Assignment 1: Gravity Battery (15 Marks)

ENGG1001: Programming for Engineers — Semester 1, 2025

Due: March 28th (Friday) 3:00pm

Introduction

There is a need to create more renewable energy to meet the growing demand in today's world. One way to achieve this is with a so-called "gravity battery". This kind of battery is not made from chemicals like lithium, but rather, uses the natural force of gravity for energy storage. One type of gravity battery stores water in an upper reservoir and releases it to power generating turbines at a much lower height. The water is repumped back to the upper reservoir during an off-peak period when electricity is quite cheap. It is released primarily from the upper reservoir during peak periods when electricity is expensive. The system, also known as a pumped hydro system, is generally able to generate significant revenue for the power generating company. A typical system is shown in Figure 1.

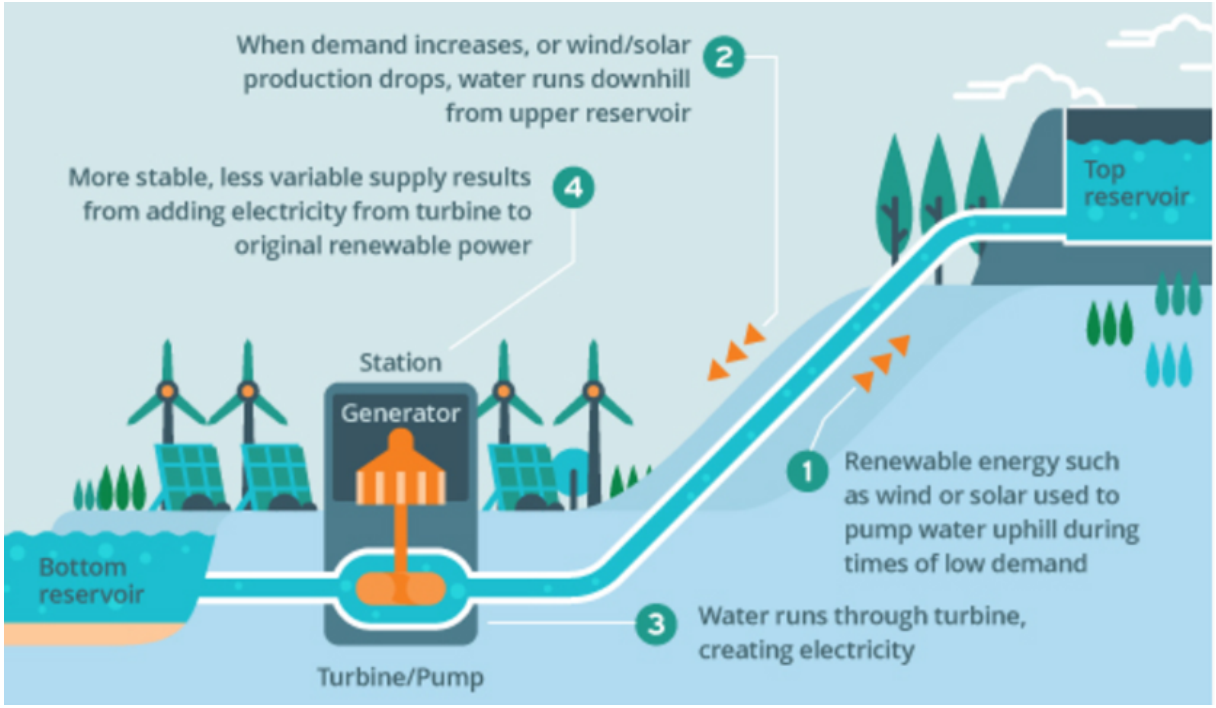

Figure 1: *An energy storage and generation system based on gravity*

In this assignment you will be doing some simulations to determine how water pumping, electricity generation and profitability are related.

Task 1 **(0.5 marks)**

The system relies on converting electrical energy to potential energy during off-peak periods and then converting it back to electrical energy during peak period. The relevant formulae for electrical and potential energy are given below in equations (1) and (2).

$$\text{Electical energy (J)} = \text{Electrical power (W)} \times \text{time (sec)} \tag{1}$$

$$\text{potential energy (J)} = \text{mass (kg)} \times \text{GRAVITY\_ACC (m/sec}^2\text{)} \times \text{elevation (m)} \tag{2}$$

where GRAVITY_ACC = 9.81 m/s^2 and elevation is the difference in height between the water in the lower and upper reservoirs. Ideally, the conversion between electrical and potential energy would occur without loss, but in practice there is some loss. Typically, the conversion efficiency is about 85%.

Making use of equations 1 and 2, write a function, `determine_power_used()`, which returns the power used **(in kW!)** to pump water upwards, given i) the amount of water that is pumped to the upper reservoir (in kg), ii) the average elevation (in m), iii) the number of hours the pumping is performed (in hours) and iv) the efficiency of energy conversion (%).

The definition line for the function should be:

```
def determine_power_used(water_mass, elevation, pumping_time, efficiency):
    """
    Calculates the power required to pump a certain mass of water a certain height
    taking into account the pumping_time and efficiency.

    Parameters:
        water_mass (float): the mass of the water pumped [kg]
        elevation (float): the height difference [m]
        pumping_time (float): the amount of time the pump is running for [hrs]
        efficiency (float): the conversion efficiency [%]

    Returns:
        (float): the power required by the pump [kW]
    """
```

A sample usage of the function is given below.

```
>>> power_used = determine_power_used(5e6, 250, 8, 85)
>>> power_used
500.9191176470588
```

HINT: Make sure to double-check what units the parameters are provided as, and what units the output is expected to be in.

Additionally, think about how the conversion efficiency would affect the power consumption: would the energy used by the pump be greater or lesser than the potential energy gain of the water?

Task 2 **(0.5 marks)**

Now write a function to determine how much water is released (in kg) to produce a given amount of electricity. The definition line for the required function is:

```
def determine_water_released(gen_power, elevation, pumping_time, efficiency):
    """
    Calculates the mass of water released required to generate a specified power

    Parameters:
        gen_power (float): the specified power to be generated [kW]
        elevation (float): the height difference [m]
        pumping_time (float): the time the pump is running for [hrs]
        efficiency (float): the conversion efficiency [%]

    Returns:
        (float): the mass of water required [kg]
    """
```

A sample usage of the function is:

```
>>> water_released = determine_water_released(300, 250, 8, 85)
>>> water_released
2994495.4128440367
```

Task 3 **(0.5 marks)**

In this task you are required to write a function to calculate the daily cost of using electricity to pump water. The cost to pump water upwards during the off-peak period is given by:

$$\text{pump\_cost} = \text{generator\_power (kW)} \times \text{pumping\_time (hrs)} \times \text{off-peak\_tariff (\$/kWh)} \quad (3)$$

The definition line for the function is given below.

```
def determine_cost_to_pump(gen_power, pumping_time, off_peak_tariff):
    """
    Calculates the cost of using the pump during off peak period

    Parameters:
        gen_power (float): the amount of power generated by the pump [kW]
        pumping_time (float): the amount of time the pump is running for [hrs]
        off_peak_tariff (float): the off-peak tarriff cost for electricity [$/kWh]

     Returns:
        (float): the cost of running the pump ($)
    """
```

A sample usage of the function is:

```
>>> daily_cost = determine_cost_to_pump(300, 8, 0.02)
>>> daily_cost
    48
```

Task 4 **(0.5 marks)**

As the peak period approaches the release valve in the upper reservoir will be opened and the pressure of the water in the reservoir will force water downwards to drive electric generators. You need to write a function, `calc_speed_at_outlet()`, to determine the water speed as it exits the bottom of the reservoir. Based on the principle of conservation of energy, the kinetic energy of the water leaving the reservoir must equal the change in potential energy of the water at the top of the reservoir. Thus:

$$0.5 \times \text{water\_mass\_out} \times \text{water\_speed}^2 = \text{water\_mass\_out} \times \text{water\_height} \times \text{GRAVIY\_ACC} \quad (4)$$

Hence:

$$\text{water\_speed} = \sqrt{2 \times \text{water\_height} \times \text{GRAVIY\_ACC}} \quad (5)$$

So the water speed at the bottom is solely dependent on the height of water in the reservoir. (Note that a number of assumptions were made in deriving (5), including the assumption that the water inside the reservoir is still). Use equation (5) to create the function, `calc_speed_at_outlet()`. The sole input parameter is water_height (m) and water_speed (m/s) is returned. The definition of the function should be:

```
def calc_speed_at_outlet(water_height):
    """
    Calculates the speed of the water at the outlet of the dam

    Parameters:
        water_height (float): the height of the water in the pipe [m]

    Returns:
        (float): the speed of the water at the outlet [m/s]
    """
```

A sample usage of the function is:

```
>>> water_speed = calc_speed_at_outlet(30)
>>> water_speed
24.261079942986875
```

Task 5 **(0.5 marks)**

As water drains from the reservoir the height of the water will gradually reduce. Write a function which determines how the water height reduces over a short time period, denoted by time_inc (secs). The function should take in the water height at the beginning of the time period (old_water_height) and return the water height at the end of the time period (new_water_height). You can use the following relations:

$$\begin{aligned} \text{water\_mass\_out} &= \text{water\_volume\_out} \times \text{WATER\_DENSITY} \\ &= \text{outlet\_area} \times \text{water\_speed} \times \text{time\_inc} \times \text{WATER\_DENSITY} \end{aligned} \quad (6)$$

$$\text{new\_water\_height} = \text{old\_water\_height} - \frac{\text{water\_mass\_out}}{\text{reservoir\_area} \times \text{WATER\_DENSITY}} \quad (7)$$

The definition line of the required function should be:

```
def calc_new_water_height(old_water_height, reservoir_area, outlet_area,
↪ time_inc):
    """
    """
```

The constant WATER_DENSITY is defined in the a1_support.py file. In addition to returning the new water height, the function should return the water mass out. A sample usage of the function is:

```
>>> new_water_height, water_mass_out = calc_new_water_height(20, 40000, 1, 30)
>>> new_water_height, water_mass_out
(19.985143183382704, 592567.1021442247)
```

Task 6 **(1.5 marks)**

In this task you need to simulate the way the height of the water in the upper reservoir reduces over an extended period of time. You also need to simulate the amount of water that flows out over this extended period. It is not possible to precisely model continuous time changes with a digital computer, so a discrete time approximation is used. The water height is assumed to be constant in relatively short discrete time intervals. For the simulation, the length of the discrete time intervals is denoted by time_inc and this length is typically set to about 30 seconds. In summary, you need to determine the water height in the upper reservoir and the amount of water out during the following time periods:

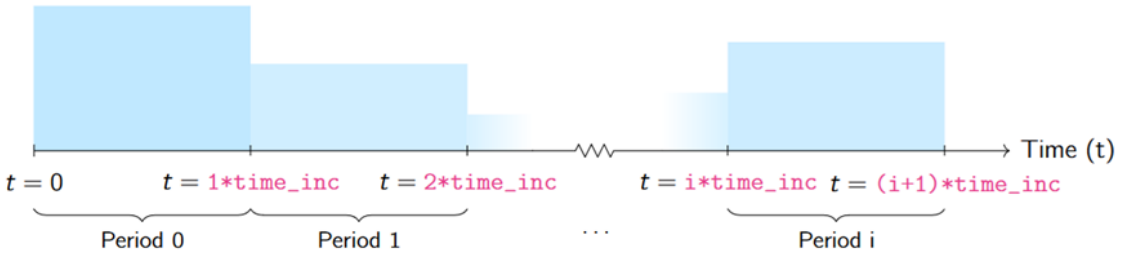

Figure 2: *Diagram of time interval approximation*

The initial water height when the release valve is opened is denoted as initial_height, and the height when the valve is closed is denoted as final_height. You can incorporate the function you wrote in Task 5 to determine the sequence of heights and water masses out. To do this simulation you should write a function which returns a tuple of the heights and a tuple of water masses out. The function should take in the initial height, the final height, the time increment, the reservoir base area and the outlet area. The required function line definition is:

```
def calc_heights_water_out(initial_height, final_height, reservoir_area,
↪ outlet_area, time_inc):
    """
    """
```

A sample use of this function is:

```
>>> heights, water_masses_out = calc_heights_water_out(40, 5, 40000, 1, 30)
>>> heights[0:3]
(39.97898928844613, 39.95798409574207, 39.93698442188801)
>>> water_masses_out
(838016.4324684857, 837796.3120398963, 837576.1916037091)
>>> len(heights), len(water_masses)
2461, 2461
```

Hint: To get started with this task you will want to use a loop of some-kind to successively iterate through the different heights. Here is a text-version of the algorithm you need to implement to make this work:

1. calculate the new height and the water mass out based upon the initial_height
2. check whether this new height is greater than the final height parameter. If it is, then go back to step 1, and change the initial_height value to the one just calculated
3. otherwise, end the loop and return all the values calculated (including the one just calculated)

Task 7 **(1 mark)**

You are now required to determine the energy produced as the upper reservoir drains. For each time period, i, the potential energy of the water is converted to electrical energy in the generators. This can be expressed mathematically as:

$$\text{energy[i]} = \text{water\_masses\_out[i]} \times \text{GRAVITY\_ACC} \times (\text{heights[i]} + \text{relative\_elevation}) \times \text{efficiency} \quad (8)$$

$$\text{power[i]} = \text{energy[i]} \ / \ \text{time\_inc} \quad (9)$$

where relative_elevation is the elevation of the upper reservoir base relative to the generators. Using the above equations and the function you wrote in the previous task, write a function to determine the electrical energy and electrical power corresponding to each successive period of time as the upper reservoir drains. The function should return a tuple of electrical energy values **[kWh]** and a tuple of electrical power values **[kW]**. Note that you will need to use appropiate unit conversions to obtain the desired output units for each calculation. The definition line for the function is:

```
1 def calc_energy_power(heights, water_mass_outs, relative_elevation, efficiency,
  ↪ time_inc):
2     """
3     """
```

A sample usage of the function is:

```
>>> water_heights, water_masses = calc_heights_water_out(30, 20, 40000, 1, 30)
>>> energy, power = calc_energy_power(water_heights, water_masses, 200, 85, 30)
>>> energy[0:3]
(387.7129370269342, 387.56468335928287, 387.4164575872225)
>>> power[0:3]
(46525.5524432321, 46507.762003113945, 46489.9749104667)
>>> len(energy), len(power)
605, 605
```

Task 8 **(0.5 marks)**

You are required to write a function, `daily_profit()`, which returns the daily profit, given the tuple, energy, from Task 8 and the following derivation:

$$\text{daily\_profit} = \text{revenue\_generated} - \text{cost\_of\_pumping} \tag{10}$$

Where:

$$\text{revenue\_generated} = \sum \text{energy\_generated} \times \text{peak\_tariff} \tag{11}$$

$$\text{pumping\_cost} = \sum \text{potential\_energy} \times \text{off\_peak\_tariff/efficiency} \tag{12}$$

Assuming that the amount of water that is pumped up is the same amount as the amount of water that is released, We can back-calculate the total potential energy for pumping thus:

$$\sum \text{potential\_energy} = \sum \text{energy\_generated/efficiency} \tag{13}$$

Therefore, our final derived equation is:

$$\text{total\_profit} = \sum \text{energy\_generated} \times (\text{peak\_tariff} - \sum \text{off\_peak\_tariffefficiency}^2) \tag{14}$$

where $\sum$ energy (kWh) is the sum of all the values in the tuple, energy, returned from `calc_energy_power()`, and peak_tariff and off_peak_tariff are the charges for energy consumption during peak and off-peak periods respectively ($/kWh). The definition line for the function is:

```
1 def calc_daily_profit(energy, peak_tariff, off_peak_tariff, efficiency):
2     """
3     """
```

A sample usage of the function is:

```
>>> water_heights, water_masses = calc_heights_water_out(30, 20, 40000, 1, 30)
>>> energy, power = calc_energy_power(water_heights, water_masses, 200, 85, 30)
>>> daily_profit = calc_daily_profit(energy, 0.02, 0.005, 85)
>>> daily_profit
2718.700033709491
```

Task 9 **(1.5 marks)**

This task involves creating a table which prints out the daily profit and total energy generated for different relative elevations. Note that the data should be printed to 2 decimal places of accuracy The function definition is below:

```
1 def print_table(start_relative_elevation, step_size, num_steps, initial_height,
  ↪ final_height, reservoir_area, outlet_area, time_inc, peak_tariff,
  ↪ off_peak_tariff, efficiency):
2     """
3     """
```

The required printout and sample usage of the function are shown below:

```
>>> print_table(280, 20, 6, 30, 20, 40000, 1, 30, 0.02, 0.005, 85)
#########################################################################################
# Relative elevation (m)  #    Daily Profit ($)     #   Total Energy (kWh)    #
#########################################################################################
#           280           #         3695.99         #        282577.18        #
#           300           #         3938.36         #        301107.34        #
#           320           #         4180.73         #        319637.50        #
#           340           #         4423.09         #        338167.66        #
#           360           #         4665.46         #        356697.82        #
#           380           #         4907.83         #        375227.97        #
#########################################################################################
```

Task 10 **(2 marks)**

This task involves writing a user interaction function. This function is called `main()`, and it has no inputs and returns nothing. An illustration of its operation is provided below. Note that the blue text is what the code outputs, and what is in black is what the user inputs.

The user will be able to input the following commands: h, r, p, s, and q. 'h' prints a help message for the user, 'r' reads test data, 'p' prints a table with the daily profit scenarios based upon a specified relative height, 's' displays a plot of the dam height for a given period of time based upon the loaded test data, 'q' quits the program.

```
Please enter a command: h

    'h' - provide help message
    'r' - read test data from a file
    'p' - {start_relative_elevation, step_size, no_steps} - print a table showing
↪   daily profit/total
    's' - plot the water height over time based on the test data
    'q' - terminate the program

Please enter a command: r
Please specify the directory: test
Please specify the filename: test_data.txt
Please enter a command: p 280 20 6
#########################################################################################
# Relative elevation (m)  #    Daily Profit ($)     #   Total Energy (kWh)    #
#########################################################################################
#           280           #         3695.99         #        282577.18        #
#           300           #         3938.36         #        301107.34        #
#           320           #         4180.73         #        319637.50        #
#           340           #         4423.09         #        338167.66        #
#           360           #         4665.46         #        356697.82        #
#           380           #         4907.83         #        375227.97        #
#########################################################################################
Please enter a command: s
Simulating water heights...
Please enter a command: q
Are you sure (y/n): n
Please enter a command: q
Are you sure (y/n): y
```

Note that when the code prints "Simulating water heights...", it will output a plot of the water heights over time. A pre-made function called `plot_water_height` is located in the `a1_support` file. An example plot produced by the function with the data `test_data_2.txt` is shown below:

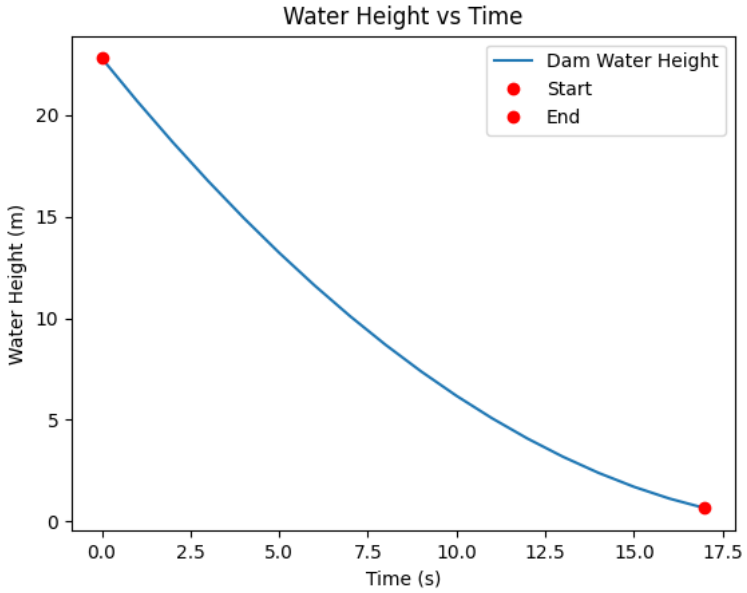

Figure 3: *Damn water height plot*

Additionally, your code is expected to be able to handle basic user errors. The errors that ought to be handled are shown in the example down below. Note that you do not need to worry about the file name being incorrect, as this error handling is out of scope.

```
Please enter a command: p 280 20 6
Please load data before using this command
Please enter a command: s
Please load data before using this command
Please enter a command: r
Please specify the directory: test
Please specify the filename: test_data.txt
Please enter a command: p
Please enter the correct number of arguments
Please enter a command: k
Please enter a valid command
Please enter a command: q
Are you sure (y/n): y
```

You are encouraged to incorporate the code you have written in the previous tasks when writing the code for main. You can also use the pre-written function, `load_data()`, in the a1_support.py to read in text from a file and return it in the form of a tuple of floats. Assuming that the relevant file to be read is called test_data.txt and that it is in the directory test. Then a sample usage of the load_data() function is:

```
>>> initial_height, final_height, reservoir_area, outlet_area, time_inc,
 ↪  peak_tariff, off_peak_tariff, efficiency = load_data("test", "test_data.txt")
```

Writing Your Code

You must download the file, a1.py, from Blackboard, and write your code in that file. When you submit your assignment to Gradescope you must submit only one file and **it must be called a1.py**. Do not submit any other files or it can disrupt the automatic code testing program which will grade your assignments.

Design

You are expected to use good programming practice in your code, and you must incorporate at least the functions described in the previous part of the assignment sheet.

Assessment Marking

The maximum achievable mark for this assignment is 15 marks.

Functionality assessment: **(9 marks)**

The functionality will be marked out of 9. Your assignment will be put through a series of tests and your functionality mark will be automatically generated by the autograder. This grade is NOT overwritten. So if you have a syntax error in your code which causes it to fail all the tests even though the rest of your code is fine, you will get 0 marks. The autograder will be available through Gradescope, which is also your submission portal. Your final functionality mark for the assignment is based on the most recent submission. You are permitted to submit as many times as you need, but do not leave it until the last minute as there are often small bugs in one's submission that are not picked up until the tests are run. Note that as functionality tests are automated and so string outputs need to exactly match what is expected. **Do not leave it until the last minute** because it generally takes time to get your code to pass the tests.

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 (in the Assignment 1 folder). The style mark will be out of 3.

Oral Presentation Assessment **(3 marks)**

In the week following the assignment due date, students will be required to attend a short interview with a tutor in the Help Center. Students will be asked questions about their assignment code and they will be given a mark out of 3, based on the quality of their answers. Students will need to sign up for one of the interview time slots. Details on how to sign up will follow.

Assignment Submission

You must submit your completed assignment to Gradescope. The only file you submit should be a single Python file called a1.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.