Events Module Documentation

1 Overview

This module provides utilities for working with multivariate probability distributions, cumulative probabilities, and event sampling. It includes functions optimized with Numba for performance, as well as classes for representing and manipulating probability distributions.

2 Functions

2.1 `calculate_cumulative_probs(bins_lower, bins_upper, probs)`

Calculates cumulative probabilities from bin edges and probabilities.

- Parameters:
  - `bins_lower` (np.ndarray): Lower bounds of probability bins
  - `bins_upper` (np.ndarray): Upper bounds of probability bins
  - `probs` (np.ndarray): Probabilities for each bin
- Returns: Tuple of values and cumulative probabilities

2.2 `sample_from_cumulative_probs(values, cum_probs, uniforms)`

Samples from cumulative probabilities using uniform random variables.

- Parameters:
  - `values` (np.ndarray): Values corresponding to cumulative probabilities
  - `cum_probs` (np.ndarray): Cumulative probabilities
  - `uniforms` (np.ndarray): Uniform random variables for sampling
- Returns: Sampled values

2.3 `event_samples(impacts_bins, probs, nb_events, nb_samples)`

Generates event samples based on impact bins and probabilities.

- Parameters:
  - `impacts_bins` (np.ndarray): Bin edges for impact values
  - `probs` (List[np.ndarray]): List of probability arrays
  - `nb_events` (int): Number of events
  - `nb_samples` (int): Number of samples to generate
- Returns: Array of sampled event impacts

2.4 `find(elements, value)`

Performs a binary search to find a value in a sorted array.

- Parameters:
  - `elements` (np.ndarray): Sorted array to search
  - `value`: Value to find
- Returns: Index of the found value or closest lower value

3 Classes

3.1 `MultivariateDistribution` (Protocol)

A protocol class defining the interface for multivariate distributions.
Methods:

- `inv_cumulative_marginal_probs(cum_probs: np.ndarray)`

3.2 `EmpiricalMultivariateDistribution`

Represents an N-dimensional empirical probability density function.

3.2.1 Methods:

- `__init__(bins_lower, bins_upper, probs)`
  - Initializes the distribution with bin edges and probabilities
- `inv_cumulative_marginal_probs(cum_probs)`
  - Calculates inverse cumulative probabilities for marginal distributions

3.3 `CumulativeProb`

A Numba-optimized class for representing cumulative probabilities.

3.3.1 Attributes:

- `values` (np.ndarray): Values corresponding to cumulative probabilities
- `cum_probs` (np.ndarray): Cumulative probabilities

3.3.2 Methods:

- `__init__(values, cum_probs)`
  - Initializes the CumulativeProb object
- `size` (property)
  - Returns the size of the cumulative probability array

4 Numba Optimizations

Several functions and the `CumulativeProb` class are optimized using Numba:

- `sample_from_cumulative_probs` is optimized with `@njit(cache=True)`
- `event_samples_numba` is optimized with `@njit(cache=True)`
- `CumulativeProb` is optimized with `@jitclass`

Numba is being used to optimize performance-critical functions that involve numerical computations and array manipulations. This is particularly important in the field of risk assessment and probability modeling, where you might need to perform many calculations or simulations quickly.

5 Usage Notes

1. The `EmpiricalMultivariateDistribution` class is useful for representing complex, multi-dimensional probability distributions based on empirical data.
2. The `event_samples` function can be used to generate Monte Carlo samples for event-based simulations or risk assessments.
3. The Numba-optimized functions and classes provide high-performance operations for large-scale probability computations and sampling.
4. When working with `EmpiricalMultivariateDistribution`, ensure that the input bin edges and probabilities are correctly formatted and satisfy the required conditions (e.g., non-decreasing, non-overlapping bins).

5. The `find` function provides a custom binary search implementation, which might be useful in specific scenarios where numpy's built-in search functions are not suitable.

6. When using the Numba-optimized functions, be aware of the compilation overhead on the first run, which is amortized in subsequent calls.