Fundamentals: The Need for Stochastic Programming

Overview of this session:

- Newsvendor Problem
- Scenario Generation for the Newsvendor Problem
- Two-Stage, Stochastic Program
- Newsvendor as a Two-Stage, Stochastic Program
- The "Underlying Process" vs. the "Decision Process"
- Multistage Stochastic Programming
- The Winvest Case
- Exercises

The Newsvendor and Winvest cases are used for the hands-on sessions.

References this section

[BL]: Birge and Louveaux: "Introduction to Stochastic Programming", Springer.

Newsvendor Problem

The *Newsvendor Problem* is a classical example of optimization under uncertainty. It used to be called the Newsboy Problem but now has a gender-neutral name!

A person sells newspapers in the streets. Each morning she purchases a number of papers from the distributor, at a price of c_p. Each day, there is a demand, y, for papers, which is unknown[1] in the morning. Throughout the day, she sells as many papers as she can at a price of c_s. Of course, she can't sell any more papers than she purchased in the morning, but if she sells fewer, the remaing ones can be sold as fish wrapping paper at a price of c_f. In the morning, she has money for at most N papers.

Her main decision is: *How many papers, x, should I purchase?* We assume that each day looks the same in terms of the *probability distribution* of demand for papers, so that the probability that i papers are demanded on any given day is p_i, $i = 0, 1, 2, ..., \infty$.

What is the optimal number of papers, $\hat{x}$, to buy in order to

Maximize Profits!

[1]Sometimes it's clearer to use a notation such as $\tilde{y}$ for stochastic variables but we'll just y.

Of course, there is no *fixed* number of papers that will maximize profits *every day.* Some days will be better than others. So she decides to

Maximize Expected Profits.

This is a problem she can formulate mathematically. First, she determines the connection between x, the number of papers purchased, and y, the number of papers demanded, and represents this as the stochastic profit function

$$F(x,y) = \begin{cases} c_s x - c_p x & \text{if} \quad x \le y \\ c_s y + c_f(x-y) - c_p x & \text{if} \quad x > y \end{cases}$$

With this setup-up, one possible formulation is as follows:

$$\begin{array}{ll} \underset{x \in \Re}{\text{Max}} & E_y F(x,y) \\ \text{S.t.} & 0 \le x \le N, \quad x \text{ integral} \end{array}$$

where E_y is the expections operator with respect to the stochastic variable y.

This is not an "old-fashioned" optimization problem: For instance, LPs don't allow an expectations operator, and $F(x,y)$ isn't linear. But we'll get to that!

Solution by straight calculation

The problem can easily be solved by a straigt-forward calculation: For each x on $0, ..., N$, calculate the expected profit

$$G(x) = E_y F(x,y) = \sum_{y=0}^{x} p_y F(x,y) \qquad (1)$$

and pick the value $\hat{x}$ for which G is maximized.
There are also more "elegant" methods. But we're interested in formulating the problem as an optimization problem! (GAMS code from NewsVend.gms)

```
********************************************************************************
*                                                                              *
*  SETTING UP THE PROBLEM'S DATA AND INITIAL CALCULATIONS:                     *
*                                                                              *
********************************************************************************

  SET support / 1 * 100 /;   ALIAS( support, i, j );

  PARAMETER p(i) "Probility that i papers will be demanded";
  PARAMETER lambda / 20.5 /;

* Calculate p(i) as a truncated Poisson-distribution with parameter lambda:
* p(i) = exp(-lambda) * lambda^i / x!   scaled to sum to 1.

  LOOP( i,                              # Runs through i = "1", "2", ...
    IF (ord(i) = 1,                     # Calculate the first one from scratch
      p("1") = exp(-lambda) * lambda;
    ELSE                                # Calculate based on the previous one
      p(i) = p(i-1) * lambda / ord(i);
    )
  );
  p(i) = p(i) / sum(j, p(j));    # Normalize so the p(i)'s sum to 1
  DISPLAY p;


 PARAMETERS
    c_s   "Price at which papers are sold"             / 35 /,
    c_p   "Price at which papers are purchased"        / 20 /,
    c_f   "Price for firh-wrapping (unsold) papers"    /  2 /,
    N     "Max number of papers that can be purchased" / 40 /;

  PARAMETER F(i,j) "Profit if i papers are purchased, and j are demanded";
            F(i,j) = c_s * min(ord(i), ord(j)) +
             c_f * max(0, ord(i)-ord(j)) - c_p * ord(i);
  DISPLAY F;
```

```
********************************************************************************
*                                                                              *
*  STRAIGHT-FORWARD CALCULATION OF OPTIMAL VALUE:                              *
*                                                                              *
********************************************************************************

  PARAMETER G(i) "Expected profit if i papers are purchased";

* Calculate G(i), leave the profit equal to zero for i's greater than N:
  G(i) $ (ord(i) <= N) = sum(j $ (ord(j) <= ord(j)), p(j) * F(i,j) );
  DISPLAY G;

* Find the highest possible expected profit, and the associated number (i_hat) to
* purchase:
  PARAMETER i_hat, max_prof;
  max_prof = SMAX(i, G(i));

  i_hat = SMIN(i $ (G(i) = max_prof), ord(i));
  DISPLAY i_hat, max_prof;
```

A Quick-and-Dirty Solution?

Could one solve the Newsvendor Problem quickly, by inspection?

- If we *knew* the day's demand in advance, it would (probably) be optimal to purchase precisely that number of papers in the morning. This is the "Wait-and-See" solution. But we need the "Here-and-Now" solution.
- If we only know the *expected demand*, then *that* number might be the optimal to purchase. This is the "Expected-Value Solution".

But these solutions are not *necessarily* optimal – consider, for instance, what happens if c_f is relatively large, compared to c_s. So our "quick-and-dirty" solutions may not be anywhere near optimal.

The Expected-Value Solution: This solution is obtained by replacing the stochastic parameter (demand) by its expectation, and solve the resulting, simpler program. In general, this solution can be *arbitrarily bad!* For the Newsvendor, consider what happens if the probabilities are as follows:

$$p_{10} = p_{50} = 0.5.$$

The Expected-Value solution might be to purchase 30 papers, but the optimal solution could be anywhere between 10 or 50 (depending upon data)!

We cannot use these quick-and-dirty solutions, such as the expected-value solution (which can be bad) or the wait-and-see (which we don't have until it's too late)[2].

We need something better: Stochastic Programming.

[2]But we can often get bounds among them. See, e.g., [BL] for a precise and extensive treatment.

Scenario Generation for the Newsvendor Problem

Scenarios

We notice first that the *set of possible values of the stochastic quantities*, i.e., the set of possible demands, is *discrete*, but *infinite*: any demand, even arbitrarily large, is possible, in principle.
This is a problem. We cannot directly handle a possibility set (event space) which has infinitely many elements in an optimization problem.

Constructing Scenario Sets

We need the set of scenarios under consideration to be *finite*, i.e., having only finitely many scenarios – and not too many! How can we get some useful *scenario sets*?

- **Truncation:** By assuming some upper bound on possible demands, so that higher demands will not be considered (in our case for instance, N), the set of demands becomes *discrete and finite*: {0, 1, 2, 3, ..., N }.
- **Aggregation:** We could have used a much smaller, aggregate scenario set:
  1. demand no more than 10,
  2. demand more than 10, but no more than 20,
  3. ...
  4. demand more than 50.
- **Random Sampling(MC):** We could have picked just, say 5, among the "real" scenarios, at random, and ended up with, for instance, {3, 7, 19, 33, 38}.
- **Uniform Sampling:** We could have picked evenly spaced values, for instance, {0, 10, 20, 30, 40}.
- **Importance Sampling:** We could have picked "important" scenarios: Some around the expected demand, and a few extreme ones: Ω = {0, 20, 21, 22, 23, 24, 25, 100}.

In general, whenever the stochastic variables involved have intinitely (or just very) many realizations (e.g., are continuous), we have to construct a suitable, finite scenario set.

We also have to consider which *probabilities* to associate with the chosen (or constructed) scenarios: There's a whole session on Scenario Selection later!

The Two-Stage, Stochastic Program

A decision is made at the present time facing future uncertainties. At a future time the uncertainties are resolved, and a recourse action is taken.

- Uncertainties are represented by *stochastic variables*,
- The values of the stochastic variables are (of course) unknown, but we assume that we know their *probability distributions.*
- *First-Stage Decision:* The decision to be made now.
- *Second-Stage Decision:* The decision to be made in the future, after we have observed a realization of the stochastic variables.

Notice that the second-stage decision depends upon both the first-stage decision, and on the realization of the stochastic parameters. The second-stage decision is also called "the recourse decision".

The problem can be formulated as follows:

[**SLP**]

$$\begin{aligned}
&\underset{x\in\Re^{n_1}}{\text{Minimize}} && c^T x + \mathcal{Q}(x) && (2)\\
&\text{Subject to} && Ax \;\;\, = b, && (3)\\
& && 0 \le x \le u. && (4)
\end{aligned}$$

where

$$\mathcal{Q}(x) = E\{Q(\mathbf{d}, \mathbf{h}, \mathbf{T}, \mathbf{v}, \mathbf{W} \mid x)\},$$

E is the expectation operator defined on some probability space $(\Omega, \mathcal{F}, P)$, and[3]

$$\begin{aligned}
Q(d,h,T,v,W \mid x) = \quad &\underset{y\in\Re^{n_2}}{\text{Minimize}} && d^T y && (5)\\
&\text{Subject to} && Wy = h - Tx, && (6)\\
& && 0 \le \text{y} \; \le v. && (7)
\end{aligned}$$

- In this formulation, bold letters indicate stochastic quantities.
- If the second minimization problem is infeasible, we take its value to be $+\infty$. We could also just assume that the problem has *relatively complete recourse*, i.e., problem (4) – (6) has a feasible solution for any value of the first-stage variable x which satisfies (2)–(3).
- In this formulation, n_1 and n_2 are the number of first-stage and second-stage decision variables, respectively.

[3]Possible confusion: In the formal model, y represents the second-stage variables, but in our Newsvendor example, y represents the demands!

- There are m_1 first-stage constraints (2), and m_2 second-stage constraints (5).
- Uncertainty of the second-stage problem is represented by the stochastic quantities $\mathbf{d}$, $\mathbf{h}$, $\mathbf{T}$, $\mathbf{v}$ and $\mathbf{W}$.

The Deterministic Equivalent Formulation

Consider the case where the stochastic quantities have a discrete and finite joint distribution, represented by the *scenario set* $\Omega = \{1, 2, 3, \ldots, S\}$.

In this case, we have

$$\mathcal{Q}(x) = \sum_{s=1}^{S} p_s Q(d_s, h_s, T_s, v_s, W_s \mid x), \tag{8}$$

where p_s is the probability of realization of scenario s,

$$p_s = P\{(\mathbf{d}, \mathbf{h}, \mathbf{T}, \mathbf{v}, \mathbf{W}) = (d_s, h_s, T_s, v_s, W_s)\}, \text{ for all } s \in \Omega. \tag{9}$$

It is assumed that $p_s > 0$ for all $s \in \Omega$ and that $\sum_{s=1}^{S} p_s = 1$.
Assuming that the uncertainties are represented by a finite scenario set, the stochastic program [SLP] can be reformulated (Wets [1974]) as the following *deterministic equivalent* program:

$$
\begin{array}{llllll}
[\mathbf{DELP}] & \underset{x \in \Re^{n_1}, y_s \in \Re^{n_2}}{\text{Minimize}} & c^T x + \sum_{s=1}^{S} p_s d_s^T y_s & & & (10) \\
& \text{Subject to} & Ax & = & b, & (11) \\
& & T_s x + W_s y_s & = & h_s, \text{ for all } s \in \Omega, & (12) \\
& & 0 \le x & \le & u, & (13) \\
& & 0 \le y_s & \le & v_s, \text{ for all } s \in \Omega. & (14)
\end{array}
$$

This large-scale, linear problem consists of $n_1 + S \cdot n_2$ variables and $m_1 + S \cdot m_2$ equality constraints.

The reason that we insist on having a finite set of scenarios is precisely to be able to transform the abstract formulation to the Deterministic Equivalent, which is *"just" an LP!*

Newsvendor as a Two-Stage Stochastic Program

The Newsvendor problem can be cast as a two-stage, stochastic program using:

1. x : The first-stage decision: How many papers to purchase,
2. z_s : The second-stage decision: How many papers to sell under scenario $s \in \Omega$.

In this example, the second-stage decision is very simple: Just sell whatever is demanded, but of course no more than we have purchased in the morning – but in general, this could be a complete optimization problem (for instance, if the price of fish wrapping paper, c_f, were also stochastic and higher than c_s under some scenarios...).

Mathematical Formulation

$$
\begin{array}{lrl}
\underset{x \in \Re, z_s \in \Re}{\text{Max}} & \sum_{s \in \Omega} p_s \cdot (c_s z_s + c_f(x - z_s)) - c_p x & \\
\text{S.t.} & x \leq N & \\
 & z_s \leq x, & \text{for all } s \in \Omega, \\
 & z_s \leq y_s, & \text{for all } s \in \Omega, \\
 & x, z_s \geq 0 & \text{and integral}
\end{array}
$$

where y_s is the demand under scenario s. This is a linear program (if we ignore the integrality constraints, which is OK in this case - why?).

File: NewsVend.gms.

```
*******************************************************************************
*                                                                             *
*  NEWSVENDOR PROBLEM AS A TWO-STAGE, STOCHASTIC PROGRAM.                     *
*                                                                             *
*******************************************************************************

* We now formulate the problem as a two-stage, linear stochastic program (SP).
* The first-stage decision is: How many papers to purchase, represented by x.
* The second-stage decision is: How many papers to sell, represented by z(s).
* The index s is the "scenario", that is, how many papers are demanded.
*
* Notice the two-stage decision process (two decision points):
* 1. Purchases in the morning, then wait and see how many are demanded/sold,
* 2. Decide how many papers to sell as fish wrapping, in this case a trivial decision.

* The scenario set (demands) is just the "support" set defined above, but we'd
* like to use the index "s" over scenarios:
  ALIAS( support, s );

  POSITIVE VARIABLES x      "How many papers to purchase",
                     z(s)   "How many papers are sold";

* Constraints on the first-stage variable(s):
  EQUATION max_prch "Max number to purchase";
  max_prch ..  x =L= N;

* Constraints on the second-stage variables:
  EQUATIONS Cnst2_a(s), Cnst2_b(s);

  Cnst2_a(s)  .. z(s) =L= x;         # Can't sell more than we have purchased
  Cnst2_b(s)  .. z(s) =L= ord(s);    # Under scenario s, ord(s) papers are demanded

* Calculate the objective:
  VARIABLES prof(s)     "Profit under each scenario",
            Exp_Prof    "Expected profit of the whole day";
  EQUATIONS Prof_Def(s), Exp_Prof_Def;

  Prof_Def(s)  .. prof(s) =E= c_s * z(s) + c_f * (x-z(s)) - c_p * x;
  Exp_Prof_Def .. exp_prof =E= sum(s, p(s) * prof(s) );

  MODEL TWO_STAGE /ALL/;
  SOLVE two_stage MAXIMIZING exp_prof USING LP;
```

The "Underlying Process" vs. the "Decision Process"

We now move towards "dynamic", or multi-period models. We have to make a decision at the beginning of the first period, and possibly at later periods.

Underlying Process(es): In many cases, we are modeling a world that evolves according to one or more *stochastic processes.* These processes are usually beyond our control, and we don't know their future values – although we may have statistics for them (expectations, (co)variances).

Examples:

- A stock price, $S_t, t \geq 0$,
- Short (1-year) interest rates in Germany in the future, $r_t^{GE}, t \geq 0$,
- Exchange rates EUR-USD,
- The yield curve in Germany, $y_t^u, 0 \leq u \leq 30, t \geq 0$, i.e., the yield, at time t in the future, of a u-year default-free Treasury bond,
- (r_t^1, r_t^{30}) might be a two-factor (two processes) interest-rate model, modeling the 1- and the 30-year rates.

All of these processes have two things in common:

1. They are *continuous-state:* At any point t in time, they have a continuous set of possible values (states);
2. They are *continuous-time:* They have a value at any point in time, t.

There are limited theorical models that can handle:

1. continuous-state processes: For example, the Black-Scholes formula for European options account for a continuum of future stock prices,
2. continuous-time processes: For example, stopping time formulas might tell us the best time to sell a stock.

but *in general, for stochastic programming*, we need *discrete time and space*: Decisions can only be handled at certain, pre-specified time points (epochs), and the set of states at each epoch must be finite (represented, e.g., by scenarios).

Decision Process(es): Assuming that we "have a handle" on the relevant underlying process(es), i.e. have relevant models and statistics for them, and that we have discretized them in time and space, we can then build a model that, for each epoch, determines an optimal decision - this sequence of decisions is called the *decision process.*

Ultra-brief summary:

Underlying processes $\rightarrow$ discrete-time/space scenarios $\rightarrow$ decision processes

Much more on scenario generation later on!

Multistage Stochastic Programming

The models seen so far have been *static*, consisting of just a single decision to be made at one point in time, or *two-stage stochastic programs*, which have two decision stages: first- and second-stage.

Real-life decision processes are more complicated:

> In addition to the initial decision we make now, there will be many opportunities to adjust down the road – many more points in time where decisions can be taken.

We need to model a multi-period setting, where decisions can be made at the beginning or end at each period, and where events during each time period are *uncertain*.

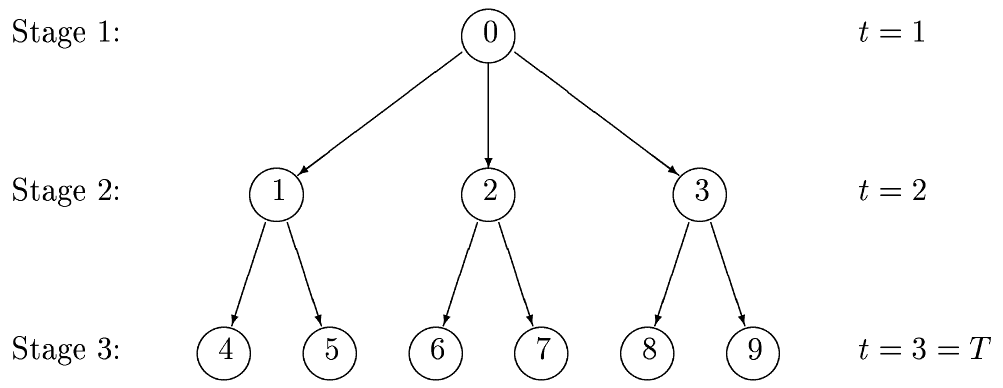
Multistage Stochastic Programming Models do this. They are *dynamic*, covering multiple time periods with associated, separate decisions, and they account for the *stochastic decisions process*.

How does it work? The main features of MS-SP are:

1. **Scenarios:** The uncertainty about future events are captured by a set of *scenarios*; a representative and comprehensive set of possible realizations of the future.
2. **Stages:** SP recognizes that future decisions happen in *stages:* A first-stage decision now. Then, after a certain time period, a second-stage decision, which depends upon (1) the first-stage decision, and (2) the events that occurred during the time period. Possible third-, fourth- etc. stage decisions.

Consider a model with 10 time period, and consider the decision to be made in period 5. When we reach period 5, we know *some* of the (today) unknown future (the first 4 periods), but not the rest of it! This is what makes multistage stochastic programming (MS-SP) complicated!

The Scenario Tree



Scenario tree for a 3-stage program ($T = 3$) having 6 scenarios. In this example, ξ_2 has three possible realizations. ξ_3 has two possible realizations for each realization of ξ_2.

The T-stage stochastic program:

$$[\mathbf{MS}] \quad \min_{x_1} \left\{ c_1 x_1 + \mathrm{E}_{\xi_2} \left[\min_{x_2} \left(\mathbf{c}_2 x_2 + \mathrm{E}_{\xi_3|\xi_2} \left(\min_{x_3} \mathbf{c}_3 x_3 + \cdots + \mathrm{E}_{\xi_T|\xi_2,\ldots,\xi_{T-1}} \min_{x_T} \mathbf{c}_T x_T \right) \right) \right] \right\}$$

$$\begin{array}{llcl}
\text{s.t.} & A_1 x_1 & = & b_1, \\
& \mathbf{B}_2 x_1 + \mathbf{A}_2 x_2 & = & \mathbf{b}_2, \\
& \qquad \mathbf{B}_3 x_2 + \mathbf{A}_3 x_3 & = & \mathbf{b}_3, \\
& \qquad\qquad \ddots & \vdots & \\
& \qquad\qquad\qquad \mathbf{B}_T x_{T-1} + \mathbf{A}_T x_T & = & \mathbf{b}_T, \\
& 0 \le x_t \le u_t, \quad \text{for } t = 1, \ldots, T, & &
\end{array}$$

where

$$\xi_t = (\mathbf{A}_t, \mathbf{B}_t, \mathbf{b}_t, \mathbf{c}_t) \;\; \text{for } t = 2, \ldots, T$$

are random variables, i.e., $\mathcal{F}_t$-measurable functions $\xi_\mathbf{t} : \Omega_t \mapsto \Re^{M_t}$ on some probability spaces $(\Omega_t, \mathcal{F}_t, P_t)$.

What does it mean? Where do the scenarios come from? Which structure should the tree have?

There's a whole session on these questions later in the course.

The Winvest Case

The rest of this session uses the Winvest case as the main example. The Winvest case (see Winvest.pdf) is a semi-realistic, 3-stage stochastic program. It introduces several important concepts:

1. Scenarios: The case consists of 4 interest rate scenarios.
2. Network Modeling: Each scenario is a network problem - very common in financial settings.
3. Stages: The model is a three-stage, stochastic program, with decisions to be made at times (epochs) 0, 1/2 year, 1 year.
4. Risk Attitudes: Different objective functions illustrate risk-neutrality, growth-optimality, extreme risk-aversion.

We focus in this session on points 1 and 3.

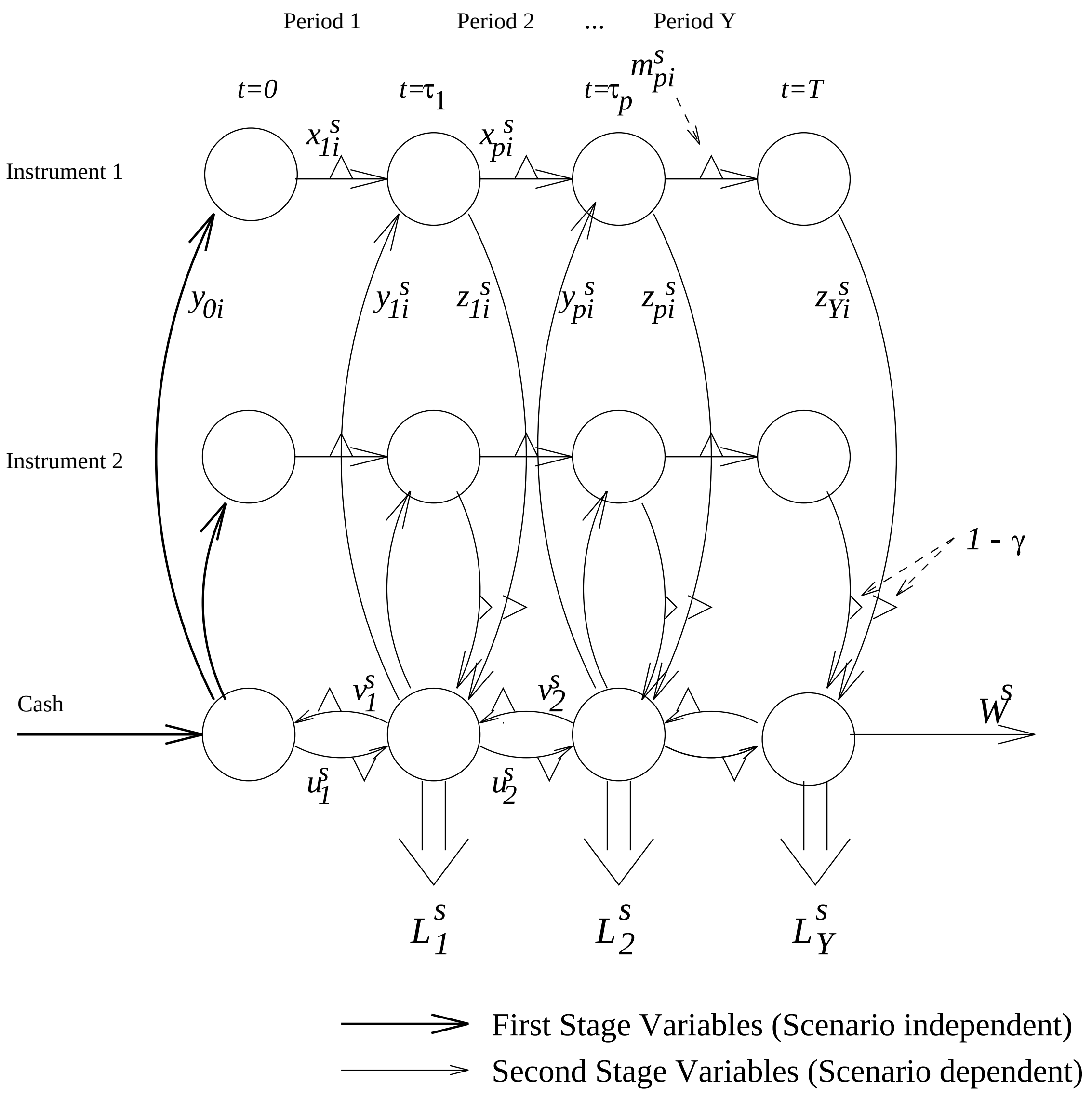
Winvest Data

There's an interesting story about MBSs, Mortgage-Backed Securities[4]

```
 set Cscen /uu, ud, dd, du/;                alias(s, Cscen);
 set Cmbs  /io2, po7, po70, io90/;          alias(i, Cmbs);
 set Ctime /t0, t1, t2/;                    alias(t, Ctime);

table yield(i,t,s)
                       UU         UD         DD         DU
      IO2 .T0    1.104439   1.104439   0.959238   0.959238
      IO2 .T1    1.110009   0.975907   0.935106   1.167817
      PO7 .T0    0.938159   0.938159   1.166825   1.166825
      PO7 .T1    0.933668   1.154590   1.156536   0.903233
      PO70.T0    0.924840   0.924840   1.167546   1.167546
      PO70.T1    0.891527   1.200802   1.141917   0.907837
      IO90.T0    1.107461   1.107461   0.908728   0.908728
      IO90.T1    1.105168   0.925925   0.877669   1.187143 ;

table cash_yield(t, s)

                       UU         UD         DD         DU
          T0     1.030414   1.030414   1.012735   1.012735
          T1     1.032623   1.014298   1.009788   1.030481 ;

 table liab(t,s)
                       UU         UD         DD         DU
          T1    26.474340  26.474340  10.953843  10.953843
          T2    31.264791  26.044541  10.757200  13.608207 ;

 parameter val(s)
   / uu = 47.284751, ud = 49.094838, dd = 86.111238, du = 83.290085/;

 parameter transcost; transcost = 0.01
```

[4]Nielsen and Zenios: A Stochastic Programming Model for funding Single Premium Deferred Annuities.

Network Scenarios



Network model underlying the multistage, stochastic network model. This figure includes two instruments, and depicts a 4-period model. That might correspond to a 4-stage model. Stochastic quantities are denoted by a superscript, s.

The Algebra Behind the Network

1. *Arcs* are decisions: How big is the flow?
2. *Nodes* are constraints: Flow in = Flow out.
3. *Multipliers* indicate gains/losses: One dollar invested (entering an arc) may be more or less when realized (exiting the arc).
4. *Scenarios:* There's only one set of first-stage variables, but second-stage (and 3rd, ...) variables for each scenario

The Winvest GAMS files are:

- WinIndep.gms: Solves independently for each scenario,
- WinSplit.gms: Solves the split-variable formulation,
- WinStoch.gms: Solves the 3-stage scenario-tree version (the "deterministic equivalent"),
- WinData.inc: A simple, 4-scenario data set,
- WinGen.inc: Generates data for an arbitrarily large 3-stage scenario tree.
- WinNodes.inc: Helper file to convert from split-variable to scenario tree representation.

Recommendation: Model a single scenario first. Then add (1) a scenario index to all data, variables, and constraints, and (2) *non-anticipativity constraints* to force first-stage variables to assume the same values across scenarios, etc.

Winvest: First Steps

The first version of the Winvest model is found in WinIndep.gms.

It solves for each scenario, independently (although simultaneously).

What is the optimal solution, if any? Are the results useful in any way?

WinSplit: Making it an SP

A few small modifications will make the model a true Stochastic Program:

1. Include a scenario index, s on all variables and constraints. Use it when referencing data instead of "UU".
2. We now have a final wealth variable for *each scenario*, $W(s)$. What do we optimize?

   One possibility: Define the expected final wealth,

   $$EW = \sum_{s \in S} p^s W(s)$$

   and maximize that. In Winvest, all $p^s = 1/4$.
3. Add constraints that force the first-stage decisions to be the same across scenarios (equivalently, one could remove the scenario index from the first-stage variables. This we will do later...).
4. The second-stage decisions must also agree between the UU and the UD scenario (because all we know at the second stage is that interest rates went up - we don't yet know what they do next). Similarly for the DU and DD scenarios. Add the appropriate non-anticipativity constraints. Now we have a true three-stage SP! (WinSplit.gms, WinData.inc).
5. The objective suggested above corresponds to a risk-neutral attitude. Would a real-life investor feel happy about the optimal solution? Or is the worst outcome just a little too bad for comfort?

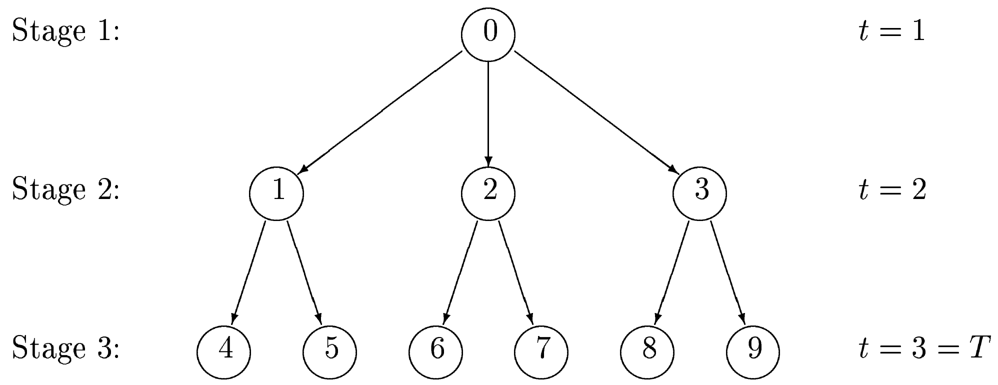
The Scenario-Tree Representation

This representation (WinStoch.gms) is quite different from the split-variable representation. Here, we associate a set of decision variables with *each node of the scenario tree*, while the split-variable representation uses a set of decision variables for each time point and each scenario.

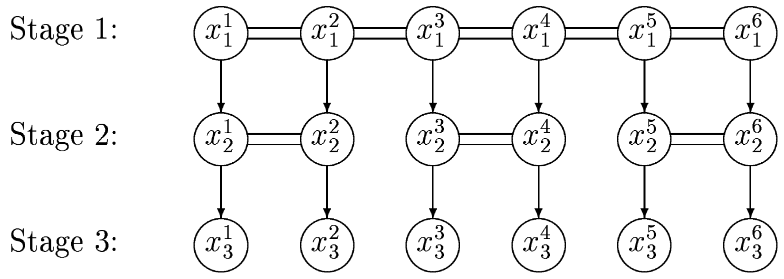
One can visualize the difference by looking at the following figures.

The Split-Variable Formulation

Original 3-stage scenario tree:



"Split" tree:


Stage 1:
Stage 2:
Stage 3:
x_1^1 x_1^2 x_1^3 x_1^4 x_1^5 x_1^6
x_2^1 x_2^2 x_2^3 x_2^4 x_2^5 x_2^6
x_3^1 x_3^2 x_3^3 x_3^4 x_3^5 x_3^6

- Make a copy of *all* variables for each scenario
- Add "non-anticipativity constraints" to force logically identical variables (all but last stage) to agree across scenarios.
- Winvest had non-ancitipativity constraints to enforce agreement among all $t = 0$ variables across all scenarios, and among $t = 6$ months variables between (UU, UD) and between (DU, DD) scenarios.

The Split-Variable Formulation

The deterministic equivalent 2-stage problem has the following split-variable formulation:

$$\begin{array}{lllll}
\underset{x^s\in\Re^{n_1},y^s\in\Re^{n_2}}{\text{Minimize}} & \sum_{s=1}^{S} p^s(f(x^s)+g^s(y^s)) & & & \\
\text{Subject to} & Ax^s & = & b & \text{for all } s\in <S> \\
 & C^sx^s+B^sy^s & = & r^s & \text{for all } s\in <S> \\
 & 0\le x^s & \le & u^s & \text{for all } s\in <S> \\
 & x^1 & = & x^s & \text{for all } s\in <S> \\
 & 0\le y^s & \le & v^s & \text{for all } s\in <S>
\end{array}$$

– compare to the original problem: –

$$\begin{array}{lllll}
\underset{x\in\Re^{n_1},y^s\in\Re^{n_2}}{\text{Minimize}} & f(x)+\sum_{s=1}^{S} p^sg^s(y^s) & & & \\
\text{Subject to} & Ax & = & b & \\
 & C^sx+B^sy^s & = & r^s & \text{for all } s\in <S> \\
 & 0\le x & \le & u^x & \\
 & 0\le y^s & \le & v^s & \text{for all } s\in <S>
\end{array}$$

Working with Risk-Attitudes

1. The worst outcome in the risk-neutral case is pretty bad. Implement an objective that *maximizes the final wealth under the worst outcome.*

   It's convenient to introduce a variable, Worst, whose value is equal to the worst of the 4 outcomes, then maximize it.

   Hint: This variable has a simple property:

   $$Worst \leq W(s); \text{ for all } s \in S$$

   What happens to the expected final wealth? Is the worst outcome better? What happens to the initial portfolio — for instance, diversification?

2. Finally, try a little utility theory. Find the portfolio that maximizes the expected utility, using $U(x) = \log x$.

   This is non-trivial if you haven't seen it before. First, you have to make sure that the arguments to the log functions are always positive, second, you now have an NLP (non-linear program) instead of and LP (linear program).

Newsvendor Exercises

1. Using the data $c_p = 20, c_s = 35, c_f = 2, N = 25$ and $p_i = \frac{-\lambda\lambda^x}{x!}$, scaled so that $\sum_{p_i} = 1$, with $\lambda = 20.5$, find $\hat{x}$.

2. In the above question, why is it necessary to scale the p_i?

3. Using the probability distribution

$$p_{10} = p_{50} = 0.5$$

   with the data above, find the optimal solution.

4. Do you expect the optimal solution to always be 10 or 50, with the above data? Then try:

   With the above probability distribution and data, change c_f to 25. What happens to the optimal solution?

Winvest Exercises

1. Run the WinIndep, WinSplit and WinStoch models on the WinData.inc data set and compare the objective values and solutions.

2. Run WinSplit and WinStoch using the WinGen.inc data generator. Create a tree that has 200 scenarios, and has 10 branches out of the root node! (i.e., fanout = 10, fanout2 = 20).

3. Copy WinStoch.gms to WinWorst.gms. Now, modify the model to find a portfolio which *maximizes the worst-case outcome*. You should use the WinData.inc data for this.

4. Copy WinStoch.gms to WinUtil.gms. Now, modify the model to find a portfolio which *maximizes the expected utility of outcomes*. The utility function should be the logarithm ("log" in GAMS). You should use the WinData.inc data for this.

GAMS Advanced Summer School Heidelberg, Germany, September 1–3, 2003

Scenario Generation Techniques – Overview of this session:

- Basic Ideas and Principles
- Moments
- The main example: the CRR process
- Multistage Scenario Generation by Sampling
- Improved Sampling Techniques
- Scenario Generation from Historical Data
- Multistage scenario generation - advanced material.

The Basic Problem

We have argued that the general Stochastic Programming (SP) problem needs *scenarios* to be implementable. But the situations, or the environment, in which we are modeling, often consists of some economic processes which may be

- specified through some formal economic/financial model, e.g. stochastic processes, which could be continuous in time and/or space,
- based on historical observations.

Examle 1: We wish to construct a portfolio consisting of stocks, $S_i, i \in \mathcal{U}$, from some *universe* of possible stock investments. We have observations of the current price of each stock, S_i^0. In addition, we have a *formal model*[1] for the evolution of each stoch's price:

$$dS_{i,t}/S_{i,t} = \mu_i dt + \sigma_i dz_i, t \geq 0,$$

which states that the *relative change* in stoch i's price has a drift of μ_i and a volatility σ_i, and where the dz_i are (correlated) Wiener processes.
How do we get from these continuous-time, continuous-state processes to a suitable scenario set? How do we get, or estimate, μ_i and σ_i?

Examle 2: We wish to construct a portfolio consisting of stocks, $S_i, i \in \mathcal{U}$, from some *universe*, $\mathcal{U}$, of possible stock investments. We have observations of the current price of each stock, S_i^0, and in addition, *historical observations* of the stock prices $S_{i,t}$ in the past. For instance, $t = 0, ..., T$ might indicate observations for (prices) the past T months.
How do we get from these historical observations to a suitable scenario set?

We will show principles for scenario generation based on both formal models and historical observations (and combinations of these!).

[1]This is the well-known log-normal Cox-Ross-Rubinstein (CRR) stock price model

Scenario Generation: Basic Ideas and Principles

We generally require the scenario set used in a SP to be[2]

Comprehensive: It should capture all aspects, both extreme and "normal" instances, of the underlying distributions or historical observations,

Consistent: It must capture *trends* and *volatilities* of the underlying distributions or historical observations.

Satisfying the *comprehension* requirement is often a question of including "enough" scenarios. For instance, when sampling scenarios randomly, we will quickly get a lot of "normal" (close to "average") scenarios, but it may take many scenarios (or non-random sampling, such as importance sampling) to get a good representation of extreme events.

Satisfying the *consistency* requirement is basically a matter of making sure that the *expectations* and *volatilities/correlations* of the scenario set is the same as those of the underlying processes', or as those exhibited in any historical observations.

Another way to state this is that the "first two *moments*", that is, the expectations (or trends), and volatilities (equivalently, variances and covariances/correlations) of the underlying data (processes/observations) and of the generated scenarios much match. For distributions which are non-symmetric or *skewed*, it might be desirable to match the third moment as well (examples: log-normal, exponential).

[2]See Nielsen: fin_notes.pdf, Section 2.4 for more discussion

Moments: Definitions and Uses

Let X be a stochastic, real variable. If X is discrete, i.e., assumes at most a countable number of values, we define the *expectation* of X as:

$$\mu = E\{X\} = \sum_{k=1}^{\infty} x_k p_k$$

where $p_k = P\{X = x_k\}$ is the probability that X equals x_k. If X is continuous (assumes a continuum of values), we define

$$\mu = E\{X\} = \int_{-\infty}^{\infty} x f(x) dx$$

where $f(x)$ is the *probability distribution function*, pdf, for X[3]

Now define the *k'th moment (or moment of order k) around* c, for some real number c, as:

$$\mu^k = E\{(X - c)^k\}.$$

In particular, for $c = 0$ we obtain the *k'th moment (around the origin)*:

$$m_k = E\{X^k\}$$

and for $c = \mu = E\{X\}$ we obtain the *k'th moment around the mean* as:

$$m_k = E\{(X - \mu)^k\}$$

m_2 is called the *variance* of X and is denoted σ^2, or $Var\{X\}$. Its square root, σ, is called the *standard deviation* of X, or (especially in continuous-time finance) the *volatility* of X, and denoted $SD\{X\}$.

We finally define the *covariance* of two stochastic variables, X and Y, as

$$cov\{X, Y\} = \sigma_{X,Y} = E\{(X - E\{X\})(Y - E\{Y\}\},$$

and their *correlation* as

$$corr\{X, Y\} = \rho_{X,Y} = cov\{X, Y\}/(SD\{X\}SD\{Y\}).$$

The quantities m_k, μ_k, σ etc. that summarize stochastic variables (and their correlations) are called *statistics.*

[3]For these expectations to be defined, we require that $\sum_{k=1}^{\infty} \mid x_k \mid p_k < \infty$, or $\int_{-\infty}^{\infty} \mid x \mid f(x)dx < \infty$, respectively. The remaining quantities defined on this page have similar requirements for existence, see, e.g., Rohatgi: "An Introduction to Probability Theory and Mathematical Statistics", Wiley..

Example: The CRR Stock Price Process

Consider a stochastic process, such as the stock price process $S_{i,t}$ defined by

$$dS_{i,t}/S_{i,t} = \mu_i dt + \sigma_i dz_i, \quad t \geq 0. \tag{1}$$

The term on the left-hand side, $dS_{i,t}/S_{i,t}$ is the relative change in the stock price over an infitesimal (”extremely short”) time interval, dt. The price has an expected, annualized change of μ_i and an annualized volatility of σ_i.
We can simulate this process in discrete time, that is, generate ”observations” that are Δt apart (e.g., $\Delta t = 1/12$ for one-month intervals) by rewriting[4]

$$\Delta S_{i,t}/S_{i,t} = \mu_i \Delta t + \sigma_i \epsilon_t \sqrt{\Delta t}, \quad t = 1, 2, ... \tag{2}$$

Given the price $S_{i,t}$, we draw a number ϵ_t from the standard normal distribution (NORMAL(0,1) in GAMS) and calculate the price change, $\Delta S_{i,t}$ and hence the next ”observation”, $\Delta S_{i,t+1} = S_{i,t} + \Delta S_{i,t}$. A complete sequence of such generated prices from $t = 0, 1, 2, ..., T$ constitutes a scenario, or a *Monte-Carlo Sample* of the stock price process.

```
* For simplicity, we model only a single stock, i.e., drop the i index.
* In addition, however, we get a scenario index, l (the letter!).

  SET time /0 * 36/ ;    ALIAS(time, t);
  SET scenario / 1*20 /; ALIAS(scenario, l);

  PARAMETER
    S_0    "Initial price"      /  100 /,
    mu     "Drift, annualized" / 0.15 /,     # 15% expected, annual return
    sigma  "Volatility"        / 0.25 /,
    Delta_t "Time intervals"   / 0.833333 /, # Monthly
    epsilon(l,t) "Process increments",
    S(l,t)  "Price in scenario l at time t";

* Do the simulation:
  epsilon(l,t) = NORMAL(0,1);     # All increments are standard normal

  S(l,"0") = S_0;               # Start all simulations off at S_0
  LOOP(t $ (ord(t) > 1),        # Do the remaining ones (t > 0)
    S(l, t) = S(l, t-1) +
              mu * Delta_t + sigma * epsilon(l,t) * sqrt(Delta_t);
  );
```

[4]Using the fundamental relationship $dz = \epsilon\sqrt{dt}$, or, in discrete time, $\Delta z = \epsilon\sqrt{\Delta t}$, which is a statement of the change, Δz, of the Wiener process dz over a (short) time interval, Δt.

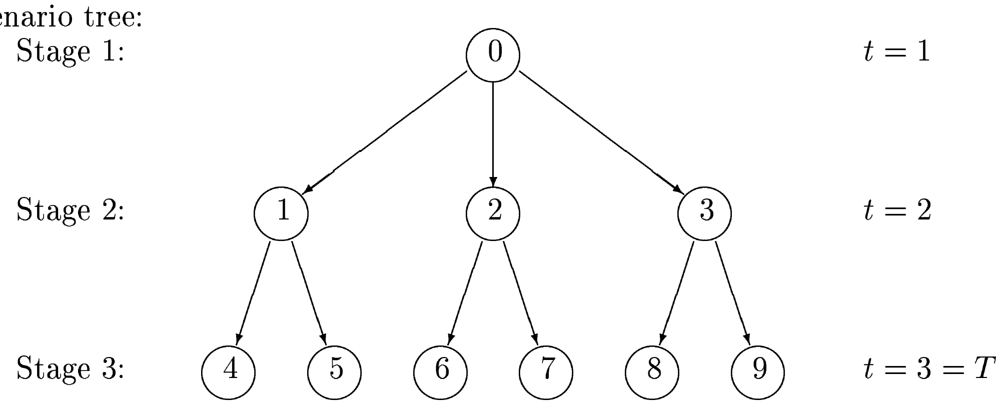
Parameter Estimation

Where do the process parameters, μ_i and σ_i, come from? Several possibilities:

- μ_i according to the *efficient market hypothesis* equal r_f, the risk-free rate, for all i! That is, every stock has the same expected return as the risk-free investment (a bank deposit at a safe bank, or a Treasury Zero-Coupon Bond) because any differing expected return can be hedged away (using options). This is a risk-neutral (or perhaps, risk-ignorant) world view.
- μ_i and σ_i could be based on historical observations: Simply the average annualized return over the observation period. (But why would a stock which had a high return over the past few years continue to have that?)
- σ_i could be set to the *implied volatility*: Observe prices for *Europaean options* on the stock, then solve the Black-Scholes option pricing formula for the volatility that satisfies it.
- Many interest rate models of the short rate (e.g., Vasicek and Cox-Ross-Rubinstein) allows closed formulas for the pricing of default-free zero-coupon bonds (ZCBs), such as Treasury notes. To assure *no-arbitrage*, the interest rate parameters should be chosen so that observable prices of ZCBs are consistent with the model's predictions. This is somewhat similar to using implied volatilities for stock price models.
- Other approaches could be based on various Portfolio Performance Measurements, e.g., Sharpe Ratios. See, e.g., Bodie, Kane, Marcur: *Investments*, 1989, Ch. 24.

Generating Multi-stage Scenarios

How can we generate scenarios suitable for a multistage SP? Consider a typical (small) scenario tree:



Each scenario "runs" from the root to a leaf node. Scenarios that go through common nodes must be identical up to (the time associated with) that node. For instance, the first two scenarios must be identical between times $t = 1$ and $t = 2$, but can of course differ after $t = 2$.

This assures that there is exactly one optimal solution at, e.g., node 2, even for path-dependent instruments (because any paths leading *to* node 2 are identical up to that point).
Doing this in GAMS requires some non-trivial data manipulations (as we saw in WinStoch.gms and WinGen.inc example) but in principle the scenario generation process is straight-forward.

Example: Monte-Carlo Sampling the CRR Process

Now, conceptually it is straight-forward to generate a random scenario tree for the CRR process. Starting from its initial value $S_{i,0}$, draw 3 samples $\epsilon_{i,1}$ from the standard normal distribution. Using (2) for each of these 3 values, we get 3 new values for $\Delta S_{i,t}$, and hence for $S_{i,t} = S_{i,t-1} + \Delta S_{i,t}$.

These values correspond to the nodes at $t = 2$ in the tree above. For each node, generate 2 normal random samples, apply (2) for each, and we have values for the leaves.

We now have a 3-stage tree like that shown above.

Improved Sampling Techniques

A Monte-Carlo simulation is simple and fast but has a huge drawback: It may take a huge number of scenarios generated this way to achieve *consistency* and *comprehensiveness*:
It may be difficult to generate "extreme" scenarios (comprehensiveness), and it may be difficult to generate enough scenarios that their averages and volatilities are close to the prescribed values (consistency). We mention a few ways to alleviate these problems:

Antithetic sampling: When generating random deviates, ϵ_t, only generate every other one (the odd ones) – use their negative values as the even ones. So if we generate the values $0.23, -2.18, 0.41$, we really have 6 samples: 0.23, -0.23, -2.18, 2.18, 0.41, -0.41.

Antithetic sampling guarantees that the odd moments (expectation, skewness) are precisely hit, hence helps consistency.

Moment Matching (or Quadratic Resampling): Let m and s be the sample mean and standard deviation, calculated after sampling $\epsilon_i, i = 1, ..., N$. By using $y_i = \frac{\epsilon_i - m}{s}$ instead of ϵ_i in all calculations, the first two moments will be matched. This is useful for one-factor models only.

Moment Matching by NLP: For distributions of dimension 2 or higher (such as multinomial distributions), or for distributions other than multinomial ones, it may be impossible to solve for moment matching in closed form. One can then solve the non-linear system of equations associated with each node of the scenario tree (or lattice) by numerical methods (Hoyland and Wallace).

Importance Sampling: Sample the "important" scenarios with a higher probability than the "unimportant" ones. "Importance" has a well-defined meaning: Scenarios are important if they have a high probability, or if they have a high "impact" on the optimal solution. The precise definition depends on the problem domain. See, e.g., Nielsen: importance.pdf on your CD, and references therein (see figures following slides). Can be designed for multi-factor models.

Quasi-Random Sequences: The numbers generated by computer random number routines are known as "pseudo-random" ("false-random") numbers: They are not really random but behave statistically as if they were. Quasi-random ("almost random") numbers are designed to "fill out" the sample space uniformly, whereas random or pseudo-random numbers will often exhibit clusters of close samples. See paper by Joy, Boyle, Tan in Management Science (1996?). Can be designed for multi-factor models.

Scenario Reduction: It is generally easy to generate a huge number of scenarios using for instance Monte-Carlo simulation, but the resulting SP then also becomes huge and, perhaps, impossible to solve. *Scenario Reduction* attempts to reduce the number of scenarios present in the program without affecting its precision (i.e., the optimal solution). There's a whole session on this later!

More Ad-Hoc Techniques:

Expert Opinion: Very often the user (expert) of the model has a very definitive outlook of the future – this could (and should) be used to generate user-relevant, though subjective, scenarios.

Regulations: There may be regulatory requirements to scenario sets used, for instance, by financial companies to show *solvency* to regulators and/or shareholders. An example is the "NY-7 scenario set": 7 interest-rate scenarios under each of which financial institutions in New York are required to show that they would remain solvent (fin_notes.pdf, 2.4).

End-of-Horizon Effects

The future extends forever, but our model has a finite *horizon* — The time corresponding to the leaves of the tree.

It is important to consider End-of-Horizon Effects, i.e., making sure that the results of the model will allow "the world", or at least the decision maker, to continue past the horizon.

Example: In a multistage optimization of water reservoir usage in Texas, the reservoirs tented to be completely depleted at the end of the planning horizon. in every solution to the optimization model! The model didn't "know" that there was a world after its own horizon!

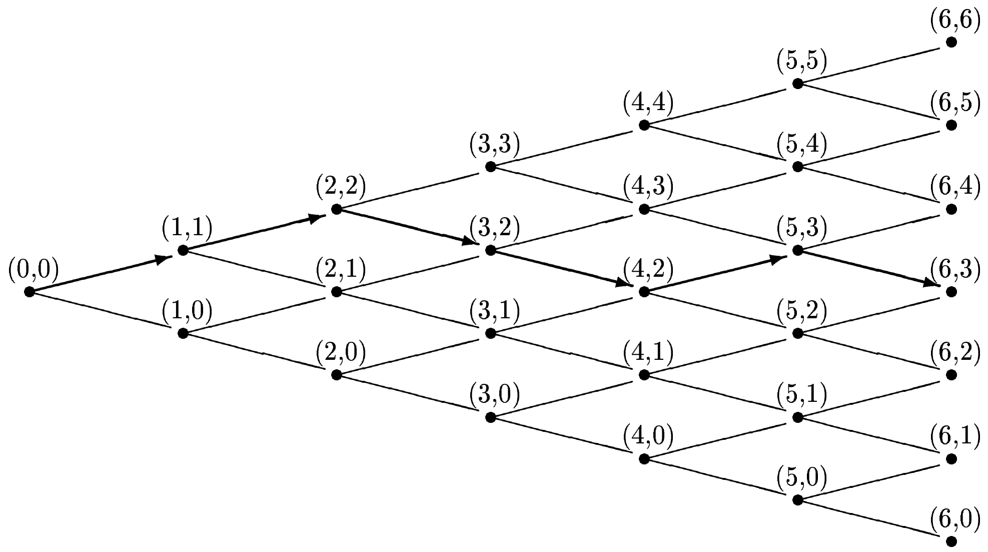
In financial applications it will be necessary to impose horizon constraints on solution portfolios to make sure there is a viable future after the model's solutions.

Example: Importance Sampling the CRR Process

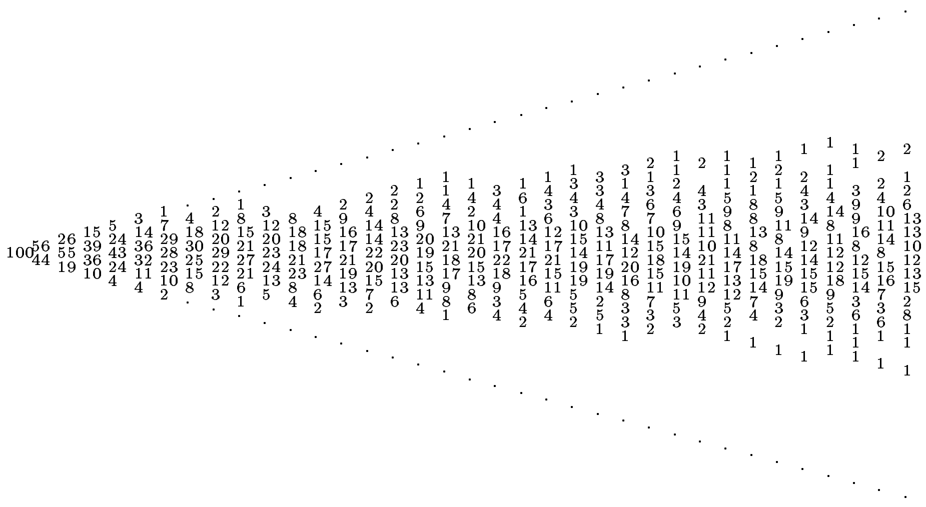
Since the CRR process is a one-factor (one-dimensional) process, it is actually possible to generate, without Monte-Carlo sampling, a binomial (recombining) tree which automatically has the correct first two moments, expectation and variance/volatility. The precise formulas for doing this are well-known, see, e.g., Hull.

But with many time periods (a fine time-discretization), the number of scenarios could still be extremely large. One possibility around this, then, is to *importance sample* the tree.

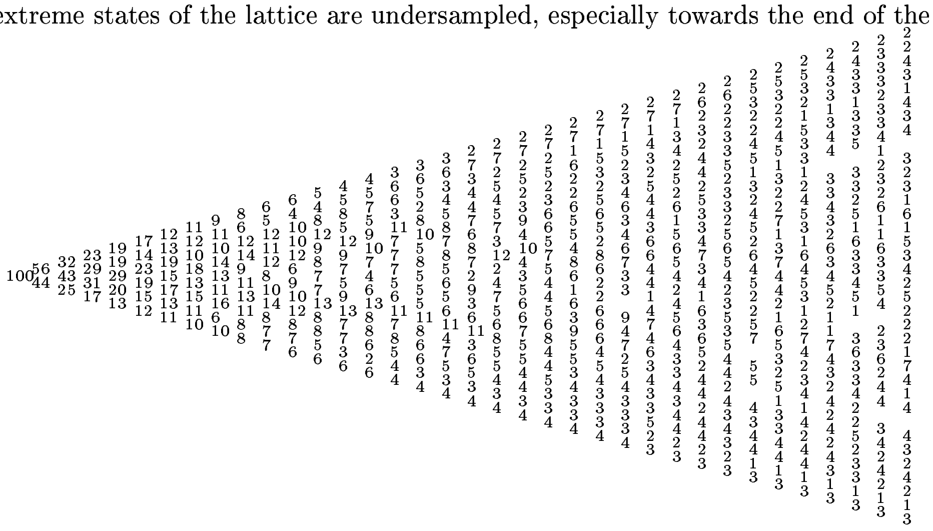
For details, see the paper importance.pdf by Nielsen.



Tree with $N = 6$ time periods. Coordinates (i, j) and the sample path $\omega = (1, 1, 0, 0, 1, 0)$ are shown. Here shown in its recombining, or lattice, version.



100 uniformly sampled paths through a lattice with $N = 36$ time steps. At each grid node is indicated the number of samples passing through the node. There is a strong centralizing tendency, and the extreme states of the lattice are undersampled, especially towards the end of the time horizon.



100 paths sampled according to the distribution $q_{ij} = \frac{j+1}{i+2}$, where q_{ij} is the probability of branching up at node (i, j).

Scenario Generation from Historical Data

Assume that we have available historical observations of stock prices, $S_{i,t}$ of stocks $i \in \mathcal{U}$ over a time period, indexed by $t = 0, ..., T$, where the observations are Δt apart. There are now two possibilities for using these data:

1. We can use them to estimate parameters for a formal (underlying) process such as the CRR model: Define annualized returns:

$$r_{i,t} = \frac{S_{i,t}}{S_{i,t-1}\Delta t}, \quad t = 1, ..., T$$

(or, if we use continuous compounding:

$$r_{i,t} = \frac{\log(S_{i,t}/S_{i,t-1})}{\Delta t}, \quad t = 1, ..., T),$$

and the observed means and variances:

$$\hat{\mu_i} = \frac{1}{T}\sum_{t=1}^{T} r_{i,t},$$

$$\hat{\sigma_i} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_{i,t} - \hat{\mu_i})^2}.$$

These estimates can now be used directly in the CRR sceneration generation.

If we are modeling correlated processes (correlated stock prices in the CRR model, for instance), we need estimates of the correlations $\rho_{i,j}$ between stocks i and j:

$$\hat{\rho_{i,j}} = \frac{\hat{\sigma_{i,j}}}{\hat{\sigma_i}\hat{\sigma_j}} \text{ where } \hat{\sigma_{i,j}} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_{i,t} - \hat{\mu_i})(r_{j,t} - \hat{\mu_j})}.$$

Hull (Ch. 15) now gives the following procedure[5]: For each time step t, standard normal deviates $x_i, i = 1, ..., |\mathcal{U}\Updownarrow\rangle\lceil$, are sampled. Then we use as "noise" in the correlated processes the values

$$\epsilon_i = \sum_{k=1}^{i} \alpha_{i,k} x_k$$

where the α_i's must satisfy:

$$\sum_{k=1}^{i} \alpha_{i,k}^2 = 1, \quad \forall i, \text{ and } \sum_{k=1}^{i} \alpha_{i,k}\alpha_{k,j} = \hat{\rho_{i,j}}, \text{ for } j = 1, ..., i-1.$$

Set ϵ_1 equal to x_1. One can then calculate ϵ_2, ϵ_3 etc. one at a time (including the $\alpha_{i,j}$'s). $\alpha_{i,j} = 0$ for $i < j$.

[5]The $\alpha_{i,j}$s are actually the Cholesky factorization of the correlation matrix. For more details, consult "Numerical Recipies in C" or Nielsen: cholesky.pdf on your CD.

2. Bootstrapping: To generate N scenarios, divide the return data $r_{i,t}$ up into N equally sized chunks along the t dimension, and use each chunk as a scenario. This procedure is very simple and preserves any statistical structure present in the data, such as correlations between instruments (i, j pairs).

   For instance, the CD contains historical returns (IntlAssets.inc) for a number of stock (and other) indices, over 120 monthly time periods (10 years). We could generate, for instance, 10 time series covering one year each from this.

   Alternative, for single-period models we immediately have 120 scenarios, each covering only one time period, but fully correlated across instruments. Such data are used by the MAD.gms and other models on the CD (directory Models).

As always when using historical data we have to assume that "history repeats itself" for these procedures to make sense.

Exercises

1. Copy the CRR.gms model to CRRlog.gms and modify it to simulate the log of the stock price instead of the stock price itself. Do you get a better agreement between the average and expected paths?

2. Write a GAMS model which generates scenarios for the short interest rate using one of the following mean-reverting processes:

   $$\text{Vasicek: } dr = a(b - r)dt + \sigma dz$$

   or

   $$\text{Cox, Ingersol, Ross: } dr = a(b - r)dt + \sigma\sqrt{(}r)dz.$$

   Use your own reasonable values for the mean reversion level b and the "deviation from the mean"-parameter a.

Algorithms for SPs: Overview

Overview of this session:

- Straight-forward solution
- Specialized SP-solvers
- Algorithms for Two-stage Programs:
  - ”Fixing” the first-stage decision
  - The L-shaped (Benders) Method Decomposition
  - Variations on Benders Decomposition
  - Progressive Hedging and other algorithms
- Algorithms for Multistage Programs
- Exercises

References this section

[BL]: Birge and Louveaux: ”Introduction to Stochastic Programming”, Springer, 1997, Chapters 5-7.
[DHS]: Dupačová, Hurd, Štěpán: ”Stochastic Modeling in Economics and Finance”, Kluwer, 2002, Section II.8.

Straight-forward Solution

By this we mean to formulate the SP (in its deterministic equivalent form) directly in, e.g., GAMS, then solve it directly.

Modern LP-solvers, such as CPLEX, are becoming very sophisticated when it comes to detecting special structure in the LP. It is worthwhile to try a direct solve before using specialized algorithms.

Example: Large-Scale Multistage SP in GAMS

Nielsen and Poulsen solved a large-scale, multistage problem directly in GAMS, using CPLEX:

- 9 stages, 19,683 scenarios
- 29524 nodes in the scenario tree
- Deterministic Equivalent contained 1.3M variables and 1.3M constraints
- Solved in 5.9 hours on a 800 MHz Windows laptop

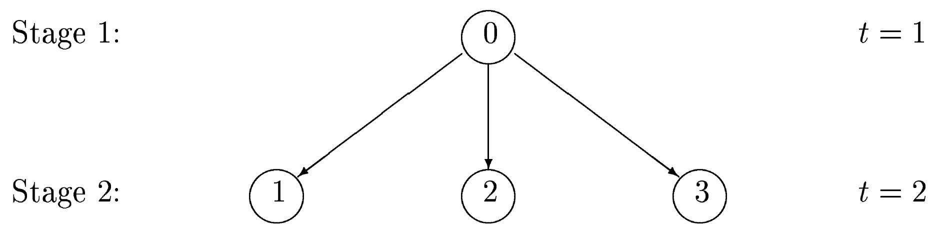
The paper is on the CD (Mortgage.pdf) as well as a version of the GAMS model (DanMort.gms, documented in DanMort.pdf).

Specialized SP Solvers

There are also special SP solvers available, which can be used directly from GAMS. They allow a formulation of the deterministic equivalent in, for instance GAMS, but still use specialized SP algorithms for the solution.

An example is OSL/SE which implements the L-shaped (or Benders) algorithm, and is available with GAMS. Another one is DECIS.

Algorithms for Two-Stage SPs



[DELP]

$$\begin{array}{lllll} \underset{x\in\Re^{n_1},y_s\in\Re^{n_2}}{\text{Minimize}} & c^Tx+\sum_{s=1}^{S}p_sd_s^Ty_s & & & (1)\\ \text{Subject to} & Ax & = & b, & (2)\\ & T_sx+W_sy_s & = & h_s,\ \text{ for all } s\in\Omega, & (3)\\ & 0\le x & \le & u, & (4)\\ & 0\le y_s & \le & v_s,\ \text{ for all } s\in\Omega. & (5) \end{array}$$

The algorithms we mention here are all based on the idea of *decomposition*: We would like to "split up" the problem by scenario. This is because it is far easier, computationally, to solve S scenario problems separately than 1 large program containing all the scenarios.

Key Idea: "Fixing" the first-stage decision

Consider "fixing" the first-stage decision x to some fixed value, $\hat{x}$. Then the problem "decomposes" by scenario and is (presumably) easy to solve. This could be done sequentially, scenario subproblem by scenario subproblem, or in parallel on a parallel computer.

But how should $\hat{x}$ be chosen?

L-Shaped (Benders) Decomposition

The "L-shaped" algorithm (Van Slyke and Wets 1986) is a specialization of Benders Decomposition to two-stage SPs.

Informal Overview:

1. Fix a candidate first-stage decision, $\hat{x}$,
2. Solve the individual scenario subproblems
3. Use dual information from the solution to generate a *cut*: This is information about where potentially better proposals for $\hat{x}$ might be found
4. Solve the "Master Problem" using the collection of cuts generated so far. The optimal solution is the next $\hat{x}$. If not converged, go to Step 2.

The precise algorithm is given next. For more details, see [BL] or [DHS].

A GAMS implementation of Benders algorithm is given in Bend.gms on the CD (this is not an SP).

Variations on Benders Decomposition

Many improvements of the basic algorithm have been devised, such as *multicut*, *bunching*, etc.

A particularly important one is Regularized Decomposition (Ruszczynski 1986): This address in particular two problems with Benders:

1. Initial iterates may "jump" wildly around, not yielding much progress
2. Slow tail convergence

Regularized Decomposition puts a penalty term into the objective,

$$\frac{1}{2} \mid\mid x - a^{\mu} \mid\mid^2$$

where a^{μ} is the previous iterate (or one of the previous iterates, or a good starting point). The effect is that the iterates tend to stay "close" to each other, which improves convergence. Also, this allows the use of a starting point (for instance, in an operational setting we may have a good idea of what the optimal solution is).

However, it is a non-linear (quadratic) program and cannot be solved by, e.g., CPLEX. Details: [BL].

Step 1: (Initialization.) $\nu \leftarrow 1$. Solve the initial master problem

$$\begin{array}{lll} \underset{x \in \Re^{n_1}}{\text{Minimize}} & c^T x & \\ \text{Subject to} & Ax = b, & \\ & 0 \le x \le u. & \end{array} \tag{6}$$

Let x^ν be an optimal solution, and let $\underline{z}^\nu = c^T x^\nu$.

Step 2: (Subproblem.) For each $s \in \Omega$ solve the subproblem:

$$\begin{array}{llll} \underset{y_s \in \Re^{n_2}}{\text{Minimize}} & d_s^T y_s & & \\ \text{Subject to} & W_s y_s & = & h_s - T_s x^\nu, \\ & 0 \le y_s & \le & v_s. \end{array} \tag{7}$$

Let (y_s^ν, π_s^ν) be the optimal primal and dual solution, where π_s^ν are the dual prices associated with (17). Let $\bar{z}^\nu = c^T x^\nu + \sum_{s \in \Omega} p_s d_s^T y_s^\nu$.

Step 3: (Test for convergence.) Let $\underline{z} = \min_{\ell=1,\ldots,\nu} \underline{z}^\nu$, and $\bar{z} = \max_{\ell=1,\ldots,\nu} \bar{z}^\nu$, which are lower and upper bounds on the optimal objective value, respectively. If they are equal, then terminate with (x^ν, y_s^ν) as the optimal solution. Otherwise, set $\nu \leftarrow \nu + 1$ and proceed from Step 4.

Step 4: (Master problem.) Solve the problem

$$\begin{array}{lll} \underset{x \in \Re^{n_1}, z \in \Re}{\text{Minimize}} & c^T x + z & \\ \text{Subject to} & Ax = b, & \\ & z \ge \sum_{s \in \Omega} p_s \left(-\pi_s^\ell (T_s x + W_s y_s^\ell - h_s) \right), & \ell = 1, \ldots, \nu - 1 \\ & 0 \le x \le u. & \end{array} \tag{8}$$

Let (x^ν, z^ν) be an optimal solution, let $\underline{z}^\nu = c^T x^\nu + z^\nu$ and proceed from Step 2.

Figure 1: Benders decomposition algorithm, specialized to the two-stage, stochastic program.

Progressive Hedging

The Progressive Hedging algorithm (Rockafellar and Wets, 1991) is an adaptation of Rockafellar's proximal point method.
It uses the "split-variable" formulation of the two-stage SP. It solves each scenario subproblem including copies of the first-stage decision variables, x.
Then it calculates a weighted average (weights e.g. probabilities) of the resulting first-stage decisions, and uses this average as a penalty "anchor" in the objective.

Details: [BL] or [DHS]. Mulvey and Vladimirou, 1991, successfully used PH on generalized networks.

Other Algorithms

Many other algorithms and variations exist (Monte Carlo-methods, stochastic gradient, row-action, approximations, ...) but they do not seem to be used in practice. See [BL].

Algorithms for Multistage SPs

$$\begin{tikzcd}
\text{Stage 1:} & & & & & & \textcircled{0} \arrow[dllll] \arrow[d] \arrow[drrrr] & & & & & & t=1 \\
\text{Stage 2:} & & \textcircled{1} \arrow[dl] \arrow[dr] & & & & \textcircled{2} \arrow[dl] \arrow[dr] & & & & \textcircled{3} \arrow[dl] \arrow[dr] & & t=2 \\
\text{Stage 3:} & \textcircled{4} & & \textcircled{5} & & \textcircled{6} & & \textcircled{7} & & \textcircled{8} & & \textcircled{9} & t=3=T
\end{tikzcd}$$

Multistage SP (MSSP) algorithms are generally extensions of two-stage algorithms.

Nested Benders Decomposition

View the MSSP as a two-stage SP where the subproblems themselves happen to be SPs.
Then use Benders Decomposition on the "top two-stage SP". Solve the subproblems by any method – for instance, Benders Decomposition again. This is "Nested Benders", or the "Nested L-Shaped Algorithm".

See [BL, 7.1] or [DHS, 8.4] for many more details.

Other algorithms

The Progressive Hedging algorithm also has variants for MSSPs - see [DHS, 8.4].

Dualizing Non-anticipativity Constraints

“Split” tree:

$$\begin{tikzcd}
\text{Stage 1:} & x_1^1 \arrow[r, equal] \arrow[d] & x_1^2 \arrow[r, equal] \arrow[d] & x_1^3 \arrow[r, equal] \arrow[d] & x_1^4 \arrow[r, equal] \arrow[d] & x_1^5 \arrow[r, equal] \arrow[d] & x_1^6 \arrow[d] \\
\text{Stage 2:} & x_2^1 \arrow[r, equal] \arrow[d] & x_2^2 \arrow[d] & x_2^3 \arrow[r, equal] \arrow[d] & x_2^4 \arrow[d] & x_2^5 \arrow[r, equal] \arrow[d] & x_2^6 \arrow[d] \\
\text{Stage 3:} & x_3^1 & x_3^2 & x_3^3 & x_3^4 & x_3^5 & x_3^6
\end{tikzcd}$$

Yet another idea is to ”relax” (temporarily ignore) the non-anticipativity constraints in the split-variable formulation, then solve each complete scenario separaterly (sequentially, or in parallel).

Then the NA-constraints can be enforced iteratively in the objective (penalty, Lagrangian, ...).

Exercises

The file Bend.gms on the CD contains an implementation of Benders Decomposition. It is not a stochastic program, though. Instead, it contains 2 sets of variables, which we can interpret as "first-stage", and "second-stage" variables for a single scenario.

1. Write down the problem being solved on a piece of paper. Which variables are "first-stage", which ones are "second-stage"?[1]
2. Solve the model. How many Benders iterations does it take?
3. Write down the sequence of candidate "first-stage" solutions encountered along the way. What is the optimal solution?

[1] In Large-Scale optimization literature, the "first-stage" variables are often called "complicating" variables.