III. Model documentation and write-up

You can respond to these questions either in an e-mail or as an attached file (any common document format is acceptable such as plain text, PDF, DOCX, etc.) **Please number your responses.**

1. Who are you (mini-bio) and what do you do professionally?

Hoa Nguyen Phuong *received her B.Sc. degree from FPT University in Vietnam with the major on Computer Science in 2017. She is going to start her master program on Computer Science this September.*
Huyen Tran Ngoc Nhat *has recently graduated from FPT University in Vietnam. She now focuses on applying machine-learning techniques to solve real-life problems.*

2. High level summary of your approach: what did you do and why?

   Formulate a linear programming model for the optimization problem at each step with forecast data and use an open-source tool, which can be installed by pip to solve it.

   Scatter the energy charged (or discharged) among steps to avoid buying superfluous energy due to the uncertainty of next forecasts.

3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

   Part 1*: formulate mathematically the problem as a linear programming model using OR-Tools and solve it with glop*

```
solver = pywraplp.Solver('Battery_controller', pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)
        infinity = solver.infinity()
        # Variables
        capa = battery.capacity
        c_p_l =  ( battery.charging_power_limit * battery.charging_efficiency) / 4.0
        d_p_l = ( battery.discharging_power_limit / battery.discharging_efficiency) /
4.0

        X = [ solver.NumVar(0.0, capa, "X" + str( i)) for i in range( n+1)]
        Y = [ solver.NumVar(0.0, c_p_l, "Y"+ str( i)) for i in range( n)]
        Z = [ solver.NumVar(0.0, infinity, "Z"+ str( i)) for i in range( n)]


        eff = 1.0 / battery.charging_efficiency
        dis_eff =  battery.discharging_efficiency
        price_buy = price_buy.tolist()
```

```
        price_sell = price_sell.tolist()
        load_forecast = load_forecast.tolist()
        pv_forecast = pv_forecast.tolist()

        objective = solver.Objective()

        objective.SetCoefficient( X[n], price_sell[n-1] * dis_eff)

        for i in range( n):
            objective.SetCoefficient( Z[i],  price_buy[i] - price_sell[i])
            objective.SetCoefficient( Y[i], price_buy[i]/2000.0 + price_sell[i] * ( eff
- dis_eff))
            if( i>=1): objective.SetCoefficient( X[i], (price_sell[i-1] -
price_sell[i]) * dis_eff)
        objective.SetMinimization()
        # Constraints
        g_constraint = [None]*n
        ec_constraint = [solver.Constraint( 0.0, infinity) for i in range( n)]
        x_constraint = [solver.Constraint( d_p_l , infinity) for i in range( n)]

        init = solver.Constraint( battery.current_charge * capa, battery.current_charge
* capa)
        init.SetCoefficient( X[0], 1.0)
        for i in range( n):
            x_constraint[i].SetCoefficient( X[i+1], 1.0)
            x_constraint[i].SetCoefficient( X[i], -1.0)

            ec_constraint[i].SetCoefficient( Y[i], 1.0)
            ec_constraint[i].SetCoefficient( X[i+1], -1.0)
            ec_constraint[i].SetCoefficient( X[i], 1.0)

            if( pv_forecast[i]<20): pv_forecast[i] = 0.0

            g_constraint[i] = solver.Constraint( load_forecast[i] - pv_forecast[i],
infinity)
            g_constraint[i].SetCoefficient( Z[i], 1.0)
            g_constraint[i].SetCoefficient( Y[i], dis_eff - eff)
            g_constraint[i].SetCoefficient( X[i+1], -dis_eff)
            g_constraint[i].SetCoefficient( X[i], dis_eff)
        # Solver

        solver.Solve()
```

Part 2: *Avoid buying energy due to errors of forecasts by scattering the energy charged (discharged) among steps*.

```
if(  ( X[1].solution_value() >= X[0].solution_value()) & (pv_forecast[0] >
load_forecast[0])):
            charging = X[1].solution_value() - X[0].solution_value()
            energy = pv_forecast[0] - load_forecast[0]
            for i in range( 1, n):
```

```
                if( ( price_sell[i] != price_sell[i-1]) | ( pv_forecast[i] <
load_forecast[i]) | ( X[i].solution_value() > X[i+1].solution_value()) ): break
                energy = energy + pv_forecast[i] - load_forecast[i]
                charging = charging + X[i+1].solution_value() - X[i].solution_value()
            if( energy <= 0.0): return X[1].solution_value() / capa
            return battery.current_charge + ( ( pv_forecast[0] - load_forecast[0]) *
(charging / energy)) / capa


        if(  ( X[1].solution_value() <= X[0].solution_value()) & (pv_forecast[0] <
load_forecast[0])):
            charging = X[0].solution_value() - X[1].solution_value()
            energy = load_forecast[0] - pv_forecast[0]
            for i in range( 1, n):
                if( ( price_sell[i] != price_sell[i-1]) | ( price_buy[i] !=
price_buy[i-1]) | ( pv_forecast[i] > load_forecast[i]) | ( X[i].solution_value() <
X[i+1].solution_value()) ): break
                energy = energy + load_forecast[i] - pv_forecast[i]
                charging = charging + X[i].solution_value() - X[i+1].solution_value()
            if( energy <= 0.0): return X[1].solution_value() / capa
            return battery.current_charge + ( ( pv_forecast[0] - load_forecast[0]) *
(charging / energy)) / capa
        return  X[1].solution_value() / capa
```

Part 3: *Some other tricks but they can help to improve our scores just a bit.*

```
self.id = self.id + 1
n = max(1, min(96, 960 - self.id))
```

For each period, we will discharge all power in battery at the end. It means that the battery should end up at 0 in 10th day.

```
if (pv_forecast[i] < 20): pv_forecast[i] = 0.0
```

If pv_forecast is too small, we can set it to zero. We observed that in many forecasts, the pv could produce energy even in midnight. This line of code was used to avoid the situation.

4. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
   We tried to search for some tools and integrated solvers which can solve Linear Programming (LP) and Mixed Integer Programming with Python, and tested them on our model. Finally, we concluded that OR-tools is the best.

Tool	Integrated Solver	LP	MIP	Install by pip	Note	Result
PySCIPOpt	SCIP	Yes	Yes	Yes	- fastest but require SCIP Optimization Suite (must be installed separately)	Reject

OR-Tools	- CBC(default solver) for lp and mip - Glop for lp	Yes	Yes	Yes	- glop is good and really fast	**Accept**
Gurobi	Gurobi	Yes	Yes	No	- cannot install on docker	Reject
PyGLPK	GLPK	Yes	No	No	- cannot install on docker	Reject
Pyomo	GLPK(default solver)	Yes	No	Yes	- slowest running time - least effective	**Accept**
PuLP	CBC (default solver)	Yes	Yes	Yes	- CBC is quite slow - instances solved is less efficient than others	**Accept**
CVXOPT	CVXOPT	Yes	No	Yes	- faster than GLPK	**Accept**
CVXPY	ECOS, CVXOPT, and SCS	Yes	No	Yes	- cvxopt faster than GPLK	**Accept**

5. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
   Not yet.

6. How did you evaluate performance of the model other than the provided metric, if at all?
   We only used the provided metric to test our model. However, we could compute the maximum scores possible for the submit data by using the real values of building load and PV production as the input for the linear programming model. We then could compare our scores (for forecasting data) with these upper bounds to further analyze how good our method was.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
   No.

8. Do you have any useful charts, graphs, or visualizations from the process?
   Not yet.

9. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

In the future, we should use cplex – the best lp/mip tool to improve the running time. From the historical data, we could find the probability distribution function for the variance in the forecast of pv and load; and then consider a robust optimization problem. More precisely, we would search for a solution such that the probability of buying additional energy from the grid due to uncertainty of the forecast is restricted to a given threshold.