Project Report in partial fulfillment of Spring 2021 EC/ME/SE 543 Course Requirements

Title: Demand Scheduling of EV Charging

Date Submitted: 5/6/21

Acknowledgment:

As the sole author of this report I certify that I am aware of the following citation/referencing requirements and that failure to abide by them may result in academic misconduct disciplinary action and a failing grade.

- All passages, figures, tables, graphs and any other material lifted from a web site document, an existing report whether archived or informally made available to me, must be included in quotes and referenced in the text by exact bibliographic form and page number.
- The above holds for paraphrased or modified information or ideas taken from any source. In this case the reference should appear at the bottom of a table or graph or end of a paragraph without quotes.

Furthermore, I certify that I have carefully followed the above requirements in putting together the paper that follows.

Name: Zachary Weiss

Signature: Zachary Weiss

Abstract

As electric vehicles (EVs) continue to gain market share, the question of charging becomes an important one at the transmission-grid level. Owners of EVs wish to charge their vehicles when electric prices are lowest while still ensuring sufficient charge by the time of utilization, and system operators wish to level the demand curve, and, where possible, leverage additional storage connected to the grid to abate supply fluctuations that might force more expensive generation online. In an ideal market, this is fully coordinated and modulated by price; this project sets out to determine the optimal demand schedule of EV cohorts over the course of a day with synthetic data, and seeks to be abstractable to modeling of both greater and lesser complexity.

1. Introduction

1.1 Problem Formulation

In its simplest form, the problem of scheduling EV demand is for a single cohort, available to charge during any hour, with the additional load having negligible impact on clearing price / grid-level constraints, and with accurate price forecasts. Discretizing the problem into hourly chunks, this can be formalized as:

$$\min_{D_t \, \forall t} J = \sum_{t=0}^{N} \Pi_{base,t} * D_t \quad (1)$$

Subject to:

$$0 \leq S_t \leq S_{MAX} \;\; \forall t \in [0, N] \quad (2)$$

$$-R_{MAX} \leq D_t \leq R_{MAX} \; \forall t \in [0, N] \quad (3)$$

$$S_N = S_{MAX} \quad (4)$$

$$S_t = S_0 + \sum_{i=0}^{t} D_i \text{ or equivalently, } S_{t+1} = S_t + D_t \ \forall t \in [0, N] \quad (5)$$

Where J is the cost function, S is the stored energy, D is the demand from the EV cohort, R is the charging rate, and Π is the base forecast clearing price. This makes some key assumptions, namely: the EV must be fully charged at the end of the scheduled window, the EV can sell back to the grid at the clearing price of purchasing energy, the EV load has negligible impact on the grid and clearing price, the minimum storage capacity is zero, and that the maximum charge and discharge rates are symmetric. Given perfect knowledge ahead of time of the hourly clearing price, one can intuitively see demand will be scheduled during hours in order of increasing clearing price, and energy will be sold back to the grid during hours in order of decreasing clearing price, with the rates of (dis)charge and scheduled hours determined by the constraint of being fully charged by the final hour. As this is fairly trivially solved in a linear manner, we increase the complexity of the problem to allow the load scheduling from the EV to impact the clearing price. In the most naïve form, we can formalize this as the clearing price equaling the base price forecast in addition to our load multiplied by some coefficient determining the degree of impact.

$$\Pi_t = \Pi_{\text{base}} + P * D_t \ \forall t \in [0, N] \quad (6)$$

Where P is our price impact coefficient. This is of course overly simplistic, but at least turns the problem into a nonlinear one (referred to as NLP from here on out), and requires the optimal schedule to consider the amount it schedules by hour on more than just the order of hourly price forecasts. We can complicate the model further by restricting the set of hours during which an EV is allowed to charge: a realistic and important constraint given the hours during which

vehicles would be parked at home, or parked at an office with EV outlets, for example. This changes the formalized problem to:

$$\min_{D_t\ \forall t} J = \sum_{t=0}^{N} \Pi_t * D_t \quad (7)$$

Subject to:

$$0 \leq S_t \leq S_{MAX}\ \ \forall t \in [0, N] \quad (8)$$

$$\begin{cases} -R_{MAX} \leq D_t \leq R_{MAX}, & \forall t \in H \in [0, N] \\ D_t = 0, & \forall t \notin H \in [0, N] \end{cases} \quad (9)$$

$$S_{t+1} = S_t + D_t\ \ \forall t \in [0, N] \quad (10)$$

$$S_N = S_{MAX} \quad (11)$$

$$\Pi_t = \Pi_{\text{base}} + P * D_t\ \ \forall t \in [0, N] \quad (12)$$

Where H is a set containing the active (available for charging and discharging) hours. Lastly, we can add complexity to the problem by attempting to find the optimal schedule across multiple cohorts of EVs, all with their own capacities, initial charges, maximum charging rates, and hours of activity. This rewrites the problem to be:

$$\min_{D_{i,t}\ \forall t \in [0,N]\ \forall i \in [0,M]} J = \sum_{i=0}^{M} \sum_{t=0}^{N} \Pi_t * D_{i,t} \quad (13)$$

Subject to:

$$0 \leq S_{i,t} \leq S_{i,MAX}\ \ \forall t \in [0, N]\ \ \forall i \in [0, M] \quad (14)$$

$$\begin{cases} -R_{i,MAX} \leq D_{i,t} \leq R_{i,MAX}, & \forall t \in H_i \in [0, N] \\ D_{i,t} = 0, & \forall t \notin H_i \in [0, N] \end{cases} \quad \forall i \in [0, M] \quad (15)$$

$$S_{i,t+1} = S_{i,t} + D_{i,t}\ \ \forall t \in [0, N]\ \ \forall i \in [0, M] \quad (16)$$

$$S_{i,N} = S_{i,MAX} \quad \forall i \in [0, M] \quad (17)$$

$$\Pi_t = \Pi_{\text{base}} + \sum_{i=0}^{M} P_i * D_{i,t} \quad \forall t \in [0, N] \quad (18)$$

With the problem fully described, the next challenge is finding the optimal solution subject to the given constraints (with the presumption that $\exists D$ that fulfills the given initial conditions and constraint parameters). To do so, we turn to Python, its applicable libraries, and compatible NLP solver frameworks.

1.2 Prior Work

1.2.1 Pyomo

The first python library to be leveraged is pyomo. Pyomo is, "a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities". It flexibly interfaces with 3rd party solvers, supporting linear, nonlinear, mixed-integer, stochastic, disjunctive, differential, and bilevel programming, alongside mathematical programs with equilibrium constraints. Here it is used to express the above stated formalization of the demand scheduling problem, for subsequent use in the NLP solver. The use of pyomo in this project falls under its custom license agreement, copywrite and redistribution terms. [1] [2]

1.2.2 Interior Point Optimizer (Ipopt)

Interior Point Optimizer is, "an open source software package for large-scale nonlinear optimization. It can be used to solve general nonlinear programming problems of the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$s.t. \quad g^L \leq g(x) \leq g^U$$

$$x^L \leq x \leq x^U$$" [3]

This form is compatible with the problem formulation of the previous section, and as such it was selected for this project. Ipopt is one of the many solvers pyomo can interface with, and even provides a wrapper for. In later code one will see references to the "multistart" solver—this is simply a wrapper for the Ipopt solver with multiple starting points for the optimization. This package's use falls under the terms of the Eclipse Public License.

1.2.3 NumPy

Finally, the NumPy library is utilized. NumPy is a scientific computing package for python, simplifying many numeric computations and matrix operations, and as such is considered a standard in Python. It sees use here throughout, handling any matrix operations outside the NLP solver itself. NumPy is distributed under the BSD license. [4]

2. Methods

2.1 Overview

After selecting the aforementioned libraries, the formulated problem was solved in steps of increasing complexity. The first iteration solely optimized for a single cohort, available to charge whenever during the day. The second iteration optimized a single EV cohort with limitations to the charging hours. The third iteration abstracted to multiple EV cohorts, made minor changes to the synthetic data used, and implemented human-interpretable readouts; by modifying the configuration of the third iteration, it can be made to optimize the simpler scenarios of the first two project iterations, but with its interface improvements.

2.2 Single Cohort, Always Online

In the case of the simplest optimization problem formulated, upon which the later problems are built, the code (listed in full in the appendix: *File 1*) begins by defining global parameters for the number of hours, and the configuration of the EV cohort to be scheduled. The EV configuration entails its starting charge, "S_0", its maximum capacity "S_max", its maximum charging rate "R_max", and its price influence coefficient "P" (as defined in *Section 1.1*). Additionally, a base clearing price forecast signal is generated over the duration of the hours specified—in this iteration this takes the form of scaled random numbers, in later iterations the synthetic data is made more realistic.

After defining all parameters, the model is instantiated in pyomo, with decision variables created for the charge, charging rate, and adjusted clearing price. The objective function, *Equation 1*, and constraints, *Equations 2* through *5*, are applied, with all non-boundary condition constraints being applied over each hour in the set of hours specified. This problem, with applied decision variables, objective function, and constraints is passed to Ipopt through the multistart wrapper, and returned results are displayed.

2.3 Single Cohort, Limited Hours

To modify the single cohort, always online case for the limited hours problem formulation, a new field for active hours, "H", is added to the EV config global setting. To avoid issues when passing to the solver, we check to ensure all hours specified active are within the window of total hours to be optimized over, and if not, throw an error requesting the user modify the configuration such that active hours are within the overall duration they selected.

To modify the constraints to agree with *Equation 9*, within the loop adding constraints for each hour, a check is added that compares the current hour to the configuration; if the current hour does not appear in the configuration, the charge rate is constrained to zero for that hour block, else it is bounded, as before, by its upper and lower limits specified.

2.4 Multiple Cohorts, Limited Hours

For the final project iteration, the optimization is expanded to multiple cohorts of EVs, all individually configurable. The EV config global variable is expanded to an array of dicts, of arbitrary length greater than or equal to one (within the limit of one's computing power), each entry containing the same fields as in the former versions.

To improve useability, a set of notes, and a key to understanding variable names is printed on the screen at the start of the program execution. It additionally lists the current EV configuration, for ease of reference when interpreting the results, and to allow users to ensure they have not misconfigured it. The input sanitization checks remain the same, now applied iteratively to each EV listed within the config.

As random noise does not particularly well represent standard hourly pricing, the base clearing price forecast signal is modified to be a simple gaussian, scaled and shifted, centered at 6pm, a standard summer peak hour, approximating the hourly profile provided by the U.S. Energy Information Administration (EIA) [5]. To accommodate the iteration over an arbitrary number of EV cohorts, the dimensionality of the decision variables is increased, with one axis holding the hours as before, and the other: the EV cohort to which it corresponds. Constraint instantiation is similarly iterated over per EV, as per *Equations 14* through *18*. Due to technicalities of pyomo variable handling, the inequality constraints, previously handled by

limiting the bounds of the relevant decision variables, now are applied directly as inequalities at each hour.

Once solved, the results are passed to a custom data wrangling function. Here the solutions are extracted from pyomo, basic stats calculated, and the results printed in a pleasant human-readable form via the console. The storage values, charging rates, and clearing price are listed by hour, with the storage and charging listed on a per-EV basis, to the number of EVs specified. Within the results, active hours (those during which the given EV can charge or discharge) are represented by colored readouts: green for positive values, red for negative. Finally, the total cost per EV cohort, and the average price of energy each received, is listed. All code can be found in the *Appendix* below, and in downloadable form and more pleasantly displayed at https://github.com/zacharyweiss/demandscheduling.

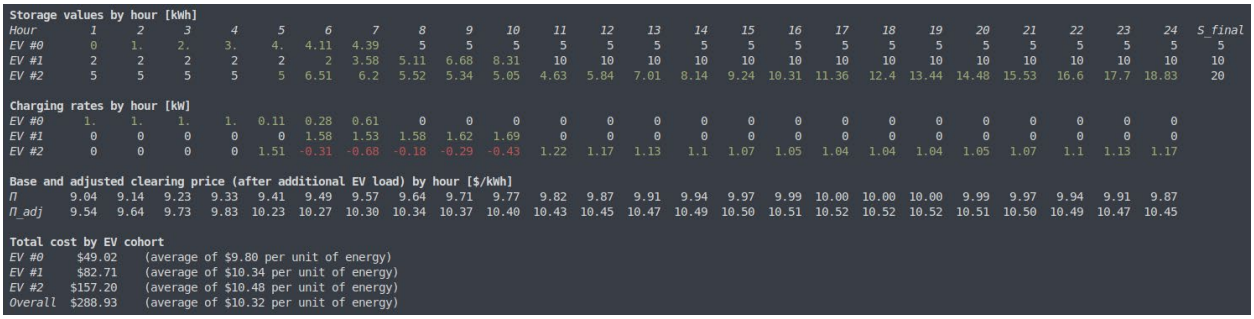
3. Results

As described in Section 1.2.3, running the final program iteration first yields the key and EV configuration specified.

```
optimize ❯ python multi_bounded.py

Key and Notes
Dependencies: pyomo, numpy, and the multistart NLP solver (may come preinstalled with pyomo)
S -> stored energy [kWh]
R -> charge rate [kW] (as everywhere referenced the rate is applied over an hour—implied "*1hr" after each instance—
     it is effectively in units of kWh)
H -> (array of) hours available to charge during [unitless indexes]
P -> price influence coefficient, zero makes price independent of EV demand [$/kWh^2]

Values highlighted in green or red indicate the EV was available for charge or discharge (selling back to the grid)
during that hour, whereas the EV cohort is offline for all entries in the default text color. To compute for other EV
configurations, edit the EV_CONFIG variable at the top of this file and re-run, following the example entry format for
each new EV cohort, separated by commas. If tables appear weird, expand your window horizontally or disable text wrapping.

EV Cohort Configuration
EV #0       S_0: 0          S_max: 5        R_max: 1        H: range(0, 7)   P: 0.5
EV #1       S_0: 2          S_max: 10       R_max: 2        H: range(5, 10)  P: 0.5
EV #2       S_0: 5          S_max: 20       R_max: 3        H: range(4, 24)  P: 0.5
```

Figure 1. Key and configuration readout from multi_bounded.py

This is followed by the readout of the optimal schedule found:



Storage values by hour [kWh]

Hour	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	S_final
EV #0	0	1.	2.	3.	4.	4.11	4.39	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
EV #1	2	2	2	2	2	2	3.58	5.11	6.68	8.31	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
EV #2	5	5	5	5	5	6.51	6.2	5.52	5.34	5.05	4.63	5.84	7.01	8.14	9.24	10.31	11.36	12.4	13.44	14.48	15.53	16.6	17.7	18.83	20

Charging rates by hour [kW]

EV #0	1.	1.	1.	1.	0.11	0.28	0.61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
EV #1	0	0	0	0	0	1.58	1.53	1.58	1.62	1.69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
EV #2	0	0	0	0	1.51	-0.31	-0.68	-0.18	-0.29	-0.43	1.22	1.17	1.13	1.1	1.07	1.05	1.04	1.04	1.04	1.05	1.07	1.1	1.13	1.17	

Base and adjusted clearing price (after additional EV load) by hour [$/kWh]

Π	9.04	9.14	9.23	9.33	9.41	9.49	9.57	9.64	9.71	9.77	9.82	9.87	9.91	9.94	9.97	9.99	10.00	10.00	10.00	9.99	9.97	9.94	9.91	9.87	
Π_adj	9.54	9.64	9.73	9.83	10.23	10.27	10.30	10.34	10.37	10.40	10.43	10.45	10.47	10.49	10.50	10.51	10.52	10.52	10.52	10.51	10.50	10.49	10.47	10.45	

Total cost by EV cohort

EV #0	$49.02	(average of $9.80 per unit of energy)
EV #1	$82.71	(average of $10.34 per unit of energy)
EV #2	$157.20	(average of $10.48 per unit of energy)
Overall	$288.93	(average of $10.32 per unit of energy)

Figure 2. Sample console-logged solutions from multi_bounded.py

As demonstrated in the sample results, pictured above, we can see the optimization seeking to schedule more charging during hours of lower prices, up until the impact of the additional EV load brings the clearing price near equal by hour, in hours where other constraints do not prevent this behavior. Additionally, as seen in hours 7 through 10, when multiple EVs are online, and one EV has significant charge already stored, the EV finds the optimal schedule to involve one of the cohorts selling back to the grid, while the others charge in their more limited charging windows.

4. Future Work

The majority of future work lies in challenging the many simplifying assumptions made in this current problem formulation and solving the more complex problem that yields. The first adjustment would be to generate more realistic synthetic data; at present the project imitates the general profile over the course of a day with a gaussian, but this is easily redefinable, to produce values that reflect actual energy prices, yielding results that can provide some level of insight rather than simply demonstrate a working model. Along the same lines, the assumption of perfect prior knowledge of pricing over the scheduling window is unrealistic. Addressing the creation of a more robust scheduling optimizer in the face of confidence-interval bounded prices / system

loads is a more difficult one, potentially approachable by either applying input noise and solving a range of conditions, and choosing one within the acceptable failure rate, or alternatively baking the robustness into the model as a constraint in some form—requiring some amount be withheld. One could even implement a preemption schema for live use of the scheduler, updating scheduled demand in a probabilistic manner.

Another current assumption is the desire to charge all cohorts to their maximum capacity by the end of the charging window. In the real world however, not all EV users care about this; if one is charging their vehicle in the office parking lot, the owner may simply care about having sufficient charge to get home. A consumer-facing consideration as well, is the number of charge and discharge cycles, which owners may wish to minimize due to wear on the battery, in some amount of balance with ability to charge and sell back to the grid.

Regarding the sale of energy back to the grid, there's two key assumptions here that could be expanded. The smaller of the two is the assumption of symmetric maximum charge and discharge rates, but this is easily fixed by adding another parameter to the configuration. The other consideration is the price EVs can sell back to the grid at, which currently is modeled to be the same as the clearing price, but in real scenarios is likely to be less than the clearing price for consumers. This in addition to some inefficiency in the charge and discharge, will lead to decreased revenue to EVs for acting as grid-attached storage compared to the present model.

Scheduling windows for different EV cohorts could use a minor overhaul as well. Presently, they do not account for modular mathematics—if one wished to schedule a cohort overnight within the default 24 hour period, such as from 10pm to 8am, this would be parsed as 12am to 8am occurring first, followed by 10pm to 12am. To circumvent this one could extend the hours modeled to two days and schedule the cohort across the contiguously-indexed night,

but this would require changing the price signal as well, in addition to overcomplicating the setup for a user. The alternative would be to recognize the hour index as being modulo 24, and specifying start and end hours for the initial and max charge of each EV cohort.

Finally, the largest undertaking but perhaps the most interesting, would be to implement a more intelligent price fluctuation function of demand. The first possible iteration would be synthetic supply and demand curves, which would introduce stepwise price changes due to added EV demand, as compared to the current smooth price increases and decreases. One level more complex would be to model a grid of a given size with loads and generators at each bus, each with independent demand and supply capacities / schedules / etc. The complexity here grows quickly, but even implemented on a small scale, such as a three or four bus system, would allow for the pricing impact of binding constraints to be considered, and add an additional layer of challenges in finding the optimal hours to schedule EV demand or resale.

5. Usage Notes

All code, in addition to being listed in the appendix, is available for viewing and download at https://github.com/zacharyweiss/demandscheduling. To run locally, one must download the primary file, *multi_bounded.py*, install the dependencies, configure the EV_CONFIG parameters as one wishes, and run the file.

To install dependencies, if one has Anaconda, a Python environment manager, installed on their machine, the command "`conda install numpy && conda install -c conda-forge pyomo ipopt=3.11.1`" can be run from the terminal.

6. Appendix

```python
#!/usr/bin/env python
__author__ = "Zachary Weiss"

import pyomo.environ as pyo
import numpy as np

# global settings
N_HOURS = 24
EV_CONFIG = {"S_0": 0, "S_max": 5, "R_max": 1, "P": 1}
# S -> stored energy
# R -> charge rate (per hour)
# P -> price influence coefficient, zero makes price independent of EV demand

# price / load signals
base_prices = np.random.rand(N_HOURS) * 10


def main():
    model = pyo.ConcreteModel()

    # index
    hours = range(N_HOURS)

    # decision vars
    model.S = pyo.Var(range(N_HOURS+1), bounds=(0, EV_CONFIG["S_max"]), within=pyo.NonNegativeReals)
    model.R = pyo.Var(hours, bounds=(-EV_CONFIG["R_max"], EV_CONFIG["R_max"]), within=pyo.Reals)
    model.P = pyo.Var(hours, within=pyo.NonNegativeReals)

    # objective function
    cost = sum(model.P[t] * model.R[t] for t in hours)
    model.cost = pyo.Objective(expr=cost, sense=pyo.minimize)

    # constraints
    model.cons = pyo.ConstraintList()
    # boundary condition: after final hour, storage must equal maximum charge
    model.cons.add(model.S[N_HOURS] == EV_CONFIG["S_max"])
    # boundary condition: storage begins at initial value
    model.cons.add(model.S[0] == EV_CONFIG["S_0"])

    # constraints applied each hour (bounds already handled in pyomo variable declaration)
    for t in hours:
        # update rule, storage at next time point is current storage plus amount charged
        model.cons.add(model.S[t+1] == model.S[t] + model.R[t])
        # price at each hour is sum of base price and amount of price increase from the load scheduled
        model.cons.add(model.P[t] == base_prices[t] + (EV_CONFIG["P"] * model.R[t]))

    # cbc, glpk, gurobi, cplex, pico, scip, xpress: LP/MIP solvers
    # conopt, cyipopt, ipopt: NLP
    # path: MCP
    # more can be found via "pyomo help --solvers"
    results = pyo.SolverFactory('multistart').solve(model, suppress_unbounded_warning=True)

    # display results
    model.display()
    print("done")


if __name__ == '__main__':
    main()
```

File 1. single_unbounded.py: simplest form of model

```python
#!/usr/bin/env python
"""
Single EV cohort, limited hours of (dis)charge, able to influence price

Presumes EV should be fully charged after final hour connected to the grid, in addition to all assumptions
stated in
the earlier single_unbounded case.
"""

__author__ = "Zachary Weiss"

import pyomo.environ as pyo
import numpy as np

# global settings
N_HOURS = 24
EV_CONFIG = {"S_0": 0, "S_max": 5, "R_max": 1, "H": range(4, 18), "P": 1}
# S -> stored energy
# R -> charge rate (per hour)
# H -> (array of) hours available to charge during
# P -> price influence coefficient, zero makes price independent of EV demand

# price / load signals
base_prices = np.random.rand(N_HOURS) * 10
base_loads = np.random.rand(N_HOURS) * 2


def main():
    model = pyo.ConcreteModel()

    # check valid hour configuration (no online hours specified beyond N_HOURS)
    if max(EV_CONFIG["H"]) >= N_HOURS or min(EV_CONFIG["H"]) < 0:
        raise SystemExit("Hours specified for EV (dis)charge must be between zero and N_HOURS. Modify the EV"
                         "config and rerun.")

    # index
    hours = range(N_HOURS)

    # decision vars
    model.S = pyo.Var(range(N_HOURS + 1), bounds=(0, EV_CONFIG["S_max"]), within=pyo.NonNegativeReals)
    model.R = pyo.Var(hours, bounds=(-EV_CONFIG["R_max"], EV_CONFIG["R_max"]), within=pyo.Reals)
    model.P = pyo.Var(hours, within=pyo.NonNegativeReals)

    # objective function
    cost = sum(model.P[t] * model.R[t] for t in hours)
    model.cost = pyo.Objective(expr=cost, sense=pyo.minimize)

    # constraints
    model.cons = pyo.ConstraintList()
    # boundary condition: storage begins at initial value
    model.cons.add(model.S[0] == EV_CONFIG["S_0"])
    # boundary condition: after final hour, storage must equal maximum charge
    model.cons.add(model.S[N_HOURS] == EV_CONFIG["S_max"])

    # constraints applied each hour (bounds already handled in pyomo variable declaration)
    for t in hours:
        # if the EV is not able to (dis)charge during the current hour, the rate must be zero
        if t not in EV_CONFIG["H"]:
            model.cons.add(model.R[t] == 0)
        # update rule, storage at next time point is current storage plus amount charged
        model.cons.add(model.S[t + 1] == model.S[t] + model.R[t])
        # price at each hour is sum of base price and amount of price increase from the load scheduled
        model.cons.add(model.P[t] == base_prices[t] + (EV_CONFIG["P"] * model.R[t]))

    # cbc, glpk, gurobi, cplex, pico, scip, xpress: LP/MIP solvers
    # conopt, cyipopt, ipopt: NLP
    # path: MCP
    # more can be found via "pyomo help --solvers"
    results = pyo.SolverFactory('multistart').solve(model, suppress_unbounded_warning=True)

    # display results
    model.pprint()
    print("\n" + "#"*150 + "\n")
    results.write()


if __name__ == '__main__':
    main()
```

File 2. single_bounded.py: limited hours of scheduling

```
#!/usr/bin/env python
"""
Multiple EV cohorts, limited hours of (dis)charge, able to influence price

To install all dependencies with conda, run "conda install numpy && conda install -c conda-forge pyomo
ipopt=3.11.1"
"""

__author__ = "Zachary Weiss"

import pyomo.environ as pyo
import numpy as np

# global settings
N_HOURS = 24
EV_CONFIG = [{"S_0": 0, "S_max": 5, "R_max": 1, "H": range(7), "P": 0.5},
             {"S_0": 2, "S_max": 10, "R_max": 2, "H": range(5, 10), "P": 0.5},
             {"S_0": 5, "S_max": 20, "R_max": 3, "H": range(4, N_HOURS), "P": 0.5},
             ]
# Example entry: {"S_0": 0, "S_max": 5, "R_max": 1, "H": range(7), "P": 0.5}
# S -> stored energy [kWh]
# R -> charge rate [kW] (as everywhere referenced the rate is applied over an hour--implied "*1hr" after each
instance--
#      it is effectively in units of kWh)
# H -> (array of) hours available to charge during [unitless indexes]
# P -> price influence coefficient, zero makes price independent of EV demand [$/kWh^2]


def main():
    notes()

    model = pyo.ConcreteModel()

    # price signal: array of prices at each hour [$/kWh], peak value at 6pm
    # base_prices = np.random.rand(N_HOURS) * 10
    base_prices = 5 * gaussian(np.linspace(0, N_HOURS - 1, N_HOURS), 17, 26) + 5

    # check valid hour configuration (no online hours specified beyond N_HOURS)
    for ev in EV_CONFIG:
        if max(ev["H"]) >= N_HOURS or min(ev["H"]) < 0:
            raise SystemExit("Hours specified for EV (dis)charge must be between zero and N_HOURS. Modify the
EV "
                             "config and rerun.")

    # index
    hours = range(N_HOURS)
    model.i = pyo.Set(initialize=[i for i, ev in enumerate(EV_CONFIG)])
    model.t = pyo.Set(initialize=hours)
    model.t_1 = pyo.Set(initialize=range(N_HOURS + 1))

    def ij_init(m):
        # key pairs for S and R matrices w/i pyomo
        # i is the EV number, j is the hour (t)
        return ((i, j) for i in m.i for j in m.t)

    # same as above, but initializes for one extra hour to be compatible with the update rule
    def ij_init_1(m):
        return ((i, j) for i in m.i for j in m.t_1)

    # decision vars
    model.S = pyo.Var(pyo.Set(dimen=2, initialize=ij_init_1), within=pyo.NonNegativeReals)
    model.R = pyo.Var(pyo.Set(dimen=2, initialize=ij_init), within=pyo.Reals)
    model.P = pyo.Var(hours, within=pyo.NonNegativeReals)

    # objective function
    cost = sum(sum(model.P[t] * model.R[i, t] for t in hours) for i, ev in enumerate(EV_CONFIG))
    model.cost = pyo.Objective(expr=cost, sense=pyo.minimize)

    # constraints
    model.cons = pyo.ConstraintList()
    for i, ev in enumerate(EV_CONFIG):
        # boundary condition: storage begins at initial value
        model.cons.add(model.S[i, 0] == ev["S_0"])
        # boundary condition: after final schedule-able hour, storage must equal maximum charge
        model.cons.add(model.S[i, max(ev["H"]) + 1] == ev["S_max"])

    # constraints applied each hour (bounds already handled in pyomo variable declaration)
    for t in hours:
        for i, ev in enumerate(EV_CONFIG):
            # if the EV is not able to (dis)charge during the current hour, the rate must be zero. Else,
bounded by max
```

```
            # and min (now added as constraint as cannot be easily added in variable bounds at time of
declaration)
            if t not in ev["H"]:
                model.cons.add(model.R[i, t] == 0)
            else:
                model.cons.add(pyo.inequality(-ev["R_max"], model.R[i, t], ev["R_max"]))
            # stored energy must be between 0 and the maximum for each EV cohort
            model.cons.add(pyo.inequality(0, model.S[i, t], ev["S_max"]))
            # update rule, storage at next time point is current storage plus amount charged for each EV cohort
            model.cons.add(model.S[i, t + 1] == model.S[i, t] + model.R[i, t])
        # price at each hour is sum of base price and amount of price increase from the load scheduled
        model.cons.add(model.P[t] == base_prices[t] + sum([ev["P"] * model.R[i, t] for i, ev in
enumerate(EV_CONFIG)]))

    results = pyo.SolverFactory('multistart').solve(model, suppress_unbounded_warning=True)

    readout(model, base_prices)


def lrange(*args):
    """drop in replacement for 'range()', if one wishes to easily concatenate ranges with '+' in the
'EV_CONFIG'"""
    return list(range(*args))


def gaussian(x, mu, sig):
    return np.exp(-np.power(x - mu, 2.) / (2 * np.power(sig, 2.)))


def notes():
    print("""\n\033[1mKey and Notes\033[0m
\033[3mDependencies\033[0m: pyomo, numpy, ipopt, and multistart (NLP solver, should come pre-installed with
pyomo)
If not fully installed, run "conda install numpy && conda install -c conda-forge pyomo ipopt=3.11.1" without
quotes.
\033[3mS\033[0m -> stored energy [kWh]
\033[3mR\033[0m -> charge rate [kW] (as everywhere referenced the rate is applied over an hour\u2014implied
"*1hr" after each instance\u2014
     it is effectively in units of kWh)
\033[3mH\033[0m -> (array of) hours available to charge during [unitless indexes]
\033[3mP\033[0m -> price influence coefficient, zero makes price independent of EV demand [$/kWh^2]

Values highlighted in green or red indicate the EV was available for charge or discharge (selling back to the
grid)
during that hour, whereas the EV cohort is offline for all entries in the default text color. To compute for
other EV
configurations, edit the EV_CONFIG variable at the top of this file and re-run, following the example entry
format for
each new EV cohort, separated by commas. If tables appear weird, expand your window horizontally or disable
text wrapping.""")

    print("\n\033[1mEV Cohort Configuration\033[0m")
    for i, ev in enumerate(EV_CONFIG):
        print("\033[3m{:<12s}\033[0m".format(f"EV #{i}") + "".join("{:<25s}".format(f"\033[3m{k}\033[0m:
{ev[k]}") for k
                                                                  in ev))


def readout(model, base_prices):
    S_sol = np.array([v.value for i, v in model.S.items()]).reshape(len(EV_CONFIG), N_HOURS + 1)
    R_sol = np.array([v.value for i, v in model.R.items()]).reshape(len(EV_CONFIG), N_HOURS)
    P_sol = np.array([v.value for i, v in model.P.items()])

    costs = np.array([np.array(ev_R * P_sol) for ev_R in R_sol])
    ev_tot_cost = costs.sum(axis=1)
    ev_avg_price = ev_tot_cost / R_sol.sum(axis=1)

    def pretty_print(arr):
        # zero out floating point errors within tolerance
        tol = 1e-16
        arr[(arr < tol) & (-tol < arr)] = 0

        def trimmer(n):
            return np.format_float_positional(n, 2, trim="-")

        for i, row in enumerate(arr):
            num_str = ''.join(
                (f'\033[92m{trimmer(val): >7s}\033[0m' if val >= 0 else f'\033[91m{trimmer(val): >7s}\033[0m')
                if t in EV_CONFIG[i]["H"] else f'{trimmer(val): >7s}' for t, val in enumerate(row))
            print("\033[3m{:<6s}\033[0m{}".format(f"EV #{i}", num_str))
```

```
    print("\n\033[1mStorage values by hour [kWh]\033[0m")
    hour_arr = np.append(np.add(lrange(N_HOURS), 1), "S_final")
    print("\033[3m{:<9s}".format("Hour") + ''.join("{:^7s}".format(hr) for hr in hour_arr) + "\033[0m")
    pretty_print(S_sol)

    print("\n\033[1mCharging rates by hour [kW]\033[0m")
    pretty_print(R_sol)

    print("\n\033[1mBase and adjusted clearing price (after additional EV load) by hour [$/kWh]\033[0m")
    print("\033[3m{:<6s}\033[0m".format("Π") + "".join(f"{item: >7.2f}" for item in base_prices))
    print("\033[3m{:<6s}\033[0m".format("Π_adj") + "".join(f"{item: >7.2f}" for item in P_sol))

    print("\n\033[1mTotal cost by EV cohort\033[0m")
    for i, ev in enumerate(EV_CONFIG):
        print("\033[3m{:<8s}\033[0m{:>8s}    {}".format(f"EV #{i}", f"${ev_tot_cost[i]:.2f}", "(average "
                                                                                                f"of
${ev_avg_price[i]:.2f} per unit of energy)"))
    print(f"\033[3mOverall\033[0m  ${model.cost():.2f}    (average of ${model.cost() / sum(sum(R_sol)):.2f} per
unit "
          f"of energy)")


if __name__ == '__main__':
    main()
```

File 3. multi_unbounded.py: full model with custom readout

Works Cited

[1] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson and D. L. Woodruff, Pyomo--optimization modeling in python, Third ed., vol. 67, Springer Science & Business Media, 2021.

[2] W. E. Hart, J.-P. Watson and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computation,* vol. 3, no. 3, pp. 219--260, 2011.

[3] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming,* vol. 106, no. 1, p. 25–57, 2006.

[4] C. R. Harris, K. J. Millman, S. J. van der Walt and et al., "Array programming with NumPy," *Nature,* vol. 585, no. 7825, pp. 357--362, 2020.

[5] T. Hodge, "Hourly electricity consumption varies throughout the day and across seasons - Today in Energy," U.S. Energy Information Administration, 21 February 2020. [Online]. Available: https://www.eia.gov/todayinenergy/detail.php?id=42915. [Accessed 5 May 2021].