12-770: Assignment #2: More Building Physics

Mario Bergés

2026-02-09

We will follow a very similar procedure as last time for completing this assignment. Use Jupyter Notebooks to edit the file.

Thermal Comfort (25%)

Let's get some thermal comfort topics out of the way before going into more interesting territories.

Problem #1 (15%):

Consider a room with the following conditions:

- Dimensions are 2.5m, 4m and 5m (height, width and length).
- One side has dimensions 2.5 x 4m and is entirely glazed with interior surface temperature of 10°C.
- The other surfaces are at 20°C.
- The air is at 22°C dry-bulb temperature and 30% relative humidity.

Given those conditions, answer the following questions:

What is the MRT? (5%)

Generate the answer in the Python cell below:

```
# Your answer goes here
```

What is the operative temperature? (5%)

Generate the answer in the Python cell below:

```
# Your answer goes here
```

Are these conditions within the comfort limits of ASHRAE? (5%)

Your answer goes here.

Problem #2 (10%):

A person feels very comfortable in their house when wearing light clothing if the thermostat is set at 22°C and the MRT = 22°C. During a cold day, the MRT drops to 18°C.

To what value must the indoor dry-bulb air temperature be raised to maintain the same level of comfort? (5%)

Generate the answer in the Python cell below:

```
# Your answer goes here
```

If the person sits near a window and receives solar radiation, the thermostat can be lowered. What types of factors will influence the amount by which the thermostat can be lowered? (5%):

Your answer goes here.

Thermal Network Models (10%)

We continue with more problems that are related to concepts covered during the first few weeks of the class. For these, you will need to read Chapter 8 of Reddy and any other references related to thermal network models.

Time constants (5%)

What is the time constant of a building with total heat loss coefficient $K_{tot} = 30kW/K$ and effective heat capacity $C_{eff} = 2.4GJ/K$? Estimate how long it takes the indoor temperature to drop from 20°C to 15°C after the heating system is shut off when $T_0 = 0$°C.

```
# Your answer goes here
```

Duration (5%)

The air conditoiner of a building is to be switched off during a hot day so as to reduce electricity demand during the on-peak period. The building has a time constant of 15h, and the sum of internal loads (i.e., equipment, lights and people) and the solar loads divided by UA is 4°C. Assume $T_{i,b}$ (the internal air temperature at the beginning of the float-up period) to be 18°C. Calculate the number of hours it takes for the building to heat up by 4°C when:

- (a) The temperature difference between outdoor air and indoor at the start of the ramp-up is 10°C
- (b) The temperature difference between outdoor air and indoor at the start of the ramp-up is 20°C

Comment on your findings.

```
# Your code goes here
```

Calibration of RC Models (20%)

Now we will slowly work our way into the building controls (which we have not yet covered in much detail during class) by expanding on what the thermal network models allows us to do. We will begin with the calibration of a simple 1R1C thermal network using actual data from an experiment, and then move on to the simulation of a slightly more complex thermal network that we will attempt to control using simple controllers.

We will assume that we have a very simple insulated box in which an incandescent light bulb has been placed along with a thermometer.[1] The bulb supplies heat to the box, which is then dissipated through the (not perfectly) insulated walls. Data was collected from this experiment:

[1]The data for this experiment, as well as the examples in this assignment all came from an online book by a French researcher. I would like to attribute this correctly to them, but unfortunately I have lost track of the website and cannot find it anymore. If you happen to know the source, please let me know!

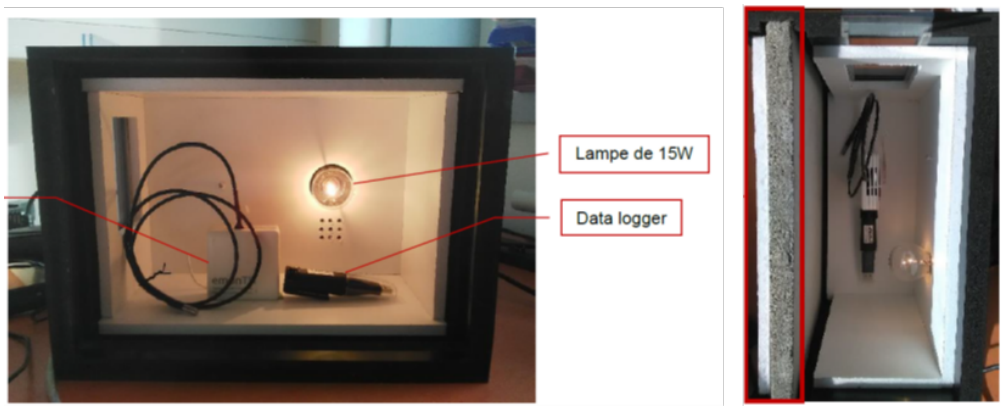


Figure 1: Test setup, from the original publication accompanying the dataset we're using

All the data is contained in the file named "test_box.xlsx". Take a second to open the file with your preferred spreadsheet parser to familiarize yourself with the data. This dataset contains three experiments with different wall compositions **(concrete, wood, insulated_concrete)**. For each test the lamp inside the box is turned on and off a few times, and kept on overnight.

Variables:

- **time**: timestamp for the sample
- **Ti**: internal temperature (C)
- **Ta**: ambient temperature (C)
- **P**: heating power (W)
- **dt**: time interval between samples (s)

Problem 1 (5%)

Finish the code cell below to be able to load the XLSX file called *test_box.xlsx* into memory with the option to select different sheets by changing a function parameter.

Hint: Use the `pandas.read_excel` and make sure you understand the `sheet_name` parameter to select the different experimental runs.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Insert your answer here


# There are some missing values. Apply the function below to your loaded dataframe.
# df.interpolate(inplace=True)
```

Problem 2 (10%)

Your goal is to estimate the values of R and C that "best" fit the data, where by "best" we mean the values of R and C that minimize an empirical loss function over the data. For simplicity, you may assume that the loss function ℓ is the squared error loss (i.e. $\ell = \sum_{j=1}^{N} \left(T_{i,j} - \hat{T}_{i,j}(R,C)\right)^2$, where $T_{i,j}$ is the indoor temperature at the j-th sample, and $\hat{T}_{i,j}$ is the estimated function for indoor temperature using the 1R1C network model).

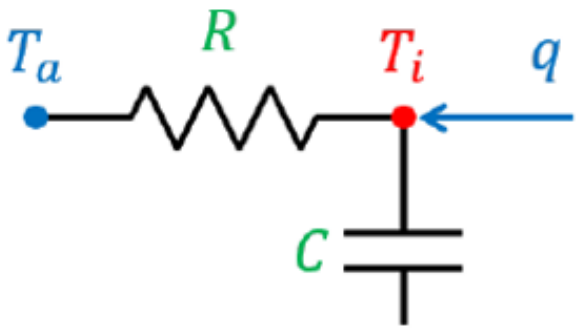


Figure 2: The 1R1C network

The equation of this model is:

$$C\frac{\mathrm{d}T_i}{\mathrm{d}t} = \frac{1}{R}(T_a - T_i) + P$$

For our application want to use a discrete model that calculates the value of T_i as a function of the inputs (T_a, P) and the parameters (R, C):

$$\left(\frac{C}{\mathrm{d}t} + \frac{1}{R}\right)T_i^{t+1} = \frac{C}{\mathrm{d}t}T_i^t + \frac{1}{R}T_a^{t+1} + P^{t+1}$$

```
# We can implement this function as Python code to calculate the next step Ti
def ti(inputs, R, C):

    ta = inputs[:,0]   # ambient temp
    p  = inputs[:,1]   # power input
    dt = inputs[:,2]   # timestep
    y = np.zeros(len(inputs))
    y[0] = inputs[3,0] # use first value from data

    for t in range(1,len(y)):
        y[t] = 1/(C/dt[t]+1/R) * (C/dt[t]*y[t-1] + 1/R*ta[t] + p[t])
    return y
```

Using the data we just loaded into memory, the function we implemented above, and a curve fitting toolbox such as `scipy.optimize.curve_fit` to estimate the parameters for **each** of the test runs. Report R and C for each of the three runs.

Hint 1: write the equation above using pen and paper to understand the difference between the inputs and the parameters. Remember, we are only trying to find R and C that best fit our data.

Hint 2: you only have to implement the code once and change the `sheet_name` to get the parameters for different runs.

```
# Insert your code here
```

Comparing to Reality (5%)

What is the comparison of the R and C values you found by answering Problem 2? Do they make sense to you regarding the insulation materials (i.e., concrete, wood, insulated_concrete)?

> Your answer goes here.

Building controls (15%)

You are now given:

- a simple **2R2C** thermal model of a building (implemented for you, the same one we saw in class),
- a disturbance dataset: outdoor temperature `To(t)` and solar irradiance `I_sol(t)`,
- a baseline schedule signal `hea_on(t)` (again, the same one we saw in class)

Your task is to design and evaluate a **Proportional (P) heating controller** that tracks a temperature setpoint schedule while respecting actuator limits. We will begin just as we did in the example we presented in class. We will use a 15-minute simulation timestep (`dt = 900 s`).

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Plot formatting
plt.rcParams["figure.figsize"] = (10, 4)
df = pd.read_csv("input_data.csv", parse_dates=["Time"])
df = df.sort_values("Time").reset_index(drop=True)
```

```
time = df["Time"].values
To = df["To"].values          # Outdoor temperature [°C]
Isol = df["I_sol"].values     # Solar irradiance [W/m^2]
hea_on = df["hea_on"].values  # Baseline schedule (0/1), optional

dt = 900  # seconds (15 minutes)
dt_hours = dt / 3600

df.head()
```

	Time	To	I_sol	hea_on
0	2019-06-01 00:00:00	5.00	0.0	0
1	2019-06-01 00:15:00	4.85	0.0	0
2	2019-06-01 00:30:00	4.70	0.0	0
3	2019-06-01 00:45:00	4.55	0.0	0
4	2019-06-01 01:00:00	4.40	0.0	0

Provided plant model: 2R2C building thermal network

State variables:

- T_i: indoor air temperature [°C]
- T_e: envelope temperature [°C]

Inputs/disturbances:

- P_{hea}: heater power [W] (control input)
- T_o: outdoor temperature [°C]
- I_{sol}: solar irradiance [W/m²]

Discrete-time update (forward Euler, fixed $dt = 900s$):

$$T_i^{k+1} = T_i^k + \frac{dt}{C_i}\left(\frac{T_e^k - T_i^k}{R_i} + P_{hea}^k + A_i I_{sol}^k\right)$$

$$T_e^{k+1} = T_e^k + \frac{dt}{C_e}\left(\frac{T_i^k - T_e^k}{R_i} + \frac{T_o^k - T_e^k}{R_o} + A_e I_{sol}^k\right)$$

You will use this model *as-is* for simulation and control design.

```python
class Building:
    def __init__(self, Ti, Te, Ri, Ro, Ci, Ce, Ai, Ae, dt=900):
        self.Ri = Ri  # [K/W]
        self.Ro = Ro  # [K/W]
        self.Ci = Ci  # [J/K]
        self.Ce = Ce  # [J/K]
        self.Ai = Ai  # [m^2]
        self.Ae = Ae  # [m^2]
        self.Ti = Ti  # [°C]
        self.Te = Te  # [°C]
        self.dt = dt  # [s]

    def next_step(self, P_hea, To, I_sol):
        dt = self.dt

        self.Ti = self.Ti + (dt / self.Ci) * (
            (self.Te - self.Ti) / self.Ri
            + P_hea
            + self.Ai * I_sol
        )
        self.Te = self.Te + (dt / self.Ce) * (
            (self.Ti - self.Te) / self.Ri
            + (To - self.Te) / self.Ro
            + self.Ae * I_sol
        )

    def get_state(self):
        return float(self.Ti), float(self.Te)
```

Model parameters and initial conditions

These parameters are assumed to be *already calibrated* (you do not need to estimate them).

You may change only:

- initial conditions `Ti0`, `Te0` (if you want),
- controller parameters (gain, saturation, setpoints).

Do **not** change `Ri, Ro, Ci, Ce, Ai, Ae`.

```
# --- Given model parameters (do not change) ---
Ri = 4.50e-03  # [K/W] indoor   envelope thermal resistance
Ro = 5.80e-02  # [K/W] envelope   outdoor thermal resistance
Ci = 5.80e+06  # [J/K] indoor thermal capacitance
Ce = 2.05e+07  # [J/K] envelope thermal capacitance
Ai = 3.20e-01  # [m^2] indoor solar gain coefficient
Ae = 2.20e-01  # [m^2] envelope solar gain coefficient

# Initial conditions
Ti0 = 20.0  # [°C]
Te0 = 20.0  # [°C]
```

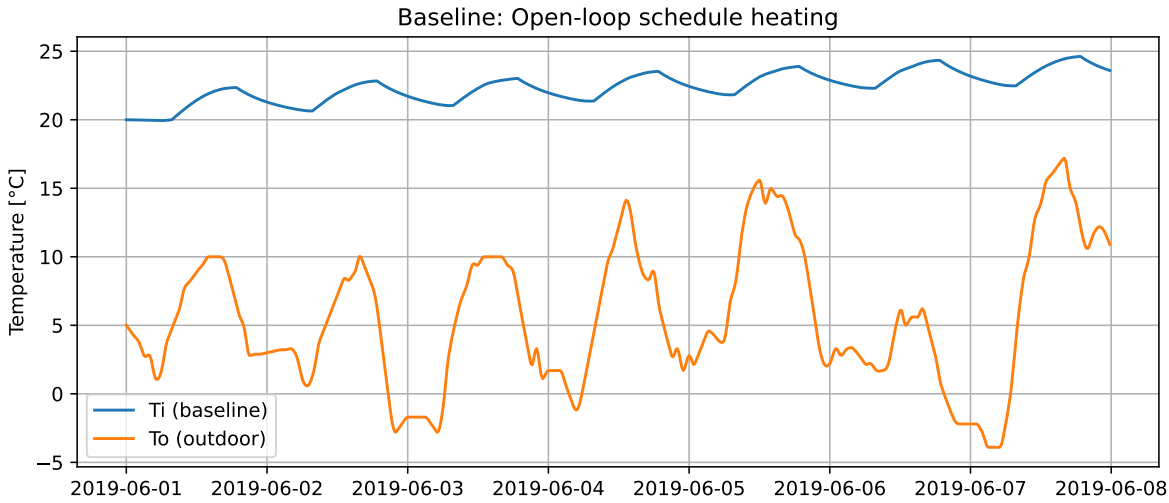
Baseline (optional): open-loop schedule

Before you design your controller, simulate a baseline case where the heater is either OFF/ON according to `hea_on(t)`.

This is not the main deliverable; it's here to give you a reference trajectory.

```
def simulate_open_loop(Phea_series_watts, Ti0=Ti0, Te0=Te0):
    bldg = Building(Ti=Ti0, Te=Te0, Ri=Ri, Ro=Ro, Ci=Ci, Ce=Ce, Ai=Ai, Ae=Ae, dt=dt)

    N = len(time)
    Ti_hist = np.zeros(N)
    Te_hist = np.zeros(N)

    for k in range(N):
        bldg.next_step(Phea_series_watts[k], To[k], Isol[k])
        Ti_hist[k], Te_hist[k] = bldg.get_state()

    return Ti_hist, Te_hist

P_nom = 500.0  # [W] nominal heating power for the baseline schedule
Phea_baseline = hea_on * P_nom

Ti_base, Te_base = simulate_open_loop(Phea_baseline)

plt.plot(time, Ti_base, label="Ti (baseline)")
plt.plot(time, To, label="To (outdoor)")
plt.title("Baseline: Open-loop schedule heating")
plt.ylabel("Temperature [°C]")
plt.legend()
```

```
plt.grid(True)
plt.show()
```



Setpoint schedule

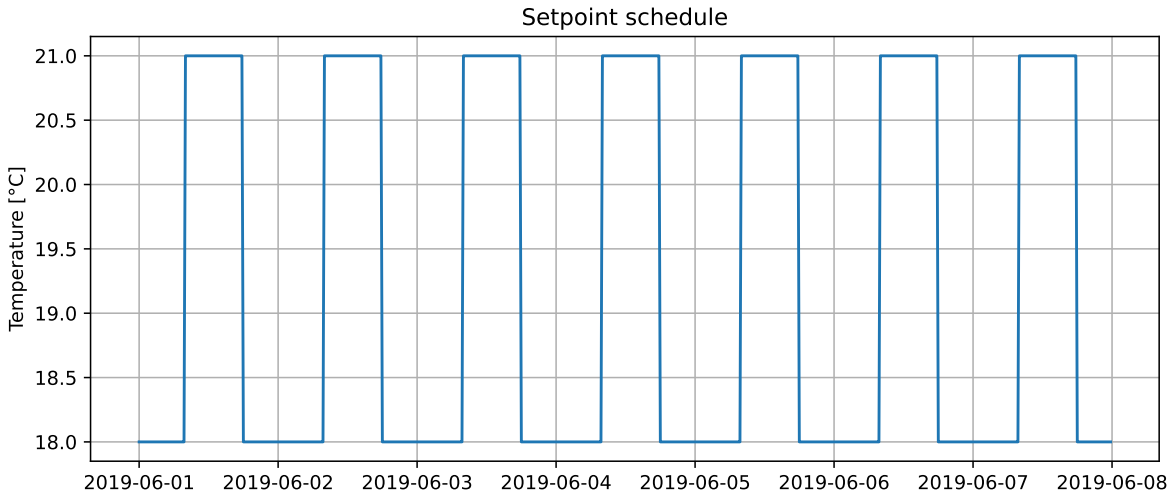
We will use a simple day/night schedule:

- **Occupied hours:** 08:00–18:00 → setpoint = 21°C
- **Unoccupied hours:** otherwise → setpoint = 18°C

You may adjust the setpoints (e.g., 20/17), but keep a clear occupied/unoccupied schedule for fair comparisons.

```
def is_occupied(ts: np.datetime64) -> bool:
    # Convert to pandas Timestamp for convenient hour extraction
    t = pd.Timestamp(ts)
    return (t.hour >= 8) and (t.hour < 18)

def get_setpoint(ts: np.datetime64, sp_occ=21.0, sp_unocc=18.0) -> float:
    return sp_occ if is_occupied(ts) else sp_unocc

# Visualize the setpoint profile
sp_profile = np.array([get_setpoint(t) for t in time])
plt.plot(time, sp_profile, label="Setpoint")
plt.title("Setpoint schedule")
```

```
plt.ylabel("Temperature [°C]")
plt.grid(True)
plt.show()
```



Task: Design and implement a P controller (5%)

Implement a heating controller that computes heater power (P_{hea}) [W] from the current indoor temperature (T_i) and the current setpoint (T_{sp}).

Requirements:

1. **Proportional control law**: $u = K_p(T_{sp} - T_i)$
2. **Saturation**: $0 \leq u \leq P_{max}$
3. **No cooling** (heater cannot be negative)
4. Implement as a Python function `p_controller(...)` below.

Hint: A practical implementation often includes a small deadband to reduce chattering, but this is optional.

```
# TODO: Implement the P controller.

def p_controller(Ti: float, Tsp: float, Kp: float, Pmax: float, deadband: float = 0.0) -> fl
    """Proportional heater controller.

    Requirements:
    - u = Kp*(Tsp - Ti)
    - 0 <= u <= Pmax
```

```
    - No cooling (u cannot be negative)
    - Optional deadband: if (Tsp - Ti) <= deadband then u=0

    Return:
    - Heater power command u [W]
    """
    # YOUR CODE HERE
```

Closed-loop simulation

Do not change the simulation loop below. Your controller will be called once per timestep.

```
def simulate_closed_loop(Kp: float, Pmax: float, deadband: float = 0.0, Ti0=Ti0, Te0=Te0,
                         sp_occ: float = 21.0, sp_unocc: float = 18.0):
    bldg = Building(Ti=Ti0, Te=Te0, Ri=Ri, Ro=Ro, Ci=Ci, Ce=Ce, Ai=Ai, Ae=Ae, dt=dt)

    N = len(time)
    Ti_hist = np.zeros(N)
    Te_hist = np.zeros(N)
    Phea_hist = np.zeros(N)
    Tsp_hist = np.zeros(N)

    for k in range(N):
        Ti_k, Te_k = bldg.get_state()
        Tsp_k = get_setpoint(time[k], sp_occ=sp_occ, sp_unocc=sp_unocc)

        P_k = p_controller(Ti=Ti_k, Tsp=Tsp_k, Kp=Kp, Pmax=Pmax, deadband=deadband)

        bldg.next_step(P_k, To[k], Isol[k])

        Ti_hist[k], Te_hist[k] = bldg.get_state()
        Phea_hist[k] = P_k
        Tsp_hist[k] = Tsp_k

    return Ti_hist, Te_hist, Phea_hist, Tsp_hist

# Example run (this will fail until you implement p_controller in the cell above)
Kp_example = 1500.0
Pmax_example = 5000.0
deadband_example = 0.1
```

```
Ti_cl, Te_cl, Phea_cl, Tsp_cl = simulate_closed_loop(Kp=Kp_example, Pmax=Pmax_example, deadba

plt.plot(time, Ti_cl, label="Ti (closed-loop)")
plt.plot(time, Tsp_cl, "--", label="Setpoint")
plt.plot(time, To, label="To (outdoor)")
plt.title("Closed-loop with P controller")
plt.ylabel("Temperature [°C]")
plt.legend()
plt.grid(True)
plt.show()

plt.plot(time, Phea_cl, label="Phea [W]")
plt.title("Heater power command")
plt.ylabel("Power [W]")
plt.grid(True)
plt.show()
```

Quantitative evaluation

Compute and report:

1. **Energy consumption** in kWh
2. **Comfort violation** in degree-hours during occupied periods:

$$V = \sum_{k \in \text{occupied}} \max(0, T_{sp}^k - T_i^k) \Delta t$$

Lower violation is better; lower energy is better.

You will tune K_p and optionally a deadband to explore the trade-off.

```
def energy_kwh(Phea_watts: np.ndarray) -> float:
    return float(np.sum(Phea_watts) * dt_hours / 1000.0)

def comfort_violation_degree_hours(Ti: np.ndarray, Tsp: np.ndarray) -> float:
    occ_mask = np.array([is_occupied(t) for t in time], dtype=bool)
    diff = np.maximum(0.0, Tsp - Ti)
    return float(np.sum(diff[occ_mask]) * dt_hours)

def summarize_run(name: str, Ti: np.ndarray, Phea: np.ndarray, Tsp: np.ndarray):
    return {
        "case": name,
```

```
        "energy_kWh": energy_kwh(Phea),
        "comfort_violation_degHr": comfort_violation_degree_hours(Ti, Tsp),
        "Ti_min": float(np.min(Ti)),
        "Ti_max": float(np.max(Ti)),
    }
```

Task: Tune the controller (10%)

Run at least **three** different K_p values and compare results.

Suggested starting points: - 500 W/°C - 1500 W/°C - 3000 W/°C

What to submit - A table comparing energy and comfort violation for your tested gains - A short paragraph explaining the trade-off and your final chosen gain

```
# TODO: Choose at least 3 values of Kp and run the simulations.
Kp_list = [500.0, 1500.0, 3000.0]  # you may modify
Pmax = 5000.0
deadband = 0.1

rows = []

for Kp in Kp_list:
    Ti_cl, Te_cl, Phea_cl, Tsp_cl = simulate_closed_loop(Kp=Kp, Pmax=Pmax, deadband=deadband)
    rows.append(summarize_run(name=f"Kp={Kp:.0f}", Ti=Ti_cl, Phea=Phea_cl, Tsp=Tsp_cl))

results = pd.DataFrame(rows)
results
```

Choose the best gain (based on your preferred trade-off) and compare:

- Indoor temperature trajectories (baseline vs closed-loop)
- Heater power (baseline vs closed-loop)

Include setpoint on the temperature plot.

```
# TODO: Choose your best Kp and generate the comparison plots.
Kp_best = 1500.0  # replace with your choice

Ti_best, Te_best, Phea_best, Tsp_best = simulate_closed_loop(Kp=Kp_best, Pmax=Pmax, deadband=

# YOUR PLOTS HERE
```

1. What gain did you choose, and why?
2. Describe the **energy vs comfort** trade-off you observed.
3. What are two limitations of a pure P controller in this application?

Insert your answers in a new Markdown cell, below:

AC Power (25%)

Using the values of current and voltage contained in the file called *current_voltage.csv*, answer the following questions:

- What is the frequency of the signals? (5%)

```
# Enter your code here
```

- What is the sampling frequency? (5%)

```
# Enter your code here
```

- For one period (cycle), what is the apparent power? (5%)

```
# Enter your code here
```

- For one period (cycle), what is the real (or active) power? (5%)

```
# Enter your code here
```

- For one period (cycle), what is the reactive power? (5%)

```
# Enter your code here
```

Fingerprinting Appliances (15%)

We're not going to do as much hand-holding on this one. You have basically two tasks, both applied to a large dataset of electrical power measurements for home appliances called PLAID, which is described in this paper. The two tasks are:

1) **Task:** Calculate the steady-state active (P) and reactive (Q) power for all appliances in the 2017 dataset and plot them in a single plot of Q vs. P. (10%)
2) **Task:** Propose a feature that can be extracted from the raw voltage and current signals of each appliance and can better discriminate between different types of light bulbs and produce a 1-dimensional plot of all light bulb types along this feature to visually demonstrate the separability.