Assignment 2: Modelling Electricity Grid Demand (15 Marks)

ENGG1001: Programming for Engineers — Semester 2, 2022

Due: Friday 23 September 2022 at 5:00pm

Introduction

The shift to renewable electricity in Australia has involved widespread installation of rooftop solar panels, which has reduced daytime grid electricity demand, and thereby increased the variability in grid demand over the day, as shown in Figure 1.

The power supplied by rooftop solar depends on a range of factors, including local variability in weather, regional and seasonal variation in day-length and intensity of solar radiation, differences in solar installation incentives between states and year-on-year increase in energy exported from rooftop solar to the grid. Changes in battery technology and solar feed-in tariffs (i.e. how much individual households are paid for exporting solar energy to the grid) add to the complexity of predicting grid electricity demand and price.

Predicting demand for grid electricity is important for both short-term and long-term planning, to inform decarbonization of the sector and to manage energy security. In this assignment, you will use your programming skills to apply and evaluate a simple model of grid electricity demand in Queensland.

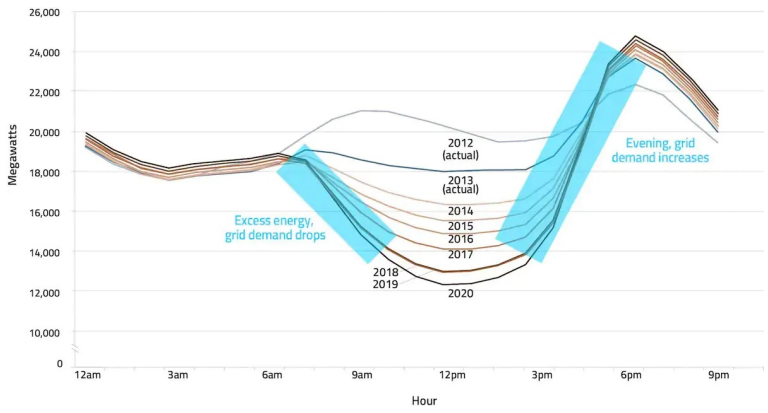

Figure 1: *Characteristic daily cycle in grid demand, referred to as the "duck curve". Over time, the increase in rooftop solar leads to a decrease in grid demand in the middle of the day. Source: AGL Energy* `https://www.agl.com.au/thehub/articles/2020/03/explainer-the-duck-curve`, *modified from* `https://www.caiso.com/documents/dr-eeroadmap.pdf`

Predicting grid demand

Demand for grid electricity varies over multiple time-frames. Grid demand varies on a 12 hourly pattern, with highest demand in the morning and evenings. It also varies over the day as shown in Figure 1, driven by the daily cycle in solar radiation and hence electricity generated roof-top solar. Furthermore solar radiation (and hence solar-generated electricity) varies sinusoidally over the year, with higher noon-time irradiance in summer compared to winter. Finally, despite increased uptake of energy efficient appliances, there has been an observed year-on-year increase in consumption of grid electricity in Australia's National Energy Market (NEM) for the past two decades.

These processes are captured in Equation 1, a simple model for grid demand (MW) in Queensland which you will apply in this assignment:

$$gd(t) = \sum_{i=0}^{2} A_i sin(\frac{2\pi t}{T_i} + \phi_i) + A_3 + \beta t \tag{1}$$

where t is the time in hours since 1 January 2017 at time 00:00, T_i is the period in hours of the cycle with amplitude A_i, A_3 is the baseline grid demand (MW) and β is the rate of increase in grid demand over time. ϕ_i is the phase offset for each of the cycles: it accounts for the timing of the sinusoidal curves relative to t = 0 at 1 January 2017 at time 00:00. All parameter values and corresponding units are defined in Table 1 (however unit conversion will be required to apply some of the values in Table 1 in Equation 1).

Parameter	Value	Units	Physical significance
A_0	800	MW	Amplitude of 12 h cycle in grid demand
A_1	-400	MW	Amplitude of 24 h cycle in grid demand
A_2	200	MW	Amplitude of annual (365 d) cycle in grid demand
A_3	6000	MW	Baseline grid demand
ϕ_0	0	radians	Phase offset for 12 h cycle in demand
ϕ_1	0	radians	Phase offset for 24 h cycle in demand
ϕ_2	0	radians	Phase offset for 365 d cycle in demand
β	0.14	MW / d	Rate of increase in baseline grid demand

Table 1: *Parameter values for Equation 1*

Task 1 - Predict grid demand **(1 mark)**

Write a function `predict_demand` that receives the number of hours since 1 January 2017 00:00, and the model parameters in the form of list of floats $a = [A0, A1, A2, A3]$, list of floats $phi = [\phi_0, \phi_1, \phi_2]$ and float $beta = \beta$, and returns the predicted grid demand in MW at the time t_hours.

```
def predict_demand(t_hours, a, p, b):
    """
    Calculates the grid demand D at time t_hours, for parameters a, p, b

    Parameters
    ----------
    t_hours : float
        Hours since 1 January 2017 00:00.
    a : list[float]
        Demand (MW) corresponding to 12 h cycle, 24 h cycle, 365 d cycle and
        baseline demand.
    p : list[float]
        Phase shift (radians) corresponding to 12 h cycle, 24 h cycle, 365 d
        cycle.
    b : float
        Rate of change in baseline demand (0.14 MW/d).

    Returns
    -------
    gd : float
        Predicted grid demand (MW) at time t_hours.
    """
    ...
```

You can easily check if your function is working by substituting values into Equation 1 using your calculator, and comparing them against the output. Sample output:

```
>>> A = [800, -400, 200, 6000]
>>> phi = [0, 0, 0]
>>> beta = 0.14
>>> gd = predict_demand(125.5, A, phi, beta)
>>> gd
    5829.188262566639
```

HINT: In implementing equations in your code, pay particular attention to units: incorrect units is a common source of errors. `predict_demand` MUST work for arguments given in the units specified in Table 1.

Task 2 - Predict half-hourly grid demand for 1 year, using lists (1 mark)

Write a function `demand_year_2` that receives a year in the range 2017 - 2019 and model parameters as before (list A, list ϕ and float β, with units as defined in `predict_demand` specification). If the input year is not in the required range, the function runs for the year 2017, and prints a message "Year not in range! Default value of 2017 will be used". The function returns a list containing the number of hours since 1 January 2017 00:00, at thirty minute intervals, throughout the year of interest (or for 2017 if the input year is not in range), predictions of grid demand (MW) corresponding to each time period stored in the first list, and a tuple which contains the mean, the standard deviation and the coefficient of variation of the predicted grid demand for the year in question. It also returns the year for which the lists were generated (= 2017 if the input year is out of range)

For n observations of a variable X, the mean $\overline{X}$ is calculated as follows:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{2}$$

The standard deviation s_X is a measure of variability, and for large samples such as these (n » 30), can be calculated.

$$S_X = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n}} \tag{3}$$

The coefficient of variation CV indicates the scale of variability in the data, by expressing the standard deviation as a proportion of the mean:

$$CV_X = \frac{S_X}{\overline{X}} \tag{4}$$

(Optional: simplify Eqn 3 by substituting in Eqn 2 to reduce the number of loops needed)

```
def demand_year_2(year, a, p, b):
    """ Predicts grid demand for year at half-hourly time steps, using lists

    Generates a list of 30 minute time intervals over year, and corresponding
    predictions of grid demand (MW) predicted using parameters a, p, b.
    If year is not in the range 2017 - 2019, the function runs for year 2017
    and prints a statement indicating that the year 2017 is used

    Parameters
    ----------
    year : int
        Year for which demand is to be calculated.
    a : list[float]: Demand (MW) corresponding to 12 h , 24 h and 365 d cycles,
    ↪ and baseline demand.
    p : list[float]: Phase shift (radians) corresponding to 12 h, 24 h and 365 d
        cycles.
    b : float: Rate of change in baseline demand (0.14 MW/d).

    Returns
    -------
    h_list : list[float]: Hours since 1 January 00:00 for year, or 2017 if year
    ↪ out of range.
    gd_list : list[float]: Grid demand (MW) predicted for each element in h_list.
```

```
    stats: tuple[float]: Mean, standard deviation and coefficient of variation of
↪   data in         gd_list, all rounded to 2 decimal places.
    year : int: year for which the function is run (2017 if year parameter is out
↪   of
        range).    """
    ...
```

An example is shown in the console session shown below. You should also get the same results if you run the function for the year 2017.

```
>>> A = [800, -400, 200, 6000]
>>> phi = [0, 0, 0]
>>> beta = 0.14
>>> hours_2, demand_2, stats_2, year_2 = demand_year_2(2021, A, phi, beta)
    Year not in range! Default value of 2017 will be used
>>> year_2
    2017
>>> hours_2[0:5]
    [0.0, 0.5, 1.0, 1.5, 2.0]
>>> demand_2[0:5]
    [6000.0, 6154.919401718796, 6296.621666999367, 6412.835979540702,
↪   6493.118893034501]
>>> stats_2
     (6025.54854166668, 645.7279228110234, 0.10716500221445625)
```

HINT: Debugging is an essential part of programming, and you can make debugging more efficient by finding ways to test and check your codes and your results as you go. As an engineer, these skills are highly valuable, because engineers need to find ways to check their own work: how do you know if your result is correct? Here are some other ways to test the function `demand_year_2l` :

- *Check that the returned values are the right size* Use the `len()` command to check the length of the lists returned by the function: the two lists should be the same length, and the length should be double the number of hours in a non-leap year, since the lists should cover a year with half-hourly intervals
- *Check what happens when an incorrect value is input to the function* What happens when you run the function with a year outside the range 2017-2019? It should return the same values as shown above for the year 2017, along with a message that the default year is being used
- *Qualitative check on results* From Eqn 1, you can see that the half-hourly grid demand model has a background value, oscillating signals with periods of 12h, 24 h and one year, and a gradual increase over time. The sine curves will cancel over the year, so the mean should increase gradually from years 2017 to 2019, due to the trend. Check the mean when you run the function for 2017, 2018, 2019: does it gradually increase at the rate expected from β in Table 1?
- *Quantitative check on results* You can predict analytically from Equation 1-2 and Table 1 that the mean half-hourly grid demand over the period Δt will be $\overline{X} = A_0 + 0.5\beta\Delta t$. Does this match your results?

Task 3 - Predict half-hourly grid demand for 1 year using arrays(1 mark)

So far you have used the programming fundamentals you learnt in Module 1 to run the grid model (Equation 1). However the same tasks can be written more simply and executed more efficiently using **NumPy** arrays. Therefore for the rest of this assignment you will use arrays to run, evaluate and explore the grid demand model.

Write a function `demand_year_3` that receives a year and model parameters as before (list A, list ϕ and float β), to perform the same task as `demand_year_2`, using NumPy arrays instead of lists. The function returns two NumPy arrays (hours since 1 January 2017 00:00 in 30 minute intervals for the year specified, and an array of the corresponding predicted grid demand, in MW), a tuple containing the mean, standard deviation and coefficient variation of predicted grid demand, and the year for which the information was produced (=2017 if the input year was out of range). If the input year is not in range, the message "Year not in range! Default value of 2017 will be used" is printed.

```
def demand_year_3(year, a, p, b):
    """
    Predicts grid demand for year at half-hourly time steps, using arrays.

    Generates arrays of 30 minute time intervals over year, and corresponding
    predictions
    of grid demand (MW) using parameters a, p, b If year is not in the range
    2017 - 2019, the function runs for year 2017 and prints a statement
    indicating the the year 2017 is used

    Parameters
    ----------
    year : int
        Year for which demand is to be calculated.
    a : list[float]
        Demand (MW) corresponding to 12 h, 24 h, and 365 d cycles and
        baseline demand.
    p : list[float]
        Phase shift (radians) corresponding to 12 h, 24 h, and 365 d cycles.
    b : float
        Rate of change in baseline demand (0.14 MW/d).

    Returns
    -------
    h_array : ndarray[float]
        Hours since 1 January 00:00 for year, or 2017 if year out of range.
    gd_array : ndarray[float]
        Grid demand (MW) predicted for each element in h_array.
    stats: tuple[float]
        Mean, standard deviation and coefficient of varation of data in gd_array.
    year : int
        year for which the function is run (2017 if year parameter is out of
    range).
    """
    ...
```

HINT:

- The function `predict_demand` was written to take in lists of floats and return lists of floats, but it can also take in and return Numpy arrays, which will simply your code in `demand_year_3`
- The function `demand_year_3` should produce the same results at `demand_year_2`, you can check the output of the two functions against each other for each year. There may be very small differences due to inaccuracies in storing floating point numbers.

An example is shown in the console session shown below, using the parameter values shown earlier.

```
>>> hours_3, demand_3, stats_3, year = demand_year_3(2018, A, phi, beta)
>>> hours_3[0:5]
    array([8760. , 8760.5, 8761. , 8761.5, 8762. ])]
>>> demand_3[0:5]
    array([6051.1       , 6206.01940172, 6347.721667  , 6463.93597954,
       6544.21889303])
>>> stats_3
     (6076.6485416666665, 645.7279228111984, 0.10626382592041303)
```

Task 4 - Plot grid demand from arrays **(2 marks)**

Write a function `plot_demand_byday` which receives two numpy arrays and a string. The first array contains hours since 1 January 2017 00:00, and the second column contains grid demand (MW). The string contains the super title for the plot. If no string is provided, the function should run with a default empty super title. See sample plot Fig 2.

```
def plot_demand_byday(x, y, plot_title = ""):
    """
    Plots grid demand for the first 28 days of data, and vs hours since midnight
    for all data

    Parameters
    ----------
    x : ndarray[floats]
        1d array of hours since 1 January 2017 00:00.
    y : ndarray[floats]
        Grid demand (MW) at corresponding time in x for each element.
    plot_title : str, optional.
        Super title of the figure. The default is ""
    Side-effects
    -------
    Generates two plots, stacked vertically.
    Top plot: Line plot of grid demand (MW) on y axis, vs time
              (hours since 1 January 2017 00:00) for first 28 days of data
    Lower plot: Scatter plot of grid demand (MW) on y axis,
              vs hours since midnight on x axis for all data
    Returns
    -------
    None.
```

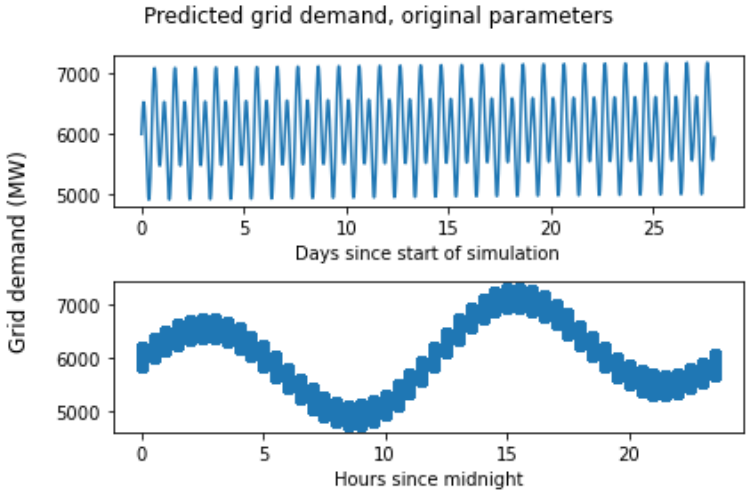

Figure 2: *Sample plot generated by* `plot_demand_byday`

Task 5 - Import data from file for comparison with predictions (2 marks)

It's important to test models against reality. One way to do this is to compare predictions with observations. Write a function `import_predict` which receives a filename and model parameters A, ϕ and β, imports measured hourly-hourly grid demand data from the file, and returns a Numpy array of floats which contains hours since 1 January 2017 00:00 in the first column, observed grid demand (MW) in the second column, predicted grid demand (MW) in the third column, and residuals (observed - predicted grid demand, MW), where the values in each row of the final three columns correspond to the time in that row in the first column.

```
def import_predict(fname, a, p, b):
    """ Generates array with time and grid demand from file, predicted grid demand
    and residuals

    Parameters
    ----------
    fname : str
        Name of file to import.
    a : list[float]
        Demand (MW) corresponding to 12 h, 24 h, and 365 d cycles and
        baseline demand.
    p : list[float]
        Phase shift (radians) corresponding to 12 h, 24 h, and 365 d cycles.
    b : float
        Rate of change in baseline demand (0.14 MW/d).

    Returns
    -------
    data_model : ndarray[float]: This array has four columns: The first column is
    ↪ hours since 1 January 2017 00:00, and the second column is grid demand (MW),
    ↪ both imported from file fname. The third column is grid demand (MW), predicted
    ↪ at each corresponding time in column 1, using parameters a, p, b. The fourth
    ↪ column contains the model residuals: observed - predicted grid demand (MW) """
    ...
```

Sample output, using the same parameter values used earlier:

```
>>> fname = 'ass2_qld.csv'
>>> data_model0 = import_predict(fname, A, phi, beta)
>>> data_model0[0:4,:]
array([[ 0.00000000e+00,  6.72899000e+03,  6.00000000e+03,
         7.28990000e+02],
       [ 5.00000000e-01,  6.46214000e+03,  6.15491940e+03,
         3.07220598e+02],
       [ 1.00000000e+00,  6.35282000e+03,  6.29662167e+03,
         5.61983330e+01],
       [ 1.50000000e+00,  6.26102000e+03,  6.41283598e+03,
        -1.51815980e+02]])
```

HINT: You can also check the values generated against the values in the file imported, and corresponding predictions from `demand_year_3`.

Task 6 - Evaluate the grid demand model **(2 marks)**

Having imported the grid demand data from file in the previous task, the next step is to compare the data with the model. This comparison should always involve a combination of visualisation (plotting) and quantitative assessment of model performance.

Write a function `evaluate_model` which accepts an ndarray with four columns (time, observed grid demand, predicted grid demand and residuals, as per output from `import_predict`), generates a figure with four subplots as shown in Fig 3, and returns a tuple which contains three elements: the mean of the data, the standard deviation of the data, and the root mean square error (RMSE) of the grid demand model, all rounded to 2 decimal places.

Root mean square error is a useful measure of model performance, and is calculated as follows, from the difference between the observations y_i and the predictions $\hat{y}_i$ over the n data points:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{5}$$

It's useful to compare the RMSE to the mean and the standard deviation in the original data, to get a sense of the scale of error in the model, compared to the mean and variability in the data.

```
def evaluate_model(data_model, plot_title = ""):
    """
    Generates four subplots comparing model and predictions in data_model

    Parameters
    ----------
    data_model : ndarray[float]
        This array has four columns:
        The first column is hours since 1 January 2017 00:00.
        The second column is observed grid demand (MW),
        corresponding to each time in column 1.
        The third column is grid demand (MW), predicted at each corresponding
        time in column 1, using parameters a, p, b
        The fourth column contains the model residuals: observed - predicted
        grid demand (MW).

    Side-effects
    -------
    Generates four sub-plots:
        Upper left: Scatter plot observed vs predicted grid demand, with 1:1 line
        Upper right: Boxplot of observed and predicted grid demand
        Lower left: Scatter plot of residuals vs hours since midnight
        Lower right: Histogram of residuals, with 10 bins

    Returns
    -------
    mstats: tuple[float]
        Mean and standard deviation of observed grid demand, and RMSE
        of model compared to data, all in MW and rounded to 2 decimal places.
    """
    ...
```

Sample output, using the array generated using the previously specified filename and parameters:

```
>>> data_model0 = import_predict(fname, A, phi, beta)
>>> pt = '''Grid demand and residuals (observed - predicted)
        for Eqn 1 with original parameters'''
>>> mstats0 = evaluate_model(data_model0, pt)
>>> mstats0
    (6228.0, 873.96, 1085.54)
```

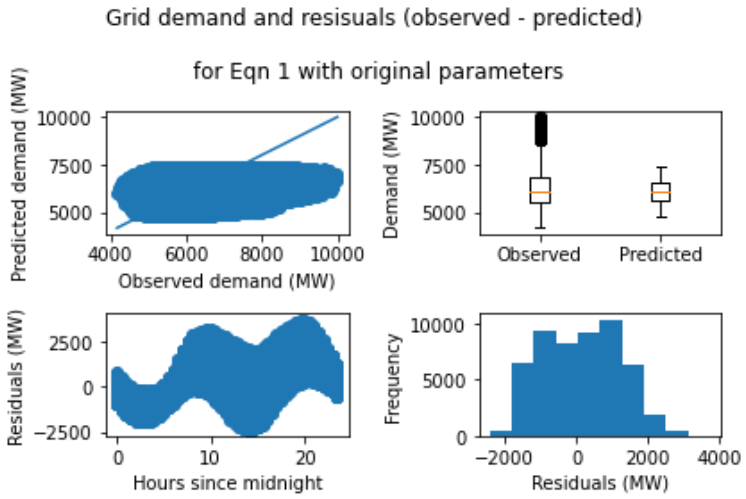

Figure 3: *Sample plot generated by* `evaluate_model`

Refining the grid demand model

Since all models are wrong and only some are useful, modelling is an iterative process of evaluation and refinement. The engineer's task is not to make the "perfect" model, but to ensure that the model is fit for purpose: as good as reasonably possible given the available time, resources and data, and able to generate predictions of the required accuracy for the specific application in which the model will be used.

From the comparison between the grid demand model and the half-hourly Queensland grid demand data shown in Figure 3 and from the output of `evaluate_model`, we can see that:

1. The grid demand model predicts approximately the right magnitude for the 25th, 50th and 75th percentiles
2. Variability in the data is much higher than predicted however: the lowest observed grid demand is more than 20 % below than the lowest predicted values, and the highest observed grid demand is more than 50 % higher than the highest predicted values.
3. Hence the RMSE is almost 20 % of the mean, and 25 % higher than the standard deviation
4. There is also something wrong with the way the model represents variability in the grid demand over the day. This is called a "systematic error", because there is an obvious pattern in the residuals over the daily cycle: the model consistently under-predicts in some parts of the day, and over-predicts in other parts of the day.

Task 7 Refine the model **(3 marks)**

The current model clearly doesn't capture the daily changes in grid demand very well, as shown in Fig 3. From reviewing Equation 1 and the plots generated from `evaluate_model`, we can now trial some different parameters in Equation 1.

Write a function `compare_model` which receives an array containing four columns (time, observed grid demand, predicted grid demand and residuals, as per output from `import_predict`), new values of the model parameters A, ϕ and β. The function returns an array with four columns: the first two columns are the same as those in the original array, the third column is grid demand predicted from the values of time in the first column, using the parameters provided to the function, and the final column is the new residuals.

```
def compare_model(data_model, anew, pnew, bnew):
    """
    Runs grid demand model with new parameter values, and compares with gd0

    Parameters
    ----------
    data_model :ndarray[float]:
        This array has four columns: (time, observed grid demand, predicted grid demand and residuals using the original parameters, as per output from \verb|import_predict|).
    anew : list[float]
        New demand (MW) parameters corresponding to 12 h, 24 h, and 365 d cycles and baseline demand.
    pnew : list[float]
        New values of phase shift (radians) corresponding to 12 h, 24 h, and 365 d cycles
    bnew : float
        New rate of change in baseline demand (0.14 MW/d).

    Returns
    -------
    data_model_new :ndarray[float]:
        This array has four columns: The first column is hours since 1 January
        2017 00:00. The second column is observed grid demand (MW),
        corresponding to each time in column 1. The third column is grid demand
        (MW), predicted at each corresponding time in column 1, using
        parameters in Model1 (a1, p1, b1) The fourth column contains the model
        residuals: observed - predicted grid demand (MW).
    rmses: list[float]
        RMSE of the original model, and of the revised model, rounded to 2 decimal places, for the new model parameters.


    Side-effects
    ------
        Generates plots by calling first plot_demand_byday, then evaluate_model.
        Prints a statement "The revised parameters produce better fit to the data: ...", if RMSE of the revised parameters is less than the original RMSE, and otherwise prints "The original parameters produce better fit to the data ..." as per the sample output  """
    ...
```

Sample output:

```
>>> A1 = [800, -400, 200, 6000]
>>> phi1 = [-np.pi/2, -np.pi/2, np.pi/2]
>>> data_model1, rmse01 = compare_model(data_model0, A1, phi1, beta)
The revised parameters produce better fit to the data:
             RMSE = 971.5 < 1085.54
```

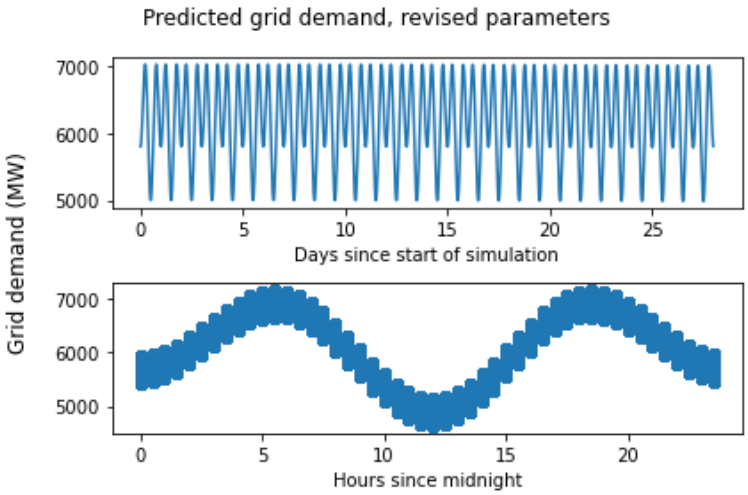

Figure 4: *Sample plot generated by* `plot_demand_byday`*, called by* `compare_model`*, for revised parameters*

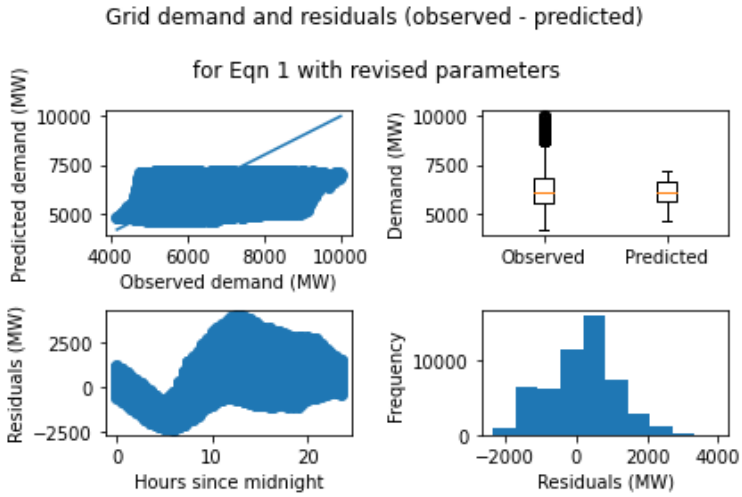

Figure 5: *Sample plot generated by* `evaluate_model`*, called by* `compare_model`*, for revised parameters*

Hint:

- Check your code by running `compare_model` with the original parameters.
- Gradescope requires you to produce Fig 4 first, then Fig 5.

Assessment and Marking Criteria

The maximum achievable mark for this assignment is **15 marks**.

Functionality Assessment — 12 marks

The functionality will be marked out of 12. For each task, your assignment will be put through a set of tests, and the functionality mark will be based on how your code performs on the tests. You will be given the functionality tests before the due date for the assignment so that you can gain a good idea of the correctness of your assignment yourself before submitting. You should, however, make sure that your program meets all the specifications given in the task sheet. That will ensure that your code passes all the tests. **Note: Functionality tests are automated and so string outputs need to exactly match what is expected.**

Hint: Do not leave it until the last minute because it generally takes time to get your code to pass the tests. Take a systematic approach to code testing:

- Tackle one task at a time
- Test your code in IDLE as you write it, to ensure that it runs
- Once your code is running, check that the values produced make sense (code can run successfully but not complete the task correctly)
- Temporarily modifying functions, e.g. adding print statements within functions, can help you track down the issue if your code is producing erroneous results
- Once you complete each Task, test your assignment code in Gradescope
- Gradescope will test if your function completes the specified task, not just if it completes the task for the example data provided. For example, it will test if `predict_demand` produces the correct predictions if different parameters are used, or if `import_predict` works for files other than the one provided.

The maximum achievable functionality assessment marks is **12 marks**.

Code Style Assessment — 3 marks

The style of your assignment will be assessed by one of the tutors, and you will be marked according to the style rubric provided with the assignment. The style mark will be out of 3.

Assignment Submission

You must submit your completed assignment to Gradescope. The only file you submit should be a single Python file called **a2.py** (use this name — all lower case). This should be uploaded to Gradescope. You may submit your assignment multiple times before the deadline; only the last submission will be marked.

Late submission of the assignment will **not** be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on time, you may submit a request for an extension. See the course profile for details on how to apply for an extension.

Requests for extensions **must** be made according to the guidelines in the ECP.