# Solving Linear Programming Problems in Python

GLPK IN PYTHON

# GLPK with Python



- There are several Python language bindings to choose from. Each provides a differing level of abstraction. All are open source software. Some them are listed below:
  - PyGLPK
  - PyMathProg (we are going to use it)
  - Pyomo
  - Python-GLPK
  - PuLP
  - CVXOPT
  - Sage
  - ecyglpki

# PyMathProg



- PyMathProg provides an easy and flexible modelling syntax using Python to create and optimize mathematical programming models.
- It is kind of a reincarnation of AMPL and GNU MathProg in Python.
- Great features offered by PyMathProg include:
  - Ergonomic syntax for modelling
  - Friendly interactive session
  - Sensitivity report
  - Advanced solver options
  - Automatic model update on parameter changes
  - Parameters sharable between models
  - Deleting variables/constraints
  - Supporting both Python 2 and 3
  - Supporting all major platforms

# PyMathProg (Installation)



- Assuming you already have Python 2 or Python 3 installed, now open a terminal window (also known as a command window), and type in this line of command and hit return:
  - pip install pymprog

# PyMathProg (Exp1)



- We will solve this tiny LP model here using PyMathProg.

```
maximize  15 x + 10 y        # profit
S.T.
          x <=  3            # mountain bike limit
          y  <=  4           # racer limit
          x + y  <=  5       # frame limit
          x >=0, y >=0       # non-negative
```

# PyMathProg (Exp1-Code)



```
begin('bike production')
  verbose(True)
  x, y = var('x, y')  # variables
  maximize(15 * x + 10 * y, 'profit')
  x <= 3  # mountain bike limit
  y <= 4  # racer production limit
  x + y <= 5  # metal finishing limit
  solve()
  print("###>Objective value: %f" % vobj())
  sensitivity()
  end()  # Good habit: do away with the model
```

# PyMathProg (Exp1-Output)



Max profit: 15 * x + 10 * y

R1: x + y <= 5

GLPK Simplex Optimizer 5.0

1 row, 2 columns, 2 non-zeros

*  0: obj =  -0.000000000e+00 inf =  0.000e+00 (2)

*  2: obj =  6.500000000e+01 inf =  0.000e+00 (0)

OPTIMAL LP SOLUTION FOUND

###>Objective value: 65.000000

# PyMathProg (Exp1-Output)



PyMathProg 1.0 Sensitivity Report Created: 2022/11/25 Fri 17:44PM

==========

| Variable | Activity | Dual.Value | Obj.Coef | Range.From | Range.Till |
|---|---|---|---|---|---|
| x | 3 | 5 | 15 | 10 | inf |
| *y | 2 | 0 | 10 | 0 | 15 |

==========

Note: rows marked with a * list a basic variable.

==========

| Constraint | Activity | Dual.Value | Lower.Bnd | Upper.Bnd | RangeLower | RangeUpper |
|---|---|---|---|---|---|---|
| R1 | 5 | 10 | -inf | 5 | 3 | 7 |

==========

# PyMathProg (Exp1-Output)



Note: normally, RangeLower is the min for the binding bound, and RangeUpper gives the max value. However, when neither bounds are binding, the row is marked with a *, and RangeLower is the max for Lower.Bnd(whose min is -inf), and RangeUpper is the min for Upper.Bnd(whose max value is inf). Then the columns of RangeLower, RangeUpper and Activity all have identical values.

__del__ is deleting problem: bike production

# PyMathProg



- We can also do many other interesing things to a model, even after solving it, for example:
  - Conduct sensitivity analysis
  - Change the value of a parameter
  - Fix a variable to an arbitrary value
  - Manage the bounds of a variable or constraint
  - Change the type of a variable
  - Adding/deleting variables or constraints

# Define Variables in PyMathProg



- The routine var(...) is the only tool to create variables. Yet there are quite a few different ways to do so, depending on the modelling situation.
  - x, y = var('x, y') # many names -> many vars
  - z = var('z', 3) # an array of 3 variables
  - v = var('v', kind=bool) # 0/1 variable
  - w = var('w', bounds=(0,5)) # specify the bounds
  - colors = ('red', 'green', 'blue') # index set → clr = var('color', colors, bool) # using an index set
  - clr # a dictionary with keys from the index set
  - help(var) # obtain help on this function

# PyMathProg (Exp2)



- This example is employed to show the use of the ‘primal’ and ‘dual’ values. A zero-sum two-player game is a game between two players, where the gain of one player is the loss of the other, hence their pay-offs always sums up to zero. The value a[i,j] is the pay-off for player one when player one plays strategy i and player two plays strategy j. Here is an LP formulation to find the equilibrium mixed strategies and the value of the game.

# PyMathProg (Exp2)



Solve this 2-player zero-sum game:

Gain for player 1

(Loss for player 2)

| | Player 2 | |
|---|---|---|
| Player 1 | B1 | B2 |
| A1 | 5 | 9 |
| A2 | 8 | 6 |

# PyMathProg (Exp2-Code)



```
begin('game')
# gain of player 1, a free variable
v = var('game_value', bounds=(None,None))
# mixed strategy of player 2
p = var('p', 2)
# probability sums to 1
sum(p) == 1
# player 2 chooses p to minimize v
minimize(v)
# player 1 chooses the better value
r1 =  v >= 5*p[0] + 9*p[1]
r2 =  v >= 8*p[0] + 6*p[1]
solve()
print('Game value: %g'% v.primal)
print("Mixed Strategy for player 1:")
print("A1: %g, A2: %g"%(r1.dual, r2.dual))
print("Mixed Strategy for player 2:")
print("B1: %g, B2: %g"%(p[0].primal, p[1].primal))
end()
```

In this block of code, two variables r1 and r2 are employed to save the constraints for the sake of reporting. Note that in this model, the primal value of the variables gives the probability for player 2's mixed strategy, and the dual value of the constraints r1 and r2 gives the mixed strategy of player 1.

# PyMathProg (Exp2-Output)



GLPK Simplex Optimizer, v4.60

3 rows, 3 columns, 8 non-zeros

0: obj = 0.000000000e+00 inf = 1.000e+00 (1)

3: obj = 7.000000000e+00 inf = 0.000e+00 (0)

OPTIMAL LP SOLUTION FOUND

Game value: 7

Mixed Strategy for player 1:

A1: 0.333333, A2: 0.666667

Mixed Strategy for player 2:

B1: 0.5, B2: 0.5

# Thanks for your time

## Lets See The Code

PyMathProg-Documentation
https://www.youtube.com/watch?v=1SJgYPRHZpQ