This is a pdf version of https://git.tu-berlin.de/lis-public/ai-student-workspace/-/blob/main/02/README.md

To obtain the code for this assignment, you will need to fetch and pull new commits from git@git.tu-berlin.de:lis-public/ai-student-workspace.git

As always, only modify the file `solution_??.py`. And even in `solution_??.py`, only modify what the functions do - don't change the function's names. Don't add any additional files, put all your code in `solution_??.py`.

You can run tests by navigating to the task folder `??`, and then simply typing `python3 -m pytest`. If you haven't yet, you will need to install pytest first: `sudo apt install python3-pytest`.

Assignment 2: Single-Step and Discrete Decision Making

2.1: Single Step, Continuous, Data-based (aka supervised learning)

The following information is available:

- You have to choose an action a from a continuous interval $a \in [l, u] \subset \mathbb{R}$.
- Outcomes $y \in \mathbb{R}$ depend on the decision a in some unknown way: There is an underlying function f unknown to you, and observations are Gauss distributed around this function with $\sigma = 0.1$, i.e. $y \sim P(\cdot \mid a) = \mathcal{N}(f(a) \mid \sigma^2)$. You can assume that f is smooth.
- You have a data set $D = (a_i, y_i)_{i=1}^n$ of previous decisions $a_i \in \mathbb{R}$ and outcomes $y_i \in \mathbb{R}$.

For this part of the exercise, you will need to modify the following function in `solution_02.py`:

```
regression_based_on_data(l, u, a, y)
```

The inputs of this function are as follows:

- `l` (single float) is the left side of the interval in which the action a can be
- `u` (single float) is the right side of the interval in which the action a can be
- `a` is a `np.array` of length `n`, and contains actions a
- `y` is a `np.array` of length `n`, and contains the $y \sim P(\cdot \mid a)$ obtained for the `a` in `A`.

This function should return:

- The action $a_{\max}$ (single float) from the interval $[l, u]$ that maximizes the expectation $E[y \mid a]$. Since we know that $y \sim P(\cdot \mid a) = \mathcal{N}(f(a) \mid \sigma^2)$, this is the action $a_{\max}$ that maximizes $f(a)$

To do this, you will first need to perform the following steps:

1. Use the data (`a`,`y`) to find the approximate function $\hat{f}$ (this is your statistical estimator for the true f
2. Find the $a_{\max}$ that maximizes $\hat{f}(a)$ on $a \in [l, u]$.

You are free to use any function approximation to find $\hat{f}$ in the first step, but we suggest the following regression method:

1. We choose F different feature functions $\phi_0, \phi_1, \dots, \phi_{F-1}$. For example, we can use polynomial features: $\phi_k(a) = a^k$.
2. We make the ansatz $\hat{f}(a) = \sum_{k=0}^{F-1} \hat{\alpha}_k \; a^k$.
3. The estimated $\hat{\alpha}_k$ are chosen such that our estimated function $\hat{f}$ maximizes the likelihood of the data (`a`,`y`). As will be discussed in the tutorials, this is equivalent to calculating

$$\hat{\alpha} = (A^T A)^{-1} A^T y$$

Here, $\hat{\alpha}$ is a vector with elements $\hat{\alpha}_k$, and A is a $n \times F$ matrix whose elements are as follows:

$$A_{ij} = \phi_j(a_i) \quad ,$$

where the a_i are the elements of the vector `a`.

The larger you choose F, the larger the variance of your estimator $\hat{f}$ becomes. The smaller you choose F, the larger the bias of your estimator $\hat{f}$ becomes. There is a sweet spot somewhere in the middle! To get an intuition, it could be useful to plot `a`, `y`, and $\hat{f}$ together!

2.2: Single Step, Continuous, Simulation-based (aka active learning)

This is the same setting as in 2.1, but this time you don't have data $D = (a_i, y_i)_{i=1}^n$. Instead, the agent can query n times from $y \sim P(\cdot \mid a)$ (i.e. sample a resulting y for a given value n).

This part of the exercise is not scored, but be prepared to present your results in the tutorial. Write the code for this part of the exercise in the separate file `exercise_022.py`, not in `solution_02.py`. At the top of the file, the function `get_y_given_a` implements sampling from the black-box distribution $P(\cdot \mid a)$. These are the instructions:

1. Sample a data pair for an action of your choice. Let's denote the data after drawing $k \leq n$ samples as $D_k = (a_i, y_i)_{i=1}^k$. Start by choosing a couple of actions randomly, then proceed with step 3.
2. Generate $B = 10$ bootstrap resamples from the data D_k, which we denote as D_k^b. In other words: Randomly select k data pairs (with replacement!) from D_k, resulting in a new dataset D_k^b each time, and repeat this $B = 10$ times. For more details on bootstrapping, read https://isis.tu-berlin.de/pluginfile.php/2284732/mod_resource/content/1/04-notes.pdf.
3. For each bootstrapped dataset D_k^b, perform the regression described in 2.1. This results in an estimated function $\hat{f}_k^b$ for each of the datasets. Now you can calculate
    1. The bootstrap estimate of f: $\hat{f}_k(a) = \frac{1}{B} \sum_b \hat{f}_k^b(a)$
    2. The estimated variance of $\hat{f}_k$: $\hat{\sigma}_k(a) = \frac{1}{B-1} \sum_b (\hat{f}_k^b(a) - \hat{f}_k(a))^2$
4. Create a plot of $\hat{f}_k(a) \pm \sqrt{\hat{\sigma}_k(a)}$ alongside with D_k and the true f. You can use `matplotlib.pyplot.fill_between` to visualize the error bars.
5. Choose the next action to sample using UCB with $\beta = 2$. Then repeat from step 1.