STAT 243 Final Project Report

GitHub repository: https://github.berkeley.edu/yihong-zhu/ars-dev.git.

Soohyun Kim, Xiaotong Zhan, Yihong Zhu

I. Package description

1.1 Overview

Our Adaptive Rejection Sampling (ARS) package is designed using the functional programming paradigm. It contains two modules: `ARS.py` and `helper_functions.py`. When designing function, we emphasize modularity, reusability, and immutability of functions. The functional design enables easy testing and integration while simplifying debugging and maintenance. In following sections we briefly introduce the functions in each module, and the detailed description can be found by applying `help()` to the function.

1.2 `helper_function` moduel

The `helper_functions` moduel contains essential building blocks for the ARS algorithm:

1. `Numerical_Gradient`: This function computes the numerical derivative (ND) of a given log-density function.

2. `Automatic_differentiation_JAX`: Uses the JAX library to compute the gradient of h(x) automatically. We apply just-in-time compilation (jit) in the function to improve speed.

3. `Tangents_and_Intersections`: Computes tangents (list of tuples of (x, h(x), h'(x)) and intersections(z_js) of the piecewise linear upper hull.

4. `Lower_bound`: Constructs the piecewise linear lower hull for h(x) and evaluates the lower bound at a candidate point (x^*).
5. `Upper_bound`: Constructs the piecewise linear upper hull for h(x) and evaluates the upper bound at a candidate point.

6. `Sample_x_star`: Samples a candidate point x^* from the normalized piecewise exponential distribution formed by the tangents.

   - For each segment, it computes the weight based on the integral of the exponential of the tangent line, normalizes the weights, and chooses a segment proportional to its weight. After a segment is selected, we use Inverse Transform sampling method using the CDF of the selected segment. CDF of segment is given calculated as:

   $$\frac{\exp\{h(x)(x^* - x)\} - \exp\{h(x)(z_1 - x)\}}{\exp\{h(x)(z_2 - x)\} - \exp\{h(x)(z_1 - x)\}}$$

   Then, we set the equation equal to u=unif(0,1) and simply solve for x^*. If the tangent's slope is near zero, we can sample uniformly from the segment's bounds (z_1, z_2).

7. `Plot_distribution`: Make comparison plot of (1) the density and (2) the CDF of the sampled distribution compared to the target distribution. It is designed to save the plot to the `plot` folder and would not show the figure.

1.4 `ARS` moduel (the main function)

The `ARS` moduel contains the main function `ars`, which orchestrates the Adaptive Rejection Sampling algorithm. The function iteratively constructs piecewise linear bounds (upper and lower) for h(x). Candidate points (x^*) are sampled and subjected to squeezing and rejection tests to ensure correctness. The function dynamically refines the set of abscissae to improve sampling efficiency. It returns a list of generated samples and the acceptance rate (i.e.: proportion of proposed samples accepted). See next section for example use and results.

The function has an optional boolean argument `use_ad`, which allows the user to decied whether they want to use automatic differentiation (AD). The default is set to False, which uses numerical differentiation. Since AD in general takes more time, we recommend the user to start from a small sample size before running on a large sample size. Next section also presents a speed comparison of two methods.

II. Results and examples

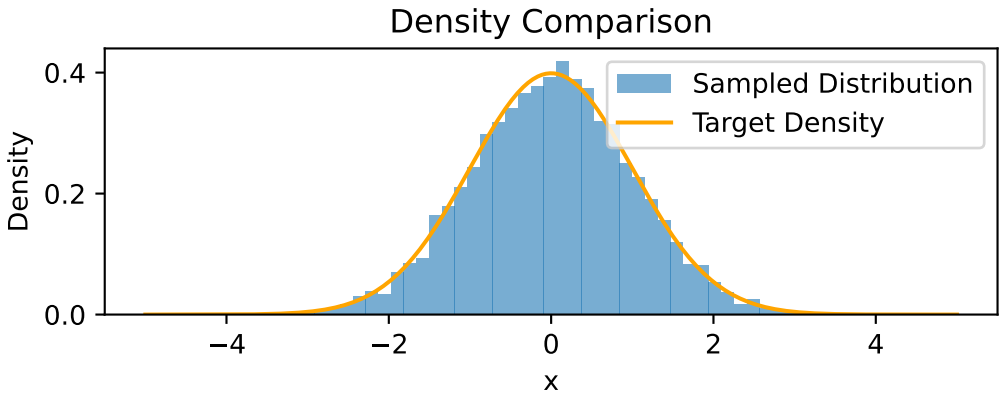
Example use on Gaussian Distribution

```
# Define a normal distribution for example
mu = 0;sigma = 1
def h_normal(x):
    return -0.5 * np.log(2 * np.pi) - np.log(sigma) - (x - mu)**2 / (2 * sigma**2)

# Call ars, default using numerical differentiation
start = time.time()
samples, acceptance_rate = ars(h_normal, domain=(-np.inf, np.inf), n_samples=10000, \
                               initial_points=[-2.0, 0.0, 2.0])
end=time.time()
print(f"Acceptance rate: {acceptance_rate:.3f}")
print(f"Time taken when using numerical differentiation: {end-start:.3f} s")

plot_distribution(samples, h=h_normal, domain=(-5, 5), cdf_func=norm.cdf)
```

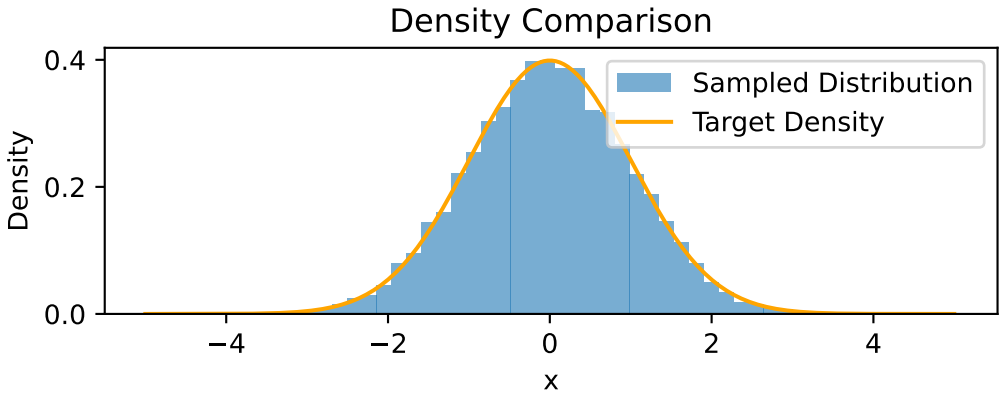
```
Acceptance rate: 1.000
Time taken when using numerical differentiation: 10.050 s
```



```
# ars using automatic differentiation
start = time.time()
samples, acceptance_rate = ars(h_normal, domain=(-np.inf, np.inf), n_samples=10000,\
                               initial_points=[-2.0, 0.0, 2.0],use_ad=True)
end=time.time()
print(f"Acceptance rate: {acceptance_rate:.3f}")
print(f"Time taken when using automatic differentation: {end-start:.3f} s")

plot_distribution(samples, h=h_normal, domain=(-5, 5), cdf_func=norm.cdf)
```

```
Note: Using AD usually is longer, one may consider reduce the number of iterations...
Acceptance rate: 1.000
Time taken when using automatic differentation: 167.120 s
```



III. Tests

3.1 Overview

We apply the following test suite to validate the functionality and performance of our `ars` package. The tests cover the following aspects: (1) Correctness in sampling from different distributions and compliance with acceptance rate expectations; (2) Proper error handling for non-log-concave distributions or incorrect domains. Each test function is documented with a description, explaining the purpose and expected outcomes.

3.2 Log-concave domain of different tested distributions

- Distributions with log-concave domain:

Beta Distribution: only log-concave on (0,1); Gamma Distribution, only log-concave on (0,inf).

t-student Distribution: only log-concave on $(-\sqrt{df},\sqrt{df})$; Cauchy Distribution:only log-concave on (-1,1).

- Not log-concave distributions:

  Quadratic Distribution(x^2); Sinusoidal Distribution

3.3 Test Cases

1. Sampling from different distributions on the log-concave domain using ND and AD

Functions: `test_normal_distribution_ND`,`test_beta_distribution_ND`, `test_gamma_distribution_ND`,

`test_t_student_distribution_with_correct_domain_ND`,`test_cauchy_distribution_ND`;

`test_normal_distribution_AD`, `test_Gamma_distribution_AD`

Expected Behavior: The test should pass if the acceptance rate is above the threshold (0.9 for ND, 0.85 for AD), and outputs the plots correctly.

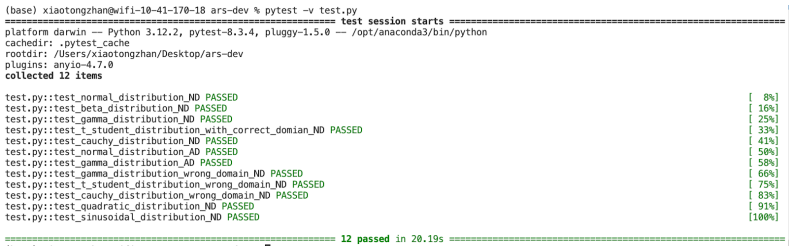
2. Proper error handling for non-log-concave distributions or incorrect domains

Functions:`test_gamma_distribution_wrong_domain_ND`,`test_t_student_distribution_wrong_domain_ND`,

`test_cauchy_distribution_wrong_domain_ND`, `test_quadratic_distribution_ND`,

`test_sinusoidal_distribution_ND`

Expected Behavior: These tests should raise a ValueError indicating the distribution is non-log-concave.



IV. Member Contribution

The ARS package was a collaborative effort, starting with the initial group brainstorming to design the package framework (i.e.: required functions, modules, and tests). **Shana Kim** wrote the foundational code, including `helper_functions.py` and `ars.py`, tested on simple densities, and effectively explained the code logic to the team. She brainstormed the specifics of each function and listed various densities for testing. **Xiaotong Zhan** developed a robust `__init__.py` for module initialization, implemented defensive programming to handle log-concavity errors, and designed comprehensive test files (`test.py`) to validate various density functions, including log-concave, non-log-concave, edge cases, and user errors, ensuring reliable system functionality. **Yihong Zhu** guided the group to collaborate on GitHub and ensured people pull/push/work on branch/open pull-request appropriately without influencing others' work. She added the `AD` functionality to the package and tried to improve the speed of using AD. She conducted code reviews on each stage's code, fixed errors related to precision loss, helped sloved issues found during test stage and revised the code accordingly, and wrote the `setup.py` and `Readme.md`. Yihong wrapped up each member's draft sections to the final report in Quarto. All the members reviewed the final report.