4403 Computational Modelling

- Chapter One
  - **Computational Modelling:**
    - **Definition**: The use of computers to simulate complex systems using **mathematics, physics, and computer science**.
    - **Simulation vs. Modelling**:
      - **Modelling** creates a computational model contains numerous variables that characterize the system being studied that captures system behavior.
      - **Simulation** involves running the model, adjusting variables, and observing outcomes.**Simulation** is the process of running a computational model.
        - Simulations often involve running multiple iterations and are crucial for understanding **complex systems**
  - **Complex Systems:**
    - **Definition**: A system composed of many components (sub-systems) interacting with each other.由许多相互影响的组件（子系统）组成的系统。
    - **Features of Complex Systems**:
      - **1 Interactions**: Interactions may generate novel information that make it difficult to study components in isolation or to completely predict their future.
      - **2 Emergence**: Properties of the whole system cannot be predicted solely from its parts.The whole is more than the sum of its parts.
        - Phenomenon of Emergence： Interaction between components generate novel information and exhibit non-trivial collective structures and behaviors at larger scales.各组成部分之间的相互作用会产生新的信息，并在更大尺度上表现出非同寻常的集体结构和行为。
      - **3 Dynamism**: Systems change states unpredictably (e.g., weather). state: sets of variables that best characterize the system.
      - **4 Self-Organization**: Systems develop global pattern or behaviour without external control. single cell dividing and self-organizing into complex shape of an organism 系统在不受外部控制的情况下发展出全局模式或行为。如单个细胞分化形成器官。
      - **5 Adaptation**:适应性 Systems evolve and adapt to their environment. When the components are damaged or removed, these systems are often able to adapt and recover their previous functionality, and sometimes they become even better than before.当组合受损或者被清除，系统通常可以自适应和恢复之前的功能
      - 6 **Interdisciplinarity**:跨学科性 Complex systems are found across various fields like biology, physics, social sciences, etc.
  - **Complexity Science:**

- **Definition**: Focuses on modelling **complex systems** through **simulation** rather than mathematical analysis. 通过模拟而不是数学分析
- **Comparison with Classical Science**:
  - Classical science focuses on **law-based** equations (e.g., Newton's laws).
  - Complexity science is **rule-based**, using computational models to simulate interactions in systems instead of mathematical derivation(e.g., Schelling's model for urban segregation).
- **Key Differences**:
- Classical science is deterministic, reductionist, and predictive. 经典科学是决定论、还原论和预测论。
- Complexity science is stochastic, holistic, and explanatory. 复杂性科学是随机的、整体的和解释性的。

Chapter Two

- **Types of Graphs:**
  - **Graph Definition**: A representation of a system with discrete and interconnected elements (nodes/vertices and edges/links).

    $$G = (V, E)$$

  - 𝑉: vertices or nodes, to represent the elements in the system
  - 𝐸: edges or links, to represent the interconnections between elements
  - This notation describes the basic structure of a graph:
    - Each edge in E connects two vertices in V.
    - For example, in a social network, the vertices V could represent people, and the edges E could represent friendships or interactions between them.
  - **Types of Graphs**:
    - **Directed Graph**: A graph where all the edges are directed from one node to another node. 所有边都有自一个点指向另一个点的方向。
    - **Undirected Graph**: No specific direction for edges.没有明确的方向The edges indicate a two-way relationship,所有边都指向双向关系 in that each edge can be traversed穿过 in both directions.
    - **Weighted Graph**: Edges have numerical weights.
    - **Complete Graph**: Every node is connected to every other node.
      - Degree of a node : the number of neighbours of each node 每个点的邻居数量或者是他们连接的边的数量
      - A complete graph of 𝑛 nodes has $n(n-1)/2$ edges, with the degree of each node equals to **$n-1$.**

- code： **yield** allows a function to **produce a series of values** over time, whereas return sends back **one final value** and exits the function.

```
def all_pairs(nodes):
    for i, u in enumerate(nodes):
        for j, v in enumerate(nodes):
            if i>j:
                yield u, v
```

We can use `all_pairs` to construct a complete graph:

```
def make_complete_graph(n):
    G = nx.Graph()
    nodes = range(n)
    G.add_nodes_from(nodes)
    G.add_edges_from(all_pairs(nodes))
    return G
```

- yield 是 Python 中的一个关键字，它用于将一个函数变成 **生成器（generator）** 。生成器是一种特殊的迭代器，它能逐个生成值，而不是一次性返回所有的结果。当函数中包含 yield 时，它不会像普通函数那样运行并退出，而是可以暂停执行，返回一个值，然后在需要时继续执行。这在处理大数据集或数据流时非常有用，因为它不会一次性将所有数据加载到内存中。
- **Connected graphs**
  - A graph is connected if there is a path from every node to every other node.

```
def reachable_nodes(G, start):
    seen = set()
    stack = [start]
    while stack:
        node = stack.pop()
        if node not in seen:
            seen.add(node)
            stack.extend(G.neighbors(node))
    return seen
```

```
def reachable_nodes_bfs(G, start):
    seen = set()
    queue = deque([start])
    while queue:
        node = queue.popleft()
        if node not in seen:
            seen.add(node)
            queue.extend(G.neighbors(node))
    return seen
```

- **Multi-Layered Graph**: Different layers represent various aspects of relationships.

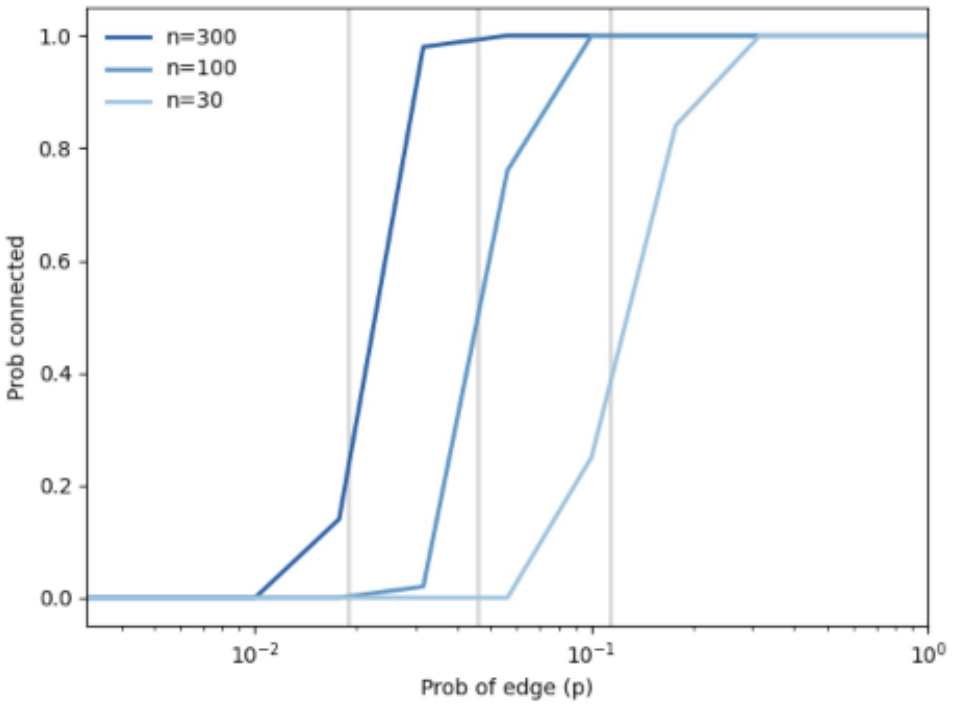
- **Regular graph** is a graph that each node has the **same** number of neighbours, or every node has the same degree.
- 𝑘-regular graph or regular graph of degree 𝑘: a regular graph with vertices of degree 𝑘

- **Erdős–Rényi (ER) Graph:**
  - **Random Graph**: Nodes and edges are generated randomly.

    $$G(n,p)$$

    - 𝑛 is the number of nodes and 𝑝 is the probability that there is an edge between any two nodes.
    - **用numpy.random.random** returns a number between 0 and 1 去判断
  - **Graph Connectivity:** Measures whether *there is a path between every pair of nodes.* Connectivity increases as ***p*** increases, with a critical value临界值 p^* = \frac{\ln n}{n} beyond which the graph is likely to be connected. The graph is unlikely to be connected is p is smaller than the critical value and more likely to be connected if p is larger than the critical value

    $$p^* = \frac{\ln n}{n}$$



- As ⊠ increases, the critical value ⊠⊠ gets smaller, and the *transition gets more abrupt*.随着 ⊠ 的增大，临界值 ⊠⊠ 越来越小，过渡也越来越突然。

- **Random vs. Regular Graph:**
  - **Random Graph**: Low clustering and low path lengths.
  - **Regular Graph**: High clustering and high path lengths.
  - **Important Graph Properties**:(three features)
    - 1 **Emergence**：Local Interaction -> Global Patterns
    - **2 Clustering Coefficient**: Measures the likelihood that two nodes connected to the same node are also connected to each other.
      - 1.For a node u with k neighbors：If all neighbors of node u are connected to each other, the number of possible edges between them is given by the formula {k(k - 1)}/{2} . This represents the maximum number of edges that could exist among the neighbors of u .

```
def node_clustering(G, u):
    neighbors = G[u]
    k = len(neighbors)
    if k < 2:
        return np.nan

    possible = k * (k-1) / 2
    exist = 0
    for v, w in all_pairs(neighbors):
        if G.has_edge(v, w):
            exist +=1
    return exist / possible
```

- 写步骤：
  - 1 Finding neighbours of node u
  - 2 Ensuring sufficient neighbours for triangle to form
  - 3 Determining the possible edges: k(k − 1)/2
  - 4 Counting the edges that actually exist
  - 5 Returning proportion of edges that exist
- 2. **Local Clustering Coefficient** (*Cu*):
  - This is the ratio of the actual number of edges between **the neighbors of *u*** to the **total number of possible edges between them**.

$$C_u = \frac{\text{number of edges between neighbors of } u}{\text{total possible edges between neighbors of } u}$$

    - □□ is the **fraction of those edges that actually exist.** **□□ 是这些边中实际存在的部分。**

      - If node u has k neighbors, the total possible edges between these neighbors is $\frac{k(k-1)}{2}$.
      - The value of C_u ranges from 0 to 1:
        - $C_u = 1$ means all neighbors of node u are connected to each other (forming a complete subgraph or clique).
        - $C_u = 0$ means none of the neighbors are connected.
  - It shows how connected the neighbors of a particular node are to each other.
- 3.**Graph Average Clustering Coefficient ((or Average Clustering Coefficient):**):

$$(\bar{C})$$

  - This is the average of the local clustering coefficients (*Cu*) across all nodes in the graph.

$$\bar{C} = \frac{1}{|V|}\sum_{u \in V} C_u$$

- $|V|$ is the total number of nodes in the graph.
- This provides an indication of the overall tendency of nodes to form tightly connected clusters across the entire network.

- 3 **Path Length**: The average distance (in edges) between any two nodes in the graph.
  - Average distance between nodes is measured in number of edges on the shortest path. 节点之间的平均距离以最短路径上的边数衡量。
- Neither is good for modelling graphs of high clustering and short path length
  - 1. **规则格子网络（Regular Lattice Network）**：
  - • 规则格子网络的特点是每个节点只与其相邻的节点连接，因此具有**高聚类**特性。然而，由于节点只能通过相邻节点逐步传递信息，导致其**平均路径长度较长**，信息传递效率低。
  - • 这种网络结构在局部上有很多连接（高聚类），但整体上缺乏全局的快速信息传递机制，因此不符合“短路径长度”的要求。
  - 2. **随机图模型（Random Graph Model）**：
  - • 随机图模型（如Erdős–Rényi随机图）中，节点之间的连接是随机分配的，这通常会导致**较短的平均路径长度**，即信息可以在较少的步骤内传递到全局。
  - • 然而，随机图模型的连接是随机的，所以**聚类系数较低**，即一个节点的邻居之间通常没有直接连接。这种结构缺乏局部的高连接性，因此无法满足“高聚类”的特性。

Chapter Three

Chapter Four

- **Introduction to Cellular Automata (CA)**:
  - Cellular Automata are discrete computational models used to simulate complex systems. They consist of grids of cells where each cell has a finite state (often binary), and evolves in discrete time steps based on a set of rules.**元胞自动机（Cellular Automata, CA）** 是一种离散的计算模型，常用于模拟复杂系统。它由一个由许多小单元格组成的网格构成，每个单元格称为一个“细胞”，每个细胞都处于一种有限的状态，通常是二进制状态
- **Wolfram's Rules**:
  - Stephen Wolfram studied 1-D Cellular Automata and defined rules for how cells evolve. Each cell's state depends on its current state and the state of its neighbors. He categorized CA rules, with Rule 50 and Rule 90 as examples of how different patterns can emerge.
- **Classes of Cellular Automata**:
  - **Class 1**: Simple systems that evolve into stable, homogeneous states.简单系统在经过一段时间后会演变成**稳定的、同质的状态**，
  - **Class 2**: Systems that generate repetitive or fractal-like patterns. 这类系统产生**重复的或类分形的图案**，通常会出现周期性结构。

    - **Class 3**: Systems that evolve chaotically, with pseudo-random behavior. 这类系统表现出**混沌的、伪随机的行为**，没有明显的周期性或稳定的图案。
    - **Class 4**: Complex systems that exhibit local structures that can interact over time, often used to model real-world systems like computing.是**最复杂的类别**，它们生成**局部结构**，这些结构可以相互作用去模拟真实世界系统
- **Examples and Applications**:
    - Cellular Automata are abstract models of physical systems, such as the patterns observed on cone snail shells. They provide a framework to understand complex behaviors in a simplified way.

- Chapter Five
    - **Game of Life:** developed by John H. Conway in 1970
        - This section explains Conway's **Game of Life**, a 2D cellular automaton with cells in a grid, where each cell can be alive or dead. The state of each cell **evolves** based on the number of live neighbors. 是一种二维元胞自动机，细胞分布在网格中，每个细胞都有可能存活或死亡。每个单元格的状态根据活邻居的数量而变化。
        - **Rules:**
            - The next state of each cell depends on its current state and its number of live neighbours.
            - Alive cells with 2 or 3 neighbors survive; others die.
            - Dead cells with exactly 3 live neighbors become alive.
        - The lecture presents examples of patterns that arise in the Game of Life, including
        - **still life** a **pattern** does not change from one generation to the next.不会改变的模式
        - **oscillators** 振荡器:A **pattern** that repeats itself after a fixed number of generations (known as its period).
        - **spaceships** :is a finite **pattern** that returns to its initial state after a number of generations or periods, but in a different location 一个会回到最初状态的模式，但在不同的位置。

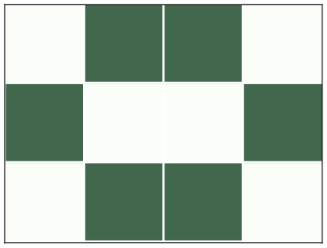


Figure 6.1: A stable pattern called a beehive.

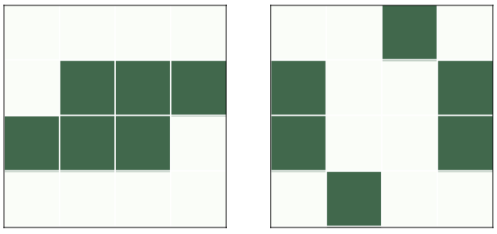


Figure 6.2: An oscillator called a toad.

Figure 6.3: A spaceship called a glider.

- **Complex Patterns in Game of Life**:
  - patterns are so long-lived, they are called “Methuselahs”.
  - Patterns like the **R-pentomino** (Methuselah) take many generations to stabilize.
  - The lecture introduces complex, long-lived structures initial patterns that never stabilize like the **Gosper Glider Gun** and **Puffer Train**, which exhibit unbounded growth.
  - a = np.random.randint(2, size=(10, 10), dtype=np.uint8)

The output will be a 10x10 matrix with random 0s and 1s, such as

```
[[1 0 1 0 1 1 0 0 1 0]
 [0 1 0 0 1 0 1 1 0 1]
 [1 1 0 1 0 0 0 1 0 1]
 [0 0 1 1 0 0 1 0 1 0]
 [1 1 0 0 1 1 0 1 0 1]
 [0 1 1 1 0 1 1 0 0 0]
 [1 0 0 1 0 1 0 1 1 0]
 [1 0 0 0 1 0 1 1 0 1]
 [0 1 1 0 1 1 0 0 1 0]
 [1 0 0 1 0 0 1 1 0 0]]
```

- c = correlate2d(a, kernel, mode='same') 这一步是把cell周围活着的邻居数量 计算出来
- c = [[3, 2, 1],
- [2, 3, 0],
- [1, 2, 3]]
- 判断活着邻居的格子 true = 1 false = 0

```
b = (c==3) | (c==2) & a
b = b.astype(np.uint8)
```

- 优化版，使用表格来查找单元格值

```
table = np.zeros(20, dtype=np.uint8)
table[[3, 12, 13]] = 1
c = correlate2d(a, kernel, mode='same')
b = table[c]
```

- **table = np.zeros(20, dtype=np.uint8)** This line creates a 1D NumPy array named table with 20 elements, all initialized to zero. The data type of the array is np.uint8, which allows storing integer values between 0 and 255.
- table[[3, 12, 13]] = 1: This line sets the elements at indices 3, 12, and 13 in the table array to 1.
- [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
- 在NumPy数组中，确实可以使用类似 array[[1, 2, 3]] 的方式同时修改多个位置的值，需要注意的是，这种用法是NumPy数组（ndarray）的特性，而不是Python的原生列表（list）的特性。对于Python的原生列表，你不能直接使用这种方式来修改多个位置的值。你需要使用循环或者列表推导来实现类似的功能。

Suppose the array c is:

```
c = [[3, 2, 12],
     [5, 13, 0],
     [1, 7, 3]]
```

And the `table` is:

```
table = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
```

The lookup for each element in c would be:

- `c[0, 0] = 3 → b[0, 0] = table[3] = 1`
- `c[0, 1] = 2 → b[0, 1] = table[2] = 0`
- `c[0, 2] = 12 → b[0, 2] = table[12] = 1`
- `c[1, 0] = 5 → b[1, 0] = table[5] = 0`
- `c[1, 1] = 13 → b[1, 1] = table[13] = 1`
- `c[1, 2] = 0 → b[1, 2] = table[0] = 0`
- `c[2, 0] = 1 → b[2, 0] = table[1] = 0`
- `c[2, 1] = 7 → b[2, 1] = table[7] = 0`
- `c[2, 2] = 3 → b[2, 2] = table[3] = 1`

The resulting array b would be:

```
b = [[1, 0, 1],
     [0, 1, 0],
     [0, 0, 1]]
```

- **Reaction-Diffusion Systems**:
  - Cellular Automata are used to simulate **diffusion** (e.g., the spreading of chemicals on a 2D grid) and **reaction-diffusion** processes, where two chemicals react and diffuse, producing complex patterns.两种化学物质发生反应并扩散，产生复杂的图案。
  - Varying **feed rate and kill rates** can generate different behaviors in the system.
  - With different parameters, this model can produce patterns similar to the stripes and spots on a variety of animals. In some cases, the similarity is striking, especially when the feed and kill parameters vary in space.

- Chapter Six
  - ## Reaction- diffusion
  - definition:A **simulation** of two chemicals reacting and diffusing on a 2-D grid. Reaction consumes chemical A and produces chemical B。

```
    def step(self):
        A = self.array
        B = self.array2
        ra, rb, f, k = self.params

        cA = correlate2d(A, self.kernel, **self.options)
        cB = correlate2d(B, self.kernel, **self.options)

        reaction = A * B**2
        self.array += ra * cA - reaction + f * (1-A)
        self.array2 += rb * cB + reaction - (f+k) * B
```

The parameters are

> The diffusion rate of `A` (analogous to `r` in the previous section).
>
> The diffusion rate of `B`. In most versions of this model, `rb` is about half of `ra`.
>
> The "feed" rate, which controls how quickly `A` is added to the system.
>
> The "kill" rate, which controls how quickly `B` is removed from the system.

The arrays `cA` and `cB` are the result of applying a diffusion kernel to `A` and `B`. Multiplying by `ra` and `rb` yields the rate of diffusion into or out of each cell.

The term `A * B**2` represents the rate that `A` and `B` react with each other. Assuming that the reaction consumes `A` and produces `B`, we subtract this term in the first equation and add it in the second.

The term `f * (1-A)` determines the rate that `A` is added to the system. Where `A` is near 0, the maximum feed rate is `f`. Where `A` approaches 1, the feed rate drops off to zero.

Finally, the term `(f+k) * B` determines the rate that `B` is removed from the system. As `B` approaches 0, this rate goes to zero.

As long as the rate parameters are not too high, the values of `A` and `B` usually stay between 0 and 1.

- ra * cA：表示 A 的扩散，受参数 ra 控制。 rb * cB：表示 B 的扩散，受参数 rb 控制。
- - reaction：扣除 A 在反应中消耗的部分。+ reaction：增加 B 在反应中生成的部分。
- + f * (1 - A)：恢复项，使 A 有一定几率恢复到原有状态，受参数 f 控制。- (f + k) * B：衰减项，表示 B 会自然衰减，受参数 f 和 k 控制。

- **Percolation**渗流

- **Percolation is a process in which a fluid flows through a semi-porous material.** Examples include oil in rock formations, water in paper, and hydrogen gas in micropores. **渗流（Percolation）** 是指一种流体流过半多孔材料的过程。

  In this model:

  - Initially, each cell is either "porous" with probability `q` or "non-porous" with probability `1-q`.
  - When the simulation begins, all cells are considered "dry" except the top row, which is "wet".
  - During each time step, if a porous cell has at least one wet neighbor, it becomes wet. Non-porous cells stay dry.
  - The simulation runs until it reaches a "fixed point" where no more cells change state.

- If there is a path of wet cells from the top to the bottom row, we say that the CA has a "percolating cluster"渗流群.
- 在渗流模型中，一个系统（通常是二维或三维网格）由多个单元组成，每个单元都有一定的概率 q 是"多孔的"（允许流体通过），或者以概率 1-q 是"非多孔的"（不允许流体通过）
- **There is a rapid change of the probability of a percolation clustering at the critical value.在临界值处出现渗流聚类的概率会迅速变化
- 在渗流模型中，临界值是指系统发生显著变化的点。在渗流聚集的情况下，这个临界值通常代表流体在多孔介质中迅速传播或停止传播的概率阈值
- Starting from an initial value of q, we construct a Percolation object and check whether it has a percolating cluster渗流群. If so, q is probably too high, so we decrease it. If not, q is probably too low, so we increase it.

```
def find_critical(n=100, q=0.6, iters=100):
    """Estimate q_crit by random walk.

    returns: list of q that should wander around q_crit
    """
    qs = [q]
    for i in range(iters):
        perc = Percolation(n, q)
        if test_perc(perc):
            q -= 0.005
        else:
            q += 0.005
        qs.append(q)
    return qs
```

- **Critical Systems**:
  - **Phase Change** 相变
    - The **rapid change** in behaviour near the critical value is called a phase change by analogy with phase changes in physical systems, like the way water changes from liquid to solid at its freezing point.临界值附近行为的快速变化被称为相变，类似于物理系统中的相变，如水在冰点时从液态变为固态的过程。
    - A wide variety of systems display a common set of behaviours and characteristics when they are at or near a critical point. These behaviours are known collectively as critical phenomena.各种系统在临界点或临近临界点时都会表现出一系列共同的行为和特征。这些行为统称为临界现象。
  - Critical Systems demonstrate common behaviours 临界系统展示共同行为：
    - **Fractal geometry**: 分形几何
      - freezing water tends to form fractal patterns, including snowflakes and other crystal structures.
      - Fractals are characterized by **self-similarity**; that is, parts of the pattern are similar to scaled copies of the whole.**自相似性**和**分形结构分形几何意味着系统在不同尺度下呈现相似的结构** 图案的某些部分类似于整个图案的缩放副本
    - **Heavy-tailed distributions**: In freezing water, the distribution of crystal sizes is characterized by a power law.
    - **Pink Noise**:Complex signals can be decomposed into their frequency components. In pink noise, low-frequency components have more power than high-frequency components.复杂信号可以分解成频率成分。在粉红噪声中，低频成分的能量大于高频成分。
- **“self-organized Criticality”** 自组织临界状态 (SOC) means that from any initial condition, the system moves toward a critical state, and stays there, without external control.从任何初始条件开始，系统都会向临界状态移动，并停留在那里，不受外部控制。
  - **Sand Pile Model**:沙堆模型 a 2-D cellular automaton where the state of each cell represents the slope of a part of a sand pile.一个二维元胞自动机，其中每个单元的状态代表沙堆某部分的坡度。
    - 1 During each time step, each cell is checked to see whether it exceeds a critical value, ⊠, which is usually 3。 在每个时间步长内，每个单元都会被检查是否超过临界值(通常为3)。
    - 2 If so, it “topples” and transfers sand to **four** neighbouring cells -- *the slope of the cell is decreased by 4, and each of the neighbours is increased by 1.如果超过，它就会“倾覆”，并将沙子转移到相邻的四个单元--该单元的斜率减小 4，而相邻单元的斜率均增大 1。*
    - 3 Each cell on the boundary has a constant slope 0, so the excess spills over the edge.边界上的每个单元格的斜率恒为 0，因此多余的沙子会溢出边界。
  - 代码理解：

What to do in the experiments:

1. All cells are initialized at a level greater than K and run the model until it stabilizes.

2. Observe the effect of small perturbations:
   - choose a cell at random
   - increment its value by 1
   - and run the model again until it stabilizes.

3. For each perturbation, measure T, the number of time steps the pile takes to stabilize, and S, the total number of cells that topple.

```
class SandPile(Cell2D):
    """Diffusion Cellular Automaton."""

    kernel = np.array([[0, 1, 0],
                       [1,-4, 1],
                       [0, 1, 0]], dtype=np.int32)

    def __init__(self, n, m=None, level=9):
        """Initializes the attributes.

        n: number of rows
        m: number of columns
        level: starting value for all cells, greater than the critical value.
        """
        m = n if m is None else m
        self.array = np.ones((n, m), dtype=np.int32) * level
        self.toppled_seq = []

    def step(self, K=3):
        """Executes one time step.

        returns: number of cells that toppled
        """
        toppling = self.array > K
        num_toppled = np.sum(toppling)
        self.toppled_seq.append(num_toppled)

        c = correlate2d(toppling, self.kernel, mode='same')
        self.array += c
        return num_toppled
```

All values in the array are initialised to level, which is generally greater than the toppling threshold, K

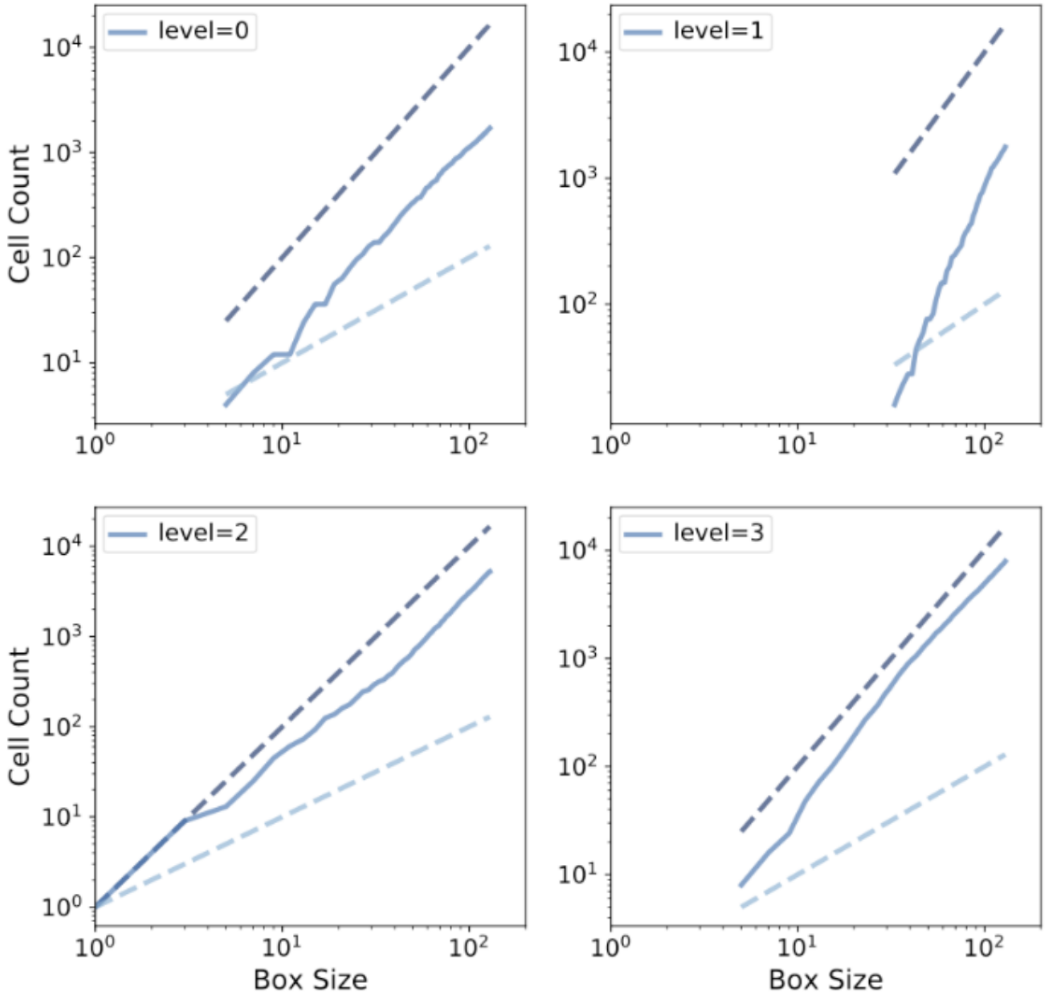
The step method finds all cells above K and topples them:

- If the sand pile model is in a critical state, we expect to find **heavy-tailed distributions** of ⊠, the number of time steps the pile takes to stabilize, and ⊠, the total number of cells that topple.在沙堆模型中，当沙堆达到临界点时，一个沙粒的增加可能会引发多个相邻沙粒的连锁坍塌，形成较大的崩塌事件。这种连锁效应就会导致事件大小分布呈现长尾特征。
- **Estimate Fractal Dimension**

- **Box counting**: count the number of cells in a small box at the centre of the pile, then see how the number of cells increases as the box gets bigger.**盒计数法：在沙堆中心统计这个小盒子内的细胞数量**，随着盒子尺寸加大，观察细胞数量是如何增加的

```
def count_cells(a):
    n, m = a.shape
    end = min(n, m)

    res = []
    for i in range(1, end, 2):
        top = (n-i) // 2
        left = (m-i) // 2
        box = a[top:top+i, left:left+i]
        total = np.sum(box)
        res.append((i, i**2, total))

    return np.transpose(res)
```

  - The size of the box is initially 1. Each time through the loop, it increases by 2;
  - • Each time through the loop, box is a set of cells with width and height ⬚, centred in the array;
  - • Sum the total number of “on” cells each time.
  - • The result is a list of tuples, where each tuple contains ⬚, ⬚ ⬚⬚ 2, and the number of cells in the box.
- If we draw ⬚ = ⬚2 and ⬚ = ⬚ on log-log scale, we will have the slope of 2 and 1, respectively.



- The estimated slope for level 0, 1, 2 and 3 are around 1.871, 3.502, 1.781 and 2.084, respectively.
- This suggests that level 0, 1, 2 have a “fractional dimension”, that is, it is a fractal.

- **pink noise 也称为1/f噪声，是一种在自然界和各种系统中广泛存在的噪声类型。它的特点是随着频率的增加，信号的强度（功率）会逐渐减弱。**
  - Any signal can be decomposed into a set of frequency components with different levels of power. 任何信号都可以被分解为一系列不同强度的频率组成
  - The power spectrum of a signal is a function that shows the power of each frequency component.
  - 𝑃(𝑓) versus 𝑓 on a log-log scale, we expect a straight line with slope −𝛽.

    Relationships between frequency and power can be written as:

    $$\log P(f) = -\beta \log f$$

  - When 𝛽 = 0, this is describing the white noise; when 𝛽 = 2, this is describing the red noise; when 0 < 𝛽 < 2, it is the pink noise.

- Chapter Seven
  - **Agent-based Modelling (ABM)**:

- Agent-based Model (ABM) is a computational model for simulating the actions and interactions of autonomous agents自主代理 (both individual or collective entities such as organisations or groups) to understand the behaviour of a system and what governs its outcomes. ABMs are stochastic and include randomness. ABM 是包含随机性的随机模型
- The three main components of ABMs are:
  - **Agent**: Autonomous, can be homogeneous or heterogeneous.
    - Cells in Cellular Automata can be considered as the representation of agents without spatial movement.
    - What makes ABM different from CA is that ABM allows the agent to roam in spaces whereas in Cellular Automata, the cell (agent) remains in a fixed spatial coordinate.
  - **Environment**: The environment is the encompassing system in which the agents exist. 环境是代理人所处的包罗万象的系统。 Defines the interaction space, either spatial (2D grid) or abstract.
  - **Interaction**:
    - Agent-to-environment： the Agent influences the Environment by changing the state of the cell in the Environment. Likewise, the changing state of environment influences the agent.改变自身状态影响环境
    - Agent-to-agent： query or obtaining information for the agent from other agents holds the most basic interaction in ABM.查询或从其他代理那里为代理获取信息是 ABM 中最基本的互动。
- **Schelling's Segregation Model**:
  - Simulates racial segregation based on agent satisfaction in neighborhoods. Unhappy agents move to empty cells, leading to emergent segregation even without extreme preferences.

```
num_empty = np.sum(empty)
for source in unhappy_locs:
    i = np.random.randint(num_empty)
    dest = empty_locs[i]

    a[dest] = a[source]
    a[source] = 0
    empty_locs[i] = source
```

> `i` is the index of a random empty cell; `dest` is a tuple containing the coordinates of the empty cell.
>
> In order to move an agent, we copy its value (1 or 2) from `source` to `dest`, and then set the value of `source` to 0 (since it is now empty).
>
> Finally, we replace the entry in `empty_locs` with `source`, so the cell that just became empty can be chosen by the next agent.

- 步骤

> **What pieces of machinery do we need?**
>
> - Initialise a grid randomly with 3 possible states
> - Determine neighbourhood situation for each cell
> - Assess whether happy or unhappy by comparing neighbourhood ratio with some defined model parameter
> - Find unhappy agents
> - Find empty locations
> - Select (randomly) an unhappy agent and move it to a randomly selected empty location.
> - Update neighbourhood, agent states, empty locations
> - Repeat the process

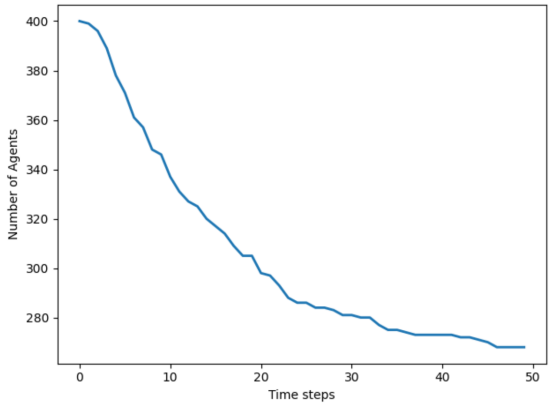
- conclude：
  - This is a very good demonstration of the unpredictable relationship between individual decisions and system behaviour. 很好地证明了个体决策和系统行为之间的不可预测的关系。
  - Schelling's model demonstrates a possible cause of segregation but says nothing about actual causes.

- **Sugarscape Model**:
  - Developed by Epstein and Axtell, it simulates an "artificial society" where agents gather sugar (economic wealth) from a 2D grid. The model explores wealth distribution and inequality, with agents differing in metabolism, vision, and lifespan.
  - **Agents move around on a 2-D grid, harvesting and accumulating "sugar",** which represents economic wealth. Some parts of the grid produce more sugar than others, and some agents are better at finding it than others.一部分格子会产生更多的糖，有人会比其他人更好地找到他们。
  - attributes：

- **Sugar:** how much sugar they start with;
- **Metabolism**: 新陈代谢 how much sugar they consume per time step;
- **Vision:** how far they can see (radius of ⊠).

- rules:
  - 1 The agent surveys ⊠ cells in each of the 4 compass directions, where ⊠ is the range of the agent's vision **视野范围内的资源勘探**
  - 2 It chooses the unoccupied cell with the most sugar. In case of a tie, it chooses the closer cell; among cells at the same distance, it chooses randomly.**选择目标格子**：未被占据含糖量高，距离近
  - 3 The agent moves to the selected cell and harvests the sugar, adding the harvest to its accumulated wealth and leaving the cell empty.**移动和采集**：
  - 4 The agent consumes some part of its wealth, depending on its metabolism. If the resulting total is negative, the agent "starves" and is removed.**消耗财富**
  - **Some other factors can be involved into the model?**

    We can watch the evolution of our simulation and observe snapshots in time:

    - Agents born in the areas with the least sugar are likely to starve unless they have a high initial endowment and high vision.
    - The population will drop because when sugar grows back at 1 unit per time step, there is not enough sugar to sustain the 400 agents we started with.

  - Agent's age can increase at each time step
  - Agents have finite lifespan代理的寿命是有限的
  - New agent can be added to the environment if an existing agent died, to make the population unchanged

- **Emergence**: 由agent之间的交互产生，很难基于个体行为预测。
  - Emergent properties arise from agent interactions and are difficult to predict from individual behavior, as seen in both the Schelling and Sugarscape models.
  - The segregation we see in Schelling's model is an emergent property because it is not caused by racist agents. Even when the agents are only mildly preference, the outcome of the system is substantially different from the intention of the agent's decisions.但即使它们只有轻微的

偏好（例如希望一部分邻居和自己相同），整个系统最终也会表现出显著的和代理不同的意图。

- Sugarscape model with all agents start in the lower-left corner of the grid shows the outcome that — groups or “aggregates” with properties and behaviours that the agents don’t have.在 Sugarscape 模型中， 如果所有代理都从网格的左下角开始移动，最终会在模型中形成各种**群体（aggregates）**或聚集现象。这些群体拥有一些代理本身不具备的特性和行为。

- Chapter Eight
  - Focuses on **Agent-based Modelling (ABM)**, with key topics including traffic jam simulations and the Boids model for simulating flocking behavior.
  - **Traffic Jam Simulation**:
    - The lecture discusses how to model traffic jams using ABM in a 1-D space (highway). Key components include Drivers in the cars (agents)司机, distances between them距离, speed limits限速, and randomness in acceleration随机加速度. Simulations demonstrate how even small variations in speed can lead to crashes and traffic jams.

      What pieces of machinery do we need?

      - Initialize 1D array with equal distance between cars
      - Determine the speed limit on this highway
      - Determine the next car for each car
      - Setup the initial speed for each car
      - Cars will move in order on this highway
      - For the movement of each car:
        - Determine the distance that each car can move
        - Choose the acceleration rate for each driver
        - Compute the speed of each car (with randomness involved)
        - Update the new location for each car
      - If speed > distance from next car, crash happens, and the car will stop with speed set to zero

    - **Randomness in this model:**（reflect the real world)
      - With this random noise, even if every driver’s acceleration rate is the same, the eventual speed would be different, and crash might occur, causing some cars to stop.

        ```
        # add random noise to speed
        speed *= np.random.uniform(1-self.eps, 1+self.eps)
        ```

        A random noise is added to the speed computation at each step of driver’s movement to reflect the imperfect speed up in real world.

- **Boids Model (Flocking Simulation)**:
  - The **Boids model** simulates the behavior of birds flocking. Each agent (called a boid) follows three key behaviors:
    - **Flock centering**:鸟群居中 Moving towards the center of the group.向羊群中心移动。
    - **Collision avoidance**:避免碰撞 Avoiding other boids and obstacles.避开障碍物，包括其他boids。
    - **Velocity matching**: 速度匹配Aligning speed and direction with neighbors.使速度（速度和方向）与邻近的boids一致。
    - 如何实施模型

      ### What pieces of machinery do we need?

      - Initialize number of boids with random velocity and random positions.
      - Boids movement rules:
        - **Find center**: Finds other Boids within range (radius and angle of the field of view) and computes a vector toward their centroid.
        - **Avoid collision**: Finds objects, including other Boids (and carrots), within a given range, and computes a vector that points away from their centroid.
        - **Align velocity**: finds other Boids within range and computes the average of their headings.
        - **Love (find carrot)**: computes a vector that points toward the carrot.

```python
def get_neighbors(self, boids, radius, angle):
    """Return a list of neighbors within a field of view.

    boids: list of boids
    radius: field of view radius
    angle: field of view angle in radians

    returns: list of Boid
    """
    neighbors = []
    for boid in boids:
        if boid is self:
            continue
        offset = boid.pos - self.pos

        # if not in range, skip it
        if offset.mag > radius:
            continue

        # if not within viewing angle, skip it
        diff = self.vel.diff_angle(offset)
        if abs(diff) > angle:
            continue

        # otherwise add it to the list
        neighbors.append(boid)

    return neighbors
```

- 1. **Find center（寻找中心）**：Boids 寻找在自己视野范围内的其他 Boids，并计算一个指向它们中心的向量。这使得 Boids 会趋向于聚集在一起，形成一个群体。
- 2. **Avoid collision（避免碰撞）**：Boids 会识别一定范围内的障碍物，包括其他 Boids 和“胡萝卜”。它们会计算一个远离这些障碍物的向量，以避免发生碰撞。
- 3. **Align velocity（对齐速度）**：Boids 会与视野范围内的其他 Boids 对齐，调整自己的速度和方向，以便与群体的平均速度保持一致。
- 4. **Love (find carrot)**：这是一个额外的行为，Boids 会计算一个指向“胡萝卜”的向量，表示被“胡萝卜”吸引。这意味着 Boids 会试图朝着“胡萝卜”的方向移动。
- Only boids within the range (in the neighbourhood) will be considered for the movement.
- **center** uses get_neighbors to get a list of Boid objects that are in the field of view. vecs is a list of Vector objects that represent their positions
- align is also similar to center; the big difference is that it computes the **average of the neighbors' velocities**, rather than their positions. If the neighbors point in a particular direction, the Boid tends to steer toward that direction

```
def align(self, boids, radius=0.5, angle=1):
    """Return the average heading of other boids in range.

    boids: list of Boids
    """
    neighbors = self.get_neighbors(boids, radius, angle)
    vecs = [boid.vel for boid in neighbors]
    return self.vector_toward_center(vecs)


def love(self, carrot):
    """Returns a vector pointing toward the carrot."""
    toward = carrot.pos - self.pos
    return limit_vector(toward)
```

- The parameter *mu* determines h**ow quickly the birds can change speed and direction.**

```
def move(self, mu=0.1, dt=0.1):
    """Update the velocity, position and axis vectors.

    mu: how fast the boids can turn (maneuverability).
    dt: time step
    """

    self.vel = (1-mu) * self.vel + mu * self.goal
    self.vel.mag = 1
    self.pos += dt * self.vel
    self.axis = self.length * self.vel
```

- The model includes parameters such as **neighborhood radius, angle and weight for each behavio**r, maneuverability,

- **Emergent Behavior**:
  - Emergent behaviors, such as flock formation, arise from the simple rules each agent follows. The lecture emphasizes how agent interactions lead to complex and unpredictable outcomes.

- Chapter Ten
  - **Theory of Evolution**:
    - Evolution is driven by natural selection, where genetic variations affect an organism's ability to survive and reproduce.进化是由自然选择驱动的，遗传变异会影响生物的生存和繁殖能力。
    - The process leads to adaptation, increasing diversity, and complexity over generations.在这个过程中，人们会一代代地适应环境，增加多样性和复杂性。
  - **Simulating Evolution**: 模拟进化
    - Key features include replicators (a population of agents that can reproduce), variability (differences between agents), and differential survival or reproduction(differences affecting survival or reproduce chances).复制者（可以复制的媒介）、变异性（媒介之间的差异）以及不同的生存或复制（影响生存机会的差异）。
    - machinery机制：

      The basic machinery we may need:

      - Agents have genetic information, called their "genotype".
      - A fitness value is assigned to each genotype to specify their adaptation to the current environment.
      - Some agents would die over the generations and some agents are born.

    - 每个agent都有一个基因信息，叫做基因型。
    - 为每个基因型分配一个适应度值，以明确其对当前环境的适应性。
    - 有些agent会死经过一代之后，有的会出生。
  - Agents have genetic information (genotype) and a fitness value representing their adaptation to the environment.
- **Modelling decisions to make:**
  - How to represent the genotype?
    - **Genotype**: the genetic information of each agent that to be copied when the agent replicates.
    - A genotype can be represented by a sequence of 𝑘 binary digits (0 or 1), where 𝑘 is a parameter that we can choose.
  - How to setup the fitness value? How to relate it to different genotypes?
    - **Fitness**: a quantity related to the ability of an agent to survive or reproduce.
    - how to relate to genotypes - **Fitness landscape**: a function that maps genotype to fitness.
    - This fitness is the "height" of the landscape. Genotypes which are similar are said to be "close" to each other, while those that are very different are "far" from each other.这种适应性就是景观的“高度”。相似的基因型被称为相互“接近”，而差异很大的基因型则相互“远离”。
    - Each genotype corresponds to a location on the fitness landscape.每种基因型都对应着适合度景观上的一个位置。
    - If all genotypes have the same replication rate, on the other hand, a fitness landscape is said to be flat.当所有基因型的**复制率**（replication rate）相同，即没有基因型在适应性上有任何优势，那么适应性地形就是**平坦的**。
  - Which agents will die over generations? How to choose them?

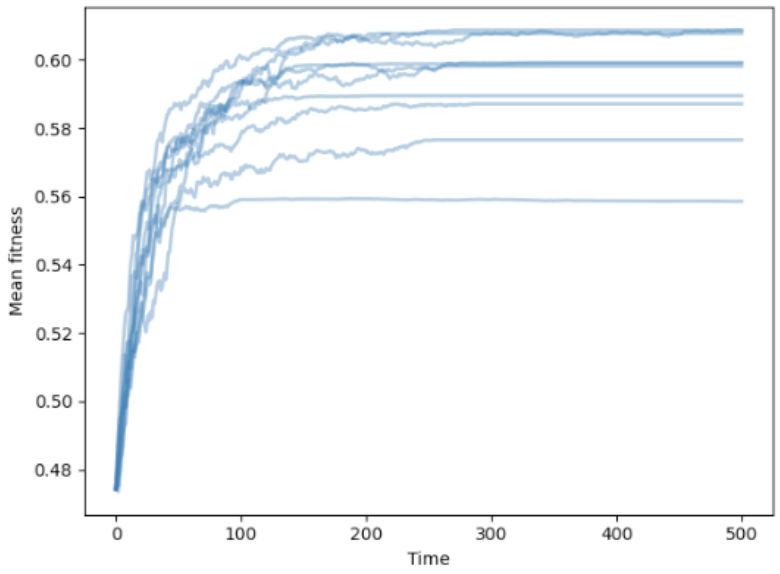
```
def choose_dead(self, ps):
    """Choose which agents die in the next timestep.

    ps: probability of survival for each agent

    returns: indices of the chosen ones
    """
    n = len(self.agents)
    is_dead = np.random.random(n) < 0.1
    # np.nonzero: Return the indices of the elements that are non-zero.
    # which are True elements in is_dead array
    index_dead = np.nonzero(is_dead)[0]
    return index_dead
```

  - How many agents will be reproduced? Which agents will be reproduced?
- **Simulation Approach**（步骤）
  - 1 get the array containing the fitness of each agent; 获取包含每个代理的适应度的数组；

- 2 decides which agents to die;决定哪些代理死亡
- 3 decides which agents reproduce 决定哪些代理繁殖

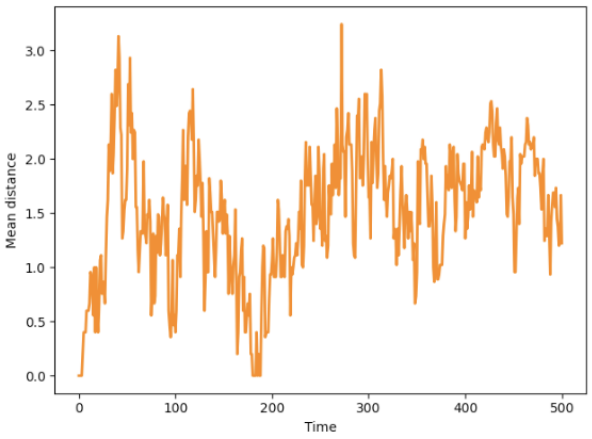
- **Evidence of Evolution** 进化的证据
  - 进化最广泛的定义是种群中基因型分布的变化。进化是一种总体效应：换句话说，个体不会进化，种群才会进化。
  - The most inclusive definition of evolution is a change in the distribution of **genotypes** in a population. Evolution is an aggregate effect: in other words, individuals don't evolve; populations do.
  - 如果基因型发生变化，我们预计它们的适应性也会发生变化。因此，我们将把适应性分布的变化作为进化的证据。If the genotypes change, we expect their fitness to change as well. So, we will use changes in the distribution of fitness as evidence of evolution.
  - 改进模型：
    - Agents with low fitness are more likely to die, agents with high fitness are more likely to survive long enough to reproduce.

```
def choose_dead(self, fits):
    n = len(self.agents)
    is_dead = np.random.random(n) > fits
    index_dead = np.nonzero(is_dead)[0]
    return index_dead
```



    - *increasing fitness means that the species is getting better at surviving in its environment.*
    - 多样性的证明：
      - Since locations are represented with arrays of bits, we'll define distance as the number of bits that differ between locations. **To quantify the dispersion of a population, we can compute the mean of the distances between pairs of agents.为了度量个体之间的基因差异，我们可以将不同的比特数量定义为个体之间的"距离"。为了量化种群的分散程度，我们可以计算所有个体对之间的距离的平均值**

Quantify Speciation



N=8, start with 100 agents at the same location, *with differential survival and mutation.*

As mutations occur, mean distance increases, reaching a maximum while the population migrates across the landscape.

- N=8 表示基因型的长度（即比特位数）
- As mutations occur, mean distance increases, reaching a maximum while the population migrates across the landscape.
- 在某个时间点，平均距离达到一个最大值，意味着基因多样性达到了顶峰。这可以解释为种群在基因空间中达到了最大的分散程度。

- **Speciation and Environmental Change**:（待补充）
  - Speciation occurs when genetic clusters form due to mutations and differential reproduction.

- Chapter Eleven
  - **Prisoner's Dilemma**:囚徒困境
    - A game theory scenario where **two individuals** choose to cooperate or defect, with rewards or punishments depending on their choices.在博弈论情景中，**两个人**选择合作还是叛变，奖惩取决于他们的选择。
    - Rational agents would defect to minimize their penalty, but real-life experiments show that people often cooperate more than expected.
  - **The Problem of Altruism**: 适应性的
    - Altruism appears counterintuitive from an evolutionary perspective, as selfish individuals seem more likely to survive. However, altruism may still be adaptive, enhancing survival and reproduction.
  - **Simulating Evolution of Cooperation**:
    - The Prisoner's Dilemma can be used to simulate cooperation strategies in populations over time.
    - The simulations can encode strategies as genotypes and introduce mutations to explore how cooperative traits evolve.
  - **Tournament Simulations**:

- The Tournament class models repeated games, tracking agents' choices and updating their scores. TFT以牙还牙
- TFT always cooperates during the first round of an iterated match; after that, it copies whatever the opponent did during the previous round.
    - **Properties of Good Strategies:**
        - **Nice**: 友善The strategies that do well cooperate during the first round, and generally cooperate as often as they defect in subsequent rounds.策略会在第一轮中进行合作，而且在随后的几轮中，合作的频率通常与叛变的频率相当。
        - **Retaliating**: 报复性Strategies that cooperate all the time did not do as well as strategies that retaliate if the opponent defects.如果对手叛变，则采取报复策略。
        - **Forgiving**: But strategies that were too vindictive tended to punish themselves as well as their opponents.意味着它在对手恢复合作后也会选择合作，避免无限循环的背叛
        - **Non-envious**: 不妒忌Some of the most successful strategies seldom outscore their opponents; they are successful because they do well enough against a wide variety of opponents.
- **Encode a PD strategy as a genotype**

```python
class Agent:

    keys = [(None, None),
            (None, 'C'),
            (None, 'D'),
            ('C', 'C'),
            ('C', 'D'),
            ('D', 'C'),
            ('D', 'D')]

    def __init__(self, values, fitness=np.nan):
        """Initialize the agent.

        values: sequence of 'C' and 'D'
        """
        self.values = values
        self.responses = dict(zip(self.keys, values))
        self.fitness = fitness
```

TFT

C
C
D
C
D
C
D

Map the agent's choice in each round to the opponent's choice in the previous two rounds.

The genotype of a strategy is a sequence of choice with each item corresponds to the opponent's different choice.

- The genotype of a strategy is a sequence of choice with each item corresponds to the opponent's different choice.策略的基因型是一个选择序列，每个项目都对应着对手的不同选择。
- **Agents can copy with a probability of mutation**

Mutation works by choosing a random value in the genotype and flipping from 'C' to 'D' or vice versa.

```
def mutate(self):
    """Makes a copy of this agent's values, with one mutation.

    returns: sequence of 'C' and 'D'
    """
    values = list(self.values)
    index = np.random.choice(len(values))
    values[index] = 'C' if values[index] == 'D' else 'D'
    return values
```

The Tournament class encapsulates the details of the PD competition:

```
class Tournament:

    payoffs = {('C', 'C'): (3, 3),
               ('C', 'D'): (0, 5),
               ('D', 'C'): (5, 0),
               ('D', 'D'): (1, 1)}

    num_rounds = 6

    def play(self, agent1, agent2):
        """Play a sequence of iterated PD rounds.

        agent1: Agent
        agent2: Agent

        returns: tuple of agent1's score, agent2's score
        """
        agent1.reset()
        agent2.reset()
        for i in range(self.num_rounds):
            resp1 = agent1.respond(agent2)
            resp2 = agent2.respond(agent1)

            pay1, pay2 = self.payoffs[resp1, resp2]

            agent1.append(resp1, pay1)
            agent2.append(resp2, pay2)

        return agent1.score, agent2.score
```

Asks each agent for their response, given the opponent's previous responses.

```
keys = [(None, None),
        (None, 'C'),
        (None, 'D'),
        ('C', 'C'),
        ('C', 'D'),
        ('D', 'C'),
        ('D', 'D')]
```

```
def past_responses(self, num=2):
    """Select the given number of most recent responses.

    num: integer number of responses

    returns: sequence of 'C' and 'D'
    """
    return tuple(self.hist[-num:])

def respond(self, other):
    """Choose a response based on the opponent's recent re

    other: Agent

    returns: 'C' or 'D'
    """
    key = other.past_responses()
    resp = self.responses[key]
    return resp
```

- 以牙还牙的特点，模仿对手上两次的回应 It uses the following methods from the Agent class:
  - **reset:** Initializes the agents before the first round, resetting their scores and the history of their responses.
  - **respond**: Asks each agent for their response, given the opponent's previous responses.
  - **append:** Updates each agent by storing the choices and adding up the scores from successive rounds
- Over time, populations of defectors may be invaded by cooperative strategies, leading to higher average fitness.

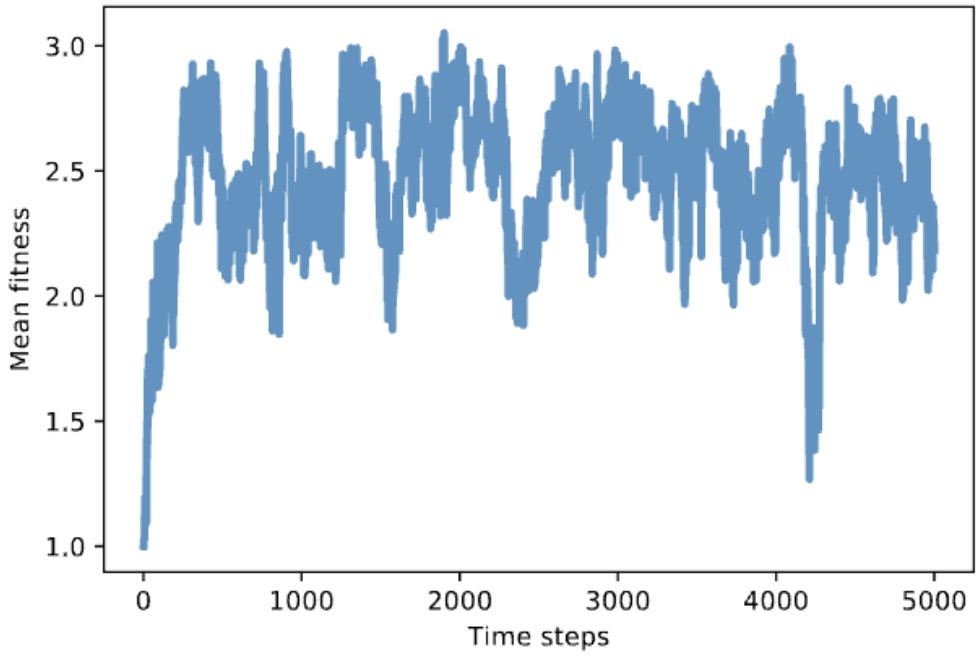
- **Results**:
  - The simulations show that niceness (cooperation) tends to increase, and a balance between cooperation and retaliation can help sustain high levels of cooperation.模拟结果表明，友好（合作）倾向于增加，而合作与报复之间的**平衡**有助于维持高水平的合作。

Conclusion

The agents in these simulations are simple, and the Prisoner's Dilemma is a highly abstract model of a limited range of social interactions.

- Populations of defectors are vulnerable to invasion by nicer strategies.
- The average amount of niceness is generally high.
- The average level of fitness is generally closer to a utopia of cooperation.
- Some degree of retaliation may be adaptive, but it might not be necessary for all agents to retaliate.
- If there is enough retaliation in the population as a whole, that might be enough to prevent invasion by defectors.

- 1 叛逃者的种群很容易受到较好策略的入侵。
- 2 友好的平均数量普遍较高
- 3 适应度的平均值普遍接近合作的乌托邦
- 4 一定程度的报复可能是适应性的，但可能并不需要所有代理人都进行报复。
- 5 如果整个群体有足够多的报复行为，就可能足以防止叛逃者的入侵。
- **Quantify the fitness** 适应度的量化结果



- 1. **初始低平均适应度（约 1）：**
  - 在模拟的初期，所有代理人都选择背叛，因为没有其他策略。在囚徒困境的规则下，当两个背叛者相遇时，每个背叛者只能获得 1 分。因此，初始时的平均适应度约为 1 分。

- 2. **适应度上升至接近 3：**
  - 随着时间的推移，平均适应度逐渐上升到接近 3。原因可能是某些代理人改变了策略或出现了某种合作趋势，这使得他们在相互合作的回合中获得更高的分数。
- 3. **适应度的波动：**
  - 图中可以看到适应度的变化并不是平稳的，而是有较大波动。这可能是因为代理人之间的策略选择偶尔回归背叛导致适应度下降，也可能是因为一些合作的代理人被淘汰或重新背叛，导致适应度不稳定。
-

图 2：友善度（Niceness）

- 图 2 左侧显示了友善度（Niceness）在整个种群中的变化趋势。友善度衡量了基因型中表现出合作行为的比例。
- 在早期阶段，友善度迅速上升至接近 0.75，然后在 0.4 到 0.85 之间波动。这表明，尽管个体间合作的程度存在波动，但整体上维持在较高的水平。
- 这种波动反映了合作和背叛策略之间的动态平衡。在某些时间步长，合作策略可能会更占优势，而在其他时间步长，背叛策略会有所回升。

图 3：开放性（Opening）

- 图 2 右侧的图展示了种群中在第一轮合作的个体比例随时间步长的变化。开放性表示在每一轮游戏开始时选择合作的比例。
- 图中显示，开放性在 0.2 到 1.0 之间剧烈波动。这表明，个体在第一轮中选择合作的比例波动较大，可能受到之前游戏轮次中对方策略的影响。
- 尽管存在波动，总体上仍有较高比例的个体在第一轮选择合作，这说明策略演化过程中保留了较多的友好行为。