1. 整体设计和思路

受到捕食者被捕食者模型（logistic-volterra模型）的启发，我们设计了如下程序，用来模拟生态系统中一条简单的食物链——草-食草动物-食肉动物，并发现捕食者和被捕食者的数量在一稳定值附近可实现周期性变化。

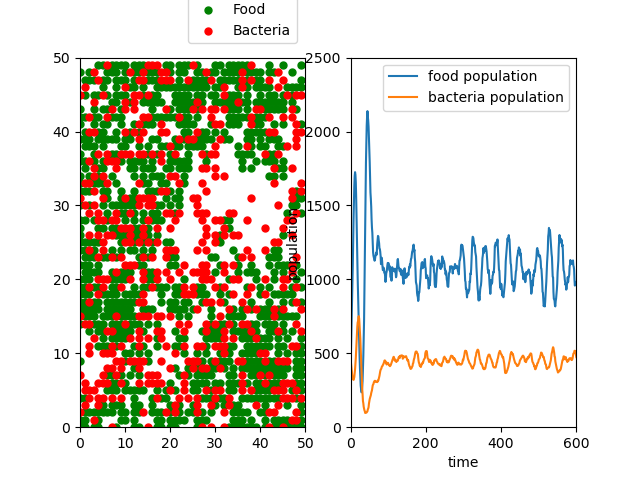
在最初的代码编写时，我们设计了随机产生的不可移动和繁殖的食物和可自由移动能够和繁殖的细菌，也能够得到类似的图像和结果，但由于在实际应用中难以解释，于是加入了草(grass)的设定。在这个模型里，每一个格子代表的是单位数量的种群还是独立的个体有待商榷。

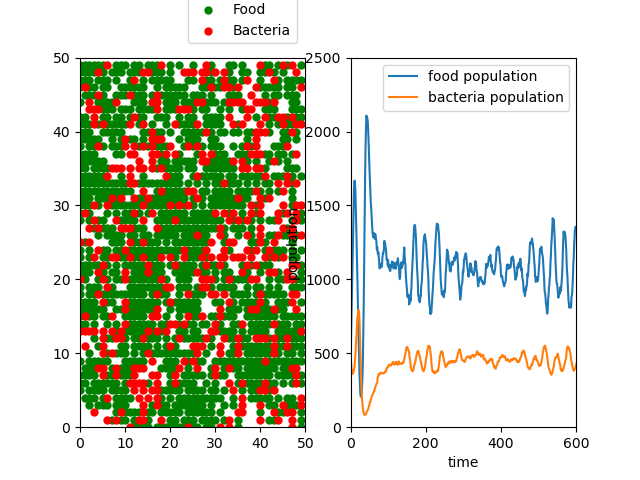
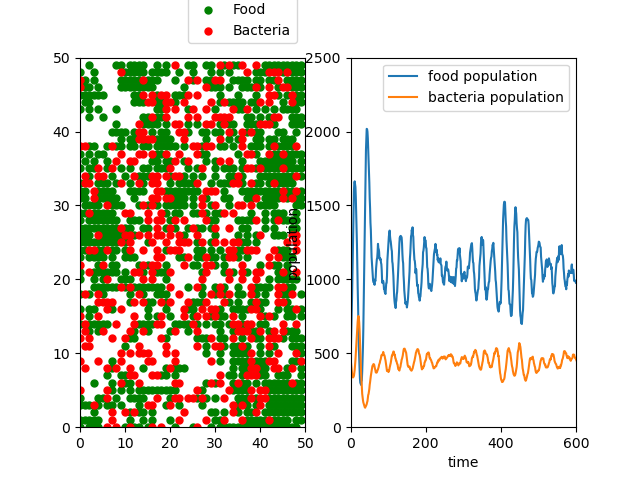
这是一个极为简化的模型，在一块可自定义设置大小的样方地中，草：有生长能力且生长不受周围环境限制，有食草动物存在时减少。食草动物：自由移动，饥饿会导致死亡，有寿命，在不饥饿时繁殖。食肉动物：自由移动，饥饿会导致死亡，有寿命，在不饥饿时繁殖

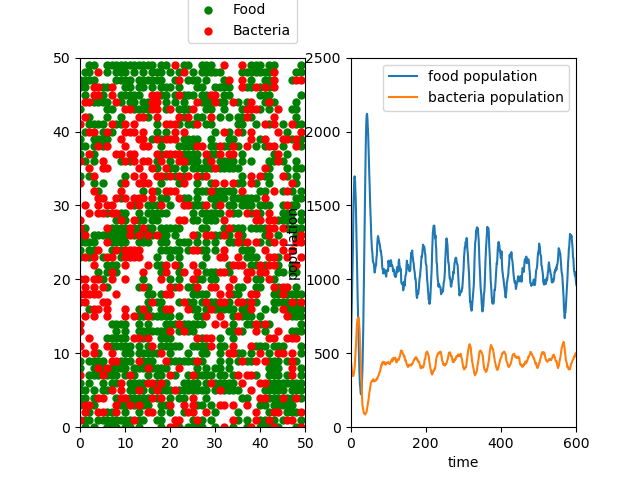
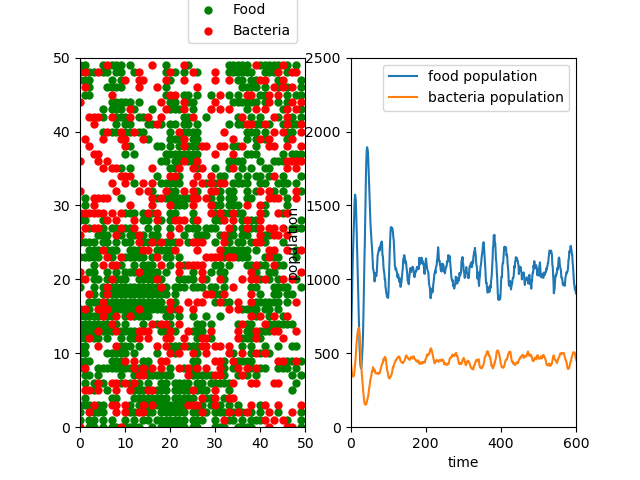
1. 主代码代码分析

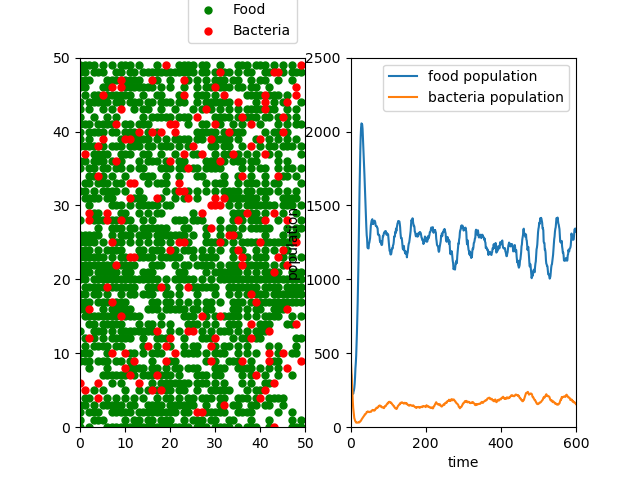
import numpy as np  
import matplotlib.pylab as plt  
import pickle  
  
  
class Bacteria():  
 *"""  
 bacteria类  
 可以认为这是对会繁殖、会运动、有寿命、会饿死的捕食者的模拟  
 """* \_\_num = 0 *# 记录Bacteria类中实例数量* rp\_prob = 0.9 *# 记录Bacteria实例达到分裂条件时的分裂概率* def \_\_init\_\_(self,lifetime=40,hunger=3,tig=5):  
 self.lifetime = lifetime *# bacteria寿命* self.hunger = hunger *# 饥饿值，值越低而越饥饿，饥饿值为1而仍未找到食物则会死亡* self.tig = tig *# 繁殖的触发条件，只有在饥饿值高于此值的时候才能繁殖* Bacteria.\_\_num += 1 *# 当创建实例时，记录数量的\_\_num加一* def \_\_del\_\_(self):  
 *"""  
 定义了实例被删除时的行为  
 """* if Bacteria.\_\_num >= 0:  
 Bacteria.\_\_num -= 1 *# 被删除时\_\_num减一* else:  
 raise ValueError(**"Bacteria.\_\_del\_\_()类中没有实例"**)  
  
 def can\_divide(self):  
 return self.hunger > self.tig*# 表示捕食者能够进行繁殖的最低饥饿值条件*  
  
 @classmethod  
 def get\_num(cls):  
 return Bacteria.\_\_num  
  
 @classmethod  
 def set\_rp\_prob(cls, prob=0.9):  
 *"""设置满足条件时bacteria繁殖的概率，此处0.9为预设值"""* Bacteria.rp\_prob = prob  
  
  
class Food():  
 *"""  
 food类  
 考虑food是一种会运动、会繁殖的被捕食者，会吃草，有饥饿值  
 """* \_\_num = 0 *# 记录Food实例数量* rp\_prob = 0.6 *# 记录Food类实例的繁殖概率* def \_\_init\_\_(self, lifetime=20, hunger=5, tig=1):  
 Food.\_\_num += 1  
 self.hunger = hunger  
 self.tig = tig *# food繁殖的触发条件，一般考虑food繁殖能力强，此值默认比较小* self.lifetime = lifetime *# Food实例的寿命* def \_\_del\_\_(self):  
 if Food.\_\_num >= 0:  
 Food.\_\_num -= 1  
 else:  
 raise ValueError(**"Food.\_\_del\_\_()类中没有实例"**)  
  
 def can\_divide(self):  
 return self.hunger > self.tig*# 表示被捕食者能够进行繁殖的最低饥饿值条件*  
  
 @classmethod  
 def get\_num(cls):  
 return Food.\_\_num  
  
 @classmethod  
 def set\_rp\_prob(cls, prob=0.6):  
 Food.rp\_prob = prob*"""设置满足条件时food繁殖的概率，此处0.6为预设值"""*  
class Field():  
 *"""  
 表示一块可供bacteria活动的区域，要求此类有且仅有一个实例  
 """  
 # 以下参数的设置都在类方法中完成* \_\_grass\_growth\_list = [] *# 记录“草”的增长模式，二维列表，每一格存储“草”的增长速率  
 # 希望可以设计出除了平均分布以外的分布* \_\_average\_rate = 0 *# 记录平均增长时增长速率* size = 0  
 \_\_food\_num = [] *# 记录每次更新之后food的数量* \_\_bacteria\_num = [] *# 记录每次更新之后bacteria的数量* \_\_direction = {  
 0: (0, 1),  
 1: (-1, 1),  
 2: (-1, 0),  
 3: (-1, -1),  
 4: (0, -1),  
 5: (1, -1),  
 6: (1, 0),  
 7: (1, 1)  
 }*# food 和 bacteria可能的移动方向，不包括留在原地的选项，因此后续代码要注意加上这种可能性*  
 def \_\_init\_\_(self, n, mode=**"plain"**, grass\_i=5):  
 *"""  
 n: 区域大小，表示这是一个(n, n)的区域  
  
 考虑了food和bacteria都有运动能力并且可以繁殖  
 food要依靠吃“草”来生活，预设每格草的量为5  
 在初始化时更新所有的类变量  
  
 期待着通过mode这个参数调节food的分布规律  
 """* Field.clear\_bacteria\_num()  
 Field.clear\_food\_num()  
 Field.set\_average\_rate()  
 self.\_\_size = n  
 Field.size = n  
 self.\_\_state = [[None for i in range(n)] for i in range(n)] *# 初始化构造空区域  
 # 需要调用set\_food与set\_bacteria向区域中加入food和bacteria* if mode == **"plain"**:  
 *# 设置草的分布规律，草另行储存* Field.plain\_mode() *# 这个构造了草的增长速率列表* self.\_\_grass = [[grass\_i for i in range(n)] for j in range(n)]  
 *# grass是food的food，此属性储存了当前区域grass数量分布* def \_\_str\_\_(self):  
 *"""  
 调试的时候用一用  
 """* return str(self.transform())  
  
 def set\_bacteria(self, coordinate=(0, 0)):  
 *"""  
 设置bacteria初始位置，每次执行只能设置一个bacteria的位置  
 """* i, j = coordinate  
 if i >= self.\_\_size or j >= self.\_\_size:  
 raise IndexError(**"Field.set\_bacteria()设置位置超过了区域大小"**)  
  
 old = self.\_\_state[i][j]  
 self.\_\_state[i][j] = Bacteria()  
 del old  
  
 def set\_food(self, coordinate):  
 *"""  
 设置food初始位置，每次执行只能设置一个food位置  
 """* i, j = coordinate  
 if i >= self.\_\_size or j >= self.\_\_size:  
 raise IndexError(**"Field.set\_food()设置位置超过了区域大小"**)  
 old = self.\_\_state[i][j]  
 self.\_\_state[i][j] = Food()  
 del old  
  
 def evolution(self, coordinate):  
 *"""  
 选取区域中的一个位置coordinate，分析这里可能发生的情况  
 分为三类，food、bacteria、空地  
 注意只要是food没有吃grass，grass就会增加  
 """* i, j = coordinate[0], coordinate[1]  
 if i >= self.\_\_size or j >= self.\_\_size:  
 raise IndexError(**"Field.evolution()选取位置超过区域大小"**)  
  
 old = self.\_\_state[i][j] *# 取出i, j位置的对象的引用* if isinstance(old, Food): *# 如果是food* empty\_pos = [] *# 储存附近有grass的空地，表示这是food可以去的空地* food\_pos = [] *# 储存附近的food* for d in range(8):  
 direction = Field.\_\_direction[d]  
 position = i + direction[0], j + direction[1] *# 周围所有的八个位置*  
 if not Field.out\_of\_range(Field.size, position):  
 sur = self.\_\_state[position[0]][position[1]]  
 if sur is None:  
 if self.\_\_grass[position[0]][position[1]] > 0:  
 empty\_pos.append(position) *# 如果这个位置是有grass且没有food和bacteria的，则作为一个可去的位置存储下来*  
 elif isinstance(sur, Food):  
 food\_pos.append(position)  
 if self.\_\_grass[i][j] > 0:  
 empty\_pos.append(coordinate) *# food有可能不动* def move\_food(old):  
 *"""描述food在不分裂时的行为，假设food所在的地方grass不生长  
 同时假设food只会向有grass的地方前进"""* old.lifetime -= 1 *# 寿命减1* if empty\_pos == []: *# 表示周围所有地方的草都没有了，待在原地* old.hunger -= 1 *# 不考虑此位置草的生长* else:  
 index = np.random.randint(len(empty\_pos)) *# 随机选取一个可行的方向* nxt\_pos = empty\_pos[index] *# 表示food下一步前进的位置* self.\_\_state[i][j] = None  
 self.\_\_state[nxt\_pos[0]][nxt\_pos[1]] = old  
 self.\_\_grass[nxt\_pos[0]][  
 nxt\_pos[1]] -= 1 *# 表示前去的那个地方grass被吃掉1* old.hunger += 1  
  
 if old.lifetime == 1:  
 self.\_\_state[i][j] = None  
 del old *# 删除food的引用，Food类中记录实例数量的变量减1* self.grass\_grow(coordinate) *# 没有food后，grass会生长* elif old.hunger == 1: *# 如果饥饿值为1，要吃grass* if empty\_pos == []: *# 表示周围没有空位，此时被饿死* self.\_\_state[i][j] = None  
 del old  
 else:  
 move\_food(old)  
 elif old.hunger > 1: *# 饥饿值大于1，考虑分裂或者向其他方向运动，但都会吃grass* if old.hunger > old.tig: *# 说明达到了分裂条件，有概率分裂* positions = empty\_pos[:-1] *# 除了当前位置的有grass的所有相邻位置* if len(positions) == 0: *# 没有空间支持food繁殖，则不分裂，可能向有grass的位置移动* move\_food(old)  
 else:  
 r = np.random.random\_sample()  
 if r < Food.rp\_prob: *# 此时表示可以繁殖，向周围任意有grass空地繁殖而自己本身不移动* old.hunger -= 1  
 idx = np.random.randint(len(positions))  
 new\_pos = positions[idx]  
 self.\_\_state[new\_pos[0]][new\_pos[1]] = Food(  
 hunger=1) *# 繁殖结果为一个饥饿值为1的寿命为初始值的food*  
 self.\_\_grass[new\_pos[0]][new\_pos[1]] -= 1  
 else:  
 move\_food(old)  
 else: *# 未达到分裂条件，只能考虑运动，注意只能向没有Bacteria的地方运动* move\_food(old)  
 elif isinstance(old, Bacteria):  
 *# 如果是bacteria，考虑向周围运动，同时也要考虑bacteria的死亡情况* empty\_pos = [] *# 表示空位的位置* food\_pos = [] *# 表示food的位置* for x in range(8): *# 注意i, j已经用过了* direction = Field.\_\_direction[x]  
 position = i + direction[0], j + direction[1]  
 if not Field.out\_of\_range(Field.size, position):  
 sur = self.\_\_state[position[0]][position[1]]  
 *# 分析周围环境* if isinstance(sur, Food):  
 food\_pos.append(position)  
 elif isinstance(sur, Bacteria):  
 continue  
 elif sur is None:  
 empty\_pos.append(position)  
 else:  
 raise Exception(**"Field.evolution()未知错误"**)  
  
 def move\_bacteria(old):  
 *"""描述了bacteria不分裂时的行为"""* num\_food = len(food\_pos)  
 num\_empty = len(empty\_pos)  
 move\_prob = (num\_empty + 1) / (num\_empty + num\_food + 1) *# 不移动的可能性是没有food的位置和原地比上总共九个位置*  
 r = np.random.random\_sample()  
 if food\_pos != [] and r > move\_prob: *# 如果周围有food，吃掉食物且向有food的位置随机前进* old.hunger += 1  
 nxt\_pos = food\_pos[np.random.randint(len(food\_pos))]  
 food = self.\_\_state[nxt\_pos[0]][nxt\_pos[1]]  
 self.\_\_state[i][j] = None  
 self.\_\_state[nxt\_pos[0]][nxt\_pos[1]] = old  
 del food  
 else:  
 old.hunger -= 1 *# 如果周围没有food留在原地或者移动，变得更加饥饿*  
 empty = [(i, j)] + empty\_pos  
 nxt\_pos = empty[np.random.randint(len(empty))]  
 self.\_\_state[i][j] = None  
 self.\_\_state[nxt\_pos[0]][nxt\_pos[1]] = old  
  
  *# 下面要讨论bacteria面对空位和food的具体选择问题，这时候会包含bacteria的繁殖* if old.lifetime == 1:  
 self.\_\_state[i][j] = None  
 del old  
 else:  
 old.lifetime -= 1 *# 如果寿命不是1，寿命减一* if old.hunger == 1: *# 讨论饥饿值  
 # 首先是饥饿值仅剩1的情况* if food\_pos == []: *# 周围没有food，会被饿死* self.\_\_state[i][j] = None  
 del old  
 else:  
 nxt = food\_pos[np.random.randint(  
 len(food\_pos))] *# 随机挑选一个位置* food = self.\_\_state[nxt[0]][nxt[1]]  
 old.hunger += 1  
 self.\_\_state[nxt[0]][nxt[1]] = old  
 *# 将被选中food所占的位置替换为原来的bacteria* self.\_\_state[i][j] = None  
 del food *# 删除被吃food的引用，food总数减一* elif old.hunger > old.tig:  
 *# 当饥饿值大于设定值，触发分裂条件，有一定概率分裂* r = np.random.random\_sample()  
 if r < Bacteria.rp\_prob and empty\_pos != []: *# 判断是否分裂* new\_bac = Bacteria(hunger=3)  
 old.hunger -= 1 *# 这里的改动对结果影响十分大* new\_pos = empty\_pos[np.random.randint(len(empty\_pos))]  
 self.\_\_state[new\_pos[0]][new\_pos[1]] = new\_bac  
 else:  
 move\_bacteria(old)  
 else:  
 *# 剩余的情况，只有向空位移动或者吃food* move\_bacteria(old)  
 elif old is None:  
 *# 如果是空地，我们只考虑grass增长* self.grass\_grow(coordinate)  
 else:  
 raise Exception(**"Field.evolution()未知错误"**)  
  
 def transform(self):  
 *"""  
 将self.\_\_state的数据转化为容易观察的数据  
 food用0代表，bacteria用1代表  
 """* list\_output = [[] for i in range(Field.size)]  
 for x in range(Field.size):  
 for y in range(Field.size):  
 now = self.\_\_state[x][y]  
 if isinstance(now, Food):  
 list\_output[x].append(0)  
 elif isinstance(now, Bacteria):  
 list\_output[x].append(1)  
 elif now is None:  
 list\_output[x].append(None)  
 return list\_output  
  
 def update(self):  
 *"""  
 执行一次算作对于状态的一次更新，返回一个由0，1，None组成的二维列表  
 """* cnt = 0  
 while cnt < Field.size\*\*2:  
 i, j = np.random.randint(0, Field.size), np.random.randint(  
 0, Field.size)  
 self.evolution((i, j))  
 cnt += 1  
 Field.\_\_bacteria\_num.append(Bacteria.get\_num())  
 Field.\_\_food\_num.append(Food.get\_num())  
 return self.transform()  
  
 def grass\_grow(self, coordinate, max=5):  
 *"""描述一个位置草的生长，写成函数用起来方便嘛"""* i, j = coordinate  
 if i >= self.\_\_size or j >= self.\_\_size:  
 raise IndexError(**"Field.grass\_grow()所选位置超出了区域范围"**)  
 if self.\_\_grass[i][j] < 9:  
 self.\_\_grass[i][j] += self.\_\_grass\_growth\_list[i][j]  
  
 def get\_grass(self):  
 *"""调试的时候用一用"""* return self.\_\_grass  
  
 @staticmethod  
 def out\_of\_range(size, pos):  
 if (pos[0] < size and pos[0] >= 0) and (pos[1] < size and pos[1] >= 0):  
 return False  
 else:  
 return True  
  
 @classmethod  
 def plain\_mode(cls): *# 我的梦想是可以让草的增长实现一定的分布规律  
 """平均增长列表的构造，注意要先调用set\_average\_rate()与设置size"""* Field.\_\_grass\_growth\_list = [[  
 Field.\_\_average\_rate for i in range(Field.size)  
 ] for i in range(Field.size)]  
  
 @classmethod  
 def set\_average\_rate(cls, rate=1):  
 Field.\_\_average\_rate = rate  
  
 @classmethod  
 def clear\_food\_num(cls):  
 Field.\_\_food\_num = []  
  
 @classmethod  
 def get\_food\_num\_list(cls):  
 return Field.\_\_food\_num  
  
 @classmethod  
 def clear\_bacteria\_num(cls):  
 Field.\_\_bacteria\_num = []  
  
 @classmethod  
 def get\_bacteria\_num\_list(cls):  
 return Field.\_\_bacteria\_num  
  
 @classmethod  
 def get\_grass\_growth\_list(cls):  
 return Field.\_\_grass\_growth\_list  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 **"""调试的时候用一用"""** size = 50  
 field = Field(size)  
 for i in range(size\*\*2 // 10):  
 x = np.random.randint(0, size - 1)  
 y = np.random.randint(0, size - 1)  
 field.set\_bacteria((x, y))  
 for i in range(size\*\*2 // 10):  
 x = np.random.randint(0, size - 1)  
 y = np.random.randint(0, size - 1)  
 field.set\_food((x, y))  
 for i in range(500):  
 field.update()  
 if i % 20 == 0:  
 print(**"done%s"** % i)  
 list1 = open(**"food\_num"**, **"wb"**)  
 pickle.dump(Field.get\_food\_num\_list(), list1)  
 list2 = open(**"bacteria\_num"**, **"wb"**)  
 pickle.dump(Field.get\_bacteria\_num\_list(), list2)  
 plt.plot(Field.get\_food\_num\_list(), label=**"food population"**)  
 plt.plot(Field.get\_bacteria\_num\_list(), label=**"bacteria population"**)  
 plt.xlabel(**"time"**)  
 plt.ylabel(**"numbers"**)  
 plt.legend()  
 plt.show()

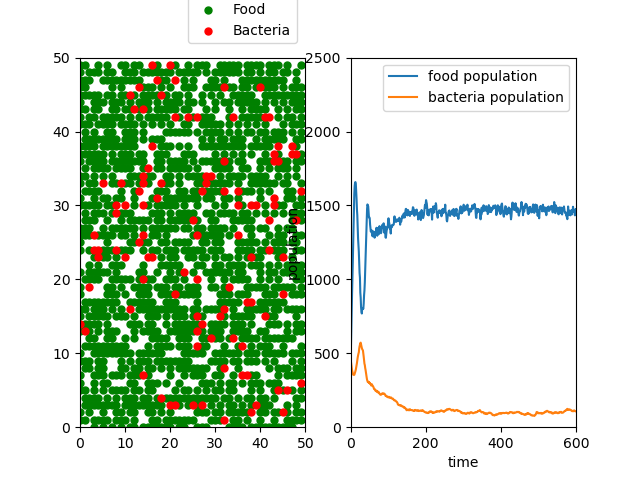
1. 使用说明
2. 参数设置：在使用时可选择的参数有，图像大小，用于自定义生成一个n\*n大小的图像，下面所有的数据都没有单位，捕食者寿命和和被捕食者寿命均代表更新的次数，捕食者饥饿值和被捕食者饥饿值是初始生成时或繁殖产生新捕食者或被捕食者的饥饿值，当捕食者和被捕食者获得食物时，饥饿值加一，否则减一，当饥饿值降为0时，会死亡，捕食者和被捕食者饥饿触发值，是指仅当饥饿值大于此值时，分裂才有可能进行，此值设置越高，则分裂越不容易发生，二者的繁殖概率均指二者到达触发条件后繁殖发生的概率，概率越高，繁殖能力越强，更新次数可以自定义设置
3. 运行：运行时，实时显示捕食者和被捕食者数量，实时显示捕食者和被捕食者在样方中的分布，捕食者数量/被捕食者数量-更新次数图像实时描绘，按空格键可控制暂停和继续,将光标移到曲线上，可以看到曲线上每个点对应的x,y值
4. 运行结果分析
5. 参数设置为50\*50大小，捕食者寿命40，饥饿值5，繁殖触发值3，繁殖概率0.5，被捕食者寿命20，饥饿值5，繁殖触发值为1，繁殖概率0.7时，可以看出图像具有一定的周期型，且满足捕食者数量曲线略滞后于被捕食者曲线



（2）分别改变捕食者繁殖触发值为1 和改变被捕食者繁殖触发值为3，均不改变捕食者被捕食者的稳定值的位置

（3）分别改变捕食者寿命为20和被捕食者寿命为40，与(1)中对比，二者中任一个寿命增长，都会使波动变大

（4）分别改变捕食者繁殖率为0.1和被捕食者繁殖率为0.2，可以明显地观察到，捕食者数量维持在较低水平，被捕食者数量维持在较高水平，且波动幅度减小；



1. 优化方向
2. 在空格进行暂停时，图像无法被显示出来
3. 最后保存图片时，无法将图片上的数据点信息保存下来
4. 草随着捕食者被捕食者的分布情况的变化无法显示