P-P 模型

一、模型假设

1. 假设海洋中资源足够丰富，食饵和捕食者在独立生存时以指数规律增长；
2. 因只研究食饵和捕食者之间存在的一般关系，忽略其他种群对这两个种群数量的影响；
3. 假设食饵和捕食者构成的食物链在该时间段内保持平衡状态，不受外界干扰。

二、符号说明

表 1 符号说明

符号	说明
x	食饵数量
y	捕食者数量
r	食饵增长率
a	捕食者掠取食物的能力
d	捕食者独自存在时的死亡率
b	食饵对捕食者的供养能力

三、模型的建立与求解

3.1 问题一模型的建立与求解

给出数学模型

已知 $x'(t)$ $y'(t)$ 分别表示食饵和捕食者种群相对于时间的变化，因此该问题由非线性微分方程组表示

$$x'(t) = rx(t) - ax(t)y(t) \tag{1}$$

$$y'(t) = -dy(t) + bx(t)y(t) \tag{2}$$

假设初始时刻食饵和捕食者的数量分别为 x_0, y_0，则常微分方程组初值问题的数学模型为

$$\begin{cases} x'(t) = rx(t) - ax(t)y(t) \\ y'(t) = -dy(t) + bx(t)y(t) \\ x(0) = x_0, y(0) = y_0 \end{cases} \tag{3}$$

求得数值解

在给定 x_0, y_0, r, a, b, d 的情况下，我们决定使用四阶 Adams 预测校正系统解出 $0 \leqslant x \leqslant 10$ 范围内的数值解。

首先选取合适间隔的离散点，即确定步长 h。公式如下：

$$h_{\text{new}} = h \cdot \left(\frac{\text{TOL}}{\text{err}}\right)^{1/(p+1)} \tag{4}$$

其中 p 为低阶方法的阶数，此处 $p=4$。通过设定合理的容差值，将四阶和五阶 Runge-Kutta 的差值作为误差估计 err，从而调整 h 的取值。最终确定 $h=0.01$。然后根据方法通过 Python 软件求解得到 $t=0.01, 0.02, ..., 10.00$ 时 x, y 的数值。我们从中取得若干点列在下表。

表 2 0 ⩽x⩽ 10 数值解

t	x	y	t	x	y
1.00	4392.88	1512.23	6.00	388.88	527.49
2.00	288.00	3175.16	7.00	2701.28	664.35
3.00	32.21	2042.43	8.00	1602.59	3240.39
4.00	25.39	1257.67	9.00	68.37	2556.42
5.00	69.19	781.57	10.00	23.41	1584.25

画出解图形

得到图 1，为问题解的图形。

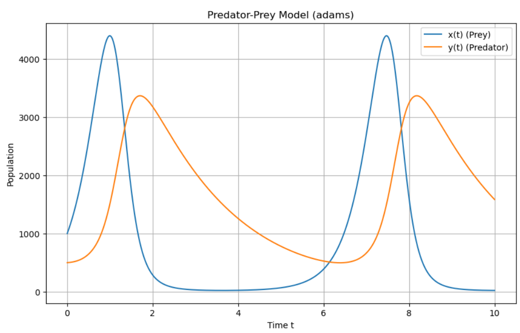

图 1 x, y 随时间 t 的变化图

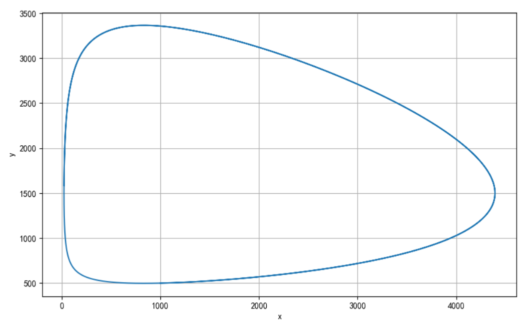

图 2 相轨线 $y(x)$

确定周期

根据模型假设，两种群结构关系处于稳定状态，相轨线 $y(x)$（图 2）为一固定曲线。根据提给背景可知该曲线封闭，所以 $x(t), y(t)$ 是周期函数。不妨选取 $x(t)$ 的两个相邻峰值来判断周期，易得周期 $T=6.47$。

确定均值

为求得 $x(t)$ 在一个周期内的平均值 $\overline{x}$，可通过以下公式：

$$\overline{x}=\frac{1}{T}\int_{t}^{t+T}x(\tau)\mathrm{d}\tau \tag{5}$$

同理可得 $\overline{y}$。经计算得到：$\overline{x}$=823.65，$\overline{y}$=1497.71。

3.2 问题二模型的建立与求解

对于问题二，已知数学模型和 7 组观测值 $(x_i,y_i)(i=0,1,...,6)$，我们用非线性最小二乘优化算法求得参数组 (a,r,b,d)。

因为 $x(0)=1000,y(0)=500$,与问题一相似,我们不妨设置初始值 $(a_0,r_0,b_0,d_0)=(0.002,3,0.0006,0.5)$。误差平方和 $erro$ 为

$$error=\sum_{i=0}^{6}(x_{\mathrm{pre}}-x_{\mathrm{obs}})^2+(y_{\mathrm{pre}}-y_{\mathrm{obs}})^2 \tag{6}$$

其中，$x_{\mathrm{pre}},y_{\mathrm{pre}}$ 为当前预测值，$x_{\mathrm{obs}},y_{\mathrm{obs}}$ 为观测值。

因为该问题得不到解析解，我们利用 Nelder-Mead 方法不断进行参数优化，得到最佳参数 $(a,r,b,d)=(2.68,0.0018,0.00079,0.48)$, 此时得到最小误差 0.507。

附录

```python
import numpy as np
import matplotlib.pyplot as plt
r = 3,a = 0.002,b = 0.0006,d = 0.5
# 定义ODE系统
def f(t, u):
    x, y = u
    dxdt = r * x - a * x * y
    dydt = -d * y + b * x * y
    return np.array([dxdt, dydt])
# 四阶Runge-Kutta方法
def rk4(f, t0, u0, t_end, h):
    t_values = np.arange(t0, t_end + h, h)
    n = len(t_values)
    u_values = np.zeros((n, len(u0)))
    u_values[0] = u0

    for i in range(1, n):
        t = t_values[i-1]
        u = u_values[i-1]
        k1 = f(t, u)
        k2 = f(t + h/2, u + h/2 * k1)
        k3 = f(t + h/2, u + h/2 * k2)
        k4 = f(t + h, u + h * k3)
        u_values[i] = u + h/6 * (k1 + 2*k2 + 2*k3 + k4)

    return t_values, u_values

# 四阶Adams预测校正方法
def adams_pc(f, t0, u0, t_end, h):
    t_values = np.arange(t0, t_end + h, h)
    n = len(t_values)
    u_values = np.zeros((n, len(u0)))
    u_values[0] = u0

    # 用RK4计算前4个点
    for i in range(1, min(4, n)):
        t = t_values[i-1]
        u = u_values[i-1]
        k1 = f(t, u)
        k2 = f(t + h/2, u + h/2 * k1)
        k3 = f(t + h/2, u + h/2 * k2)
        k4 = f(t + h, u + h * k3)
        u_values[i] = u + h/6 * (k1 + 2*k2 + 2*k3 + k4)

    for i in range(4, n):
        t = t_values[i-1]
        f_n = f(t_values[i-1], u_values[i-1])
        f_n1 = f(t_values[i-2], u_values[i-2])
        f_n2 = f(t_values[i-3], u_values[i-3])
        f_n3 = f(t_values[i-4], u_values[i-4])
        u_pred = u_values[i-1] + h/24 * (55 * f_n - 59 * f_n1 + 37 * f_n2 - 9 * f_n3)
        f_pred = f(t_values[i], u_pred)
        u_values[i] = u_values[i-1] + h/24 * (9 * f_pred + 19 * f_n - 5 * f_n1 + f_n2)
```

```python
    return t_values, u_values

t0 = 0
u0 = np.array([1000, 500])
t_end = 10
h = 0.01
t_adams, u_adams = adams_pc(f, t0, u0, t_end, h)
print("\nAdams方法10个时间点的解：")
for i in range(0,1100,100):
    print(f"t = {t_adams[i]:.2f}, x = {u_adams[i, 0]:.2f}, y = {u_adams[i, 1]:.2f}")

import pandas as pd

# 保存结果到表格
results = []
for i in range(0, 1100, 100):
    results.append({
        "t": t_adams[i],
        "x": u_adams[i, 0],
        "y": u_adams[i, 1]
    })
df = pd.DataFrame(results)
df.to_csv("adams_results.csv", index=False, encoding="utf-8-sig")
print("结果已保存到 adams_results.csv 文件中。")

# 绘制解图形
plt.figure(figsize=(10, 6))
plt.plot(t_adams, u_adams[:, 0], label='x(t) (Prey)')
plt.plot(t_adams, u_adams[:, 1], label='y(t) (Predator)')
plt.xlabel('Time t')
plt.ylabel('Population')
plt.title('Predator-Prey Model (adams)')
plt.legend()
plt.grid(True)
plt.show()

#绘制相轨线
from matplotlib import rcParams
# 设置字体为支持中文的字体，例如 SimHei (黑体)
rcParams['font.sans-serif'] = ['SimHei'] # 用黑体显示中文
rcParams['axes.unicode_minus'] = False # 解决负号显示问题

# 绘制图形
plt.figure(figsize=(10, 6))
plt.plot(u_adams[:, 0], u_adams[:, 1])

plt.xlabel('x')
plt.ylabel('y')
# plt.title('相轨线') # 中文标题
# plt.legend(['轨迹']) # 添加图例
plt.grid(True)
plt.show()

#计算周期
from scipy.signal import find_peaks

peaks, _ = find_peaks(u_adams[:, 0], distance=50)
```

```python
if len(peaks) >= 2:
    period = t_adams[peaks[1]] - t_adams[peaks[0]]
    print(f"估计的周期: {period:.2f}")
else:
    print("峰值不足, 无法估计周期")

#计算平均值
start_idx = peaks[0]
end_idx = peaks[1]

t_cycle = t_adams[start_idx:end_idx]
x_cycle = u_adams[start_idx:end_idx, 0]
y_cycle = u_adams[start_idx:end_idx, 1]

# 使用梯形法则计算积分
x_integral = np.trapz(x_cycle, t_cycle)
y_integral = np.trapz(y_cycle, t_cycle)

# 计算平均值
x_avg = x_integral / period
y_avg = y_integral / period

print(f"一个周期内x的平均值: {x_avg:.2f}")
print(f"一个周期内y的平均值: {y_avg:.2f}")

#最小二乘法求参数
import numpy as np
from scipy.optimize import minimize

t_obs = np.array([0, 1, 2, 3, 4, 5, 6])
x_obs = np.array([1000, 2996, 217, 29, 22, 49, 214])
y_obs = np.array([500, 1737, 3069, 2017, 1266, 800, 537])
def predator_prey(t, u, r, a, b, d):
    x, y = u
    dxdt = r * x - a * x * y
    dydt = -d * y + b * x * y
    return np.array([dxdt, dydt])
def adams_pc(f, t0, u0, t_end, h, params):
    r, a, b, d = params
    t_values = np.arange(t0, t_end + h, h)
    n = len(t_values)
    u_values = np.zeros((n, len(u0)))
    u_values[0] = u0
    for i in range(1, min(4, n)):
        t = t_values[i-1]
        u = u_values[i-1]
        k1 = f(t, u, r, a, b, d)
        k2 = f(t + h/2, u + h/2 * k1, r, a, b, d)
        k3 = f(t + h/2, u + h/2 * k2, r, a, b, d)
        k4 = f(t + h, u + h * k3, r, a, b, d)
        u_values[i] = u + h/6 * (k1 + 2*k2 + 2*k3 + k4)
    for i in range(4, n):
        t = t_values[i-1]
        f_n = f(t_values[i-1], u_values[i-1], r, a, b, d)
        f_n1 = f(t_values[i-2], u_values[i-2], r, a, b, d)
        f_n2 = f(t_values[i-3], u_values[i-3], r, a, b, d)
        f_n3 = f(t_values[i-4], u_values[i-4], r, a, b, d)
```

```
            u_pred = u_values[i-1] + h/24 * (55 * f_n - 59 * f_n1 + 37 * f_n2 - 9 * f_n3)
            f_pred = f(t_values[i], u_pred, r, a, b, d)
            u_values[i] = u_values[i-1] + h/24 * (9 * f_pred + 19 * f_n - 5 * f_n1 + f_n2)
    return t_values, u_values

def solve_ode_at_obs(params):
    t0 = 0
    u0 = np.array([1000, 500])
    t_end = 6
    h = 0.01
    t_values, u_values = adams_pc(predator_prey, t0, u0, t_end, h, params)
    indices = (t_obs / h).astype(int)
    return u_values[indices, 0], u_values[indices, 1]

def error_function(params):
    x_pred, y_pred = solve_ode_at_obs(params)
    error = np.sum((x_pred - x_obs)**2 + (y_pred - y_obs)**2)
    return error

initial_guess = [3, 0.002, 0.0006, 0.5]
result = minimize(error_function, initial_guess, method='Nelder-Mead')
best_params = result.x
print("最佳参数 r, a, b, d:", best_params)
print("最佳参数 r, a, b, d: [{:.2f}, {:.4f}, {:.5f}, {:.2f}]".format(*best_params))
print("最小误差:", result.fun)
```