En AI-driven strategi för att underlätta efterfrågeflexibilitet inom elmarknader

Kalle Lahtinen

Kandidatavhandling i datateknik
Handledare: Jerker Björkqvist
Fakulteten för naturvetenskap och teknik
Åbo Akademi
2023

**Abstrakt**

TODO

**Innehållsförteckning**

s. 1 Introduktion

Belastande faktorer för elnätet

Efterfrågestyrning

AMI

CNNs

Modellering

Resultat

Diskussion

**1. Introduktion**

Elektricitet som handelsvara kännetecknas av en speciell egenskap som påverkar levereringsprocessen: eftersom det finns inget sätt att lagra el på stor skala [SRC] måste den alltid levereras till konsumenterna i realtid. Efterfrågan och utbudet av elektricitet måste alltid balanseras så att kvaliteten (frekvensen och spänningen) inte ändras, vilket innebär att varje ökning i konsumtion måste följas av en motsvarande ökning i produktionen. Traditionellt har utbudet betraktats som den enda av de två som kan påverkas, men på grund av vissa faktorer som diskuteras i kapitel 2 har idéen att styra efterfrågan börjat vinna mark.

Efterfrågerespons (eng. demand response) är ett föreslaget system som är avsett för att upprätthålla balansen och stabiliteten i elnätet genom att antigen öka eller minska efterfrågan av energi med hjälp av ekonomiska incitament [SRC]. En del av detta kan styras direkt av nätoperatören enligt ett avtal eller indirekt i form av en konsument som reagerar på en prissignal. En flexibel efterfrågan gynnar nätoperatören i och med att balanseringen blir enklare och att stora konsumtionspikar kan undvikas, vilket i sin del minskar behovet att förstärka nätet och därmed investeringskostnaderna []. Också konsumenten nyttas av en potentiellt stabilare elförsörjning och framför allt minskade elräkningar och överföringsavgifter. Dessutom kan efterfrågestyrning främja en högre utbredning av oregelbundna förnybara energikällor liksom vind- och solkraft [SRC], vilket har sin betydelse i kampen mot klimatförändringen.

Arbetsgruppen för smarta elnät (*fi. Älyverkkotyöryhmä*), som bildades av Arbets- och Näringsministeriet beskriver i sin slutrapport efterfrågeflexibilitet som en väsentlig del av framtida smarta elnät. Enligt rapporten för att kunna åstadkomma måste IT-infrastrukturen kring elnät förnyas och nya modeller. [src]

I denna avhandling kommer efterfrågeflexibilitet betraktas genom en modell som beskriver hur mekanismer kunde automatiseras på lokalnivå.

**2 Belastande faktorer för elnätet**

**2.1 Vindkraft**

Vindkraftsproduktionen i Finland växer i snabb takt. Trots att inmatningstariffsystemet för vindkraft avskaffades för ny vindkraft år 2017 [src], har vindkraftskapaciteten nästan tredubblats sedan dess – till och med 75% bara under år 2022 [src]. Enligt Finlands Vindkraftsförening kommer kapaciteten fortsätta öka markant under de kommande åren, upp till så mycket som 30 TWh år 2030 vilket motsvarar en mer än 10 gångers ökning i 10 år. [src] Denna utveckling är positiv ur ett klimatperspektiv, eftersom vindkraften eventuellt ersätter fossilintensiva produceringsmetoder [?]. Eftersom vindkraften producerar el oregelbundet väcks dock frågan om hur balansen i elnätet upprätthålls med högre andelar av vindkraft. Vindkraftverk kan även förbjudas att mata in elektricitet när det blåser i tider av låg elkonsumtion [src].

**2.2 Elbilar**

Elektrifieringen av trafiken är också en utveckling som sker snabbt och har potential att påverka elkonsumtionsprofilen. Från år 2018 till 2022 har antalet fullelektrifierade personbilar ungefär dubblats varje år i Finland. En motsvarande ökning varje år fram till 2030 skulle mena att det kommer att finnas över 800 000 elbilar. [src] Det är svårt att förutse när bilägare tenderar att ladda sina bilar i framtiden, men om det överlappar med timmar med hög elkonsumtion sker det en betydlig ökning i toppbelastningen, vilket eventuellt menar att stora delar av elnätet måste förstärkas.

**2.3 Nätinfrastruktur**

Finland elektrifierades med snabb takt under 1960-talet. Idag är ännu en betydande del av infrastrukturen från denna tid och måste snart förnyas. Elnätsinvesteringar är dyra och görs långsamt över decennier [src]. Ökande toppbelastning leder till att bland annat ledningarna måste förstärkas för att kunna tåla mer effektöverföring. Detta kan leda till behov av väldigt stora investeringar på relativt kort sikt.

Ett annat problem är att nätet var ursprungligen inte avsedd att kunna handla stora mängder distribuerad produktion. Detta beror på att kraft producerades traditionellt i enstaka stora produktionsanläggningar och sedan överfördes till konsumenterna [src]. Om distribuerade produktionen fortsätter växa, kan detta också leda till storinvesteringar.

**2.4 Lagring**

Det finns inte tillräckligt lagringskapacitet för att kunna spara överskottselektricitet som kunde annars användas i tider av låg produktion från varierande källor.

**3. Efterfrågeflexibilitet**

Efterfrågeflexibilitet är en strategi för att på kort sikt påverka efterfrågan, antingen genom ekonomiska incitament eller avtal mellan leverantör och kund, för att hjälpa till att balansera belastningen [2, s.281]. Begreppet skall inte förväxlas med efterfrågestyrning, vilket är en mer långsiktig hantering av belastningsprofiler [2, s.281].

Ett exempel på efterfrågeflexibilitet kunde vara byggnader som fungerar som värmelager under låga elförbrukningsförhållanden och sänker sin uppvärmning något under perioder med hög elförbrukning. [source]

V2G (vehicle-to-grid) är en mekanism där elbilar använder sin laddning för att mata in elektricitet till elnätet när det behövs. [source]

Efterfrågeflexibilitetsprogram är uppdelade i två kategorier: belastningsresponsprogram (?) (eng. load response program) och prisresponsprogram (eng. price response program) [2, s.274].

Belastningsresponsprogram är en typ av program där elbolag gör avtal med kunder för att minska deras energiförbrukning under högbelastningsperioder. Dessa program innebär vanligtvis att elbolaget erbjuder betalningar till kunder som är villiga att minska sin energiförbrukning under specifika tider eller under vissa förhållanden. Det kan också innebära att elbolaget får direkt kontroll att under vissa förutsättningar styra energiförbrukningen för vissa apparater. [1, s. 276] Ett exempel på detta skulle vara om elbolaget har ett avtal om att automatiskt styra smarta termostater i tusentals hushåll och sänka temperaturen inomhus inom en överenskommen gräns mot en ersättning.

Prisresponsprogram ger å andra sidan kunder prisignaler, som uppmuntrar dem att justera sin elförbrukning under tider när belastningen på elnätet är hög eller det finns ett överskott av elektricitet. I utbyte mot att minska sin efterfrågan kan kunderna få ekonomiska incitament, såsom lägre elpriser. [1, s. 276] Med hjälp av prisinformationen kan användarna antingen reagera på prissättningen manuellt eller definiera inställningar som automatiskt stänger av eller påverkar konsumtionen av deras apparater.

Båda typerna av demand response behövs, eftersom de har potential att komplettera varandra. Om exempelvis prisrespons är föredraget för verkställigheten, till exempel på grund av kostnad, kan det förlita sig på prisrespons för största delen av sin demand response och täcka resten med direkt kontroll.

AMI

För att implementera belastnings- och prisresponsprogram är det viktigt att etablera effektiva kanaler för dataflöden mellan företag och kunder, samt ett pålitligt system för att hantera data. Avancerad mätningsinfrastruktur (?) (eng. advanced metering infrastructure, AMI) är en omfattande lösning som använder olika teknologier, inklusive smarta mätare som kan kommunicera tvåvägs (?), för att ge både kunder och elbolaget detaljerad information om elförbrukning och priser. Denna data inkluderar exakt tid och mängd av el som används, bland andra mått. [1, s. 18]

[TODO: översätt]

AMI will provide the following functionalities essential for enabling demand response [1, s. 18]:

1. Ability to convey pricing information to the consumer. This could even involve an interactive display of some kind [2, s. 218].
2. Ability to process and report energy consumption data to the utility and possibly third parties that participate in the demand response market [SRC].
3. Ability to measure load characteristics, even reporting failures and outages.
4. Remote controlled connection and disconnection.

This communication infrastructure has very harsh requirements to ensure that the communication is reliable and has low-enough latency for each given task [2, s.221], and that it is secure enough to resist unauthorized access or compromising customer privacy [2, s. 220]. Moreover, the amount of data produced is massive and must be handled in an efficient way [2. s.249-250].

A proposed communication infrastructure is one that consists of HANs, NANs, and a WAN. A HAN connects a smart meter and appliances within a household. A NAN is formed by units that aggregate data additional smart meters in a neighborhood, with metering data ultimately gathered by the MDMS through a high-speed WAN. The MDMS stores, manages, and processes meter data for use by other power system applications and services.

Konvolutionella neurala nätverk

Konvolutionella neurala nätverk (eng. convolutional neural network, CNN) är ett typ av artificiellt neuralt nätverk som används för bildklassificering. De består av lager, varav vissa utför matematiska operationer som kallas konvolutioner för att filtrera ut information i indata. Dessa konvolutionslager är uppbyggda av individuella enheter som kallas neuroner [2, Considering the Many Different Roads to Learning]. [1, s.205-210] Varje lager har 2 nyckelparametrar: kärnan och aktiveringsfunktionen. [1, s.210]

[ TODO: Förklara konvolution]

Kärnan är formen på delmängden av indatavärden som inkluderas åt gången i konvolutionsoperationen [1, s.210-211]. Konvolutionen kan till exempel ta in 3 värden åt gången och producera ett utdatavärde baserad på dem. För ett 2-dimensionellt svartvitt bild kan konvolutionsoperationen utföras till exempel en 2x2 kvadrat i taget.

Aktiveringsfunktioner är matematiska operationer som används för att applicera en icke-linjär transformation på utdata från individuella neuroner. De kan användas för att ställa ett villkor för när neuronen skall aktivera – det vill säga producera utdata. [2, Hitting Complexity with Neural Networks]

Konvolutionella neurala nätverk använder ofta något som kallas för pooling, vilket delar upp indatan i lika stora delar och summerar värdena, till exempel genom att beräkna genomsnittet. Detta görs för att minska dimensionerna på indatan, vilket minskar det totala behovet av beräkningar. Ett exempel på pooling är maxpooling, som delar upp indatan och ger det högsta värdet i varje del. [1, s. 218]

[ TODO: förklara loss functions]

Modellering

Genom att använda datapunkter som samlats in av AMI kommer elnätverksoperatorer och andra aktörer som är ansvariga för att utföra efterfrågeresponsen att behöva modellera den tidigare, nuvarande och kommande statusen för varje nod i nätverket. Några av de saker som kan behövas för efterfrågerespons inkluderar:

* Vad är den nuvarande belastningssituationen i varje nod och hur förväntas den utvecklas under de kommande timmarna eller dagarna? Vad gäller produktionen?
* Närmar sig någon nod maximal konsumtion?
* Finns det någon nod som verkar ha förändrats tillräckligt mycket att förbrukningsmodellen bör uppdateras?
* Om det finns storskalig variabel produktion, som en vindkraftpark, som är ansluten till en viss nod i elnätet, vilken potential har lasterna i noden att absorbera överskottsproduktionen som för överföringskapacitetsskäl inte kan exporteras från noden?
* Vilka är referensvärdena för att automatiskt ge ekonomiska incitament?
* Hur förväntas en given nod eller kund reagera på incitament?
	+ Vilket är sambandet mellan storleken på incitamentet och reaktionen?
	+ Till vilken del finns finns det direkt stryning av laster och hur användningen av den och prisresponsen balanseras?

The model put forward in this thesis describes a part of what could be in a system in a NAN. The code is provided in appendix A. The node has a set number of customers, a maximum demand that it can withstand and some amount of distributed generation.

Each customer has a number of consumption patterns that roughly describe their consumption in a given day. Associated with each customer is also a machine learning algorithm which classifies which consumption pattern best describes the consumption so far in the day. Moreover, each customer has a preprogrammed response model that describes how much the customer will reduce consumption at a given reward at different consumption levels.

The model starts at the start of the day, and advances in 15minute intervals. At every time interval, the model does the following three things:

1. It tries to predict each customers consumption pattern based on materialized consumption.
2. It determines when there might be need for demand response and how much.
3. It determines a minimized cost that produces the desired amount of demand response for the times when it is needed.

The produced information could then hypothetically be used to automatically make incentives for clients.

**The consumption models**

Each customer has been 10 different hypothetical consumption patterns that roughly indicate how the customer will behave within the timespan of a given day. These simulate what would really be gathered from observing customer behavior over a longer time-period. At the start of each day, each customer is assigned one of these models at random. As the consumption materializes throughout the day, a random component of +-10% is also added to it. Some examples of consumption models is given in figure 1.



a)

b)

c)

Figur 1: examples of different consumption patterns

**The machine learning algorithm**

The neural network consists of two convolutional layers each followed by a by a max pooling layer, then a flattening layer, a fully connected layer and finally an output layer.

The convolutional layer’s job is to discern patterns in the inputs. The pooling layers reduce the size of the inputs. The flattening layer puts the neuron data in the right format for the fully connected layer. Finally, there is an output layer with a softmax activation function that gives probabilities of each category.

**The response model**

**Determining the need for demand response**

**Optimizing the cost of the demand response**

RESULTS

DISKUSSION

The model that is used in this thesis gives a good base insight to how demand response could be to a degree automated on a neighborhood level. Given more data and research, aspects of the model could be made more realistic. There are also room for bettering some aspects within the scope of the existing model. Here are some of them:

It could have been useful to use a larger variety of pattern types, for example modeling a base load scenario for when the customer is gone. This would make the sample data that the machine learning algorithm is trained on more diverse, which would in turn affect how well prediction works.

In the part where the demand response is optimized there could have been a mix of direct control and price response.

The model for local generation could be expanded on so that it would actually play a role in how much demand response is needed.

The length of the patterns could have been shortened to 3-6 hours.

Include consumption patterns for entire nodes.

Here are some of the questions for the future that could help expand on the model:

* What is a good length for a consumption pattern? A day is a fairly long time, but if the pattern was one hour (4 measurements) there would be too many patterns to confuse between each other. Perhaps this could still work in a different model that calculates expectation values over a larger pool of customers.
* What is an optimal amount of consumption models? This highly depends on how much consumption actually varies from day to day.
* Can consumption actually be categorized into patterns, or does it vary too much for predicting behavior?
* Do different clients tend to have similar enough consumption patterns to be considered one?
* How does demand response itself affect the predictability of customers? To what degree does demand shift to later hours? How would you detect that?
* How could local temperature or weather readings be incorporated into the model? Perhaps even smart thermometers in customers homes could be considered.
* When customer behavior cannot be very well predicted, could price response offers be done based on some other tendency?

Källor

[1] Smart Grids roadmap

<https://www-oecd-ilibrary-org.ezproxy.vasa.abo.fi/docserver/9789264115071-en.pdf?expires=1679331265&id=id&accname=ocid177530&checksum=1BDD703BEE103DED6A5DD8F8110A03D6>

[2] Smart grids and enabling technologies

<https://ieeexplore-ieee-org.ezproxy.vasa.abo.fi/xpl/ebooks/bookPdfWithBanner.jsp?fileName=9509795.pdf&bkn=9509795&pdfType=book>

[3] Rothman, Denis. Artificial Intelligence by Example : Acquire Advanced AI, Machine Learning, and Deep Learning Design Skills, Packt Publishing, Limited, 2020. ProQuest Ebook Central, [https://ebookcentral-proquest-com.ezproxy.vasa.abo.fi/lib/abo-ebooks/detail.action?docID=6126529](https://ebookcentral-proquest-com.ezproxy.vasa.abo.fi/lib/abo-ebooks/detail.action?docID=6126529%20)

[4] Mueller, J. P., & Massaron, L.. (2019). Deep learning (1st edition.). For Dummies.

<https://2masteritezproxy.skillport.com/skillportfe/assetSummaryPage.action?assetid=RW$11243:_ss_book:145072>

APPENDIX A: Code

consupmtion\_model\_recognizer.py

# hyperparameters

NUM\_EPOCHS = 10

BATCH\_SIZE = 32

VALIDATION\_SPLIT = 0.2

# minimum confidence so that prediction is considered valid

MINIMUM\_CONFIDENCE = 0.9

"""

Convolutional Neural Network (CNN) with max pooling meant to recognize

Different consumption patterns

"""

class ConsumptionModelRecognizer:

    def \_\_init\_\_(

      self,

      input\_data\_dimension,

      output\_data\_dimension

    ):

      model = Sequential()

      # 1D convolutional layer

      model.add(

        Conv1D(

          filters=32,

          kernel\_size=3,

          activation='relu',

          input\_shape=(input\_data\_dimension,1)

        )

      )

      # Max pooling layer

      model.add(MaxPooling1D(pool\_size=2))

      # Another 1D convolutional layer

      model.add(Conv1D(filters=64, kernel\_size=3, activation='relu'))

      # Another max pooling layer

      model.add(MaxPooling1D(pool\_size=2))

      # Flatten the output from the convolutional layers

      model.add(Flatten())

      # Fully connected layer with a ReLU activation function

      model.add(Dense(128, activation='relu'))

      # Output layer with a softmax activation function

      model.add(Dense(output\_data\_dimension, activation='softmax'))

      # Compile the model

      model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

      self.model = model

    def train(self, X, Y):

      history = self.model.fit(

        X,

        Y,

        epochs=NUM\_EPOCHS,

        batch\_size=BATCH\_SIZE,

        validation\_split=VALIDATION\_SPLIT

      )

      # Checking accuarcy at the end of the training

      accuracy = history.history['val\_accuracy'][-1]

      print('Test accuracy:', accuracy )

    def predict(self, X):

      Y = self.model.predict(X)

      model\_index = np.argmax(Y)

      confidence = Y[model\_index]

      return (model\_index, (confidence > MINIMUM\_CONFIDENCE))

grid\_model.py

import numpy as np

import matplotlib.pyplot as plt

from customer import Customer, QUARTER\_HOURS\_PER\_DAY, CONSUMPTION\_RANDOMNESS

from price\_response\_optimizer import optimize\_price\_response

# price response will be conducted a maximum of 6 hours ahead of time

MAX\_PRICE\_RESPONSE\_INTERVAL = 6\*4

NUM\_CUSTOMERS = 5

# generate customers, in an actual case these would be retrieved from a database

customers = [

   Customer() for i in range(NUM\_CUSTOMERS)

]

# in this scenario, the maximum demand that this node can withstand is

# 80% of the sum of all customer maximum consumptions. This also includes a margin of safety.

MAX\_DEMAND = np.sum([customer.max\_consumption for customer in customers])

# note: only for demonstration purposes

def get\_time():

    # the time can be anywhere between the first quarter and the second last quarter

    return np.random.randint(0, QUARTER\_HOURS\_PER\_DAY-1)

def get\_predictable\_customers(customers):

    # customers whose behavior at this point can be predicted

    predictable\_customers = []

    for customer in customers:

        if (customer.predictable):

            predictable\_customers.append((customer, model\_index))

    return predictable\_customers

def get\_estimated\_demand(customers, time):

    demand = 0

    for customer in customers:

        if customer.predictable:

            demand += customer.get\_predicted\_consumption(time) + CONSUMPTION\_RANDOMNESS

        else:

            demand += customer.get\_default\_consumption(time)

    return demand

def get\_predicted\_local\_generation(time):

    # TODO: expand on this model

    return 0.2 \* MAX\_DEMAND

# NOTE: if the amount of unpredictable customers is large

def get\_needed\_price\_responses(customers, current\_time):

    time\_limit = current\_time+MAX\_PRICE\_RESPONSE\_INTERVAL

    if time\_limit > QUARTER\_HOURS\_PER\_DAY:

        time\_limit = QUARTER\_HOURS\_PER\_DAY

    needed\_price\_responses = []

    for quarter in range(current\_time+1, time\_limit):

        demand = get\_estimated\_demand(customers, quarter)

        # check for potential overloading

        if (demand - get\_predicted\_local\_generation(quarter) > MAX\_DEMAND):

            needed\_price\_responses.append((MAX\_DEMAND - demand, quarter))

        # TODO: check for other needed

def get\_price\_response\_cost(amount, time, customers):

    min\_price\_responses = []

    max\_price\_responses = []

    for customer in customers:

        if not customer.predictable:

            continue

        min\_consumption = customer.get\_predicted\_consumption(time) - CONSUMPTION\_RANDOMNESS/2

        max\_consumption = customer.get\_predicted\_consumption(time) + CONSUMPTION\_RANDOMNESS/2

        # NOTE: relies on the price response being a growing function, arguably a fair assumption

        min\_price\_response = customer.generate\_price\_response(min\_consumption)

        min\_price\_responses.append(min\_price\_response)

        max\_price\_response = customer.generate\_price\_response(max\_consumption)

        max\_price\_responses.append(max\_price\_response)

    result = optimize\_price\_response(min\_price\_responses, amount)

    result = optimize\_price\_response(max\_price\_responses, amount)

"""

The day starts, each customer starts consuming electricity following a consumption model.

"""

# time measured in quarter hours

time = get\_time()

print("Hello! The time is " + time)

# start new day for all customers and set the time

for customer in customers:

    customer.set\_time(time)

for (amount, time) in get\_needed\_price\_responses(customers):

    (min\_cost, max\_cost) = get\_price\_response\_cost(amount, time, customers)

    decide\_price\_response()

price\_model\_optimizer.py

import numpy as np

from scipy.optimize import minimize

def optimize\_price\_response(price\_response\_models, desired\_output):

  # The objective function is the sum of the inputs, ie. the total cost

  def objective(costs):

      return np.sum(costs)

  # Define the constraint function

  def constraint(costs):

      sum = -desired\_output

      sum += np.sum([price\_response\_models[i](costs[i]) for i in range(len(costs))])

      return sum

  # Define the bounds for the input variables

  bounds = [(0, 100) for i in range(len(price\_response\_models))]

  # Define the optimization problem

  problem = {'objective': objective,

            'constraints': [{'type': 'eq', 'fun': constraint}],

            'bounds': bounds}

  # Solve the optimization problem

  result = minimize(\*\*problem)

  return result

customer.py

import numpy as np

import matplotlib.pyplot as plt

from consumption\_model\_recognizer import ConsumptionModelRecognizer

QUARTER\_HOURS\_PER\_DAY = 24\*4

NUM\_CONSUMPTION\_MODELS = 15

GLOBAL\_MAX\_CUSTOMER\_CONSUMPTION = 200

CONSUMPTION\_RANDOMNESS = 20

PRICE\_RESPONSE\_RANDOMNESS = 10

# TODO: ratio of shifting consumption as opposed to reducing it

SHIFTING\_FACTOR = 70

REDUCTION\_FACTOR = 100 - SHIFTING\_FACTOR

class Customer:

    def \_\_init\_\_(

      self

    ):

      # the max consumption of this customer is between 50-100% of the highest possible consumption of a customer

      self.max\_consumption = GLOBAL\_MAX\_CUSTOMER\_CONSUMPTION \*( np.random.rand()\*0.5 + 0.5)

      self.num\_consumption\_models = NUM\_CONSUMPTION\_MODELS

      self.consumption\_models = [self.\_generate\_consumption\_model() for i in range(self.num\_consumption\_models)]

      self.price\_response\_model = self.\_generate\_price\_response\_model(),

      self.consumption\_model\_recognizer = self.\_generate\_consumption\_model\_recognizer(),

      (self.consumption, self.model\_index) = self.\_generate\_consumption()

      self.default\_consumption = self.\_generate\_default\_consumption()

      self.time = 0

      self.\_predict()

    def \_predict(self):

      materialized\_consumption = self.consumption.reshape(-1, QUARTER\_HOURS\_PER\_DAY, 1)

      materialized\_consumption[self.time:] = 0

      prediction = self.consumption\_model\_recognizer.predict(materialized\_consumption)

      self.predictable = prediction[0]

    def \_generate\_consumption(self):

      model\_index = np.random.randint(0,  self.num\_consumption\_models)

      consumption = self.consumption\_models[model\_index] + CONSUMPTION\_RANDOMNESS\*(np.random.rand()-0.5)

      return (consumption, model\_index)

    def \_generate\_default\_consumption(self):

      # the curve has a base component of 20

      curve = np.zeros(QUARTER\_HOURS\_PER\_DAY) + 20

      # Define the start and end indices of the sine component

      start\_index = 6 \* 4  # 6 am

      peak\_index = 9 \* 4  # 9 am

      end\_index = 12 \* 4  # 12 pm

      # Define the x values for the sine component

      x\_values = np.arange(start\_index, end\_index)

      # Define the sine component as a function of the x values

      sine\_component = np.sin((x\_values - peak\_index) \* np.pi / 24)

      # Add the sine component to the zeros array at the appropriate indices

      curve[start\_index:end\_index] += sine\_component

      # Print the resulting array

      print(curve)

      return curve

    """

      sets consumption at random using one of the models

    """

    def set\_time(self, time):

      self.time = time

      self.\_predict()

    def get\_predicted\_consumption(self, quarter):

      return self.consumption\_models[self.model\_index][quarter]

    def get\_default\_consumption(self, quarter):

      return self.default\_consumption[quarter]

    def \_generate\_consumption\_model(self):

      # the number of sine components in the consumption function

      num\_subcomponents = 10

      # the max size for each component

      max\_component\_size = (100-CONSUMPTION\_RANDOMNESS/2) / num\_subcomponents

      component\_sizes = np.random.rand(num\_subcomponents) \* max\_component\_size

      max\_component\_speed = np.pi/3

      component\_speeds = np.random.rand(num\_subcomponents) \* max\_component\_speed

      max\_component\_phase = 2\*np.pi

      component\_phases = np.random.rand(num\_subcomponents) \* max\_component\_phase

      x = np.linspace(0,QUARTER\_HOURS\_PER\_DAY-1, QUARTER\_HOURS\_PER\_DAY)

      consumption = np.zeros\_like(x)

      for i in range(num\_subcomponents):

          consumption += component\_sizes[i]\*(

              np.sin(component\_speeds[i] \* x + component\_phases[i]) + 1

          )

      consumption \*= self.max\_consumption

      plt.plot(x, consumption)

      plt.show()

      return consumption

    def \_generate\_consumption\_model\_recognizer(self):

      # TODO

      set\_size = 100

      X = np.random.rand(set\_size, QUARTER\_HOURS\_PER\_DAY)

      Y = np.random.rand(set\_size, self.num\_consumption\_models)

      for i in range(set\_size):

          (consumption, model\_index) = self.\_generate\_consumption()

          X[i, :] = consumption

          output\_vector = np.zeros( self.num\_consumption\_models)

          output\_vector[model\_index] = 1

          Y[i, :] = output\_vector

      model = ConsumptionModelRecognizer(

          QUARTER\_HOURS\_PER\_DAY,

           self.num\_consumption\_models

      )

      model.train(X, Y)

      print(type(model))

      return model

    def \_generate\_price\_response\_model(self):

      x = np.linspace(0, 99, 100)

      # a\*100\*\*2 = 1 => a = 1/100\*\*2

      relation\_to\_consumption = x\*\*2 \* 1/100\*\*2

      relation\_to\_price = np.log2(x+1) + (np.cos(x)+1)

      relation\_to\_price = relation\_to\_price / relation\_to\_price.max()

      relation\_to\_price = np.sort(relation\_to\_price)

      price\_response\_model = lambda consumption : (

          relation\_to\_price

          \* relation\_to\_consumption[consumption]

          \* (100 - PRICE\_RESPONSE\_RANDOMNESS)

      )

      plt.plot(x, price\_response\_model(80))

      plt.show()

      plt.plot(x, price\_response\_model(60))

      plt.show()

      plt.plot(x, price\_response\_model(40))

      plt.show()

      return price\_response\_model

    def generate\_price\_response(self, consumption):

      response = lambda price : (

        self.price\_response\_model(consumption)[price]

        + PRICE\_RESPONSE\_RANDOMNESS\*np.random.rand()

      )

      return response

# TODO: REMOVE

Customer()