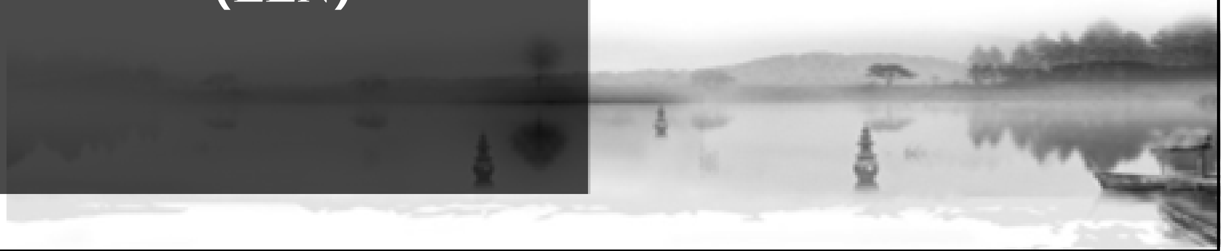
# 大数定理
# the law of large numbers (LLN)

*Zhang Yinsheng (PhD)*
*oo@zju.edu.cn*

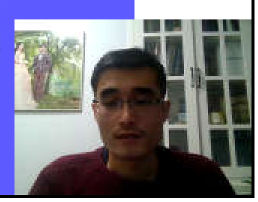


## 大数定律的客观背景

大量随机试验中 { 事件发生的频率稳定于某一常数; 测量值的算术平均值具有稳定性 }

大量抛掷硬币
正面出现频率

字母使用频率



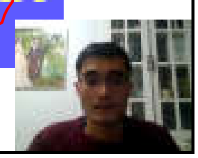
## 案例：历史上一些概率统计学家的试验

| 试验者 | $n$ | $n_H$ | $f_n(H)$ |
|---|---|---|---|
| 德·摩根 | 2048 | 1061 | 0.5181 |
| 蒲 丰 | 4040 | 2048 | 0.5069 |
| $K$·皮尔逊 | 12000 | 6019 | 0.5016 |
| $K$·皮尔逊 | 24000 | 12012 | 0.5005 |

$$f_n(H) \xrightarrow{n\text{的增大}} 1/2$$

## 案例：密码学的一个故事

YFTODXQHFTDPTOGHFQPBQWAQJJTODXQ
HFOQPWTBDHHIXQVAPBFZQHCFWPFHPBFI
PBQWKFABVYYDZBOTHPBQPQJTQOTOGHF
QAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFF
ACFCCFHQWAUVWFLQHGFXVAFXQHFUFHI
LTTAVWAFFAWTEVOITDHFHFQAITIXPFHXA
FQHEFZQWGFLVWPTOFFA



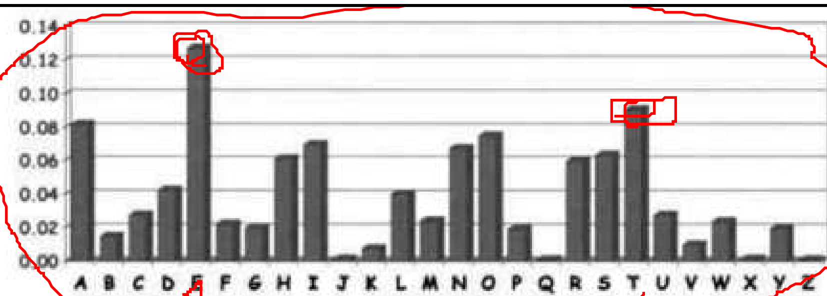



图 2-2 英文字母的频率统计

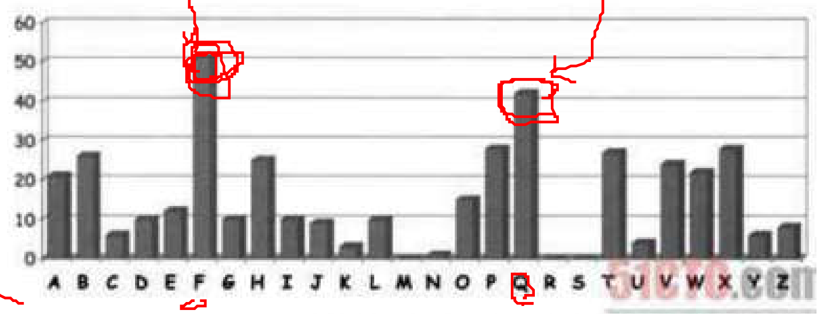
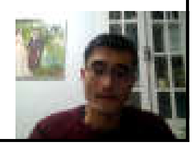



图 2-3 密文字母的频率统计



# Monte Carlo方法

The justification for a Monte Carlo method lies in the law of large numbers.

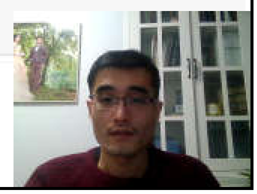
$\pi$

$\frac{\pi}{4}$

In [3]:

```python
# Monte-Carlo: PI

import numpy as np

N    = 1000000
pts = np.random.uniform(-1,1,(N,2))

# Select the points according to your condition
idx = (pts**2).sum(axis=1)  <= 1.0
print("frequency = {}/{} = {}".format(idx.sum(), N, idx.sum()/N))
print("estimated PI = {}".format(idx.sum()/N*4))
```

```
frequency = 785892/1000000 = 0.785892
estimated PI = 3.143568
```



- **双向击鼓传花**
- A、B、C、D、E五个人围成圆圈进行传球游戏，规定每人只能传给相邻的人（向左传或向右传）。
  由A开始游戏。
- 问：传球10次后，球回到A手中的概率是多少？
  请使用Monte Carlo方法进行计算，并与经典概率计算法比较

经典概率解法：

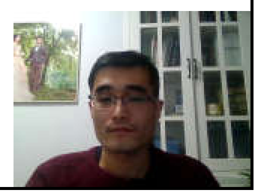
$$P = \frac{2+C_{10}^{5}}{2^{10}} = 24.8\%$$

两者值（近似）相同。仿真次数越多，值越相近。

In [11]:
```
def rollingGame(num_rounds, num_players, num_ops):
    '''模拟实验函数。
    [球回到A手中的实验次数, 球回到A手中的实验编号数组] = RollingGame(实验次数, 玩家数目, 每次实验传球
    '''
    L=0
    history = []

    for iter in range(num_rounds):
        position = 0
        for op in range(num_ops):
            position = (position + random.choice([-1, +1]) + num_players) % num_players

        history.append(position)

        if(position == 0):
            L += 1
    return L/num_rounds, history
```

In [12]:
```
# 调用模拟实验函数。模拟10万次。
num_players = 10
num_ops = 10
N = 100000

P, history = rollingGame(N, num_players, num_ops)
print('仿真结果:', P)
```

仿真结果: 0.24887



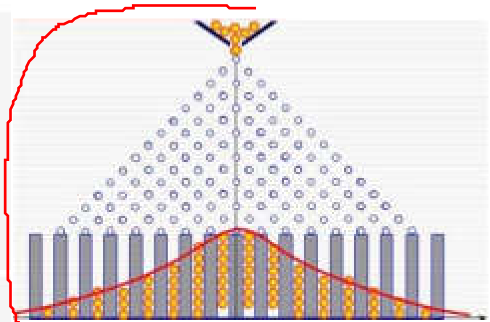
- **高尔顿钉板实验**
- 钉板上有20层钉板、21个落槽，请使用Monte Carlo算法求解球落入各个槽的概率(模拟10万次)。

## 高尔顿钉板实验

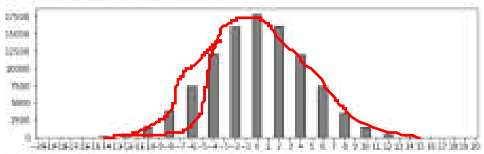
```
def galtonBoard(num_rounds, num_layers):
    '''
    '''
    history = []
    tracks = []
    for iter in range(num_rounds):
        position = 0 # 初始位置
        track = []
        for layer in range(num_layers):
            position = position + random.choice([-1, +1]) # -1 向左摆, +1 向右摆
            track.append(position)

        tracks.append(track)
        history.append(position)

    return history, tracks
```

```
num_rounds = 100000
num_layers = 20
hist, _ = galtonBoard(num_rounds, num_layers)
```

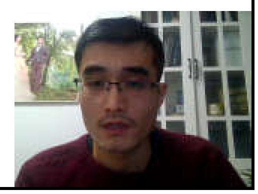
```
import collections
c = collections.Counter(hist)
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
plt.figure(figsize = [17,7])
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.bar(c.keys(), c.values(), color = 'gray', edgecolor='black', linewidth=1.2)
# plt.plot(list(c.keys()), list(c.values()))
```



b(n,p)



$P(i) = C_L^i/2^L$



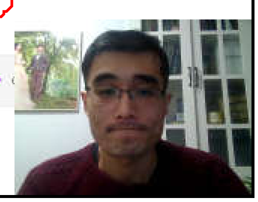
## 三个骰子点数

投掷3个色子。得分表如下：
计算各情况的概率

| Result | Score |
|---|---|
| 456 | 16 |
| 123 | 16 |
| 三个相同 | 8 |
| 两个相同 | 2 |
| 各不相同 | 0 |

理论值：

```
In [31]:
dict_cnt = {}
dict_cnt['ooo'] = 6*(1/6)**3
dict_cnt['123'] = 6*(1/6)**3
dict_cnt['456'] = 6*(1/6)**3
dict_cnt['xyz'] = 6*5*4/(6**3)-dict_cnt['123']-dict_cnt['456']
dict_cnt['oox'] = 6*5*3/(6**3)

dict_cnt

Out[31]:
{'ooo': 0.027777777777777776,
 '123': 0.027777777777777776,
 '456': 0.027777777777777776,
 'xyz': 0.5,
 'oox': 0.41666666666666667}

In [32]:
dict_cnt['ooo'] + dict_cnt['xyz'] + (

Out[32]: 1.0
```



## 验证 Pearson's Chi-Square Goodness-of-Fit Test

皮尔逊卡方拟合优度检验有一个基本的卡方分布假设，即

$$\chi^2 = \sum_{i=1}^{k} \frac{(f_i - np_i)^2}{np_i} \sim \chi^2(k-1)$$

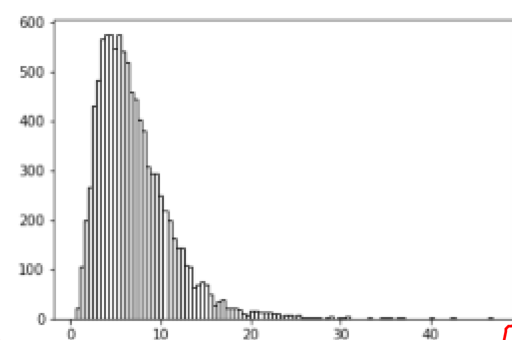
当 $n$ 充分大($n \geq 50$)时，$\chi^2$ 近似服从 $\chi^2(k-1)$ 分布.

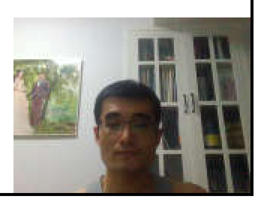
### 1. 下面用高尔顿钉板实验 b(n,p) 来验证

In [56]:

```
# test with b(n,p)

import collections
from scipy.stats import binom
from tqdm import tqdm

chisqs = []
num_rounds = 100 # 公式中n
n = 8 # num_layers, 公式中 k
p = 0.5

for i in tqdm(range(10000)): # MC试验次数

    hist, _ = galtonBoard(num_rounds, n) # rounds, layers
    c = collections.Counter(hist)

    chisq = 0

    for j in range(n):
        pj = binom.pmf(j,n,p)
        npj = num_rounds * pj
        fj = c[j]
        # print(pj, npj, fj)

        chisq = chisq + (fj - npj)**2 / npj

    chisqs.append(chisq)

plt.hist(chisqs, density=False, bins=100, facecolor="none", edgecolor = "black")
plt.show()
```

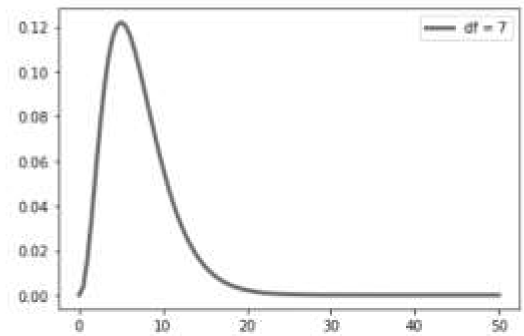
100%|████████████████████| 10000/100
s]



In [57]:

```
from scipy.stats import chi2
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,50,100)
k = n -1
plt.plot(x, chi2.pdf(x, df = k), lw=3, alpha=0.6, label='df = ' + str(k), c = "black")

plt.legend()
plt.show()
```

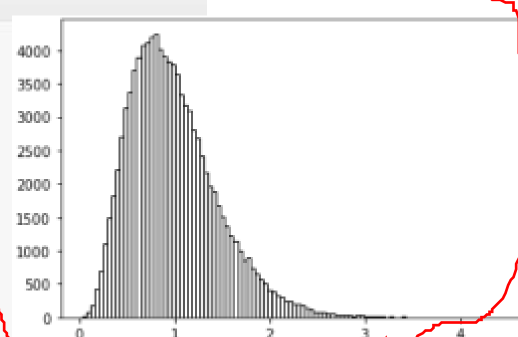


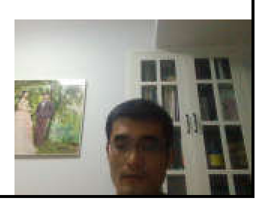

## 验证 ANOVA的F分布假设

F = MSTR/MSE ~ F(k-1, n-k)

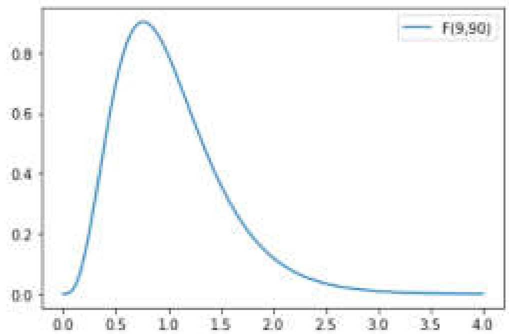
```
import collections
from scipy.stats import binom
from tqdm import tqdm
import numpy as np

FS = []
n = 100 # each class has n samples. Total sample count is kn
k = 10

for i in tqdm(range(100000)): # MC试验次数

    X = np.random.normal (0, 1, size=(n,k))
    SSTR = n*((X.mean(axis = 0)-X.mean())**2).sum()
    MSTR = SSTR/(k-1)
    SSE = ((X - X.mean())**2).sum()
    MSE = SSE/(k*n-k) # 此处k*n为公式中n, 样本总量

    F = 1.0*MSTR/MSE
    FS.append(F)

plt.hist(FS, density=False, bins=100, facecolor="none", edgecolor = "black")
plt.show()
```

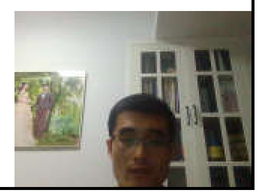
```
100%|████████████████████████████████████████| 100000
s]
```



```python
# F distribution

import numpy
import scipy.stats
import matplotlib.pyplot as plt
x=numpy.linspace(0,4,100)


plt.plot(x,scipy.stats.f.pdf(x,dfn=k-1,dfd=n-k))
plt.legend(['F(' + str(k-1) + ',' + str(n-k) + ')'])
plt.show()
```



# 6. 本福特定律

- Benford's Law.ipynb

## Benford's law

Also called the Newcomb–Benford law, the law of anomalous numbers, or the first-digit law, is an observation about the frequency distribution of leading digits in many real-life sets of numerical data. The law states that in many naturally occurring collections of numbers, the leading significant digit is likely to be small.

十进制中，首位数字出现的概率为：

d 1 2 3 4 5 6 7 8 9
p 30.1% 17.6% 12.5% 9.7% 7.9% 6.7% 5.8% 5.1% 4.6%

# 6. 本福特定律

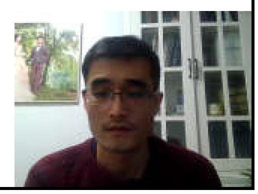
• Benford's Law.ipynb

## Benford's law

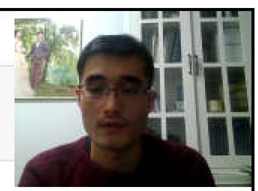
Also called the Newcomb–Benford law, the law of anomalous numbers, or the first-digit law, is an observation about the frequency distribution of leading digits in many real-life sets of numerical data. The law states that in many naturally occurring collections of numbers, the leading significant digit is likely to be small.

十进制中，首位数字出现的概率为：

d 1 2 3 4 5 6 7 8 9
p 30.1% 17.6% 12.5% 9.7% 7.9% 6.7% 5.8% 5.1% 4.6%



## Case Study 2: Stock Price (AAPL)



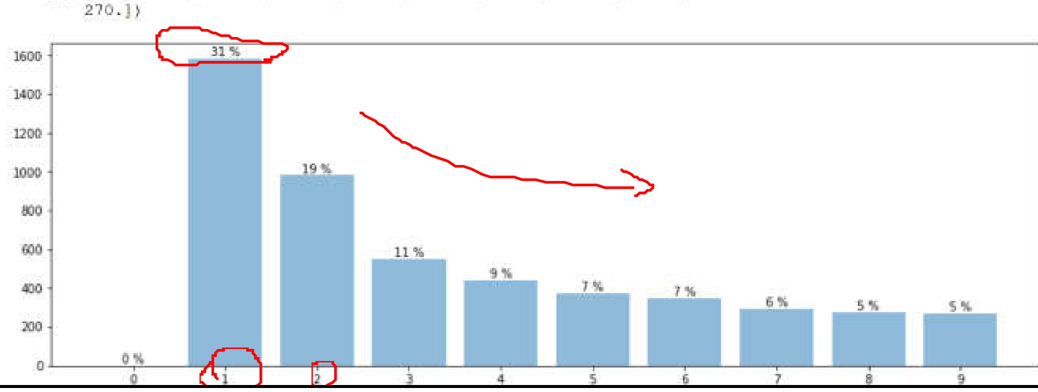
In [50]:
```
import yfinance as yf

data = yf.download('AAPL','2000-01-01','2020-05-01')
data.to_csv('AAPL.csv')
```

```
[*********************100%***********************]  1 of 1 completed
```

In [51]:
```
data = pd.read_csv('AAPL.csv')
volumes = data['Volume'].values
```

In [52]:
```
Analyze(volumes)
```

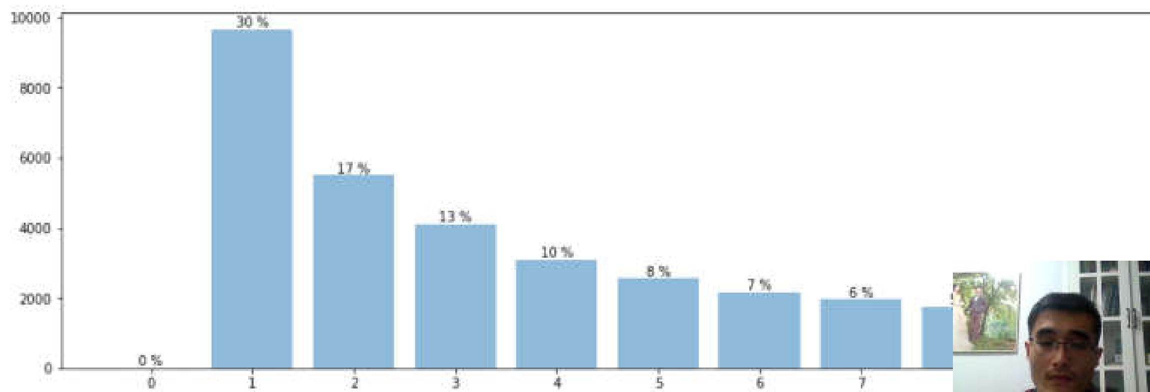
Out[52]:
```
array([   0., 1585.,  982.,  548.,  441.,  375.,  347.,  292.,  275.,
        270.])
```



## Case Study 4: Annual Trade for Countries

In [14]:
```python
import pandas as pd

data = pd.read_csv('MBSComtrade.csv')
Analyze(data['value'].values)
```

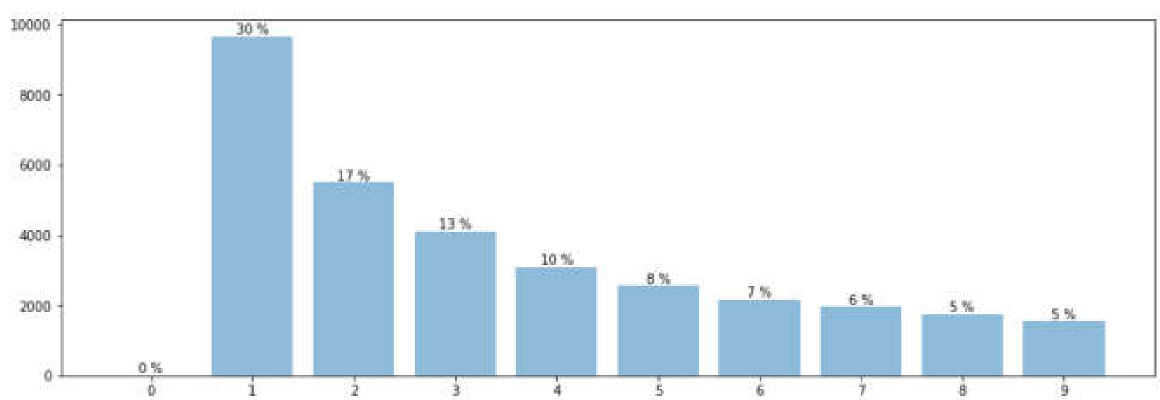
Out[14]:
```
array([   0., 9656., 5512., 4112., 3106., 2552., 2158., 1967., 1745.,
       1544.])
```




## Case Study 4: Annual Trade for Countries

In [14]:
```python
import pandas as pd

data = pd.read_csv('MBSComtrade.csv')
Analyze(data['value'].values)
```

Out[14]:
```
array([   0., 9656., 5512., 4112., 3106., 2552., 2158., 1967., 1745.,
       1544.])
```



# Benford's Law and COVID-19 reporting ☆

Christoffer Koch [a] ... Ken Okamura [b]

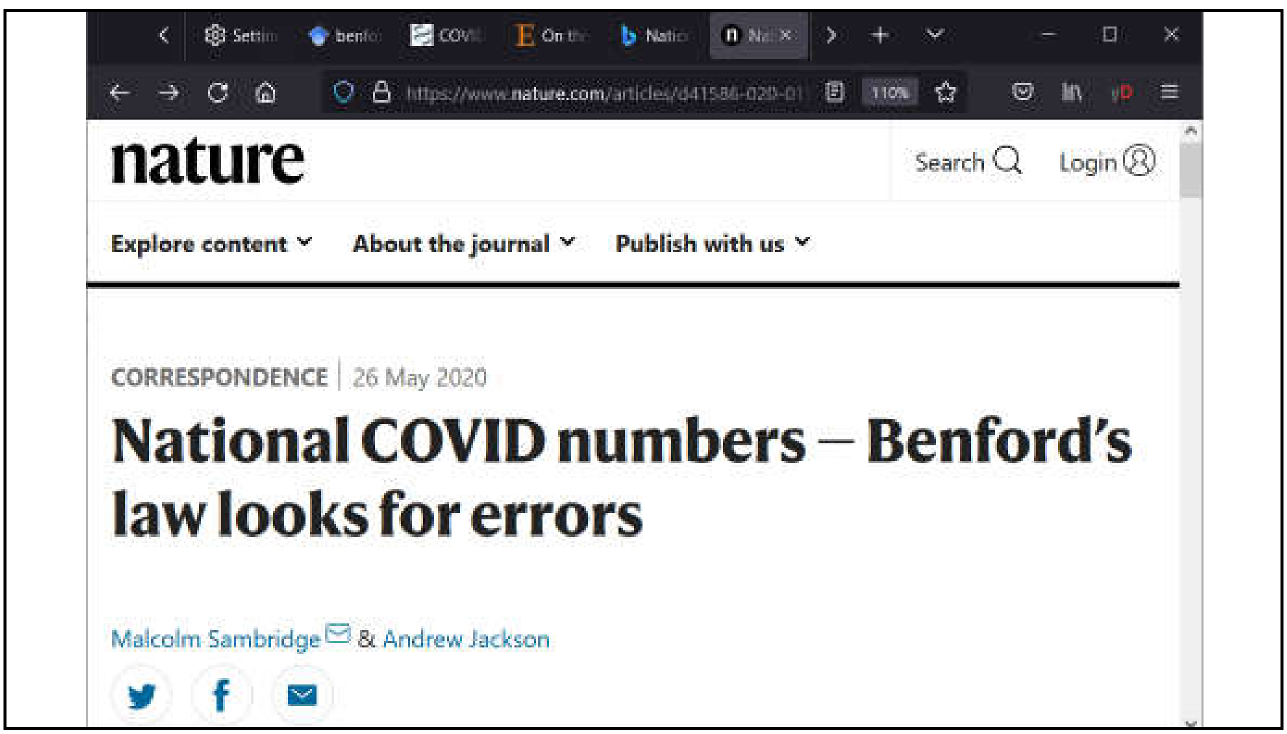
Show more

Outline | Share | Cite

https://doi.org/10.1016/j.econlet.2020.109573

Get rights and content

Highlights

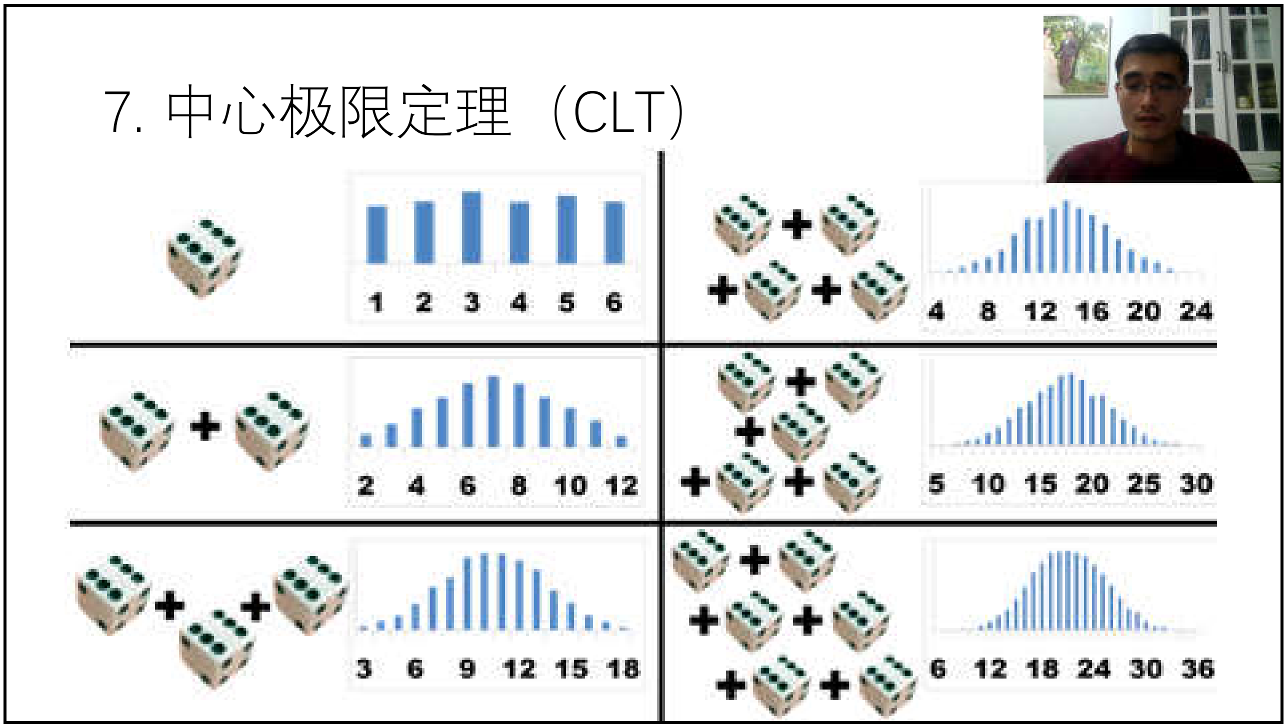


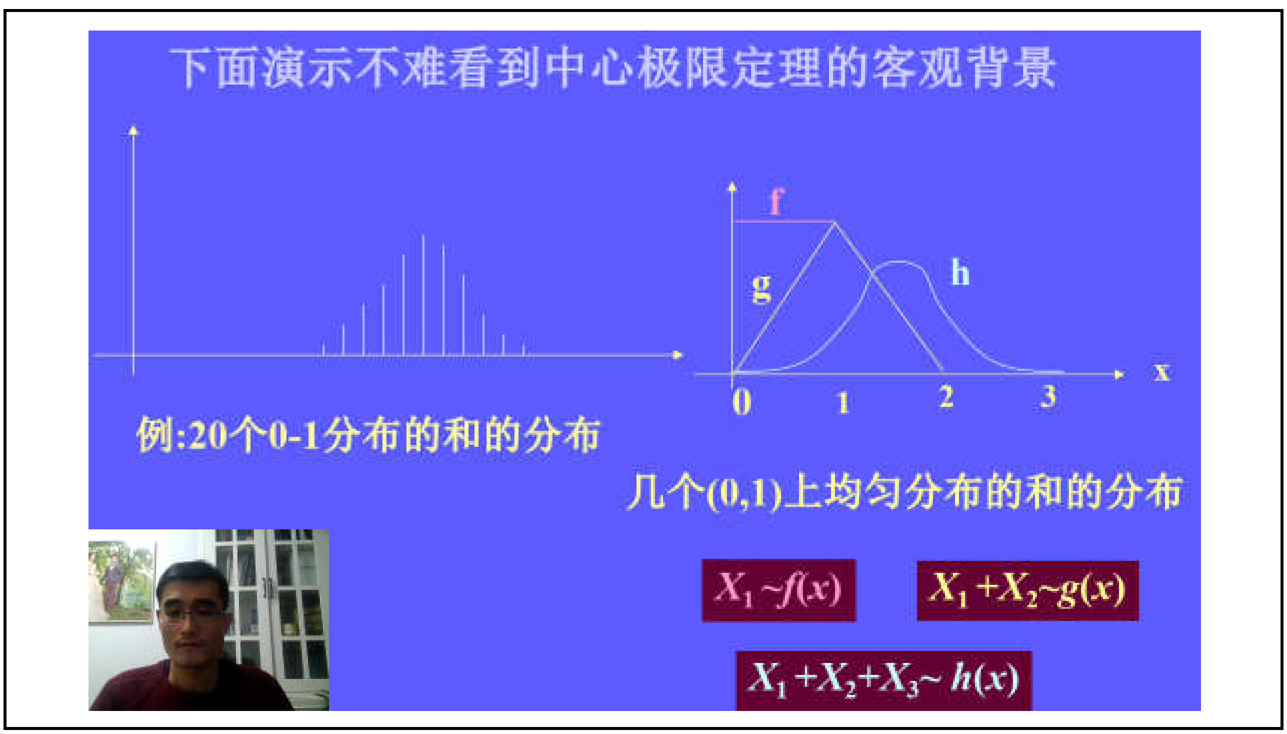


CORRESPONDENCE | 26 May 2020

# National COVID numbers — Benford's law looks for errors

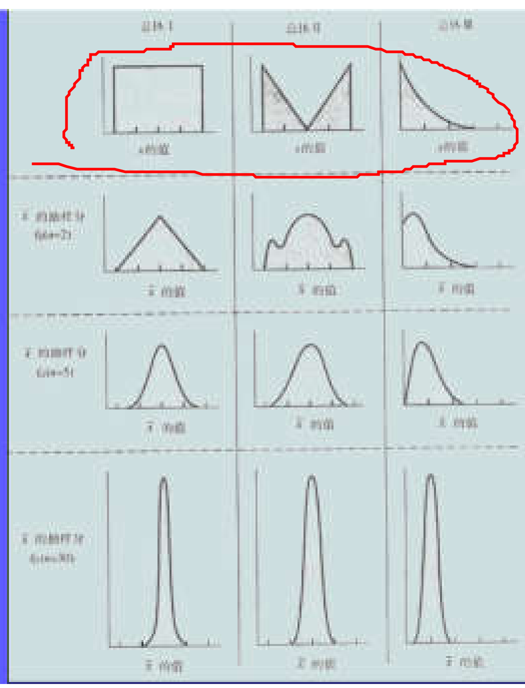
Malcolm Sambridge & Andrew Jackson


7. 中心极限定理（CLT）
1 2 3 4 5 6
4 8 12 16 20 24
2 4 6 8 10 12
5 10 15 20 25 30
3 6 9 12 15 18
6 12 18 24 30 36


下面演示不难看到中心极限定理的客观背景
f
g
h
0 1 2 3
x
例:20个0-1分布的和的分布
几个(0,1)上均匀分布的和的分布
$X_1 \sim f(x)$
$X_1+X_2 \sim g(x)$
$X_1+X_2+X_3 \sim h(x)$

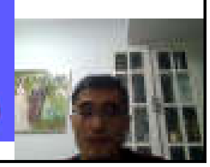
中心极限定理是概率论中最著名的结果之一，它不仅提供了计算独立随机变量之和的近似概率的简单方法，而且有助于解释为什么很多自然群体的经验频率呈现出钟形曲线这一值得注意的事实.

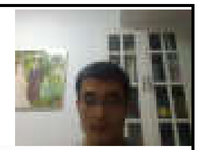
The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.



概率论

中心极限定理指的是样本均值的抽样分布接近于期望为u的正态分布。
大数定理指的是当样本量无穷大时，样本均值接近于总体均值u。
如图，当n逐渐增大，样本均值的方差越来越小，x拔收敛于u。



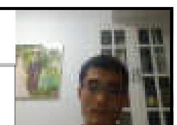


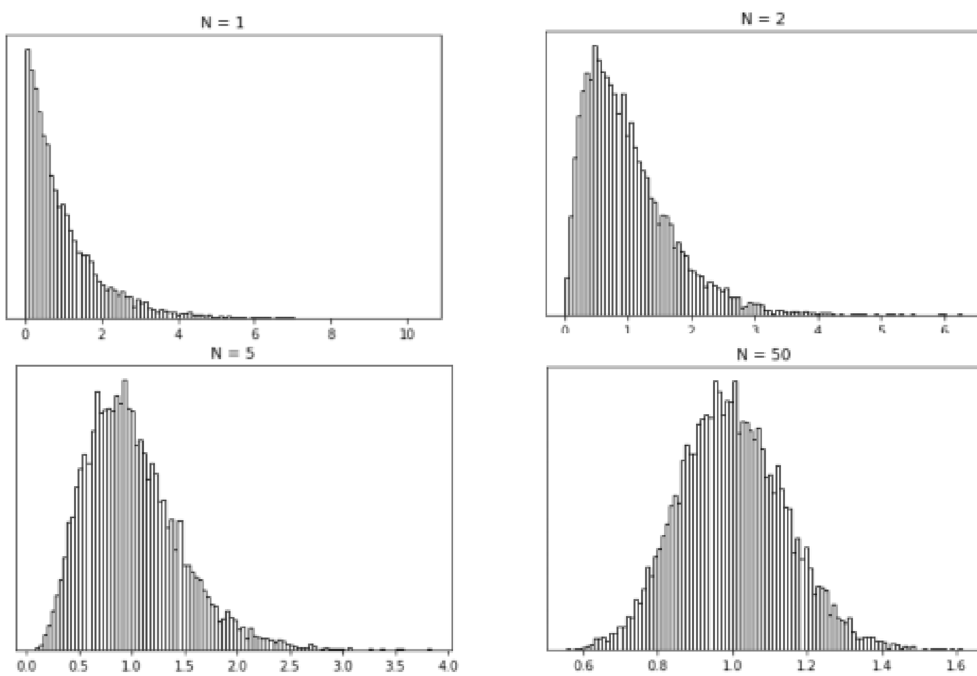
示例：使用指数分布作为底层/原子分布；Monte Carlo验证

underlying distribution: exponential.

```
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import collections
from scipy.stats import binom
from tqdm import tqdm
import numpy as np

for N in [1,2,5,50]:
    xbars = []
    for i in tqdm(range(10000)): # MC试验次数
        xbar = np.random.exponential(scale = 1, size = N).mean()
        xbars.append(xbar)
    plt.figure()
    plt.hist(xbars, density=False, bins=100, facecolor="none", edgecolor = "black")
    plt.title('N = ' + str(N))
    plt.yticks([])
    plt.show()
```

```
100%|████████████████████████████████████████| 100
s]
```



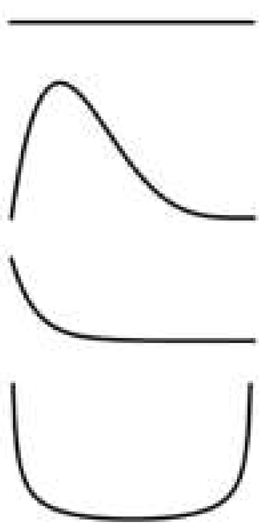


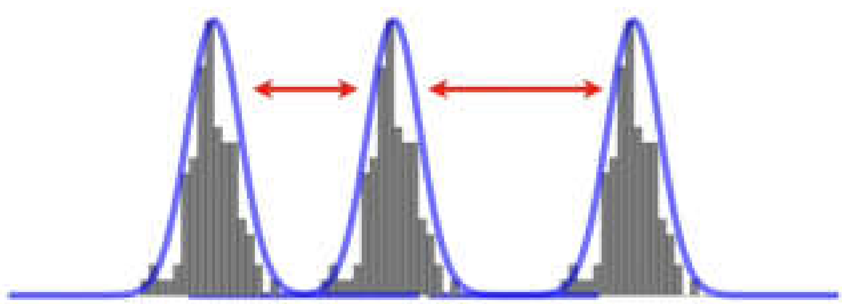
# Even if you're not normal...

中心极限定理使得很多参数检验成为可能，只要样本数量足够多（通常要求>30）即可，底层分布如何没有关系

因此，t检验对正态偏离的鲁棒性高

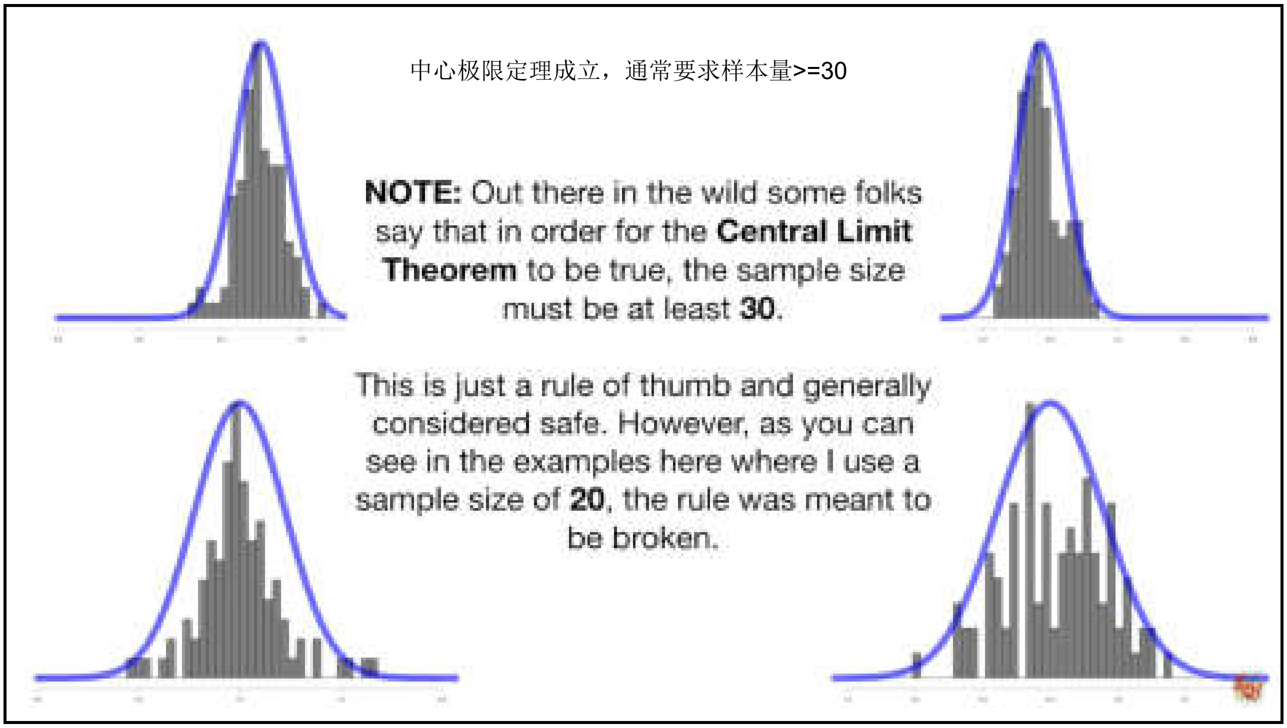
...and pretty much any statistical test that uses the sample mean.

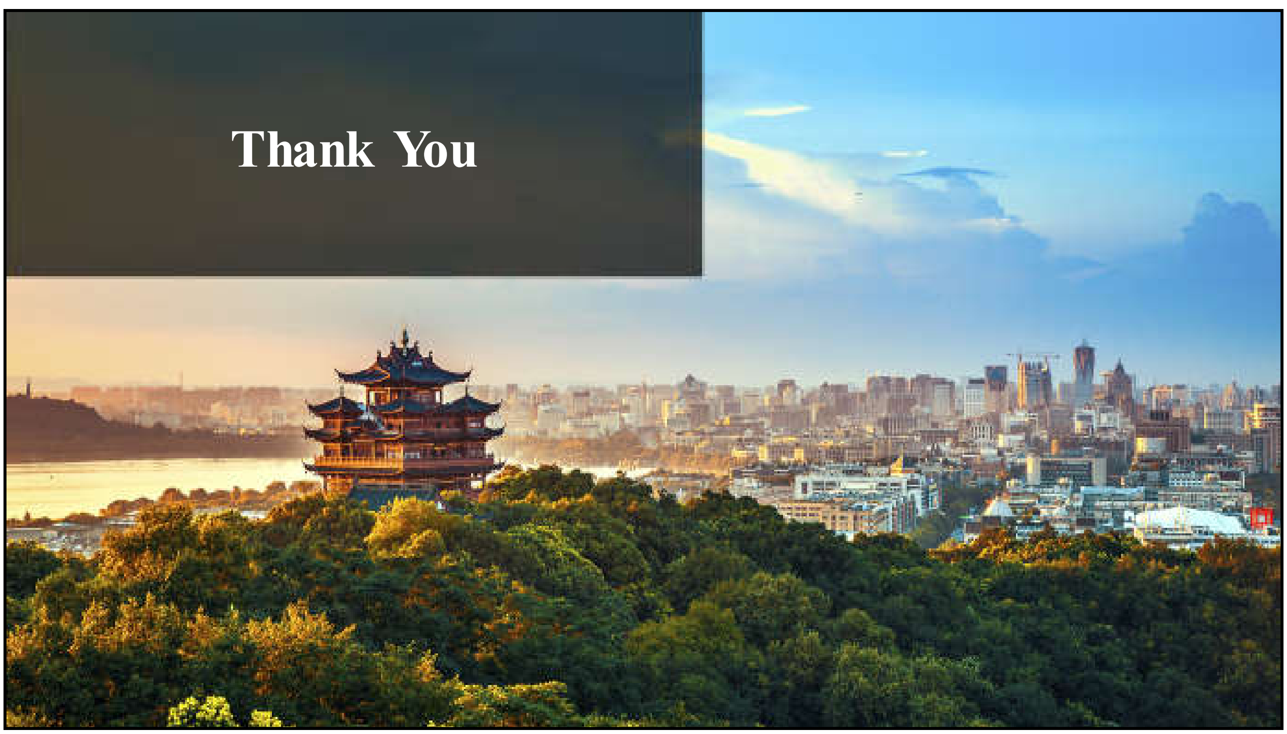




CLT makes many parametric mean tests **FEASIBLE** when the sample size is large enough (e.g., >30). We **don't care** the underlying distribution.

E.g., In the end, the t statistic depends only on the mean and variance of the two samples. The CLT says that (under most circumstances) those rapidly become normal even when the underlying population distribution is not. So the t-test is quite robust to (most) departures from normality. This has been verified by many simulation studies. Note, by the way, that it is not at all robust to departures from homogeneity of variance.


中心极限定理成立，通常要求样本量>=30
**NOTE:** Out there in the wild some folks say that in order for the **Central Limit Theorem** to be true, the sample size must be at least **30**.
This is just a rule of thumb and generally considered safe. However, as you can see in the examples here where I use a sample size of **20**, the rule was meant to be broken.


Thank You