Notebook

February 2, 2019

Local date & time is : 02/02/2019 11:24:50 PST

```
In [4]: def tvd(b, g):
            return sum(abs(b - g)) / 2
```

```
In [6]: tvd(b, g)
```

```
Out[6]: 0.15
```

1 newpage

In [7]:
```
def prob_matches(k, n):
    x_even = np.arange(0, n - k + 1, 2)
    x_odd = np.arange(1, n - k + 1, 2)
    eventerms = 1 / (special.factorial(x_even))
    oddterms = 1 / (special.factorial(x_odd))
    return (1 / special.factorial(k)) * (sum(eventerms) - sum(oddterms))
```

In [8]:
```
prob_matches(99, 100)
```

Out[8]:
```
0.0
```

$$P(M_n = k) \;=\; \frac{1}{k!} \cdot (1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots (-1)^{n-k} \frac{1}{(n-k)!})$$

Looking at the formula, for$M_n = 0, 1$, $\frac{1}{k!}$both equal to 1. And the latter terms of the equation for$M_n = 0$differ only by$(-1)^n \frac{1}{(n)!}$from the equation for$M_n = 1$. For a large n, such term is very small. Hence$P(M_n = 0)$is very close to$P(M_n = 1)$when$n$is large.

In [9]:
```
prob_matches(0, 100), prob_matches(1, 100)
```

Out[9]:
```
(0.36787944117144233, 0.36787944117144233)
```

In [10]:
```
def match_dist(n):
    k = np.arange(n + 1)
    prob = make_array()
    for i in k:
        prob = np.append(prob, prob_matches(i, n))
    return prob
```

In [11]:
```
sum(match_dist(10))
```

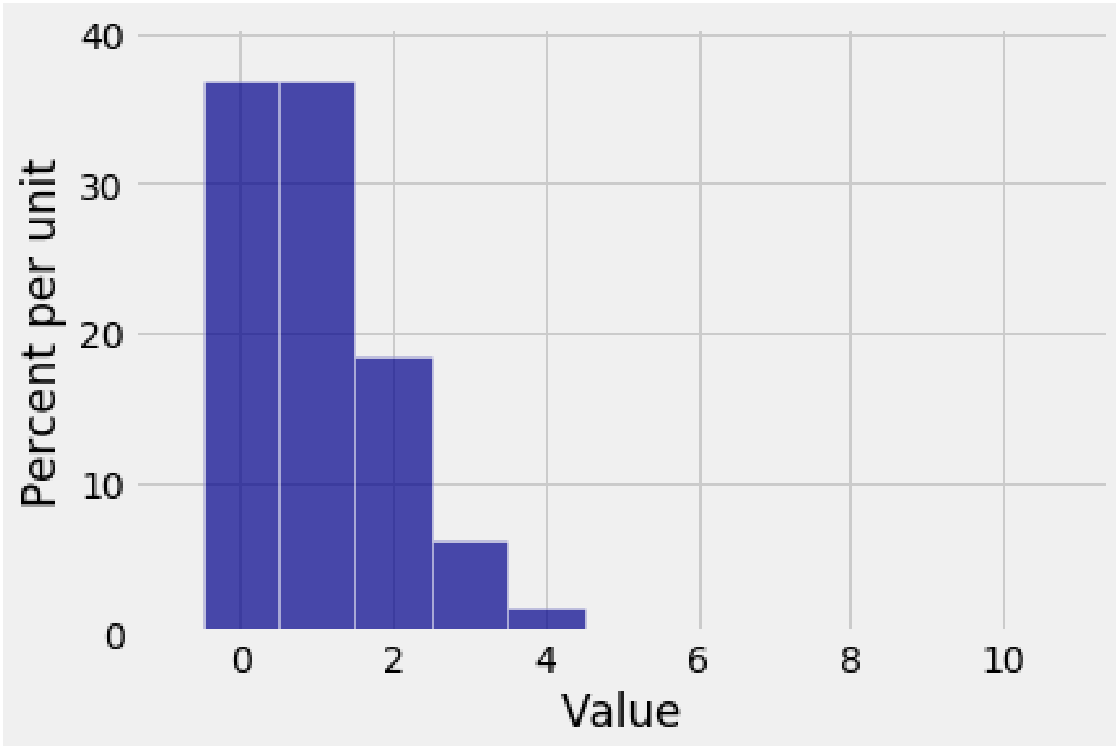
Out[11]:
```
1.0000000000000002
```

In [12]:
```
sum(match_dist(100)[0:11])
```

Out[12]:
```
0.9999999899522336
```

The possible values of$M_n$is from 0 to 100. With the sum of probabilties for$M_n =0$*to*10*equalto*1, *itsaysthatthedistributionof*M_n$lies in between 0 and 10.

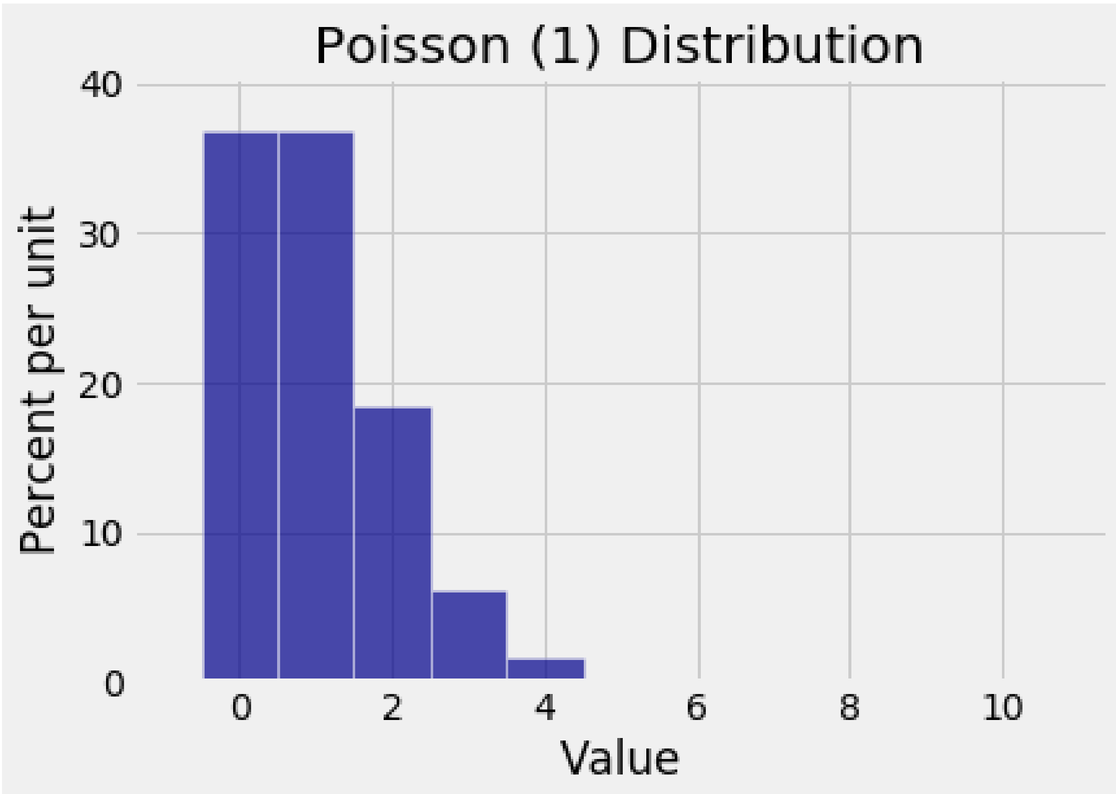
In [13]:
```
k = np.arange(11)

prob = match_dist(10)

matches_100 = Table().values(k).probabilities(prob)
Plot(matches_100)
```



40
30
20
10
0
Percent per unit
0
2
4
6
8
10
Value

2 newpage

```
In [14]: k = np.arange(11)          # selected possible values
         poisson_1_probs = stats.poisson.pmf(k, 1)     # array of corresponding Poisson (1) probabilitie
         poisson_1_dist = Table().values(k).probabilities(poisson_1_probs)
         Plot(poisson_1_dist)
         plt.title('Poisson (1) Distribution');
```




```
In [15]: def matches_Poisson_tvd(n):
             mn = match_dist(n)

             k = np.arange(n+1)
             poisson = stats.poisson.pmf(k, 1)

             return tvd(mn, poisson)
```

I think `matches_Poisson_tvd(5)` should be larger. The poisson approximation uses$e^{-1}$is a better approximation for the actual match distribution with higher number of n. Hence the distance away from the actual values will be further away for n = 5 compared to n = 100

```
In [17]: matches_Poisson_tvd(5), matches_Poisson_tvd(100)

Out[17]: (0.03411123088143108, 2.0236745963801594e-17)

In [18]: #matches_Poisson_tvd caculates half of the sum of all the absolute difference
         #between actual distribution and Poisson approximation,
```

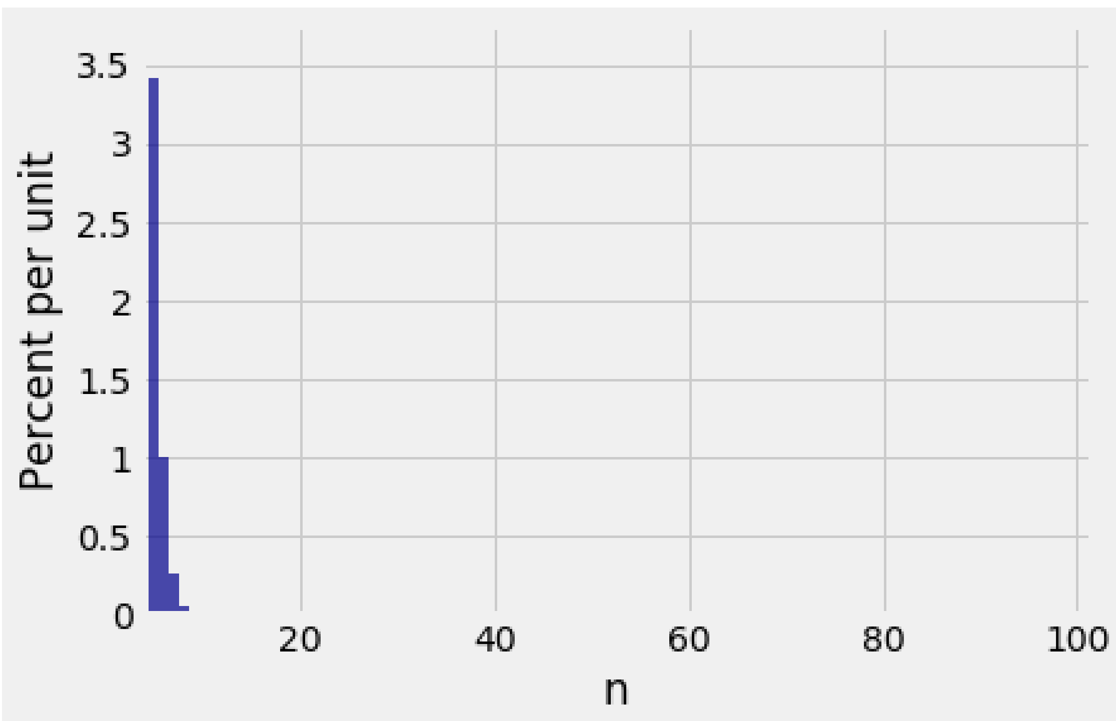
```
#Hence, assuming all the differences are positive,
#the largest possible error is two times the tvd.

matches_Poisson_tvd(100) * 2
```

Out[18]: 4.047349192760319e-17

In [19]:
```
tvd_table = Table().with_column('n', np.arange(5, 101))

matches_tvds = tvd_table.with_column('Matches (n)', tvd_table.apply(matches_Poisson_tvd, 0))

tvd_table = matches_tvds

tvd_table
```

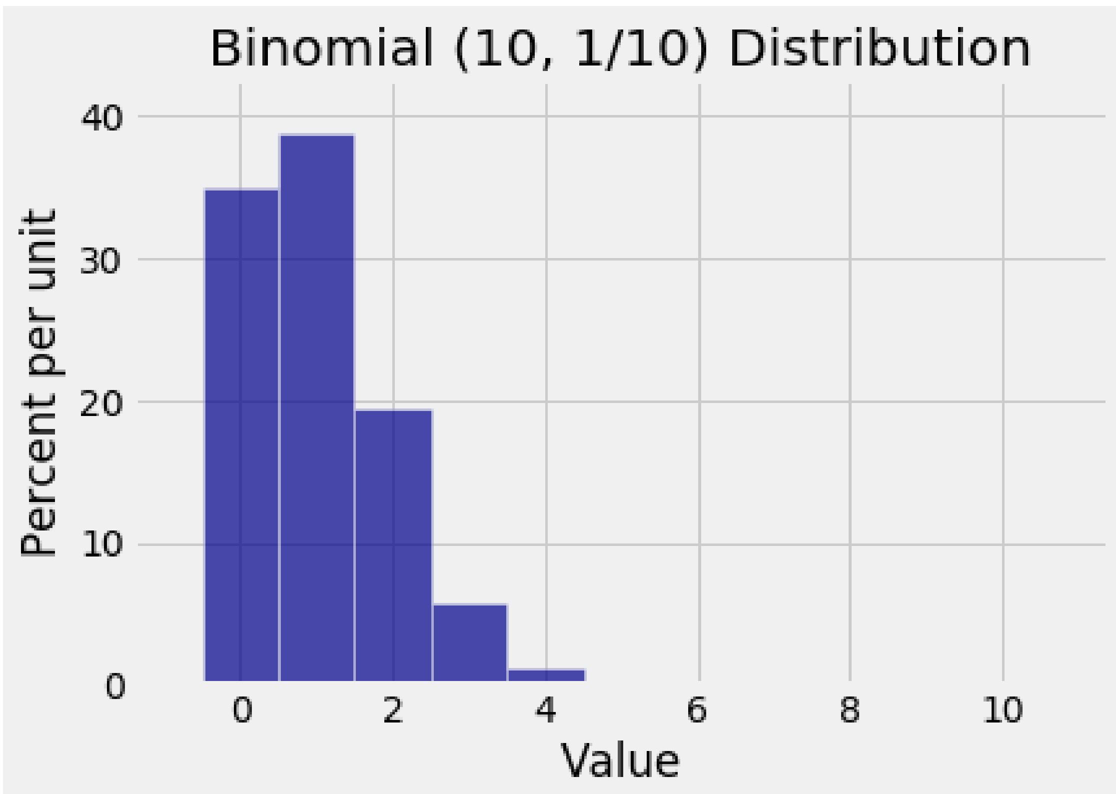
Out[19]:
```
n    | Matches (n)
5    | 0.0341112
6    | 0.0100777
... Omitting 5 lines ...
13   | 8.28377e-08
14   | 1.11286e-08
... (86 rows omitted)
```

In [20]:
```
Plot(tvd_table)
```



3 newpage

```
In [21]: # binomial (n, 1/n) distribution

         n = 10                             # number of trials

         k = np.arange(11)                            # possible values
         binom_probs = stats.binom.pmf(k, n, 1/n)                  # binomial (n, 1/n) probabilities


         binom_dist = Table().values(k).probabilities(binom_probs)

         Plot(binom_dist)
         plt.title('Binomial (10, 1/10) Distribution');
```



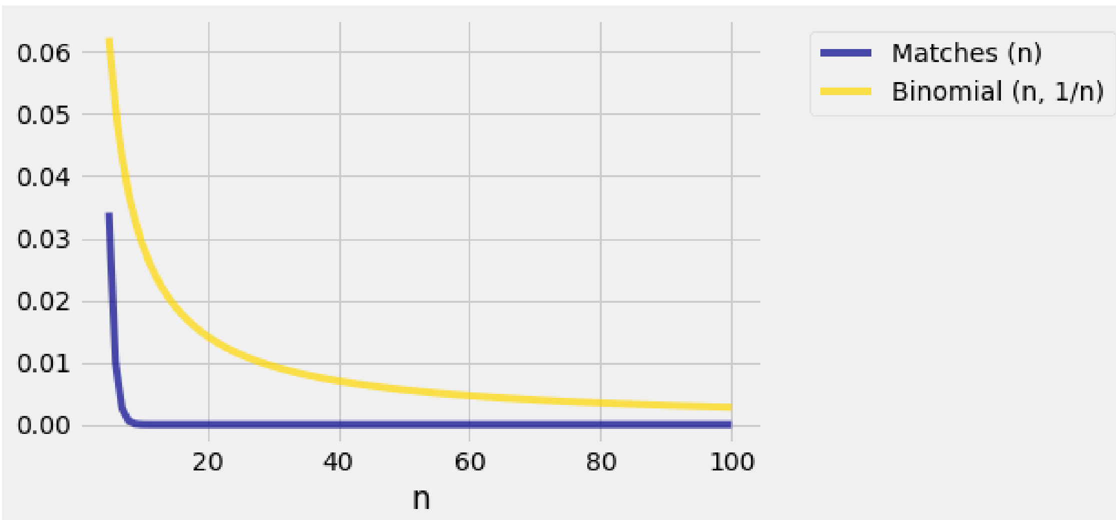

```
In [23]: def binomial_Poisson_tvd(n):
             k = np.arange(n+1)
             binom = stats.binom.pmf(k, n, 1 / n)
             poisson = stats.poisson.pmf(k, 1)
             return tvd(binom, poisson)
```

```
In [24]: binomial_Poisson_tvd(30)

Out[24]: 0.009379738441886479
```

```
In [25]: binom_tvds = tvd_table.apply(binomial_Poisson_tvd, 0)

         tvd_table = tvd_table.with_column('Binomial (n, 1/n)', binom_tvds)

         tvd_table
```

```
Out[25]: n    | Matches (n) | Binomial (n, 1/n)
         5    | 0.0341112   | 0.0622837
         6    | 0.0100777   | 0.0509556
         ... Omitting 5 lines ...
         13   | 8.28377e-08 | 0.0222259
         14   | 1.11286e-08 | 0.0205686
         ... (86 rows omitted)
```

```
In [26]: tvd_table.plot(0)
```



For values of n that are about 60 or more, Poisson(1) approximations to binomial$(n, 1/n)$ probabilities will be off by at most 0.5%.

```
In [27]: Table().with_columns('n', tvd_table[0], 'diff', abs(tvd_table[1] - tvd_table[2])).show()
```

```
<IPython.core.display.HTML object>
```

```
In [ ]:
```