关于生态群落中丰度波动和平均灭绝时间的随机共振现象的有关讨论

王治平

指导老师：关剑月、吴枝喜

2020 级 物理学二班

320200908151

zhpwang20@lzu.edu.cn

摘要

本文基于 2021 年的一篇探究周期性外场作用下种群参数随机共振的文章 [1], 对其进行工作复现；用求解常微分方程的常见方法成功复现出了论文中所有图片；同时对于数值算法进行了总结与介绍、对部分算法的敛散性、收敛速度、复杂度进行了初步讨论；并且，对于各个图片的物理意义、数值算法的选取、鞍点的意义与逃离方法发表部分个人观点，并将其与深度学习中的常见思路进行类比，提出了其他逃离本题鞍点的方法。

Key words: 常微分方程数值解; 随机共振; 异宿轨道；逃离鞍点；算法选取；周期性外场；线性响应

课题程序实现已上传至 https://github.com/YingDo/level2/tree/main/work3

Compiled on: 2021 年 12 月 26 日.

目录

1 引言

本文基于一篇本年度 8 月份发表于 *PHYSICAL REVIEW E* 上的一篇文章———*Stochastic resonance of abundance fluctuations and mean time to extinction in an ecological community*[1], 文章讲述了生态群落中丰度波动和平均灭绝时间的随机共振现象及其成因, 文章作者巧妙的用外场 $h(t)$ 作为环境对于种群的扰动类比。其分析过程中进行了不少常微分方程的数值计算，与当前计算物理的课程学习较为贴切，因此我决定复现本文的思路以及部分图片，作为课题论文提交。

该文章共有四张图，分别是以 *Lyapunov* 函数着色的 n-m 空间相图,χ_D 作为纵轴的种群数量随机共振图, 灭绝曲线在 q_1, q_2 以及 p_1, p_2 平面上的相轨迹投影，以及 χ_S 作为纵轴的种群灭绝时间随机共振图。

其中，图片 2，4 均可以看出外场对于生物群落的贡献——分别造成了两种重要参数的参数共振现象（在本文成为随机共振）. 另外，其中大多数图片涉及偏微分方程数值解问题，不过在本题目中对于方法和精度大多没有过多要求。为结合所学内容本课题报告在复现图片的同时通过 *Butcher* 表对 *Runge–Kutta* 方法做了简单的介绍与总结, 对齐收敛性稳定性以及算法选取方面进行了些许讨论，同时发表了一些对于作图过程中关键点（例如：逃离鞍点的方法）的个人想法与方法总结。

2 成果预展示

本课题最引人注目的成果便是文献图片的复现，先将复现结果提前展示如下，版面问题暂且不按照顺序展示，增加了阅读难度以及降低了美观性，敬请见谅

各图的复现如下图所示，各参数设定大都参照原文献，图 4 方面有个人认为的改进:

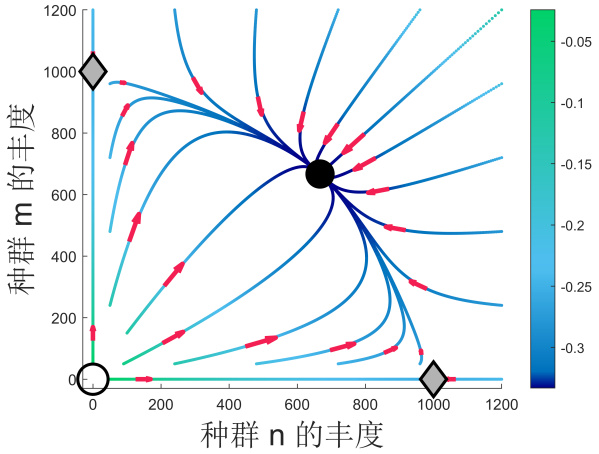


图 1. 对于论文 FIG.1 的复现, 表示在没有外场的情况下，丰度动力学和式 (2) 的不动点。其中 2 个灰色填充菱形为鞍点，和黑色填充圆为稳定点，黑色圆环为不稳定点。箭头的颜色表示 *Lyapunov* 函数的值，(此处选取的 *Lyapunov* 函数为 $\phi(n,m) = \frac{\mu}{2M^2}(n^2 + m^2) - \frac{\lambda\mu}{M}(n+m) + \frac{mn}{M^2}$). 参数选取 $M = 2000, \lambda_b = 0.6, \lambda_d = 0.1, \mu = 2$.

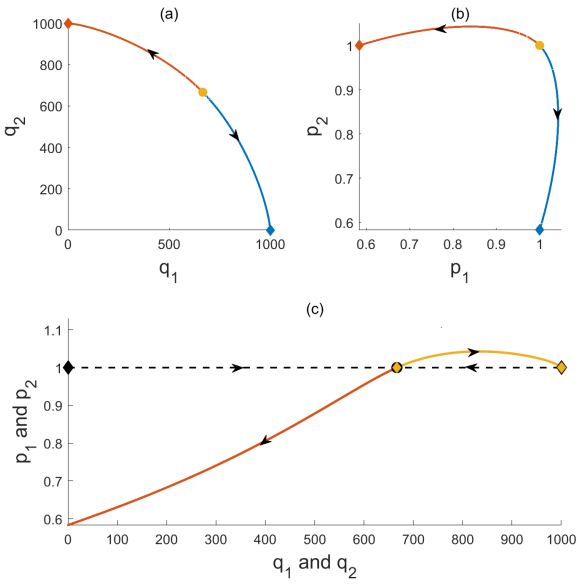


图 2. 对于论文 FIG.3 的复现, 从共存状态 η_c 到灭亡状态 $\eta_i (i \in \{1,2\})$ 的最可能异宿轨道 (η_i 为终点的异宿轨道标记为 τ_i) 的相轨迹被投影在 (a) 坐标空间和 (b) 动量空间上。在图 (c) 中，$\tau_i (i \in \{1,2\})$ 被绘制在物种 1(橙色) 和物种 2(黄色) 的相空间中。虚线表示确定性情况（即由二维空间中从鞍点到稳定点的轨迹）. 参数 $M = 2000, \lambda_b = 0.6, \lambda_d = 0.1, \mu = 2$.

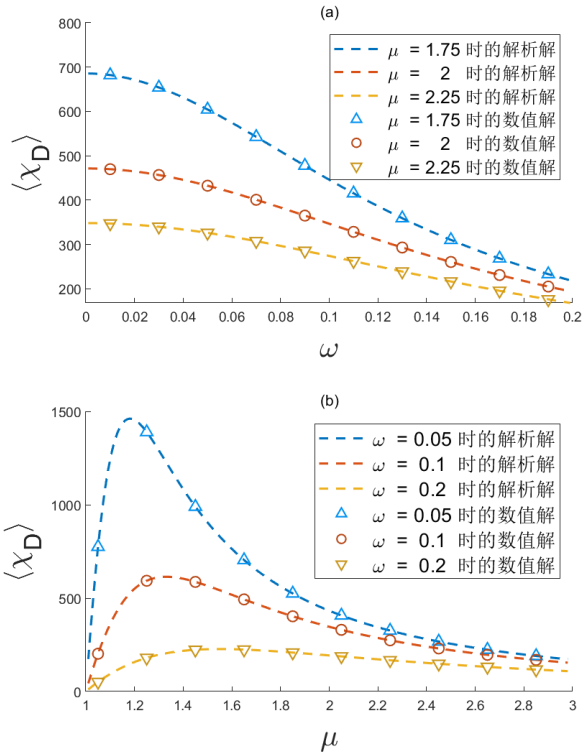


图 3. 对于论文 FIG.2 的复现,χ_D 的时间均值 $\langle\chi_D\rangle$ 在图 (a)ω 为自变量，μ=1.75, 2.00, 2.25; 图 (b)μ 为自变量，ω=0.05, 0.1, 0.2。(a) 和 (b) 中的虚线和标记符号分别表示理论和数值结果，显示出很好的一致性。参数选取 $M = 2000, \lambda_b = 0.6, \lambda_d = 0.1, \mu = 2, \epsilon = 0.01$.

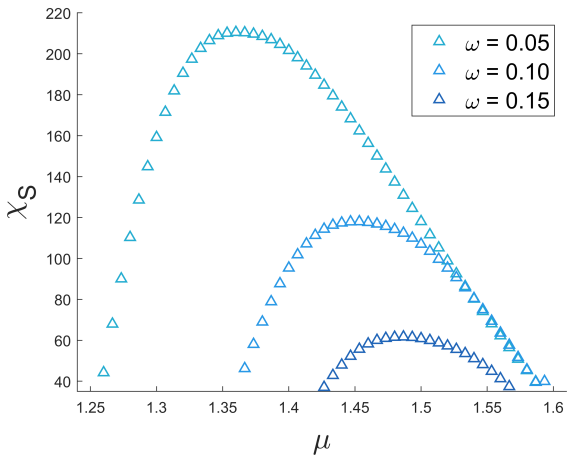


图 4. 对于论文 FIG.4 的复现,μ 为自变量，ω 分别为 0.05, 0.1, 0.2 的对数敏感性的 χ_S 如图所示。在每一个 ω 值中，可以观察到一个随机共振的峰。注意，此处和论文图没有完美贴合是由于取了积分步长更小以及更小的微扰导致的 SR 峰更加尖锐，更易于观察峰值所在，即随机共振的对应参数。参数选取 $M = 2000, \lambda_b = 0.6, \lambda_d = 0.1$.

3 文献思路与整理

引言部分已经说明，本文主要工作在于复现前人论文的成果，因此描述该文章的思路是十分必要的，现将其行文思路概述如下。

3.1 研究背景

作者认为环境在大尺度之下是随机变化的，然而由于阳光供给、地球自转、大气循环等宏观因素的周期性，环境影响因素也可以认为是周期性的。在这个周期性变化的的环境下，不同的生种群群有可能接收到或有利或有害的外界条件，不同种群对于环境的响应会使得他们在竞争中存在优劣势的差异，因此可能存在原有的生物平衡被破坏甚至生物灭绝。

现在大多数的研究都未曾考虑环境的周期性变化问题，大多数作为稳态外环境来进行考虑。同时作者认为环境的变化会极大的影响种群之间的平衡，因此作者试图用周期性的环境变化绘制出更加符合实际情况的理论模型。

3.2 理论模型

为了建立理论化体系，作者建立了如下假设模型：认为

- 每个种群的每个个体都会以恒定的速度生育和死亡，(称其为 λ_b, λ_d)
- 任何两个个体之间都会发生竞争，其中一个会以一定的速度死亡
- 两个种群的出生率和自发死亡率是相同的，而竞争死亡率则取决于竞争个体的种群

文字表述过于抽象，形象的可以等价于如下反应:

$$\begin{aligned} X_i &\to X_i + X_i && \text{概率为 } \lambda_b \\ X_i &\to \emptyset && \text{概率为 } \lambda_d \\ X_i + X_j &\to X_j && \text{概率为 } \alpha_{ij} \end{aligned}$$

其中 $i, j \in \{1, 2\}$，1，2 分别代表两个不同的种群。也就是说，反应 3 在种间和种外都有可能发生，在本文种，只考虑种间竞争。作者认为，竞争死亡率与其回报之间的关系可以用竞争博弈论描述，同时设竞争死亡率是回报率 $\Lambda_{ij}(>0)$ 的减函数，有

$$\alpha_{ij} = (M\Lambda_{ij})^{-1}$$

M 参数度量了种群竞争率的强度，该强度与种群大小和个体竞争率有关。这个公式表明，拥有巨额收益的个体将会同时拥有较小的死亡概率。

避免不必要的麻烦（不难证明忽略种内竞争该假设是完全合理的），设竞争收益矩阵为对称矩阵：

$$\Lambda = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$$

矩阵中的对角元素代表种内竞争的收益率，表示种内相互作用的收益，非对角元素表示种间相互作用的收益。在竞争过程中不同种群是等价的，但竞争死亡率与竞争方式有关。为了保证两个种群的共存，该文只考虑种间竞争收益大于种内竞争收益的情况 $(a < b)$.

现在通过调节种间收益来响应周期性变化的的环境因素，同时，对收益矩阵进行单位化，有

$$\begin{bmatrix} 1 & \mu + \epsilon \sin \omega t \\ \mu - \epsilon \sin \omega t & 1 \end{bmatrix} \tag{1}$$

基于此假设，不同的种群对环境有不同的影响(竞争优势或劣势)。两个种群共存的条件为 $1 < \mu - \epsilon$. 此后，作者将环境变化视为一个周期性的“外场”(磁场)，和 ϵ 和 ω 是外场的振幅和角频率。作者认为该场是近乎无穷小的，即 $\epsilon \ll 1$ 接下来的目标便是研究系统对外场的响应。

3.3 种群数量波动的随机共振现象

为了研究环境变化引起的丰度波动，列出了以下微分方程组：

$$\begin{cases} \dot{n} = [\lambda - \frac{1}{M}(n + \frac{m}{\mu + h(t)})] \\ \dot{m} = m[\lambda - \frac{1}{M}(m + \frac{n}{\mu - h(t)})] \end{cases} \tag{2}$$

其中 n 代表种群 1 的个体数目; m 代表种群 2 的个体数目 $h(t) = \epsilon \sin \omega t, \lambda = \lambda_b - \lambda_d$. 在绝热近似下 $(\omega \ll 1)$ 可以找到这个系统的四个不动点 (满足 $\dot{n} = 0; \dot{m} = 0$):A(0,0)、B(λM,0)、C(0，λM) 和 D(n^*(t)，m^*(t))。

由于之前提到过的共存条件 $(a < b)$，可以得出 A 为不稳定不动点，D 为稳定不动点，B,C 均为

鞍点。同时 D 点会随着一个非零场 h(t) 而随时间变化，时间演化方程如下

$$\begin{cases} n^*(t) = \frac{(\mu-h)(\mu+h-1)}{\mu^2-h^2-1}\lambda M \\ m^*(t) = \frac{(\mu+h)(\mu-h-1)}{\mu^2-h^2-1}\lambda M \end{cases} \tag{3}$$

此处已经可以说明这说明丰度的波动是由外部场 h(t) 引起的。进行简单的运算，作者得到了归一

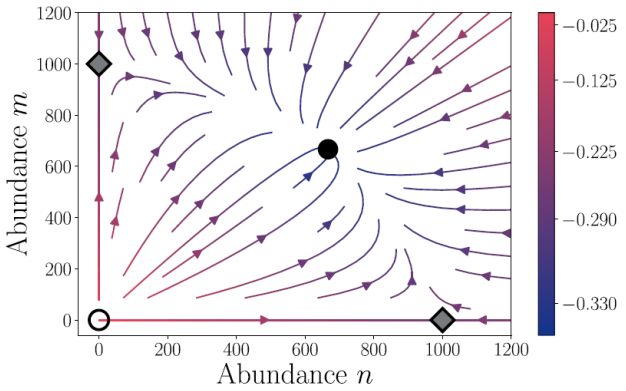


图 5. 论文原图 FIG.1

化的种群丰度波动

$$\begin{cases} \dot{\delta_x} = \frac{\lambda^2}{(\mu+1)^2}h - \frac{\lambda\mu}{\mu+1}\delta_x - \frac{\lambda}{\mu+1}\delta_y + O(\delta^2, h\delta) \\ \dot{\delta_y} = -\frac{\lambda^2}{(\mu+1)^2}h - \frac{\lambda\mu}{\mu+1}\delta_y - \frac{\lambda}{\mu+1}\delta_x + O(\delta^2, h\delta) \end{cases} \tag{4}$$

这些波动描述了随机噪声，但是由于其耦合造成的运算困难，作者用最常见的卷积运算进行了脱耦，得到

$$\begin{cases} \dot{\delta_{||}} = \sqrt{2}h_{||} - \frac{\mu-1}{\mu+1}\lambda\delta_{||} \\ \dot{\delta_{\perp}} = -\lambda\delta_{\perp} \end{cases} \tag{5}$$

此处 $h_{||} = [\lambda^2/(\mu+1)^2]h$，观察 Eq.5, 不仅可以发现去耦之后的简洁还可以发现 $\delta_\perp$ 是指数衰减的，至此在长时平均下（也就是该文章的研究范围下）得到了一维化的随机噪声动力学方程。当 h(t) = 0 时，弛豫时间满足 $\tau_{||}^{-1} = \lambda(\mu-1)/(\mu+1)$;$\tau_{\perp}^{-1} = \lambda$

求解 Eq.5 的平行分量的方程，显然有 $\delta_{||}(t) = \delta_0 sin(\omega t - \sigma)$ 形式的解，解析求解得到 $\delta_0 = \frac{\sqrt{2}\lambda^2}{(\mu+1)^2}\frac{\epsilon\tau_{||}}{\sqrt{1+\omega^2\tau_{||}^2}}$ 不妨将外场等效为弱周期磁场，

由一阶线性微分方程组所给出解的单调性，不难看出仅仅变动 Ω 均会在 ω = 0 时取最大值，并无随机共振现象，在该文献的 FIG.3 种也可看出。

和其他常见的系统一样，*Lyapunov* 函数可以作为自由等的等势表出，即可以吧种间竞争的收益 μ 类比为温度。由此，可以得出 μ 值的变动可以引起随机共振，同样的在文献图片中可以看出该结论。

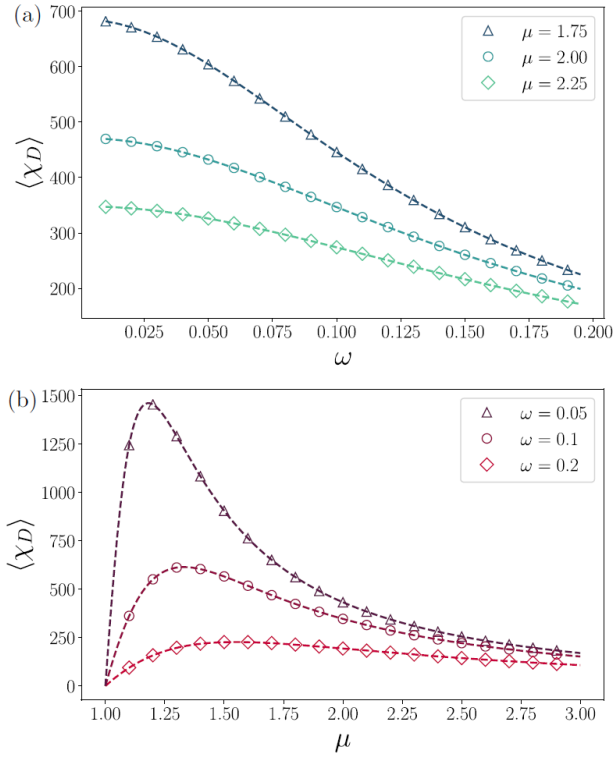


图 6. 论文原图 FIG.2

$$N_{||}(t) = \int_{-\infty}^{\infty}\frac{\delta N_{||}(t)}{\delta h(t')}h(t')dt' \tag{6}$$

为证明随机共振现象，应用线性响应定理，引入响应系数 χ_D 代入 $N_{||}(t) = M\delta_{||}(t)$ 并进行傅里叶变换，得到

$$\chi_D(\omega,\mu) = Re[\lim_{\delta h\to 0}\frac{\delta N_{||}(\omega)}{\delta h(\omega)}] = \frac{\sqrt{2}\lambda^2 M}{(\mu+1)^2}\frac{\tau_{||}}{1+\omega^2\tau_{||}^2} \tag{7}$$

即为 χ_D 的解析值。作图即可观察到种群数量波动的随机共振现象。

显然的，随机共振极值对应的 μ_D 满足

$$\frac{\partial\chi_D}{\partial\mu} = \chi_D\left[1 + 2\tau_{||}\left(\frac{1}{\lambda\tau_{||}^2 - \tau_{||}} - \frac{\omega^2\tau_{||}}{1+\omega^2\tau_{||}^2}\right)\right]\frac{d\tau_{||}}{d\mu} \tag{8}$$

等价于

$$\omega = \tau_{||}^{-1}\sqrt{(\lambda\tau_{||})/(\lambda\tau_{||}-3)}$$

该理论与数值的完美统一也在 FIG.3 种有所体现。

3.4 种群平均灭绝时间的随机共振现象

由于丰度存在随机共振现象，作者怀疑在平均灭绝时间这一维度上也存在这关于环境的随机共振。原有的确定性方程中由于其共存点为稳定点，无法观察到随机共振现象，因此有必要扩展为在四维像空间种的讨论。由 WKB 近似加上如下主方程便可以计算出零场条件下的平均灭绝时间：

$$\begin{aligned}\dot{P}_{n,m} = {}&\lambda_b[(n-1)P_{n-1,m} + (m-1)P_{n,m-1} - (n+m)P_{n,m}]\\ &+\lambda_d[(n+1)P_{n+1,m} + (m+1)P_{n,m+1} - (n+m)P_{n,m}]\\ &+\frac{1}{M}[(n+1)nP_{n+1,m} + (m+1)mP_{n,m+1}\\ &-n(n-1)P_{n,m} - m(m-1)P_{n,m}]\\ &+\frac{1}{\mu M}[(n+1)mP_{n+1,m} + n(m+1)P_{n,m+1} - 2nmP_{n,m}]\end{aligned} \tag{9}$$

引入生成函数

$$G(p_1,p_2,t) = \sum_{n,m} P_1^n P_2^m P_{n,m}(t) \tag{10}$$

生成函数由主方程给出的哈密顿算符给出，可以列出虚时薛定谔方程

$$\partial_t G = -\hat{H}_0 G$$

. WKB 近似下的解为 $G(p_1,p_2,t) = e^{-t/\tau}e^{-S_0(p_1,p_2)}$，其中 $1/\tau$ 为 $\hat{H}_0$ 的较小本征值。

对哈密顿量进行一阶近似，有

$$\begin{aligned}H_0 = {}&\lambda_b[(1-p_1)p_1q_1 + (1-p_2)p_2q_2]\\ &+\lambda_d[(p_1-1)q_1 + (p_2-1)q_2]\\ &+\frac{1}{M}[(p_1-1)p_1q_1^2 + (p_2-1)p_2q_2^2]\\ &+\frac{1}{\mu M}[(p_1-1)p_2q_1q_2 + (p_2-1)p_1q_1q_2]\end{aligned} \tag{11}$$

不妨记 $q_1 \equiv -\partial_{p_1}S, q_2 \equiv -\partial_{p_2}S$，

由上述方程给出的动力学关系，带入哈密顿正则方程求出对应的广义坐标与广义动量变化率，即可得到该四维相空间的动力学方程

$$\begin{cases}\dot{q}_1 = -\partial_{p_1}H_0 = \lambda_b(2p_1-1)q_1 - \lambda_d q_1 - \frac{1}{M}(2p_1-1)q_1^2 - \frac{1}{\mu M}(2p_2-1)q_1q_2\\ \dot{q}_2 = -\partial_{p_2}H_0 = \lambda_b(2p_2-1)q_2 - \lambda_d q_2 - \frac{1}{M}(2p_2-1)q_2^2 - \frac{1}{\mu M}(2p_1-1)q_1q_2\\ \dot{p}_1 = \partial_{q_1}H_0 = \lambda_b(1-p_1)p_1 + \lambda_d(p_1-1) + \frac{2}{M}(p_1-1)p_1q_1 + \frac{1}{\mu M}(2p_1p_2-p_1-p_2)q_2\\ \dot{p}_2 = \partial_{q_2}H_0 = \lambda_b(1-p_2)p_2 + \lambda_d(p_2-1) + \frac{2}{M}(p_2-1)p_2q_2 + \frac{1}{\mu M}(2p_1p_2-p_2-p_1)q_1\end{cases} \tag{12}$$

由于广义动量的广义性，不妨设为 $p_1 = p_2 = 1$. 考察四维相图的不动点，可以发现，$(H_0 = 0)$ 原有的两个鞍点仍未鞍点：$\eta_1 = (q_1,q_2,p_1,p_2) = \left(0,\lambda M,\frac{\lambda+\mu\lambda_d}{\mu\lambda_b},1\right)$，$\eta_2 = \left(\lambda M,0,1,\frac{\lambda+\mu\lambda_d}{\mu\lambda_b}\right)$，同时在二维空间中的稳定点变成了四维相空间中的鞍点 $\eta_c = \left(\frac{\mu}{\mu+1}\lambda M,\frac{\mu}{\mu+1}\lambda M,1,1\right)$ 这使得由 η_c 走向 $\eta_i(i \in \{1,2\})$ 存在了可能，显然这在之前的确定性方程中是被禁阻的。

计算平均灭绝时间 τ_E 需要将作用量沿着 η_c 走向 $\eta_i(i \in \{1,2\})$ 在相空间之中的轨迹进行积分：

$$\begin{aligned}S_0 &= \int_{-\infty}^{\infty}\mathcal{L}_0(p_i,\dot{p}_i,t)dt\\ &= \int_{-\infty}^{\infty}(-\sum_i q_i\dot{p}_i - H_0)dt\\ &= -\int_{-\infty}^{\infty}(q_1\dot{p}_1 + q_2\dot{p}_2)dt\end{aligned} \tag{13}$$

显然的，η_1,η_2 具有极高的对称性，因此作者此

后的讨论仅以 η_c 走向 η_1 为例.

作者使用 *Chernykh – Stepanov* 迭代方法，求解 Eq.12, 同时才得到了关系式 $\tau_E \propto exp(S_0)$.

求解方程的大概思路如下，首先进行初次试探：在微扰下从 η_1 出发，以确定性数值达到 η_c, 得到该确定性异宿轨道的相关参数，取有效值 q_1, q_2 进行后续迭代, 之后用 *Chernykh – Stepanov* 迭代方法，先由 q_1, q_2 反向求解 Eq.12(η_1 走向 η_c) 得到 p_1, p_2，再 p_1, p_2 由正向求解 Eq.12(η_c 走向 η_1) 得到，q_1, q_2，迭代多次得到稳定轨迹即为最概然异宿轨道.

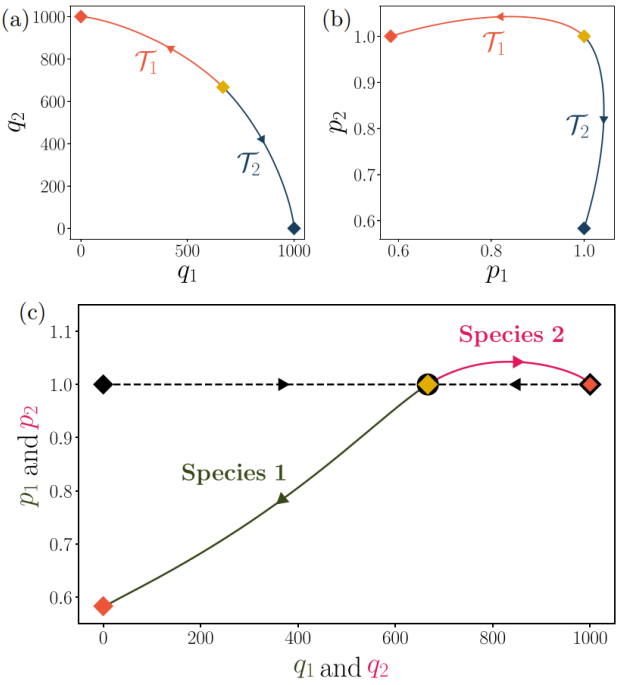


图 7. 论文原图 FIG.3

对于极小外场扰动，有

$$H_0 + \epsilon H_p$$

其中

$$H_p(t, t_0) = \frac{\sin \omega(t - t_0)}{\mu^2 M}(p_2 - p_1)q_1 q_2 \qquad (14)$$

计算 t_0 任意时最小的附加作用量，有

$$S_p = \min_{t_0} \left\{ \int_{-\infty}^{\infty} H_p(t, t_0) dt \right\} \qquad (15)$$

作为初相位，可以认为 $t_0 \in (0, 2\pi]$, 考虑微扰 η_c 到 η_1 的作用量 S 可以写作 $S = S_0 + \epsilon S_p$，即

$$\tau_E \propto exp(S_0 + \epsilon S_p) \qquad (16)$$

展示了环境对于平均灭绝时间的影响

同样的，对平均灭绝时间应用线性响应定理，求解线性响应系数的对数值，可以展现平均灭绝时间对于环境的敏感程度，

$$\chi_S(\omega, \mu) = \left| \lim_{\delta h \to 0} \frac{\delta \ln \tau_E}{\delta h} \right| = |S_p| \qquad (17)$$

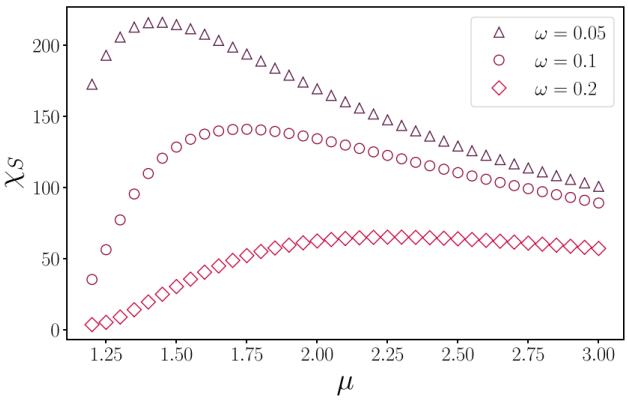


图 8. 论文原图 FIG.4

可以看出，任给 ω，均存在 μ 使得 χ_S 存在 SR 峰。因此，作者认为的平均灭绝时间也存在随机共振。

4 复现思路与算法选取

按照计划，我准备在大致了解文章思路与流程的情况下尽量对于四张图片进行复现，期间大多数图片的绘制过程运用到了常微分方程的数值解技巧，因在复现思路之前需要先进行一系列的算法选取。

4.1 算法优劣量度

算法选取不能是毫无章法的，在本题目中，收敛速度较快且迭代较为简单，因此在算法选取可以优先考虑下述方案，值得一提的是，以下算法讨论仅仅考虑在 C 语言中开启-O2 优化时的条件下，因此部分最基本的优化问题不做讨论。

4.1.1 复杂度

首先要提到的便是算法复杂度，在本课题中一般来讲迭代次数较少，C 语言的数据缓存区完全足够，因此不讨论空间复杂度，后续仅仅讨论时间复杂度。

4.1.2 敛散性

之后介绍的常微分方程数值解方法大都基于迭代，在本题的讨论中，大多数算法也都是收敛的。

4.1.3 光滑鲁棒性

在很多有关数值计算的书本中都有提到，在函数变化较为剧烈的范围内，有时高阶方法反而没有低阶方法稳定性更好，由此产生了较多经验式的结论，但是查阅资料暂未发现有效的体系化判定有函数的光滑性判断算法优劣的判据。不过本题的实质是对于他人结果的复现，观察 FIG.1,2,3,4，不难发现本题中曲线均极为光滑，因此可以大致认为所有方法都满足求解条件。

以上大段话看似无用——得出结论较为宽松，但个人认为以上过程实质上是算法选取的一个重要意义，保证算法存在有效的可能性才会使后续分析存在价值。

4.2 算法介绍与选取 [2]

目前广泛使用的算法实质上都属于 *Runge–Kutta* 法，该方法由数学家 Carl Runge 与 Wilhelm Kutta. 于 1900 年左右发明.

4.2.1 *Runge–Kutta* 方法以及 *Butcher* 表介绍

在通常的讨论中，常用以下约定俗成的符号来表示：

$$\frac{dy}{dt} = f(t, y).$$

显式 Runge–Kutta 方法采用以下形式

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i \\ k_1 &= f\left(t_n, y_n\right), \\ k_2 &= f\left(t_n + c_2 h, y_n + h\left(a_{21} k_1\right)\right), \\ k_3 &= f\left(t_n + c_3 h, y_n + h\left(a_{31} k_1 + a_{32} k_2\right)\right), \\ &\vdots \\ k_i &= f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j\right). \end{aligned} \tag{18}$$

s 阶段的隐式方法的阶段采用更一般的形式

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} k_j\right). \tag{19}$$

要指定特定方法，需要提供整数 s（级数）和系数 a_{ij}（对于 $1 \le j < i \le s$）、b_i（对于 $i = 1, 2, \ldots, s$）和 c_i（对于 $i = 2, 3, \ldots, s$）。矩阵 a 被称为 Runge-Kutta 矩阵，而 b 和 c 被称为权重和节点，这些数据通常被安排在一个助记符装置中，称为 *Butcher* 表（以 John C. Butcher 的名字命名）：

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

可以说，给定一个 *Butcher* 表，就给定了一种 *Runge–Kutta* 法。[2]

接下来要介绍 *Runge–Kutta* 的阶数和有效性判定，阶数可以说是很显然的，矩阵的行列数目便是阶数，但是空有阶数是没有意义的：一般的，如果要求方法具有一定的阶数 p，则还有要求局部截断误差为 $O(h^{p+1})$，最终误差为 $O(h^p)$。这些可以从截断错误本身的定义中得出。因此对于矩阵 a 便有了一定要求，要求过于繁杂暂不赘述，相见附录.

首先来介绍显式 *Runge–Kutta* 法，方法过于繁多，根据上述给出的 *Butcher* 表格以及附录中的

条件便可批量生产 *Runge–Kutta* 法，因此我仅仅挑选几个有代表性的进行介绍，同时只给出 *Butcher* 表，具体过程可以由 Eq.17 进行推导或者参照课本。

• 向前欧拉法

0	0
	1

注意到，向前欧拉法是唯一满足结束条件的一阶方法。

•Heun 方法 (改进欧拉)

Heun 的方法是具有两个阶段的二阶方法。它也称为显式梯形规则、改进的欧拉方法或改进的欧拉方法。(有趣的是：“eu” 的发音与 “Euler” 相同，因此 “Heun” 与 “coin” 押韵)：

0	0	0
1	1	0
	1/2	1/2

这便是作者多次提到的 *improved Euler method*，按照一些教材的逻辑，用该方法是由隐式方法预估矫正得出，在这里我不讨论方法的正统性，只需要了解该方法满足阶数条件，达到了局部三阶精度即可。一般来说，本方法的精度已经满足要求，尤其是对于本画图复现课题，对于一般尺寸的图片已经足够精确。

• 经典四阶方法

“原始” Runge-Kutta 方法，也就是本文中用到的方法，缺点是截断误差较大（同阶）但是利于书写与得出，他便是最开始被提出的 *Runge–Kutta* 法

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	1/3	1/3	1/6

• 3/8 规则四阶方法

这种方法不像 “经典” 方法那样 “臭名昭著”，但同样经典，同时它与经典方法在同一篇论文中被提出。

0	0	0	0	0
1/3	1/3	0	0	0
2/3	−1/3	1	0	0
1	1	−1	1	0
	1/8	3/8	3/8	1/8

• 拉尔斯顿的四阶方法

可以证明这种四阶方法具有最小截断误差。

0	0	0	0	0
.4	.4	0	0	0
.455737	.296977	.158759	0	0
1	.218100	−3.05096	3.832864	0
	.174760	−.551480	1.205535	.171184

以上距离了部分景点的显示 *Runge–Kutta* 法，可以证明均满足算法可选条件，

给定阶数，用附录方法可以求出任意阶的 *Runge–Kutta* 显式 *Butcher* 表，即可求出任意阶 *Runge–Kutta* 表达形式，值得一提的是，在求解微分方程的过程中，实际上是按照 $f(x,y)$ 的积分方程进行处理，所以经常可以观察到和龙贝格积分的系数相似之处（该数学本质的另外一种解释为两种算法俩都可以用泰勒展开进行系数推导）. 不过随着阶数上升，时间复杂度指数上升，结合误差需求不大，一般的选取改进欧拉法或者四阶 *Runge–Kutta* 足矣。

4.2.2 隐式 *Runge–Kutta* 法介绍与讨论

提到稳定性，不得不介绍一下隐式算法，其中显式方法是收敛条件课本里已讲说，在此不再赘述。

到目前为止提到的所有 Runge-Kutta 方法都是显式方法。显式 Runge-Kutta 方法通常不适合刚性方程的解，因为它们的绝对稳定区域很小；特别是，它是有界的，这个问题在偏微分方程的求解中尤为重要。

显式 Runge-Kutta 方法的不稳定性引出了隐式方法的发展。隐式 Runge-Kutta 方法具有以下形式：

$$\begin{cases} y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \\ k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} k_j\right), \quad i = 1, \ldots, s \end{cases} \tag{20}$$

与显式方法的区别在于，在显式方法中，j 的求和上限为 i-1，隐式则为 s。这也出现在 *Butcher* 表中体现：系数矩阵 $a_{ij}a_{ij}$ 的显式方法是下三角矩阵。在隐式方法中，则不然，如下图所示。

$$\begin{array}{c|ccc} c_1 & a_{11} & a_{12} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & \ldots & a_{2s} \\ \vdots & \vdots & \vdots & \vdots & \\ c_s & a_{s1} & a_{s2} & \ldots & a_{ss} \\ \hline & b_1 & b_2 & \ldots & b_s \\ & b_1^* & b_2^* & \ldots & b_s^* \end{array} = \begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b^T} \end{array}$$

b^*b^* 给出见下

$$y_{n+1}^* = y_n + h\sum_{i=1}^{s} b_i^* k_i,$$

误差为

$$e_{n+1} = y_{n+1} - y_{n+1}^* = h\sum_{i=1}^{s} (b_i - b_i^*) k_i,$$

同样的显式中由如下通表

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & & \ddots & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \\ & b_1^* & b_2^* & \cdots & b_{s-1}^* & b_s^* \end{array} \tag{21}$$

自适应算法也可以通过误差提出，不过由于篇幅原因暂不介绍。

稳定性篇幅过长，证明隐式稳定性的优越详见附录。

同时本篇文章满足最基本的收敛条件，因此没有进行隐式迭代，篇幅所限不进行隐式方法的 *Butcher* 表列举。

5 图片复现与数值计算

有了合适的算法，就可以开始进行复现工作，下面分别展示了四个图片的复现工作，程序实现见附录，具体讨论见正文下一部分。

5.1 复现实现

下面是每一张图片的复现工作介绍

5.1.1 FIG1 复现

对于观察原论文 FIG1

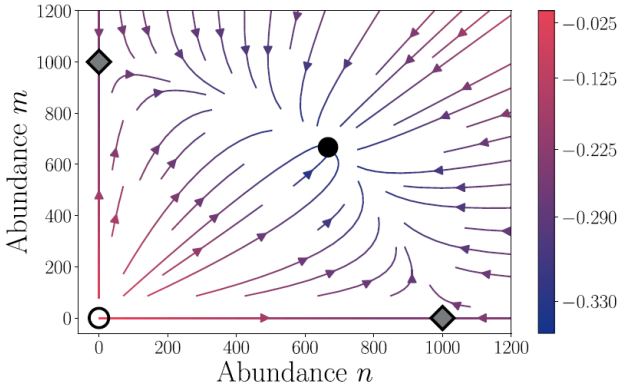


图 5*：论文原图 FIG.1

得出该图片实质是上 n,m 像空间中的相流图像。由于满足零场条件，有一个固定的稳定点：(2000/3,2000/3)，同时在边缘有两个鞍点 (1000,0),(0,1000), 以及在原点处的不稳定点。

由方程 3

$$\begin{cases} n^*(t) = \frac{(\mu-h)(\mu+h-1)}{\mu^2-h^2-1}\lambda M \\ m^*(t) = \frac{(\mu+h)(\mu-h-1)}{\mu^2-h^2-1}\lambda M \end{cases}$$

加以不同的初值条件，便可得出不同的相轨迹，着色以 *Lyapunov* 函数：$\phi(n,m) = \frac{\mu}{2M^2}(n^2 + m^2) - \frac{\lambda\mu}{M}(n+m) + \frac{mn}{M^2}$ 即可得出每一个点的稳定性，标记于图即可。

复现过程我选取了以 240 为步长绕以 (0,0),(1200,0),(1200,1200),(0,1200) 四点为顶点的方形区域作为初值（由于相空间的性质，在方形区域内取初值意义不大），分别以 *Lyapunov* 着色。

实现方式为先以 c 语言实现与原论文相同的改进欧拉法函数，再将每一个初值作为条件进行迭代，对于每一次迭代的结果都计算出该店的 *Lyapunov*

函数数值，并把 $,t,n,m,Lyapunov$ 数值写入二进制文件，最后用 matlab 进行二进制文件读取，读取为 5*n 的数组，用行向量进行着色以及画图，并按照时间顺序用函数插入箭头，为了美观进行了箭头放大，但因此由于实际大小并非取箭头的大小，所以会造成一定的角度便宜，不过结合图片还能判断相流方向。其中，颜色偏向深蓝色证明稳定性强，绿色代表稳定性差。方便起见，t 取 (0,200),dt=0.01，使用散点图作图。

最终成果如图1所示，引用见下

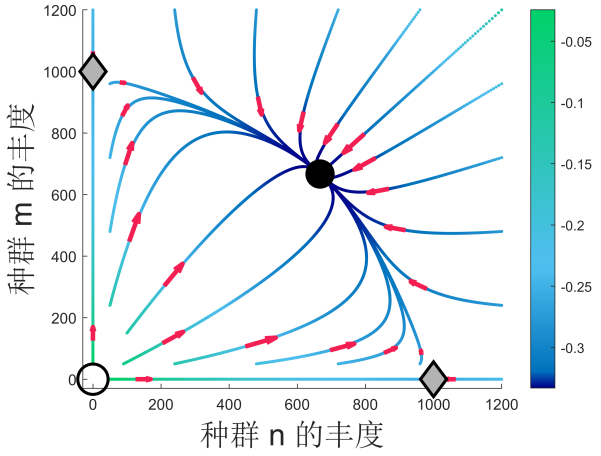


图 1*：对于论文 FIG.1 的复现，表示在没有外场的情况下，丰度动力学和式 (2) 的不动点。其中 2 个灰色填充菱形为鞍点，和黑色填充圆为稳定点，黑色圆环为不稳定点。箭头的颜色表示 *Lyapunov* 函数的值，(此处选取的 *Lyapunov* 函数为 $\phi(n,m)=\frac{\mu}{2M^2}(n^2+m^2)-\frac{\lambda\mu}{M}(n+m)+\frac{mn}{M^2}$). 参数选取 $M=2000,\lambda_b=0.6,\lambda_d=0.1,\mu=2.$

其，还可以做得 *Lyapunov* 函数在全空间的图像大致如下

5.1.2 FIG2 复现

对于观察原论文 FIG2（见右）

可见该图片实质上由数值解和解析解两部分组成，解析解为虚线，数值解为散点。因变量 χ_D 在 μ,ω 取不同初值时，均满足 Eq.7

$$\chi_D(\omega,\mu)=Re[\lim_{\delta h\to 0}\frac{\delta N_{||}(\omega)}{\delta h(\omega)}]=\frac{\sqrt{2}\lambda^2 M}{(\mu+1)^2}\frac{\tau_{||}}{1+\omega^2\tau_{||}^2}$$

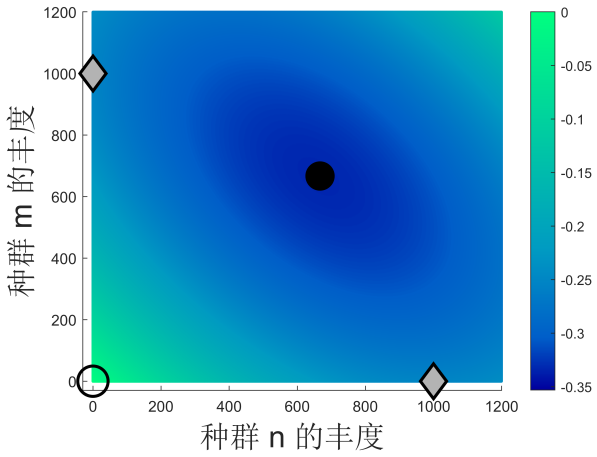


图 9. *Lyapunov* 函数全空间示意图

首先讨论解析解，其中，$\tau_{||}$ 满足

$$\tau_{||}^{-1}=\lambda(\mu-1)/(\mu+1)$$

带入上式，即可得出 χ_D 解析表达式。

接下来讨论数值解，同样在满足 Eq.7 的情况下，需要解出 $\tau_{||}$ 才能得到 χ_D 的数值，然而观察 Eq.5

$$\begin{cases}\dot{\delta_{||}}=\sqrt{2}h_{||}-\frac{\mu-1}{\mu+1}\lambda\delta_{||}\\ \dot{\delta_{\perp}}=-\lambda\delta_{\perp}\end{cases}$$

显然，平行分量的方程的解为

$$\delta_{||}(t)=\delta_0 sin(\omega t-\sigma)$$

形式，解析得到

$$\delta_0=\frac{\sqrt{2}\lambda^2}{(\mu+1)^2}\frac{\epsilon\tau_{||}}{\sqrt{1+\omega^2\tau_{||}^2}}$$

即可通过该方程求出 $\tau_{||}$

δ_0 的大小实质上就是 Eq.5 的振幅长时平均，为了增加效率，复现过程仅取后 0.5% 进行振幅平均。判定振幅方式为前后导数相乘为异号。

数值解的复现过程我选取了 200000 作为最大截止时间，步长选取 0.1，取后 0.5% 选取极值点进行极值平均，以此作为 $\bar{\delta_0}$，并由 $\bar{\delta_0}$ 求出 $\bar{\chi_D}$.

至于不断变换的 μ,ω 取值，我采取循环的方式进行，美观起见作为自变量时取 10 个等间距数值，写入得到二进制文件，用 matlab 读取画图。

至于解析解，我按照论文的形式直接通过复合函数的方式进行求出各点的数值解，写入文件作图。

最终成果如图3所示，引用见下

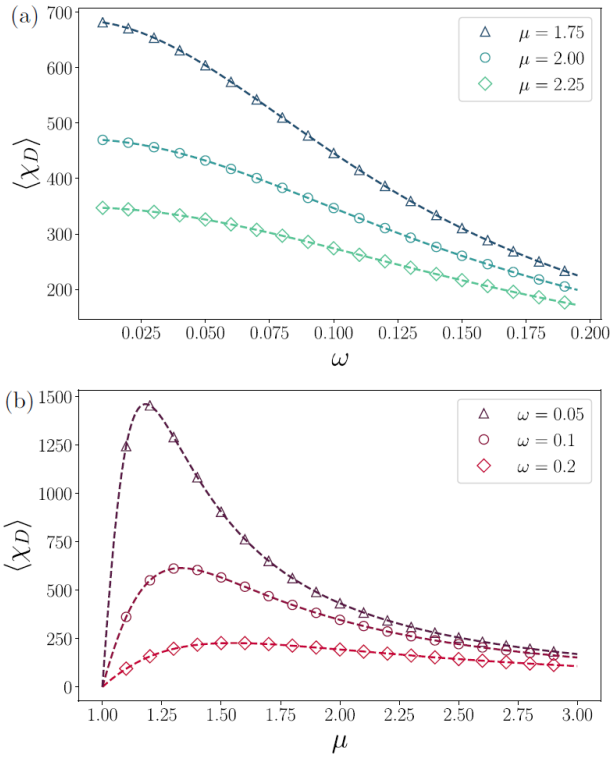


图 6*：论文原图 FIG.2

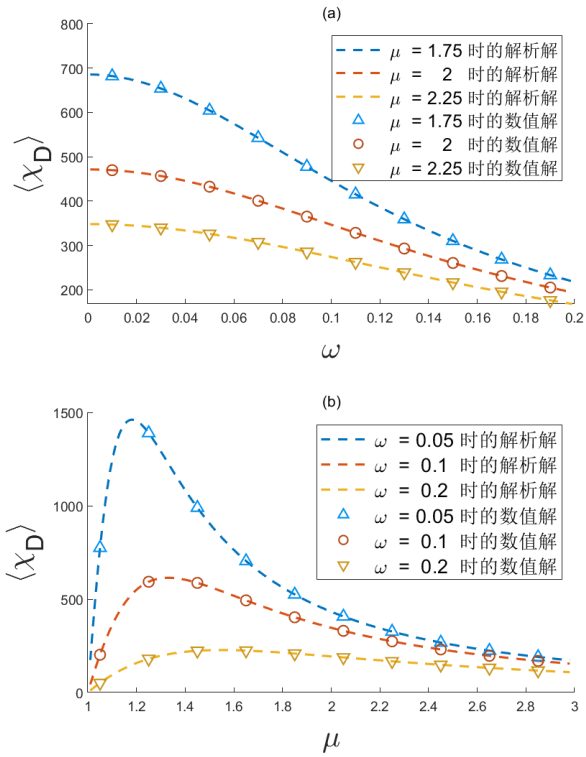


图 3*：对于论文 FIG.2 的复现，χ_D 的时间均值 $\langle \chi_D \rangle$ 在图 (a)ω 为自变量，μ=1.75, 2.00, 2.25；图 (b)μ 为自变量，ω=0.05, 0.1, 0.2。(a) 和 (b) 中的虚线和标记符号分别表示理论和数值结果，显示出很好的一致性。参数选取 $M = 2000, \lambda_b = 0.6, \lambda_d = 0.1, \mu = 2, \epsilon = 0.01$.

5.1.3 FIG3 复现

对于观察原论文 FIG3（见右）

图片实际为两条最概然灭绝路径（即四维空间内的异宿轨道）在 p_1, p_2 平面和 q_2, q_2 平面上的投影以及 pq 投影空间的迭加图。显然，任意时刻均满足主方程 Eq.12(方程过长，在此不引用)

$$\chi_D(\omega, \mu) = Re[\lim_{\delta h \to 0} \frac{\delta N_{||}(\omega)}{\delta h(\omega)}] = \frac{\sqrt{2}\lambda^2 M}{(\mu+1)^2} \frac{\tau_{||}}{1+\omega^2\tau_{||}^2}$$

按照论文的思路，问题实质是一个寻找固定鞍点之间的异宿轨道问题，关于逃离鞍点的过程是一个非常有意思的话题，将会在下一部分有所讨论，此处仅讨论论文中提出的方法，即 *Chernykh – Stepanov iteration method* .

按照文章的思路求解方程的大概为，先由 $q_!, q_2$ 反向求解 Eq.12(η_1 走向 η_c) 得到 p_1, p_2，再 p_1, p_2 由正向求解 Eq.12(η_c 走向 η_1)，即可得到轨迹。但是在实际操作中，我发现如此方法对于不好选取的初值会出现发散状态（即逃离鞍点梯度极大路径）. 同样，一下讨论以 η_1 走向 η_c 的异宿轨道 τ_1 来进行讨论 (可以证明 τ_1 与 τ_2 之间的对称性).

逃离鞍点是流体力学与神经网络中的一个常见问题，原文章引用了两篇关于逃离鞍点的流体文章 [3][4]，在查阅这两篇文章以及其他相关文章 [5][6]、更加深入的了解 *Chernykh – Stepanov iteration method* 之后，意识到需要先进行初次试探：在微扰下从 η_1 出发，以确定性数值达到 η_c，得到该确定性异宿轨道的相关参数，取有效值 q_1, q_2 进行后续迭代，之后再用 *Chernykh – Stepanov* 迭代方法，以此按照目的与需求带入 q_1, q_2 或 p_1, p_2 的数值，按照论文给出的方法继续迭代 (C-S 迭代示意图见下页)。

顺利的是，以上方法有着极好的收敛速度，在微扰仅仅在 q_1 方向取 10^{-3} 时，迭代两次以后的迭代该变量已经小于 10^{-6}。

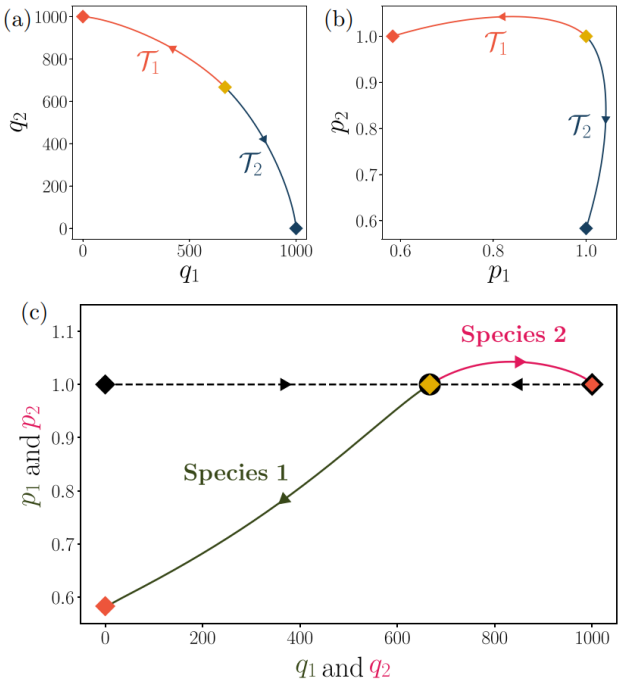
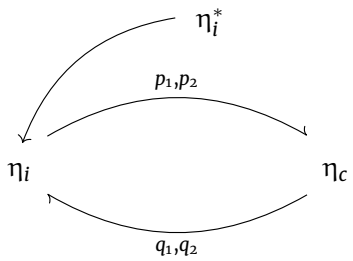


图 7*：论文原图 FIG.3



由以上方法，同样的构造函数迭代画图可以获得下图

如图2所示，引用见下

5.1.4 FIG4 复现

对于观察原论文 FIG4（见右）

图片由 χ_S 在同 μ,ω 下的散点图构成，其中 χ_S 与 μ,ω 的关系满足 Eq17.Eq14.,Eq15.,Eq17.

即

$$\begin{cases} H_p(t,t_0) = \frac{\sin\omega(t-t_0)}{\mu^2 M}(p_2-p_1)q_1q_2 \\ S_p = \min\limits_{t_0}\{\int_{-\infty}^{\infty} H_p(t,t_0)dt\} \\ \chi_S(\omega,\mu) = |\lim\limits_{\delta h\to 0}\frac{\delta \ln\tau_E}{\delta h}| = |S_p| \end{cases} \tag{22}$$

$$\chi_D(\omega,\mu) = Re[\lim_{\delta h\to 0}\frac{\delta N_{||}(\omega)}{\delta h(\omega)}] = \frac{\sqrt{2}\lambda^2 M}{(\mu+1)^2}\frac{\tau_{||}}{1+\omega^2\tau_{||}^2}$$

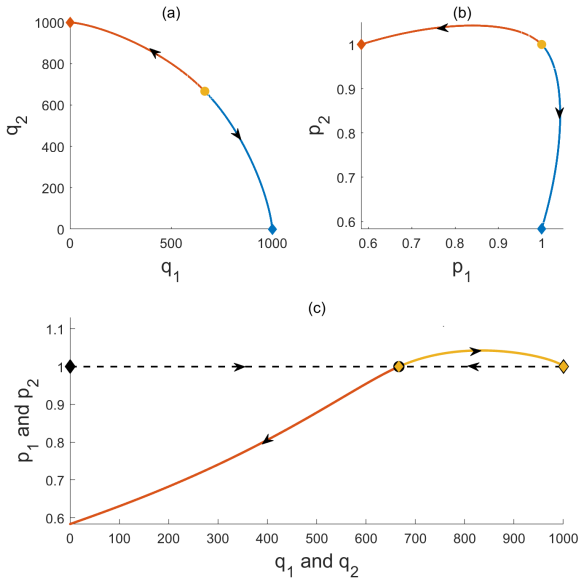


图 2*：对于论文 FIG.3 的复现，从共存状态 η_c 到灭亡状态 $\eta_i(i\in\{1,2\})$ 的最可能异宿轨道（η_i 为终点的异宿轨道标记为 τ_i）的相轨迹被投影在 (a) 坐标空间和 (b) 动量空间上。在图 (c) 中，$\tau_i(i\in\{1,2\})$ 被绘制在物种 1(橙色) 和物种 2(黄色) 的相空间中。虚线表示确定性情况（即由二维空间中从鞍点到稳定点的轨迹）. 参数 $M=2000,\lambda_b=0.6,\lambda_d=0.1,\mu=2.$

只需要求出函数 H_p 的积分即可，由于在 $(0,t_m)$ 以外，积分系数为 0，因此积分上下限可以转化为 0 与 t_m.

同样的，由于求解微分方程时步长固定，同时被积函数没有合适的解析形式，因此不能用龙贝格进行误差自适应积分，转而采取了梯形积分方法。

注意到：此处积分步长取的较小同时积分步长取的也较小，因此有 χ_S 的 SR 峰值较为尖锐的特点，和原论文并非完全一样.

由以上方法，同样的构造函数循环迭代写入读取文件画图可以获得下图（见下页）

如图4所示，引用见下

5.2 结果讨论

至此，成果复现出了文献中的所有图片，这些图片都有自己独特的物理意义，观察图1, 图2，可以看出零场与存在外场之时的运动性质区别，观察图3, 图2可以观察到明显的震荡图样。

至于图2，与文献的区别仅仅在于非峰值区域，与图片本身的物理意义无关，由此可以看出初值微

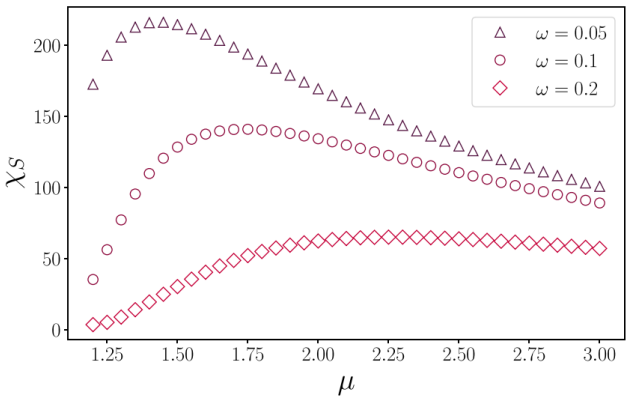


图 8*：论文原图 FIG.3

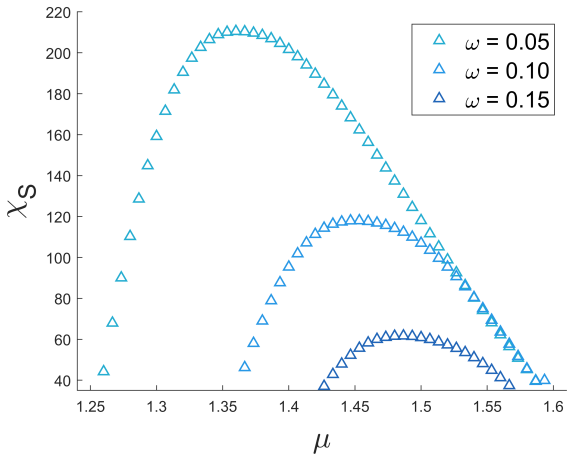


图 4*：对于论文 FIG.4 的复现，μ 为自变量，ω 分别为 0.05, 0.1, 0.2 的对数敏感性的 χ_S 如图所示。在每一个 ω 值中，可以观察到一个随机共振的峰。注意，此处和论文图没有完美贴合是由于取了积分步长更小以及更小的微扰导致的 SR 峰更加尖锐，更易于观察峰值所在，即随机共振的对应参数。参数选取 $M = 2000, \lambda_b = 0.6, \lambda_d = 0.1$.

扰合适的改变不会影响其震荡的存在与否，侧面证明了随机振动在周期弱场之中的普遍存在性.

6 问题分析与探究

基于本次课题，有不少有意思的问题值得讨论. 大致基于以下方面，很遗憾由于时间有限以及能力有限，大多问题仅能浅尝谈辄止.

6.1 方法选取

在本次课题中，方法选取部分宛如聒噪——改进欧拉和龙格库塔均可以十分方便的得到理想的数值解，因此绝大部分方法均为可用。

然而我认为在具体问题分析之前的方法选取仍然具有必要性。我认为在具体问题之前可以先选取简单算法进行尝试，如有不妥再行分析.

6.2 逃离鞍点的方法

之前说过逃离鞍点在机器学习以及流体力学有着广泛应用，在深度学习方面对于逃离鞍点主要有以下几种方法.

- 梯度下降算法 (GD)[11][12]
- 随机梯度下降法 (SGD)[11]
- 动量法 (Momentum)，加速动量法 (Nesterov)[12]
- 自适应学习方法 (Adagrad,Rmsprop,Adadelta,Adam,Amsgrad)[13]

在这里不做方法的讨论，仅仅了解其大都是基于周围路径积分以及惩罚自适应的机制混合而成的

图 10. 鞍点 (局部最佳) 与全局最佳示意图 (图源网络)

方法即可。

深度学习中同样常常陷入局部最优或者低维最优的情况，逃离鞍点的过程便显得尤为重要 [7][8]。

然而逃离鞍点，首先要确定驻点是鞍点，一般的，在物理问题中，稳定点都有较好的物理意义，通过这一点可以较好的判断；然而在很多非表层问题中，鞍点微扰法进行判断是最常用的。

在本课题中同样可以：进行方向不合适的微扰之后，很可能会有发散的结果，如图

毕竟按照鞍点的定义——不同方向稳定性不同的驻点，发散是很正常的事情。

确定完是鞍点之后，就可以进行一些迭代。

本文中采用的 Chernykh-Stepanov iteration method 较为简单，仅仅采用了最基本的迭代，其实质其实类似于改进梯度下降法，C-S 方法一个常

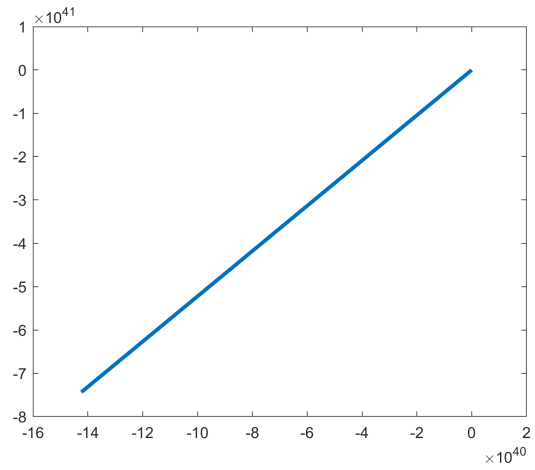


图 11. p_1, p_2 发散示意图

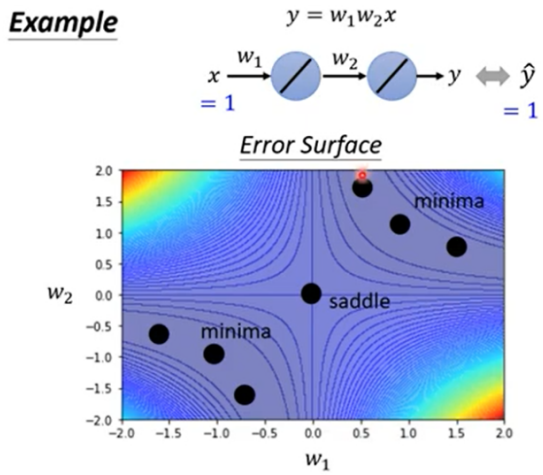


图 12. 鞍点判断示意图 (图源网络)

用的迭代表达式为

$$u^{(k+1)}(x,t) = \alpha u^{(k)}(x,t) + (1-\alpha)\tilde{u}^{(k)}(x,t) \quad (23)$$

其中 *alpha* 取令其结果较小的数值.

在大多数文章中，需要尝试不同的数值 *alpha* 来得到满意的结果 (同样类似于模型训练).

由于完全大致已知其像轨迹的轨迹 (知道终点起点与确定性轨迹)，SGD 方法或许可行 (相对于梯度下降与自适应监督学习)

同时，本课题中的鞍点是严格鞍点，根据引文，其迭代的稳定大致可以保证.[11]

不过由于与最近所学内容——微分方程数值解联系不够密切，暂且不做过多讨论，仅仅指出相通性并提出算法建议。

然而，关于逃离鞍点以及异宿轨道的选取仍然是一个热点话题，但是其难以构造一个普适的方法

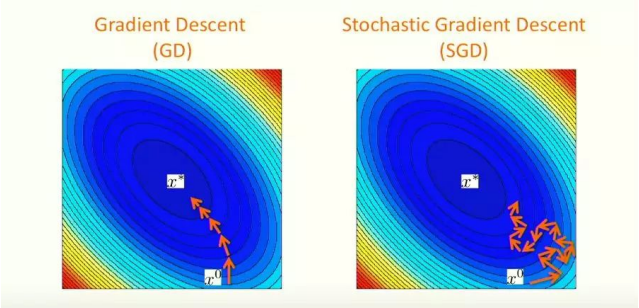


图 13. 梯度下降与随机梯度下降示意图 (图源网络)

高效实现. 故而本课题也仅仅是罗列一些常用的方法，就在最近两个月，谷歌提出了一种新的寻找全局最优的算法，在此希望能有更多高效算法的提出，为数值计算模拟以及其他相关方面提供更多选择.

6.3 写入二进制文件的速度提升

通览文献，发现其有大量迭代以及最小值选取，因此本文章选择使用了 c 语言 + 二进制文件存储的方式. 根据同学以及以往经验比较，fwrite 比 fprintf 能有效提升大约 50% 以上的效率，在写入更多的时候则更加明显。

查阅相关内容，发现在大文件写入方面 fwirte 确实占优，同时在 matlab 的文件读取方面，二进制文件的读取优势极大.

7 致谢

感谢兰州大学超算中心对本课题的支持，在计算 fig2 收敛周期以及 fig3 梯度下降、C-S 方法进行迭代，fig4 寻找最小值的过程中运用了中心部分资源，特此致谢。

8 参考文献

References

1. Park J I, Kim B J, Park H J. Stochastic resonance of abundance fluctuations and mean time to extinction in an ecological community[J]. Physical Review E, 2021, 104(2): 024133.
2. Iserles A. A first course in the numerical analysis of differential equations[M]. Cambridge university press, 2009.
3. Chernykh A I, Stepanov M G. Large negative velocity gradients in Burgers turbulence[J]. Physical Review E, 2001, 64(2): 026306.
4. Grafke T, Grauer R, Schäfer T. The instanton method and its numerical implementation in fluid mechanics[J]. Journal of Physics A: Mathematical and Theoretical, 2015, 48(33): 333001.
5. Grigorio L S, Bouchet F, Pereira R M, et al. Instantons in a Lagrangian model of turbulence[J]. Journal of Physics A: Mathematical and Theoretical, 2017, 50(5): 055501.
6. Grafke T, Grauer R, Schäfer T. Instanton filtering for the stochastic Burgers equation[J]. Journal of Physics A: Mathematical and Theoretical, 2013, 46(6): 062002.
7. Yu Y, Xu P, Gu Q. Third-order smoothness helps: Even faster stochastic optimization algorithms for finding local minima[J]. arXiv preprint arXiv:1712.06585, 2017.
8. Mokhtari A, Ozdaglar A, Jadbabaie A. Escaping saddle points in constrained optimization[J]. arXiv preprint arXiv:1809.02162, 2018.
9. Aji S M, McEliece R J. The generalized distributive law[J]. IEEE transactions on Information Theory, 2000, 46(2): 325-343.
10. Ahmed A, Aly M, Gonzalez J, et al. Scalable inference in latent variable models[C]//Proceedings of the fifth ACM international conference on Web search and data mining. 2012: 123-132.
11. Loshchilov I, Hutter F. Sgdr: Stochastic gradient descent with warm restarts[J]. arXiv preprint arXiv:1608.03983, 2016.
12. Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning[C]//International conference on machine learning. PMLR, 2013: 1139-1147.
13. Zeiler M D. Adadelta: an adaptive learning rate method[J]. arXiv preprint arXiv:1212.5701, 2012.
14. 张韵华, 奚梅成, 陈效群. 数值计算方法与算法[M]. 北京: 科学出版社, 2006.

A 程序实现

A.1 fig1 程序实现

•C 语言数据写入程序实现

```c
#include <stdio.h>
#include <math.h>
#define M (2000)
#define Mu (2)
#define Lambda (0.5)
#define Epsilon (0)
#define Omega (0)

double h(double t){
    return Epsilon*sin(Omega*t);
}

double f1(double t,double n,double m){
    return n*(Lambda-(n+(m/(Mu+h(t))))/M);
}
double f2(double t,double n,double m){
    return m*(Lambda-(m+n/(Mu-h(t)))/M);
}

void RK_4(double t,double n,double m,double tm,double h,FILE *fp){
    double k11,k12,k13,k14,k21,k22,k23,k24,fai = 0;
    while(t <= tm ){
        fai = Mu*(n*n+m*m)/2/M/M-Lambda*Mu/M*(n+m)+n*m/M/M;
        fwrite(&t, sizeof(double) , 1, fp );
        fwrite(&n, sizeof(double) , 1, fp );
        fwrite(&m, sizeof(double) , 1, fp );
        fwrite(&fai, sizeof(double) , 1, fp );
        k11 = f1(t,n,m);
        k21 = f2(t,n,m);
        k12 = f1(t+h/2,n+h/2*k11,m+h/2*k21);
        k22 = f2(t+h/2,n+h/2*k11,m+h/2*k21);
        k13 = f1(t+h/2,n+h/2*k12,m+h/2*k22);
        k23 = f2(t+h/2,n+h/2*k12,m+h/2*k22);
        k14 = f1(t+h  ,n+h*k13  ,m+h*k23);
        k24 = f2(t+h  ,n+h*k13  ,m+h*k23);
        n += h/6*(k11+2*k12+2*k13+k14);
        m += h/6*(k21+2*k22+2*k23+k24);
        t += h;
```

```
    }
}

int main(){
    int N = 240;
    FILE *fp;
    fp = fopen( "fig1_rk4.bin" , "w" );
    int i,j;
    for (i = 0;i<1200;i=i+N){
        RK_4(1,50,i,200,0.02,fp);
        RK_4(1,1200,i,200,0.02,fp);
        RK_4(1,i,50,200,0.02,fp);
        RK_4(1,i,1200,200,0.02,fp);
    }
    RK_4(1,90,50,200,0.02,fp);
    RK_4(1,100,150,200,0.02,fp);
    RK_4(1,1200,1200,200,0.02,fp);
    printf("\n%d\n",2+(i/N)*(4));
    fclose(fp);

    return 0;
}
```

- **Matlab** 图片生成程序实现

```
way = 'D:/Code/C/level2/work3/fig1_rk4.bin';

fileID1=fopen(way,'r');
A = fread(fileID1,[4 inf],'double');
fclose(fileID1);
N = 9950;

figure;
hold on

for i=0:22
    paint(A,1+N*i,N*i+N)
    i;
end
scatter(1000,0,300,"d", ...
    'MarkerEdgeColor',"black",'MarkerFaceColor',[.7,.7,.7],"LineWidth",2)
scatter(0,1000,300,"d", ...
    'MarkerEdgeColor',"black",'MarkerFaceColor',[.7,.7,.7],"LineWidth",2)
scatter(666.7,666.7,300,"o", ...
```

```
    'MarkerEdgeColor',"black",'MarkerFaceColor',"black","LineWidth",2)
scatter(0,0,400,"o", ...
    'MarkerEdgeColor',"black","LineWidth",2)

xlim([-30 1200])
ylim([-30 1200])
hold off

xlabel('种群 n 的丰度',"FontSize",20)
ylabel('种群 m 的丰度',"FontSize",20)
colorbar

function paint(A,x1,x2)
scatter(A(2,x1:x2),A(3,x1:x2),4,(A(4,x1:x2)),'filled')
dist = sqrt((A(2,x1:x2)-666.7).^2 + (A(3,x1:x2)-666.7).^2);
s.AlphaData = dist;
s.MarkerFaceAlpha = 'flat';
quiver(A(2,100+x1),A(3,100+x1),A(2,100+x1+2)-A(2,100+x1),A(3,100+x1+2)-A
    (3,100+x1),"MaxHeadSize",100,"MarkerSize",100,"LineWidth",3,"
    AutoScaleFactor",20,"Color",[0.6350 0.0780 0.2140]*1.5)
end
```

A.2 fig2 程序实现

•c 语言数据写入程序实现

```
#include <stdio.h>
#include <math.h>

#define M (2000)
#define Lambda (0.5)
#define Epsilon (0.01)
#define L1 (0.6)
#define L2 (0.1)

double hh(double t,double Omega){
    return Epsilon*sin(Omega*t);
}

double f(double Omega,double t,double x,double mu){
    double hp = hh(t,Omega)*Lambda*Lambda/(mu+1)/(mu+1);
    return sqrt(2)*hp-(mu-1)*x*Lambda/(mu+1);
}

```

```
19 void figaa(double mu,double Omega,double t,double x,double tm,double h,FILE*
       fp){
20     double k1,k2,k3= 0;
21     double tau =  (mu+1)/Lambda/(mu-1);
22     double max = 0;
23     double num = 0;
24     double chid =0;
25     while(t <= tm ){
26         k1 = f(Omega,t,x,mu);
27         k2 = f(Omega,t+0.5*h , x + 0.5*h*k1,mu);
28         k3 = f(Omega,t+h , x -h*k1 + h*2*k2,mu);
29         x += h*(k1+4*k2+k3)/6;
30         t += h;
31         chid = M*x/Epsilon/sqrt(1+Omega*Omega*tau*tau);
32         if ((t>tm*0.995)&&(f(Omega,t,x,mu)*f(Omega,t+h,x+ h * f(Omega,t,x,mu)
               ,mu)<0)) {
33             max += fabs(chid);
34             num++;
35             }
36         chid = max/num;
37     }
38 
39     fwrite(&Omega, sizeof(double) , 1, fp );
40     fwrite(&chid, sizeof(double) , 1, fp );
41 }
42 
43 void figbb(double mu,double Omega,double t,double x,double tm,double h,FILE*
       fp){
44     double k1,k2,k3= 0;
45     double tau =  (mu+1)/Lambda/(mu-1);
46     double max = 0;
47     double num = 0;
48     double chid =0;
49     while(t <= tm ){
50         k1 = f(Omega,t,x,mu);
51         k2 = f(Omega,t+0.5*h , x + 0.5*h*k1,mu);
52         k3 = f(Omega,t+h , x -h*k1 + h*2*k2,mu);
53         x += h*(k1+4*k2+k3)/6;
54         t += h;
55         chid = M*x/Epsilon/sqrt(1+Omega*Omega*tau*tau);
56         if ((t>tm*0.999)&&(f(Omega,t,x,mu)*f(Omega,t+h,x+ h * f(Omega,t,x,mu)
               ,mu)<0)) {
57             max += fabs(chid);
```

```
58              num++;
59              }
60          chid = max/num;
61      }
62
63      fwrite(&mu, sizeof(double) , 1, fp );
64      fwrite(&chid, sizeof(double) , 1, fp );
65  }
66
67  void figa(double Mu ,FILE *fp){
68      double tau =  (Mu+1)/(Lambda*(Mu-1));
69      for (int i =1;i<200;i++){
70          double omega = i/1000.;
71          double xs = (sqrt(2)*Lambda*Lambda*M/(Mu+1)/(Mu+1)* tau)/(1+omega*
                omega*tau*tau);
72          fwrite(&omega, sizeof(double) , 1, fp );
73          fwrite(&xs, sizeof(double) , 1, fp );
74      }
75      fwrite(&Mu, sizeof(double) , 1, fp );
76      fwrite(&Mu, sizeof(double) , 1, fp );
77
78  }
79
80  void figb(double omega ,FILE *fp){
81
82      for (int i =100;i<299;i++){
83          double Mu = i/100.;
84          double tau =  (Mu+1)/(Lambda*(Mu-1));
85          double xs = (sqrt(2)*Lambda*Lambda*M/(Mu+1)/(Mu+1)* tau)/(1+omega*
                omega*tau*tau);
86          fwrite(&Mu, sizeof(double) , 1, fp );
87          fwrite(&xs, sizeof(double) , 1, fp );
88      }
89      fwrite(&omega, sizeof(double) , 1, fp );
90      fwrite(&omega, sizeof(double) , 1, fp );
91  }
92
93  int main(){
94      int N = 240;
95      FILE *fp;
96      fp = fopen( "fig2.bin" , "w" );
97      figa(1.75,fp);
98      figa(2,fp);
```

```
    figa(2.25,fp);
    figb(0.05,fp);
    figb(0.1,fp);
    figb(0.2,fp);

    for (int j = 0;j<3;j++){
        double mu = 1.75+0.25*j;
    for (int i = 0;i<10;i++){
        double omega = 0.01+i*0.02;
        figaa(mu,omega,0,1,200000,0.1,fp);
    }}

    for (int j = 0;j<3;j++){
        double omega;
        if (j==0) omega = 0.05;
        if (j==1) omega = 0.1;
        if (j==2) omega = 0.2;
    for (int i = 0;i<10;i++){
        double mu = 1.05+i*0.2;
        figbb(mu,omega,0,1,200000,0.1,fp);
    }}

    fclose(fp);
    return 0;
}
```

- **Matlab** 图片生成程序实现

```
way1 = 'D:/Code/C/level2/work3/fig2.bin';

fileID1=fopen(way1,'r');

A = fread(fileID1,[2 inf],'double');

fclose(fileID1);

figure;
subplot(2,1,1)
title('(a)')
hold on;
for i=0:2
    plot(A(1,1+i*200:200*i+199),A(2,1+i*200:199+i*200),"--","LineWidth",1.5)
end
scatter(A(1,1201:1210),A(2,1201:1210),50,"^","MarkerEdgeColor
```

```
    ",[0,114,189]/200,"LineWidth",1)
17 scatter(A(1,1201+10:1210+10),A(2,1201+10:1210+10),50,"o","MarkerEdgeColor
    ",[217,83,25]/300,"LineWidth",1)
18 scatter(A(1,1201+20:1210+20),A(2,1201+20:1210+20),50,"v","MarkerEdgeColor
    ",[237,177,32]/300,"LineWidth",1)
19
20 legend(strcat(" \mu  = ",num2str(A(1,200))," 时的解析解"), ...
21     strcat(" \mu  =    ",num2str(A(1,400))," 时的解析解"), ...
22     strcat(" \mu  = ",num2str(A(1,600))," 时的解析解"), ...
23     strcat(" \mu  = ",num2str(A(1,200))," 时的数值解"), ...
24     strcat(" \mu  =    ",num2str(A(1,400))," 时的数值解"), ...
25     strcat(" \mu  = ",num2str(A(1,600))," 时的数值解"), ...
26     "FontSize",12)
27 xlabel("\omega","FontSize",20)
28 ylabel("\langle\chi_D\rangle","FontSize",20)
29 hold off
30
31 subplot(2,1,2)
32 title('(b)');
33 hold on;
34 for i=3:5
35     plot(A(1,1+i*200:200*i+199),A(2,1+i*200:199+i*200),"--","LineWidth",1.5)
36 end
37 scatter(A(1,1231:1240),A(2,1231:1240),50,"^","MarkerEdgeColor
    ",[0,114,189]/200,"LineWidth",1)
38 scatter(A(1,1231+10:1240+10),A(2,1231+10:1240+10),50,"o","MarkerEdgeColor
    ",[217,83,25]/300,"LineWidth",1)
39 scatter(A(1,1231+20:1240+20),A(2,1231+20:1240+20),50,"v","MarkerEdgeColor
    ",[237,177,32]/300,"LineWidth",1)
40
41 legend(strcat(" \omega  = ",num2str(A(1,800))," 时的解析解"), ...
42     strcat(" \omega  =  ",num2str(A(1,1000))," 时的解析解"), ...
43     strcat(" \omega  =  ",num2str(A(1,1200))," 时的解析解"), ...
44     strcat(" \omega  = ",num2str(A(1,800))," 时的数值解"), ...
45     strcat(" \omega  =  ",num2str(A(1,1000))," 时的数值解"), ...
46     strcat(" \omega  =  ",num2str(A(1,1200))," 时的数值解"), ...
47     "FontSize",13)
48 xlabel("\mu","FontSize",20)
49 ylabel("\langle\chi_D\rangle","FontSize",20)
50 hold off
```

A.3 fig3 程序实现

- c 语言数据写入程序实现

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define M (2000)
#define Mu (2)
#define Lambda (0.5)
#define Epsilon (0)
#define Omega (0)
#define L1 (0.6)
#define L2 (0.1)

double ht(double t){
    return Epsilon*sin(Omega*t);
}

double* f(double t,double A[],int nn){
    double* B=(double*)malloc(sizeof(double)*nn);
    double q1=A[0];
    double q2=A[1];
    double p1=A[2];
    double p2=A[3];
    B[0] = L1*(2*p1-1)*q1 - L2*q1 - 1*(2*p1-1)*q1*q1/M - 1*(2*p2-1)*q2*q1/Mu/
       M;
    B[1] = L1*(2*p2-1)*q2 - L2*q2 - 1*(2*p2-1)*q2*q2/M - 1*(2*p1-1)*q1*q2/M/
       Mu;
    B[2] = L1*(1-p1)*p1 + L2*(p1-1) + 2*(p1-1)*q1*p1/M + 1*(2*p1*p2-p1-p2)*q2
       /Mu/M;
    B[3] = L1*(1-p2)*p2 + L2*(p2-1) + 2*(p2-1)*q2*p2/M + 1*(2*p1*p2-p1-p2)*q1
       /Mu/M;
    return B;
}



double* make(double a,double b,double c,double d,int n){
    double* B=(double*)malloc(sizeof(double)*n);
    B[0] = a;
    B[1] = b;
    B[2] = c;
    B[3] = d;
    return B;
}
```

```
40
41 void improvedeuler(double* tem,double t,double q1,double q2,double p1 ,double
       p2,double tm,double h,FILE *fp){
42     int ii = 0;
43     while (t<tm){
44
45         fwrite(&t, sizeof(double) , 1, fp );
46         fwrite(&q1, sizeof(double) , 1, fp );
47         fwrite(&q2, sizeof(double) , 1, fp );
48         fwrite(&p1, sizeof(double) , 1, fp );
49         fwrite(&p2, sizeof(double) , 1, fp );
50
51         tem[0*20001+20000-ii] = t;
52         tem[1*20001+20000-ii] = q1;
53         tem[2*20001+20000-ii] = q2;
54         tem[3*20001+20000-ii] = p1;
55         tem[4*20001+20000-ii] = p2;
56
57         double q1_ = q1 + h*f(t,make(q1,q2,p1,p2,4),4)[0];
58         double q2_ = q2 + h*f(t,make(q1,q2,p1,p2,4),4)[1];
59         double p1_ = p1 + h*f(t,make(q1,q2,p1,p2,4),4)[2];
60         double p2_ = p2 + h*f(t,make(q1,q2,p1,p2,4),4)[3];
61
62         double q1__ = q1 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[0]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[0]);
63         double q2__ = q2 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[1]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[1]);
64         double p1__ = p1 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[2]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[2]);
65         double p2__ = p2 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[3]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[3]);
66
67         q1 = q1__;
68         q2 = q2__;
69         p1 = p1__;
70         p2 = p2__;
71         t += h;
72         ii += 1;
73     }
74 }
75
76
77 void improvedeuler2(double* tem,double t,double q1,double q2,double p1 ,
```

```
    double p2,double tm,double h,FILE *fp){
78   int ii = 0;
79   while (t<tm){
80       fwrite(&t, sizeof(double) , 1, fp );
81       fwrite(&q1, sizeof(double) , 1, fp );
82       fwrite(&q2, sizeof(double) , 1, fp );
83       fwrite(&p1, sizeof(double) , 1, fp );
84       fwrite(&p2, sizeof(double) , 1, fp );
85
86       tem[0*20001+ii] = t;
87       tem[1*20001+ii] = q1;
88       tem[2*20001+ii] = q2;
89       p1 = tem[3*20001+ii];
90       p2 = tem[4*20001+ii];
91
92       double q1_ = q1 + h*f(t,make(q1,q2,p1,p2,4),4)[0];
93       double q2_ = q2 + h*f(t,make(q1,q2,p1,p2,4),4)[1];
94       double p1_ = p1 + h*f(t,make(q1,q2,p1,p2,4),4)[2];
95       double p2_ = p2 + h*f(t,make(q1,q2,p1,p2,4),4)[3];
96
97       double q1__ = q1 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[0]+f(t,make(q1_,
             q2_,p1_,p2_,4),4)[0]);
98       double q2__ = q2 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[1]+f(t,make(q1_,
             q2_,p1_,p2_,4),4)[1]);
99       double p1__ = p1 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[2]+f(t,make(q1_,
             q2_,p1_,p2_,4),4)[2]);
100      double p2__ = p2 + h/2*(f(t,make(q1,q2,p1,p2,4),4)[3]+f(t,make(q1_,
             q2_,p1_,p2_,4),4)[3]);
101
102      q1 = q1__;
103      q2 = q2__;
104      p1 = p1__;
105      p2 = p2__;
106      t += h;
107      ii += 1;
108  }
109 }
110
111
112 void improvedeuler3(double* tem,double t,double q1,double q2,double p1 ,
    double p2,double tm,double h,FILE *fp){
113  int ii = 0;
114  while (t<tm){
```

```
115         tem[3*20001+20000-ii] = p1;
116         tem[4*20001+20000-ii] = p2;
117         q1 = tem[1*20001+20000-ii];
118         q2 = tem[2*20001+20000-ii];
119         fwrite(&t, sizeof(double) , 1, fp );
120         fwrite(&q1, sizeof(double) , 1, fp );
121         fwrite(&q2, sizeof(double) , 1, fp );
122         fwrite(&p1, sizeof(double) , 1, fp );
123         fwrite(&p2, sizeof(double) , 1, fp );
124 
125         double q1_ = q1 - h*f(t,make(q1,q2,p1,p2,4),4)[0];
126         double q2_ = q2 - h*f(t,make(q1,q2,p1,p2,4),4)[1];
127         double p1_ = p1 - h*f(t,make(q1,q2,p1,p2,4),4)[2];
128         double p2_ = p2 - h*f(t,make(q1,q2,p1,p2,4),4)[3];
129 
130         double q1__ = q1 - h/2*(f(t,make(q1,q2,p1,p2,4),4)[0]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[0]);
131         double q2__ = q2 - h/2*(f(t,make(q1,q2,p1,p2,4),4)[1]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[1]);
132         double p1__ = p1 - h/2*(f(t,make(q1,q2,p1,p2,4),4)[2]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[2]);
133         double p2__ = p2 - h/2*(f(t,make(q1,q2,p1,p2,4),4)[3]+f(t,make(q1_,
               q2_,p1_,p2_,4),4)[3]);
134 
135         q1 = q1__;
136         q2 = q2__;
137         p1 = p1__;
138         p2 = p2__;
139         t += h;
140         ii += 1;
141     }
142 }
143 
144 
145 int main(){
146     FILE *fp;
147     double tm = 200;
148     fp = fopen( "fig3_2.bin" , "w" );
149     double tem[5*20001] = {0};
150     improvedeuler(tem,0,1.e-3,1000,1,1,tm,0.01,fp);
151     fclose(fp);
152 
153 
```

```
154     for (int i = 0; i<10;i++){
155         fp = fopen( "fig3_1.bin" , "w" );
156         improvedeuler3(tem,0,0,1000,7/12.0,1,tm,0.01,fp);
157         fclose(fp);
158 
159         fp = fopen( "fig3_2.bin" , "w" );
160         improvedeuler2(tem,0,2000./3,2000./3,1,1,tm,0.01,fp);
161         fclose(fp);
162     }
163 
164     fp = fopen( "fig3_4.bin" , "w" );
165     improvedeuler(tem,0,1000,1.e-3,1,1,tm,0.01,fp);
166     fclose(fp);
167 
168 
169     for (int i = 0; i<10;i++){
170         fp = fopen( "fig3_3.bin" , "w" );
171         improvedeuler3(tem,0,1000,0,1,7/12.0,tm,0.01,fp);
172         fclose(fp);
173 
174 
175         fp = fopen( "fig3_4.bin" , "w" );
176         improvedeuler2(tem,0,2000./3,2000./3,1,1,tm,0.01,fp);
177         fclose(fp);
178     }
179 
180     return 0;
181 }
```

- **Matlab** 图片生成程序实现

```
1  way1 = 'D:/Code/C/level2/work3/fig3_1.bin';
2  way2 = 'D:/Code/C/level2/work3/fig3_2.bin';
3  way3 = 'D:/Code/C/level2/work3/fig3_3.bin';
4  way4 = 'D:/Code/C/level2/work3/fig3_4.bin';
5  
6  fileID1=fopen(way1,'r');
7  fileID2=fopen(way2,'r');
8  fileID3=fopen(way3,'r');
9  fileID4=fopen(way4,'r');
10 
11 
12 A = fread(fileID1,[5 inf],'double');
13 B = fread(fileID2,[5 inf],'double');
```

```
C = fread(fileID3,[5 inf],'double');
D = fread(fileID4,[5 inf],'double');

fclose(fileID1);
fclose(fileID2);
fclose(fileID3);
fclose(fileID4);
tiledlayout(2,2)
nexttile
title("(a)")
ylabel('q_2',"FontSize",13)
xlabel('q_1',"FontSize",13)
hold on;
plot(D(2,:),D(3,:),'LineWidth',2)
plot(B(2,:),B(3,:),'LineWidth',2)
scatter(666.666,666.666,66,"filled","o")
scatter(1000,0,66,[0 114 189]/255,"filled","d")
scatter(0,1000,66,[217 83 13]/255,"filled","d")
hold off

nexttile
title("(b)")
ylabel('p_2',"FontSize",13)
xlabel('p_1',"FontSize",13)
hold on;
plot(C(4,:),C(5,:),'LineWidth',2)
plot(A(4,:),A(5,:),'LineWidth',2)
scatter(1,1,66,[237 177 32]/255,"filled","o")
scatter(1,7/12,66,[0 114 189]/255,"filled","d")
scatter(7/12,1,66,[217 83 13]/255,"filled","d")
ylim([0.583 1.05])
xlim([0.583 1.05])
hold off


nexttile([1 2])
title("(c)")
ylabel('p_1 and p_2',"FontSize",13)
xlabel('q_1 and q_2',"FontSize",13)
hold on;
plot([0 1000],[1 1],"LineWidth",1.5,"LineStyle","--","Color",[0 0 0])
scatter(B(2,:),B(4,:),3)
scatter(D(2,:),D(4,:),3)
```

```
x1 = 5;
scatter(2000/3,1,100,[0 0 0],"filled","o")
scatter(2000/3,1,66,[237 177 32]/255,"filled","d")
scatter(0,1,100,[0 0 0],"filled","d")
scatter(1000,1,100,[0 0 0],"filled","d")
scatter(1000,1,66,[237 177 32]/255,"filled","d")
ylim([0.583 1.13])
hold off
```

A.4 fig4 程序实现

- c 语言数据写入程序实现

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define M (2000)
#define Lambda (0.5)
#define Epsilon (0)
#define L1 (0.6)
#define L2 (0.1)



double* f(double Mu,double t,double A[],int nn){
    double* B=(double*)malloc(sizeof(double)*nn);
    double q1=A[0];
    double q2=A[1];
    double p1=A[2];
    double p2=A[3];
    B[0] = L1*(2*p1-1)*q1 - L2*q1 - 1*(2*p1-1)*q1*q1/M - 1*(2*p2-1)*q2*q1/Mu/
       M;
    B[1] = L1*(2*p2-1)*q2 - L2*q2 - 1*(2*p2-1)*q2*q2/M - 1*(2*p1-1)*q1*q2/M/
       Mu;
    B[2] = L1*(1-p1)*p1 + L2*(p1-1) + 2*(p1-1)*q1*p1/M + 1*(2*p1*p2-p1-p2)*q2
       /Mu/M;
    B[3] = L1*(1-p2)*p2 + L2*(p2-1) + 2*(p2-1)*q2*p2/M + 1*(2*p1*p2-p1-p2)*q1
       /Mu/M;
    return B;
}
```

```
28 double* make(double a,double b,double c,double d,int n){
29     double* B=(double*)malloc(sizeof(double)*n);
30     B[0] = a;
31     B[1] = b;
32     B[2] = c;
33     B[3] = d;
34     return B;
35 }
36
37 void improvedeuler(double Mu,double* tem,double t,double q1,double q2,double
       p1 ,double p2,double tm,double h,FILE *fp){
38     int ii = 0;
39     while (t<tm){
40
41         fwrite(&t, sizeof(double) , 1, fp );
42         fwrite(&q1, sizeof(double) , 1, fp );
43         fwrite(&q2, sizeof(double) , 1, fp );
44         fwrite(&p1, sizeof(double) , 1, fp );
45         fwrite(&p2, sizeof(double) , 1, fp );
46
47         tem[0*20001+20000-ii] = t;
48         tem[1*20001+20000-ii] = q1;
49         tem[2*20001+20000-ii] = q2;
50         tem[3*20001+20000-ii] = p1;
51         tem[4*20001+20000-ii] = p2;
52
53         double q1_ = q1 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[0];
54         double q2_ = q2 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[1];
55         double p1_ = p1 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[2];
56         double p2_ = p2 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[3];
57
58         double q1__ = q1 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[0]+f(Mu,t,make(
               q1_,q2_,p1_,p2_,4),4)[0]);
59         double q2__ = q2 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[1]+f(Mu,t,make(
               q1_,q2_,p1_,p2_,4),4)[1]);
60         double p1__ = p1 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[2]+f(Mu,t,make(
               q1_,q2_,p1_,p2_,4),4)[2]);
61         double p2__ = p2 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[3]+f(Mu,t,make(
               q1_,q2_,p1_,p2_,4),4)[3]);
62
63         q1 = q1__;
64         q2 = q2__;
65         p1 = p1__;
```

```
66          p2 = p2__;
67          t += h;
68          ii += 1;
69      }
70  }
71
72
73  void improvedeuler2(double Mu,double* tem,double t,double q1,double q2,double
        p1 ,double p2,double tm,double h,FILE *fp){
74      int ii = 0;
75      while (t<tm){
76          fwrite(&t, sizeof(double) , 1, fp );
77          fwrite(&q1, sizeof(double) , 1, fp );
78          fwrite(&q2, sizeof(double) , 1, fp );
79          fwrite(&p1, sizeof(double) , 1, fp );
80          fwrite(&p2, sizeof(double) , 1, fp );
81
82          tem[0*20001+ii] = t;
83          tem[1*20001+ii] = q1;
84          tem[2*20001+ii] = q2;
85          p1 = tem[3*20001+ii];
86          p2 = tem[4*20001+ii];
87
88          double q1_ = q1 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[0];
89          double q2_ = q2 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[1];
90          double p1_ = p1 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[2];
91          double p2_ = p2 + h*f(Mu,t,make(q1,q2,p1,p2,4),4)[3];
92
93          double q1__ = q1 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[0]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[0]);
94          double q2__ = q2 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[1]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[1]);
95          double p1__ = p1 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[2]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[2]);
96          double p2__ = p2 + h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[3]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[3]);
97
98          q1 = q1__;
99          q2 = q2__;
100         p1 = p1__;
101         p2 = p2__;
102         t += h;
103         ii += 1;
```

```
104     }
105 }
106
107
108 void improvedeuler3(double Mu,double* tem,double t,double q1,double q2,double
        p1 ,double p2,double tm,double h,FILE *fp){
109     int ii = 0;
110     while (t<tm){
111         tem[3*20001+20000-ii] = p1;
112         tem[4*20001+20000-ii] = p2;
113         q1 = tem[1*20001+20000-ii];
114         q2 = tem[2*20001+20000-ii];
115         fwrite(&t, sizeof(double) , 1, fp );
116         fwrite(&q1, sizeof(double) , 1, fp );
117         fwrite(&q2, sizeof(double) , 1, fp );
118         fwrite(&p1, sizeof(double) , 1, fp );
119         fwrite(&p2, sizeof(double) , 1, fp );
120
121         double q1_ = q1 - h*f(Mu,t,make(q1,q2,p1,p2,4),4)[0];
122         double q2_ = q2 - h*f(Mu,t,make(q1,q2,p1,p2,4),4)[1];
123         double p1_ = p1 - h*f(Mu,t,make(q1,q2,p1,p2,4),4)[2];
124         double p2_ = p2 - h*f(Mu,t,make(q1,q2,p1,p2,4),4)[3];
125
126         double q1__ = q1 - h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[0]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[0]);
127         double q2__ = q2 - h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[1]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[1]);
128         double p1__ = p1 - h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[2]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[2]);
129         double p2__ = p2 - h/2*(f(Mu,t,make(q1,q2,p1,p2,4),4)[3]+f(Mu,t,make(
                q1_,q2_,p1_,p2_,4),4)[3]);
130
131         q1 = q1__;
132         q2 = q2__;
133         p1 = p1__;
134         p2 = p2__;
135         t += h;
136         ii += 1;
137     }
138 }
139
140
141
```

```
142 double Hp(double Mu,double *tem,double omega,double t0,double t,int ii){
143     double q1 = tem[1*20001+ii];
144     double q2 = tem[2*20001+ii];
145     double p1 = tem[3*20001+ii];
146     double p2 = tem[4*20001+ii];
147     return sin(omega*(t-t0))*(p2-p1)*q1*q2/Mu/Mu/M;
148 }
149 
150 int main(){
151     double omega=0;
152     FILE *fp2;
153     fp2 = fopen( "fig4.bin" , "w" );
154 
155 
156 
157     for (int o = 0;o<3;o+=1){
158         omega += 0.05;
159 
160     double Mu = 2;
161     for (double _ =180;_<240;_+=1)
162 {
163     Mu = _/150;
164 
165     FILE *fp;
166     double tm = 200;
167     double tem[5*20001] = {0};
168 
169     fp = fopen( "fig3_4.bin" , "w" );
170     improvedeuler(Mu,tem,0,1000,1.e-3,1,1,tm,0.01,fp);
171     fclose(fp);
172 
173 
174     for (int i = 0; i<3;i++){
175         fp = fopen( "fig3_3.bin" , "w" );
176         improvedeuler3(Mu,tem,0,1000,0,1,(Lambda+0.1*Mu)/(0.6*Mu),tm,0.01,fp)
                ;
177         fclose(fp);
178 
179         fp = fopen( "fig3_4.bin" , "w" );
180         improvedeuler2(Mu,tem,0,2000./3,2000./3,1,1,tm,0.01,fp);
181         fclose(fp);
182     }
183 
```

```
    double min = 10000;
    double I = 0;
    for (int j = 0;j<62;j++){
        double t0 = j/10.;
        for (int i = 0;i<20000;i++){
            I += Hp(Mu,tem,omega,t0,i/100.0,i)*0.01;
        }
        I =- I;
        if (I<min) min=I;
    }
    min = fabs(min);
    printf("%f,%f,%f,%f\n",omega,Mu,min,(Lambda+0.1*Mu)/(0.6*Mu));

    fwrite(&Mu, sizeof(double) , 1, fp2 );
    fwrite(&min, sizeof(double) , 1, fp2 );
    }
        }
    return 0;
}
```

•**Matlab** 图片生成程序实现

```
way1 = 'C:\Users\w3397\Downloads/fig4.bin';

fileID1=fopen(way1,'r');

A = fread(fileID1,[2 inf],'double');

fclose(fileID1);

figure;
hold on

scatter(A(1,1:60),A(2,1:60),"^","MarkerEdgeColor",[31,149,180]/220,"LineWidth
    ",1)
scatter(A(1,61:120),A(2,61:120),"^","MarkerEdgeColor",[31,119,180]/200,"
    LineWidth",1)
scatter(A(1,121:180),A(2,121:180),"^","MarkerEdgeColor",[31,99,180]/250,"
    LineWidth",1)
hold off

legend("\omega = 0.05","\omega = 0.10","\omega = 0.15","FontSize",15)
ylim([35 220])
xlim([1.24 1.61])
```

```
20
21 xlabel("\mu","FontSize",20)
22 ylabel("\chi_S","FontSize",20)
```

B Runge–Kutta 法有关证明（方法选取）

B.1 隐式 Runge–Kutta 法稳定性证明

此处仅仅考虑 **A** 稳定性（绝对稳定性）至于 **B** 稳定性（代数稳定性）在大多数情况下不用考虑，此处暂不标出

与显式方法相比，隐式 **Runge-Kutta** 方法的优势在于它们具有更大的稳定性，尤其是在应用于刚性方程时。考虑线性测试方程 $y' = \lambda y$. 应用于此方程的 **Runge-Kutta** 方法简化为迭代 $y_{n+1} = r(h\lambda)y_n$

$$r(z) = 1 + zb^T(I - zA)^{-1}e = \frac{\det(I - zA + zeb^T)}{\det(I - zA)},$$

其中 e 代表全为 **1** 的向量。函数 **r** 称为稳定性函数。可以得出，如果方法具有 **s** 级，则 **r** 是两个 **s** 次多项式的商。显式方法有一个严格的下三角矩阵 **A**，这意味着 **det** $(I - zA) = 1$ 并且稳定性函数是多项式。

如果，线性测试方程的数值解衰减为零，即 $r(z) \quad |< 1, z = h\lambda_0$ 这种的集合称为绝对稳定性域。特别是，如果所有具有 **Re** $(z) < 0$ 的都处于绝对稳定域中，则该方法称为**绝对稳定**。

显式 **Runge-Kutta** 方法的稳定性函数是多项式，因此显式 **Runge-Kutta** 方法永远不能是稳定的。

如果该方法具有阶 p ，则稳定性函数满足 $r(z) = e^z + O(z^{p+1})$ 如 $z \to 0$. 同时，给定次数的多项式的最接近指数函数。该式子被称为帕德多项式。如果多项式的分子是 **m** 阶，分子为 **n** 阶，满足稳定的条件是：

$$m \le n \le m + 2$$

s 阶算法的帕德多项式是 $2s$ 阶的，其可以满足 $m = n = s$ 。因此，该方法是是符合稳定条件的。任意高阶的隐式 *Runge–Kutta* 总存在满足 **A** 稳定性的 *Butcher* 表格。与之相对的，稳定线性多步法的阶数是有限的，通常不超过 **2**。

B.2 Runge–Kutta 法的推导举例

本处仅仅以四阶 *Runge–Kutta* 法中经典形式 **(1/6** 式**)** 举例进行推导，高阶方法与其他方法可思路相同不再赘述。

一般而言，**s** 阶 **Runge-Kutta** 方法可以写成：

$$\begin{cases} y_{t+h} = y_t + h \cdot \sum_{i=1}^{s} a_i k_i + \mathcal{O}(h^{s+1}) \\ k_i = y_t + h \cdot \sum_{j=1}^{s} \beta_{ij} f(k_j,\ t_n + \alpha_i h) \end{cases} \tag{24}$$

s = 4 时候，考虑起点、终点、末点，有

$$\begin{array}{cc} \alpha_i & \beta_{ij} \\ \alpha_1 = 0 & \beta_{21} = \frac{1}{2} \\ \alpha_2 = \frac{1}{2} & \beta_{32} = \frac{1}{2} \\ \alpha_3 = \frac{1}{2} & \beta_{43} = 1 \\ \alpha_4 = 1 & \end{array}$$

其他情况下

$$\beta_{ij} = 0$$

定义：

$$\begin{aligned} y^1_{t+h} &= y_t + hf(y_t,\ t) \\ y^2_{t+h} &= y_t + hf\left(y^1_{t+h/2},\ t + \frac{h}{2}\right) \\ y^3_{t+h} &= y_t + hf\left(y^2_{t+h/2},\ t + \frac{h}{2}\right) \end{aligned}$$

这里的

$$y^1_{t+h/2} = \frac{y_t + y^1_{t+h}}{2} y^1_{t+h/2} = \frac{y_t + y^1_{t+h}}{2}$$

$$y^2_{t+h/2} = \frac{y_t + y^2_{t+h}}{2} y^2_{t+h/2} = \frac{y_t + y^2_{t+h}}{2}$$

即，如果有

$$\begin{aligned} k_1 &= f(y_t,\ t) \\ k_2 &= f\left(y^1_{t+h/2},\ t + \frac{h}{2}\right) = f\left(y_t + \frac{h}{2}k_1,\ t + \frac{h}{2}\right) \\ k_3 &= f\left(y^2_{t+h/2},\ t + \frac{h}{2}\right) = f\left(y_t + \frac{h}{2}k_2,\ t + \frac{h}{2}\right) \\ k_4 &= f(y^3_{t+h},\ t + h) = f(y_t + hk_3,\ t + h) \end{aligned}$$

可以证明

$$
\begin{cases}
k_2 = f\left(y^1_{t+h/2},\ t+\dfrac{h}{2}\right) = f\left(y_t+\dfrac{h}{2}k_1,\ t+\dfrac{h}{2}\right) \\
\quad = f(y_t,\ t) + \dfrac{h}{2}\dfrac{d}{dt}f(y_t,\ t) \\
k_3 = f\left(y^2_{t+h/2},\ t+\dfrac{h}{2}\right) = f\left(y_t+\dfrac{h}{2}f\left(y_t+\dfrac{h}{2}k_1,\ t+\dfrac{h}{2}\right),\ t+\dfrac{h}{2}\right) \\
\quad = f(y_t,\ t) + \dfrac{h}{2}\dfrac{d}{dt}\left[f(y_t,\ t) + \dfrac{h}{2}\dfrac{d}{dt}f(y_t,\ t)\right] \\
k_4 = f(y^3_{t+h},\ t+h) = f\left(y_t+hf\left(y_t+\dfrac{h}{2}k_2,\ t+\dfrac{h}{2}\right),\ t+h\right) \\
\quad = f\left(y_t+hf\left(y_t+\dfrac{h}{2}f\left(y_t+\dfrac{h}{2}f(y_t,\ t),\ t+\dfrac{h}{2}\right),\ t+\dfrac{h}{2}\right),\ t+h\right) \\
\quad = f(y_t,\ t) + h\dfrac{d}{dt}\left[f(y_t,\ t) + \dfrac{h}{2}\dfrac{d}{dt}\left[f(y_t,\ t) + \dfrac{h}{2}\dfrac{d}{dt}f(y_t,\ t)\right]\right]
\end{cases}
\tag{25}
$$

是符合 $\mathcal{O}(h^2)\mathcal{O}(h^2)$ 的其中

$$
\frac{d}{dt}f(y_t,\ t) = \frac{\partial}{\partial y}f(y_t,\ t)\dot{y}_t + \frac{\partial}{\partial t}f(y_t,\ t) = f_y(y_t,\ t)\dot{y} + f_t(y_t,\ t) := \ddot{y}_t
$$

是 f 关于时间的导数

一般的，有

$$
\begin{aligned}
y_{t+h} &= y_t + h\left\{a\cdot f(y_t,\ t) + b\cdot\left[f(y_t,\ t) + \frac{h}{2}\frac{d}{dt}f(y_t,\ t)\right] + \right. \\
&\quad + c\cdot\left[f(y_t,\ t) + \frac{h}{2}\frac{d}{dt}\left[f(y_t,\ t) + \frac{h}{2}\frac{d}{dt}f(y_t,\ t)\right]\right] + \\
&\quad \left. + d\cdot\left[f(y_t,\ t) + h\frac{d}{dt}\left[f(y_t,\ t) + \frac{h}{2}\frac{d}{dt}\left[f(y_t,\ t) + \frac{h}{2}\frac{d}{dt}f(y_t,\ t)\right]\right]\right]\right\} + \mathcal{O}(h^5) \\
&= y_t + a\cdot hf_t + b\cdot hf_t + b\cdot\frac{h^2}{2}\frac{df_t}{dt} + c\cdot hf_t + c\cdot\frac{h^2}{2}\frac{df_t}{dt} + \\
&\quad + c\cdot\frac{h^3}{4}\frac{d^2f_t}{dt^2} + d\cdot hf_t + d\cdot h^2\frac{df_t}{dt} + d\cdot\frac{h^3}{2}\frac{d^2f_t}{dt^2} + d\cdot\frac{h^4}{4}\frac{d^3f_t}{dt^3} + \mathcal{O}(h^5)
\end{aligned}
$$

与泰勒级数进行比较 （其他截断误差判断方法均可）

$$
\begin{aligned}
y_{t+h} &= y_t + h\dot{y}_t + \frac{h^2}{2}\ddot{y}_t + \frac{h^3}{6}y^{(3)}_t + \frac{h^4}{24}y^{(4)}_t + \mathcal{O}(h^5) = \\
&= y_t + hf(y_t,\ t) + \frac{h^2}{2}\frac{d}{dt}f(y_t,\ t) + \frac{h^3}{6}\frac{d^2}{dt^2}f(y_t,\ t) + \frac{h^4}{24}\frac{d^3}{dt^3}f(y_t,\ t)
\end{aligned}
$$

得到一个约束如下

$$
\begin{cases}
a+b+c+d = 1 \\
\frac{1}{2}b + \frac{1}{2}c + d = \frac{1}{2} \\
\frac{1}{4}c + \frac{1}{2}d = \frac{1}{6} \\
\frac{1}{4}d = \frac{1}{24}
\end{cases}
$$

解得

$$
a = \frac{1}{6}, b = \frac{1}{3}, c = \frac{1}{3}, d = \frac{1}{6}
$$