Probly Documentation

Release 0.1.0

Benjamin Wallace

Sep 04, 2022

Documentation

CHAPTER 1

Installation

Probly can be installed using pip from GitHub as follows:

```
pip install git+https://github.com/bencwallace/probly#egg=probly
```

Note: Probly makes use of NumPy, SciPy, and Matplotlib.

CHAPTER 2

Getting started

We begin by importing `probly`.

```
>>> import probly as pr
```

Next, we initialize some pre-packaged random variables.

```
>>> # A Bernoulli random variable with parameter 0.5
>>> X = pr.Ber()
>>> # A Bernoulli random variable independent of X with parameter 0.9
>>> Y = pr.Ber(0.9)
>>> # A uniform random variable on the interval [-10, 10]
>>> Z = pr.Unif(-10, 10)
```

Calling a random variable produces a random sample from its distribution. In order to obtain reproducible results, we pass a seed as an argument to the random variable. Calling the same random variable with the same seed will produce the same result.

```
>>> seed = 99   # An arbitrary but fixed seed
>>> Z(seed)
-4.340731821079555
>>> Z(seed)
-4.340731821079555
```

Note: An entire Probly session can be seeded by using `pr.seed`. This will determine the sequence of outputs produced by sampling a sequence of random variables initialized in a given order with a given sequence of seeds; it is distinct from seeding the random variables themselves.

Random variables can be combined via arithmetical operations.

```
>>> W = (1 + X) * Z / (5 + Y)
>>> # W is a new random object
```

(continues on next page)

(continued from previous page)

```
>>> type(W)
<class 'probly.core.RandomVariable'>
```

The result of such operations is itself a random variable whose distribution may not be know explicitly. We can nevertheless sample from this unknown distribution!

```
>>> W(seed)
-1.4469106070265185
```

We can also compute properties of a random variable, such as its mean.

```
>>> W.mean()
0.023611159797914952
```

CHAPTER 3

Dependence

Note that `W` is *dependent* on `X`, `Y`, and `Z`. This essentially means that the following must output `True`.

```
>>> x = X(seed)
>>> y = Y(seed)
>>> z = Z(seed)
>>> w = W(seed)
>>> w == (1 + x) * z / (5 + y)
True
```

CHAPTER 4

Independent copies

Separate instantiations of a random variable will produce independent copies: for instance, samples from two instantiations of a normal random variable will be independent of one another, even with the same seed.

```
>>> pr.Normal()(seed)
-0.8113001427396095
>>> pr.Normal()(seed)
0.09346601550504334
```

Independent copies of a random variable can also be produced as follows.

```
>>> Wcopy = W.copy()
>>> Wcopy(seed)
2.430468450181704
```

CHAPTER 5

Random matrices

Random NumPy arrays (in particular, random matrices) can be formed from other random variables.

```
>>> M = pr.array([[X, Z], [W, Y]])
>>> type(M)
<class 'probly.core.RandomVariable'>
```

Random arrays can be manipulated like ordinary NumPy arrays.

```
>>> M[0, 0](seed) == X(seed)
True
>>> import numpy as np
>>> S = np.sum(M)
>>> S(seed) == X(seed) + Z(seed) + W(seed) + Y(seed)
True
```

CHAPTER 6

Function application

Any functions can be lifted to a map between random variables using the `@pr.lift` decorator.

```
>>> from numpy.linalg import det
>>> det = pr.lift(det)
```

An equivalent way of doing this is as follows:

```
@pr.lift
def det(m):
        return np.linalg.det(m)
```

The function `det` can now be applied to `M`.

```
>>> D = det(M)
>>> D(seed)
-5.280650914177544
```

CHAPTER 7

Conditioning

Random variables can be conditioned as in the following example:

```
>>> C = W.given(Y == 1, Z > 0)
>>> C(seed)
1.97965814796514
```

Any boolean-valued random variable can be used as a condition.

CHAPTER 8

Random parameters

Random variables can themselves be used to parameterize other random variables, as in the following example:

```
>>> U = pr.Unif()
>>> B = pr.Ber(U)
>>> B(seed)
0
```

CHAPTER 9

Custom models

Custom models can be constructed by applying the `pr.model` decorator, evaluated on a list of parameter names, to a function of these parameters whose return value is a sampler (a function from a random seed to a random sample).

```
>>> @pr.model('a', 'b')
>>> def SquareOfUniform(a, b):
>>>     def sampler(seed):
>>>         np.random.seed()
>>>         return np.random.uniform(a, b) ** 2
>>>     return sampler
```

This makes `SquareOfUniform` into a class whose instances are random variable objects that can be manipulated as above. To construct classes of random variables with additional functionality (e.g. built-in mean, variance, etc.), one can directly subclass `Distribution` as in the example at *Custom distributions*.

CHAPTER 10

Examples

10.1 The central limit theorem

Let X be a Bernoulli random variable.

```
>>> import probly as pr
>>> X = pr.Ber()
```

We are interested in the sum of many independent copies of X. For this example, let's take "many" to be 1000.

```
>>> num_copies = 1000
>>> Z = np.sum(pr.iid(X, num_copies))
```

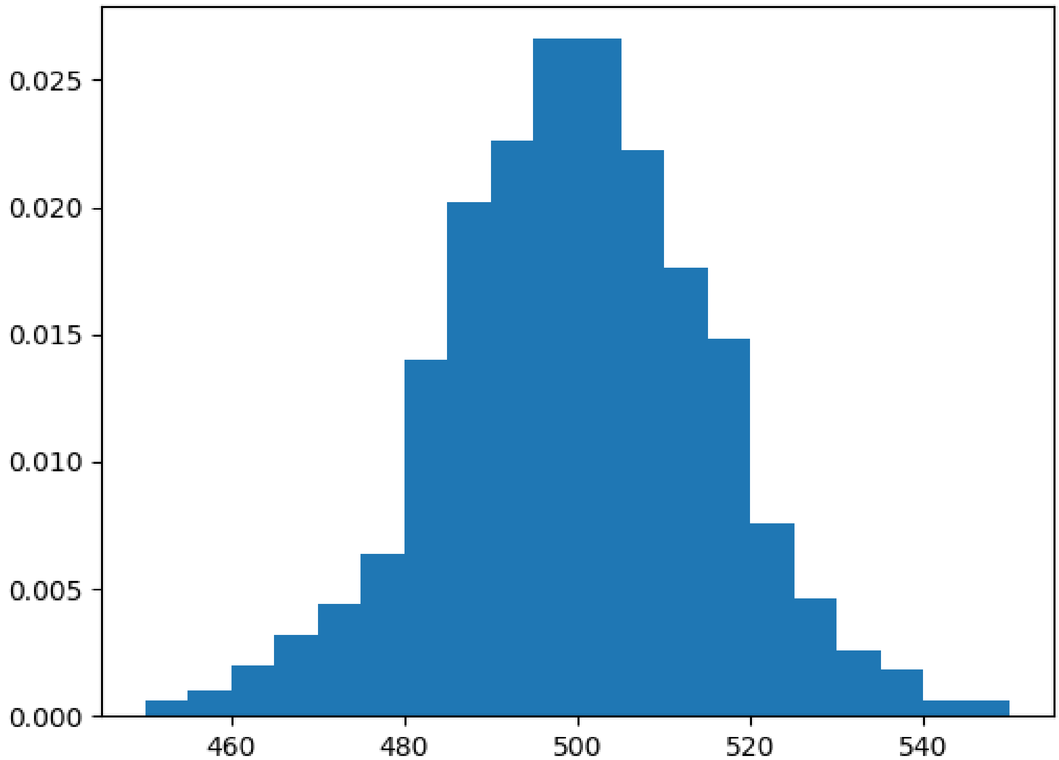
The sum Z is itself a random variable, but its precise distribution, unlike that of X, is unknown.

Nevertheless, the central limit theorem states, roughly, that Z is approximately normally distributed. We can check this empirically by plotting a histogram of the distribution of Z.

The more samples of Z we use to produce the histogram, the better an approximation it will be to the variable's true distribution. But each time we sample Z, we must sample 1000 Bernoulli random variables and sum the results, so computing a histogram from very many samples can take a long time. Below we use 1000 samples, but you may want to reduce this number if running the code takes too long.

```
>>> pr.hist(Z, num_samples=1000)
```

The result resembles the famous bell-shaped curve of the normal distribution.



10.2 The semicircle law

A Wigner random matrix is a random symmetric matrix whose upper-diagonal entries are independent and identically distributed. We can construct a Wigner matrix using `Wigner`. For instance, let's create a 1000-dimensional Wigner matrix with normally distributed entries.

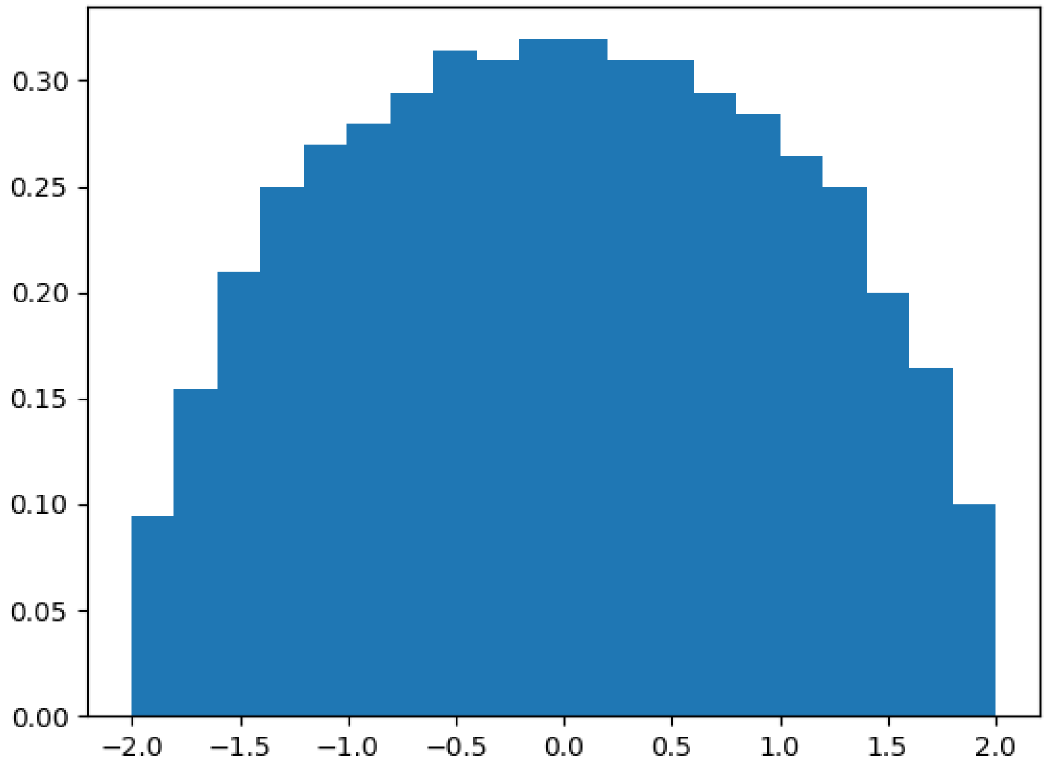
```
>>> import probly as pr
>>> dim = 1000
>>> M = pr.Wigner(dim)
```

The *semicircle law* states that if we normalize this matrix by dividing by the square root of 1000, then the eigenvalues of the resulting (random) matrix should follow the semicircle distribution. Let's check this empirically. First, we normalize `M` and then we construct its (random) eigenvalues by applying NumPy's numpy.linalg.eigvals using `lift()`.

```
>>> from numpy.linalg import eigvals
>>> M = M / np.sqrt(dim)
>>> eigvals = pr.lift(eigvals)
>>> E = eigvals(M)
```

The distribution of the eigenvalues can be visualized using the `hist()` function. Note that we need only take 1 sample.

```
>>> pr.hist(E, num_samples=1)   # doctest: +SKIP
```



10.3 Custom distributions

The following example shows how to create a custom distribution. We'll start by constructing a simple non-random class.

```
>>> class Human:
>>>     def __init__(self, height, weight):
>>>         self.height = height
>>>         self.weight = weight
```

We'd like to create a kind of normal distribution over possible humans. We can do this as follows.

```
>>> import numpy as np
>>> from probly.distr.distributions import Distribution
>>> class NormalHuman(Distribution):
>>>     def __init__(self, female_stats, male_stats):
>>>         self.female_stats = female_stats
>>>         self.male_stats = male_stats
>>>         super().__init__()
>>>     def _sampler(self, seed):
```

(continues on next page)

(continued from previous page)

```
>>>         np.random.seed(seed)
>>>         gender = np.random.choice(2, p=[0.5, 0.5])
>>>         if gender == 0:
>>>             height_mean, weight_mean, cov = self.female_stats
>>>         else:
>>>             height_mean, weight_mean, cov = self.male_stats
>>>         means = [height_mean, weight_mean]
>>>         np.random.seed(seed)
>>>         height, weight = np.random.multivariate_normal(means, cov)
>>>         return Human(gender, height, weight)
```

All the capabilities of random variables, including all those discussed above, will be available to our new random variable objects.

Note: Of course, certain operations may result in errors on sampling. For instance, sampling from the "sum" of two random humans will raise an error unless we overload addition for humans by defining `__add__(self, other)` in the `Human` class.

Let's initialize an instance of this random variable.

```
>>> f_cov = np.array([[80, 5], [5, 99]])
>>> f_stats = [160, 65, f_cov]
>>> m_cov = np.array([[70, 4], [4, 11]])
>>> m_stats = [180, 75, m_cov]
>>> H = NormalHuman(f_stats, m_stats)
```

We can sample from and manipulate such a random variable as usual.

```
>>> @pr.lift
>>> def bmi(human):
>>>     return human.weight / (human.height / 100) ** 2
>>> BMI = bmi(H)
>>> BMI(seed)
23.57076738620301
```

CHAPTER 11

API Reference

11.1 Distributions

`Distribution`

11.1.1 Discrete random variables

`RandInt`
`Multinomial`
`Bin`
`Ber`
`NegBin`
`Geom`
`HyperGeom`
`Pois`

11.1.2 Continuous random variables

`Gamma`
`ChiSquared`
`Exp`
`Unif`
`Normal`
`Beta`
`PowerLaw`
`F`
`StudentT`
`Laplace`

Continued on next page

Table 3 – continued from previous page

`Logistic`
`VonMises`

11.1.3 Random arrays

`Wigner`
`Wishart`

11.2 Utilities

CHAPTER 12

Repository

Probly is open source and available on GitHub.

CHAPTER 13

Indices and tables

- genindex
- search