DLP, PNLP Network Problem with two periods (Jae Yul Woo)

2 (a) Starting from 60 days prior to departure, suppose that there are no passengers from the economy plus category, but there are 102 customers who bid for itinerary 3, 26 for itinerary 4, and 58 for itinerary 5. Assuming that the customer bids are drawn from a uniform distribution in the range during that period, determine the total revenue during the first 30 day interval. Replicate this experiment 25 times (using 25 different seeds for the random number generator). Report the confidence interval of the average revenue during the 30 day interval.

First, we get new dual prices for the period {60 days : 30 days} prior to departure.

❏ DLP Formulation

```
var x3 integer;
var x4 integer;
var x5 integer;

maximize Profit: 190*x3+220*x4+140*x5;

subject to c1: x4<=100;
subject to c2: x3<=100;
subject to c3: x3+x4+x5<=300;
subject to c4: x4<=100;
subject to c5: x5<=100;
subject to c6: x3<=100;

subject to d3: x3<=200;
subject to d4: x4<=100;
subject to d5: x5<=160;
```

Results

```
ampl: reset;
ampl: model hw4.1.mod;
ampl: solve;
MINOS 5.51: ignoring integrality of 3 variables
MINOS 5.51: optimal solution found.
3 iterations, objective 55000
ampl: display c1,c2,c3,c4,c5,c6;
c1 = 0
c2 = 0
c3 = 0
c4 = 0
c5 = 140
c6 = 190

ampl: display x3,x4,x5;
x3 = 100
x4 = 100
x5 = 100
```

The Dual Price for itineray 3 is 190, itinerary 4 is 0 and itinerary 5 is 140.

❏ PNLP Formulation

```
# DLP with 2 classes, period 1
import numpy as np
import pandas as pd
import itertools
from pyomo.environ import *
from pyomo.core import *
from pyomo.pysp.annotations import (PySP_ConstraintStageAnnotation, StochasticConstraintBoundsAnnotation,
                                    StochasticConstraintBodyAnnotation)
model = ConcreteModel()

v = {1:280,2:300,3:190,4:220,5:140}
c = [[0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,0,1,0,1,0]]
capa = {1:100,2:100,3:300,4:100,5:100,6:100}
d_mean = {1:50,2:30,3:200,4:100,5:160}

# Possible Demands
d3_rhs_table = [100,200,300]
d4_rhs_table = [50,100,150]
d5_rhs_table = [80,160,240]

model.constraint_stage = PySP_ConstraintStageAnnotation()
model.stoch_rhs = StochasticConstraintBoundsAnnotation()
num_scenarios = len(d3_rhs_table) * len(d4_rhs_table) * len(d5_rhs_table)
```

```
scenario_data = dict(('Scenario'+str(i), (d3val, d4val, d5val))
                     for i, (d3val, d4val, d5val) in
                     enumerate (itertools.product(d3_rhs_table, d4_rhs_table, d5_rhs_table),1))
# Declare Variables
model.x3 = Var(within=NonNegativeIntegers)
model.x4 = Var(within=NonNegativeIntegers)
model.x5 = Var(within=NonNegativeIntegers)

model.t3 = Var(within=NonNegativeIntegers)
model.t4 = Var(within=NonNegativeIntegers)
model.t5 = Var(within=NonNegativeIntegers)

model.d3_rhs = Param(mutable=True, initialize=0.0)
model.d4_rhs = Param(mutable=True, initialize=0.0)
model.d5_rhs = Param(mutable=True, initialize=0.0)

# Objective
model.FirstStageCost = Expression(initialize=0)
model.SecondStageCost = Expression(initialize=model.t3*v[3]+model.t4*v[4]+model.t5*v[5])

# Capacity Constraints
c1 =np.transpose(c)[0]
model.c1 = Constraint(expr= model.x3*c1[2]+model.x4*c1[3]+model.x5*c1[4] <= capa[1])
model.constraint_stage.declare(model.c1,1)
c2 =np.transpose(c)[1]
model.c2 = Constraint(expr= model.x3*c2[2]+model.x4*c2[3]+model.x5*c2[4] <= capa[2])
model.constraint_stage.declare(model.c2,1)
c3 =np.transpose(c)[2]
model.c3 = Constraint(expr= model.x3*c3[2]+model.x4*c3[3]+model.x5*c3[4] <= capa[3])
model.constraint_stage.declare(model.c3,1)
c4 =np.transpose(c)[3]
model.c4 = Constraint(expr= model.x3*c4[2]+model.x4*c4[3]+model.x5*c4[4] <= capa[4])
model.constraint_stage.declare(model.c4,1)
c5 =np.transpose(c)[4]
model.c5 = Constraint(expr= model.x3*c5[2]+model.x4*c5[3]+model.x5*c5[4] <= capa[5])
model.constraint_stage.declare(model.c5,1)
c6 =np.transpose(c)[5]
model.c6 = Constraint(expr= model.x3*c6[2]+model.x4*c6[3]+model.x5*c6[4] <= capa[6])
model.constraint_stage.declare(model.c6,1)

# Second Stage Min{X,D} Constraint (when D is stochastic)
model.d31 = Constraint(expr=model.t3 <= model.d3_rhs)
model.constraint_stage.declare(model.d31,2)
model.stoch_rhs.declare(model.d31)

model.d32 = Constraint(expr=model.t3 - model.x3 <= 0)
model.constraint_stage.declare(model.d32,2)
model.d41 = Constraint(expr=model.t4 <= model.d4_rhs)
model.constraint_stage.declare(model.d41,2)
model.stoch_rhs.declare(model.d41)

model.d42 = Constraint(expr=model.t4 - model.x4 <= 0)
model.constraint_stage.declare(model.d42,2)

model.d51 = Constraint(expr=model.t5 <= model.d5_rhs)
model.constraint_stage.declare(model.d51,2)
model.stoch_rhs.declare(model.d51)

model.d52 = Constraint(expr=model.t5 - model.x5 <= 0)
```

```
model.constraint_stage.declare(model.d52,2)

# Final Objective
model.obj = Objective(expr=(model.FirstStageCost+model.SecondStageCost)*(-1))

def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model import \
        CreateConcreteTwoStageScenarioTreeModel
    st_model = CreateConcreteTwoStageScenarioTreeModel(num_scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()

    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('x3')
    st_model.StageVariables[first_stage].add('x4')
    st_model.StageVariables[first_stage].add('x5')
    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'
    st_model.StageVariables[second_stage].add('t3')
    st_model.StageVariables[second_stage].add('t4')
    st_model.StageVariables[second_stage].add('t5')
    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):

    instance = model.clone()
    d3_rhs_val, d4_rhs_val, d5_rhs_val  = scenario_data[scenario_name]
    instance.d3_rhs.value = d3_rhs_val
    instance.d4_rhs.value = d4_rhs_val
    instance.d5_rhs.value = d5_rhs_val

    return instance
```

```
Problem:                              itinerary
First Stage Rows:                     7
First Stage Columns:                  3
First Stage Non-zeros:                8
Mean solution is recommended for this instance.
Number of replications:               3
Status:                               MEAN SOLUTION
Total Objective Function Upper Bound:   -50290.609,[-50541.940188,-50039.277797],half-width:251.331,
stdev:128.230
Total Objective Function Lower Bound:   -50557.112,[-50663.181388,-50451.041899],half-width:106.070,
stdev:24.835

First Stage Solutions:
   No.   Row name   Activity      Lower bound   Upper bound   Dual          Dual STDEV
------ ------------ ------------ ------------ ------------ ------------ ------------
     0 obj          -5.029061e+04
     1 c_u_c1_       1.000000e+02               1.000000e+02  0.000000e+00  0.000000e+00
     2 c_u_c2_       1.000000e+02               1.000000e+02 -1.787758e+02  1.691334e+01
     3 c_u_c3_       3.000000e+02               3.000000e+02  0.000000e+00  0.000000e+00
     4 c_u_c4_       1.000000e+02               1.000000e+02 -9.428991e+01  2.155608e+01
     5 c_u_c5_       1.000000e+02               1.000000e+02 -8.950978e+01  4.219948e+00
     6 c_u_c6_       1.000000e+02               1.000000e+02  0.000000e+00  0.000000e+00

   No. Column name  Activity      Lower bound   Upper bound   Reduced Cost  RC STDEV
------ ------------ ------------ ------------ ------------ ------------ ------------
     1 x3           1.000000e+02  0.000000e+00               0.000000e+00  0.000000e+00
     2 x4           1.000000e+02  0.000000e+00               0.000000e+00  0.000000e+00
     3 x5           1.000000e+02  0.000000e+00               0.000000e+00  0.000000e+00
```

The Dual Price for itineray 3 is 178.78, itinerary 4 is 94.29 and itinerary 5 is 89.51.

Second, we generate 25 replications with random seeds deriven from seed(1) and calculate the expected total revenue with respect to the dual price for each itinerary. Then we find the 95% Confidence Interval for the total revenue.
The customer bids of period 1 is driven from the uniform distribution $\{vi-0.3vi, vi+0.3vi\}$.

❏ DLP : 95% Confidence Interval for mean total revenue

```
np.random.seed(1)
seed_lst= np.random.randint(100,size=25)

# General Form for 5 bids
v = [280,300,190,220,140]
v_dual = [0,0,190,0,140]  # dual price for DLP formulation

v_bid = []
for i in range(len(seed_lst)): # create a list w.r.t. seeds with nested v_i lists
    np.random.seed(seed_lst[i])
    v_bid.append([])
    for j in range(5):
        v_bid[i].append(np.random.randint(low=v[j]*0.7,high=v[j]*1.3)) # fills nested lists with data

for i in range(len(seed_lst)):
    for j in range(5):
        if v_bid[i][j] >= v_dual[j]:
            v_bid[i][j] = 0
        else: v_bid[i][j]
# print(v_bid)

# For revenue calculation of itinerary 3,4,5
for i in range(len(seed_lst)):
    for j in range(2):
        v_bid[i][j] = 0
# print(v_bid)

# Total revenue for 25 replications
total_revenue=[]
for i in range(len(seed_lst)):
    total_revenue.append(v_bid[i][0]*0+v_bid[i][1]*0+v_bid[i][2]*102+v_bid[i][3]*26+v_bid[i][4]*58)
# print(total_revenue)

import scipy.stats as st
st.t.interval(0.95, len(total_revenue)-1, loc=np.mean(total_revenue),scale=st.sem(total_revenue))
print(ci1)

# number of passengers who bought during the first interval
d_origin = [0,0,102,26,58]
d_new1 = []
for i in range(len(seed_lst)):
    d_new1.append([])
    for j in range(5):
        d_new1[i].append(d_origin[j] if v_bid[i][j]>0 else 0)
```

The 95% Confidence Interval of the mean (total revenue) is **(8099.3010577808809, 16166.458942219117).**

❏ PNLP : 95% Confidence Interval for mean total revenue

```
np.random.seed(1)
seed_lst= np.random.randint(100,size=25)

# General Form for 5 bids
v = [280,300,190,220,140]
v_dual = [0,0,178.78,94.29,89.51] # dual price for PNLP formulation

v_bid = []
for i in range(len(seed_lst)): # create a list w.r.t. seeds with nested v_i lists
```

```
    np.random.seed(seed_lst[i])
    v_bid.append([])
    for j in range(5):
        v_bid[i].append(np.random.randint(low=v[j]*0.7,high=v[j]*1.3)) # fills nested lists with data

for i in range(len(seed_lst)):
    for j in range(5):
        if v_bid[i][j] >= v_dual[j]:
            v_bid[i][j] = 0
        else: v_bid[i][j]
# print(v_bid)

# For revenue calculation of itinerary 3,4,5
for i in range(len(seed_lst)):
    for j in range(2):
        v_bid[i][j] = 0
# print(v_bid)

# Total revenue for 25 replications
total_revenue=[]
for i in range(len(seed_lst)):
    total_revenue.append(v_bid[i][0]*0+v_bid[i][1]*0+v_bid[i][2]*102+v_bid[i][3]*26+v_bid[i][4]*58)
# print(total_revenue)

import scipy.stats as st
st.t.interval(0.95, len(total_revenue)-1, loc=np.mean(total_revenue),scale=st.sem(total_revenue))
print(ci2)

# number of passengers who bought during the first interval
d_origin = [0,0,102,26,58]
d_new2 = []
for i in range(len(seed_lst)):
    d_new2.append([])
    for j in range(5):
        d_new2[i].append(d_origin[j] if v_bid[i][j]>0 else 0)
```

The 95% Confidence Interval of the mean (total revenue) is **(3570.3093264873924, 10122.170673512606).**

Formulation Methodology	95% Confidence Interval for the mean
DLP	(8099.30, 16166.46)
PLNP	(3570.31, 10122.17)

Hence, we got the better result with DLP when passengers are 60 days to 30 days prior to departure.

2 (b) For each replication in part 2a, re-solve the models of 1a) and 1c) at the 30 day mark, and simulate the remaining arrivals during the last 30 days, with the customer bids. Moreover, the total number of requests for each itinerary should be drawn from the uniform distribution.

First, get the dual solutions of DLP and PNLP for each replication.

```
import numpy as np
import pandas as pd
import itertools
from pyomo.environ import *
```

```
import pyomo.environ as pyo
from pyomo.core import *
from pyomo.pysp.annotations import (PySP_ConstraintStageAnnotation, StochasticConstraintBoundsAnnotation,
                                    StochasticConstraintBodyAnnotation)
model = ConcreteModel()
# --------k is the replication number-------
k = 0
# ----------k is [0,1, ..., 24]-------------

v = {1:280,2:300,3:190,4:220,5:140}
v_bid1 = [[339, 317, 209, 176, 0], [271, 365, 0, 284, 0], [215, 256, 207, 255, 167], [322, 302, 0, 210, 0],
[308, 229, 214, 180, 168],[295, 328, 0, 227, 0], [283, 284, 200, 236, 0], [315, 376, 0, 205, 146], [317, 279,
198, 277, 166], [233, 350, 205, 233, 162], [357, 345, 0, 180, 140], [303, 383, 0, 209, 0], [334, 316, 242,
233, 178], [328, 272, 223, 215, 0], [335, 319, 0, 224, 168], [295, 225, 228, 229, 0], [238, 229, 202, 220, 0],
[285, 266, 211, 263, 162], [349, 290, 224, 235, 153], [197, 215, 0, 186, 0], [311, 244, 0, 250, 0], [303, 298,
0, 225, 0], [335, 319, 0, 224, 168], [259, 377, 0, 178, 0], [273, 353, 0, 205, 0]]
v_bid2 = [[339, 317, 209, 176, 133], [271, 365, 139, 284, 101], [215, 256, 207, 255, 167], [322, 302, 187,
210, 120], [308, 229, 214, 180, 168], [295, 328, 149, 227, 106], [283, 284, 200, 236, 99], [315, 376, 187,
205, 146], [317, 279, 198, 277, 166], [233, 350, 205, 233, 162], [357, 345, 133, 180, 140], [303, 383, 144,
209, 138], [334, 316, 242, 233, 178], [328, 272, 223, 215, 121], [335, 319, 166, 224, 168], [295, 225, 228,
229, 120], [238, 229, 202, 220, 106], [285, 266, 211, 263, 162], [349, 290, 224, 235, 153], [197, 215, 155,
186, 101], [311, 244, 173, 250, 122], [303, 298, 145, 225, 126], [335, 319, 166, 224, 168], [259, 377, 169,
178, 121], [273, 353, 139, 205, 107]]
c = [[0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,0,1,0,1,0]]
capa = {1:100,2:100,3:300,4:100,5:100,6:100}

d_mean = [50,30,200,100,160]
d_new1 = [[0, 0, 102, 26, 0], [0, 0, 0, 26, 0], [0, 0, 102, 26, 58], [0, 0, 0, 26, 0], [0, 0, 102, 26, 58],
          [0, 0, 0, 26, 0], [0, 0, 102, 26, 0], [0, 0, 0, 26, 58], [0, 0, 102, 26, 58], [0, 0, 102, 26, 58],
          [0, 0, 0, 26, 58], [0, 0, 0, 26, 0], [0, 0, 102, 26, 58], [0, 0, 102, 26, 0], [0, 0, 0, 26, 58],
          [0, 0, 102, 26, 0], [0, 0, 102, 26, 0], [0, 0, 102, 26, 58], [0, 0, 102, 26, 58], [0, 0, 0, 26, 0],
          [0, 0, 0, 26, 0], [0, 0, 0, 26, 0], [0, 0, 0, 26, 58], [0, 0, 0, 26, 0], [0, 0, 0, 26, 0]]
d_new2 = [[0, 0, 0, 0, 0], [0, 0, 102, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0],
          [0, 0, 102, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0],
          [0, 0, 102, 0, 0], [0, 0, 102, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 102, 0, 0],
          [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 102, 0, 0],
          [0, 0, 102, 0, 0], [0, 0, 102, 0, 0], [0, 0, 102, 0, 0], [0, 0, 102, 0, 0], [0, 0, 102, 0, 0]]

# Possible Demands
d1_rhs_table = [ceil((d_mean[0]-d_new1[k][0])*0.9),floor((d_mean[0]-d_new1[k][0])*1.1)+1]
d2_rhs_table = [ceil((d_mean[1]-d_new1[k][1])*0.9),floor((d_mean[1]-d_new1[k][1])*1.1)+1]
d3_rhs_table = [ceil((d_mean[2]-d_new1[k][2])*0.9),floor((d_mean[2]-d_new1[k][2])*1.1)+1]
d4_rhs_table = [ceil((d_mean[3]-d_new1[k][3])*0.9),floor((d_mean[3]-d_new1[k][3])*1.1)+1]
d5_rhs_table = [ceil((d_mean[4]-d_new1[k][4])*0.9),floor((d_mean[4]-d_new1[k][4])*1.1)+1]

model.constraint_stage = PySP_ConstraintStageAnnotation()
model.stoch_rhs = StochasticConstraintBoundsAnnotation()
num_scenarios = len(d1_rhs_table) * len(d2_rhs_table) * len(d3_rhs_table) * len(d4_rhs_table) *
len(d5_rhs_table)
scenario_data = dict(('Scenario'+str(i), (d1val, d2val, d3val, d4val, d5val))
                     for i, (d1val, d2val, d3val, d4val, d5val) in
                     enumerate (itertools.product(d1_rhs_table, d2_rhs_table, d3_rhs_table, d4_rhs_table,
d5_rhs_table),1))

# Declare Variables
```

```
model.x1 = Var(within=NonNegativeIntegers)
model.x2 = Var(within=NonNegativeIntegers)
model.x3 = Var(within=NonNegativeIntegers)
model.x4 = Var(within=NonNegativeIntegers)
model.x5 = Var(within=NonNegativeIntegers)

model.t1 = Var(within=NonNegativeIntegers)
model.t2 = Var(within=NonNegativeIntegers)
model.t3 = Var(within=NonNegativeIntegers)
model.t4 = Var(within=NonNegativeIntegers)
model.t5 = Var(within=NonNegativeIntegers)

model.d1_rhs = Param(mutable=True, initialize=0.0)
model.d2_rhs = Param(mutable=True, initialize=0.0)
model.d3_rhs = Param(mutable=True, initialize=0.0)
model.d4_rhs = Param(mutable=True, initialize=0.0)
model.d5_rhs = Param(mutable=True, initialize=0.0)

# Objective
model.FirstStageCost = Expression(initialize=0)
model.SecondStageCost =
Expression(initialize=model.t1*v_bid1[k][0]+model.t2*v_bid1[k][1]+model.t3*v_bid1[k][2]
                      +model.t4*v_bid1[k][3]+model.t5*v_bid1[k][4])

# Capacity Constraints
c1 =np.transpose(c)[0]
model.c1 = Constraint(expr= model.x1*c1[0]+model.x2*c1[1]+model.x3*c1[2]
                      +model.x4*c1[3]+model.x5*c1[4] <= capa[1])
model.constraint_stage.declare(model.c1,1)
c2 =np.transpose(c)[1]
model.c2 = Constraint(expr= model.x1*c2[0]+model.x2*c2[1]+model.x3*c2[2]
                      +model.x4*c2[3]+model.x5*c2[4] <= capa[2])
model.constraint_stage.declare(model.c2,1)
c3 =np.transpose(c)[2]
model.c3 = Constraint(expr= model.x1*c3[0]+model.x2*c3[1]+model.x3*c3[2]
                      +model.x4*c3[3]+model.x5*c3[4] <= capa[3])
model.constraint_stage.declare(model.c3,1)
c4 =np.transpose(c)[3]
model.c4 = Constraint(expr= model.x1*c4[0]+model.x2*c4[1]+model.x3*c4[2]
                      +model.x4*c4[3]+model.x5*c4[4] <= capa[4])
model.constraint_stage.declare(model.c4,1)
c5 =np.transpose(c)[4]
model.c5 = Constraint(expr= model.x1*c5[0]+model.x2*c5[1]+model.x3*c5[2]
                      +model.x4*c5[3]+model.x5*c5[4] <= capa[5])
model.constraint_stage.declare(model.c5,1)
c6 =np.transpose(c)[5]
model.c6 = Constraint(expr= model.x1*c6[0]+model.x2*c6[1]+model.x3*c6[2]
                      +model.x4*c6[3]+model.x5*c6[4] <= capa[6])
model.constraint_stage.declare(model.c6,1)

# Second Stage Min{X,D} Constraint (when D is stochastic)
model.d11 = Constraint(expr=model.t1 <= model.d1_rhs)
model.constraint_stage.declare(model.d11,2)
model.stoch_rhs.declare(model.d11)

model.d12 = Constraint(expr=model.t1 - model.x1 <= 0)
model.constraint_stage.declare(model.d12,2)

model.d21 = Constraint(expr=model.t2 <= model.d2_rhs)
```

```
model.constraint_stage.declare(model.d21,2)
model.stoch_rhs.declare(model.d21)

model.d22 = Constraint(expr=model.t2 - model.x2 <= 0)
model.constraint_stage.declare(model.d22,2)

model.d31 = Constraint(expr=model.t3 <= model.d3_rhs)
model.constraint_stage.declare(model.d31,2)
model.stoch_rhs.declare(model.d31)

model.d32 = Constraint(expr=model.t3 - model.x3 <= 0)
model.constraint_stage.declare(model.d32,2)

model.d41 = Constraint(expr=model.t4 <= model.d4_rhs)
model.constraint_stage.declare(model.d41,2)
model.stoch_rhs.declare(model.d41)

model.d42 = Constraint(expr=model.t4 - model.x4 <= 0)
model.constraint_stage.declare(model.d42,2)

model.d51 = Constraint(expr=model.t5 <= model.d5_rhs)
model.constraint_stage.declare(model.d51,2)
model.stoch_rhs.declare(model.d51)

model.d52 = Constraint(expr=model.t5 - model.x5 <= 0)
model.constraint_stage.declare(model.d52,2)

# Final Objective
model.obj = Objective(expr=(model.FirstStageCost+model.SecondStageCost)*(-1))

def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model import \
        CreateConcreteTwoStageScenarioTreeModel

    st_model = CreateConcreteTwoStageScenarioTreeModel(num_scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()

    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('x1')
    st_model.StageVariables[first_stage].add('x2')
    st_model.StageVariables[first_stage].add('x3')
    st_model.StageVariables[first_stage].add('x4')
    st_model.StageVariables[first_stage].add('x5')
    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'
    st_model.StageVariables[second_stage].add('t1')
    st_model.StageVariables[second_stage].add('t2')
    st_model.StageVariables[second_stage].add('t3')
    st_model.StageVariables[second_stage].add('t4')
    st_model.StageVariables[second_stage].add('t5')
    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):

    instance = model.clone()
    d1_rhs_val, d2_rhs_val, d3_rhs_val, d4_rhs_val, d5_rhs_val  = scenario_data[scenario_name]
```

```
    instance.d1_rhs.value = d1_rhs_val
    instance.d2_rhs.value = d2_rhs_val
    instance.d3_rhs.value = d3_rhs_val
    instance.d4_rhs.value = d4_rhs_val
    instance.d5_rhs.value = d5_rhs_val

    return instance
```

```
!python -m pyomo.pysp.convert.smps -m /Users/yul/Desktop/itinerary_network_2periods_py/itinerary_network_2periods_first.py
--basename Itinerary1.0 \--output-directory /Users/yul/Desktop/itinerary_network_2periods_py/sdinput/Itinerary1.0
--symbolic-solver-labels
```

```
# --------k is the replication number-------
k = 0
# ----------k is [0,1, ..., 24]-------------

d_table1 = []
for i in range(25):
    d_table1.append([])
    for j in range(5):
        d_table1[i].append(str(ceil((d_mean[j]-d_new1[i][j])*0.9))+' ~
'+str(floor((d_mean[j]-d_new1[i][j])*1.1)+1))
print(d_table1[k])

d_table2 = []
for i in range(25):
    d_table2.append([])
    for j in range(5):
        d_table2[i].append(str(ceil((d_mean[j]-d_new2[i][j])*0.9))+' ~
'+str(floor((d_mean[j]-d_new2[i][j])*1.1)+1))
print(d_table2[k])
```

Change the .sto file according to the demand probabilities when **k=0** :
d1_table = ['45 ~ 56', '27 ~ 34', '89 ~ 108', '67 ~ 82', '144 ~ 177'] (DLP)
d2_table = ['45 ~ 56', '27 ~ 34', '180 ~ 221', '90 ~ 111', '144 ~ 177'] (PNLP)
→ Do the same procedure for k=0 to k=24 for DLP and PNLP repectively.

❏ Derive each dual solution for replications of DLP scenario using SD solver.

Scenario for DLP Formulation (The dual solution for each leg)					
0	[0,0,0,0,0,208.51]	9	[0,205.00,0,0,0,0]	18	[0,0,0,0,0,223.46]
1	[0,0,0,0,0,0]	10	[0,0,0,0,0,0]	19	[0,0,0,0,0,0]
2	[0,0,0,0,166.39,206,24]	11	[0,0,0,0,0,0]	20	[0,0,0,0,0,0]
3	[0,0,0,0,166.39,206,24]	12	[0,241.75,0,0,0,0]	21	[0,0,0,0,0,0]
4	[0,0,0,0,0,206.24]	13	[0,0,0,0,0,222.43]	22	[0,0,0,0,0,0]
5	[0,0,0,0,166.98,206.97]	14	[0,0,0,0,0,0]	23	[0,0,0,0,0,0]
6	[0,199.78,0,0,0,0]	15	[0,0,0,0,0,227.80]	24	[0,0,0,0,0,0]
7	[0,0,0,0,0,0]	16	[0,0,0,0,0,201.35]		
8	[0,0,0,0,0,197.53]	17	[0,0,0,0,0,210.40]		

❏ Derive each dual solution for replications of PNLP scenario using SD solver.

Scenario for DLP Formulation (The dual solution for each leg)					
0	[0,0,63.48,112.10,69.20,145.02]	9	[0,0,76.20,156.01,85.25,128.11]	18	[0,0,74.37,159.91,78.16,148.95]
1	[0,0,51.85,231.31,48.85,86.74]	10	[0,0,54.18,125.27,85.40,78.43]	19	[0,0,44.43,140.84,56.17,109.96]
2	[0,0,76.77,177.19,89.55,129.39]	11	[0,0,61.21,147.34,76.49,82.48]	20	[0,0,54.42,187.88,66.73,117.38]
3	[0,0,57.11,152.34,62.57,129.40]	12	[0,0,65.44,143.12,72.27,78.55]	21	[0,0,58.39,165.97,67.25,86.20]
4	[0,0,74.94,104.53,92.56,138.43]	13	[0,0,64.50,149.86,56.14,157.84]	22	[0,0,77.71,145.70,89.85,87.85]
5	[0,0,55.20,171.21,50.52,93.41]	14	[0,0,77.71,145.70,89.85,87.85]	23	[0,0,56.33,121.22,64.37,112.25]
6	[0,0,55.20,179.82,43.39,143.97]	15	[0,0,60.59,163.20,58.77,166.19]	24	[0,0,52.46,152.09,54.30,86.23]
7	[0,0,66.19,138.33,79.47,120.38]	16	[0,0,55.50,163.50,50.02,145.58]		
8	[0,0,76.49,198.67,88.41,120.20]	17	[0,0,78.65,182.85,82.42,131.15]		

Now, we can get dual prices of every itinerary for each replication.

```
# list of dual solutions
dual_sol1 = [[0,0,0,0,0,208.51], [0,0,0,0,0,0], [0,0,0,0,166.39,206,24], [0,0,0,0,166.39,206,24],
[0,0,0,0,0,206.24], [0,0,0,0,166.98,206.97], [0,199.78,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,197.53],
[0,205.00,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,0], [0,241.75,0,0,0,0], [0,0,0,0,0,222.43], [0,0,0,0,0,0],
[0,0,0,0,0,227.80], [0,0,0,0,0,201.35], [0,0,0,0,0,210.40], [0,0,0,0,0,223.46], [0,0,0,0,0,0],
[0,0,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,0]]
dual_sol2 = [[0,0,63.48,112.10,69.20,145.02], [0,0,51.85,231.31,48.85,86.74], [0,0,76.77,177.19,89.55,129.39],
[0,0,57.11,152.34,62.57,129.40], [0,0,74.94,104.53,92.56,138.43], [0,0,55.20,171.21,50.52,93.41],
[0,0,55.20,179.82,43.39,143.97], [0,0,66.19,138.33,79.47,120.38], [0,0,76.49,198.67,88.41,120.20],
[0,0,76.20,156.01,85.25,128.11], [0,0,54.18,125.27,85.40,78.43], [0,0,61.21,147.34,76.49,82.48],
[0,0,65.44,143.12,72.27,78.55], [0,0,64.50,149.86,56.14,157.84], [0,0,77.71,145.70,89.85,87.85],
[0,0,60.59,163.20,58.77,166.19], [0,0,55.50,163.50,50.02,145.58], [0,0,78.65,182.85,82.42,131.15],
[0,0,74.37,159.91,78.16,148.95], [0,0,44.43,140.84,56.17,109.96], [0,0,54.42,187.88,66.73,117.38],
[0,0,58.39,165.97,67.25,86.20], [0,0,77.71,145.70,89.85,87.85], [0,0,56.33,121.22,64.37,112.25],
[0,0,52.46,152.09,54.30,86.23]]

# Dual price for itinerary
dual_price1 = []
for i in range(len(seed_lst)):
    dual_price1.append([0,0,0,0,0])

for i in range(len(seed_lst)):
    dual_price1[i][0] = dual_sol1[i][1]+dual_sol1[i][2]+dual_sol1[i][5]
    dual_price1[i][1] = dual_sol1[i][0]+dual_sol1[i][2]+dual_sol1[i][3]
    dual_price1[i][2] = dual_sol1[i][1]+dual_sol1[i][2]+dual_sol1[i][5]
    dual_price1[i][3] = dual_sol1[i][0]+dual_sol1[i][2]+dual_sol1[i][3]
    dual_price1[i][4] = dual_sol1[i][2]+dual_sol1[i][4]

dual_price2 = []
for i in range(len(seed_lst)):
    dual_price2.append([0,0,0,0,0])

for i in range(len(seed_lst)):
    dual_price2[i][0] = dual_sol2[i][1]+dual_sol2[i][2]+dual_sol2[i][5]
    dual_price2[i][1] = dual_sol2[i][0]+dual_sol2[i][2]+dual_sol2[i][3]
    dual_price2[i][2] = dual_sol2[i][1]+dual_sol2[i][2]+dual_sol2[i][5]
```

```
    dual_price2[i][3] = dual_sol2[i][0]+dual_sol2[i][2]+dual_sol2[i][3]
    dual_price2[i][4] = dual_sol2[i][2]+dual_sol2[i][4]

# Demand for period 2 only (take-out the used-up demand in period 1).
d_period2_1 = []
for i in range(len(seed_lst)):
    d_period2_1.append([])

for i in range(len(seed_lst)):
    np.random.seed(seed_lst[i])
    for j in range(5):
        d_period2_1[i].append(np.random.randint(low=ceil((d_mean[j]-d_new1[i][j])*0.9),
                                                high=floor((d_mean[j]-d_new1[i][j])*1.1)+1))
# print(d_period2_1)
d_period2_2 = []
for i in range(len(seed_lst)):
    d_period2_2.append([])

for i in range(len(seed_lst)):
    np.random.seed(seed_lst[i])
    for j in range(5):
        d_period2_2[i].append(np.random.randint(low=ceil((d_mean[j]-d_new2[i][j])*0.9),
                                                high=floor((d_mean[j]-d_new2[i][j])*1.1)+1))
# print(d_period2_2)
```

Second, we generate 25 replications with random seeds deriven from seed(1) and calculate the expected total revenue with respect to the dual price for each itinerary. Then we find the 95% Confidence Interval for the total revenue.
The customer bids of period 2 is driven from the uniform distribution $\{vi-0.1vi, vi+0.1vi\}$.

- ❏ DLP : 95% Confidence Interval for the mean total revenue

```
np.random.seed(1)
seed_lst= np.random.randint(100,size=25)

v = [280,300,190,220,140]

v_bid = []
for i in range(len(seed_lst)): # create a list w.r.t. seeds with nested v_i lists
    np.random.seed(seed_lst[i])
    v_bid.append([])
    for j in range(5):
        v_bid[i].append(np.random.randint(low=v[j]*0.9,high=v[j]*1.1)) # fills nested lists with data
# print(v_bid)

# DLP
for i in range(len(seed_lst)):
    for j in range(5):
        if v_bid[i][j] >= dual_price1[i][j]:
            v_bid[i][j] = 0
        else: v_bid[i][j]
# print(v_bid)

# Total revenue for 25 replications
total_revenue_1=[]
for i in range(len(seed_lst)):
    total_revenue_1.append(v_bid[i][0]*d_period2_1[i][0]+v_bid[i][1]*d_period2_1[i][1]
```

```
                        +v_bid[i][2]*d_period2_1[i][2]+v_bid[i][3]*d_period2_1[i][3]
                        +v_bid[i][4]*d_period2_1[i][4])
# print(total_revenue)

import scipy.stats as st
ci2_1 = st.t.interval(0.95, len(total_revenue_1)-1, loc=np.mean(total_revenue_1),
              scale=st.sem(total_revenue_1))
print(ci2_1)
```

The 95% Confidence Interval of the mean (total revenue) is **(7190.245275440534, 21480.074724559465)**.

- ❏ PNLP : 95% Confidence Interval for the mean total revenue.

```
# PNLP
for i in range(len(seed_lst)):
    for j in range(5):
        if v_bid[i][j] >= dual_price2[i][j]:
            v_bid[i][j] = 0
        else: v_bid[i][j]
# print(v_bid)

# Total revenue for 25 replications
total_revenue_2=[]
for i in range(len(seed_lst)):
    total_revenue_2.append(v_bid[i][0]*d_period2_2[i][0]+v_bid[i][1]*d_period2_2[i][1]
                        +v_bid[i][2]*d_period2_2[i][2]+v_bid[i][3]*d_period2_2[i][3]
                        +v_bid[i][4]*d_period2_2[i][4])
# print(total_revenue)

ci2_2 = st.t.interval(0.95, len(total_revenue_2)-1, loc=np.mean(total_revenue_2),
              scale=st.sem(total_revenue_2))
print(ci2_2)
```

The 95% Confidence Interval of the mean (total revenue) is **(8670.8071419958796, 25169.912858004122)**.

Finally, derive the 95% confidence interval for both periods.

Formulation Methodology	95% Confidence Interval for the mean total revenue		
	Period 1	Period 2	Period 1 and 2
DLP	(8099.30, 16166.46)	(7190.25, 21480.07)	(15289.55, 37646.53)
PLNP	(3570.31, 10122.17)	(8670.81, 25169.91)	(12241.12, 35292.08)

Hence, we got the better result with DLP when passengers are 60 days to 30 days prior to departure while the better result was with PNLP when passengers are closer to the departure date. We can conclude that DLP formulation made the better expected total revenue compared to the PNLP formualation when we take both periods into account.