Washington University in St.Louis
OLIN BUSINESS SCHOOL

OMM 500N – Fall 2018
Prescriptive Analytics
L. Dong

Introduction to Optimization in Python

Limitations of Excel's Standard Solver

- Problem size limits:
  - 200 decision variables;
  - No limit on constraints for problems where "Assume Linear Model" is checked. A limit of 100 cell constraints on nonlinear problems (excluding simple variable bounds and integrality).
- Excel models are not easily scalable.
- Generating large-scale models with tabular structure is inefficient because these models are often sparse.
- Excel models do not cleanly separate the "model" from the "data".
- Does not incorporate state-of-the art algorithms.

Optimization Platforms

Modeling Environments	Solvers
MS Excel	Risk Solver
LINGO	LINDO
IBM ILOG	Cplex
AMPL	Gurobi
GAMS	CLP
MPL	GLPK
C++	
PuLP	

- See the following link for a recent (2013) survey of LP solvers: *https://prod.sandia.gov/techlib-noauth/access-control.cgi/2013/138847.pdf*

PuLP Overview (1/2)

- PuLP is a library for the Python scripting language that enables users to describe mathematical programs.
- PuLP works entirely within the syntax and natural idioms of the Python language by providing Python objects that represent optimization problems and decision variables, and allowing constraints to expressed in a way that is very similar to the original mathematical expression.
- PuLP has focused on supporting <u>linear</u> and <u>mixed-integer linear</u> models.
- PuLP can easily be deployed on any system that has a Python interpreter, as it has no dependencies on any other software packages.
- It supports a wide range of both commercial and open-source solvers, and can be easily extended to support additional solvers.

PuLP Overview (2/2)

- Free, Open Source, Portable
- Interfacing with Solvers
  - Base generic solver classes are included with PuLP in addition to specific interfaces to the currently popular solvers.
- Very close to the way optimization problems are formulated mathematically. This enables you to write optimization models very compactly.
- However, this requires careful formulation and practice in using the Python syntax and data structures. This can be frustrating for novices, especially those with little background in computer programming. **Perseverance pays off**.
- PuLP tutorial is available online at:
  - https://www.coin-or.org/PuLP/#pulp-internal-documentation

(https://projects.coin-or.org/PuLP/export/340/trunk/doc/pulp.pdf)

- PuLP source code is available at:
- https://github.com/coin-or/pulp/blob/master/src/pulp/pulp.py

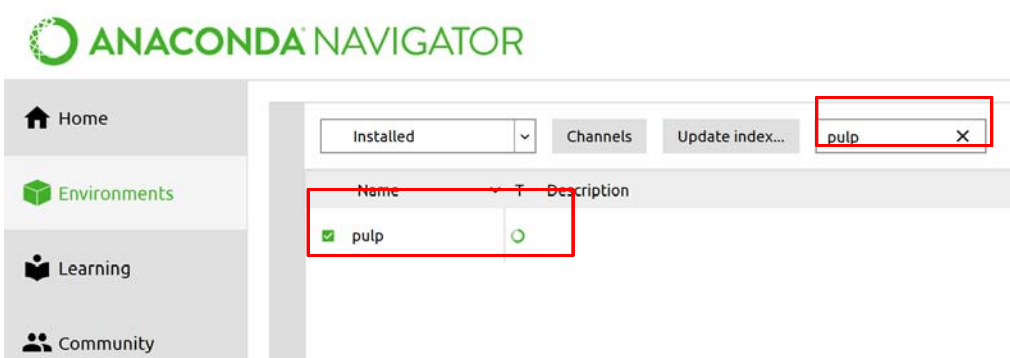
Installation: PuLP

To install on Anaconda:

- Mac OS:
  1. Open…
  2. Type: conda install -c conda-forge pulp
- Windows:
  1. Open **Anaconda Prompt** window
  2. Type: conda install -c conda-forge pulp

To check PuLP has been properly installed for Anaconda:

1. Go to Anaconda Navigator, Choose “Environments”
2. Type pulp in the box in the top right corner. If pulp shows up in the list, then you have PuLP installed currently.



Set Up BlueSky LP Formulation using PuLP

Define an LP Problem Using PuLP (1/3)

1. Import PuLP modeler functions

```
from pulp import *
```

2. Define an LP problem using **LpProblem**

```
probA=LpProblem("Problem A",LpMaximize)
```

- give the variable a name, for example **probA**
- use **LpProblem** class: **LpProblem(name='NoName', sense)**
  - **name**: name of the problem used in the output .lp file
  - **sense**: type of the LP problem objective. Either **LpMinimize** (default) or **LpMaximize**.
- **LpProblem** returns an LP problem

Define an LP Problem Using PuLP (2/3)

3. Define decision variables using **LpVariable**

```
xhc=LpVariable("xhc",lowBound=0,upBound=123,cat='Continuous')
xhm=LpVariable("xhm",lowBound=0,upBound=80,cat='Continuous')
xhp=LpVariable("xhp",lowBound=0,upBound=110,cat='Continuous')
xch=LpVariable("xch",lowBound=0,upBound=130,cat='Continuous')
xcm=LpVariable("xcm",lowBound=0,upBound=98,cat='Continuous')
xcp=LpVariable("xcp",lowBound=0,upBound=88,cat='Continuous')
xmh=LpVariable("xmh",lowBound=0,upBound=72,cat='Continuous')
xmc=LpVariable("xmc",lowBound=0,upBound=105,cat='Continuous')
xmp=LpVariable("xmp",lowBound=0,upBound=68,cat='Continuous')
xph=LpVariable("xph",lowBound=0,upBound=115,cat='Continuous')
xpc=LpVariable("xpc",lowBound=0,upBound=90,cat='Continuous')
xpm=LpVariable("xpm",lowBound=0,upBound=66,cat='Continuous')
```

- give each decision variable a name, e.g., **xhc** represents the number passengers to fly from Houston to Chicago
- use **LpVariable** class: **LpVariable(name, lowBound=None, upBound=None, cat='Continuous')**.

Parameters are explained below:

- **name**: The name of the variable used in the output .lp file
- **lowBound**: The lower bound on this variable's range. Default is negative infinity
- **upBound**: The upper bound on this variable's range. Default is positive infinity
- **cat**: The category this variable is in, *Integer, Binary or Continuous (default)*

Use PuLP to Define an LP Problem (3/3)

4. Add the objective function and constraints to the LP problem using **+=** operator

```
probA+=(197*xhc+110*xhm+125*xhp+190*xch+282*xcm+195*xcp+
        108*xmh+292*xmc+238*xmp+110*xph+192*xpc+230*xpm)
probA+=xhc+xmc+xpc<=240,"cHC"
probA+=xhm+xcm+xpm<=240,"cHM"
probA+=xhp+xcp+xmp<=240,"cHP"
probA+=xch+xcm+xcp<=240,"cCH"
probA+=xmh+xmc+xmp<=240,"cMH"
probA+=xph+xpc+xpm<=240,"cPH"
```

Use the name of the LP problem to display the LP formulation

```
probA
```

```
Problem A:
MAXIMIZE
190*xch + 282*xcm + 195*xcp + 197*xhc + 110*xhm + 125*xhp + 292*xmc + 108*xmh + 238*xmp + 192*xpc + 110*xph + 230*xpm
+ 0
SUBJECT TO
cHC: xhc + xmc + xpc <= 240

cHM: xcm + xhm + xpm <= 240

cHP: xcp + xhp + xmp <= 240

cCH: xch + xcm + xcp <= 240

cMH: xmc + xmh + xmp <= 240

cPH: xpc + xph + xpm <= 240

VARIABLES
xch <= 130 Continuous
xcm <= 98 Continuous
xcp <= 88 Continuous
xhc <= 123 Continuous
xhm <= 80 Continuous
xhp <= 110 Continuous
xmc <= 105 Continuous
xmh <= 72 Continuous
xmp <= 68 Continuous
xpc <= 90 Continuous
xph <= 115 Continuous
xpm <= 66 Continuous
```


Invoke Solver to Solve the LP Problem

5. Run solver

```
probA.writeLP("Bluesky1.lp")
probA.solve()
print("Status:",LpStatus[probA.status])
```

- use **name.writeLP("name.lp")** where name is the LP problem variable defined by LpProblem to store the LP problem formulation in a .lp file.
- use **name.solve(solver=None)**, where name is the LP problem variable defined by LpProblem to solve the given Lp problem.
  - This function changes the problem to make it suitable for solving and then calls the solver to find the solution.
  - solver – Optional: the specific solver to be used, defaults to the default solver.
- use **LpStatus[name.status]** to obtain the status of the LP solution.

Status of the LP Solution

- There are five status codes that can be returned from a solver in PuLP:

1	2	3	4	5
Optimal	Not Solved	Infeasible	Unbounded	Undefined

- **Optimal:** Optimal solution exists and is found.
- **Not Solved:** Is the default setting before a problem has been solved.
- **Infeasible:** The problem has no feasible solution.
- **Unbounded:** The objective function is unbounded.
- **Undefined:** Feasible solution hasn't been found (but may exist).

Print the Optimal Solution

```
for v in probA.variables():
    print(v.name, "=", v.varValue,"\tReduced Cost =", v.dj)
print("Total revenue=", value(probA.objective))
```

```
xch = 84.0       Reduced Cost = -0.0
xcm = 94.0       Reduced Cost = -0.0
xcp = 62.0       Reduced Cost = -0.0
xhc = 123.0      Reduced Cost = 5.0
xhm = 80.0       Reduced Cost = 18.0
xhp = 110.0      Reduced Cost = 120.0
xmc = 100.0      Reduced Cost = -0.0
xmh = 72.0       Reduced Cost = 8.0
xmp = 68.0       Reduced Cost = 133.0
xpc = 17.0       Reduced Cost = -0.0
xph = 115.0      Reduced Cost = 110.0
xpm = 66.0       Reduced Cost = 138.0
Total revenue= 185593.0
```

Print Shadow Prices

```
print("\nSensitivity Analysis\nConstraint\t\tShadow Price\tSlack")
for name, c in list(probA.constraints.items()):
    print(name, ":", c, "\t", c.pi, "\t\t", c.slack)
```

```
Sensitivity Analysis
Constraint              Shadow Price    Slack
cHC : xhc + xmc + xpc <= 240     192.0          -0.0
cHM : xcm + xhm + xpm <= 240     92.0           -0.0
cHP : xcp + xhp + xmp <= 240     5.0            -0.0
cCH : xch + xcm + xcp <= 240     190.0          -0.0
cMH : xmc + xmh + xmp <= 240     100.0          -0.0
cPH : xpc + xph + xpm <= 240     -0.0           42.0
```


BlueSky – First PuLP Model

```
# define the LP problem
probA=LpProblem("Problem A",LpMaximize)
# define decision variables
xhc=LpVariable("xhc",lowBound=0,upBound=123,cat='Continuous')
xhm=LpVariable("xhm",lowBound=0,upBound=80,cat='Continuous')
xhp=LpVariable("xhp",lowBound=0,upBound=110,cat='Continuous')
xch=LpVariable("xch",lowBound=0,upBound=130,cat='Continuous')
xcm=LpVariable("xcm",lowBound=0,upBound=98,cat='Continuous')
xcp=LpVariable("xcp",lowBound=0,upBound=88,cat='Continuous')
xmh=LpVariable("xmh",lowBound=0,upBound=72,cat='Continuous')
xmc=LpVariable("xmc",lowBound=0,upBound=105,cat='Continuous')
xmp=LpVariable("xmp",lowBound=0,upBound=68,cat='Continuous')
xph=LpVariable("xph",lowBound=0,upBound=115,cat='Continuous')
xpc=LpVariable("xpc",lowBound=0,upBound=90,cat='Continuous')
xpm=LpVariable("xpm",lowBound=0,upBound=66,cat='Continuous')
#define the objective function and constraints
probA+=(197*xhc+110*xhm+125*xhp+190*xch+282*xcm+195*xcp+
    108*xmh+292*xmc+238*xmp+110*xph+192*xpc+230*xpm)
probA+=xhc+xmc+xpc<=240,"cHC"
probA+=xhm+xcm+xpm<=240,"cHM"
probA+=xhp+xcp+xmp<=240,"cHP"
probA+=xch+xcm+xcp<=240,"cCH"
probA+=xmh+xmc+xmp<=240,"cMH"
probA+=xph+xpc+xpm<=240,"cPH"
```

PuLP

Construct LP

LpProblem()
- Define an LP problem variable, say probA

LpVariable()
- Define decision variables

lpSum() and +=
- Construct and add objective function and constraints

probA.solve()

Outputs of Solver

probA.status
- Status of the LP solution
- LpStatus(**probA.status**)

probA.variables()
- .name
- .varValue
- .dj *[reduced cost]*

probA.objective
- Obj function value
- value(probA.objective)

probA.constraints.items()
- .pi [shadow price]
- .slack [slack]

BlueSky – An Improved PulP Model (1/2)

Use data structures such as List, list of tuples, dictionary to build the LP model in a concise, scalable fashion.
"assignment2-tutorial-concise.ipynb"

```
# creates a list of all cities
City=['H','C','M','P']
# Creates a list of tuples containing all the possible routes for transport
Routes = [(fr, to) for fr in City for to in City]
# Creates a list of demand for each route
MaxDemand= [[0,123,80,110],\
            [130, 0, 98, 88],\
            [72, 105, 0, 68],\
            [115,90,66,0]]
# The demand data is made intot a dictionary
MaxDemand= makeDict([City,City],MaxDemand)

# Creates a list of fares
Fares = [#'Houston':\
    [0,197,110,125],\
    #'Chicago':
    [190, 0, 282, 195],\
    #'Miami':
    [108, 292, 0, 238],\
    #'Phoenix':
    [110,192,230,0]]
# The fares data is made intot a dictionary
Fares = makeDict([City,City],Fares,0)
Capacity = 240
passenger_vars = LpVariable.dicts("x", (City, City),lowBound=0,upBound=Capacity, cat='Continuous')
```

BlueSky – An Improved PuLP Model (2/2)

"assignment2-tutorial-concise.ipynb"

```
#objective function
probA+=lpSum([passenger_vars[fr][to]*Fares[fr][to] for (fr,to) in Routes])
# outbound capacity constraint
for i in City[1:]:
    probA += lpSum([passenger_vars[i][j] for j in City]) <= Capacity,"outbound_%s"%i

# inbound capacity constraint
for j in City[1:]:
    probA += lpSum([passenger_vars[i][j] for i in City]) <= Capacity,"inbound_%s"%j

# demand constraint
for i in City:
    for j in City:
        probA += passenger_vars[i][j] <= MaxDemand[i][j],"demand_%s to %s"%(i,j)
```

The **lpSum()** function: given a list of the form [a1*x1, a2*x2, …, an*xn] will construct a linear expression to be used as a constraint or variable

Solver Outcome

```
for v in probA.variables():
    print(v.name, "=", v.varValue,"\tReduced Cost =", v.dj)
print("Total revenue=", value(probA.objective))
```

```
x_C_C = 0.0      Reduced Cost = -382.0
x_C_H = 84.0     Reduced Cost = 0.0
x_C_M = 94.0     Reduced Cost = 0.0
x_C_P = 62.0     Reduced Cost = 0.0
x_H_C = 123.0    Reduced Cost = 0.0
x_H_H = 0.0      Reduced Cost = 0.0
x_H_M = 80.0     Reduced Cost = 0.0
x_H_P = 110.0    Reduced Cost = 0.0
x_M_C = 100.0    Reduced Cost = 0.0
x_M_H = 72.0     Reduced Cost = 0.0
x_M_M = 0.0      Reduced Cost = -192.0
x_M_P = 68.0     Reduced Cost = 0.0
x_P_C = 17.0     Reduced Cost = 0.0
x_P_H = 115.0    Reduced Cost = 0.0
x_P_M = 66.0     Reduced Cost = 0.0
x_P_P = 0.0      Reduced Cost = -5.0
Total revenue= 185593.0
```

Reduced costs are incorrect!

Solver Outcome

```
print("\nSensitivity Analysis\nConstraint\t\tShadow Price\tSlack")
for name, c in list(probA.constraints.items()):
    print(name, ":", c, "\t", c.pi, "\t\t", c.slack)
```

```
Sensitivity Analysis
Constraint                  Shadow Price     Slack
outbound_C : x_C_C + x_C_H + x_C_M + x_C_P <= 240         190.0            -0.0
outbound_M : x_M_C + x_M_H + x_M_M + x_M_P <= 240         100.0            -0.0
outbound_P : x_P_C + x_P_H + x_P_M + x_P_P <= 240         -0.0             42.0
inbound_C : x_C_C + x_H_C + x_M_C + x_P_C <= 240          192.0            -0.0
inbound_M : x_C_M + x_H_M + x_M_M + x_P_M <= 240          92.0             -0.0
inbound_P : x_C_P + x_H_P + x_M_P + x_P_P <= 240          5.0              -0.0
demand_H_to_H : x_H_H <= 0       -0.0            -0.0
demand_H_to_C : x_H_C <= 123     5.0             -0.0
demand_H_to_M : x_H_M <= 80      18.0            -0.0
demand_H_to_P : x_H_P <= 110     120.0           -0.0
demand_C_to_H : x_C_H <= 130     -0.0            46.0
demand_C_to_C : x_C_C <= 0       -0.0            -0.0
demand_C_to_M : x_C_M <= 98      -0.0            4.0
demand_C_to_P : x_C_P <= 88      -0.0            26.0
demand_M_to_H : x_M_H <= 72      8.0             -0.0
demand_M_to_C : x_M_C <= 105     -0.0            5.0
demand_M_to_M : x_M_M <= 0       -0.0            -0.0
demand_M_to_P : x_M_P <= 68      133.0           -0.0
demand_P_to_H : x_P_H <= 115     110.0           -0.0
demand_P_to_C : x_P_C <= 90      -0.0            73.0
demand_P_to_M : x_P_M <= 66      138.0           -0.0
demand_P_to_P : x_P_P <= 0       -0.0            -0.0
```

Shadow prices are correct.

BlueSky – Read Data from External Sources

"assignment2-tutorial-read-external-data.ipynb"

- Import pandas and numpy.
  - Make sure you have an updated version of pandas >= 0.23.0.
  - To update pandas, use the following command in Anaconda Prompt (Windows)/Terminal(Mac)

    conda install -c anaconda pandas

```
from pulp import *
import pandas as pd
import numpy as np
```

- Use pd.read_excel to read tables from excel sheets into dataframes.
- Prepare your Excel workbook such that each sheet contains exactly one table

```
df_rev=pd.read_excel('Data_BlueSkyAirlines_Network_py.xlsx',sheet_name='revenue')
df_demand=pd.read_excel('Data_BlueSkyAirlines_Network_py.xlsx',sheet_name='demand')
```

- Convert dataframes to lists

```
Fares = [np.array(df_rev.iloc[4+i,3:7],dtype=float) for i in range(4)]
MaxDemand =[np.array(df_demand.iloc[5+i,3:7],dtype=float) for i in range(4)]
```

BlueSky – Read Data from External Sources

```
# creates a list of all cities
City=['H','C','M','P']
# Creates a list of tuples containing all the possible routes for transport
Routes = [(fr, to) for fr in City for to in City]

MaxDemand= makeDict([City,City],MaxDemand)

Fares = makeDict([City,City],Fares,0)
Capacity = 240
passenger_vars = LpVariable.dicts("x",(City,City),lowBound=0,upBound=Capacity, cat='Continuous')
```


Solver Outcome

```
for v in probA.variables():
    print(v.name, "=", v.varValue,"\tReduced Cost =", v.dj)
print("Total revenue=", value(probA.objective))
```

```
x_C_C = 0.0     Reduced Cost = -382.0
x_C_H = 84.0    Reduced Cost = 0.0
x_C_M = 94.0    Reduced Cost = 0.0
x_C_P = 62.0    Reduced Cost = 0.0
x_H_C = 123.0   Reduced Cost = 0.0
x_H_H = 0.0     Reduced Cost = 0.0
x_H_M = 80.0    Reduced Cost = 0.0
x_H_P = 110.0   Reduced Cost = 0.0
x_M_C = 100.0   Reduced Cost = 0.0
x_M_H = 72.0    Reduced Cost = 0.0
x_M_M = 0.0     Reduced Cost = -192.0
x_M_P = 68.0    Reduced Cost = 0.0
x_P_C = 17.0    Reduced Cost = 0.0
x_P_H = 115.0   Reduced Cost = 0.0
x_P_M = 66.0    Reduced Cost = 0.0
x_P_P = 0.0     Reduced Cost = -5.0
Total revenue= 185593.0
```

Reduced costs are incorrect!

Solver Outcome

```
print("\nSensitivity Analysis\nConstraint\t\tShadow Price\tSlack")
for name, c in list(probA.constraints.items()):
    print(name, ":", c, "\t", c.pi, "\t\t", c.slack)
```

```
Sensitivity Analysis
Constraint              Shadow Price    Slack
outbound_C : x_C_C + x_C_H + x_C_M + x_C_P <= 240       190.0           -0.0
outbound_M : x_M_C + x_M_H + x_M_M + x_M_P <= 240       100.0           -0.0
outbound_P : x_P_C + x_P_H + x_P_M + x_P_P <= 240       -0.0            42.0
inbound_C : x_C_C + x_H_C + x_M_C + x_P_C <= 240        192.0           -0.0
inbound_M : x_C_M + x_H_M + x_M_M + x_P_M <= 240        92.0            -0.0
inbound_P : x_C_P + x_H_P + x_M_P + x_P_P <= 240        5.0             -0.0
demand_H_to_H : x_H_H <= 0       -0.0           -0.0
demand_H_to_C : x_H_C <= 123     5.0            -0.0
demand_H_to_M : x_H_M <= 80      18.0           -0.0
demand_H_to_P : x_H_P <= 110     120.0          -0.0
demand_C_to_H : x_C_H <= 130     -0.0           46.0
demand_C_to_C : x_C_C <= 0       -0.0           -0.0
demand_C_to_M : x_C_M <= 98      -0.0           4.0
demand_C_to_P : x_C_P <= 88      -0.0           26.0
demand_M_to_H : x_M_H <= 72      8.0            -0.0
demand_M_to_C : x_M_C <= 105     -0.0           5.0
demand_M_to_M : x_M_M <= 0       -0.0           -0.0
demand_M_to_P : x_M_P <= 68      133.0          -0.0
demand_P_to_H : x_P_H <= 115     110.0          -0.0
demand_P_to_C : x_P_C <= 90      -0.0           73.0
demand_P_to_M : x_P_M <= 66      138.0          -0.0
demand_P_to_P : x_P_P <= 0       -0.0           -0.0
```

Shadow prices are correct.

Exercise

- Implement the Blending problem in Session 4 using PuLP
  - *Variables:*

    x_{ij} = the amount of input i to use to blend output j, i=S, I, j= C, V, EV

  - *Formulation:*

maximize $10(x_{SC}+x_{IC})+12(x_{SV}+x_{IV})+15(x_{S,EV}+x_{I,EV})$
$-6.5(x_{SC}+x_{SV}+x_{S,EV})-5.75(x_{IC}+x_{IV}+x_{I,EV})$

subject to

(Supply capacity) $x_{SC}+x_{SV}+x_{S,EV}\le 3000$ [Spanish oil]
$x_{IC}+x_{IV}+x_{I,EV}\le 3000$ [Italian oil]

(Demand) $x_{SC}+x_{IC}\le 700$ [Commercial]
$x_{SV}+x_{IV}\le 2200$ [Virgin]
$x_{S,EV}+x_{I,EV}\le 1400$ [Extra Virgin]

(Blending percentage requirements)
$x_{IC}\le 0.35(x_{SC}+x_{IC})$
$x_{S,EV}\ge 0.55(x_{S,EV}+x_{I,EV})$

(Non-negativity) $x_{ij}\ge 0$