IDS 435 - Optimization via Gurobi

Spring 2022

Table of Contents

Installing Gurobi

We follow the Gurobi installation guidelines on this page.

1. Run the following lines in your anaconda terminal to add Gurobi repository and install it:

   - `conda config --add channels https://conda.anaconda.org/gurobi`
   - `conda install gurobi`
2. Set up academic license:

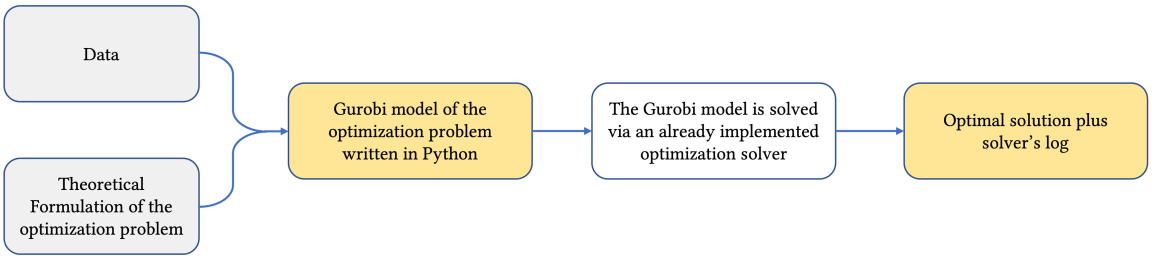
   - Sign up at the Gurobi website using your UIC email.
   - Log into your account.
   - Visit Academic License Registration page.
   - Accept terms and conditions.
   - Copy and paste the code generated by Gurobi (i.e., `grbgetkey 11111111-1111-1111-111-1111111111`) into your operating system's terminal and run it.
   - Your academic license should be set up!
3. Create a Jupyter notebook and test if you can run the code below in your notebook or not. If Gurobi is installed properly, then following code must run without any error.

In [ ]:
```
""" Testing Gurobi Installation"""

import gurobipy as gb
model = gb.Model('Test')
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2022-05-07
```

A Procedure to Model an Optimization Problem in Gurobi

The main task to use Gurobi in practice is to convert the mathematical formulation of an optimization problem into a Gurobi model that can be solved efficiently.



The Procedure

1. Sets and indices

   Use python `list`, `range`, `arrange`, etc to define index sets to count over decision variables and constraints.

2. Parameters (i.e., data)

   Data should be formatted in a way that is easy to define decision variables, objective function, and constraints in Gurobi. Data can be represented in different formats such as

   - Python lists
   - Python dictionaries
   - Numpy array
   - Pandas `dataframe`
   - Gurobi multidict

   Since Numpy arrays are widely-used data structures in python, we employ them to represent data that later on used in a Gurobi model

3. Decision variables

   - Specify type of the variable (real-valued, binary, integer, etc)

   - Specify if the variable is signed or not (positive or negative)

   - Try to define put related decision variables of the model in an array of variables instead of defining them individually

   ```
   # For loop
   for i in range (N):
   ```

```python
        model.addVar( ... )

# One line definition
model.addMvar( shape=N)
```

4. Constraints

    - Specify type of a constraint (linear, quadratic, etc)

    - Find an appropriate Gurobi function to model the constraint (oftentimes you can use `addConstrs`)

    - Try to define multiple constraints in a single line of code via python inline for loop

    ```python
    # For loop
    for i in range (N):
            model.addConstr( ... )

     # Inline for loop (preferred)
    model.addConstrs( ... for i in range (N))
    ```

5. Objective function

    - Define the objective function using data and decision variables
    - Specify if the problem is maximization or minimization

6. Optimize

    - Choose a solver
    - Specify parameters of the solver (i.e., stopping criteria, feasibility tolerance, etc)
    - Solve the Gurobi model

7. Analyze results (*Gurobi solved the model*)

    - Is model well-conditioned (i.e., no numerical issues encountered while optimization)?
    - Is the model "normalized"?
    - What is an optimal solution?
    - What is the optimal value?
    - How long did it take to solve the model?

8. Troubleshooting (*Gurobi could not solved the model*)

    - Is the issue with the numerical errors?
    - Is the issue with solver? Try a different optimization solver. Try to change the parameters of the solver (i.e., feasibility tolerance)?
    - Double-check the type of variables and their signs as well as the definition of constraints Gurobi model?

A Toy Example

We use the following optimization problem to illustrate using Gurobi and aforementioned procedure for using Gurobi:

$$\min_{x_1, x_2} \quad -x_1 - x_2$$
$$x_1 + 2x_2 \le 1$$
$$2x_1 + x_2 \le 1$$
$$x_1, x_2 \ge 0$$

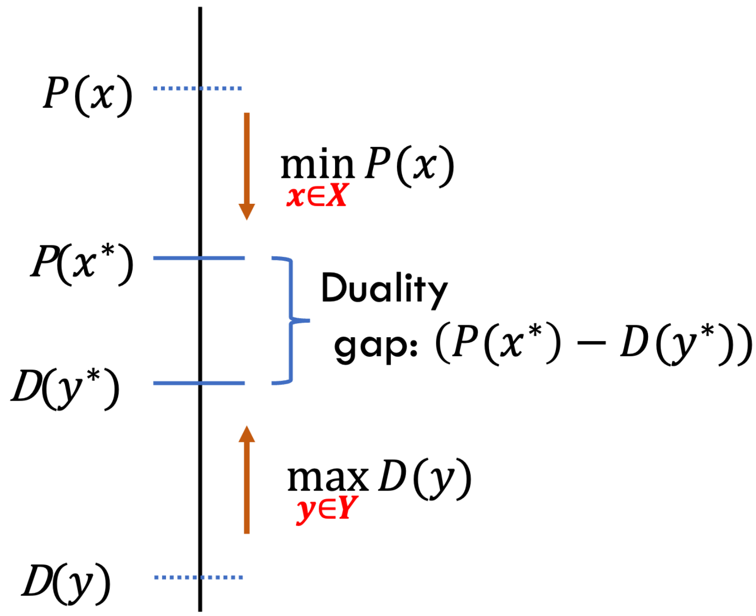
In [ ]:

```
import gurobipy as gb
import numpy as np

if __name__ == "__main__":

    """         Step 1. Sets and indices            """

    """         Step 2. Parameters                  """

    """         Step 3. Decision variables          """

    model   = gb.Model('Toy Example')
    x_1     = model.addVar(
                        name     = 'x_1',
                        vtype    = gb.GRB.CONTINUOUS,
                        lb       = 0,
                        ub       = gb.GRB.INFINITY)

    x_2     = model.addVar(
                        name     = 'x_2',
                        vtype    = gb.GRB.CONTINUOUS,
                        lb       = 0,
                        ub       =  gb.GRB.INFINITY )

    """         Step 4. Constraints                 """
    model.addConstr(x_1 + 2*x_2  <= 1)
    model.addConstr(2*x_1 + x_2  <= 1)

    """         Step 5. Objective function          """
    model.setObjective(-x_1 - x_2)

    """         Step 6. Optimize                    """
    print('='*100)
    model.setParam('Method',2)
    model.setParam('Crossover',0)
    model.update()
    model.optimize()
    print('='*100)

    """         Step 7. Analyze results         """
    print('The optimal x_1:         \t',        x_1.X)
    print('The optimal x_2:         \t',        x_2.X)
```

```python
    print('The optimal value is:    \t',        model.ObjVal)
    print('='*100)
```

```
====================================================================================================
Set parameter Method to value 2
Set parameter Crossover to value 0
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
Optimize a model with 2 rows, 2 columns and 4 nonzeros
Model fingerprint: 0xed33d8fe
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros
Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 1.000e+00
 Factor NZ  : 3.000e+00
 Factor Ops : 5.000e+00 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -8.67927042e-01 -4.61538462e-01  1.51e-01 3.08e-01  2.86e-01     0s
   1  -6.05231787e-01 -6.96010401e-01  0.00e+00 0.00e+00  2.27e-02     0s
   2  -6.66536989e-01 -6.66799107e-01  0.00e+00 0.00e+00  6.55e-05     0s
   3  -6.66666537e-01 -6.66666799e-01  0.00e+00 0.00e+00  6.55e-08     0s
   4  -6.66666667e-01 -6.66666667e-01  0.00e+00 2.22e-16  6.55e-11     0s

Barrier solved model in 4 iterations and 0.01 seconds (0.00 work units)
Optimal objective -6.66666667e-01

====================================================================================================
The optimal x_1:                 0.3333333332685205
The optimal x_2:                 0.3333333332685205
The optimal value is:            -0.666666666537041
====================================================================================================
```

What is happening?



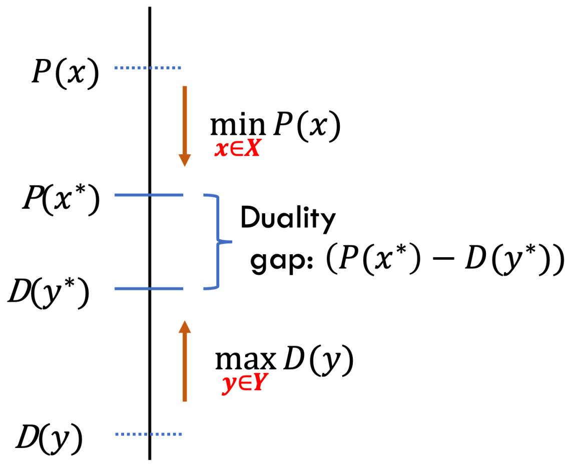
A Finer Implementation

In [ ]:
```
import gurobipy as gb
import numpy as np

if __name__ == "__main__":

    """          Step 1. Sets and indices            """
    num_var             = 2
    num_constr          = 2
    var_index           = range(num_var)
    constr_index        = range(num_constr)

    """          Step 2. Parameters                  """
    constr_matrix       = np.array([[1.,2.],
                                    [2.,1.]])
    rhs                 = np.array([1.,1.])


    """          Step 3. Decision variables          """

    model              = gb.Model('Toy Example')
    x                  = model.addMVar(
                                      shape     = num_var,
                                      name      = 'x',
                                      vtype     = gb.GRB.CONTINUOUS,
                                      lb        = 0.,
                                      ub        = gb.GRB.INFINITY )

    """          Step 4. Constraints                 """
    model.addConstrs(gb.quicksum(constr_matrix[i][j]*x[j]  for j in var_inde

    """          Step 5. Objective function          """
    model.setObjective(-gb.quicksum(x))

    """          Step 6. Optimize                    """
```

```
    print('='*100)
    model.setParam('Method',2)
    model.setParam('Crossover',0)
    model.update()
    model.optimize()
    print('='*100)

    """         Step 7. Analyze results             """
    print('The optimal x_1:         \t',        x[0].X)
    print('The optimal x_2:         \t',        x[1].X)
    print('The optimal value is:    \t',        model.ObjVal)
    print('='*100)
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2022-05-07
==============================================================================
========================
Set parameter Method to value 2
Set parameter Crossover to value 0
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 10 physical cores, 10 logical processors, using up to 10 threa
ds
Optimize a model with 2 rows, 2 columns and 4 nonzeros
Model fingerprint: 0xed33d8fe
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros
Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 1.000e+00
 Factor NZ  : 3.000e+00
 Factor Ops : 5.000e+00 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -8.67927042e-01 -4.61538462e-01  1.51e-01 3.08e-01  2.86e-01     0s
   1  -6.05231787e-01 -6.96010401e-01  0.00e+00 0.00e+00  2.27e-02     0s
   2  -6.66536989e-01 -6.66799107e-01  0.00e+00 0.00e+00  6.55e-05     0s
   3  -6.66666537e-01 -6.66666799e-01  0.00e+00 0.00e+00  6.55e-08     0s
   4  -6.66666667e-01 -6.66666667e-01  0.00e+00 2.22e-16  6.55e-11     0s

Barrier solved model in 4 iterations and 0.01 seconds (0.00 work units)
Optimal objective -6.66666667e-01

==============================================================================
========================
The optimal x_1:                 0.3333333332685205
The optimal x_2:                 0.3333333332685205
The optimal value is:            -0.666666666537041
==============================================================================
========================
```

What is happening?



Marketing Campaign Optimization (Direct Approach)

Determine what products to offer to each customer in a way that maximizes the marketing campaign return on investment (ROI) while considering the following business rules:

- limits on funding available for the campaign
- restrictions on the minimum number of product offers that can be made in a campaign
- campaign return-on-investment hurdle rates that must be met

In the direct approach that we learned in class, we formulate an integer program to assign products to customers that satisfy the business rules and maximizes profit. This integer program is given by

$$
\begin{aligned}
\max_{x,z} \quad & \sum_{j \in J} \sum_{i \in I} r_{j,i} x_{j,i} - Mz & & \\
& \sum_{j \in J} \sum_{i \in I} c_{j,i} x_{j,i} & \leq B + z, & \\
& \sum_{j \in J} \sum_{i \in I} r_{j,i} x_{j,i} & \geq (1+R) \sum_{j \in J} \sum_{i \in I} c_{j,i} x_{j,i}, & \\
& \sum_{i \in I} x_{j,i} & \geq Q_j, & \quad \forall j \in J, \\
& x_{j,i} \in \{0,1\}, & & \quad \forall i \in I, \forall j \in J, \\
& z \geq 0. & &
\end{aligned}
$$

In [ ]:
```
import gurobipy as gb
import numpy as np
if __name__ == "__main__":

    """           Step 1. Sets and indices            """
    num_customer          = 10
```

```python
num_prod              = 2
customer_index        = range(num_customer)
product_index         = range(num_prod)

"""         Step 2. Parameters                  """
budget                = 200            # (B) Budget for marketing campaign
offer_lb              = 2              # (Q_j) Minimum number of offers for
hurdle_rate           = 0.20           # (R) ROI hurdle rate of 20%
budget_inc_cost       = 11              # (M) Increasing the budget by $1 c
profit                = np.array([
                          # c1    c2      c3      c4      c5      c6
                          [2050,  1950,   2000,   2100,   1900,   3000,
                          [1050,  950,    1000,   1100,   900,    2000,
                          ])

cost                  = np.array([
                          # c1    c2      c3      c4      c5      c6
                          [205,   195,    200,    210,    190,    300,
                          [105,   95,     100,    110,    90,     200,
                          ])


"""         Step 3. Decision variables          """
model           = gb.Model('Marketing')

# Assignment of products to customers
x               = model.addMVar(    shape       = (num_prod,num_customer
                                    vtype       = gb.GRB.BINARY )

# Amount by which budget is increased
z               =  model.addVar(    vtype       = gb.GRB.CONTINUOUS,
                                    lb          = 0.0,
                                    ub          = gb.GRB.INFINITY )

"""         Step 4. Constraints             """
realized_profit = gb.quicksum(profit[j,i]*x[j,i]    for i in customer_ir
realized_cost   = gb.quicksum(cost[j,i]*x[j,i]      for i in customer_ir


# Cost equals B plus any increase in budget
model.addConstr( realized_cost <= budget + z)


# Campaign profit should exceed cost by the hurdle rate
model.addConstr( realized_profit >= (1. + hurdle_rate)*realized_cost)

# Product p should be offered to least offer_lb customers
model.addConstrs(gb.quicksum(x[j,i] for i in customer_index) >= offer_lb
                      for j in product_index)

"""         Step 5. Objective function      """
model.setObjective(realized_profit - budget_inc_cost*z, gb.GRB.MAXIMIZE)

"""         Step 6. Optimize                """
print('='*100)
model.update()
```

```
    model.optimize()
    print('='*100)

    """         Step 7. Analyze results         """

    print('Optimal campaign:')
    print('{:^25}{:^25}{:^25}'.format(' ','Product 1','Product 2'))
    for i in customer_index:
        print('{:^25}{:^25.0f}{:^25.0f}'.format('Customer '+str(i),x[0,i].X,

    print('\nOptimal amount by which budget is increased: ',    z.X)
    print('The optimal profit value:                    ',      sum(profit[j
                                                                for i in cus
    print('The optimal objective value is:              ',      model.ObjVal
    print('='*100)
```

```
================================================================================
========================
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 10 physical cores, 10 logical processors, using up to 10 threa
ds
Optimize a model with 4 rows, 21 columns and 61 nonzeros
Model fingerprint: 0xb9b69412
Variable types: 1 continuous, 20 integer (20 binary)
Coefficient statistics:
  Matrix range     [1e+00, 3e+03]
  Objective range  [1e+01, 3e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 2e+02]
Found heuristic solution: objective -1800.000000
Presolve removed 1 rows and 0 columns
Presolve time: 0.00s
Presolved: 3 rows, 21 columns, 41 nonzeros
Variable types: 0 continuous, 21 integer (20 binary)

Root relaxation: objective 1.630000e+03, 4 iterations, 0.00 seconds (0.00 wo
rk units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0    1630.0000000 1630.00000  0.00%     -    0s

Explored 1 nodes (4 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 10 (of 10 available processors)

Solution count 2: 1630 -1800

Optimal solution found (tolerance 1.00e-04)
Best objective 1.630000000000e+03, best bound 1.630000000000e+03, gap 0.000
0%
================================================================================
========================
Optimal campaign:
                                 Product 1                Product 2
       Customer 0                   -0                       -0
       Customer 1                    1                        1
       Customer 2                    0                        0
       Customer 3                   -0                       -0
       Customer 4                    1                        1
       Customer 5                   -0                       -0
       Customer 6                   -0                       -0
       Customer 7                   -0                       -0
       Customer 8                   -0                       -0
       Customer 9                   -0                       -0

Optimal amount by which budget is increased:  370.0
The optimal profit value:                     5700.0
The optimal objective value is:               1630.0
================================================================================
========================
```