1')  uniform  3.50 4.50
```

```
3 correlation  h('work-1')   t('work-1','class-1') 1.00
4 stage 2 v lbal h t
```

The first line says that the parameter h('work-1') is a normal random variable with mean 6000 and standard deviation 100; the second line says that the parameter t('work-1','class-1') is uniformly distributed between 3.50 and 4.50; line 3 specifies the correlation (in this case 1.00) between the two random variables, and finally line 4 enumerates all the variables, equations and random parameters pertinent to stage 2. Note that, as demonstrated in this model, when all elements of a vector (i.e. variable, equation or random parameter) are pertinent to the same stage, it is not necessary to enumerate every element in the stage declaration statement. Listing the vector name will suffice (as shown in line 4). But this is not the general rule elsewhere. For example, even if all elements of the parameter h, i.e. h('work-1') and h('work-2'), followed the same distribution, e.g. Normal(6000,100), the statement

```
randvar h normal 6000 100
```

would not be acceptable. Instead, we need to enumerate for each element by writing

```
randvar  h('work-1')  normal  6000 100
randvar  h('work-2')  normal  6000 100
```