Explanation

- Choose any one of Q1-Q3 from the next page and write the code in python and submit it.
- Create a python library in the following directory according to the problem, and write the application code (code to run) in `main.py`.

  Q1 `van_del_pol/`

  Q2 `linear_regression/`

  Q3 `graph_mst/`
- For each problem, we specify what should be written in the library and what should be written in the `main.py`.
- Each method should be implemented in the library so that they can be reused and unittested.
- **Jupyter notebook submissions will not be accepted.** Please enter the supplied code in the specified location.
- If you used some libraries (Actually I am sure that you will), you need to list all the used libraries and their versions as text file in some formats, so that Arithmer can run the code:
  - `requirements.txt`;
  - `poetry.lock`;
  - Some other format which are well-used.
- Submit your code in the form of a zip file, and be careful not to publish your code on github, etc.

Q1. Numerical simulation of van del Pol equation

van del Pol equation

The van del Pol equation is a second-order ordinary differential equation expressed as follows:

$$x''(t) - a(1 - x(t)^2)x'(t) + x = 0$$

This equation was proposed by the Dutch electrical engineer B. van del Pol as a model for stable oscillations in a vacuum tube electric circuit. The task is to find the solution of this equation numerically and to illustrate the answer appropriately. Setting $y(t) = x'(t)$, we can rewrite this equation as the equivalent first-order vector-valued ODE as

$$\begin{aligned} x'(t) &= y(t) \\ y'(t) &= a(1 - x(t)^2)y(t) - x(t) \end{aligned}$$

So, here and henceforth, we consider the equation in this form.

Simulation method

There are various numerical methods for the general vector-valued differential equation $x'(t) = F(x(t))$. Here we consider the following two methods, where τ is the time step.

Explicit Euler method

$$x_{n+1} = x_n + \tau F(x_n)$$

Heun method

$$\begin{aligned} k_n^{(1)} &= F(x_n) \\ k_n^{(2)} &= F(x_n + \tau k_n^{(1)}) \\ x_{n+1} &= x_n + \frac{\tau}{2}(k_n^{(1)} + k_n^{(2)}) \end{aligned}$$

Assignment

Numerical computation

Create a python code to compute the solution of the vand del Pol equation using the Euler and Heun methods as a library in `van_del_pol/`.

It is up to the candidate to define the input/output function, but the rationale for the input/output is to be explained in the technical interview. You may investigate and implement other methods yourself if you wish as bonus.

Visualization

Using the `matplotlib` package, create a code in `main.py` to visualize the results of numerical computation and save the result to a file. The visualization method are as follows:

- Plot $x(t)$ and $y(t)$ versus time;
- The trajectory of $(x(t), y(t))$ in two-dimensional space.

Numerical Experiments

In the following cases, illustrate the numerical results of the Euler and Heun methods.

- $a = 2$ and $\tau = 0.1, 0.01, 0.001$.
- $a = 10$ and $\tau = 0.1, 0.01, 0.001$.

Discussion (Bonus)

Changing the value of the constant a and the time step τ may cause the calculation to break down. Find out when this happens and investigate the reason.

test/docstring (bonus)

Write appropriate docstrings and tests for your code.

Q2. Linear regression from scratch

`data/regression_train.csv` and `data/regression_test.csv` are measured human dimensional data. Using this data, the task is to create a linear regression model with height (`height`), weight (`weight`), and age (`age`) as explanatory variables and other dimension sizes (`BicepC`) as objective variables, and to evaluate it.

Linear regression model

Let N data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

be given as a pair of explanatory variables x ($1 \times m$-matrix) and a 1-dimensional objective variable y.

We want to find the coefficient β($m \times 1$-matrix) and bias β_0 of the linear regression model $y = x\beta + \beta_0$ that best explains this data. A well-known method is a least-square method, which minimizes the squared error given below:

$$E(\beta, \beta_0) = \sum_{j=1}^{N} [y_j - (x_j\beta + \beta_0)]^2 .$$

Assignment

Linear Regression (least-square method)

Write a python code that has the following two functions as a library in `linear_regression/`.

- For given data, find the coefficients β and the bias β_0 minimizing $E(\beta, \beta_0)$ defined above.
- For given coefficients β, bias β_0 and explanatory variable x, compute the estimate of y using the linear regression model.

Here,

- **Do not use existing libraries such as `scikit-learn` that can perform linear regression calculations themselves**, but write using `numpy`.
- You can freely decide how to provide input data and output.

Other regression model

Implement one regression model other than linear regression.

- **Do not use existing libraries such as `scikit-learn` that can perform linear regression calculations themselves**, but write using `numpy`.
- Create the python library code in a directory with an appropriate name other than `linear_regression`.

Then compare the error with the linear regression.

Application to data

Write the code in `main.py` to do the followings:

- For the training data (`data/regression_train.csv`), run a linear regression with height (`height`), weight (`weight`), and age (`age`) as explanatory variables and the other dimension size (`BicepC`) as the objective variable.
- Apply two regression models created above to the test data (`data/regression_test.csv`) and plot the true value of the objective variable on the horizontal axis, and the predictions of the regression model on the vertical axis. You can use `matplotlib` to create the plot.
- Calculate and compare the accuracies of two regression models.

test/docstring (bonus)

Write the appropriate docstring and test for your code.

Q3. Minimal spanning tree

Minimal spanning tree

An undirected graph G is a pair of the set V of vertices and the set E of edges connecting two vertices. A graph is called a weighted graph if the real number $w(e)$, called weight, is assigned to each edge $e \in E$.

A graph is called a tree if it has no closed path and is entirely connected. For a graph G, a spanning tree is a subgraph of G such that it contains all vertices of G and is a tree.

There can be more than one spanning tree. For a weighted graph G, the minimal spanning tree is the spanning tree of G which have the smallest sum of edge weights.

Algorithm for finding the minimal spanning tree

The following algorithms are popular.

Kruskal algorithm

This algorithm is as follows:

1. Initialize the set E_T of edges of the minimal spanning tree as an empty set.
2. Select the edge with the smallest weight in G and add it to E_T.
3. Select the edge e with te smallest weight among the previously unchecked edges, and
   - If the edges in E_T and e do not form a closed path, add e to E_T.
   - Otherwise, do not add e to E_T and skip it.
4. Repeat until all vertices are connected by edges in E_T.

Prim algorithm

This algorithm is as follows:

1. Initialize the vertex set V_T and the edge set E_T as empty sets.
2. Choose the vertex randomly and add it to V_T.
3. Add to E_T the edge e with the smallest weight among the edges connecting the vertex belonging to V_T and not belonging to V_T.
4. If e connects $u \in V_T$ and $v \notin V_T$, then add v to V_T.
5. Repeat until V_T contains all vertices.

Assignment

Implementation of the algorithm

Implement at least one of the Kruskal algorithm or Prim algorithm as python library in `graph_mst/`. The input of the graph is given as an adjacency matrix (a two-dimensional array of `numpy`) with weights as

components. For example, the graph denoted by the array

```
array(
[[0, 2, 0],
 [2, 0, 1],
 [0, 1, 0]]
)
```

is as follows:

- This graph has three vertices, which are v_1, v_2, and v_3;
- The weight of the edge between v1 and v2 is 2;
- The weight of the edge between v2 and v3 is 1;
- and there are no other edges.

Since we consider an undirected graph, the adjacency matrix should be symmetric.

You are free to choose what kind of output you want.

Calculation from a sample CSV file

`data/graph_adj.csv` is a CSV file of the adjacency matrix of a graph with 50 vertices.

Write code in `main.py` to read this CSV file and output the result of applying the algorithm for finding the minimum spanning tree above. The result will be as follows:

```
0,1
1,10
...
```

Here, each line shows the edges of the minimum spanning tree by writing the vertex numbers of the endpoints, separated by commas.

test/docstring (bonus)

Attach the appropriate test and docstring to your code.