# Preliminary Design
# Stochastic Programming using Flowsheets

David L. Woodruff

March 7, 2017

## 1 Introduction

### 1.1 Background on PySP

The PySP package is part of Pyomo and it operates on *scenarios* that are, each separately, full instances of the deterministic problem (sometimes we use the word scenario to refer to the data for the instance). In general, the scenario data come in tree form, but for two stage problems this tree is pretty simple. Each scenario has an attached probability.

PySP also needs to know which Vars in the model correspond to which stage in the stochastic program. E.g., design Vars in the first stage and operating Vars in the second stage.

When the scenarios (and/or the data for them) and a tree are given to PySP it tries to find a solution, which means one value for each first stage Var (e.g., design variables) and values for subsequent stages (e.g., second stage operating variables) that can vary scenario by scenario. In most applications, users act only on the first stage solution and discard the second stage Var values.

### 1.2 Actors

This document gives instructions for two actors: the modeller and the scenario creator, but we begin by sketching the workflow for a "user" to try to help give context.

## 2 Sketch of User Work Flow

Some of these steps may be combined, or the user may “jump in” at some intermediate step.

1. Run some software to transform data about experiments or engineering judgment into objects useful for creating scenarios (e.g., distributions). In the first months or years, this might be done with a text editor rather than software.
2. Run software that creates scenario tree node files (e.g. `NodeMaker.py`)
3. Run software that uses daps (data to PySP) objects to create inputs for Pysp. This software might have a name like `TreeMaker.py`. (Aside: By “inputs to PySP” I mean a scenario tree model (or code for creating it) and scenarios.)
4. Put the inputs into a the current working directory or else a directory pointed to by the environment variable `PySP_DIR`.
5. Run the flowsheet model optimizer in Stochastic mode.
6. Analyze the outputs.

## 3 Modeller tasks

The modeller should refer to bfb_fs_model_opt.py for an example of code. Data files are in the subdirectoyr `stoch_dir`.

### 3.1 Function

The modeler needs to create a function with exactly this signature:
def pysp_instance_creation_callback(scenario_name, node_names):

This callback is used by PySP to instantiate scenario instances. The function we implemented as an quick-and-dirty example instantiates a flowsheet and then reads a json file to get new values for some of the Params. In the future, this code may make use of stoch_solver.PySP_dir to find these json files. The callback simply appends “.json” to the scenario name and loads the dictionary therein. The file names might be ugly, or not so ugly, but the modeller does not really need to know what the names are. The dictionary read from the scenario file has Param names as indexes and Param values as values. In the future, the values of the dictionary could be dictionaries where the index is the index into an indexed Param.

#### 3.1.1 Note concerning node data versus scenario data

The way the example function is currently written, it really operates on node data (root node data is not touched). An alternative would be to supply all data for an instance as the scenario. Then this fuction is trivial and just supplies data (or a file name) to the instance constructor. I have been told there is some json format for supplying all data to a flowsheet. That might be the way to go in the not-to-distant future, but maybe not before May.

#### 3.1.2 Note concerning Param

It is possible to use Python variables instead of Pyomo Params to receive (or construct) values for scenarios. This allows flexibility and perhaps some efficiency, but I think we should not do it for IDAES because we need to reduce flexibility so that multiple programmers and multiple pieces of code can operate on these scenarios and their input data and these folks and their code can figure out where things are. There are other "standards" beside Param that could be adopted, but Params are a natural for attachment to Pyomo models (and I think attaching these data to the model offers advantages).

### 3.2 File

The modeler must provide a tree template file that names the stages, assigns Vars to stages, and names the cost Expression (or cost Var) for each stage. The template file can be AMPL format or (soon to be supported) json.

The modeller assigns Vars to stages using this file; however, the modeller might be able to omit Vars that are in the last stage (depending on what meta-solvers will be used.)

### 3.3 Stochastic Solver

The flowsheet will construct and use the stochastic solver object using code that is probably in the main routine of the flowsheet.

The call to the constructor for this class will be in the flowsheet and needs to know the filename for the required function (which may be the flowsheet file itself, i.e. `__file__`). The constructor can also be passed the name of file with the scenario tree data (e.g., ScenarioStructure.dat) (or perhaps optionally a function to construct the tree.) Then a solve method can be called and it will return some kind of object with the solution (e.g., an ef_instance). The modeller's code can display or use the solution from this object.

### 3.4 Arcane notes

- Arcane note to Pyomo developers: a flowsheet is a process_base is a concrete_model (or block).
- There is limited support right now for a function to create the tree, that could be improved, but using a file might be a better idea for now.

## 4 Scenario Creator tasks

The scenario creator writes code that creates one json file for each scenario. The files contain a dictionary with Param names as keys and values for the scenario as values. Higher dimensional Params are not supported in the bfb_fs_model_opt.py as of Feb 18, 2017.

#### 4.0.1 Notes

Raw scenario node files have names like

NODE-X-PARENT-ROOT-PROB-Y.json

Where X is replaced with the scenario number and Y is replaced by the scenario probability. E.g,

NODE-12-PARENT-ROOT-PROB-0.3333333.json

It seems pedantic to create the raw scenario node files, but maybe we should do that for now; although I guess we could just pass through the raw scenario node objects. The main thing, I think, is to jump through the daps hoops so it will be pretty easy to go to multi-stage.

Code that uses daps code should take the scenario tree template and raw scenario node files and produce a full scenario tree file (by calling the daps code that does this). This code could use daps code to take the node input files and create scenario files with names like

SCENARIO-X.json

## 5 cryptic notes by DLW for DLW

"kinetic expression"

Multiple models as in ensembles

Each model has fit errors

User gives min, max or user gives mean, std dev

User specification of what is interesting

John: three steps - data - posit a model - regress against a model

I have couple models and I fit them, categorical uncertainty (which model) and parameter uncertainty within the models.