Lecture 3

Computational methods.

- Introduction to Jupyter lab/notebook (in UBC Syzygy server), python, and PuLp package.
- The simple examples with PuLp.
- More examples (Blending problem) with Python PuLp package.

UBC Syzygy server.

https://ubc.syzygy.ca/

You can log-in with your UBC CWL id.

syzygy.ca is a project of the Pacific Institute for the Mathematical Sciences, Compute Canada and Cybera to bring JupyterLab to researchers, educators and innovators across Canada.

See the tutorial: https://intro.syzygy.ca

You can instead use this address https://pims.syzygy.ca/ and login with a Google account.
It is a different account, so, your files at UBC account do not show up here, and vice versa.

Jupyter Lab/Notebook.

* Jupyter notebook (https://jupyter.org/) is a user friendly way of writing codes in python programming.

Python. (We use Python 3.)

* For Python language, you can look at a nice introduction by Prof. Patrick Walls at https://www.math.ubc.ca/~pwalls/math-python/

Basics:
How to navigate and run the code.

- Markdown: notes that are not run in the code. Latex symbols work for writing math expressions.
- Code

PULP

* Pulp is an open source linear programming package for python.
We will consider simple but helpful example of using Python/Jupyter Notebook and PuLP:

* For general introduction to PuLp you can look at
https://coin-or.github.io/pulp/main/index.html

[Please note that you DO NOT need to install python or PuLp in your computer. Everything can be done in the UBC Syzygy website using your CWL.]
In this webpage there are several good examples:
see Case Studies https://coin-or.github.io/pulp/CaseStudies/index.html in the left menu of the webpage.

An example

I have copied most of the following from https://github.com/benalexkeen/Introduction-to-linear-programming/blob/master/Introduction%20to%20Linear%20Programming%20with%20Python%20-%20Part%202.ipynb

the LP problem.

Maximize $3x + 5y + 5$ ← objective function

Subject to

$2x + 3y \leq 12$

$-x + y <= 3$

$x \leq 4$

$y \leq 3$

$2y \leq 25 - x$

$4y \geq 2x - 8$

$y \leq 2x - 5$

and $x, y \geq 0$

x, y decision variables

constraints

We import pulp, in the following two cells.

[1]:
```
import sys
!{sys.executable} -m pip install pulp
```

```
Requirement already satisfied: pulp in /opt/conda/lib/python3.8/site-packages (2.3)
Requirement already satisfied: amply>=0.1.2 in /opt/conda/lib/python3.8/site-packages (from pulp) (0.1.2)
Requirement already satisfied: docutils>=0.3 in /opt/conda/lib/python3.8/site-packages (from amply>=0.1.2->pulp) (0.15.2)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.8/site-packages (from amply>=0.1.2->pulp) (2.4.7)
```

[2]:
```
import pulp
```

To use the PuLP package, do the previous steps first befor proceeding. After installing the PuLP, we can set up our problem to solve. First, we define it.

Run these two lines before the actual coding with 'pulp'

[3]:
```
# Create a LP Minimization problem
Lp_prob = pulp.LpProblem('Your_LP_Problem', pulp.LpMaximize)  # We set up theproblem using the command LpProblem in the PuLP package.
```

This is a pulp package usage. Here # makes the righthand side a commment, not running as a code.

pulp.LpProblme <-- pulp is the package, `pulp.LpProblem` means we are using the command `LpProblem` in the pulp pakcage.

For minimization problem, use `pulp.LpMinimize`.

Here `Your_LP_Problem` is the name of the problem which shows up when we display the problem. We used `_` as spaces are not permitted in the name.

[4]:
```
# Create problem Decision Variables
x = pulp.LpVariable("x")   # Create a variable x >= 0.  "x" means we put `x' when printing this variable.
y = pulp.LpVariable("y")   # Create a variable y >= 0
```

We used the LpVariable class.

Lower and Upper bounds can be assigned using the 'lowBound' and 'upBound' parameter instead. For example,

x = pulp.LpVariable("x", lowBound = 0) # Create a variable x >= 0

y = pulp.LpVariable("y", upBound = 10) # Create a variable y <= 10

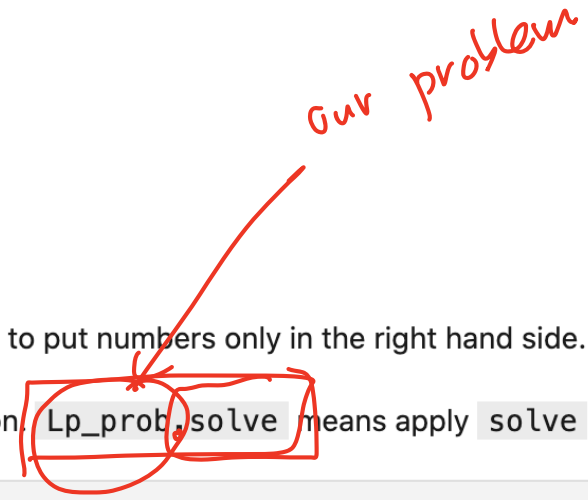
We now set up our LP problem.

[5]:
```
# Objective Function
Lp_prob += 3 * x + 5 * y +5

# We put objective function first then constraints.

# Constraints:
Lp_prob += 2 * x + 3 * y <= 12
Lp_prob += -x + y <= 3
Lp_prob += x <= 4
Lp_prob += y <= 3
Lp_prob += 2 * y <= 25 - x
Lp_prob += 4 * y >= 2 * x - 8
Lp_prob += y <= 2 * x - 5
Lp_prob += x>= 0
Lp_prob += y>= 0
```

The objective function and constraints are added using the += operator to our model. The objective function is added first, then the individual constraints.

[6]:
```
# Display the problem
print(Lp_prob)
```

```
Your_LP_Problem:
MAXIMIZE
3*x + 5*y + 5
SUBJECT TO
_C1: 2 x + 3 y <= 12

_C2: - x + y <= 3

_C3: x <= 4

_C4: y <= 3

_C5: x + 2 y <= 25

_C6: - 2 x + 4 y >= -8

_C7: - 2 x + y <= -5

_C8: x >= 0

_C9: y >= 0

VARIABLES
x free Continuous
y free Continuous
```



You realize that the inequalities are rearranged to put numbers only in the right hand side.

We can solve this LP using the `solve` function. `Lp_prob.solve` means apply `solve` to the `Lp_prob` we defined.

[7]:
```
Lp_prob.solve()
pulp.LpStatus[Lp_prob.status]
```

[7]:
```
'Optimal'
```

It solved the LP problem and gave the result: There are 5 status codes:

- Not Solved: Status prior to solving the problem.
- Optimal: An optimal solution has been found.
- Infeasible: There are no feasible solutions.
- Unbounded: The constraints are not bounded, maximising the solution will tend towards infinity.
- Undefined: The optimal solution may exist but may not have been found.

We can now view our optimal variable values and the optimal value of Z.

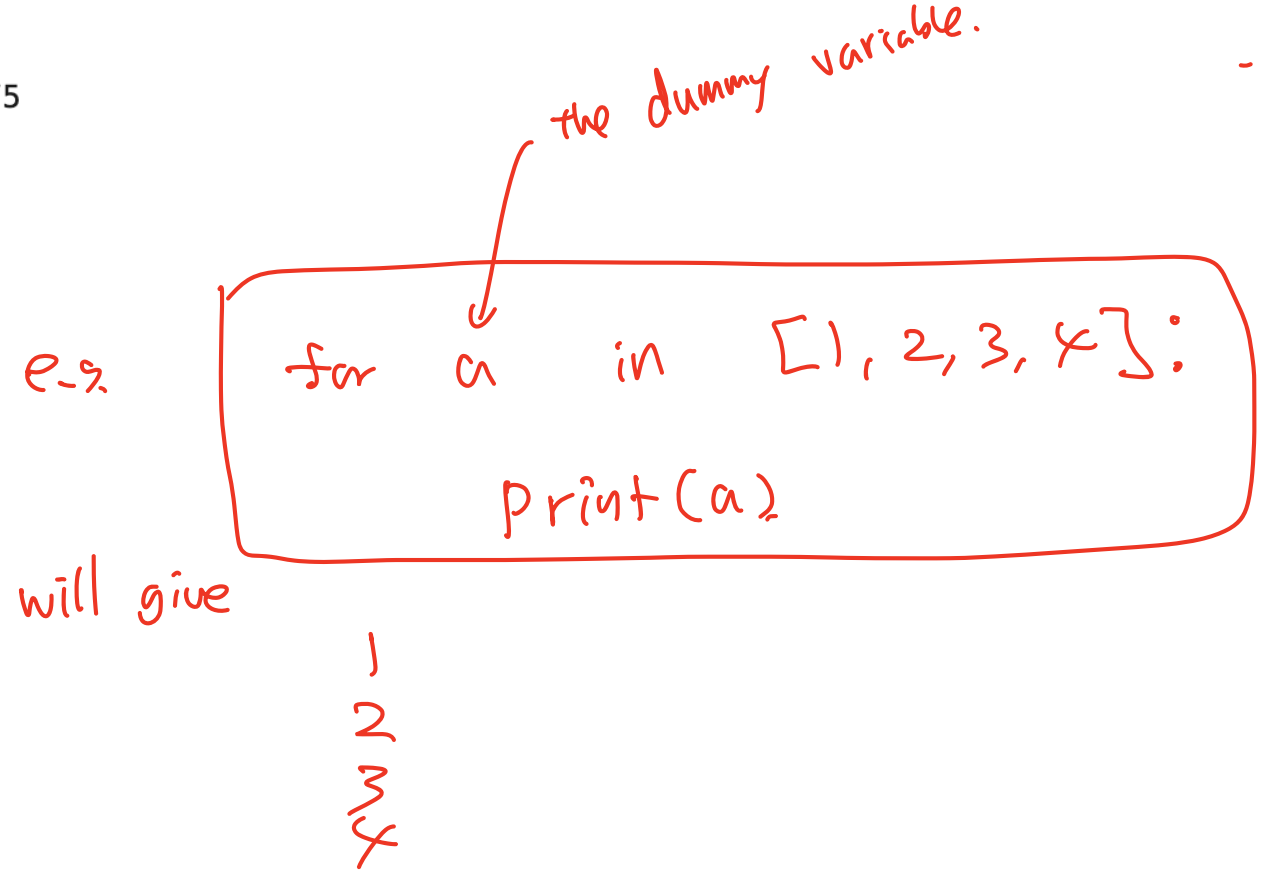
8]:
```
# Printing the final solution
print("x=", pulp.value(x), "y=", pulp.value(y), "z=", pulp.value(Lp_prob.objective))
```

```
x= 3.375 y= 1.75 z= 23.875
```

Another way to show the solutions:

9]:
```
for variable in Lp_prob.variables():
        print(variable.name, "=", variable.varValue)
print("Optimal value is z = ", pulp.value(Lp_prob.objective))
```

```
x = 3.375
y = 1.75
Optimal value is z =  23.875
```

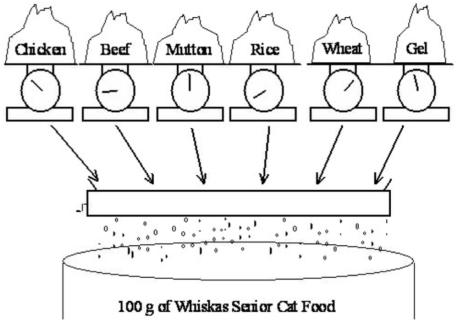


A Motivating example: Blending problem from

- https://coin-or.github.io/pulp/CaseStudies/a_blending_problem.html#problem-description

A Blending Problem

Problem Description



Nutrients per 100g.
Protein at least 8g
Fat at least 6g
Fibre at most 2g
Salt at most 0.4g

Whiskas cat food, shown above, is manufactured by Uncle Ben's. Uncle Ben's want to produce their cat food products as cheaply as possible while ensuring they meet the stated nutritional analysis requirements shown on the cans. Thus they want to vary the quantities of each ingredient used (the main ingredients being chicken, beef, mutton, rice, wheat and gel) while still meeting their nutritional standards.

The costs of the chicken, beef, and mutton are \$0.013, \$0.008 and \$0.010 respectively, while the costs of the rice, wheat and gel are \$0.002, \$0.005 and \$0.001 respectively. (All costs are per gram.) For this exercise we will ignore the vitamin and mineral ingredients. (Any costs for these are likely to be very small anyway.)

Each ingredient contributes to the total weight of protein, fat, fibre and salt in the final product. The contributions (in grams) per gram of ingredient are given in the table below.

Stuff	Protein	Fat	Fibre	Salt
Chicken	0.100	0.080	0.001	0.002
Beef	0.200	0.100	0.005	0.005
Mutton	0.150	0.110	0.003	0.007
Rice	0.000	0.010	0.100	0.002
Wheat bran	0.040	0.010	0.150	0.008
Gel	0.000	0.000	0.000	0.000

Blending problem

Copied from
https://coin-or.github.io/pulp/CaseStudies/a_blending_problem.html#problem-description

Blending problem.

Copied from https://coin-or.github.io/pulp/CaseStudies/a_blending_problem.html#problem-description

The_Whiskas_Problem:

MINIMIZE

0.018* BEEF + 0.3* CHICKEN + 0.011* GEL + 0.01*MUTTON + 0.02*RICE + 0.05*WHEAT

SUBJECT TO

PercentagesSum: BEEF + CHICKEN + GEL + MUTTON + RICE + WHEAT = 100

ProteinRequirement: 0.2 BEEF + 0.1 Ingr_CHICKEN + 0.15 MUTTON + 0.04 WHEAT >= 8

FatRequirement: 0.1 BEEF + 0.08 Ingr_CHICKEN + 0.11 MUTTON + 0.01 RICE + 0.01 WHEAT >= 6

FibreRequirement: 0.005 BEEF + 0.001 CHICKEN + 0.003 MUTTON + 0.1 RICE + 0.15 WHEAT <= 2

SaltRequirement: 0.005 BEEF + 0.002 CHICKEN + 0.007 MUTTON + 0.002 RICE + 0.008 WHEAT <= 0.4

Nonnegativity: BEEF, CHICKEN, MUTTON, RICE, WHEAT >= 0

Steps for installing PuLP

[1]:
```
import sys
!{sys.executable} -m pip install pulp
```

```
Requirement already satisfied: pulp in /opt/conda/lib/python3.8/site-packages (2.3)
Requirement already satisfied: amply>=0.1.2 in /opt/conda/lib/python3.8/site-packages (from pulp) (0.1.2)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.8/site-packages (from amply>=0.1.2->pulp) (2.4.7)
Requirement already satisfied: docutils>=0.3 in /opt/conda/lib/python3.8/site-packages (from amply>=0.1.2->pulp) (0.15.2)
```

[2]:
```
import pulp
```

[3]:
```
"""
The Full Whiskas Model Python Formulation for the PuLP Modeller

Authors: Antony Phillips, Dr Stuart Mitchell  2007
"""

# Import PuLP modeler functions
from pulp import *    # Here because of * we will not put `pulp' before each pulp command; e.g. instead of pulp.LpVariable, we simply write LpVariable.
```

Steps for Decision Variables.

[4]:
```
# Creates a list of the Ingredients
Ingredients = ['CHICKEN', 'BEEF', 'MUTTON', 'RICE', 'WHEAT', 'GEL']   # This gives the names for the indexes in the vector

# A dictionary of the costs of each of the Ingredients is created. They give vector values.
costs = {'CHICKEN': 0.30, #originally 0.013
         'BEEF': 0.018, #originally 0.008
         'MUTTON': 0.010, #originally 0.010
         'RICE': 0.02, #originally 0.002
         'WHEAT': 0.05, #originally 0.005
         'GEL': 0.011}  #originally 0.001

# A dictionary of the protein percent in each of the Ingredients is created
proteinPercent = {'CHICKEN': 0.100,
                  'BEEF': 0.200,
                  'MUTTON': 0.150,
                  'RICE': 0.000,
                  'WHEAT': 0.040,
                  'GEL': 0.000}

# A dictionary of the fat percent in each of the Ingredients is created
fatPercent = {'CHICKEN': 0.080,
              'BEEF': 0.100,
              'MUTTON': 0.110,
              'RICE': 0.010,
              'WHEAT': 0.010,
              'GEL': 0.000}

# A dictionary of the fibre percent in each of the Ingredients is created
fibrePercent = {'CHICKEN': 0.001,
                'BEEF': 0.005,
                'MUTTON': 0.003,
                'RICE': 0.100,
                'WHEAT': 0.150,
                'GEL': 0.000}

# A dictionary of the salt percent in each of the Ingredients is created
saltPercent = {'CHICKEN': 0.002,
               'BEEF': 0.005,
               'MUTTON': 0.007,
               'RICE': 0.002,
               'WHEAT': 0.008,
               'GEL': 0.000}
```

[5]:
```
# Create the 'prob' variable to contain the problem data
prob = LpProblem("The_Whiskas_Problem", LpMinimize)
```

[6]:
```
# A dictionary called 'ingredient_vars' is created to contain the referenced Variables
ingredient_vars = LpVariable.dicts("Ingr", Ingredients, 0)    # Here the last value '0' gives the lower bound for the variable.
# Here "Ingr" is what appears when we print its name; e.g. Ingr_Beef. In the code, `ingredient_vars' is the name in the code.
# We use the `dicts' command to use the previously given dictionary `Ingredient'.
```

Comment. They are ignored.

Code name of the LP problem

decision variables

lpSum([list]) adds all items in the list.

This creates a list
Note the square bracket was used.

Objective function

[7]:
```
# The objective function is added to 'prob' first
prob += lpSum([costs[i]*ingredient_vars[i] for i in Ingredients]), "Total Cost of Ingredients per can"
# Here "Total Cost of Ingredients per can" gives an explanation comment. Do not forget to put the comma `,` before it.
```

dummy variable

Ingredients = ['CHICKEN', 'BEEF', ...]

Constraints

[8]:
```
# The five constraints are added to 'prob'
prob += lpSum([ingredient_vars[i] for i in Ingredients]) == 100, "PercentagesSum"
prob += lpSum([proteinPercent[i] * ingredient_vars[i] for i in Ingredients]) >= 8.0, "ProteinRequirement"
prob += lpSum([fatPercent[i] * ingredient_vars[i] for i in Ingredients]) >= 6.0, "FatRequirement"
prob += lpSum([fibrePercent[i] * ingredient_vars[i] for i in Ingredients]) <= 2.0, "FibreRequirement"
prob += lpSum([saltPercent[i] * ingredient_vars[i] for i in Ingredients]) <= 0.4, "SaltRequirement"
```

Notice that we did not add the condition that the ingridients are >=0, as it was given in ingredient_vars = LpVariable.dicts("Ingr",Ingredients, 0) by adding the `0`.

If we did not add `0` there, we can instead add the contraints in the code as

```
for i in Ingredients:
    prob += ingredient_vars[i] >= 0
```

Show the LP problem.

[9]:
```
# You can write the problem to an .lp file
prob.writeLP("WhiskasModel.lp")
```

[9]:
```
[Ingr_BEEF, Ingr_CHICKEN, Ingr_GEL, Ingr_MUTTON, Ingr_RICE, Ingr_WHEAT]
```

[10]:
```
# Or you can directly display the problem here.

print(prob)
```

```
The_Whiskas_Problem:
MINIMIZE
0.018*Ingr_BEEF + 0.3*Ingr_CHICKEN + 0.011*Ingr_GEL + 0.01*Ingr_MUTTON + 0.02*Ingr_RICE + 0.05*Ingr_WHEAT + 0.0
SUBJECT TO
PercentagesSum: Ingr_BEEF + Ingr_CHICKEN + Ingr_GEL + Ingr_MUTTON + Ingr_RICE
 + Ingr_WHEAT = 100

ProteinRequirement: 0.2 Ingr_BEEF + 0.1 Ingr_CHICKEN + 0.15 Ingr_MUTTON
 + 0.04 Ingr_WHEAT >= 8

FatRequirement: 0.1 Ingr_BEEF + 0.08 Ingr_CHICKEN + 0.11 Ingr_MUTTON
 + 0.01 Ingr_RICE + 0.01 Ingr_WHEAT >= 6

FibreRequirement: 0.005 Ingr_BEEF + 0.001 Ingr_CHICKEN + 0.003 Ingr_MUTTON
 + 0.1 Ingr_RICE + 0.15 Ingr_WHEAT <= 2

SaltRequirement: 0.005 Ingr_BEEF + 0.002 Ingr_CHICKEN + 0.007 Ingr_MUTTON
 + 0.002 Ingr_RICE + 0.008 Ingr_WHEAT <= 0.4

VARIABLES
Ingr_BEEF Continuous
Ingr_CHICKEN Continuous
Ingr_GEL Continuous
Ingr_MUTTON Continuous
Ingr_RICE Continuous
Ingr_WHEAT Continuous
```

Notice that the lower bound >=0 for the variable is not shown, as it is the default condition. If you had changed the lowerbound to something else, then it will show up here.

Solve the LP.

[11]:
```
# The problem is solved using PuLP's choice of Solver
prob.solve()
# The status of the solution is printed to the screen
print("Status:", LpStatus[prob.status])
```

```
Status: Optimal
```

Show the values for the optimal solution

[12]:
```
# Each of the variables is printed with it's resolved optimum value
for a in prob.variables():
    print(a.name, "=", a.varValue)
```

```
Ingr_BEEF = 0.0
Ingr_CHICKEN = 0.0
Ingr_GEL = 42.857143
Ingr_MUTTON = 57.142857
Ingr_RICE = 0.0
Ingr_WHEAT = 0.0
```

⟵ This order for prob.variables() is alphabetical and may NOT be the same as the order of ingredients_vars.

Show the optimal value.

[13]:
```
print("Total Cost of Ingredients per can = ", value(prob.objective))
```

```
Total Cost of Ingredients per can =  1.042857143
```