Balls&bins–simulations

November 27, 2023

1 Maxload in Balls&bins

Here we have the standard balls&bins model, simulated in a straightforward way. We plot histogram of the maxload.

[1]:
```
import matplotlib.pyplot as plt
import random
import numpy as np
from scipy.stats import poisson
```

[2]:
```
%%time
def simulate_max_balls_in_bin(n, num_simulations=1000):
    max_balls = []

    for _ in range(num_simulations):
        bins = [0] * n
        for _ in range(n):
            bins[random.randint(0, n - 1)] += 1
        max_balls.append(max(bins))

    return max_balls
# Example usage
n = 1_000  # Number of balls and bins
num_simulations = 1_000  # Number of times to run the simulation
maxload1 = simulate_max_balls_in_bin(n, num_simulations)
print(f"Average maximum number of balls in a bin (for n = {n}): {np.
 ↪mean(maxload1)}")
```

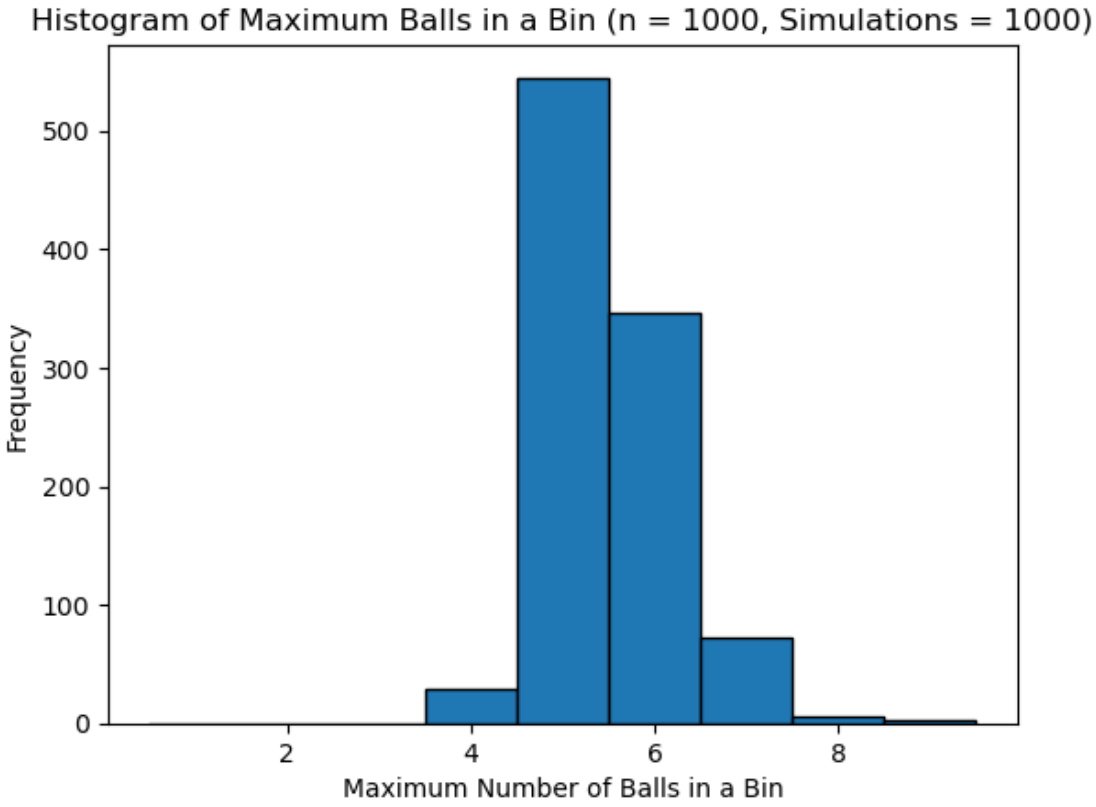
```
Average maximum number of balls in a bin (for n = 1000): 5.488
CPU times: user 662 ms, sys: 0 ns, total: 662 ms
Wall time: 665 ms
```

[3]:
```
np.log(n)/np.log(np.log(n))
```

[3]:
```
3.574249916581999
```

[4]:
```
plt.hist(maxload1, bins=range(1, max(maxload1) + 2), align='left',␣
 ↪edgecolor='black')
```

```
plt.xlabel('Maximum Number of Balls in a Bin')
plt.ylabel('Frequency')
plt.title(f'Histogram of Maximum Balls in a Bin (n = {n}, Simulations =␣
 ↪{num_simulations})')
plt.show()
```



2 Poisson simulation

Here we simulate the same proces indirectly: using independent Poisson random variables, $Y_1, \dots, Y_n$, using the notation from the class.

Then we condition on $\sum_i Y_i = n$. By theorem from class, the distribution of $\vec{Y} | \sum_i Y_i = n$ is the same, as in the exact model, that is the numbers $X_1, \dots, X_n$ (list `bins` above). Thus, also the distribution of maxload (maximum of the Y_is) is the same as the distribution of $\max_i X_i$.

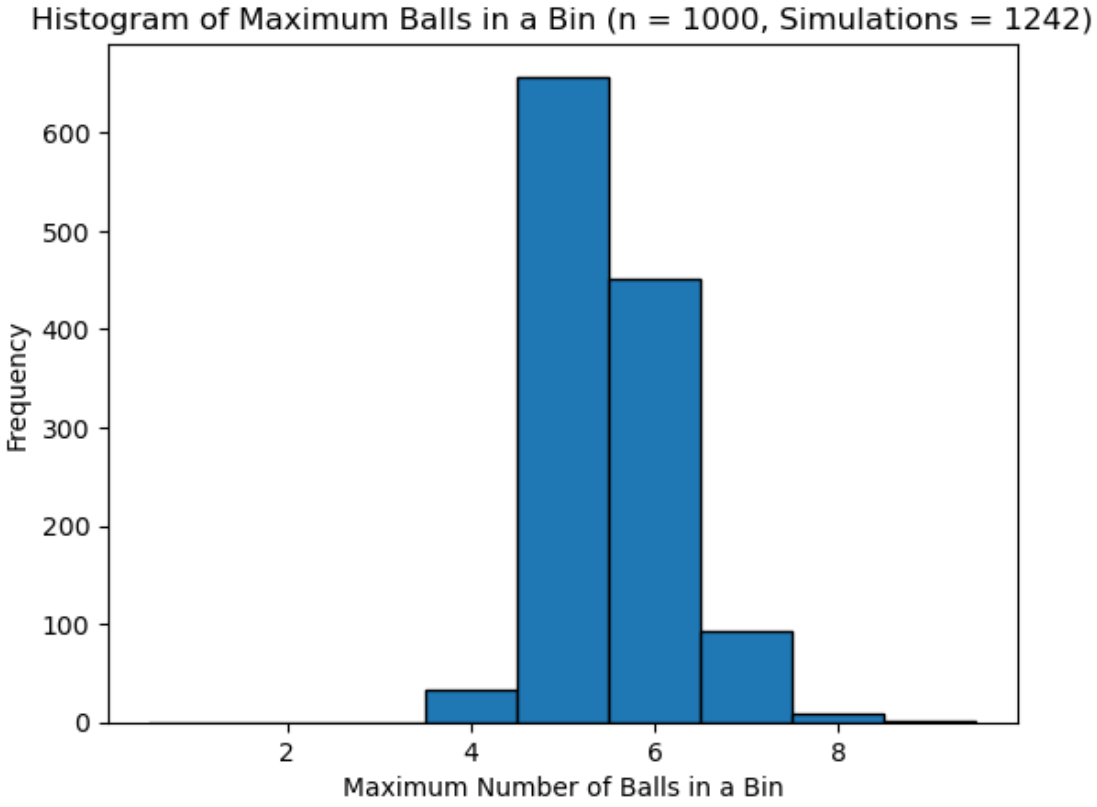
[11]:
```
%%time
pois2 = np.random.poisson(lam=1,size=n*num_simulations*100)
y2 = np.resize(pois2,[n,num_simulations*100])
m2 = np.where(np.sum(y2,0)==n, np.max(y2,0), 0)
```

```
CPU times: user 4.1 s, sys: 1.16 s, total: 5.27 s
```

```
Wall time: 5.26 s
```

[12]:
```
maxload2 = m2[m2!=0]
```

[13]:
```
plt.hist(maxload2, bins=range(1, np.max(maxload2) + 2), align='left',␣
↪edgecolor='black')
plt.xlabel('Maximum Number of Balls in a Bin')
plt.ylabel('Frequency')
plt.title(f'Histogram of Maximum Balls in a Bin (n = {n}, Simulations = {np.
↪size(maxload2)})')
plt.show()
```



The restriction to the simulations with the proper sum means, we only keep those, where $\sum_i Y_i = n$, which happens with probability $P(Pois(1000) = 1000) \doteq 0.012$. For simplicity we keep all simulations, so we have a bit more that 1000.

[14]:
```
np.size(maxload2)/(num_simulations*100)
```

[14]:
```
0.01242
```

[15]:
```
poisson.pmf(mu=1000,k=1000)
```

[15]: 0.01261461134870819

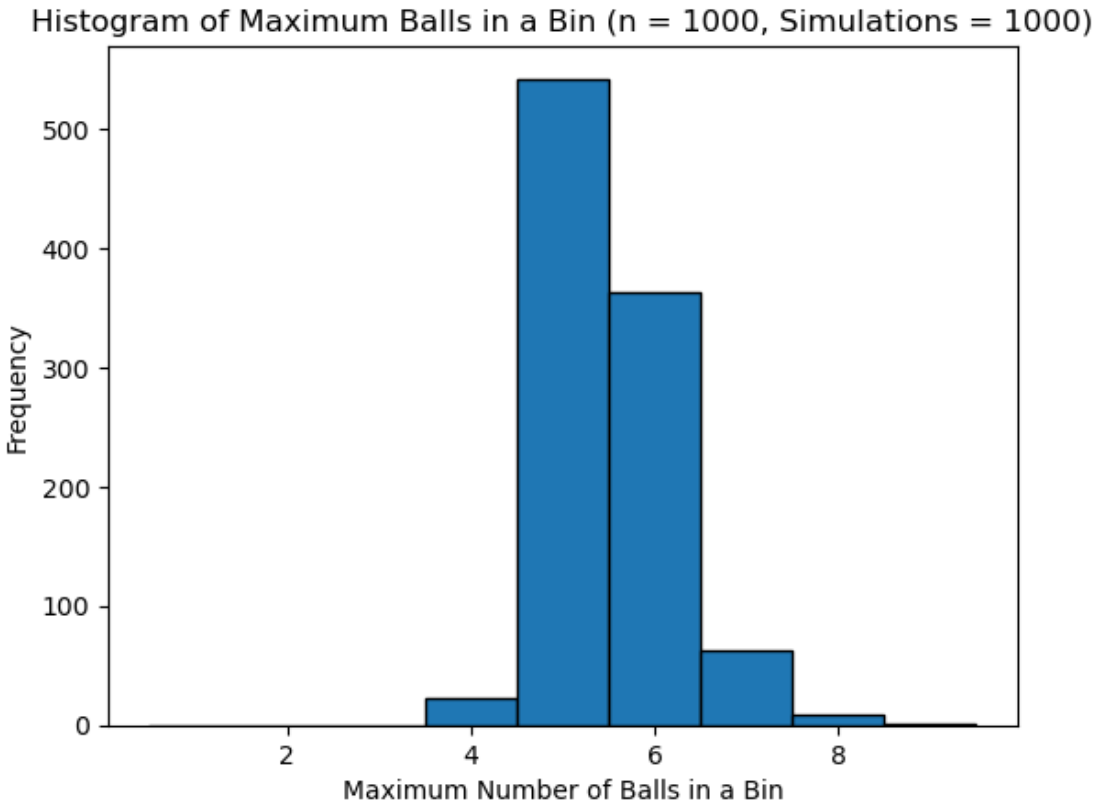
3 Poisson approximation

Finally, we do the real approximation: we ignore the conditioning on the proper sum and analyze just the independent Poisson random variables. In the theoretical approach this means easier calculations. In the practical approach a faster time to generate, as we don't throw away the cases with "incorrect number of balls".

[16]:
```
%%time
pois3 = np.random.poisson(lam=1,size=n*num_simulations)
y3 = np.resize(pois3,[n,num_simulations])
maxload3 = np.max(y3,0)
```

```
CPU times: user 63.3 ms, sys: 4.53 ms, total: 67.8 ms
Wall time: 66.2 ms
```

[17]:
```
plt.hist(maxload3, bins=range(1, np.max(maxload3) + 2), align='left',␣
 ↪edgecolor='black')
plt.xlabel('Maximum Number of Balls in a Bin')
plt.ylabel('Frequency')
plt.title(f'Histogram of Maximum Balls in a Bin (n = {n}, Simulations =␣
 ↪{num_simulations})')
plt.show()
```


Histogram of Maximum Balls in a Bin (n = 1000, Simulations = 1000)
Frequency
0
100
200
300
400
500
2
4
6
8
Maximum Number of Balls in a Bin