binomial_distributions

November 28, 2017

1 Binomial Distributions

Use NumPy to create simulations and compute proportions for the following outcomes. The first one is done for you.

```
In [1]: # import numpy
        import numpy as np
```

1.1 numpy.random.binomial(n, p, size=None)

Draw samples from a binomial distribution.

Samples are drawn from a binomial distribution with specified parameters, n trials and p probability of success where n an integer >= 0 and p is in the interval [0,1]. (n may be input as a float, but it is truncated to an integer in use)

Parameters n : int or array_like of ints Parameter of the distribution, >= 0. Floats are also accepted, but they will be truncated to integers.

p : float or array_like of floats Parameter of the distribution, >= 0 and <=1.

size : int or tuple of ints, optional Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. If size is None (default), a single value is returned if n and p are both scalars. Otherwise, np.broadcast(n, p).size samples are drawn.

Returns ndarray or scalar

Drawn samples from the parameterized binomial distribution, where each sample is equal to the number of successes over the n trials.

1.1.1 1. A fair coin flip produces heads

```
In [2]: # simulate 1 million tests of one fair coin flip
        # remember, the output of these tests are the # successes, or # heads
        tests = np.random.binomial(1, 0.5, int(1e6))

        # proportion of tests that produced heads
        (tests == 1).mean()
```

```
Out[2]: 0.50056299999999998
```

1.1.2 2. Five fair coin flips produce exactly one head

```
In [5]: aaa = np.random.binomial(5, 0.5, 25)
        aaa
```

```
Out[5]: array([2, 4, 3, 3, 3, 2, 5, 3, 4, 2, 2, 2, 1, 4, 2, 4, 3, 3, 0, 1, 2, 3, 4,
               3, 3])
```

```
In [6]: (aaa == 1)
```

```
Out[6]: array([False, False, False, False, False, False, False, False, False,
               False, False, False,  True, False, False, False, False, False,
               False,  True, False, False, False, False, False], dtype=bool)
```

```
In [8]: (aaa == 1).mean()
```

```
Out[8]: 0.080000000000000002
```

```
In [9]: # simulate 1 million tests of five fair coin flips
        tests = np.random.binomial(5, 0.5, int(1e6))

        # proportion of tests that produced 1 head
        (tests == 1).mean()
```

```
Out[9]: 0.15592600000000001
```

1.1.3 3. Ten fair coin flips produce exactly four heads

```
In [10]: # simulate 1 million tests of ten fair coin flips
         tests = np.random.binomial(10, 0.5, int(1e6))

         # proportion of tests that produced 4 heads
         (tests == 4).mean()
```

```
Out[10]: 0.204345
```

1.1.4 4. Five biased coin flips with P(H) = 0.8 produce exactly five heads

```
In [11]: bbb = np.random.binomial(5, 0.8, 25)
         bbb
```

```
Out[11]: array([3, 3, 3, 4, 4, 5, 5, 2, 5, 4, 3, 5, 5, 5, 4, 4, 5, 5, 4, 4, 3, 4, 4,
                4, 2])
```

```
In [12]: # simulate 1 million tests of five biased coin flips
         tests = np.random.binomial(5, 0.8, int(1e6))

         # proportion of tests that produced 5 heads
         (tests == 5).mean()
```

```
Out[12]: 0.32768799999999998
```

1.1.5 5. Ten biased coin flips with P(H) = 0.15 produce at least 3 heads

```
In [14]: ccc = np.random.binomial(10, 0.15, 25)
         ccc

Out[14]: array([2, 1, 1, 1, 1, 1, 1, 0, 1, 0, 2, 0, 2, 4, 1, 2, 1, 2, 1, 0, 0, 0, 0,
                0, 2])

In [18]: (ccc >= 3).mean()

Out[18]: 0.040000000000000001

In [19]: # simulate 1 million tests of ten biased coin flips
         tests = np.random.binomial(10, 0.15, int(1e6))

         # proportion of tests that produced at least 3 heads
         (tests >= 3).mean()

Out[19]: 0.17948600000000001
```