Delrapport 2

Modellering i UUBEM

Författare: Filip Senneby, Ellen Nymo

1. Bakgrund

Uppsala Urban Building Energy Model (UUBEM) är en simulationsmodel som har tagits fram av lärare och professorer på kursen Urbana och decentraliserade energisystem på Uppsala universitet. Modellen inkluderar modulerna: rörelse och aktivitet, byggnader och värmesystem, PV system samt eldistributionssystem. Statistiska samt fysikaliska modeller används för att simulera data på timbasis.

2. Teori och användning

2.1 Simuleringsverktyget UUBEM och excel

Modellens data läses in från ett excelblad där bland annat platsspecifik information om klimatet, olika solcellsmoduler och dess egenskaper, byggnadstyper och rörelsemönster definieras. Nedan visas några kategorier som behandlas i dokumentet.

Figur 1. Exempel på kategorier som används i modellen.

I projektarbetet *Effektpositiv spårvagnsdepå* används UUBEM för att simulera hur solcellsproduktion kan kombineras med energilagring. Modellen används då den kan undersöka hur solcellsproduktionen ser ut för en specifik plats. Klimatdata över långa perioder på exempelvis solinstrålning och temperaturer har en betydande roll i hur mycket effekt en solpanel kan generera. Genom användning av denna platsspecifika data vid simulering kan resultatet ses som mer validerat.

Figur 2 nedan visar ett urklipp på några timmars klimat i Uppsala där områdets koordinater bestäms. Figur 3 illustrerar hur solcellsmodulen definieras i excelfilen.

Hour of year	T	GHI	DHI	BHI	Latitude	Longitude	Time zone	Albedo
1	3,5	0	0	0	59,8586	17,6389	-1	0,2
2	3,2	0	0	0				
3	3	0	0	0				
4	2,7	0	0	0				
5	2,5	0	0	0				
6	2,3	0	0	0				
7	2	0	0	0				
8	1,8	0	0	0				
9	1,5	0	0	0				
10	1,9	4	4	0				
11	1,9	6	6	0				
12	1,8	2	2	0				
13	2,2	25	25	0				
14	2,3	13	13	0				
15	2,1	3	3	0				

Figur 2. Exempel på klimatdata för några timmar samt koordinater på platsen som undersöks.

PV module	Module area	Max power	NOCT	Temperature coefficient
A	1,62688	315	44	-0,0038

Figur 3. Exempel på solcellsmodul A som används i projektet.

2.2 PV-simulering och soldata

Solinstrålningsdatan kommer från [---] och är ett medelvärde som tagits fram över en 10 årsperiod. Datan ligger på timbasis över hela året och ger exakta värden för den tilltänkta placeringen av spårvagnsdepån med

2.3 Effektförbrukning på depån

Datan för effektförbrukningen på depån har beräknats enligt delrapport 1.
Dessa värden har extraherats och laddats in som en lista i UUBEM. Eftersom konsumtionen endast är beräknad för ett generellt dygn i januari så har samma dygnsdata använts över hela månaden.

2.4 Batterialgoritmens funktion

Batterialgoritmen fungerar på så sätt att vid varje tillfälle som det finns en nettoproduktion av el på depån så hanteras överskottet med avseende på vilken belastning som råder i regionens elnät. Belastningsgraden definieras i “Delrapport: Konsumtion” Figur 1, genom att avläsa hur effektförbrukningen ser ut över dygnet. Intervallen där effektförbrukningen är hög kommer innebära en direkt kapacitetsbrist i hela regionen och definieras som “högbelastade”, medan övriga timmar är “lågbelastade”. Om regionsnätet är högt belastat kommer överproduktionen att gå ut på nätet för att kunna stödja det och om det är låg belastning så lagras överskottsenergin i batteriet. I de fall där regionens nät är högbelastat och det finns ett underskott av solelproduktion på depån så kommer alltså ytterligare el att behövas. Datan för hur stort effektunderskottet är över dygnets alla timmar och hela Januari sparas. Utifrån detta kan sedan storleken på en generator dimensioneras.

Om regionens nät är lågt belastat så kommer spårvagnsdepån att nyttja nätet för sin egen elförsörjning och även ladda upp alla batterier så att de är fulla. Om överskottet från solelen är stort även efter att alla batterier har laddats upp, så kommer överskottet att skickas ut på nätet.

3. Bilagor

Kod för modellering/simulering UUBEM

-

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 25 14:07:41 2024

@author: ellen
"""

#%% Import modules

import uubem.readinput as r
import uubem.simulator as s
import uubem.analysis as a
import uubem.kmlexport as kml
import numpy as np
import pickle
import matplotlib.pyplot as plt
import pandas as pd

#%%
data, system, types = r.read_input('uubem_input_data_kandidatarbete.xlsx')


# Import pre-simulated occupanc
with open('occupancy.pickle', 'rb') as f:
      occupancy = pickle.load(f)

heat_balance, heat_demand, energy_use \
   = s.simulate_buildings(data, system, types, occupancy)

# Simulate all PV systems
irradiance, power_generation \
      = s.simulate_pv_systems(data, system, types)

#results = s.simulate(data,system,types)
results = s.simulate(data, system, types, ['2023-01-01', '2023-01-31'])


# PV power generation
pv_per_building \
      = a.result_per_component(power_generation,
                               'PV system output', operation='sum')

#%%
#Adding dataframes

power_generation = results['PV systems']['Power generation']
power_generation_dataframe = a.add_dataframes(power_generation)
```

```
#Energy use & production
P = power_generation_dataframe['PV system output']

#%%
#Adding time steps to series
index = pd.date_range(start='2023-01-01 00:30:00.000', end='2023-01-31
23:30:00.000', freq='H')

effekt_justerad = [197000, 174000, 161000, 187000, 183000, 183000, 157000,
115000, 150000, 147000, 174000, 167000, 150000, 156000, 153000, 161000,
147000, 136000, 104000, 112000, 159000, 226000, 227000, 220000]
effekt_ickejusterad = [134000, 111000, 98000, 125000, 121000, 121000, 94000,
114000, 212000, 209000, 236000, 229000, 212000, 218000, 216000, 223000,
210000, 199000, 104000, 112000, 159000, 164000, 164000, 158000]

effekt_data = effekt_justerad

effekt_m = effekt_data * 31
#%%

L_vec_month = pd.Series(effekt_m, index=index, dtype = float)
P_month = pd.Series(P, index=index, dtype = float)

#%%
#Defining grid capacity/load with specific hours
index_hour = pd.date_range(start='2023-01-01 00:30:00.000', end='2023-01-31
23:30:00.000', freq='H')

h = 24
Grid_state = []

for i in range(len(P)):
      if i % h >= 9 and i % h < 12:
      Grid_state.append('high')
      elif i % h > 16 and i % h <= 20:
      Grid_state.append('high')

      else:
      Grid_state.append('low')

#%%
def battery(PPV, PH, Grid_state, BP, dT):
      """
      BATTERY Simple battery model

      Outputs:
      -------
      BR: Battery results (dictionary)

      Inputs:
      ------
      PPV: Power production [W]
```

```
PH:  Power demand [W]
Grid_state: State of the grid (e.g., "low", "normal", "high")
BP:  Battery parameters (dictionary)
dT:  Time step [hours]
"""

# Length of time series
N = len(PPV)

# Initialize result arrays
E = [0] * N
PBat = [0] * N
PGrid = [0] * N
C = [0] * (N + 1)

# Initial state of charge (at minimum SOC)
C[0] = BP['Cmin']

# Loop through all time steps
for i in range(N):

E[i] = (PPV[i] - PH[i]) * dT  # Net energy to battery/grid

if PPV[i] > PH[i]:

      if BP['effIn'] * E[i] <= BP['Cmax'] - C[i]:

      C[i + 1] = C[i] + BP['effIn'] * E[i]
      PBat[i] = PPV[i] - PH[i]

      else:

      C[i + 1] = BP['Cmax']
      PBat[i] = (BP['Cmax'] - C[i]) / (BP['effIn'] * dT)

else:
      if Grid_state[i] == 'high':

      if abs(E[i] / BP['effOut']) <= C[i] - BP['Cmin']:

            C[i + 1] = C[i] + E[i] / BP['effOut']
            PBat[i] = PPV[i] - PH[i]

      else:

            C[i + 1] = BP['Cmin']
            PBat[i] = -BP['effOut'] * (C[i] - BP['Cmin']) / dT

      elif Grid_state[i] == 'low':

      gridChargePower = 1000000 #1MW kan dras från nätet

      if BP['effIn'] * gridChargePower <= BP['Cmax'] - C[i]:
```

```
                C[i + 1] = C[i] + BP['effIn'] * gridChargePower
                PBat[i] = gridChargePower

            else:
                C[i + 1] = BP['Cmax']
                PBat[i] = (BP['Cmax'] - C[i]) / (BP['effIn'] * dT)

            else:

            C[i + 1] = C[i]

      PGrid[i] = PPV[i] - PH[i] - PBat[i]

      # Save results
      BR = {'C': C, 'PBat': PBat, 'PGrid': PGrid}

      Generator_need = np.zeros_like(P)
      for i in range(len(BR['PGrid'])):
      if BR['PGrid'][i] < 0 and Grid_state[i] == 'high':
            Generator_need[i] = BR['PGrid'][i]

      return BR, -Generator_need
#%%

#%%
P_vec = P.tolist()
L_vec = L_vec_month.tolist()

#Här bestäms batteriets storlek
#batterysize = [125000, 125000*2, 125000*3, 125000*4]
batterysize = [312500, 312500*2, 312500*3]
Generator_needed = []
Battery_dependency = []
Max_per_day = []
C_bat = []
P_grid_state = []
for i in range(len(batterysize)):
      #save BP differently depending on battery size
      BP = {
      'Cmin': 100,
      'Cmax': batterysize[i],
      'effIn': 0.9,
      'effOut': 0.9
      }

      BR, Generator_need = battery(PPV=P_vec, PH=L_vec,
Grid_state=Grid_state, BP=BP, dT=1)
      Battery_dependency.append(BR)

      Generator_needed.append(Generator_need)
      Max_per_day.append(max(Generator_needed[i]))
      C_bat.append(BR['C'])
```

```
      P_grid_state.append(BR['PGrid'])

avg_vals = []

for gen_needs in Generator_needed:
      avg = np.mean(gen_needs)
      avg_vals.append(avg)
      sorted_array = np.sort(gen_needs)[::-1]  #Sorterar i fallande ordning
      max_value = sorted_array[0]
print("Max Values:", Max_per_day)
print("Average values (mean)", avg_vals)


#Tar fram totala summan för generatorbehovet i de olika batterifallen
for i in range(len(Generator_needed)):
      Januarisumma = sum(Generator_needed[i])
      Max = Max_per_day[i]
      print(f'{i+1} batteri: Summan för hela januari {Januarisumma}')

#Tar fram generatorbehovet för varje dag med datummarkering
Gen_day = []
Max_day = []
for i in range(len(Generator_needed)):
      gen_series = pd.Series(Generator_needed[i], index=index, dtype=float)
      gen_day = gen_series.resample('D').sum()
      Gen_day.append(gen_day)

for i in range(len(Generator_needed)):
      gen_series = pd.Series(Generator_needed[i], index=index, dtype=float)
      max_day = gen_series.resample('D').max()
      Max_day.append(max_day)

#Plotta generatorbehovet per dag över hela januari
axis = np.arange(1,32)
fig, ax = plt.subplots()
plt.title('Generatorbehovet per dag summerat, 90% solceller')
ax.plot(axis, Gen_day[0], label = '1 batteri', color = 'orange')
ax.plot(axis, Gen_day[1], label = '2 batteri', color = 'green')
ax.plot(axis, Gen_day[2], label = '3 batteri', color = 'red')
plt.xlabel('Tid [dag]')
plt.ylabel('Effektbehov generator [W]')
ax.legend(loc = 'lower left')

#Plottar generatorbehovet över hela månadens timmar
axis = np.arange(1,32)
fig, ax = plt.subplots()
plt.title('Största effektunderskottet varje dag, 90% solceller')
ax.plot(axis, Max_day[0], label = '1 batteri', color = 'orange')
ax.plot(axis, Max_day[1], label = '2 batteri', color = 'royalblue')
ax.plot(axis, Max_day[2], label = '3 batteri', color = 'brown') #Samma som
för 2!
plt.xlabel('Tid [dag]')
plt.ylabel('Effektbehov generator [W]')
```

```
ax.legend(loc = 'lower left')

axis = np.arange(1, 745)

fig, axs = plt.subplots(3, 1, figsize=(10, 8), sharex=True, sharey=True)

fig.suptitle('Batteriets upp- och urladdning, 90% solceller')

# Subplot för 1 batteri
axs[0].plot(axis, C_bat[0][0:744], label='1 batteri', color='orange')
axs[0].set_ylabel('Laddningstillstånd [W]')
axs[0].legend(loc='lower left')
axs[0].set_title('1 batteri')

# Subplot för 2 batteri
axs[1].plot(axis, C_bat[1][0:744], label='2 batteri', color='red')
axs[1].set_ylabel('Laddningstillstånd [W]')
axs[1].legend(loc='lower left')
axs[1].set_title('2 batteri')

# Subplot för 3 batteri
axs[2].plot(axis, C_bat[2][0:744], label='3 batteri', color='blue')
axs[2].set_xlabel('Timmar [h]')
axs[2].set_ylabel('Laddningstillstånd [W]')
axs[2].legend(loc='lower left')
axs[2].set_title('3 batteri')

# Justera layouten
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

#Plottar generatorbehovet över hela månadens timmar
axis = np.arange(1,745)
fig, ax = plt.subplots()
plt.title('Generatorbehovet månad, 90% solceller')
ax.plot(axis, Generator_needed[0], label = '1 batteri', color = 'orange')
ax.plot(axis, Generator_needed[1], label = '2 batteri', color = 'royalblue')
ax.plot(axis, Generator_needed[2], label = '3 batteri', color = 'brown')
#Samma som för 2!
plt.xlabel('Timmar [h]')
plt.ylabel('Effektbehov generator [W]')
ax.legend(loc = 'lower left')

#Plottar produktionen och användningen för olika antal solceller
fig, ax = plt.subplots()
plt.title('Produktion jämfört med användning, 90% solceller')
plt.xlabel('Timmar [h]')
plt.ylabel('Effekt [W]')
ax.plot(axis, L_vec_month, color ='red', label = 'Effektbehov')
ax.plot(axis, P_month, color = 'green', label = 'Produktion')
plt.legend(loc = 'lower left')

# Skapa en figur med 3 rader och 1 kolumn av subplots
```

```
fig, axs = plt.subplots(3, 1, figsize=(10, 8), sharex=True)

# Sätt titel för hela figuren
fig.suptitle('Effekt till/från nätet, 90% solceller')

# Bestäm y-axelns gränser
y_min = min(min(P_grid_state[0]), min(P_grid_state[1]), min(P_grid_state[2]))
y_max = max(max(P_grid_state[0]), max(P_grid_state[1]), max(P_grid_state[2]))

# Subplot 1
axs[0].plot(axis, P_grid_state[0], label='1 batteri', color='teal')
axs[0].set_ylabel('Effekt [W]')
axs[0].set_ylim(y_min, y_max)
axs[0].legend(loc='lower left')

# Subplot 2
axs[1].plot(axis, P_grid_state[1], label='2 batterier', color='pink')
axs[1].set_ylabel('Effekt [W]')
axs[1].set_ylim(y_min, y_max)
axs[1].legend(loc='lower left')

# Subplot 3
axs[2].plot(axis, P_grid_state[2], label='3 batterier', color='red')
axs[2].set_xlabel('Timmar [h]')
axs[2].set_ylabel('Effekt [W]')
axs[2].set_ylim(y_min, y_max)
axs[2].legend(loc='lower left')

# Justera layouten för att undvika överlappning
plt.tight_layout(rect=[0, 0, 1, 0.96])

# Visa plotten
plt.show()
```

-