Gerald Jones, Kimon Swanson

Home Work 2: Mississippi Agricultural Co

ISE522 Spg 22

Problem Description:

> ***Mississippi Agricultural Co. owns a wheat warehouse with a capacity of 20,000 bushels. At the beginning of month 1, they have 6,000 bushels of wheat. Each month, wheat can be bought and sold at the price per 1000 bushels given in the table below. The sequence of events during each month is as follows: i) The initial stock of wheat is counted. ii) Any amount of wheat up to your initial stock can be sold at the current month's selling price. iii) The company can buy (at the current month's buying price) as much wheat as they want, subject to the warehouse size limitation. Do the following:***
>
> - Formulate an LP that can be used to determine how to maximize the profit earned over the next 10 months. Make sure that your write out your model and don't just give a spreadsheet. Please provide a brief explanation of each constraint (if you you software like AMPL, giving a descriptive name for the constraint is enough).
> - Solve your LP using a solver of your choice.

Model Formulation

Parameters:

> ***C*** capacity of facility
>
> h_1 amount of stock in thousands of bushels for the first month
>
> ***M*** total number of months to forcast profits for

Variables:

> m month such that m={1, ..., M}
>
> b_m amount of bushels to buy at month m in thousands of bushels
>
> s_m amount of bushels to sell at month m in thousands of bushels
>
> h_m amount of bushels on hand in warehouse at the beginning of month m in thousands of bushels
>
> l_m cost per thousand bushels for month m (dollars/bushels) (LOSS)
>
> g_m profit per thousand bushels for month m (dollars/bushels) (GAIN)
>
> P_M total profit for the M month data

Constraints:

> bushels at start of month m must equal what is left at end of last month

$$h_m \quad \leq b_{m-1} - s_{m-1} + x_{m-1}$$

> can only sell what is on stack at the current month m

$$s_m \leq h_m$$

> what is left at the end of the month m must not exceed the capacity

$$b_m + h_m - s_m \leq C$$

Objective:

> Goal: maximize profit over the course of the next ten months based on the cost, and profit data eacn month by deciding how much to buy and sell for each of the ten months:

$$maximize(P_M = \sum_{i=1}^{M} s_m * g_m - b_m * l_m)$$

Import the neccessary modules and data file.

In [1]:

```python
import gurobipy as gp
from gurobipy import GRB
import pandas as pd

data_file = 'hw2.xlsx'
data_df = pd.read_excel("../_DATA/" + data_file)
```

Verify The Data is what is required

> M:= month
>
> P:= profit for a sale in given month
>
> C:= cost of purchaseing bushels in given month

In [2]:

```python
display(data_df)
```

	M	P	C
0	1	3	8
1	2	6	8
2	3	7	2
3	4	1	3
4	5	4	4
5	6	5	3
6	7	5	3
7	8	1	2
8	9	3	5
9	10	2	5

Optimize profit:

Decide the amount of bushels sold & bought for the next 10 months

In [3]:

```python
try:


    # Create a new model
    m = gp.Model("Mississippi Agricultural Co.")

    # set the capacity
    capacity_limit = 20 #"k"

###################################################### Set up some functions ###############################
############################################################################################################

    # Used to generate lists of variables for the model
    # buy value in 1k for month m such that bm = the amount to buy in month m
    def generate_n_vars(model, count, vtype=GRB.CONTINUOUS, base_name="x", lb=0):
        """
            Generate list of count variables of given vtype each named base_name<Index>
            model:= gurobi generated model
            count:= the number of variables you want to generate
            vtype: the types of the variables generated which should be an GRB.vtype object
            base_name:= string used to name the variables of the form base_name<index>
            lb:= lower bound for variables
        """
        return [m.addVar(vtype=vtype, name=base_name + str(i+1), lb=lb) for i in range(count)]


    # used to generate the objective function by generating an expression that is the sum
    # of the profits and costs for each month
    def sum_vars(bs, ss, df, cost="C", profit="P",):
        """
            Generate an expression of the form:
                sum from m = 1 to M (sell(month)*proffit(month) - buy(month)*cost(month)) for all months to ca
                the profit at the end of M months
            arguments:
            bs:= list of variables already added to the model representing the amount to buy in month s
            ss:= list of variables already added to the model representing the amount to sell in month s
            df:= dataframe containing cost and profit information where the row corresponding to
                 index i contains data for month i+1 such that the indices are zero indexed
            cost:= column name for the passed data frame df that contains the cost of buying at month/index m
            profit:= column name for the passed data frame df that contains the profit of selling at month/ind
            returns: an expression of the form described above that can be passed to the model
        """
        cnt = 0
        for idx in df.index:
            if cnt == 0:
                expr = ss[int(idx)] * df.loc[idx, profit]  - bs[int(idx)] * df.loc[idx, cost]
                cnt += 1
            else:
                expr += ss[int(idx)] * df.loc[idx, profit]  - bs[int(idx)] * df.loc[idx, cost]
        return expr


    # used to generate the step based constraints described in the problem description
    def add_sequential_constraints(model, bs, ss, xs, initial_stock, capacity):
        """
            creates constraints that mimic the process described as follows:
            1) check the amount on hand for month m held in variable xs[m] this
               represents the amount of stock in the warehouse at the beginning of month m.
               This is determined by the amount on hand during the previous month, the amount sold the previou
               and the amount bought during the previous month:
                                              xs[m] = xs[m-1] - ss[m-1] + bs[m-1]
            2) during month m a you can only sell what was available in the ware house in month m:
                                                         ss[m] <= xs[m]
            3) At the end of the the month the total amount of stock in the warehouse meaning the amount bough
               plus the amount on hand, minus the amount sold for month m can not exceed the capacity for the 
                                          bs[m] + xs[m] - ss[m] <= capacity/limit_value
        """

        for i in range(len(bs)):
            # check the amount on hand in the warehouse for this month
            # given information that the initial month of analysis
            # contains initial amount of stock
            if i == 0:
                model.addConstr(xs[i] == initial_stock)
            else:
                model.addConstr(xs[i] == bs[i-1] + xs[i-1] - ss[i-1])

            # add sell quantity constraint
            model.addConstr(ss[i] <= xs[i])

            # add capacity at end of month constraint
            model.addConstr(bs[i] + xs[i] - ss[i] <= capacity)

    # just print out the values of variables after optimization
    def basic_results_display(m):
        # display results/solution
        cnt = 0
        for v in m.getVars():
            try:
                print('{} {}'.format(v.VarName, v.X))
            except Exception as e:
                print(e)
                print('{},'.format(v.VarName))
            if "10" in v.VarName:
                print()
            cnt +=1
        print("-------------------------------------\n\n")


    # display the solution with the amount on hand, sold, and bought each month
    # along with the current expected profit
    def display_timestep_decisions(m, sell="Sell", buy="Buy", onhand="OnHand"):
        month_events = {}
        monthly_numbers = {}
        profits = 0
        for v in m.getVars():
            if "0" in v.Varname:
                mnth = int(v.VarName[-2:])
            else:
                mnth = int(v.VarName[-1])

            if mnth not in month_events:
                month_events[mnth] = {}
                monthly_numbers[mnth] = 0
            if "Sell" in v.VarName:
                month_events[mnth]["Sell"] = "Sell: " + str(v.X) + " at " + str(data_df.loc[mnth-1, "P"])
                p = data_df.loc[mnth-1, "P"]
                amnt = float(v.X)
                monthly_numbers[mnth] += amnt * p
            elif "Buy" in v.VarName:
                month_events[mnth]["Buy"] = "Buy: " + str(v.X) + " at " + str(data_df.loc[mnth-1, "C"])
                p = data_df.loc[mnth-1, "C"]
                amnt = float(v.X)
                monthly_numbers[mnth] += amnt * p *(-1)
            elif "OnHand" in v.VarName:
                month_events[mnth]["OnHand"] = "In Stock: " + str(v.X) +","

        for month in month_events:
            profits += monthly_numbers[month]
            display_string = "Month: {}\n\t".format(month) + month_events[month]["OnHand"] + " " + month_event
            print(display_string + ", Profits: {}".format(profits))
############################################################################################################
############################################################################################################



    # Add variables for the amount to buy (B), sell (S), and on hand (X) for each month
    Bs = generate_n_vars(m, 10, vtype=GRB.CONTINUOUS, base_name="Buy", lb=0)
    Ss = generate_n_vars(m, 10, vtype=GRB.CONTINUOUS, base_name="Sell", lb=0)
    Xs = generate_n_vars(m, 10, vtype=GRB.CONTINUOUS, base_name="OnHand", lb=0)

    # generate the objective expression and assign it
    objective_expression = sum_vars(Bs, Ss, data_df)
    m.setObjective(objective_expression, GRB.MAXIMIZE)

# set the constraints based on the following sequential process
#     1) check the amount on hand for month m held in variable xs[m] this
#          represents the amount of stock in the warehouse at the beginning of month m.
#          This is determined by the amount on hand during the previous month, the amount sold the previous mo
#          and the amount bought during the previous month:
#                               xs[m] = xs[m-1] - ss[m-1] + bs[m-1]
#    2) during month m a you can only sell what was available in the ware house in month m:
#                                        ss[m] <= xs[m]
#    3) At the end of the the month the total amount of stock in the warehouse meaning the amount bought
#       plus the amount on hand, minus the amount sold for month m can not exceed the capacity for the
#       ware house
#                        bs[m] + xs[m] - ss[m] <= capacity/limit_value
    add_sequential_constraints(m, Bs, Ss, Xs, initial_stock=6,  capacity=20)

    # find optimum  solution
    m.optimize()


    # display the results
    basic_results_display(m)
    display_timestep_decisions(m, sell="Sell", buy="Buy", onhand="OnHand")

    print("\n-------------Does it make sense?----------------------")
    print('Obj-Profit after 10 months: ${:,.2f}'.format(m.ObjVal*1000))





# catch some math errors
except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

```
Restricted license - for non-production use only - expires 2023-10-25
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (win64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 30 rows, 30 columns and 87 nonzeros
Model fingerprint: 0xc25f5352
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 8e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+00, 2e+01]
Presolve removed 16 rows and 13 columns
Presolve time: 0.00s
Presolved: 14 rows, 17 columns, 61 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    9.6000000e+31   8.000000e+30   9.600000e+01      0s
      11    1.6200000e+02   0.000000e+00   0.000000e+00      0s

Solved in 11 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.620000000e+02
Buy1 0.0
Buy2 0.0
Buy3 20.0
Buy4 0.0
Buy5 20.0
Buy6 20.0
Buy7 0.0
Buy8 20.0
Buy9 0.0
Buy10 0.0

Sell1 0.0
Sell2 0.0
Sell3 6.0
Sell4 0.0
Sell5 20.0
Sell6 20.0
Sell7 20.0
Sell8 0.0
Sell9 20.0
Sell10 0.0

OnHand1 6.0
OnHand2 6.0
OnHand3 6.0
OnHand4 20.0
OnHand5 20.0
OnHand6 20.0
OnHand7 20.0
OnHand8 0.0
OnHand9 20.0
OnHand10 0.0

-------------------------------------



Month: 1
        In Stock: 6.0, Sell: 0.0 at 3, Buy: 0.0 at 8, Profits: 0.0
Month: 2
        In Stock: 6.0, Sell: 0.0 at 6, Buy: 0.0 at 8, Profits: 0.0
Month: 3
        In Stock: 6.0, Sell: 6.0 at 7, Buy: 20.0 at 2, Profits: 2.0
Month: 4
        In Stock: 20.0, Sell: 0.0 at 1, Buy: 0.0 at 3, Profits: 2.0
Month: 5
        In Stock: 20.0, Sell: 20.0 at 4, Buy: 20.0 at 4, Profits: 2.0
Month: 6
        In Stock: 20.0, Sell: 20.0 at 5, Buy: 20.0 at 3, Profits: 42.0
Month: 7
        In Stock: 20.0, Sell: 20.0 at 5, Buy: 0.0 at 3, Profits: 142.0
Month: 8
        In Stock: 0.0, Sell: 0.0 at 1, Buy: 20.0 at 2, Profits: 102.0
Month: 9
        In Stock: 20.0, Sell: 20.0 at 3, Buy: 0.0 at 5, Profits: 162.0
Month: 10
        In Stock: 0.0, Sell: 0.0 at 2, Buy: 0.0 at 5, Profits: 162.0

-------------Does it make sense?----------------------
Obj-Profit after 10 months: $162,000.00
```

For testing lets look at the result of just buying and selling the max at each step

In [4]:

```python
obj = 0;
# test solution

sell_buy = [
    [6, 20],        #1
    [20, 20],        #2
    [20, 20],        #3
    [],        #4
    [],        #5
    [],        #6
    [],        #7
    [],        #8
    [],        #9
    [],        #10
]
for i in range(1, 11):
    if i == 1:
        obj += -20*data_df.loc[i-1, "C"] + 6*data_df.loc[i-1, "P"]
    else:
        obj += -20*data_df.loc[i-1, "C"] + 20*data_df.loc[i-1, "P"]
```

Resources:

- latex in notebooks
- gurobi variable attributes
- converting python notebooks to pdf