This is a pdf version of https://git.tu-berlin.de/lis-public/ai-student-workspace/-/blob/main/01/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. Of course, you can import additional libraries, and add files with own code to import.

You can run tests by changing directory 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 1: Single-Step and Discrete Decision Making

1.1: Single Step, Discrete, Full knowledge

The following information is available:

- There are A possible choices.
- Depending on your choice $a \in 0, 1, \dots, A-1$, the outcome $x \in \mathbb{R}$ will be Gaussian distributed with mean μ_a and standard deviation σ_a, in other words: $P(x|a) = N(x; \mu_a, \sigma_a)$. We have full knowledge of the outcome of our actions, meaning that all μ_a and σ_a are known.

For this part of the exercise, you will need to modify the following three functions in `solution_01.py`:

1. `maximize_mean(mu_vector, sigma_vector)`
2. `maximize_ucb(mu_vector, sigma_vector)`
3. `maximize_utility(mu_vector, sigma_vector, utility_function)`

The inputs are as follows:

- `mu_vector` is a `numpy.array` vector of length `A`. It contains the μ_a for each of the A possible choices.
- `sigma_vector` is a `numpy.array` vector of length `A`. It contains the σ_a for each of the A possible choices.
- `utility_function` is only input to the last function, not to the others. `utility_function` is a function $R: \mathbb{R} \to \mathbb{R}$. The function object `utility_function` can be called using any real number $x \in \mathbb{R}$, and will return another real number y.

Each of these functions outputs a decision $a \in 1, 2, \dots, A$. Please modify these functions as follows:

1.1.1: Maximum Mean (1 Point)

Modify `maximize_mean(mu_vector, sigma_vector)` so that it outputs the decision a that maximizes the mean outcome $\int P(x|a)\, x\, \mathrm{d}x = \mu_a$. (Yes, this is trivial to implement.)

1.1.2: Maximum UCB (1 Point)

Modify `maximize_ucb(mu_vector, sigma_vector)` so that it outputs the decision a that maximizes the maximum upper confidence bound $\mu_a + \beta\sigma_a$ for $\beta = 2$.

1.1.3: Maximum Utility (1 Point)

Modify `maximize_utility(mu_vector, sigma_vector, utility_function)` so that it outputs the decision a that maximizes the expected utility with respect to an arbitrary utility function $R : \mathbb{R} \to \mathbb{R}$. In other words, it must output the decision a that maximizes

$$\int P(x|a)\; R(x)\; \mathrm{d}x \quad . \qquad (1)$$

The utility function R is given to `maximize_utility` as the function object `utility_function`, as described above.

We assume no knowledge on R, we can only query it for different values of x. That means that in order evaluate the integral in eq. (1), we need to use numerical methods like Monte-Carlo sampling. We assume a limited budget, i.e. the function R can only be sampled $n = 1000$ times.

1.2: Single Step, Discrete, Simulation-based (aka Bandits) (2 Points)

The following information is available:

- There are A possible choices.
- Depending on your choice $a \in 0, 1, \ldots, A-1$, we know that the the outcome $x \in [0, 1]$ will be strictly in the interval [0, 1]. However, in contrast to 1.1, the distribution $P(x|a)$ is not known (obviously, it is not Gaussian!).
- We have access to a stochastic simulation that we can access n times. Each time, the agent queries an action a_i, and the simulation returns an outcome x_i, where, again, $x_i \in [0, 1]$. Put differently, the agent makes n data-collecting decisions to collect data $D = (a_i, x_i)_{i=1}^n$.

For this part of the exercise, you will need to modify the following function in `solution_01.py`:

```
maximize_mean_using_simulator(simulator, A, n)
```

The inputs are as follows:

- `simulator` is a function that simulates the outcome of the actions. It returns the outcome `x = simulator(a)` depending on the action $a \in 0, 1, \ldots, A-1$.
- `A` is the number of actions that are possible. Your method must output an integer $a \in 0, 1, \ldots, A-1$.
- `n` is the budget for querying `simulator`. If you query `simulator` more often, it will raise an exception (error).

Modify `maximize_mean_using_simulator` so that it returns the action $a \in 0, 1, \ldots, A-1$ that maximizes the expected mean outcome $\int P(x|a)\; x\; \mathrm{d}x$. This is analogous to 1.1.1, but this time you don't know P.