MSCI 332 Tutorial 1

This tutorial will be a hands-on, so don't forget to bring your laptops!

Colab notebook link (https://colab.research.google.com/drive/1F6bcQipE0uBD1sRqG4V48rW-rxrT2zjg?usp=sharing).

Setting up the programming environment

There are three main options available:

- Using Google Colab: no installation requirement, a google account is necessary, probably has a poorer performance compared to running code in your computer, unavailable offline.
- Using Jupyter (https://jupyter.org/install): more or less the same layout with Colab, needs to be installed, available offline.
- Using other IDE's (PyCharm (https://www.jetbrains.com/pycharm/), VS Code (https://code.visualstudio.com/docs/languages/python), Spyder (https://www.spyder-ide.org/), others (https://www.google.com/search?q=python+ide+list&oq=python+ide+list)): better suited coding environment, useful coding and debugging tools, some can also work with notebooks.

Tutorials will utilize Google Colab, so you should decide and setup your preferred programming environment before the first tutorial. This link (https://support.gurobi.com/hc/en-us/articles/360044290292-How-do-I-install-Gurobi-for-Python-) details the installation process for gurobi in python. In case you run into licensing issues, follow this link (https://www.gurobi.com/academia/academic-program-and-licenses/).

Setting up Gurobi on Google Colab

It's as easy as running the command:

```
In [ ]: !pip install gurobipy>=9.5.1
        import gurobipy as gp
        from gurobipy import GRB as GRB
```

Adding a license key

If your model exceeds the variable limit in the unlicensed version (more than 2000 variables and 2000 constraints), you can apply your academic license with the following command:

In [ ]:
```python
# Create environment with WLS license
e = gp.Env(empty=True)
e.setParam('WLSACCESSID', 'your wls accessid (string)')
e.setParam('WLSSECRET', 'your wls secret (string)')
e.setParam('LICENSEID', <your license id (integer)>)
e.start()

# Create the model within the Gurobi environment
model = gp.Model(env=e)
```

```
Set parameter WLSAccessID
Set parameter WLSSecret
Set parameter LicenseID
Academic license - for non-commercial use only - registered to sturhan@uwater
loo.ca
```

You can follow this link (https://support.gurobi.com/hc/en-us/articles/4409582394769-Google-Colab-Installation-and-Licensing) ('Full Gurobi License using WLS' section) to see how to get a license and locate these credentials. If you are using a local IDE such as Jupyter, Spyder or Pycharm to view this file, the license you enter with *grbgetkey* should still be valid.

Example Model

Farmer Ash has two farms that grow wheat and corn. Because of differing soil conditions, there are differences in yield and costs of growing crops on two farms. The yield and cost data is shown on the table below.

	Farm 1	Farm 1
Corn yield/acre (bushels)	50	40
Cost/acre of corn ($)	100	120
Wheat yield/acre (bushels)	65	35
Cost/acre of wheat ($)	90	80
Cultivation Area (acres)	180	200

Farm 1 has 180 acres and farm 2 has 200 acres available for cultivation. Based on the recently received orders, 11,000 bushels of wheat and 7,000 bushels of corn must be grown. Determine a planting plan to help Ash to minimize the cost while satisfying the order amounts.

Formulate the given problem as a linear optimization problem. Define the variables you use explicitly.

x_{ij}: total area (acres) dedicated for plant type $i \in \{1,2\}$, in farm $j \in \{1,2\}$

$$\begin{aligned} \min \quad & 100x_{11} + 120x_{12} + 90x_{21} + 80x_{22} \\ \text{s.t.} \quad & 50x_{11} + 40x_{12} \geq 7000 \\ & 65x_{21} + 35x_{22} \geq 11000 \\ & x_{11} + x_{21} \leq 180 \\ & x_{21} + x_{22} \leq 200 \\ & 0 \leq x_{ij} & \forall i \in \{1,2\}, j \in \{1,2\} \end{aligned}$$

In [ ]:

```
# on google colab and with license
# model = gp.Model("model_name_without_space", env=e)
# local notebook or google colab without license
model = gp.Model("model_name_without_space")
# decision variables
x = model.addVars(2, 2, lb=0.0, vtype=GRB.CONTINUOUS, name="x")
# setting the objective
model.setObjective(100 * x[0, 0] + 120 * x[0, 1] + 90 * x[1, 0] + 80 * x[1, 1
],
                   sense=GRB.MINIMIZE)
# adding the constraints
model.addConstr(50 * x[0, 0] + 40 * x[0, 1] >= 7e3, name="corn_requirement")
model.addConstr(65 * x[1, 0] + 35 * x[1, 1] >= 11e3, name="wheat_requirement")
model.addConstr(x[0, 0] + x[1, 0] <= 180, name="farm1_area_limit")
model.addConstr(x[1, 0] + x[1, 1] <= 200, name="farm2_area_limit")
# solving the model
model.optimize()
# printing out the optimal variables
for (i, j), variable in x.items():
  print(f"x {i} {j} = {variable.X}")
```

```
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 4 rows, 4 columns and 8 nonzeros
Model fingerprint: 0x74268c72
Coefficient statistics:
  Matrix range     [1e+00, 6e+01]
  Objective range  [8e+01, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 1e+04]
Presolve time: 0.01s
Presolved: 4 rows, 4 columns, 8 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.250000e+03   0.000000e+00      0s
       3    3.5692308e+04   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.02 seconds (0.00 work units)
Optimal objective  3.569230769e+04
x 0 0 = 10.769230769230774
x 0 1 = 161.53846153846152
x 1 0 = 169.23076923076923
x 1 1 = 0.0
```

It's possible to make output much more readable by utilizing custom printing functions:

```
In [ ]: for (i, j), variable in x.items():
          print(f"Area dedicated for {'corn' if i == 0 else 'wheat'} in farm {j} is "
                f"{variable.X: .2f} acres")
```

```
Area dedicated for corn in farm 0 is  10.77 acres
Area dedicated for corn in farm 1 is  161.54 acres
Area dedicated for wheat in farm 0 is  169.23 acres
Area dedicated for wheat in farm 1 is  0.00 acres
```

A complex model can get difficult to debug just by looking at the code, we can export the model in any one of the supported formats by calling model.write() function:

```
In [ ]: model.write("model.lp")   # usually the easiest to read
        model.write("model.mps")  # oldest version
        model.write("model.rew")  # similar to mps
        # printing lp output (in case you are browsing this file offline)
        with open("model.lp", 'r') as f: print("\n".join(f.readlines()))
```

```
\ Model model_name_without_space

\ LP format - for model browsing. Use MPS format to capture full model detai
l.

Minimize

  100 x[0,0] + 120 x[0,1] + 90 x[1,0] + 80 x[1,1]

Subject To

 corn_requirement: 50 x[0,0] + 40 x[0,1] >= 7000

 wheat_requirement: 65 x[1,0] + 35 x[1,1] >= 11000

 farm1_area_limit: x[0,0] + x[1,0] <= 180

 farm2_area_limit: x[1,0] + x[1,1] <= 200

Bounds

End
```

You can also read a generated model file and re-construct the model:

In [ ]:
```python
import re  # for reading variables

model2 = gp.read("model.lp")
model2.optimize()
# reading an existing variable is a bit tricky, we can only differentiate
# between variables by their names in the provided file
print("Printing variable values in optimal solution")
x2 = {}  # it'll be set individually

for variable in model2.getVars():
  print(f"{variable.VarName} = {variable.X}")

  # extracting the variable information from name, you can just use this regex
  # to read any variable in format: variable_name[index1,index2]

  m = re.findall("(.*?)\[(\d*?),(\d*?)\]", variable.VarName)[0]

  variable_name = m[0] # useful when you have multiple variables
  index1 = int(m[1])   # each match is still a string object
  index2 = int(m[2])

  x2[index1, index2] = variable

# we can use the same print function:
for (i, j), variable in x2.items():
  print(f"Area dedicated for {'corn' if i == 0 else 'wheat'} in farm {j} is "
        f"{variable.X: .2f} acres")

# disposing the model
model2.dispose()
```

```
Read LP format model from file model.lp
Reading time = 0.00 seconds
: 4 rows, 4 columns, 8 nonzeros
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 4 rows, 4 columns and 8 nonzeros
Model fingerprint: 0x74268c72
Coefficient statistics:
  Matrix range     [1e+00, 6e+01]
  Objective range  [8e+01, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 1e+04]
Presolve time: 0.01s
Presolved: 4 rows, 4 columns, 8 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.250000e+03   0.000000e+00      0s
       3    3.5692308e+04   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.02 seconds (0.00 work units)
Optimal objective  3.569230769e+04
Printing variable values in optimal solution
x[0,0] = 10.769230769230774
x[0,1] = 161.53846153846152
x[1,0] = 169.23076923076923
x[1,1] = 0.0
Area dedicated for corn in farm 0 is  10.77 acres
Area dedicated for corn in farm 1 is  161.54 acres
Area dedicated for wheat in farm 0 is  169.23 acres
Area dedicated for wheat in farm 1 is  0.00 acres
```

Errors in Python Notebooks

Since python is a dynamic language, it works really well with the arbitrary, cell-based input of the notebooks. However, this functionality can make it easier to encounter errors or unintended behaviours, especially with models. So here are some suggestions to avoid such situations:

- Always declare the entire model in one cell: gp.Model() call, variables and constraints.
- Avoid using one variable name multiple times when implementing multiple models in the same notebook.
- Use an underscore instead of spaces when naming model variables ('name' attribute) to ensure that models are exported correctly.
- Restart runtime when memory usage is high, unless you are closing models with model.dispose().