Course: IN5460/IN9460
Artificial Intelligence for Energy Informatics

Assignment 1

Coordinate energy management for microgrid using reinforcement learning

Delivery format:
(i) Github link (preferred) or a zip file of your developed code.
(ii) A PDF report of your solution and result description.

Delivery and evaluation info:
Work in assigned groups of 3-4 students.
Students in the same group receive the same score.
Assignment 1 counts for 20% of the grade.

Deliver to:
Sabita Maharjan, `sabita@ifi.uio.no`
Shiliang Zhang[1], `shilianz@ifi.uio.no`

Delivery deadline: **17.10.2023 kl 23:59**[2]

September 2023

[1]Please contact Shiliang Zhang (shilianz@ifi.uio.no) if you have questions regarding this assignment.
[2]Please notice that this is a firm deadline. Individuals/Groups that send reports after the deadline will receive the grade "F".

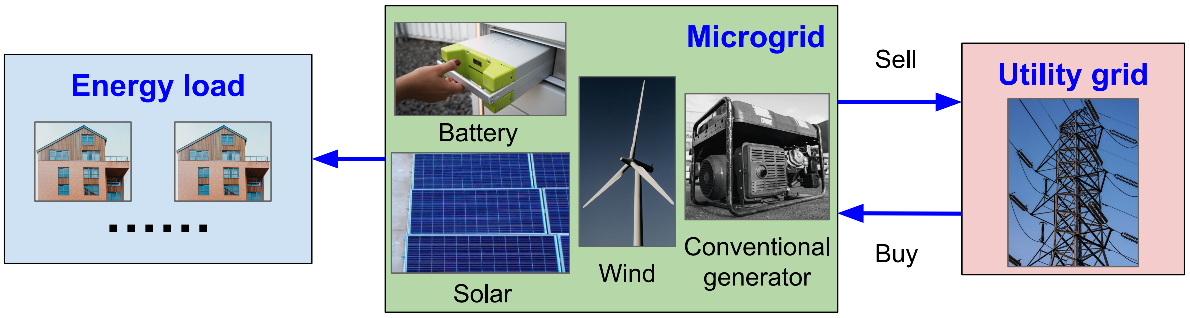

Figure 1: System architecture

1 About this assignment

Scenarios: An energy load is supplied by a microgrid as shown in figure 1. The microgrid has renewable and conventional generation and energy storage, and can buy energy from the utility grid when necessary. The profile of the energy load varies over the time.

The goal is to develop a RL mechanism that can control the demand-supply of the microgrid to reduce overall energy cost.

The following sections provide instructions and tools that might help. Note that such information is associated with the usage of Python. But you can use your preferred programming language when developing your own solution.

2 General information, instructions, and requirements

To translate the assignment into a RL task, there is a need to formulate the described scenario and interactions. We provide the following data and example code snippets in section 3.3 that might help.

2.1 Data sets

The two data sets below can be used in formulating the RL scenario:

- Solar irradiance data during 2016 from Solar Energy Local
  Download the data here.
- Wind speed data during 2016 from State Climatologist of Illinois, USA
  Download the data here.
- Price of energy purchased from the utility grid during the same period
  Download the data here
- A one-year hourly-load-profile dataset for residential energy consumption taken from

Table 1: Wind Turbine Parameters.

Parameters	Value	Parameters	Value
$v^{ci}(m/s)$-Cut in speed	3	η_t-Gearbox efficiency	0.95
$v^{co}(m/s)$-Cut off speed	11	η_g-Generator efficiency	0.95
$v^{r}(m/s)$-Rate speed	7	θ-Power coefficient	0.593
$\rho(kg/m^3)$-Air density	1.225	r^{w}_{omc}(\$/kWh)	0.085
r(m)-Blade radius	25	N_w(unit)	1

Commercial and Residential Hourly Load Profiles for all TMY3 Locations in the United States
There are lots of household energy profiles in this data silo, and you can use different numbers of household in your solution development.

With the solar irradiance and wind speed data, you can convert it to solar and wind power generation, respecitively, so as to formulate the energy dynamics from the supply side of the microgrid, as exemplified in the code snippets in section 3.3.1 and 3.3.2. When the energy generation cannot meet the demand, the microgrid needs to buy energy from the utility grid, and you can use the energy price data to calculate the cost of energy purchase. With the energy load profile dataset, you can generate the energy demand that needs to be supplied by the microgrid.

2.2 Microgrid

The microgrid system model consists of battery, wind turbine, solar PV, and gas turbine generator. The microgrid is connected with the utility grid, and provides energy supply for the energy load described in section 2.3.

At each decision-making epoch, it is possible to turn on or off the energy generation from solar PV, wind turbine, and the gas generator. For each of the three energy resource, it is possible to use their energy for three purposes: (i) support the energy load (ii) sell back to the utility grid (iii) store energy in the battery. The microgrid can purchase energy from the utility grid either for the energy load or to charge the battery in the microgrid. The microgrid battery can also support the energy load.

Note that the operation of the microgrid incurs cost, for each of the elements involved. We measure the cost rate by $\$/kWh$. Such costs are quantified in table 1,2, and 3.

We present the specifications relevant to the energy generation in the microgrid in table 1, 2, and 3. We also provide the dynamics for the microgrid shown in the code snippet in section 3.3.3 based on those microgrid specifications.

Table 2: Battery Storage Parameters.

Parameters	Value	Parameters	Value
e(kWh)-Capacity	300	b-Charging rate (kW)	2.5
SOC_{max}(%)	95	SOC_{min}(%)	5
r^b_{omc}($/kWh)	0.95	η-Efficiency	0.99

Table 3: Solar Panel and Gas Turbine Generator Parameters.

Parameters	Value	Parameters	Value
$a(m^2)$-Panel area	1400	n_g(unit)-Number of generator	1
δ-Efficiency	0.2	G_p(kW)-Generator capacity	600
r^s_{omc}($/kWh)	0.15	r^g_{omc}($/kWh)-Operating cost	0.55

2.3 The energy load

The energy load is associated with residential purpose, e.g., heating, HVAC, lights, etc., the data of which can be obtained as shown in section 2.1, which is simulated using EnergyPlus models. Note that this dataset also contains the usage of gas, and please extract from the dataset only the electricity usage.

2.4 Contents of the report to be delivered

The goal of this assignment is to use reinforcement learning to gain an optimal policy that can minimize the long-term energy cost. At a decision-making epoch t, the energy cost $C(t)$ is the sum of the cost for energy purchased $E_u(t)$ from the utility grid, plus the microgrid operational cost $O_m(t)$, and minus the sold back reward of energy to the utility grid $S_u(t)$, as shown below.

$$C(t) = E_u(t) + O_m(t) - S_u(t) \tag{1}$$

where $E_u(t) = E(t) \cdot P_g(t)$, $E(t)$ denotes the amount of energy from the utility grid, and $P_g(t)$ means the price of energy measured by $10^4\$/MWh$. The energy purchase price can be found at section 2.1. $O_m(t)$ can be calculated as below

$$O_m(t) = E_s(t) \cdot \delta^s_{omc} + E_w(t) \cdot \delta^w_{omc} + E_g(t) \cdot \delta^g_{omc} + E_b(t) \cdot \delta^b_{omc} \tag{2}$$

where δ^s_{omc}, δ^g_{omc}, δ^w_{omc}, and δ^b_{omc} are the operational cost for solar PV, conventional generator, wind turbine, and battery, respectively, as shown in table 3, 1, and 2. $E_s(t)$, $E_w(t)$, and $E_g(t)$ mean the energy generated at t by solar, wind turbine, and conventional generator, respectively. $E_b(t)$ is the amount of energy from charging or discharging the battery. $S_u(t)$ in equation 1 is calculated as

$$S_u(t) = \left(E^u_s(t) + E^u_w(t) + E^u_g(t)\right) \cdot P^u_g(t) \tag{3}$$

where $E_s^u(t)$, $E_w^u(t)$, and $E_g^u(t)$ are the energy sold back to the utility grid from solar, wind turbine, and conventional generator, respectively. $P_g^u(t)$ is the sell-back price to the utility grid, which is fixed as 0.2 $\$/KWh$.

The objective for the reinforcement learning is to learn a policy that can minimize the long-term cost of the microgrid, which is calculated by

$$\sum_{t=0}^{\infty} C(t) \tag{4}$$

The delivered report should include the following.

Question 1.1: RL solution for the microgrid with only solar power, and the microgrid supplies energy to the energy load

Please use simulation plots to show how the solar power generation changes before and after the learned RL policy. Compare the RL result with a random policy for energy management of the microgrid and provide analysis. Please use the energy load of 25 households.

Question 1.2: the same settings with Question 3.1, except that you need to use 100 household profiles

Question 1.3: the same settings with Question 3.1, except that you need to use 200-500 household profiles

For the Question 1.1, 1.2, and 1.3, please provide discussions and present insights in terms of how the decision-making changes and time complexity.

Question 2: RL solution for the microgrid with solar and wind power

Please use simulation plots to show how the solar and wind power generation shift before and after the learned RL policy. Compare the result with that where only solar power is considered and provide analysis. Please use 100 household profiles.

Question 3: RL solution for the microgrid with solar & wind power and gas turbine generator considered

Please use simulation plots to show how the solar and wind power generation changes before and after the learned RL policy in comparison with a random policy for the energy management of the microgrid. Compare the result with that where only solar power is considered in supplying the energy load and provide analysis. Please use 100 household profiles.

Question 4: the same settings with Question 3, except that we change how the energy cost is defined

We use the following equation to calculate the cost of energy purchase $E_u(t)$ in equation 1

$$E_u(t) = 0.25 \times E^2(t) \cdot P_g(t) + 0.5 \times E(t) \cdot P_g(t) \tag{5}$$

Please run the reinforcement learning and analyze how this change can affect the decision-making of the learned policy.

3 Useful libraries, and code snippets

3.1 Libraries (Python)

- Numpy: Enable array computation and diverse mathematical functions. An explanation of array can be found here.
- Pandas: Enable data loading from, e.g., CSV files; data processing like merge and reshape, etc.
- Tensorflow: Enable machine/deep learning from loading dataset, building models, to model training and logging.
- Mip: Enables the solving of mathematical programming problems.
- Scipy: Enables diverse algebraic operations and data processing.

3.2 Reinforcement learning toolkits

We provide three Python solution to RL for microgrid, which might help your solution development.

- pymgrid: an Python simulator that generate microgrid for, e.g., reinforcement learning.
- python-microgrid: an extension of pymgrid.
- easygrid: developed based on pymgrid, a simple microgrid simulation for Reinforcement Learning applications, taking into account the elements of PV, battery, local demand, and the connection with the utility grid.

3.3 Code snippets (Python)

Below are several code snippets that might be helpful for deriving the RL solution.

3.3.1 Data processing

Listing 1: Python example of importing and processing data. This code snippet using solar irradiance and wind speed and rate/price of energy consumption data in CSV format.

```
import pandas as pd
#read the solar irradiance and wind speed data from file#
#read the rate of consumption charge date from file#
file_SolarIrradiance = "SolarIrradiance.csv"
file_WindSpeed = "WindSpeed.csv"
file_rateConsumptionCharge = "rate_consumption_charge.csv"
#read the solar irradiace
data_solar = pd.read_csv(file_SolarIrradiance)
solarirradiance = np.array(data_solar.iloc[:,3])
#solar irradiance measured by MegaWatt/km^2
#read the windspeed
data_wind = pd.read_csv(file_WindSpeed)
windspeed = 3.6*np.array(data_wind.iloc[:,3])
#windspeed measured by km/h=1/3.6 m/s
#read the rate of consumption charge
data_rate_consumption_charge = pd.read_csv(file_rateConsumptionCharge)
rate_consumption_charge = np.array(data_rate_consumption_charge.iloc[:,4])
    /10
#rate of consumption charge measured by 10^4$/MegaWatt=10 $/kWh
```

3.3.2 Defining energy generation dynamics and specifications

The generation from solar PV E_s and wind turbine E_w is calculated as follows.

$$E_s = S_r \cdot a \cdot \delta / 1000 \tag{6}$$

where S_r denotes solar irradiation, a is the panel area, and δ represents efficiency of the solar PV.

The wind power generation is obtained as

$$E_w = N_w \cdot R_w \cdot (V_w - V^{ci})/(V^r - V^{ci}) \tag{7}$$

where N_w is the number of wind turbine units, R_w means rated power of the wind turbine, V_r is the rate speed, V_w is the wind speed, V^{ci} is cut-in speed. R_w can be calculated as below.

$$R_w = 0.5 * \rho \cdot \pi \cdot r^2 \cdot \overline{V_w}^3 \cdot \theta \cdot \eta_t \cdot \eta_g / (3.6 * 3.6 * 3.6) \tag{8}$$

where ρ denotes the air density, r is the wind turbine blade radius, $\overline{V_w}$ is the average wind speed, θ is the power coefficient of the wind turbine, η_t is the gearbox efficiency of the wind turbine, η_g is the generator efficiency of the wind turbine. Such calculations are reflected in the following code snippets.

Listing 2: Python example of defining system dynamics and specifications. This example showcases how to translate solar irradiation and wind speed into solar and wind power using solar PV and wind turbine.

```
cutin_windspeed=3*3.6
#the cut-in windspeed (km/h=1/3.6 m/s), v^ci#
cutoff_windspeed=11*3.6
#the cut-off windspeed (km/h=1/3.6 m/s), v^co#
rated_windspeed=7*3.6
#the rated windspeed (km/h=1/3.6 m/s), v^r#
charging_discharging_efficiency=0.95
#the charging-discharging efficiency, eta#
rate_battery_discharge=2/1000
#the rate for discharging the battery (MegaWatt), b#
unit_operational_cost_solar=0.15/10
#the unit operational and maintanance cost for generating power
#from solar PV (10^4$/MegaWattHour=10 $/kWHour), r_omc^s#
unit_operational_cost_wind=0.085/10
#the unit operational and maintanance cost for generating power
#from wind turbine (10^4$/MegaWattHour=10 $/kWHour), r_omc^w#
unit_operational_cost_generator=0.55/10
#the unit opeartional and maintanance cost for generating power
#from generator (10^4$/MegaWattHour=10 $/kWHour), r_omc^g#
unit_operational_cost_battery=0.95/10
#the unit operational and maintanance cost for battery storage system
#per unit charging/discharging cycle (10^4$/MegaWattHour=10 $/kWHour),
#r_omc^b#
capacity_battery_storage=300/1000
#the capacity of battery storage system (MegaWatt Hour=1000 kWHour), e#
SOC_max=0.95*capacity_battery_storage
#the maximum state of charge of battery system#
SOC_min=0.05*capacity_battery_storage
#the minimum state of charge of battery system#
area_solarPV=1400/(1000*1000)
#the area of the solar PV system (km^2=1000*1000 m^2), a#
efficiency_solarPV=0.2
#the efficiency of the solar PV system, delta#
density_of_air=1.225
#calculate the rated power of the wind turbine,
#density of air (10^6kg/km^3=1 kg/m^3), rho#
radius_wind_turbine_blade=25/1000
#calculate the rated power of the wind turbine,
#radius of the wind turbine blade (km=1000 m), r#
average_wind_speed=3.952*3.6
#calculate the rated power of the wind turbine,
#average wind speed (km/h=1/3.6 m/s), v_avg (from the windspeed table)#
power_coefficient=0.593
#calculate the rated power of the wind turbine, power coefficient, theta#
gearbox_transmission_efficiency=0.95
#calculate the rated power of the wind turbine,
#gearbox transmission efficiency, eta_t#
electrical_generator_efficiency=0.95
#calculate the rated power of the wind turbine,
#electrical generator efficiency, eta_g#
rated_power_wind_turbine_original
```

```
=0.5*density_of_air*np.pi*radius_wind_turbine_blade
*radius_wind_turbine_blade*average_wind_speed
*average_wind_speed*average_wind_speed*power_coefficient
*gearbox_transmission_efficiency*electrical_generator_efficiency
rated_power_wind_turbine=rated_power_wind_turbine_original/(3.6*3.6*3.6)
#the rated power of the wind turbine, RP_w (MegaWatt=10^6 W),
#with the radius_wind_turbine_blade measured in km=10^3m,
#average wind speed measured in km/hour=3.6m/s,
#RP_w will be calculated as RP_w_numerical
#then RP_w in MegaWatt=(1 kg/m^3)*(10^3 m)*(10^3 m)
#*(3.6 m/s)*(3.6 m/s)*(3.6 m/s)*RP_w_numerical
#=3.6^3*10^6 RP_w_numerical W=3.6^3 RP_w_numerical MegaWatt#
number_windturbine=1
#the number of wind turbine in the onsite generation system, N_w#
number_generators=1
#the number of generators, n_g#
rated_output_power_generator=600/1000
#the rated output power of the generator (MegaWatt=1000kW), G_p#
```

3.3.3 Defining microgrid dynamics

Listing 3: Python example of defining microgrid dynamics.

```
class Microgrid(object):
    def __init__(self,
                 workingstatus=[0,0,0],
                 #the working status of [solar PV, wind turbine, generator
    ]#
                 SOC=0,
                 #the state of charge of the battery system#
                 actions_adjustingstatus=[0,0,0],
                 #the actions of adjusting the working status (connected
    =1 or not =0 to the load) of the [solar, wind, generator]#
                 actions_solar=[0,0,0],
                 #the solar energy used for supporting [the energy load,
    charging battery, sold back]#
                 actions_wind=[0,0,0],
                 #the wind energy used for supporting [the energy load,
    charging battery, sold back]#
                 actions_generator=[0,0,0],
                 #the use of the energy generated by the generator for
    supporting [the energy load, charging battery, sold back]#
                 actions_purchased=[0,0],
                 #the use of the energy purchased from the grid for
    supporting [the energy load, charging battery]#
                 actions_discharged=0,
                 #the energy discharged by the battery for supporting the
    energy load#
                 solarirradiance=0,
                 #the environment feature: solar irradiance at current
    decision epoch#
```

```
              windspeed=0
              #the environment feature: wind speed at current decision epoch#
              ):
     self.workingstatus=workingstatus
     self.SOC=SOC
     self.actions_adjustingstatus=actions_adjustingstatus
     self.actions_solar=actions_solar
     self.actions_wind=actions_wind
     self.actions_generator=actions_generator
     self.actions_purchased=actions_purchased
     self.actions_discharged=actions_discharged
     self.solarirradiance=solarirradiance
     self.windspeed=windspeed

 def transition(self):
     workingstatus=self.workingstatus
     SOC=self.SOC
     if self.actions_adjustingstatus[1-1]==1:
         workingstatus[1-1]=1
     else:
         workingstatus[1-1]=0
     #determining the next decision epoch working status of solar PV, 1=working, 0=not working#
     if self.actions_adjustingstatus[2-1]==0 or self.windspeed>cutoff_windspeed or self.windspeed<cutin_windspeed:
         workingstatus[2-1]=0
     else:
         if self.actions_adjustingstatus[2-1]==1 and self.windspeed<=cutoff_windspeed and self.windspeed>=cutin_windspeed:
             workingstatus[2-1]=1
     #determining the next decision epoch working status of wind turbine, 1=working, 0=not working#
     if self.actions_adjustingstatus[3-1]==1:
         workingstatus[3-1]=1
     else:
         workingstatus[3-1]=0
     #determining the next decision epoch working status of generator, 1=working, 0=not working#
     SOC=self.SOC+(self.actions_solar[2-1]+self.actions_wind[2-1]+self.actions_generator[2-1]+self.actions_purchased[2-1])*charging_discharging_efficiency-self.actions_discharged/charging_discharging_efficiency
     if SOC>SOC_max:
         SOC=SOC_max
     if SOC<SOC_min:
         SOC=SOC_min
     #determining the next desicion epoch SOC, state of charge of the battery system#
     return workingstatus, SOC

```

```python
    def EnergyConsumption(self):
        #returns the energy consumption from the grid#
        return -(self.actions_solar[1-1]+self.actions_wind[1-1]+self.actions_generator[1-1]+self.actions_discharged)

    def energy_generated_solar(self):
        #calculate the energy generated by the solar PV, e_t^s#
        if self.workingstatus[1-1]==1:
            energy_generated_solar=self.solarirradiance*area_solarPV*efficiency_solarPV/1000
        else:
            energy_generated_solar=0
        return energy_generated_solar

    def energy_generated_wind(self):
        #calculate the energy generated by the wind turbine, e_t^w#
        if self.workingstatus[2-1]==1 and self.windspeed<rated_windspeed and self.windspeed>=cutin_windspeed:
            energy_generated_wind=number_windturbine*rated_power_wind_turbine*(self.windspeed-cutin_windspeed)/(rated_windspeed-cutin_windspeed)
        else:
            if self.workingstatus[2-1]==1 and self.windspeed<cutoff_windspeed and self.windspeed>=rated_windspeed:
                energy_generated_wind=number_windturbine*rated_power_wind_turbine*Delta_t
            else:
                energy_generated_wind=0
        return energy_generated_wind

    def energy_generated_generator(self):
        #calculate the energy generated bv the generator, e_t^g#
        if self.workingstatus[3-1]==1:
            energy_generated_generator=number_generators*rated_output_power_generator*Delta_t
        else:
            energy_generated_generator=0
        return energy_generated_generator

    def OperationalCost(self):
        #returns the operational cost for the onsite generation system#
        if self.workingstatus[1-1]==1:
            energy_generated_solar=self.solarirradiance*area_solarPV*efficiency_solarPV/1000
        else:
            energy_generated_solar=0
        #calculate the energy generated by the solar PV, e_t^s#
        if self.workingstatus[2-1]==1 and self.windspeed<rated_windspeed and self.windspeed>=cutin_windspeed:
            energy_generated_wind=number_windturbine*rated_power_wind_turbine*(self.windspeed-cutin_windspeed)/(
rated_windspeed-cutin_windspeed)
        else:
            if self.workingstatus[2-1]==1 and self.windspeed<cutoff_windspeed and self.windspeed>=rated_windspeed:
                energy_generated_wind=number_windturbine*rated_power_wind_turbine*Delta_t
            else:
                energy_generated_wind=0
        #calculate the energy generated by the wind turbine, e_t^w#
        if self.workingstatus[3-1]==1:
            energy_generated_generator=number_generators*rated_output_power_generator*Delta_t
        else:
            energy_generated_generator=0
        #calculate the energy generated bv the generator, e_t^g#
        operational_cost=energy_generated_solar*unit_operational_cost_solar+energy_generated_wind*unit_operational_cost_wind+energy_generated_generator*unit_operational_cost_generator
        operational_cost+=(self.actions_discharged+self.actions_solar[2-1]+self.actions_wind[2-1]+self.actions_generator[2-1])*Delta_t*unit_operational_cost_battery/(2*capacity_battery_storage*(SOC_max-SOC_min))
        #calculate the operational cost for the onsite generation system#
        return operational_cost

    def SoldBackReward(self):
        #calculate the sold back reward (benefit)#
        return (self.actions_solar[3-1]+self.actions_wind[3-1]+self.actions_generator[3-1])*unit_reward_soldbackenergy

    def PrintMicrogrid(self, file):
        #print the current and the next states of the microgrid#
        print("Microgrid working status [solar PV, wind turbine, generator]=", self.workingstatus, ", SOC=", self.SOC, file=file)
        print(" microgrid actions [solar PV, wind turbine, generator]=", self.actions_adjustingstatus, file=file)
        print(" solar energy supporting [the energy load, charging battery, sold back]=", self.actions_solar, file=file)
        print(" wind energy supporting [the energy load, charging battery, sold back]=", self.actions_wind, file=file)
        print(" generator energy supporting [the energy load, charging battery, sold back]=", self.actions_generator, file=file)
        print(" energy purchased from grid supporting [the energy load, charging battery]=", self.actions_purchased, file=file)
        print(" energy discharged by the battery supporting the energy load=", self.actions_discharged, file=file)
        print(" solar irradiance=", self.solarirradiance, file=file)
        print(" wind speed=", self.windspeed, file=file)
        print(" Microgrid Energy Consumption=", self.EnergyConsumption(), file=file)
        print(" Microgrid Operational Cost=", self.OperationalCost(), file=file)
        print(" Microgrid SoldBackReward=", self.SoldBackReward(), file=file)
        print("\n", file=file)
        return None
```