IN5410 - Assignment 1

Group 1 - davidgek, emiliehf, peterhp and siljeben

March 2024

Abstract

As the energy landscape shifts towards decentralization and digitization, smart home management has become increasingly important in the recent years. With the introduction of smart meters and smart appliances the need for smart home management and management of the power infrastructure have become more widespread and important.

Dynamic pricing schemes present a market-based solution which allow the automatic shifting of non-essential appliances' usage time. This enables the energy market to align intermittent electricity generation of renewable energy more closely with the electricity usage of consumers. Additionally, a benefit is that it may reduce consumers energy bills at the same time.

In this assignment, we compare two pricing schemes, Time-of-Use (ToU) pricing and Real-Time-Pricing (RTP), for the optimization of appliance schedules of residential households and neighborhoods using a linear program (LP).
The ToU pricing scheme is a simpler scheme, which has a fixed higher electricity price for peak usage hours. The focus on pricing based on time of consumption promotes a shift in the habits of energy consumption towards less congested hours. This shift towards a more predictable electricity usage gives the providers more flexibility as well as allowing the consumers to change their habits to benefit from a lower price, even without the use of smart home technology.
The RTP pricing scheme is more volatile, being based on the power market and having rates that change every hour. However, the more detailed representation of supply and demand in RTP pricing allows for lower electricity bills and more flexible adjustments to changes in power generation. Our solution presents a flexible object-oriented framework for solving the LP scheduling problem.

Contents

1 Introduction

The demand for energy is continuously increasing, driven by factors such as population growth, urbanisation, and technological advancements. As a result, there is a pressing need to build out more renewable generation, as well as optimize energy consumption and enhance the efficiency of electricity-using equipment. In addition, international agreements on the reduction of emissions have made incentives for introducing more renewable energy sources into the energy mix. These are known for being more variable than traditional energy generators, resulting in more volatile electricity prices than historically seen.

Traditional approaches to energy management often involve passive consumption patterns, in which appliances operate without consideration for peak demand periods or grid constraints. However, such practices contribute to inefficiencies in energy utilization and worsen the strain on power infrastructure during peak hours. Demand response management presents a paradigm shift by enabling consumers to adjust their energy usage in response to grid conditions, thereby alleviating stress on the system and enhancing overall efficiency. The gain for the consumers is a reduced electricity bill. With advancing technology, and more sophisticated and *smarter* equipment, there is an opportunity to devise strategies for effectively managing household appliances.

Linear programming, a mathematical optimization technique, offers a systematic framework for addressing the complexities inherent in demand response management. By formulating energy consumption scheduling as an optimization problem, linear programming algorithms can generate optimal solutions that minimize electricity costs, mitigate peak demand, and accommodate consumer preferences and constraints.

The main objective of this assignment is to address the issue with varying prices, and schedule the appliances in the home to minimize the electricity bill. Three different cases are investigated: a simple household with three shiftable appliances, a complex household with both shiftable and non-shiftable appliances and a neighborhood with shiftable and non-shiftable appliances. In addition, we will look at the complex household when a peak load is introduced. Furthermore, two pricing schemes are implemented, ToU and RTP, and their impact on the energy cost is shortly discussed.

2 Background and libraries

2.1 Linear programming

LP is a mathematical optimization technique in which a linear function is maximized or minimized, subject to some linear constraints. Since we are going to minimize the cost of electricity in this assignment, the discussion will continue for a *minimization* problem. In mathematical terms we want to find a vector x in this way:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & A_{ub} x \leq b_{ub} \\ & A_{eq} = b_{eq} \\ & l \leq x \leq u \end{aligned}$$

These expressions coincide with the expression used in `scipy.optimize.linprog`, used to solve the LP in this assignment.

2.1.1 scipy.optimize.linprog

`scipy.optimize.linprog` is a function which is part of the `scipy.optimze` library. In general this library provides function for minimizing or maximizing objective functions, subject to constraints [1]. It includes solvers for many types of problems, including `linprog` which is the function used for solving linear problems. A brief description of the parameters of `linprog` used in this assignment can be found in Table 1. A more detailed description of the variables can be found in the documentation [2].

The function returns the optimal vector x (the decision variables that minimize the objective function) and the optimal value of the objective function, as well as other specific information about the solving.

Table 1: The parameters for scipy.optimize.linprog [2]

Parameter	Description	Type
c^T	The coefficients of the linear objective function to be minimized	1D array
A_{ub}	The inequality constraint matrix	2D array
b_{ub}	The inequality constraint vector	1D array
A_{eq}	The equality constraint matrix	2D array
b_{eq}	The equality constraint vector	1D array
lb	Lower bound of each entry of solution vector	Sequence
ub	Upper bound of each entry of solution vector	Sequence

2.2 Pricing schemes

In order to optimally balance the supply and demand in the grid, pricing schemes are introduced. This sends signal to the user of when to use electricity in terms of monetary compensations and/or penalties. In order for the price signals to work efficiently, the consumers must be able to observe the price and be able to react to it. This will lead to lower electricity bills, but can also alleviate strain on the electricity grid. The design of the pricing schemes vary in complexity and efficiency, but have in common that they incentive usage when total consumption in grid is low and discourage when the consumption is high.

2.2.1 Time-of-Use

In ToU pricing, the prices are defined way ahead of time (years or months) for the customer to plan their usage. The electricity prices is usually split hourly and grouped into intervals of a day, and will reflect both the demand and usage [3]. For example, one implementation can be that the period between 23 and 07 all have the same *off-peak price*, the time periods 07-15 and 20-23 have the same price with *mid-peak price*, while 15-20 have *on-peak price*.

2.2.2 Real-Time Pricing

RTP is a pricing model where the supplier adjust the price as soon as a change in the market is detected. This ensures that the prices of products and services reflects the current market condition [4]. The electricity price undergoes continual adjustments to meet the supply and demand. Electricity consumption is monitored in real-time, and will lead to hourly variations in the kWh price by leveraging smart meter technology. The pricing is usually released on an hour-ahead pricing or a day-ahead pricing. In this assignment the day-ahead release is assumed.

2.3 Appliance consumption

Appliance consumption refers to the amount of electrical energy utilized by household (or industrial) appliances over a specific period. This measurement is typically expressed in kilowatt-hours (kWh) or kilowatts (kW). Understanding these units is fundamental to grasping the energy usage of various devices and appliances.

A kWh is a unit of energy equivalent to one kW of power spent for one hour. It is what is referred to when we talk about *amount of energy*. A kW is a unit of power that represents the rate at which energy is consumed. It indicates the amount of energy consumed per unit of time, i.e. how fast the appliance can draw electricity. More specifically we have:

$$\text{Energy [kWh]} = \text{Power [kW]} \cdot \text{Length [h]}$$

Being aware of how appliances and other home electronics use electricity can give us valuable insight of our own energy consumption. As a common standard today, all electricity appliances provide an estimate of the average energy use. By taking advantage of this knowledge the consumer has the possibility to estimate the electricity costs, which can be obtained by calculating the daily or annual electricity consumption and costs. Through observing the more expensive appliances, you can determine whether to invest in a more energy efficient alternative, or to *strictly* monitor the energy usage to save money [5].
More of how we define our appliances can be found in section 3.1

2.4 Running the code

Since the questions stated that we need to reuse the price information for question 2 (section 4.2) and question 4 (section 4.4) we generated the RTP and saved it in the file `rt_pricing.npy` using the function

seen in section 4.2.1. The file is then used to load the pricing for questions 2, 3 and 4.

The code is split into several files and classes, as we have created classes for appliances A.1, households A.2 and neighborhoods A.3.
To actually run the code there is several dependencies needed, they can be found in the file `requirements.txt`. Each question is answered in corresponding file named `questionX.py`, where `X` is the number of the question. This can be seen in the appendices A.5 to A.8.
The code is also included in a zip-file.

3 Developed strategy and methods

3.1 Appliances

The day is split into 24 1hr time slots. From the given data for the shiftable and non-shiftable appliances we deduced that it represents a typical weekend, due to the lighting being switched on between 10:00-20:00. Because of this, one can assume that at least one person in the household is at home in these hours. Furthermore, we assume that the people in the household are still awake some time after 20, but then uses candlelight for lighting. This made it more flexible when distributing the different appliances to their respective timeslots. We also considered which time period each appliance was expected to be switched on. Several of the optional appliances are used for less than an hour each (typically 10-20 minutes), but since the granularity is only per hour, we set the usage for 1 hour (integer numbers). As an example an average coffee maker will use around 1000-1500 Watts to produce 8-10 cups of coffee in 20 minutes [6], which we model as using 1400 Watt for one hour.

Table 2 represent the different non-shiftable and shiftable appliances, where we have specified for each appliance the length of use in hours, the total daily usages in kWh and the hourly usage in kW. The file is used in the program to instantiate appliance objects based on the data. The Shiftable column is set to 0, 1 or 2, and represent if the appliance is non-shiftable, shiftable or semi-shiftable. By semi-shiftable we mean appliances that is either shiftable or non-shiftable, but more importantly that they are appliances that is neither explicitly stated to be included (from the assignment questions) or exist in every house. The appliances and their typical energy consumption values are taken from *Energy Use Calculator*[6].

Table 2: The appliances and values used

Appliances	**Shiftable**	**Length [hours]**	**Daily usage [kWh]**	**Hourly usage [kW]**	**Alpha**	**Beta**
Lighting	No = 0	10	1.5	0.15	10	20
Heating	No	24	7.2	0.3	0	24
Refrigerator-freezer	No	24	1.32	0.055	0	24
Electric stove	No	4	3.9	0.975	15	19
TV	No	5	0.4	0.08	18	23
Computer	No	4	0.6	0.15	16	20
Dishwasher	Yes = 1	3	1.44	0.48	12	24
Laundry machine	Yes	3	1.94	0.6467	10	23
Cloth dryer	Yes	2	2.5	1.25	10	22
EV	Yes	5	9.9	1.98	0	7
Coffee maker	Semi = 2	1	0.14	0.14	10	14
Vacuum	Semi	1	0.23	0.23	10	20
Hair dryer	Semi	1	0.25	0.25	6	8
Toaster	Semi	1	0.24	0.24	6	8
Microwave	Semi	1	0.6	0.6	16	22
Router	Semi	24	0.14	0.0058	0	24
Cellphone charger	Semi	3	0.01	0.0033	0	24

Every appliance has:

- daily energy consumption in kWh
- time to be used (length) in hours
- Start time α

- deadline time β

The hourly usage [kW] was calculated by taking daily usage [kWh]/ length [h] - see Appendix A.1.

3.2 Level of optimization

The information provided is structured and integrated into Appliances, Households and Neighborhoods (see Appendices A). Each appliance contains information corresponding to their entry in Table 2. A household simply contains a list of Appliances, and a Neighborhood contains a list of Households.

All questions are optimized at Neighborhood level rather than individual Households. A single house is simply constructed as a *Neighborhood of one*. Neighborhood's function `get_linprog_input()` is used to preprocess the parameters used in the `scipy` function `linprog` which does the optimization. How these parameters look is described in Section 3.3

The function `get_linprog_input()` iterates over all the houses and their respective appliances. For each appliance, it combines pricing data and forms the lower and upper bounds for the decision variables. The lower bound is defined as zero, and the upper bound is set as the maximum hourly consumption of the appliance. The function also builds exactly one equality constraint for each appliance, representing that the total daily usage should be equal to the appliance's specified daily use. If a *peak load* value is defined (specified in section 4.4), additional inequality constraints are added to ensure that the decision variables, which distribute the appliances' usage across the 24 hours, don't exceed this limit per hour.

By declaring the constraints and objectives this way, our linear programming problem optimizes the total cost of using appliances across a neighborhood while respecting the constraints of maximum usage per hour and total daily usage for each appliance, which, if a peak load limit is enforced, also ensures not exceeding this peak power draw. One drawback from this formulation is that for a Neighborhood with several houses and a peak load, some houses might sacrifice their own best for the collectives own best. In other words, if the optimization was done at household level for m houses with peak load L/m for each house and then summarized, the total price and schedule might look different than optimizing for the whole neighborhood. However, in this assignment we are not looking at a problem where both several households and a peak load is considered at the same time.

3.3 LP formulation

In this section the LP formulation for our specific problem is presented. A quick summary of each variable is presented in Table 3. N_{app} denote the *total number of appliances*.

Table 3: The parameters for scipy.optimize.linprog [2]

Parameter	Description	Shape
c^T	The pricing scheme	$(N_{app} \cdot 24,)$
A_{ub}	"Repeated" identity matrix of 24.	$(24, 24 \cdot N_{app})$
b_{ub}	The peak load L- vector	$(24,)$
A_{eq}	Time of usage matrix	$(N_{app}, 24 \cdot N_{app})$
b_{eq}	Total consumption per appliance	$(N_{app},)$
lb	Minimal hourly usage	$(24 \cdot N_{app},)$
ub	Maximal hourly usage	$(24 \cdot N_{app},)$

3.3.1 Minimization function

As the goal is to minimize the electricity bill, the objective function is

$$c^T = \begin{bmatrix} c_1 & c_2 & \cdots & c_{24} & c_1 & c_2 & \cdots & c_{24} & \cdots & c_{24} \end{bmatrix},$$
$$x = \begin{bmatrix} x_{1,1} & \cdots & x_{1,24} & x_{2,1} & \cdots & x_{2,24} & \cdots & x_{N_{app},1} & \cdots & x_{N_{app},24} \end{bmatrix},$$

where c_1 to c_{24} is each repeated N_{app} times and the value of c_i represents the power price per kW in hour i (ToU or RTP). x is the decision vector and $x_{i,j}$ represents the power usage of appliance i during hour j. Note that x is still a *vector* and not a matrix, even though we are using the notation $x_{i,j}$.

3.3.2 Lower and upper bounds

All appliances will *consume* electricity (and not provide it), so the lower bound for all $x_{i,j}$ is 0. Furthermore, we set each appliance' upper bound as γ_{max} equal to the hourly usage between α and β and 0 otherwise. A consequence of not introducing a maximum could be that an appliance that should use a total of 3kWh in 3 hours had consumption of 2.99 kW in one hour which was cheap and 0.005 kW in the two other more expensive hours, which is not a realistic. As will be discussed in section 3.3.5, this might also be a oversimplified assumption, but it hinders the optimization to yield results where almost all consumption is in one cheap hour.

3.3.3 Inequality constraint matrix and inequality constraint vector

The inequality constraint matrix A_{ub} and inequality constraint vector are used in question 4 when a (maximal) peak load L is introduced. The peak load L ensures that the grid stability is protected as the consumption per hour is limited to a maximum. The inequality constraint matrix is a matrix of shape $(24, 24 \cdot N_{app})$ and can be described as a "repeated identity matrix". This is because it is an identity matrix of 24 which is concatenated with another N_{app} times. Specifically, it looks like:

$$A_{ub} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

The inequality constraint vector is a $(24,)$ L-vector. This means a vector of length 24 with peak load L in every entry.
The matrix multiplication $A_{ub}x \leq b_{ub}$ yields

$$\forall j \in \{1, 2, \cdots 24\} \quad \Sigma_{i=1}^{N_{app}} x_{i,j} \leq L$$

which means that the peak load L is not exceeded in any hour of the day.

3.3.4 Equality constraint matrix and equality constraint vector

The equality constraint matrix and equality constraint vector captures the constraints regarding operating hours $\alpha - \beta$ and total consumption of each appliance. Let the appliances be numbered from 1 to N_{app} and let α_k and β_k denote the start-up and deadline for appliance k. The equality constraint matrix is in general a 0-matrix, but with 1's in $[24 \cdot (k-1) + \alpha_k, 24 \cdot (k-1) + \beta_k]$ for row k where $k \in \{1, 2 \cdots, N_{app}\}$. More specifically,

$$(A_{eq})_{kl} = \begin{cases} 1 & \text{if} \quad 24 \cdot (k-1) + \alpha_k \leq l \leq 24 \cdot (k-1) + \beta_k \\ 0 & \text{otherwise} \end{cases}$$

The equality constraint vector is a vector of shape $(N_{app},)$ where each entry $i \in \{1, 2, \cdots N_{app}\}$ corresponds to the total consumption of appliance i, denoted as E_i.
The matrix multiplication $A_{eq}x = b_{eq}$ yields $\forall i \in \{1, 2, \cdots N_{app}\}$

$$\Sigma_{j=\alpha_i}^{\beta_i} x_{i,j} = E_i$$

This forces the linear program to yield results where the total consumption is met in the right times.

3.3.5 Limitations LP formulation

A drawback from our formulation is that it doesn't consider the appliances natural energy profile. Most appliances don't have a steady consumption throughout the consumption time, but have individual smaller peaks. One example to illustrate the point is the non-shiftable refrigerator. Refrigerators do not have steady consumption throughout the day, but have smaller peaks distributed as a function of time and when the temperature dips. However, these peaks are not controllable as they are a mechanism in the refrigerator in order to keep the wanted temperature and not damage the equipment (and food in the fridge).

Furthermore, we have not used a technique where we say that once you start an appliance, it must keep going. This limitation is applicable for the shiftable appliances. The result we get is that a shiftable appliance may be used for an hour, then take a break, and then continue. In reality, one would not start a dishwasher, pause it for some hours, and then come back and continue. In addition, some appliances cannot run like this at all. If a laundry machine is ran like this, the clothes might be ruined. In other words, the

optimization implemented (might) yield results that are non-realistic.

To include the complexities mentioned, it requires information about the specific equipment used as well as a different approach rather than (solely) linear programming.

3.4 Algorithm

The general structure of our solutions to all questions is as follows:

- Read appliance data from excel file
- Instantiate (either create or load from disk) Neighborhood object with desired households and appliances
- Optimize Neighborhood using the `get_linprog_input()` function and the `scipy` function `linprog`, as described in section 3.2
- Analyze results by using the functions in `plot_functions.py` to plot the optimized schedule

4 Results and discussion

4.1 Question 1

For Question 1 we assume a simple household with only three appliances: a washing machine, an EV and a dishwasher. Furthermore, the ToU pricing scheme is used and defined with a period of peak hours and off-peak hours.

4.1.1 Main considerations

The ToU pricing scheme is defined in the assignment as:

1 NOK/KWh - Peak hours (17-20)
0.5 NOK/KWh - Off peak hours (00-17, 20-24)

We devised a strategy for the usage of the appliances with regard of specifying which time slots the appliances can be scheduled for use. All assumptions are derived from the total information given in the assignment questions as discussed in Section 3.1. The assumptions for the specific appliances used in this household are:

1. The dishwasher can be used from 12:00 until midnight. A person getting up at 10:00 would eat breakfast first before using the dishwasher.
2. The laundry machine should run when someone is home so that the clothes can be transferred to the dryer or hung to dry.
3. The EV can be away from the home in the daytime should be done charging at 07:00 in the morning. This corresponds to how people in Norway generally charge their EV. [7]

A summary of the assumptions is represented in Table 4, and is a subset of all assumptions from Table 2.

Table 4: The appliances and values for a simple household

Appliances	**Shiftable**	**Length [hours]**	**Daily usage [kWh]**	**Hourly usage [kW]**	**Alpha**	**Beta**
Dishwasher	Yes = 1	3	1.44	0.48	12	24
Laundry machine	Yes	3	1.94	0.6467	10	23
EV	Yes	5	9.9	1.98	0	7

The optimization strategy used is the same as discussed in Section 3.3.

4.1.2 Minimal energy consumption

As seen in Figure 1 the energy usage for the dishwasher and laundry machine are scheduled simultaneously when the energy price is low. From experience we know that we could have distributed the usage and get the same total electricity cost. This illustrates that the LP algorithm only finds the first minimal solution, and that we would need to feed it other constraints in order to get other results.

The total energy bill is given by the solver as optimal value of the objective function: 6.64 NOK

It can also be calculated by using the schedule from Figure 1

$$\begin{aligned}&(1.98 \cdot 5 + 0.6467 \cdot 3 + 0.48 \cdot 3)\ \text{kWh} \cdot 0.5\ \text{NOK/kWh} + 0\ \text{kWh} \cdot 1.0\ \text{NOK/kWh}\\ &= 13.28\ \text{kWh} \cdot 0.5\ \text{NOK/kWh}\\ &= 6.64\ \text{NOK}\end{aligned}$$

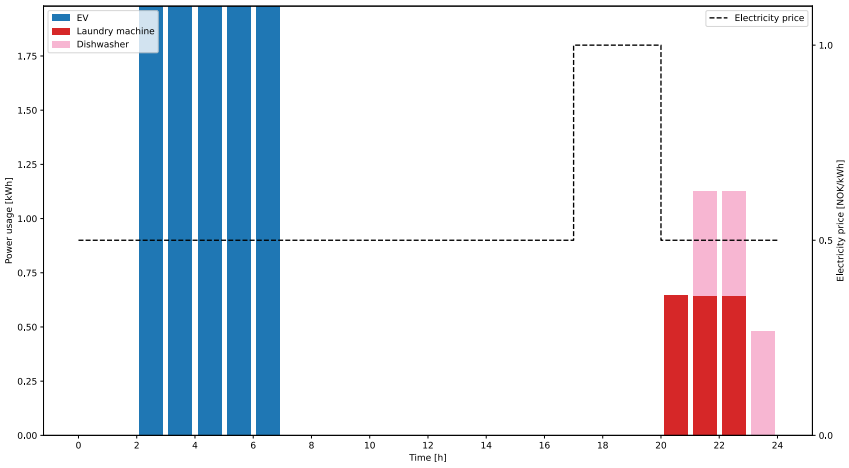

Figure 1: Three appliances for a simple household

4.2 Question 2

For Question 2 we have a household with all non-shiftable appliances, shiftable appliances and some of the semi-shiftable appliances (shown in Table 2). The same LP formulation applies for the demand response discussed in section 3.3. As opposed to Question 1, we will use RTP scheme to generate the pricing curve that considers higher price in the peak-hours throughout the day.

We decided against a flowchart detailing the algorithm here, because the process is very similar for every question, and is described in section 3.4.

4.2.1 Pricing curve

To implement the RTP scheme, the implemented function is defined as follow:

```
def generate_pricing_data():
    pricing = np.zeros(24)
    pricing[0:17] = np.random.uniform(0.45, 0.65, 17)
    pricing[17:20] = np.random.uniform(0.75, 1.0, 3)
    pricing[20:24] = np.random.uniform(0.45, 0.65, 4)
    np.save('data/rt_pricing.npy', pricing)
```

The function considers higher energy prices in the peak hours and lower prices in the off-peak hours. The generative distribution of our random pricing data was determined based on trends in the intraday and day-ahead pricing published by Nord Pool [8]. The result of the function, i.e. the pricing information, was saved for usage in Question 3 and Question 4 as well. The implemented pricing curve can be seen in Figure 2.

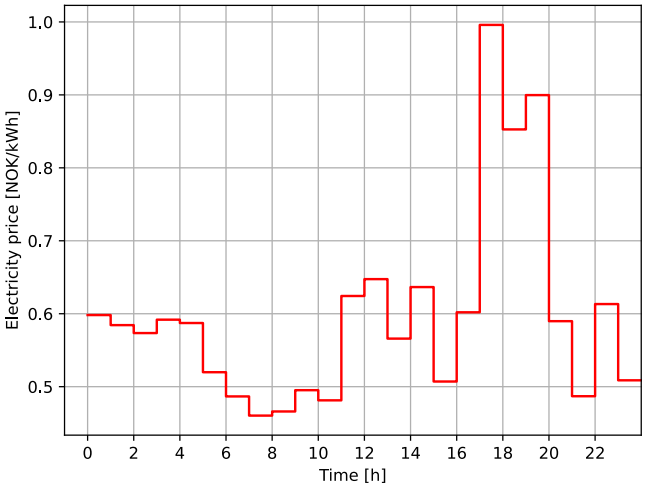

Figure 2: RTP generated by generate_pricing_data()

4.2.2 Combination of appliances

In order to get all the shiftable and non-shiftable appliances, we use the helper function `get_appliances()` and call it with argument `filter_shiftable=1` and `filter_shiftable=0`, respectively. This will extract all the shiftable and non-shiftable appliances available. To get 4 random appliances, the helper function `get_random_optional_shiftable()` was called. This function simply gives us 4 random appliances of the ones with shiftable=2. The household with these appliances was stored in the file `random_households.pkl` so that it can be loaded and reused in Question 4 (section 4.4) or if we needed to rerun our code for Question 2.

4.2.3 Minimal energy consumption

Figure 3 illustrates a random selection of non-shiftable and shiftable appliances which are scheduled for a simple household. The dotted line being the RTP pricing curve, which peaks between 17:00 and 20:00. The total energy consumption/pricing was 18.64 NOK.

Upon examining Figure 3, distinct peaks emerge at hours 10, 15, and 21. To investigate the factors driving these peaks, we visualize the distribution of shiftable and non-shiftable appliances in Figure 4, with non-shiftable appliances represented in blue and shiftable ones in orange.

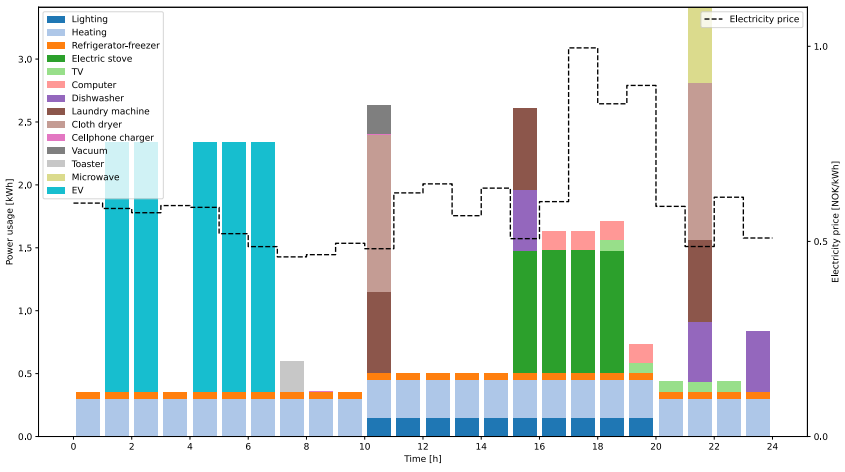

Figure 3: Scheduled non-shiftable and shiftable appliances for a simple household with RTP scheme

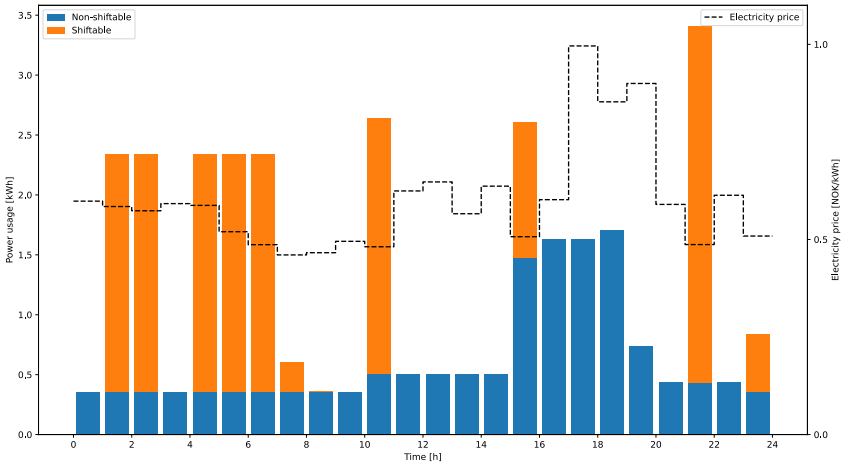

Figure 4: The distribution of shiftable and non-shiftable appliances for a simple household with RTP scheme

During hour 10, the cloth dryer, vacuum, and laundry machine are all scheduled for operation. These appliances are all shiftable and share overlapping operating windows from 10:00 to 23:00, as denoted by α and β. Between hour 15 and hour 18, the electrical stove, specified as non-shiftable in Table 2, is utilized, contributing to increased energy consumption alongside the operation of the dishwasher (shiftable) and laundry machine during hour 15. At hour 21, the dishwasher, laundry machine, cloth dryer, and microwave are all in use. The peaks before 10 are all from the EV charging. Overall, we notice that just three shiftable appliances make up a significant part of electricity usage: The EV, laundry machine and cloth dryer. Shifting these appliances out of peak usage hours contributes a sizable portion of the total energy price savings.

We observe that the power usage near hour 21 is nearing 3.5, which is less than ideal and could potentially strain the grid. To mitigate potential damage to the grid we should avoid unnecessary spikes, known as peak loads. Peak loads will constrain the use of electricity by distributing energy consumption more evenly across the day. This will be discussed in Question 4.

4.3 Question 3

For Question 3 we consider a neighborhood of 30 households. This neighborhood is created as an instance of the class Neighborhood with an integer argument of 30 denoting the number of households. This calls the class function `add_random_households(N)` which is used to add $N = 30$ households with a number of appliances. All non-shiftable and shiftable appliances are added to every household, except for the EV which has a 20% chance to be added. Of the semi-shiftable appliances, only a random subset of 4 appliances are added.

For the formulation of the optimization problem we will use the same principles that has already been established in section 3.2. We will also use RTP scheme discussed in section 4.2.1, and the same settings for a household elaborated in section 4.2, except that just a fraction (around 20%) of the households now owns an EV. Again, we decided against the usage of a flowchart and refer to section 3.4 instead.

4.3.1 Minimal energy consumption

For Question 3, we found it more reasonable to plot the distribution of shiftable and non-shiftable on a general level, as we consider 30 households. Figure 5 illustrate the distribution of the neighborhood.

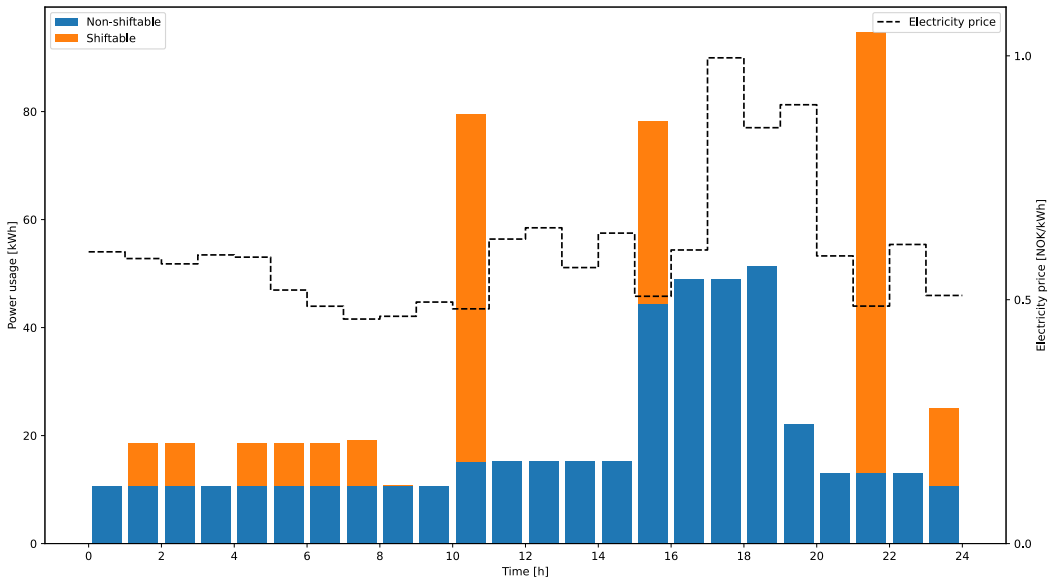

Figure 5: Plotted graph of the shiftable and non-shiftable distribution of a neighborhood with 30 households

The same peaks as we observed in Question 2 occur, but here they are enhanced and thus even stronger. This is due to the fact that all households have the same "mandatory" shiftable and non-shiftable appliances and in addition these are the ones using the most energy throughout the day.

The total energy consumption/pricing was 470.40 NOK, which gives an average of 15.68 NOK per house. This average is less than we observed for a single household with almost the same setting in Question 2. However, since now only a fraction owns an EV and the EV consumes a lot of energy, it makes sense that the average value is lower.

4.4 Question 4

In Question 2 we observed peak loads in hour 10, 15 and 21 (see Figure 3). Such peak loads may result in inefficiencies and grid strain. For Question 4 we want to address the phenomena by introducing a new variable L (previously discussed in section 3.3.3), representing the peak load.

By considering the new constraint L, the total consumption for each respective timeslot should not exceed the peak load L, which was set to 2. To get comparable plots we used the same setting of appliances for Question 4 and for Question 2. The data is stored in the file named `random_household.plk`. As can be seen in the appendix A.8, the appliances are retrieved from this file.

4.4.1 Comparison with results in Question 2

Figure 6 illustrate the scheduling of each appliance with a peak load set to 2.0 KWh. By comparing with Figure 3 the energy consumption profile has slightly changed. For Question 3 we saw three high peaks during hour 10, 15 and 21, as for Question 4 the energy consumption is more spread out through the day.

The total energy consumption/pricing for Question 2: 18.64 NOK
The total energy consumption/pricing for Question 4: 18.94 NOK

Upon comparing the two results, the introduction of L leads to a higher total electricity price for the identical set of appliances under the RTP pricing scheme. We designate L as 2.0, implying a restriction on the total power consumption within each hour to be less than 2.0 kWh. This constraint results in a narrower feasible region, limiting the model's ability to schedule appliances during periods of lowest energy prices.

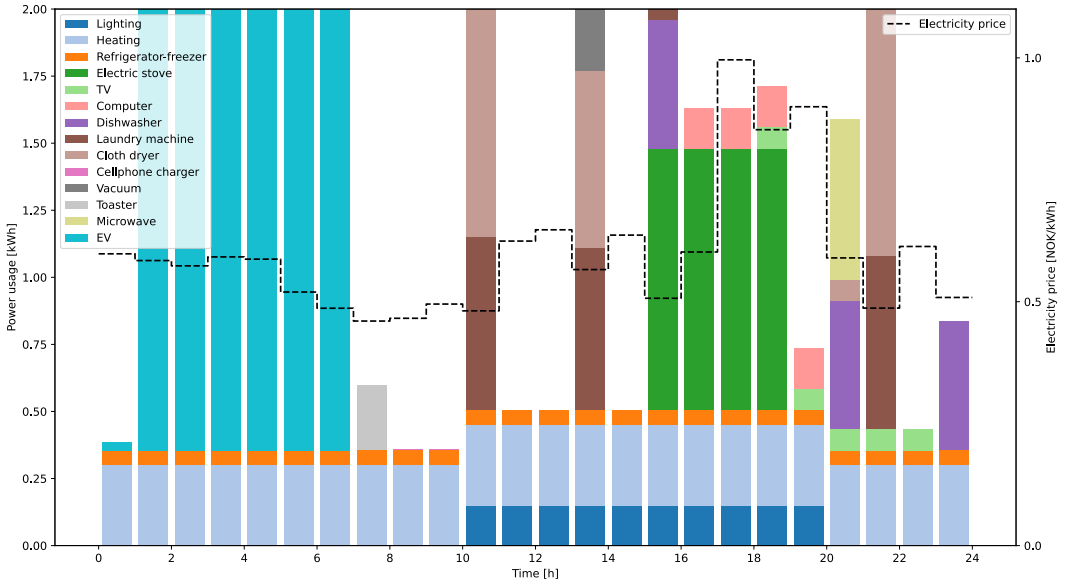

Figure 6: Appliance schedule for a comparable house to Figure 3 with peak load of 2 KWh

4.5 Impact of pricing schemes

To compare the two different pricing schemes ToU and RTP, and how they impact the energy cost we have used the same set of appliances from the file `random_household.plk` with the ToU pricing scheme illustrated in Figure 7.

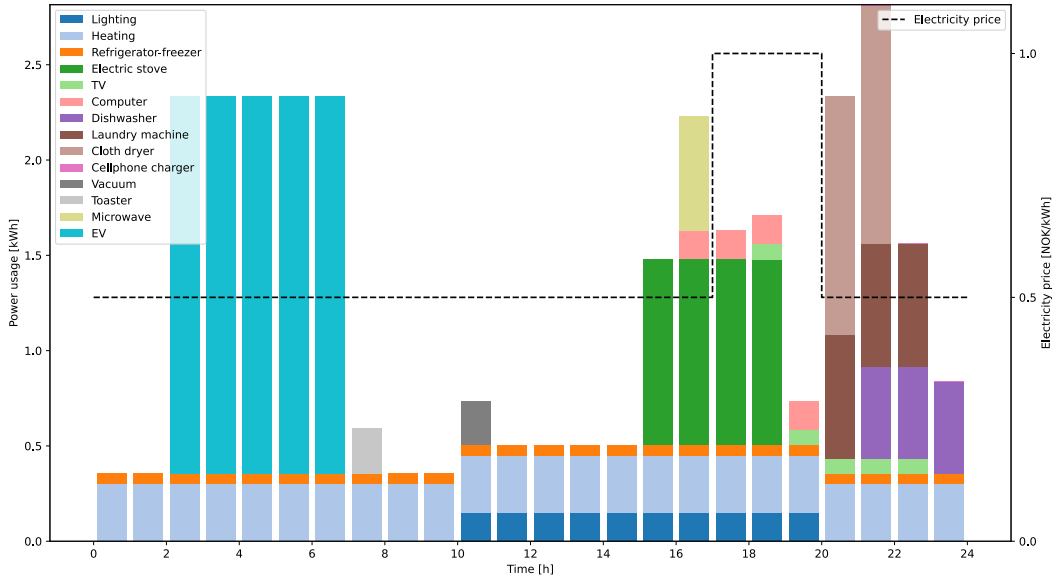

Figure 7: Question 2 results with ToU pricing

The total energy consumption/price when using the ToU pricing scheme (Figure 7): 17.9275 NOK = 31.871 · $\overline{ToU}$
The total energy consumption/price for RTP pricing scheme (Figure 3): 18.64 NOK = 31.124 · $\overline{RTP}$
Here, $\overline{ToU} = 0.5625NOK$ is the average ToU price, and $\overline{RTP} = 0.5989NOK$ is the average RTP price,

both averaged over the 24 hours of pricing information we are considering.

We can observe that, relative to the average price, the total energy consumption price is smaller for the RTP pricing scheme. This confirmed our prior belief that the linear program has more room for optimization there, due to a higher fluctuation of the price.

Conversely, the amount of optimization that is possible for a ToU pricing scheme is very limited, as 21 out of 24 hours have the same price - and all of them are in one contiguous range, that may completely encapsulate the usage hours of some shiftable appliances. Hence, the optimization problem reduces to shifting all loads out of the peak hours, as much as possible.

4.6 Conclusions and reflections

Altogether, we find that the scheduling problem can be solved effectively using our LP formulation, subject to the assumptions arising from formulating the problem in a linear way. The resulting optimal schedules successfully reduce electricity costs of the households. The lower relative cost of the RTP scheme, compared to ToU, confirm our intuition that the higher level of pricing detail allow for more leeway during optimization. Further, we observed that the biggest parts of shiftable consumer energy usage originated from EVs, Laundry machines and cloth dryers. Shifting these appliances out of peak usage hours makes up a signifcant part of the total reduction of the energy bill.

Upon having solved the problem in a rather general way for all tasks, it may have been more instructive to first solve only the simpler problem posed by ToU pricing in Q1. Due to the nature of the ToU pricing curve, a simpler algorithm that does not need to optimize a linear program would have been sufficient, and could have allowed some additional insight into the optimization process.

In hindsight, it would have been better to have different RTP in Q2 and Q3 to illustrate how different the price can be from day to day with this pricing scheme. In general, the results we observe in Q3 are very similar to the ones in Q2, hence Q3 serves mainly as a proof-of-concept that this approach can be scaled to encompass not only one, but several households up to entire neighborhoods.

Future work could consist of investigating different appliance operating times: Different values for the permissible hours in our linear program formulation might display new observations, which could be applied to homes which are inconsistent with our appliance usage assumptions.

Another area of interest is finding a solution to the limitations imposed by our chosen LP formulation, which we discussed in section 3.3.5. Subsequent works could remedy these by replacing the LP with a more general optimization problem, for instance mixed-integer programming, that allows for other constraints such as contiguous usage times. Another beneficiary of more flexible problem formulations could be special appliance cases such as the fridge, which could then be considered as shiftable with an additional temperature constraint instead of nonshiftable.

A Code

A.1 appliance.py

```python
class Appliance():
    def __init__(self, name: str, shiftable: int, usage_h: int, daily_usage_kWh: float, alpha: int, beta: int) -> None:
        """Class that models an appliance and its power usage

        Args:
            name (str): The name of the appliance
            shiftable (int): 0 - if it's non-shiftable, 1 if it's shiftable and 2 if it's an optional appliance (may be shiftable, but not required)
            usage_h (int): Amount of hours it should be used
            daily_usage_kWh (float): The daily usage of the appliance in KWh
            alpha (int): The start time where the appliance can be used
            beta (int): The end time where the appliance can be used

        Raises:
            ValueError: If the appliance data isn't conforming with the constraints
        """
        self.name: str = name
        self.shiftable = shiftable
        self.usage_h: int = usage_h
        self.daily_usage_kWh: float = daily_usage_kWh
        self.hourly_max: float = daily_usage_kWh / usage_h
        self.alpha: int = alpha
        self.beta: int = beta
        if beta - alpha < usage_h:
            raise ValueError(f"Appliance '{name}' is used for {usage_h} hours, but usage window is between {alpha} and {beta}.")
        if shiftable == 0 and beta - alpha > usage_h:
            raise ValueError(f"Appliance '{name}' is not shiftable. Window between {alpha} and {beta} gives a range of {beta - alpha} hours, but usage should be {usage_h} hours.")
        if shiftable == 1 and beta - alpha == usage_h:
            raise ValueError(f"Appliance '{name}' should be shiftable ")
    def __repr__(self) -> str:
        return f'{self.name} ({self.shiftable}, {self.usage_h}, {self.daily_usage_kWh}, {self.alpha}, {self.beta})'
```

A.2 household.py

```python
from appliance import Appliance
from typing import List
import pickle

class Household():

    def __init__(self, name: str) -> None:
        """Creation of a household object

        Args:
            name (str): Name of the house
        """
        self.name: str = name
        self.appliances: List[Appliance] = []
        self.n_appliances: int = 0

    def add_appliances(self, appliances: List[Appliance]) -> None:
        """Function to add a list of appliances to the household

        Args:
```

```
            appliances (List[Appliance]): List of appliances
        """
        self.appliances.extend(appliances)
        self.n_appliances += len(appliances)

    def save(self, path: str):
        """Function to save the household to a file

        Args:
            path (str): The path to save the file
        """
        with open(path, 'wb') as f:
            pickle.dump(self, f)

    @staticmethod
    def load(path: str):
        """Function to load a household from a file

        Args:
            path (str): The path to load the file from

        Returns:
            Household: The loaded household
        """
        with open(path, 'rb') as f:
            return pickle.load(f)

    def __str__(self):
        return f"'{self.name}'(#appliances:{self.n_appliances})"
```

A.3 neighborhood.py

```
import numpy as np
from typing import List, Optional
from scipy.optimize import linprog
import random
from household import Household
from helper_functions import get_appliances, get_random_optional_shiftable, get_pricing


class Neighborhood():
    houses: List[Household] = []
    house_schedules: List[np.ndarray] = []
    optimized_value: bool | float = False
    num_EV: int = 0
    n_households: int = 0
    schedule: np.ndarray
    pricing: np.ndarray
    peak_load: Optional[float]

    def __init__(self, name: str, households: int | List[Household] = 0, pricing:str = "RTP",
    ↪  peak_load: Optional[float] = None) -> None:
        """Function to create a Neighborhood

        Args:
            name (str): The name of a neighborhood
            households (int | List[Household], optional): a list of or number of households in the
            ↪  neighborhood. Defaults to 0.
            pricing (str, optional): The pricing scheme used. Defaults to "RTP".
            peak_load (float, optional): _description_. Defaults to None.

```

```
        Raises:
            ValueError: If the household variable isn't conforming with the constraints
        """
        self.name: str = name
        self.pricing = get_pricing(pricing)
        self.peak_load = peak_load

        if type(households) is int:
            if households < 0:
                raise ValueError("Number of households must be positive.")
            elif households > 0:
                self.add_random_households(households)
        elif all(isinstance(x, Household) for x in households):
            self.add_households(households)
        else:
            raise ValueError(f"'households' must be of type int or List[Households], but is of type
            ↪  {type(households)}")

    def add_households(self, households: List[Household]) -> None:
        """Function to add households to a neigborhood

        Args:
            households (List[Household]): A list of households
        """
        self.houses.extend(households)
        self.n_households += len(households)
        self.optimized_value = False

    def add_random_households(self, num_households: int) -> None:
        """A function to add a random number of random households

        Args:
            num_households (int): The number of random households to add
        """
        nonshiftable_appliances = get_appliances(filter_shiftable=0)
        shiftable_appliances: dict = get_appliances(filter_shiftable=1, output_dict=True)

        # Removes the EV from shiftable appliances so that it can be used to
        ev = shiftable_appliances.pop("EV")

        for i in range(num_households):
            new_house = Household(f"House {i}")

            new_house.add_appliances(nonshiftable_appliances)
            new_house.add_appliances(shiftable_appliances.values())

            if random.random() < 0.2: # 20% chance to get an EV at a house
                new_house.add_appliances([ev])

            optional_appliances = get_random_optional_shiftable()
            new_house.add_appliances(optional_appliances)

            self.add_households([new_house])


    def get_linprog_input(self):
        """Function to get the input for linprog

        Returns:
            _type_: _description_
        """
        c = np.array([])
```

```
        l = []
        u = []
        A_eq = [[]] #matrix
        b_eq = []
        A_ub = None
        b_ub = None
        appliance_counter = 0

        for house in self.houses:
            for appliance in house.appliances:
                c = np.concatenate((c, self.pricing))
                l = np.concatenate((l, [0 for _ in range(24)]))
                u_temp = np.zeros(24)
                A_eq_temp = np.zeros(24)
                for i in range(appliance.alpha, appliance.beta):
                    u_temp[i] = appliance.hourly_max
                    A_eq_temp[i] = 1

                if appliance_counter == 0:
                    A_eq = [A_eq_temp]
                else:
                    A_eq = np.append(A_eq, [[0 for _ in range(24)] for _ in range(appliance_counter)],
                    ↪  axis=1)
                    A_eq = np.append(A_eq, [np.append([0 for _ in range(24*(appliance_counter))],
                    ↪  A_eq_temp)], axis=0)

                u = np.concatenate((u, u_temp))
                b_eq = np.append(b_eq, [appliance.daily_usage_kWh])

                appliance_counter += 1

                if self.peak_load is not None:
                    if A_ub is None:
                        A_ub = np.identity(24)
                    else:
                        A_ub = np.append(A_ub, np.identity(24), 1)
                else:
                    continue
        if A_ub is not None:
            b_ub = [self.peak_load for _ in range(24)]
        return c, u, l, A_eq, b_eq, A_ub, b_ub

    def optimize(self):
        """Function to optimize the schedule using linprog

        Returns:
            _type_: The result of the optimization
        """
        # optimize, use linprog
        c, u, l, A_eq, b_eq, A_ub, b_ub = self.get_linprog_input()
        res = linprog(c, A_ub, b_ub, A_eq, b_eq, [x for x in zip(l,u)])
        self.optimized_value = res.fun
        self.schedule = res.x
        return res

    def get_schedule(self):
        """Function that returns the schedule of the neighborhood

        Returns:
            _type_: _description_
        """
        if self.optimized_value is False:
```

```
            self.optimize()
        return self.schedule

    def _calc_house_schedules(self):
        """Function to calculate the schedules of the households
        """

        if self.optimized_value is False:
            self.optimize()

        previous_index = 0
        for house in self.houses:
            n_appliances = house.n_appliances
            self.house_schedules.append(
                self.schedule[previous_index:previous_index+24*n_appliances].reshape(-1, 24)
            )
            previous_index += 24*n_appliances

    def get_house_schedules(self):
        """Function to get the schedules of the households
        """
        if len(self.house_schedules) == 0:
            self._calc_house_schedules()
        return self.house_schedules
```

A.4 helper_functions.py

```
import pandas as pd
from appliance import Appliance
import numpy as np
from matplotlib import pyplot as plt

def get_appliances(filter_shiftable=None, random_selection_n=None, output_dict=False) ->
↪  list[Appliance] | dict[str, Appliance]:
    """Function that gets all or n random appliances based on a filter and outputs it as either a
    ↪  dictionary or a list

    Args:
        filter_shiftable (int, optional): Filter for if we only want the non-shiftable, semi-shiftable
        ↪  or the shiftable appliances. Defaults to None.
        random_selection_n (int, optional): Value for selecting n random appliances. Defaults to None.
        output_dict (bool, optional): If true the function returns a dictionary and if false a list.
        ↪  Defaults to False.

    Returns:
        list[Appliance] | dict[str, Appliance]: A list or dict containing appliances.
    """

    if output_dict:
        appliances_dict = {}
    else:
        appliances_list = []

    df_appliances = pd.read_excel('data/energy_usage.xlsx')

    if filter_shiftable is not None:
        df_appliances = df_appliances[df_appliances['Shiftable'] == filter_shiftable]

    if random_selection_n is not None:
        df_appliances = df_appliances.sample(n=random_selection_n)

```

```
    for i, row in df_appliances.iterrows():
        appliance = Appliance(row['Appliances'],
                              row['Shiftable'],
                              row['Length [h]'],
                              row['Daily usage [kWh]'],
                              row['Alpha'],
                              row['Beta']
        )
        if output_dict:
            appliances_dict[appliance.name] = appliance
        else:
            appliances_list.append(appliance)
    if output_dict:
        return appliances_dict
    return appliances_list

def get_random_optional_shiftable(n=4):
    """randomly drops some optional appliances to simulate a household

    Args:
        n (int, optional): number of random appliances we want. Defaults to 4.

    Returns:
        list[Appliance] | dict[str, Appliance]: A list or dict containing appliances.
    """
    return get_appliances(filter_shiftable=2, random_selection_n=n)

def generate_pricing_data():
    pricing = np.zeros(24)
    pricing[0:17] = np.random.uniform(0.45, 0.65, 17)
    pricing[17:20] = np.random.uniform(0.75, 1.0, 3)
    pricing[20:24] = np.random.uniform(0.45, 0.65, 4)
    np.save('data/rt_pricing.npy', pricing)

def get_pricing(pricing: str) -> np.ndarray:
    """ A function that fetches the pricing in a numpy array

    Args:
        pricing (str): The type of pricing wanted, either "ToU" (Time-of-Use) or "RTP"
        ↪  (Real-Time-Pricing)

    Raises:
        ValueError: Is raised if value isn't ToU or RTP

    Returns:
        np.ndarray: A numpy array of length 24 containing the pricing data for each hour of the day
    """
    if pricing == "ToU":
        pricing = np.zeros(24)
        pricing[0:17] = 0.5
        pricing[17:20] = 1.0
        pricing[20:24] = 0.5
    elif pricing == "RTP":
        pricing = np.load('data/rt_pricing.npy')
    else:
        raise ValueError("Pricing must be either 'ToU' or 'RTP'.")
    return pricing

if __name__ == "__main__":
    print(get_appliances())
    print(get_appliances(filter_shiftable=2))
    print(get_pricing("ToU"))
```

```
    print(get_pricing("RTP"))

    rtp_pricing = get_pricing("RTP")

    plot_hours = np.concatenate((np.repeat(np.arange(24), 2)[1::], [24]))
    plot_pricing = np.repeat(rtp_pricing, 2)

    plt.plot(plot_hours, plot_pricing, 'r')

    plt.xlim(-1, 24)
    plt.xticks(np.arange(0, 24, 2))

    plt.grid()

    plt.xlabel("Time [h]")
    plt.ylabel("Electricity price [NOK/kWh]")

    plt.show()
```

A.5 question1.py

```
from household import Household
from neighborhood import Neighborhood
from helper_functions import get_appliances
from plot_functions import plot_schedule_appliances, plot_schedule_shiftable_nonshiftable


# appliances
appliance_dict = get_appliances(output_dict=True)
ev = appliance_dict["EV"]
washing_machine = appliance_dict["Laundry machine"]
dishwasher = appliance_dict["Dishwasher"]

# add appliances to the house
our_house = Household("Our first house")
our_house.add_appliances([ev, washing_machine, dishwasher])

# add house to neighborhood and optimize
lonely_neighborhood = Neighborhood("Lonely", pricing="ToU")
lonely_neighborhood.add_households([our_house])
lonely_neighborhood.optimize()

# get results from optimization
cost = lonely_neighborhood.optimized_value
plot_schedule_appliances(lonely_neighborhood, include_house_name=False)
#plot_schedule_shiftable_nonshiftable(lonely_neighborhood)
print(f"The energy bill is {cost:.2f} NOK")
```

A.6 question2.py

```
import numpy as np
from neighborhood import Neighborhood
from household import Household
from plot_functions import plot_schedule_appliances, plot_schedule_shiftable_nonshiftable
from helper_functions import get_appliances, get_random_optional_shiftable


try:
    # if the function has run before, load value
    random_household: Household = Household.load('data/random_household.pkl')
except:
```

```
    random_household: Household = Household("Our second house")

    # add all shiftable and non-shiftable appliances
    nonshiftable_appliances = get_appliances(filter_shiftable=0)
    shiftable_appliances = get_appliances(filter_shiftable=1)

    # add a random combination of optional appliances
    optional_appliances = get_random_optional_shiftable()

    random_household.add_appliances(nonshiftable_appliances + shiftable_appliances +
    ↪  optional_appliances)

    random_household.save('data/random_household.pkl')

finally:
    lonely_richer_neighborhood = Neighborhood("Another lonely", pricing="RTP")
    lonely_richer_neighborhood.add_households([random_household])
    lonely_richer_neighborhood.optimize()

    plot_schedule_appliances(lonely_richer_neighborhood, include_house_name=False)
    plot_schedule_shiftable_nonshiftable(lonely_richer_neighborhood)

    cost = lonely_richer_neighborhood.optimized_value
    print(f"The energy bill is {cost:.2f} NOK")
```

A.7 question3.py

```
import numpy as np
from plot_functions import plot_schedule_appliances, plot_schedule_shiftable_nonshiftable
from neighborhood import Neighborhood


n_households = 30

crowded_neighborhood = Neighborhood(name="New neighborhood", households=n_households, pricing="RTP")
res = crowded_neighborhood.optimize()

# plot_schedule_appliances(crowded_neighborhood)
plot_schedule_shiftable_nonshiftable(crowded_neighborhood)

cost = crowded_neighborhood.optimized_value
print(f"The energy bill is {cost:.2f} NOK")
```

A.8 question4.py

```
from neighborhood import Neighborhood
from household import Household
from plot_functions import plot_schedule_appliances, plot_schedule_shiftable_nonshiftable


try:
    random_household = Household.load('data/random_household.pkl')

except:
    raise AssertionError("Question 2 must be run first")

else:
    neighborhood_peak = Neighborhood("Neighborhood with peak", pricing="RTP", peak_load=2.0)
    neighborhood_peak.add_households([random_household])

    res = neighborhood_peak.optimize()
```

```
17
18      plot_schedule_appliances(neighborhood_peak, include_house_name=False)
19      plot_schedule_shiftable_nonshiftable(neighborhood_peak)
20
21      cost = neighborhood_peak.optimized_value
22      print(f"The energy bill is {cost:.2f} NOK")
```

References

[1] *Optimization and root finding (scipy.optimize) — SciPy v1.12.0 Manual*. URL: https://docs.scipy.org/doc/scipy/reference/optimize.html (visited on 03/07/2024).

[2] *scipy.optimize.linprog — SciPy v1.12.0 Manual*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html (visited on 03/07/2024).

[3] Sarah Dillemuth. *What does Time-of-Use pricing mean on my electricity bill? - MCE*. en-US. Aug. 2020. URL: https://www.mcecleanenergy.org/mce-news/4-things-you-need-to-know-about-time-of-use-pricing/ (visited on 03/07/2024).

[4] *Real-Time Pricing (RTP)*. en-US. URL: https://dealhub.io/glossary/real-time-pricing/ (visited on 03/07/2024).

[5] *Estimating Appliance and Home Electronic Energy Use*. en. URL: https://www.energy.gov/energysaver/estimating-appliance-and-home-electronic-energy-use (visited on 03/14/2024).

[6] *Electricity usage of a Coffee Maker - Energy Use Calculator*. URL: https://energyusecalculator.com/electricity_coffeemaker.htm (visited on 02/22/2024).

[7] Jan-Ivar Bjerke. *Nordmenn flest lader smart – og sparer penger samtidig*. nb-NO. Sept. 2023. URL: https://elbil.no/nordmenn-flest-lader-smart/ (visited on 03/17/2024).

[8] *Nordpool - day-ahead Prices*. July 2024. URL: https://data.nordpoolgroup.com/auction/day-ahead/prices?deliveryDate=latest&deliveryAreas=NO1,NO2,NO3,NO4,NO5&currency=NOK&aggregation=Hourly.