UNIVERSITÄT BIELEFELD
Faculty of Business Administration and Economics

DECISION ANALYTICS

Combining OR and Data Science

Summer Term 2022

5. Optimization under Uncertainty I: Stochastic Linear Programming

J-Prof. Dr. Michael Römer, Till Porrmann
Decision Analytics Group | Bielefeld University

Before we start

Homework

- Homework was uploaded, due date is Tuesday, June 7. Any questions?

Guest Talk

- **Wednesday, June 1st, 16:00 h**, we will have a **guest talk** by **Prof. Warren B Powell (Princeton University)**:
- Title: **Introduction to Sequential Decision Analytics: A unified framework for decisions under uncertainty**

More Information on the **Bigsem-Website**

Overview: Introduction to Optiization Under Uncertainty: Stochastic Linear Programming

In this meeting, we will

- start transferring the ideas discussed in the last weeks to the world of (mixed integer) linear programming
- have an introduction to linear programming under uncertainty (stochastic programming)
- see how we can implement these ideas in Python

We have two main parts:

- **Part 1**: Linear Programming in Python
- **Part 2**: An Introduction to Stochastic Linear Programming

Part 1: Linear Programming in Python

In this part, we will

- see how we can formulate our case study example as linear program
- learn how to implement and solve a linear program in Python using Python-MIP

Unconstrained Optimization vs (Mixed-Integer) Linear Programming

(Continous) Unconstrained Optimization (e.g. scipy.optimize)

- very flexible
- takes arbitrary functions
- only continuous variables
- constraint handling not very efficient (if possible at all)
- works well with Monte Carlo approximations
- heuristic solution approaches
- solution not necessarily optimal

(Mixed Integer) Linear Programming (e.g. Gurobi, CBC)

- restricted to linear expressions
- modeling power due to integer / binary variables
- constraints are handled efficiently
- Monte-Carlo approximation can be "embedded" in model
- exact solution approaches
- optimality proof

Motivation: Capacity Planning Case Study

In the previous weeks, we formulated the **deterministic** version of the capacity planning problem as follows:
maximize total profit = - capacity installation cost + sales margin

$$\max f(x, d) = -30x + 40 \min(x, d)$$

where
x: installed capacity in units (decision) d: demand in units (given parameter)
Let us formulate this as a linear programming problem!

Formulation as Linear Program

We modeled the (deterministic) problem as follows:

$$\begin{aligned} \max \ & f(x,d) = -30x + 40min(x,d) \\ \text{s.t. } & x \geq 0 \end{aligned}$$

- given that this involves the $\min$ operator, f is a nonlinear function

However, as you might know, we can easily reformulate the $\min$ operation in certain cases in order to linearize it:

This gives us the following LP formulation:

$$\begin{aligned} \max \ & -30x + 40z \\ \text{s.t. } & z \leq x \\ & z \leq d \\ & x \geq 0, z \geq 0 \end{aligned}$$

where z is a decision variable representing the number of units to produce / sell

Mixed Integer Linear Programming in Python

Mixed Integer Linear Programming in Python

- Python has multiple packages for mathematical programming
- some of them are bound to a single solver
    - Gurobi
    - CPLEX
- some of them can work with multiple solvers
    - Python-MIP
    - CVXPY
    - PuLP
- the syntax for writing models is very similar for most of them
- the modeling objects such as variables and constraints are a bit different

Python-MIP

In this course, we will use **Python-MIP** (**https://www.python-mip.com/**) for implementing our models

- it comes with the solver CBC
- but can also be used with Gurobi and Fico XPRESS

Documentation: **https://docs.python-mip.com/en/latest/index.html**
In particular, see

- the quick start **https://python-mip.readthedocs.io/en/latest/quickstart.html**
- a couple of example models: **https://python-mip.readthedocs.io/en/latest/examples.html**

In [4]:
```
import mip
from mip import maximize
```

Our Model in Python MIP

First, create a model object. It will use Gurobi, if installed, and CBC otherwise

In [5]:
```
m = mip.Model("Capacity_Planning_Deterministic")
```

Now, add variables. In addition to lb (lower bound) important parameters are ub (upper bound) and variable type var_type

In [6]:
```
capacity =  m.add_var(name="capacity" , lb= 0)
production = m.add_var(name="production", lb= 0)
```

Add the objective function

In [7]:
```
m.objective = maximize ( -30*capacity + 40 * production )
```

and the constraints

In [8]:
```
demand = 100
m += production <= demand
m += production <= capacity
```

Now, let's solve the model:

In [9]:
```
#call the solver
m.optimize()

print(f'Capacity decision {capacity.x}')
print(f'Total Profit: {m.objective_value}' )
```

```
Capacity decision 100.0
Total Profit: 1000.0
```

A more Abstract Formulation Using Abstract Parameters:

In [9]:

```
m = mip.Model("Capacity_Planning_Deterministic_v2")

#parameters
demand = 100
installation_cost = 30
contribution_margin = 40

#decision variables
capacity =  m.add_var(name="capacity" , lb= 0)
production = m.add_var(name="production", lb= 0)


m.objective = maximize ( -installation_cost*capacity + contribution_margin* production)

m += production <= demand
m += production <= capacity


m.optimize()

print(f'Capacity decision {capacity.x}')
print(f'Total Profit: {m.objective_value}' )
```

```
Capacity decision 100.0
Total Profit: 1000.0
```

A More Complex Model: Extending to Two-Technology Setting

Generic Compact Formulation for Multiple Production Technologies

Sets:

- I: the set of technologies

Parameters:

- c_i: capacity installation cost per unit of $i \in I$
- m_i: contribution margin per unit sold for $i \in I$
- d: demand in units

Decision Variables:

- x_i: capacity installation decision technology $i \in I$
- z_i: production / sales technology $i \in I$

$$
\begin{aligned}
\max \sum_{i \in I} (-c_i x_i + m_i z_i) & \\
z_i \leq x_i & \qquad \forall i \in I \\
\sum_{i \in I} z_i \leq d & \\
x_i \geq 0, z_i \geq 0 & \qquad \forall i \in I
\end{aligned}
$$

Multiple Prodution Techonologies: Model Implementation in Python

In [10]:

```
m = mip.Model("Capacity_Planning_Two_Technologies_Deterministic")

#sets
technologies = range(2)

#parameters
demand = 1000
installation_cost = [30, 20]
contribution_margin = [40, 28]

#decision variables
capacity = [m.add_var(name=f"capacity{i}" , lb= 0) for i in technologies]
production = [m.add_var(name=f"production{i}", lb= 0) for i in technologies]

#objective
m.objective = maximize( sum(-installation_cost[i]*capacity[i] +
                            contribution_margin[i]*production[i] for i in technologies ) )
for i in technologies:
    m += production[i] <= capacity[i]

m += sum(production[i] for i in technologies) <= demand


m.optimize()

for i in technologies:
    print(f'Capacity to install from technology {i}: {capacity[i].x}')

print(f'Total Profit: {m.objective_value}' )
```

```
Capacity to install from technology 0: 1000.0
Capacity to install from technology 1: 0.0
Total Profit: 10000.0
```

New Case Study: Belt Manufacturing

Case Study: Belt Manufacturing

- A small company manufactures two types of belts: A and B. The contribution margin is $2 for an A-belt and \$1.5 for a B-belt.
- It plans the production for a week, and the comapany can sell its full production to its customer, a small chain of shops.
- Producing a belt of type A takes twice as long as producing one of type B, and the total time available in that week would allow producing 1000 belts of type B if only B-belts were produced.
- Both types of belts require the same amount of leather, and there is enough leather to produce 800 belts.
- The total number that can be produced per type is limited by the number of available bucks: The company has 400 bucks for type A and 700 bucks for type B.

Belt Manufacturing: LP Formulation

Set

- $I = \{A, B\}$ belt types

Decision Variables

- x_i: number of belts to produce from type i

$$
\begin{aligned}
\max\ & 2x_A + 1.5x_B \\
\text{s.t. } & 2x_A + x_B \leq 1000 \\
& x_A + x_B \leq 800 \\
& 0 \leq x_A \leq 400 \\
& 0 \leq x_B \leq 700
\end{aligned}
$$

Exercise: Implementation in Python

Below, you find an **almost** complete model using Python-MIP
Please, complete the model by adding

- the objective function
- the constraints

In [15]:
```
            #sets
belt_types = [0,1]
profit_contribution = [2, 1.5]
time_consumption = [2, 1]
time_available = 1000
leather_available = 800
bucks_available = [400, 700]



# Create a new model
m = mip.Model("Belt_Production_Deterministic")

print(f'Total Profit: {m.objective_value}' )
```

```
Production belt 0: 200.0
Production belt 1: 600.0
Total Profit: 1300.0
```

Part 2: Introduction to Stochastic Linear Programming

Part 2: Introduction to Stochastic Linear Programming

In this part, we will learn

- how to use sample approximations of uncertain parameters in linear programming models
- about the structure of two-stage stochastic programming models
- how to implement and solve these models in Python

Capacity Planning Case Study: Introducing Uncertainty

Let us now consider the setting from the previous weeks:

- we assume that demand is uncertain and follows a normal distribution
- as before, we create a sample approximation for the demand

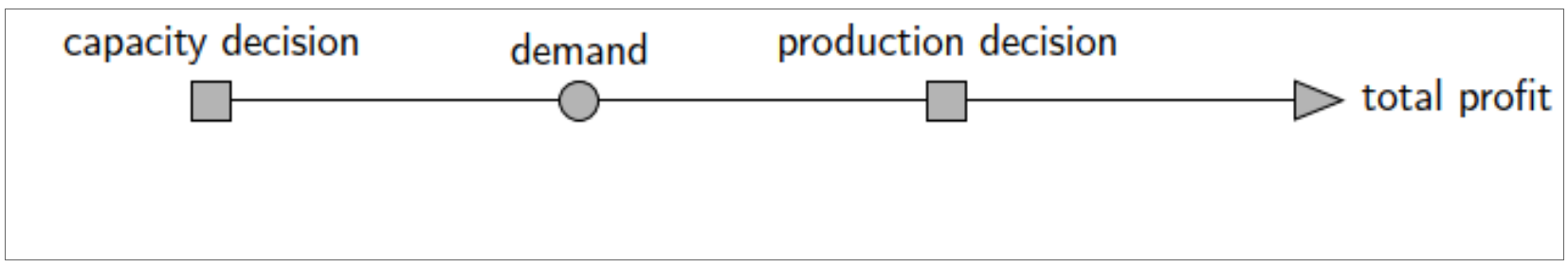
In [17]:
```
demand_dist = stats.norm(100,25)

n_samples = 10000

# we create a sample vector of demands (demand_dist was defined above), only using positive outcomes
demand_sample = np.maximum(demand_dist.rvs(n_samples),0)
```

- recall that demand uncertainty introduces a two-stage decision problem under uncertainty:

We can interpret our case study as a two-stage problem:



Before addressing this using linear programming, let us see what we did before:

Taking the Best Decisions: Formalization

We are looking for

- the decision (or decision vector, or more general, solution) x from the set of possible decisions (solutions) X
- yielding the *best* expected outcome $E(f(x, D))$ given the uncertain/random variable(s) D

We can write this as an *optimization* problem under uncertainty:

$$\max_{x \in X} E(f(x, D))$$

Using Monte Carlo, we approximate $E(f(x, D))$ by the mean of the output sample vector $\mathbf{f}(x, \mathbf{d})$, that is by $\frac{1}{|S|} \sum_{s \in S} f(x, d_s)$

This results in the following optimization problem:

$$\max_{x \in X} \frac{1}{|S|} \sum_{s \in S} f(x, d_s)$$

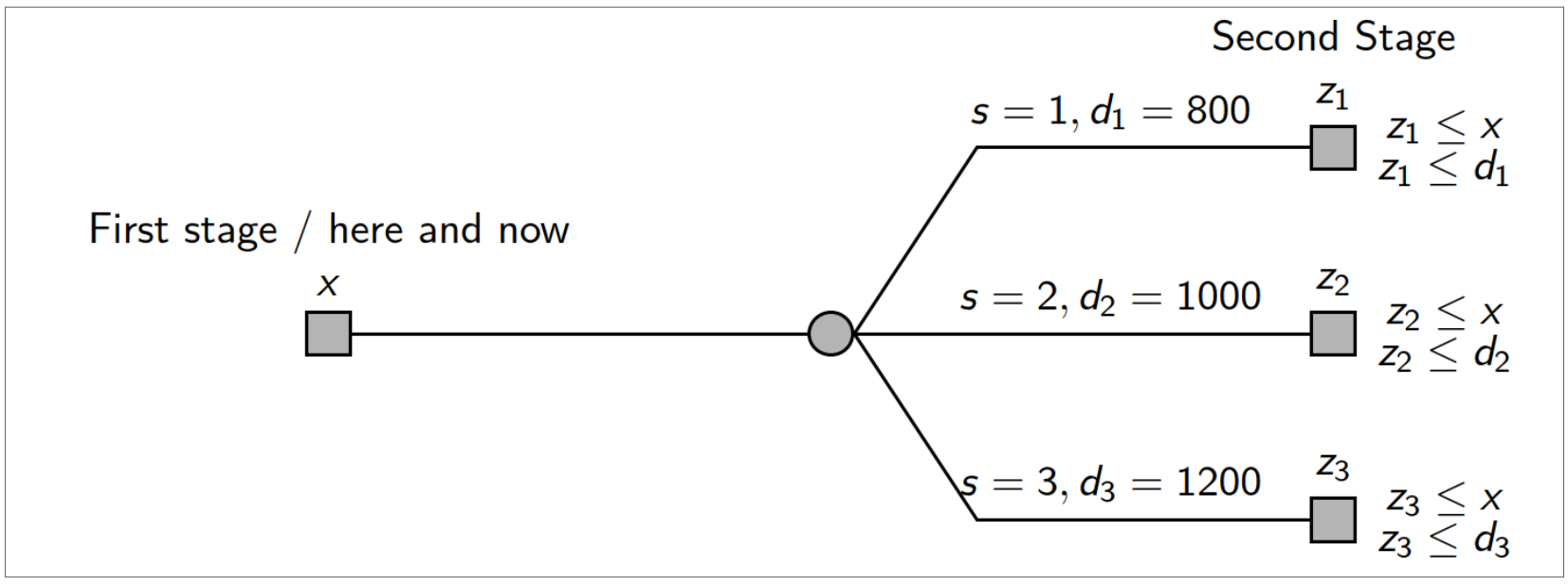
$\rightarrow$ So far, we used enumeration and unconstrained optimization to search for the best solution

$\rightarrow$ How can we use the sample approximation in the case of linear programming?

How can we Use Sample Approximation for Linear Programming?

Key Ideas (two-stage setting)

- consider first-stage and second-stage decision variables and constraints
- second-stage decision variables have a scenario index
- second-stage constraints are defined **for all** scenarios / samples



Stochastic Linear Program for our Example

Numerical example for $S = 80, 100, 120$

$$
\begin{aligned}
& \max \frac{1}{3}\sum_{s=1}^{3}(-30x + 40z_s) \\
\text{s.t.} \quad & z_1 \leq x \\
& z_1 \leq 80 \\
& z_2 \leq x \\
& z_2 \leq 100 \\
& z_3 \leq x \\
& z_3 \leq 120 \\
& x \geq 0 \\
& z_1 \geq 0, z_2 \geq 0, z_3 \geq 0
\end{aligned}
$$

Stochastic Linear Program for our Example: Compact Formulation

Sets:

- S scenario set

Parameters:

- d_s: demand in units in scenario s

Decision Variables:

- x: capacity installation decision (*first stage / here and now* decision)
- z_s: production decision for scenario s (*second stage / recourse* decision)

$$
\begin{aligned}
& \max \frac{1}{|S|} \sum_{s \in S} (-30x + 40z_s) \\
\text{s.t.} \quad & z_s \leq x && \forall s \in S \\
& z_s \leq d_s && \forall s \in S \\
& x \geq 0 \\
& z_s \geq 0 && \forall s \in S
\end{aligned}
$$

Stochastic Linear Program: Implementation in Python

In [20]:

```
                # Create a new model
m = mip.Model("Capacity_Planning_Stochastic")

#in stochastic programming, we call the samples scenarios
n_scenarios = n_samples
scenarios = np.arange(n_scenarios) # set of scenarios

#parameters
installation_cost = 30
contribution_margin = 40

#decision variables
capacity =  m.add_var(name="capacity", lb=0)

production = [m.add_var(name=f"production{s}", lb= 0) for s in scenarios]

m.objective =  maximize(1 / n_scenarios * sum(-installation_cost*capacity + contribution_margin * production[s] for s in scenarios))

for s in scenarios:
    m += production[s] <= demand_sample[s]
    m += production[s] <= capacity

m.optimize()

print(f'Capacity decision: {capacity.x:.02f}')
print(f'Expected Total Profit: {m.objective_value:.02f}' )

@vectorize
def total_profit(capacity, demand):
    return -30*capacity + 40*min(capacity, demand)

print( np.mean(total_profit (capacity.x, demand_sample ) ))
```

```
Capacity decision: 83.23
Expected Total Profit: 682.05
682.0464005141674
```

Adapting to the Conventions of Stochastic Programming

Conventions in Stochastic Programming

- S can be viewed as the index set for a sample vector $[d_s]_{s \in S}$
- in Stochastic (Linear) Programming, S is usually denoted as the set of **scenarios**
- thus, $s \in S$ is called a **scenario index** or simply a *scenario*
- a **scenario** (index) s
    - may be used for multiple parameters and decision variables and
    - represents a consistent possible state of the world
- in particular, in presence of multiple uncertain parameters, e.g. D^1 and D^2, the approximation of all scenarios $[d_s^1 d_s^2]_{s \in S}$ form an approximation of the joint probability distribution of D^1 and D^2
- also, in stochastic programming, that is, it is **not** always assumed that each scenario is equally likely, that is, the probability p_s of a scenario may be different from $\frac{1}{|S|}$
- finally, in two-stage-stochastic programming, in the objective function, the first-stage part is typically not moved into the term approximating the expectation

Stochastic Programming-Style Formulation

Sets:

- S scenario set

Parameters:

- d_s: demand in units in scenario s
- p_s: probability of scenario s

Decision Variables:

- x: capacity installation decision (*first stage / here and now* decision)
- z_s: production decision for scenario s (*second stage / recourse* decision)

$$
\begin{aligned}
\max \quad & -30x + \sum_{s \in S} p_s 40 z_s \\
\text{s.t.} \quad & z_s \leq x && \forall s \in S \\
& z_s \leq d_s && \forall s \in S \\
& x \geq 0 \\
& z_s \geq 0 && \forall s \in S
\end{aligned}
$$

Observe that in the objective function, we "removed" the first-stage term from the sum

Stochastic Programming-Style Formulation: Implementation in Python

In [7]:

```
# Create a new model
m = mip.Model("Capacity_Planning_Stochastic")


#in stochastic programming, we call the samples scenarios
#sets
n_scenarios = n_samples
scenarios = np.arange(n_scenarios)

#probability of each scenario - in our case, each scenario has prob. 1/|S|
prob = np.full((n_scenarios), 1/n_scenarios)

#parameters
installation_cost = 30
contribution_margin = 40

#decision variables
capacity =  m.add_var(name="capacity", lb=0)

production = [m.add_var(name=f"production{s}", lb= 0) for s in scenarios]

m.objective =  maximize( -installation_cost*capacity + sum(prob[s] * contribution_margin * production[s] for s in scenarios))

for s in scenarios:
    m += production[s] <= demand_sample[s]
    m += production[s] <= capacity

m.optimize()

print(f'Capacity decision: {capacity.x:.02f}')

print(f'Expected Total Profit: {m.objective_value:.02f}' )
```

```
Capacity decision: 82.80
Expected Total Profit: 672.77
```

Exercise: An additional Techology under Uncertainty

- Formulate and implement a stochastic LP model for the capacity planning problem with an additional technology!

Exercise: Belt Manufacturing under Uncertainty

Belt Manufacturing: Uncertain Machine Availability

- let us now assume that due to random machine failures, the available time is subject to uncertainty:
- we assume that available time in the same units as above is normally distributed with $\mu = 1000$ and $\sigma = 150$.
- however, due to contractual obligation, the company has to commit to a production plan *before* the machine time is known
- if the realized available time does not suffice for completnig the planned amount of belts, the company can work extra hours. Each unit of extra time costs the company $5.

Tasks:

- in the described setting, what are the first-stage and what are the second-stage decisions?
- create a two-stage stochastic programming model to model the belt manufacturing problem with uncertain machine availability
- the model should maximize the expected total profit composed of the margin of the sales and the (expected) costs for the extra hours
    - **hint:** start from the deterministic belt manufacturing model and extend it to account for uncertainty
- implement the model using Python-MIP

Understanding Two-Stage Stochastic Programs: Model Structure and Link to Monte-Carlo Simulation

Two-Stage Stochastic Programming: Model Structure Regarding Constraints

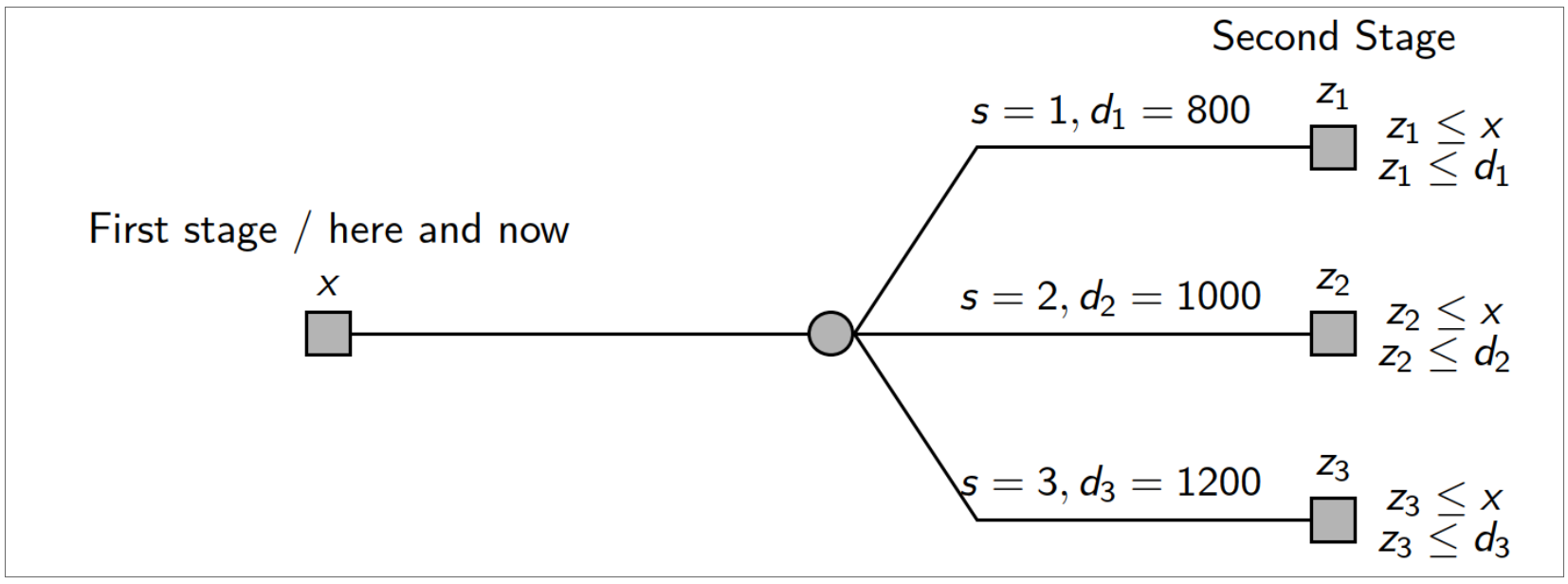
It is instructive to distinguish three types of constraints in a two-stage stochastic program:

- **First-stage constraints** only involve first-stage decision variables
- **Second-stage constraints** only involve second-stage decision variables
- **Coupling constraints** involving both types of variables

$$
\begin{aligned}
\max \; & -\sum_{i \in I} c_i x_i + \sum_{s \in S} p_s \sum_{i \in I} (m_i z_{is}) \\
\text{s.t. } & z_{is} \leq x_i && \forall i \in I, s \in S && \blacktriangleright \text{coupling constraints} \\
& \sum_{i \in I} z_{is} \leq d_s && \forall s \in S && \blacktriangleright \text{2nd stage constraints} \\
& x_i \geq 0 && \forall i \in I && \blacktriangleright \text{1st stage constraints} \\
& z_{is} \geq 0 && \forall i \in I, s \in S && \blacktriangleright \text{2nd stage constraints}
\end{aligned}
$$

Visualizing the Model Structure of Two-Stage Stochastic Programs

Let us once again consider the model structure of a two-stage stochastic program



This structure leads to two important observations:

- the second-stage part can be seen as a sample approximation / Monte-Carlo Simulation
- model parts can be seen as belonging to the first stage, the second stage or they can link both stages

Important Observation: The Second Stage in the Stochastic LP as Monte-Carlo Simulation

- only first-stage decisions really have to be taken here and now
- the second-stage decisions only will be taken after the uncertain parameters become known
- thus, in the process of taking the first-stage decision, the second-stage variables form sort of a *simulation* for evaluating the first stage-decisions
    - it is very important to recall that these decisions are never actually taken, they are only used for "evaluation" purposes

Let's verify this by using our "classical" Monte Carlo approach:

In [9]:
```
@vectorize
def total_profit(capacity,demand):
    return -30*capacity + 40* min(capacity, demand)

expected_value_monte_carlo = np.mean(total_profit(capacity.x, demand_sample))

print(f'Expected Total Profit from Stochastic LP: {m.objective_value:.02f}' )
print(f'Expected Total Profit for the LP decision as computed by Monte-Carlo Simulation: {expected_value_monte_carlo:.02f} ')
```

```
Expected Total Profit from Stochastic LP: 672.77
Expected Total Profit for the LP decision as computed by Monte-Car
lo Simulation: 672.77
```

Observe: If we fix the first-stage decision to a given value, we can even use the two-stage-stochastic program for "simulating" the second stage.

Out-of-Sample Evaluation

As described above, we use samples / scenarios within stochastic programming models

- if we evaluate our first-stage models with these samples / scenarios, we call this an **in-sample** evaluation
- the problem with this is that for complex models, it may be possible ot use only small sample sizes
- this results in a risk of overfitting the first-stage decision to the small set of samples

Out-of-sample Evaluation

- in order to check this, it is useful to perform an **out-of-sample** evaluation using a large sample in which the first-stage is fixed
- observe: if the first-stage decisions are fixed, the remaining second-stage model can be solved separately for each scenario / sample!

In [ ]:

```
number_of_samples_for_evaluation = 100000

demand_sample_new = np.maximum(demand_dist.rvs(number_of_samples_for_evaluation),0)
expected_value_out_of_sample_evaluation = np.mean(total_profit(capacity.x, demand_sample_new ))

print(f'Expected Total Profit in the out-of-sample evaluation: {expected_value_out_of_sample_evaluation:.02f}
```

More Examples for Two-Stage Problems

Two Stage Stochastic Programming Applications

Problem	*1st Stage*	*Uncertainty*	*2nd Stage*
Agricultural Planning	Planting	Yield, Price	Selling
Surgery Scheduling	Scheduling	Emergencies	Overtime, Shifting
Project Scheduling	Scheduling	Task Duration	Outsourcing, Penalty
Airline Crew Scheduling	Scheduling	Sickness, Delay	Reschedule, Standby
Aluminum Recycling	Blending	Batch Quality	Adding pure Aluminum

- in all examples, the 1st-stage decisions are different from the 2nd-stage
    - this is a key difference to multistage problems such as operating a power plant under uncertainty where the same kind of decisions are made in each stage
- the second-stage problem may involve multiple periods / stages

Case Study: A Farmer's Problem (Deterministic)

- a farmer has 450 acres of farmland and now has to decide how many acres to plant with the three crops wheat, corn and beets
- he needs some amount of each crop for feeding his cattle
- if he harvests less than needed, he can buy crops on the market
- surplus can be sold on the market

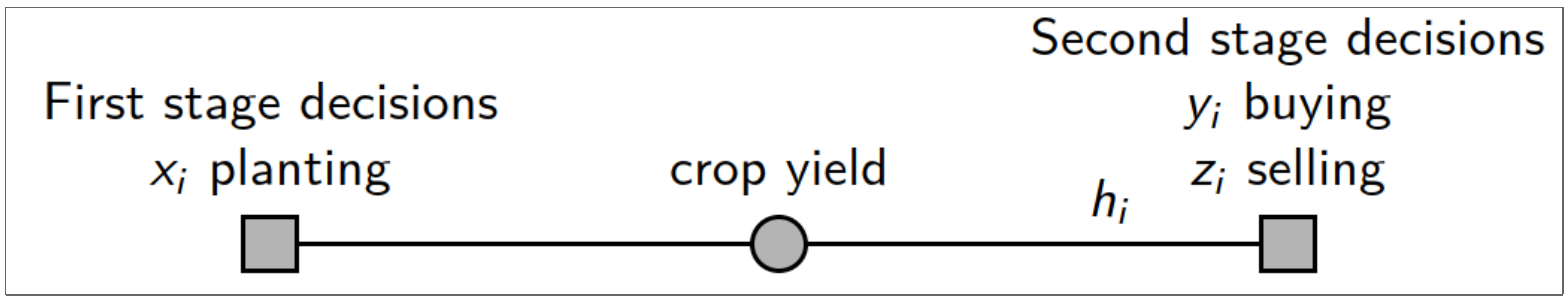
parameter	unit	wheat	corn	beets
crop yield	t per acre	2.5	3	12
planting costs	$ per acre	150	180	210
needed for feeding	t	200	250	720
buying price	$ per t	290	260	70
selling price	$ per t	200	180	45

How much area should the farmer devote to each crop to maximize his profit?

Extension: Uncertain Crop Yields h_i

Crop yields are subject to uncertainty

- we now assume that crop yields are uncertain
    - we assume that we have a **joint distribution** for yields accounting for causal structure
- planting decisions have to be taken before knowing crop yields
- buying and selling can happen after harvesting the crops
- we assume that buying and selling prices are **not** affected by uncertainty



Conclusions

In this meeting, we

- started transferring the ideas discussed in the last weeks to the world of (mixed integer) linear programming
- had an introduction to linear programming under uncertainty (stochastic programming)
- saw how we can implement these ideas in Python

In the next meeting, we will

- discuss how to model risk aversion in stochastic programming
- see how to handle settings without recourse in which we aim at ensuring feasibility with a given probability