

## 第 9 次课 随机数和简单的游戏

Python 科学计算

周吕文

宁波大学，机械工程与力学学院

2024 年 9 月 1 日



Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

### 提要

- 1 随机数
- 2 计算概率
- 3 蒙特卡罗积分
- 4 随机游动

### 在程序中使用随机数



### 随机数可以模拟不确定事件

#### 确定问题

- 通过一些确定的数值计算出准确结果的科学和技术问题
- 例如：小球上抛  $y(t) = v_0 t - gt^2/2$

#### 随机问题

- 不确定的问题
- 如：掷色子，分子运动，游戏
- 使用随机数可以模拟这些问题

random 模块可以获得随机数（均匀分布、正态分布）

```
>>> import random
>>> random.random()      # 获得 [0, 1) 区间均匀分布的随机数
0.22081626659632736
>>> random.random()
0.900245897285277
>>> random.uniform(2, 5) # 获得 [2, 5) 区间均匀分布的随机数
4.476477380965386
>>> random.uniform(2, 5)
4.588272300006601
>>> m, s = 0, 1          # 均值为 m, 标准差 s 正态分布随机数
>>> random.gauss(m, s)   # 等价于 random.normalvariate(m, s)
-0.5942641272595723
```

- 这些随机数其实是由一个确定的算法生成的，看起来随机而已
- 服从某种分布是指：生成大量的随机数后这些数服从该种分布

Notes

---

---

---

---

---

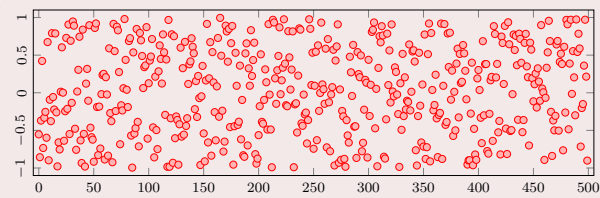
---

---

---

随机数分布可视化

```
uniform_numbers.py
import random, matplotlib.pyplot as plt
N = 500
r = [random.uniform(-1,1) for i in range(N)]
plt.plot(r, 'o')
plt.show()
```



Notes

---

---

---

---

---

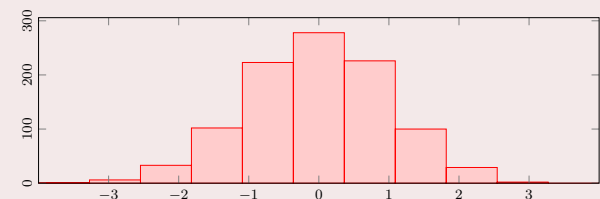
---

---

---

随机数分布可视化

```
normal_numbers.py
import random, matplotlib.pyplot as plt
m, s, N = 0, 1, 1000
r = [random.gauss(m, s) for i in range(N)]
plt.hist(r)
plt.show()
```



Notes

---

---

---

---

---

---

---

---

随机数函数

- random 中的 random() 和 gauss() 一次生成一个随机数
- numpy 中也有一个 random 模块，可以一次生成多个随机数

```
>>> from numpy import random
>>> random.random()
0.3607181765542483

>>> random.random(size=4)
array([0.51574507, 0.28846549, 0.879569 , 0.81521128])

>>> random.uniform(2, 3, size=4)
array([2.16352841, 2.000509 , 2.86011705, 2.3511122 ])

>>> random.normal(0, 1, size=4)
array([-1.7293026 , -2.36914419, -0.58049588, -1.98888084])
```

Notes

---

---

---

---

---

---

---

---

- 有时候我们需要  $[a, b]$  之间的整数随机数，而不是实数
- `random` 和 `numpy` 都提供了整数随机数函数

```
>>> import random
>>> r = random.randint(1,100)      # [1, 100]
>>> r
19

>>> import numpy as np
>>> r = np.random.randint(1,100,10) # [1, 100]
>>> r
array([53, 37, 77, 14, 65, 88, 63, 63, 24, 26])
```

例子：掷色子-出现 6 的概率是多少？

标量版本 roll\_die.py

```
import random
def six_eyes(N):      # 掷 N 次色子，计算出现 6 的概率
    M = 0
    for i in range(N):
        outcome = random.randint(1, 6)
        if outcome == 6:
            M += 1
    return M/N
```

```
>>> N = 10000
>>> six_eyes(N)
0.1688
>>> six_eyes(N)
0.1694
```

例子：掷色子-出现 6 的概率是多少？

向量版本 roll\_die.py

```
import numpy as np
def six_eyes_vec(N): # 掷 N 次色子，计算出现 6 的概率
    eyes = np.random.randint(1, 7, N)
    success = eyes == 6
    M = np.sum(success)
    return M/N
```

```
>>> N = 10000
>>> six_eyes_vec(N)
0.1707
>>> six_eyes_vec(N)
0.1652
```

- 使用数组比较和 `numpy` 中的数组求和比内置的 `random` 函数更高效

随机数程序的调试

- 随机数程序的调试困难，因为每次执行会生成不同的数据序列
- 可以设置随机种子，使随机数的生成序列固定 `random.seed(整数)`
- 缺省情况下，系统以当前时间作为种子

```
>>> import random
>>> random.seed(2)
>>> [eval('%.2f' % random.random()) for i in range(8)]
[0.96, 0.95, 0.06, 0.08, 0.84, 0.74, 0.67, 0.31]
>>> [eval('%.2f' % random.random()) for i in range(8)]
[0.61, 0.61, 0.58, 0.16, 0.43, 0.39, 0.72, 0.99]

>>> random.seed(2)
>>> [eval('%.2f' % random.random()) for i in range(8)]
[0.96, 0.95, 0.06, 0.08, 0.84, 0.74, 0.67, 0.31]
>>> [eval('%.2f' % random.random()) for i in range(8)]
[0.61, 0.61, 0.58, 0.16, 0.43, 0.39, 0.72, 0.99]
```

从列表中随机取出元素

方法 1: 使用 choice 方法

```
>>> awards = ['car', 'computer', 'ball', 'pen']
>>> import random
>>> random.choice(awards)
'ball'
```

方法 2: 生成随机序号

```
>>> index = random.randint(0, len(awards)-1)
>>> awards[index]
'pen'
```

方法 3: 洗牌

```
>>> random.shuffle(awards)
>>> awards[0]
'computer'
```

Notes

---

---

---

---

---

---

---

---

例子: 扑克牌 - 洗牌、拿牌

cards.py

```
import random
def make_deck():
    ranks = ['A', '2', '3', '4', '5', '6', '7',
             '8', '9', '0', 'J', 'Q', 'K'] # 0表示10
    suits = ['梅', '方', '红', '黑']
    deck = [s+r for s in suits for r in ranks]
    random.shuffle(deck)
    return deck
```

```
>>> deck = make_deck()
>>> deck[:6]
['梅7', '梅A', '黑2', '梅4', '梅0', '黑9']
>>> card = deck.pop(0) # 等价于 card = deck[0]; del deck[0]
>>> card
'梅7'
```

Notes

---

---

---

---

---

---

---

---

例子: 扑克牌 - 拿一手牌

cards.py

```
def deal_hand(n, deck):
    hand = [deck[i] for i in range(n)]
    del deck[:n]
    return hand, deck
```

```
>>> deck = make_deck()
>>> deck[:6]
['梅Q', '黑9', '梅6', '方2', '红J', '梅0']
>>> hand, deck = deal_hand(4, deck)
>>> hand
['梅Q', '黑9', '梅6', '方2']
>>> deck[:6]
['红J', '梅0', '梅3', '方9', '红3', '黑8']
```

Notes

---

---

---

---

---

---

---

---

例子: 扑克牌 - 发牌

cards.py

```
def deal(cards_per_hand, no_of_players):
    deck = make_deck()
    hands = []
    for i in range(no_of_players):
        hand, deck = deal_hand(cards_per_hand, deck)
        hands.append(hand)
    return hands
```

```
>>> players = deal(5, 4)
>>> from pprint import pprint
>>> pprint(players)
[['方2', '方Q', '黑5', '黑Q', '梅8'],
 ['方9', '方5', '方8', '梅J', '红Q'],
 ['黑K', '方A', '黑9', '梅A', '黑A'],
 ['方7', '红0', '梅9', '方0', '红3']]
```

Notes

---

---

---

---

---

---

---

---

例子：扑克牌 - 分析相同牌面

cards.py

```
def same_rank(hand, n_of_a_kind):
    # 给定一手牌，返回具有指定张数相同牌面的数量
    ranks = [card[1] for card in hand]
    counter = 0
    for rank in set(ranks):
        if ranks.count(rank) == n_of_a_kind:
            counter += 1
    return counter
```

```
>>> players[0]
['方2', '方Q', '黑5', '黑Q', '梅8']
>>> same_rank(players[0], 2)
1
```

周吕文 宁波大学

随机数

2024 年 9 月 1 日 17 / 47

Notes

---

---

---

---

---

---

---

---

例子：扑克牌 - 分析相同花色

cards.py

```
def same_suit(hand):
    # 给定一手牌，返回手牌中每种花色的牌的数量
    suits = [card[0] for card in hand]
    counter = {} # counter[suit] 存放 suit 花色的数量
    for suit in suits:
        count = suits.count(suit)
        if count > 1: # 只记录同种花色数量大于1的
            counter[suit] = count
    return counter
```

```
>>> players[0]
['方2', '方Q', '黑5', '黑Q', '梅8']
>>> same_suit(players[0])
{'方': 2, '黑': 2}
```

周吕文 宁波大学

随机数

2024 年 9 月 1 日 18 / 47

Notes

---

---

---

---

---

---

---

---

例子：扑克牌 - 分析一场牌局

cards.py

```
players = deal(5, 4)
for hand in players:
    print('手牌: %s\n      有%d对, %s个三条, %4s张同花' % \
          ('', '.join(hand), same_rank(hand,2), same_rank(hand,3),
           '+'.join([str(s) for s in same_suit(hand).values()])))
```

手牌: 方2, 方Q, 黑5, 黑Q, 梅8  
有1对, 0个三条, 2+2张同花  
手牌: 方9, 方5, 方8, 梅J, 红Q  
有0对, 0个三条, 3张同花  
手牌: 黑K, 方A, 黑9, 梅A, 黑A  
有0对, 1个三条, 3张同花  
手牌: 方7, 红0, 梅9, 方0, 红3  
有1对, 0个三条, 2+2张同花

周吕文 宁波大学

随机数

2024 年 9 月 1 日 19 / 47

Notes

---

---

---

---

---

---

---

---

例子：扑克牌 - 使用类实现

Deck.py (接下页)

```
import random
class Deck:
    def __init__(self):
        ranks = ['A', '2', '3', '4', '5', '6', '7',
                  '8', '9', '0', 'J', 'Q', 'K'] # 0表示10
        suits = ['梅', '方', '红', '黑']
        self.deck = [s+r for s in suits for r in ranks]
        random.shuffle(self.deck)
    def hand(self, n=1):
        hand = [self.deck[i] for i in range(n)]
        del self.deck[:n]
        return hand
    def deal(self, cards_per_hand, no_of_players):
        return [self.hand(cards_per_hand) \
                for i in range(no_of_players)]
```

Notes

---

---

---

---

---

---

---

---

例子：扑克牌 - 使用类实现

Deck.py (接上页)

```
def putback(self, card):
    self.deck.append(card)

def __str__(self):
    s = ', '.join(self.deck[:13])+'\n'+\
        ', '.join(self.deck[13:26])+'\n'+\
        ', '.join(self.deck[26:39])+'\n'+\
        ', '.join(self.deck[39:52])
    return s
def __repr__(self):
    return str(self)

def __len__(self):
    return len(self.deck)
```

Notes

---

---

---

---

---

---

---

---

例子：扑克牌 - 使用类实现

```
>>> from Deck import *; import pprint
>>> deck = Deck()
>>> deck
方 2, 方 4, 红 Q, 黑 7, 黑 Q, 黑 0, 方 6, 梅 A, 红 7, 黑 K, 黑 2, 红 2, 方 K
梅 K, 梅 0, 红 J, 红 K, 黑 6, 红 A, 梅 3, 红 6, 黑 J, 红 5, 梅 2, 方 A, 方 0
红 9, 红 0, 方 Q, 黑 5, 方 3, 方 7, 梅 8, 梅 5, 梅 Q, 方 J, 方 5, 梅 4, 方 8
梅 6, 梅 9, 梅 J, 红 3, 黑 4, 红 8, 黑 9, 方 9, 黑 8, 黑 3, 红 4, 黑 A, 梅 7
>>> len(deck)
52
>>> players = deck.deal(5, 4)
>>> pprint.pprint(players)
[['方 2', '方 4', '红 Q', '黑 7', '黑 Q'],
 ['黑 0', '方 6', '梅 A', '红 7', '黑 K'],
 ['黑 2', '红 2', '方 K', '梅 K', '梅 0'],
 ['红 J', '红 K', '黑 6', '红 A', '梅 3']]
```

Notes

---

---

---

---

---

---

---

---

课堂练习

Notes

---

---

---

---

---

---

---

---

1 猜数字

文件名: guess\_number.py

实现一个简单的游戏:

- 用计算机随机生成一个 1-100 之间的整数, 然后玩家猜测该数字。
- 对于每次猜测, 计算机机会告知该数字是太大还是太小, 直到猜中。
- 统计玩家共计猜测的次数。

Notes

---

---

---

---

---

---

---

---

提要

1 随机数

2 计算概率

3 蒙特卡罗积分

4 随机游动

使用蒙特卡洛（Monte Carlo）方法计算概率

蒙特卡洛模拟的原理：某件事发生的概率

- 模拟  $N$  次，事件  $A$  发生了  $M$  次，则  $A$  发生的概率是  $M/N$  ( $N \rightarrow \infty$ )
- 例子：两个色子，一黑一绿，统计扔色子时黑色子比绿色子大的概率

black\_gt\_green.py (标量版)

```
import random
N = 100000
M = 0
for i in range(N):
    black = random.randint(1, 6)
    green = random.randint(1, 6)
    if black > green:
        M += 1
print(' 概率: %.4f' % (M/N))
```

>>>

概率: 0.4168

Notes

---

---

---

---

---

---

---

---

使用随机数向量编写更高效的程序

black\_gt\_green\_vec.py (向量版)

```
import numpy as np
N = 100000
r = np.random.randint(1, 7, size=(2,N))
black, green = r[0,:], r[1,:]
M = np.sum(black > green)
print(' 概率: %.4f' % (M/N))
```

>>>

概率: 0.4172

black\_gt\_green\_exact.py (理论值 = 15/36)

```
combinations = [(black, green) for black in range(1, 7)
                 for green in range(1, 7)]
success = [black > green for black, green in combinations]
M = sum(success)
N = len(combinations)
print(' 概率: %.4f' % (M/N))
```

>>>

概率: 0.4167

蒙特卡洛模拟的准确度和效率如何？

```
>>> from timeit import timeit
>>> timeit("exec(open('black_gt_green.py').read())",\
...       number=1)
概率: 0.4165
0.1706356219983718

>>> timeit("exec(open('black_gt_green_vec.py').read())",\
...       number=1)
概率: 0.4137
0.003077155000937637

>>> timeit("exec(open('black_gt_green_exact.py').read())",\
...       number=1)
概率: 0.4167
0.0001891390002128901
```

Notes

---

---

---

---

---

---

---

---

例子：赌色子 - 每局1元，若黑大于绿，赢2元，玩不？

black\_gt\_green\_game.py (标量版)

```
import random
N = 10000
start_capital = 10
money = start_capital
for i in range(N):
    money -= 1 # 每局交 1 元
    black = random.randint(1, 6)
    green = random.randint(1, 6)
    if black > green:
        money += 2 # 黑>绿，赢 2 元

net_profit_total = money - start_capital
net_profit_per_game = net_profit_total/N
print(' 平均每局输赢数: %g 元' % net_profit_per_game)
```

>>>

平均每局输赢数: -0.1784 元

Notes

---

---

---

---

---

---

---

---

例子：赌色子 - 每局1元，若黑大于绿，赢2元，不玩！

black\_gt\_green\_game\_vec.py (向量版)

```
import numpy as np
N = 10000
r = np.random.randint(1, 7, size=(2,N))
start_capital = 10
money = start_capital - N
black, green = r[0,:], r[1,:]
M = np.sum(black > green)
money += 2*M
net_profit_total = money - start_capital
net_profit_per_game = net_profit_total/N
print('平均每局输赢数: %g 元' % net_profit_per_game)
```

>>>  
平均每局输赢数: -0.1886 元

思政点：远离赌博，十赌九输，十赌九诈

- 年轻人不要赌博，吸大麻：可以读博，写代码！

周启文

宁波大学

计算概率

2024 年 9 月 1 日

29 / 47

Notes

课堂练习（思政案例）

2 色宝（押大押小）

文件名: sic\_bo.py

赌场里有一种常见的押大押小的游戏，规则如下：

- 盅里有三个色子，掷出后会计算三个色子点数的总和。如果点数总和小于等于 10，则称为“小”；大于等于 11，则称为“大”。
- 如果三个骰子的点数相同（如三个 1，三个 2，...，三个 6），称为“围骰”，此时不论玩家押大还是押小，都算玩家输。
- 玩家每次投注一个筹码，押中赢取两个筹码，押错则输掉一个筹码。

请用蒙特卡洛模拟方法估算玩家押大或押小的胜率：

- 在模拟中设定赌场进行 100,000 次游戏，统计玩家押大、押小的胜率。
- 根据结果分析，赌场设置围骰规则后，玩家是否仍有公平的获胜机会？

思政点：远离赌博，十赌九输，十赌九诈

- 运气只是一时，概率才是本质，搞懂蒙特卡洛，远离欺诈赌博。

周启文

宁波大学

计算概率

2024 年 9 月 1 日

30 / 47

Notes

例子：摸球

帽子里有 12 个球，4 黑、4 红、4 蓝

```
>>> hat = [c for c in ['black', 'red', 'blue'] \
...         for i in range(4)]
```

随机从中摸出两个球

```
>>> import random
>>> index = random.randint(0, len(hat)-1)
>>> ball1 = hat[index]; del hat[index]
>>> ball2 = hat[index]; del hat[index]
>>> # or
>>> random.shuffle(hat)
>>> ball1 = hat.pop(0)
>>> ball2 = hat.pop(0)
>>> # or
>>> ball1 = random.choice(hat); hat.remove(ball1)
>>> ball2 = random.choice(hat); hat.remove(ball2)
```

Notes

例子：摸球 - 一次摸出 2 个或更多黑球的概率

balls\_in\_hat.py

```
import random
def new_hat():
    hat = [c for c in ['black', 'red', 'blue'] \
           for i in range(4)]
    random.shuffle(hat) # 打乱顺序
    return hat

n, N = 5, 10000 # 一次摸 n 个球，独立实验 N 次
M = 0
for e in range(N):
    balls = new_hat()[0:n]
    if balls.count('black') >= 2:
        M += 1
print('概率: ', M/N)
```

>>>  
概率: 0.5816

周启文

宁波大学

计算概率

2024 年 9 月 1 日

32 / 47

Notes



提要

Notes

- 1 随机数
- 2 计算概率
- 3 蒙特卡罗积分
- 4 随机游动

蒙特卡罗积分：原理

Notes

积分中值定理： $f_m$  是函数  $f(x)$  在  $[a, b]$  区间的平均值

$$\int_a^b f(x) \mathrm{d} x=f_m(b-a)$$

蒙特卡罗积分：在  $[a, b]$  区间选择  $n$  个随机数  $x_i$  估计  $f_m$

$$f_m \approx \frac{1}{n} \sum_{i=1}^n f\left(x_i\right)$$

蒙特卡罗积分：程序实现

Notes

MCint.py

```
import random, numpy as np
def MCint(f, a, b, n):
    s = 0
    for i in range(n):
        x = random.uniform(a, b)
        s += f(x)
    return (b-a)*s/n
def MCint_vec(f, a, b, n):
    x = np.random.uniform(a, b, n)
    s = np.sum(f(x))
    return (b-a)*s/n
```

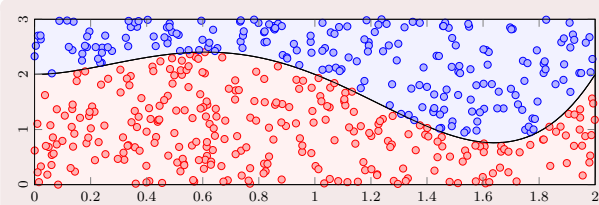
```
>>> f = lambda x: 1 + 2*x
>>> a, b, n = 1, 2, 1000
>>> MCint(f, a, b, n)
3.9834987128956767
>>> MCint_vec(f, a, b, n)
4.011263723641247
```

蒙特卡罗积分比传统梯形方法慢，但对于多变量积分更高效，如

$$\int f\left(x_1, x_2, \cdots, x_n\right) \mathrm{d} x_1 \mathrm{d} x_2 \cdots \mathrm{d} x_n$$

蒙特卡罗积分：随机投点法

Notes



- 问题：求矩形区域  $B=\left[x_L, x_H\right] \times\left[y_L, y_H\right]$  内一几何图形  $G$  的面积。
- 方法：在  $B$  中随机投  $N$  点， $M$  点落入  $G$ ，则  $G$  面积为  $\frac{M}{N} \operatorname{area}(B)$
- 特例： $G$  是  $y=f(x)$  与  $x$  轴在  $x \in[a, b]$  围成的区域，则

$$\operatorname{area}(G)=\int_a^b f(x) \mathrm{d} x \approx \frac{M}{N} m(b-a), \quad m \geq \max _{x \in[a, b]} f(x)$$

MCint\_area.py

```
import random, numpy as np
def MCint_area(f, a, b, m, N):    # 标量版本
    below = 0
    for i in range(N):
        x = random.uniform(a, b)
        y = random.uniform(0, m)
        if y <= f(x):
            below += 1
    return below/N*m*(b-a)

def MCint_area_vec(f, a, b, m, N): # 矢量版本
    x = np.random.uniform(a, b, N)
    y = np.random.uniform(0, m, N)
    below = np.sum(y < f(x))
    return below/N*m*(b-a)
```

Notes

---

---

---

---

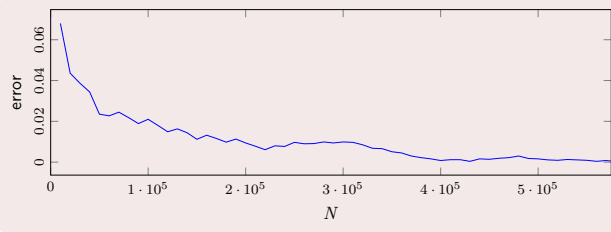
---

---

---

---

```
>>> f = lambda x: 2 + 3*x
>>> a = 1; b = 2; m = f(b); N = 1000000
>>> MCint_area(f, a, b, m, N)
6.493664
>>> MCint_area_vec(f, a, b, m, N)
6.495608
```



Notes

---

---

---

---

---

---

---

---

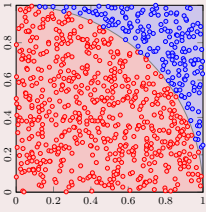
3 计算  $\pi$

文件名: MCpi.py

由于圆的面积与圆周率  $\pi$  有关，因此可以通过随机投点法来计算  $\pi$  的值，具体方法如下：

- 在  $(0,0)$ 、 $(1,0)$ 、 $(1,1)$ 、 $(0,1)$  构成的边长为 1 的正方形内，生成  $N$  个随机点。
- 统计以  $(0,0)$  为圆心，1 为半径的四分之一圆内点数量  $N_{in}$ 。
- 四分之一圆的面积与正方形的面积比为  $\pi/4$ ，因此可根据下式估计  $\pi$ ：

$$\pi \approx 4 \times \frac{N_{in}}{N}$$



请根据以上方法，编写程序计算  $\pi$ 。测试并绘制不同  $N$  值导致的误差图。

Notes

---

---

---

---

---

---

---

---

1 随机数

2 计算概率

3 蒙特卡罗积分

4 随机游动

Notes

---

---

---

---

---

---

---

---

---

---

---

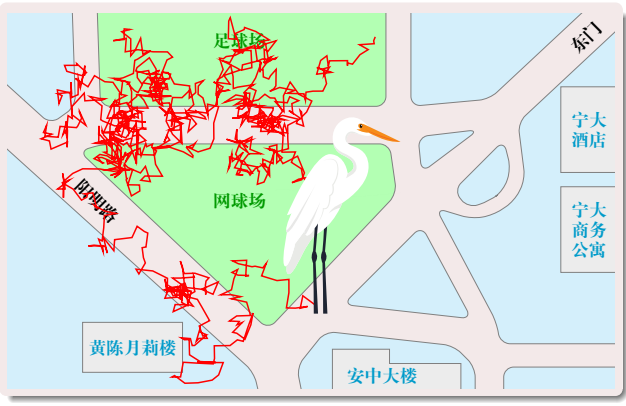
---

---

---

---

---



一维空间随机游走

基本概念

- 每个粒子每一步都等概率地向左或者向右。某粒子的运动可表示为
$$x_t = x_{t-1} + s, \quad x_0 = 0, \quad P(s = 1) = P(s = -1) = 1/2$$
- 当  $t = 0$  时,  $N$  个粒子在位置 0, 它们随时间的分布是?

应用

- 分子运动 (molecular motion)
- 热传递 (heat transport)
- 量子力学 (quantum mechanics)
- 聚合链 (polymer chains)
- 群体遗传学 (population genetics)
- ...

---

---

---

---

---

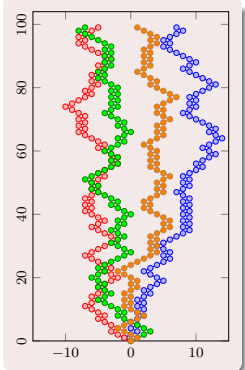
---

---

---

一维空间随机游走

```
walk1D.py
import random, numpy as np
import matplotlib.pyplot as plt
Np = 4 # 粒子数
Ns = 100 # 步数
x = np.zeros((Ns, Np))
t = np.arange(Ns)
for i in t[1:]:
    for p in range(Np):
        if random.random() < 0.5:
            x[i,p] = x[i-1,p] + 1
        else:
            x[i,p] = x[i-1,p] - 1
plt.plot(x, t, '-o')
plt.show()
```



---

---

---

---

---

---

---

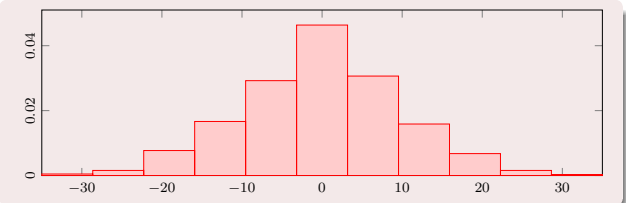
---

一维空间随机游走

相比一个粒子的运动状态, 可能更关心群体效应 (平均位置、分布)

```
x = x[-1,:]
```

```
mean_pos = np.mean(x) # x 由 Np = 1000; Ns = 100 计算得到
stdev_pos = np.std(x) # 平均位置
plt.hist(x) # 集群宽度
plt.show() # 位置分布
```



---

---

---

---

---

---

---

---

二维空间随机游走

基本概念

- 每一步随机上下左右，每个方向的概率都是 1/4
- $N_p$  个粒子，随机走  $N_s$  步，群体分布情况如何？

walk2D.py (标量版)

```
import random, numpy as np
def random_walk_2D(Np, Ns):
    x, y = np.zeros(Np), np.zeros(Np)
    for step in range(Ns):
        for i in range(Np):
            direction = random.randint(1, 4)
            if direction == 1: y[i] += 1 # 上
            elif direction == 2: y[i] -= 1 # 下
            elif direction == 3: x[i] += 1 # 右
            elif direction == 4: x[i] -= 1 # 左
    return x, y
```

Notes

---

---

---

---

---

---

---

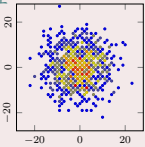
---

二维空间随机游走

walk2D.py (矢量版)

```
def random_walk_2D_vec(Np, Ns):
    x, y = np.zeros(Np), np.zeros(Np)
    moves = np.random.randint(1, 5, size=(Ns, Np))
    for step in range(Ns):
        this_move = moves[step,:]
        y += np.where(this_move==1, 1, 0) # 上
        y -= np.where(this_move==2, 1, 0) # 下
        x += np.where(this_move==3, 1, 0) # 右
        x -= np.where(this_move==4, 1, 0) # 左
    return x, y

import matplotlib.pyplot as plt
x, y = random_walk_2D_vec(1000, 100)
plt.plot(x,y,'x')
plt.show()
```



Notes

---

---

---

---

---

---

---

---

课堂练习

4 二维空间随机游走的类实现 文件名: walk2D\_classes.py

编写程序使用类实现粒子随机游走的模拟。定义类 **Particle**，实现单个粒子的定义和移动。该类包含以下属性和方法：

- 属性：粒子的位置  $x$ 、 $y$
- 方法：初始化方法及用来随机移动一步（上、下、左、右）的 **move** 方法。

定义类 **Particles**，实现多个粒子一步和多步随机游走的模拟。该类包含以下属性和方法：

- 属性：存储多个 **Particle** 实例的列表
- 方法：初始化方法、所有粒子移动一步的 **move** 方法、所有粒子移动  $n$  步的 **moves** 方法，以及绘图方法。

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

The End!