Documentation of Probability and statistics

1 Introduction

This library is self-developed to serve the purpose of practice and to have custom functions for statistical tests, as well as specific objects for notable distributions. It is written in **Python 3.10.12** and requires other libraries listed in the **requirements.txt** file. Now, I am going to introduce you to this library, including its classes, instantiation, functions, and their usage. Let's get started.

2 Content of the library

This library contains fourteen classes and a tremendous amount of functions. The classes are:

- RandomVariable (general random variable, later r.v.)
- continuous (general continuous r.v.)
- discrete (general discrete r.v.)
- Exp (exponential distributed r.v.)
- Gamma (gamma distributed r.v.)
- Uni (uniform distributed r.v.)
- Normal (normal distributed r.v.)
- Beta (beta distributed r.v.)
- Geo (geometric distributed r.v.)
- Poisson (Poisson distributed r.v.)
- Binom (binomial distributed r.v.)
- Ind (Bernoulli distributed r.v.)
- Hypergeo (hypergeometric distributed r.v.)
- Negbin (negative binomial distributed r.v.)

2.1 Instantiation of the above classes

Class	Instantiation
RandomVariable	RandomVariable(exp = 'real number', var = 'positive real number')
continuous	continuous(mass_function = 'a lambda function or any other function', lower = 'lower limit of support', upper = 'upper limit of support')
discrete	discrete(values = 'list or NumPy array of real numbers', probabilities = 'list or NumPy array of real numbers between 0 and 1')
Exp	Exp(lam = 'rate, positive real number')
Gamma	Gamma(alpha = 'shape parameter, positive real number', beta = 'rate, positive real number')
Uni	Uni(lower = 'lower limit of support', upper = 'upper limit of support')
Normal	Normal(m = 'mean, real number', sigmasq = 'variance, positive real number')
Beta	Beta(alpha = 'positive real number', beta = 'positive real number')
Geo	Geo(p = 'probability of success, positive real number between 0 and 1')
Poisson	Poisson(lam = 'positive real number')
Binom	Binom(n = 'number of trials, positive integer', p = 'probability of success')
Ind	Ind(p = 'probability of success')
Hypergeo	Hypergeo(N = 'population size, positive integer', K = 'number of marked objects, positive integer', n = 'number of draws, positive integer')
Negbin	Negbin(r = 'number of successes, positive integer', p = 'probability of success')

2.2 Functions of classes

Each class has the following functions:

- **exp()** which returns with the expectation of the r.v.
- **var()** which returns with the variance of the r.v.
- **addition** that you can do with ease just typing the + sign. Apparently this requires two random variables. In the majority of cases, it returns with a simple general **RandomVariable** object, but I have also programmed some specific cases too. For example, the sum of two Poisson distributed r.v.-s is also Poisson distributed, or the sum of two geometric distributed r.v.-s is a negative binomial distributed r.v. and so on.
- **subtraction** which also requires two r.v.-s. (Executed by typing the - sign)
- **multiplication** which multiplies the r.v. by a **non-zero** real number. It is important to note that it multiplies from the right. For instance, if you want to get the three-times version of a random variable, you have to write the following:

```
X = Poisson(1/2)
Y = X*3
```

- **division** which requires a r.v. and a **non-zero** real number. (Executed by typing the / sign)

All specific distributions have the following functions:

- The continuous ones has a function named **prob(a, b)** which returns with the probability of the (a, b) interval.
- The discrete ones has the same but in this case you have to provide a value and it returns with its probability.
- **sample(n)** which returns with a NumPy array of an n-element sample from the given distribution. For example, if you want a 100-element sample from the distribution Exp(5), you have to write the following:

```
X = Exp(5)
sample = X.sample(100)
```

The general discrete, continuous and the general r.v. classes **do not have** this kind of function.

All continuous variables (including the general continuous class) have the following specific function:

- **nth_moment(n)** which returns with the *n*th moment of the r.v.
- We note that the general discrete class also has this function but its descendant classes do not have it. (Actually they do but it returns with 0)

2.3 Functions of samples

The library also contains some functions to work with samples. Here they are:

- **Cssd** which returns with the **C**orrected **S**ample **S**tandard **D**eviation of the sample. (Cssd(sample))
- **Sample_cov** which returns with the empirical covariation of two samples. (Sample_cov(sample1, sample2))
- **Cov** which returns with the empirical covariance of two r.v.-s calculated from 1000-element samples. (Cov(X, Y))
- **Sample_corr** which returns with the empirical correlation of two samples. (Sample_corr(sample1, sample2))
- **Corr** which returns the empirical correlation of two r.v.-s calculated from 1000-element samples. (Corr(X, Y))

3 Statistical tests

This library contains five specific statistical tests: one and two-sample *u*-test, one and two-sample *t*-test and *F*-test. Here are their usage and a little explanation:

- **one_sample_u_test**. Usage: **one_sample_u_test**(sample, sigma, mu0, alpha, alternative).

  Explanation: when using *u*-test, the deviation of the sample is known, in this function it is marked with **sigma**. You also need a mean (the expectation of the distribution) as a null hypothesis, this is marked with **mu0**. Besides these you need a significance level (usually 0.05 or 0.01) marked with **alpha** and an alternative hypothesis which can be **less, greater** or **not_equal** in this case and they always mean that the mean of the null hypothesis is less, greater or not equal to the real one.
  After describing the functioning of this method, other ones are pretty similar, so I am going to mention their usage.
- **two_sample_u_test**(sample1, sample2, sigma1, sigma2, alpha, alternative)
- **one_sample_t_test**(sample, mu0, alpha, alternative)
- **two_sample_t_test**(sample1, sample2, alpha, alternative)
- **F_test**(sample1, sample2, alpha, alternative)

We need to draw attention to the fact that this documentation assumes knowledge of probability theory and statistics. We hope it would be useful for you too and wish a good work with it!