基于 logit 模型与梯度下降法的银行信贷决策系统

摘要

银行对小微企业的信贷决策问题是一个多变量、高维度的优化问题。在确定其贷款利率和贷款金额的过程中，银行同时需要尽可能增大自己的收益，也需要通过有限的信息对其中可能出现的风险进行规避。

问题一 本题中我们通过 123 家企业的违约标志与发票信息中所包含的企业规模、盈利能力、供销网络与服务质量等信息，通过二元 logit 模型对企业实际的违约概率进行估计为 $\alpha_i = -1.043 \ln Q - 1.308 \ln r + 1.795 \cdot 10^{-4} \Omega_{up} - 6.623 \cdot 10^{-4} \Omega_{down} + 2.704 t_{up} + 0.241 t_{down} + 15.20$，并以此作为企业信贷风险的量化依据。在此违约率的判断基础之上，我们将去除了高风险企业的银行信贷决策系统写成了一个高维度的优化问题。在没有给定银行预算约束的基础上，可以将其看作无约束优化问题。我们在初始解的基础上，通过梯度上升法计算出其在全空间上的最优贷款金额总和 G 在七千万元左右。当预算约束 M≥G 时，银行可以取到全局最优的信贷策略；而当 M<G 时，则我们也提出了相应的两种放贷顺序，即梯度优先与履约率优先，前者能为银行带来最大的效用，而后者能够在简便的基础上为银行提供相对高的收益。

问题二 在确定了企业风险量化标准的基础上，我们将 302 家无信贷记录企业代入问题一得到的 logit 模型中，对其中企业的违约率进行了预测，并以此对其信誉等级进行了标记。而在本题的信贷决策中，需要解决有约束版本的优化问题。在此我们通过改进后的梯度下降算法对其最优的贷款分配方式进行了计算。但由于梯度下降法中可能面临的固定顺序与收敛较慢的问题，我们采用启发式算法中的 PSO 算法，对相关结果的准确性进行了进一步验证。最终，两类算法都将为信誉评级好的、违约率低的企业提供高额贷款，而将放弃对违约率高的企业提供贷款。而这也进一步验证了我们在问题一中为银行提供的信贷策略。

问题三 为了进一步优化银行的信贷决策，以应对市场的剧烈波动性，在此我们进一步考虑了不同的外生冲击对银行信贷决策的影响。我们将外生冲击分为两类：宏观的经济冲击与个别行业的供需冲击，并分别对其影响借贷的机制进行了识别。此后我们以新冠疫情为例，在正态冲击假设下对其带来的宏观冲击与个别行业冲击效应进行了衡量。发现新冠疫情的宏观冲击让只对科技企业带来了利好，其平均风险从 9.04% 下降到了 8.38%；而其它行业企业的违约率都各有上升。同时，宏观经济波动会影响在不同利率下银行潜在顾客的损失。最终会导致贷款相对向科技企业集中。而个别行业的波动会导致行业内企业违约率的相对上升，使银行贷款撤出相关领域。但除此之外相应波动对其他行业的影响较小。

关键字： logit 模型　高维凸优化　梯度下降法　PSO 算法　信贷决策

一、 问题重述

1.1 问题背景

伴随着我国信贷体系的逐渐完善和中国特色社会主义体制下私有制经济的不断发展，越来越多的银行开始推行面向小微企业的贷款。而由于小微企业体量较小、信息较为匮乏，又缺乏相应的抵押机制，对其贷款风险的衡量也就相对困难。因此，银行需要在放贷之前对企业进行更加详尽的分析，以在其放贷为小微企业提供融资便利的同时，也能够最大化自己的收益。

1.2 问题的相关信息

题目中已经为我们提供了以下的数据信息：

附件 1 中包括 123 家已经进行信贷的小微企业信息，包括其企业信誉评级、违约情况以及相应的票据信息；

附件 2 中包括 302 家将要接受银行审查决定是否放贷的企业信息，包括企业的基本信息和票据信息；

附件 3 给出了银行贷款年利率与潜在客户流失率之间的关系。

1.3 需要解决的问题

问题一：通过附件 1 中 123 家公司的信誉评级以及相关票据信息建立相应的数学模型，对各个企业的信用评级进行量化，并对银行对各企业在不同的信贷总额情况下的放贷决策进行分析；

问题二：通过在问题一中构建的数学模型对 302 家尚待进行贷款的企业信贷风险进行分析，并对在年度信贷总额为一亿元的条件下研究对各企业的信贷策略。

问题三：在上面研究过的信贷策略的基础上，考虑现实信贷过程中可能的突发因素和外生冲击，对问题二中的信贷策略进行调整和修正，并分析不同冲击的对银行信贷策略的作用机制。

二、 模型假设和变量说明

2.1 模型假设

1. 假设每家企业提供的发票信息均真实，不存在伪造或隐瞒的发票。

2. 假设利率具有严格正向的挤出效应，即利率与相应利率下损失的潜在顾客比例严格正相关。

3. 假设银行对贷款的收益风险中性，而对放贷过程中收入的不确定性风险厌恶。

4. 假设企业会尽可能追求数额高的贷款，贷款额越高对企业经营越有利。

5. 对同行业内部企业的各指标的外生冲击服从正态分布。

2.2 变量说明

变量符号	变量说明
α_l	企业 l 的违约概率
m_1, m_2	上、下游供应商总数量
η	企业损失潜在顾客比例
M	银行的借贷约束
ζ	银行的风险规避系数
i	银行的贷款利率
g	银行的贷款金额
s	梯度上升的学习率
i	贷款利率向量
g	贷款金额向量
t	算法迭代轮数
μ	正态冲击均值
σ	正态冲击方差

三、 问题一模型的建立与求解

3.1 问题分析与数据分析

本题的核心在于，在银行对 121 家企业的信誉评级基础上，进一步通过相关的企业交易信息，建立企业信贷风险的数量化衡量机制，以进一步对企业的违约风险进行衡量和更加精确的比较。而在辨明企业违约风险的基础上，将其纳入银行期望借贷收益及其风险的考量中，为银行在贷款利率、贷款金额以及贷款公司的选择上提供进一步指导与支持。

根据附件一中的企业信息，我们发现在 27 家信用评级为 A 的企业中，没有违约企业；38 家 B 级企业中有 1 家违约；34 家 C 级企业中有 2 家违约，而所有 D 级企业全部

违约。故我们建立的风险评估系统面临着两个基本要求：(1) 甄别出违约风险极高的企业 (D 类企业)；(2) 对其他违约几率较低的企业进行一定程度上的区分，而其中标注为 A 类信用级别的企业的违约风险应当控制在相对较低的水平。

由于缺少企业的资产、债务等财务数据以及更加精细的审计过程，难以通过传统的设计信用评分系统的手段对企业的偿债能力和盈利能力做出具体而准确的估计。故此时在缺少抵押品制度的信贷体系中，影响银行风险评估的核心因素是企业的违约与否，因此我们将违约率作为量化后风控体系的核心评判要素。

而通过分析附件中提供企业的往来发票挖掘出相应的企业特征和经营情况，可以发现其中的核心信息之一在于发票中包含了 123 家企业与 13961 家上游供货商以及 24449 家下游供货商之间的往来贸易记录。我们通过对其中有效发票的价值进行汇总，建立了企业之间的供应网络矩阵 A_{n*m1}, B_{n*m2}，部分展示如下：

表 **2**　部分企业与部分上游供货商之间的往来贸易额

	A00297	A05061	A01714
E1	0	336	21162
E2	674499.77	3427	36121
E3	864	0	288
E6	313600	200	528

另外，在每家企业的进项和销项发票中都存在不少负项发票。由此我们在计算企业的收益与成本时对相应的有效发票进行了抵扣，记第 l 家企业的退款总额为 T_l，并基于此对供应网络矩阵进行了修正。而在此并不需要对作废发票进行对应的处理，原因在于作废发票并未发货入账，并且很大可能是由于会计记账的过程中发生的错误产生的，故也较难体现出企业经营过程中面对的波动性。

3.2 企业特征的甄别

通过对数据的预处理和特征的简单识别后，我们认为其中有几项指标能够影响企业的偿贷能力和违约概率：

(1) 总进货额/销货额: 体现企业的规模，以反映企业的抗风险能力;

$$P_l = \sum_{i=1}^{m_1} a_{li}, \quad Q_l = \sum_{i=1}^{m_2} b_{li}$$

(2) 销货额与进货额之比: 体现企业的盈利能力;

$$r_l = \frac{\sum_{i=1}^{m_2} b_{li}}{\sum_{i=1}^{m_1} a_{li}}$$

(3) 加权度中心性：体现企业供应链的稳定与完善与销售渠道的广泛;

$$\Omega_l = \sum_{i=1}^{m} f_i c_{li}, f_i = \frac{c_{ki}}{\sum_{j=1}^{n} c_{ji}}, m \in \{m_1, m_2\}, c \in \{a, b\}$$

(4) 退款率：体现企业经营的稳定性和产品 (服务) 的质量;

$$t_l = \frac{T_l}{R_l}, R_l \in \{P_l, Q_l\}$$

基于经济学理论和经验直觉，我们认为前三项指标对违约率的提高起到抑制作用；而退款率上升则意味着企业的经营更不稳定，有更大的概率陷入违约的困境。

3.3 Logit 风险评估模型的建立与拟合

通过以上的数据特征，可以通过二值 logit 模型对各企业的违约概率进行拟合与估计。我们将“违约”标注为标签 1，“不违约”标注为标签 0，构造出以下形式的 logit 模型：

$$\alpha_i = \pi(x_i) + \epsilon_i \tag{1}$$

其中：

$$\pi(x_i) = \frac{\exp(x_i'\beta)}{1 + \exp(x_i'\beta)} \tag{2}$$

$$x_i'\beta = \beta_1 \ln Q + \beta_2 \ln r + \beta_3 \Omega + \beta_4 t + \beta_0 \tag{3}$$

由于企业的行业与规模不尽相同，指标 r 和 Q 之间的差距较大，我们通过对两者取对数的方式使 logit 回归结果更加合理。而回归后我们得到的风险评估模型为：

$$\begin{aligned}\alpha_i = &-1.043 \ln Q - 1.308 \ln r + 1.795 \cdot 10^{-4} \Omega_{up} - 6.623 \cdot 10^{-4} \Omega_{down} \\ &+2.704 t_{up} + 0.241 t_{down} + 15.20\end{aligned}$$

其中，下标”up” 代表上游供应商，而”down” 代表下游供应商。分析 logit 回归结果我们发现，Q,r 的指标与违约率之间具有很强的反相关性，这意味着公司的规模和盈利能力对其偿债能力的影响非常大。同样，退款率指标体现出与直觉统一的正趋势，且其中上游供应商的退货率对违约的影响更大，这很可能是由于原料的退货影响了企业的生产流程，进而对利润生成以及偿债的过程造成更大的影响。

而在中心性指标上，下游供应商的加权度中心性的方向与直觉相同，但上游供应商的相对密集却提高了企业的违约几率。这则可能是由于下游供应商的范围的扩张能够增强企业在市场中的影响力、创造更多利润；而上游供应商的广泛则意味着企业与原材料供应商之间的供需关系的不稳定。而这为其信贷风险带来了更多的不确定性因素。

将拟合结果与企业信用评级、违约情况做对照可以得到下图：

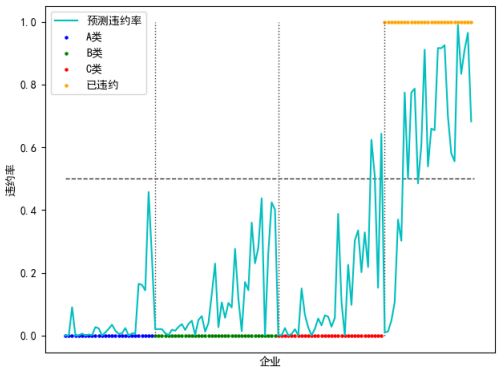

图 **1**　**logit** 模型预测值与信用评级、违约情况的关系

分析拟合结果可以发现：对于 A、B 类企业，通过 logit 回归预测出的违约率都相对较低，只有极少数由于规模等特殊原因呈现出较大的违约率 (在不剔除离群点的情况下，A 类的平均违约率小于 0.05，B 类的平均违约率小于 0.1；在剔除离群点的情况下，A 类的平均违约率约为 0.01，B 类的平均违约率约为 0.05)。尤其对于 B 类公司，其违约的波动性明显较 A 类公司高；对于 C 类公司，其违约的倾向和预测结果上波动性都较大，这也符合 C 类公司中存在部分违约公司的特征。而对于 D 类公司，可以发现预测出的违约率几乎全部大于 0.5(而在现实中这体现为 D 类公司全部违约)。这意味着对于本 logit 模型，银行应当将违约率高于 0.5 的公司全部标注为 D 类，并不向其放贷。

而对于其中 A、B、C 类中出现的少数波动率较高的企业，我们认为这是由于银行设计的评分系统与本题中通过发票数据建立的风险系统的评判方式中着重点设置的偏差导致的。而整体上信誉等级和违约率评分之间的关系还是稳定且合乎直觉的，因此我们认为其中的少量波动可以被接受。

3.4 银行效用函数与优化问题的确定

对于银行的效用函数，在存在违约不确定性的情况下，其至少应当由两个部分组成：一是银行通过利息赚取的期望收益，在此我们用 E 表示；二是由借贷过程中的信息不对称带来的道德风险带来的风险规避效用，在此我们用 D，即银行放贷收益的方差来表示。故基于均值-方差模型 [1]，银行的总体效用函数可以写作：

$$U(E, D) = E - \zeta D \tag{4}$$

对于银行的放贷收益 E，其受到银行的贷款利率 i，放贷金额 g，客户损失概率 η，以及企业的违约几率 α 的影响。由此可以表示出银行对企业 l 放贷的期望收益为：

$$E(i_l, g_l) = (1 - \eta_l)[(1 + i_l)(1 - \alpha_l)g_l - g_l] \tag{5}$$

而对于 D，基于上述的期望收益公式，易得到其表达式为：

$$D = (1 - \eta_l)^2(\alpha_l + (1 - \alpha_l)i_l^2)g_l^2 - (1 - \eta_l)^2(-\alpha_l + i - \alpha_l i_l)^2 g_l^2 \tag{6}$$

$$= (1 - \eta_l)^2 g_l^2 \alpha_l (1 - \alpha_l)(i_l + 1)^2 \tag{7}$$

在此基础上，而我们在数学上对不向违约率高于 0.5 的企业给予贷款提供相应的证明：对于 $\alpha > 0.5$ 的企业，有:

$$\frac{\partial D}{\partial \alpha} = \zeta(2\alpha - 1)(1 + i)^2(1 - \eta)^2 g^2 \tag{8}$$

当 $\alpha > 0.5$ 时，以方差衡量的风险反而会因为企业的违约可能性过大而降低。这会让 α 与效用之间呈现出无法确定的关系甚至正向的反直觉趋势。

而由附件三中的信息可以观察到，潜在客户的损失与企业的信誉类别之间具有因果关系。而通过对两者之间关系的观察，可以发现两者之间呈现明显的对数关系。

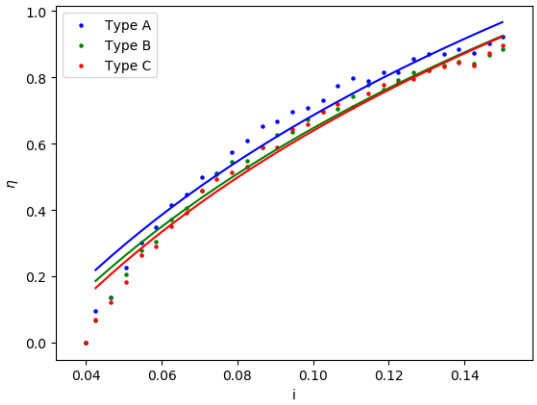

图 **2**　潜在客户损失率与贷款利率、企业信用评级的关系

而通过回归，则能够拟合出相应的对数函数曲线，其函数分别为：

$$\begin{cases} \eta_1 = & ln(12.876i + 0.697) \\ \eta_2 = & ln(12.265i + 0.682) \\ \eta_3 = & ln(12.450i + 0.649) \end{cases}$$

3.5 优化问题的求解与银行的信贷策略

综合 η 的性质，我们发现，潜在客户损失率与企业的信誉评级之间具有很大的关系。而这为我们寻找优化问题的初始解提供了很大的便利——可以为一类信誉评级的公司提供一致的利率。其原因有三：一方面，基于违约概率和信誉评级确定的利率及潜在客户损失率能大大减少银行在信贷过程中面对的调查与计算成本，且能够避免现实生活中因信息不对称对银行带来更大损失；另一方面，根据经济学的信贷理论，对于同一(风险) 类型的借贷人提供贷款时，只有在类型中设置利率相等的贷款才能为银行带来最高的收益 (即便会导致市场不出清) [2]；同时，在不同信誉类型基础上设置的利率能够对不同风险的借款企业进行一定程度上的识别，并对企业追求更高的信誉评级给予一定的激励。

因此，在求初始解的过程中，银行对某家企业贷款的效用函数亦可写作:

$$U(i_k, g_l) = (1-\eta_k)[(1+i_k)(1-\alpha_l)-1]g_l - \zeta(1-\eta_k)g_l^2\alpha_l(1-\alpha_l)(i_k+1)^2 \qquad (9)$$

其中，k 属于 A、B、C 三类信誉评级之一，而 $l \in [1, n]$。由此可以将银行的信贷决策进一步归纳为以下凸优化问题：

$$\max_{i_k, g_l} = \sum_{l=1}^{n} U(i_k, g_l)$$

$$s.t. \sum_{l=1}^{n} g_l \leq M$$

$$0.04 \leq i_k \leq 0.15$$

$$10^5 \leq g_l \leq 10^6$$

在得出优化问题的形式的基础下，可以发现其是一个有约束的超高维凸优化问题。在维度极高，且效用函数的偏导形式都较为复杂的前提下，传统的线性规划和凸优化方法的运用较为困难。而考虑到 i_k 与 g_l 分别与企业的信誉评级以及企业的个体差异相关，两者在数量级上差距较大，而 i 的未定元共只有三个，优化的过程较 g 明显更为简单，故我们先对 A 的分布进行讨论。

首先需要确定 i_A, i_B, i_C 的初始解。由于同样的信誉等级企业对应的 i 也相同，$\eta_k(i)$ 又是只与类别和利率有关的函数，故此时，U 中唯一与个体差异相关的参数为违约率 α_l。而一旦我们将违约率也控制为只与信誉等级相关，则对同一等级的公司提供的贷款数额也必定相同。

而重新观察上文计算出的 123 家公司的信誉等级分布与估计得的违约率，可以发现对 A、B、C 三类企业，除去特别大的离群值后，剩余企业违约率的平均值 $\bar{\alpha_1} < \bar{\alpha_2} < \bar{\alpha_3}$，且样本点之间也较为紧密地分布在均值周围。因此，我们用 $\bar{\alpha_k}$ 代替 $\bar{\alpha_l}$ 计算初始解。

经过测试，我们发现将 ζ 设置为 $5 * 10^{-7}$ 较为合理。这是因为 i 的范围在 0.04-0.15 之间，而 g 的范围则在六位数。E 作为 g 的一次函数，在数量级上 E 较 D 明显较小。而

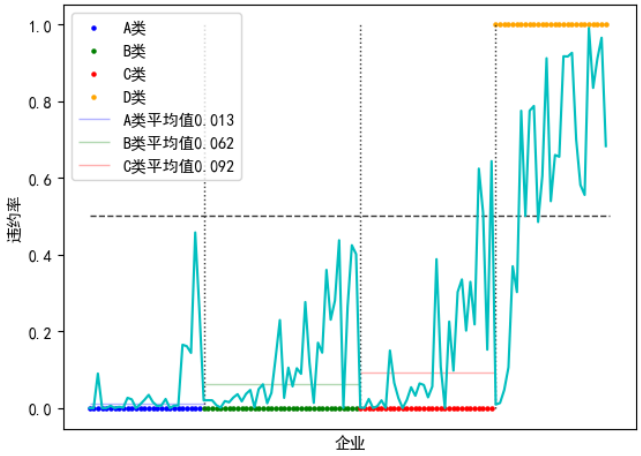

图 **3**　**123** 家企业的信誉等级分布与违约率平均值

这也会对 E 与 D 之间梯度的关系产生很大影响。通过数字规模上的差异比较，我们确定了 $\zeta = 5 * 10^{-7}$(与 g/i 数量级相当) 能够让两者梯度上的数值差距表现地尽量较小且比例相近，以帮助我们找到更好的信贷策略。

随后，我们通过在 i 与 g 的取值范围内的随机网格搜索，分别寻找对三个信誉等级企业 (银行在本策略下不向 D 类企业借贷，因此已经被排除，剩余 99 家企业) 的最优借贷策略。将 i 的抽样步长设置为 0.001%，g 的抽样步长设置为 100，得到的结果分别为：

表 **3**　网格搜索得到的初始解

	A 类企业	B 类企业	C 类企业
i	5.41%	9.15%	10.66%
g	966963.3	958954.4	784760.9

接下来我们基于找到的初始最优解，使用梯度上升法寻找目标函数的最大值。由于本题中没有设置借款约束上限，则先讨论当总效用函数最大时的银行贷款策略：

我们通过梯度上升法对最优化的 i 与 g 进行确定。首先，为了克服上面提到的梯度之间数量级的问题，我们先将 i 的 0.04-0.15 与 g 的 $1 * 10^5 - 1 * 10^6$ 投影到 0-1 进行归一化。相应地，我们对银行的风险规避系数 ζ 也由于归一化后数量级的一致转换为了 0.5，以适应两个函数之间数量级的改变。

随后，我们分别从初始点出发，沿其梯度方向，以 i 的学习率为 0.005、g 的学习率

为 0.01 进行迭代，其过程可以表示为:

$$\mathbf{i_{t+1}} = \mathbf{i_t} - s_i \nabla \mathbf{i_t} = \mathbf{i_t} - 0.005 \nabla \mathbf{i_t} \tag{10}$$

$$\mathbf{g_{t+1}} = \mathbf{g_t} - s_g \nabla \mathbf{g_t} = \mathbf{g_t} - 0.01 \nabla \mathbf{g_t} \tag{11}$$

经过 5000 轮迭代，我们得到了一组最优解如下：

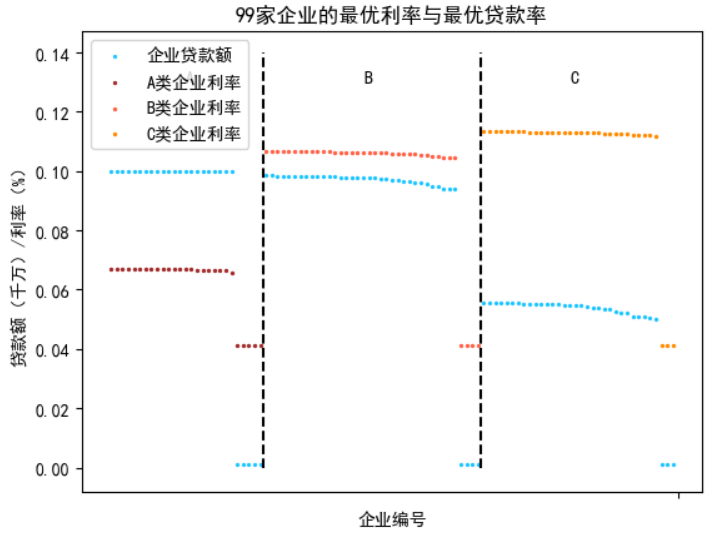

图 **4**　**99** 家企业的最优利率与最优贷款金额

观察图中结果可以发现：A、B、C 三类企业中，对于大部分企业来说，其最优借贷利率与最优借贷金额都相对稳定。这是因为其效用函数中 $\eta_k(i)$ 的关系式相同，而在 α_l 的违约率指标上，彼此之间的差距也并不会太大。导致同类企业中最终求得的最优利率与最优贷款金额的差距完全由违约率指标中的微小差距决定。使其呈现出的最优值都相对稳定。

而进一步观察可以发现，在三组中都存在个别样本，其贷款利率为 0.04，而贷款金额为 0(银行应当放弃向其贷款)。我们认为其主要原因在于，这部分企业的违约率相较于同类企业，其违约率过于高。而这对效用函数在相应变元上的梯度有及其重要的影响。以放贷金额为例：

$$\frac{\partial U}{\partial g_l} = \frac{\partial E}{\partial g_l} - \zeta \frac{\partial D}{\partial g_l}$$

对于后者，有:

$$\frac{\partial D}{\partial g_l} = 2(1-\eta_l)^2 \alpha_l (1-\alpha_l)(i_l+1)^2 g_l > 0$$

而对于前者，有：

$$\frac{\partial E}{\partial g_l} = (1-\eta_l)[(1-\alpha_l)(i_l+1)-1] \tag{12}$$

其正负性完全由 $(1-\alpha_l)(i_l+1)-1$ 决定。而 i 的上限为 0.15，这让企业的违约风险过大时，无论如何调节，U 与 g 都呈现反相关 ($\alpha > 0.13$)，同时效用和收入符号都为负。在这种情况下，银行需要放弃对相应的企业提供贷款以规避其带来的福利损失。

而对于提供贷款的企业，其平均最优利率和最优贷款金额如下 (详细数据在支撑材料中展示):

表 **4**　梯度上升法得到的最优解

	A 类企业	B 类企业	C 类企业
i	6.67%	10.60%	11.29%
g	1000000	971294.3	53.9433.8

其中，向信誉好的 A 类企业提供贷款的利率低于 B 与 C 类企业，向其提供的贷款额也更高，符合题目假设与经济学逻辑。

而通过将所有的放贷金额汇总，我们得到最终的放贷总额 $G = \sum_{l=1}^{123} \approx 7.175 * 10^7$ 元。而由于年度信贷总额 M 未知且确定，易知当 M≥G 时，银行可以为每家企业提供对应其的最优利率与最优贷款额。

而当 M<G 时，银行并不能对每家企业提供相应的贷款额。此时，需要给出银行在不同预算约束下的贷款分配规则。而根据我们在之前推导最优分配方案的过程，可以很自然地推断出通过梯度下降法得到有约束条件下的分配规则，自 M 与最优贷款分配方案开始，每次将 **g** 沿着梯度方向移动一个微小的距离使相应的 G 缩小，直到 G=M，得到的贷款分配方案仍旧为相应预算约束下的最优分配。

但这种方法的问题在于，在现实的借贷过程与预算约束下，银行难以对预算约束进行如此精细的更新和计算。因此我们提出直觉层面的第二种方法：即按照企业的种类与违约风险的顺序分配。即，优先对 A 类提供贷款；而在同类的风险评级中，优先对违约几率较低的企业提供贷款。同样，我们可以在数学上给予相应的证明：

$$\frac{\partial U}{\partial \alpha_l} = -(1-\eta)(1+i)g_l + (2\alpha_l - 1)[(1-\eta)(1+i)g_l]^2 \tag{13}$$

由于银行不应当对 $\alpha > 0.5$ 的公司放贷，且 (12) 式中已经证明过，对银行放贷的公司，应当有 $(1-\eta)(1+i) > 0$，则此时，(13) 式必定为负。则在同等放贷额 (或近似情况下)，违约风险低的公司对银行的福利有更大的改进。而这也是符合经济学的直觉的。故在 G<M 时，应当选择遵循“先风险等级、后违约概率”的顺序进行放贷。

四、 问题二的求解与分析

在本题中，由于其研究目的与问题一几乎完全相同，只是在对象设置上不同，因此在部分问题的处理上可以沿用问题一中的方法。

4.1 风险评估方案与最优化信贷策略

首先，我们沿袭问题一中使用的 logit 模型，对 302 家企业进行信贷风险的估计，得到的估计结果如下：

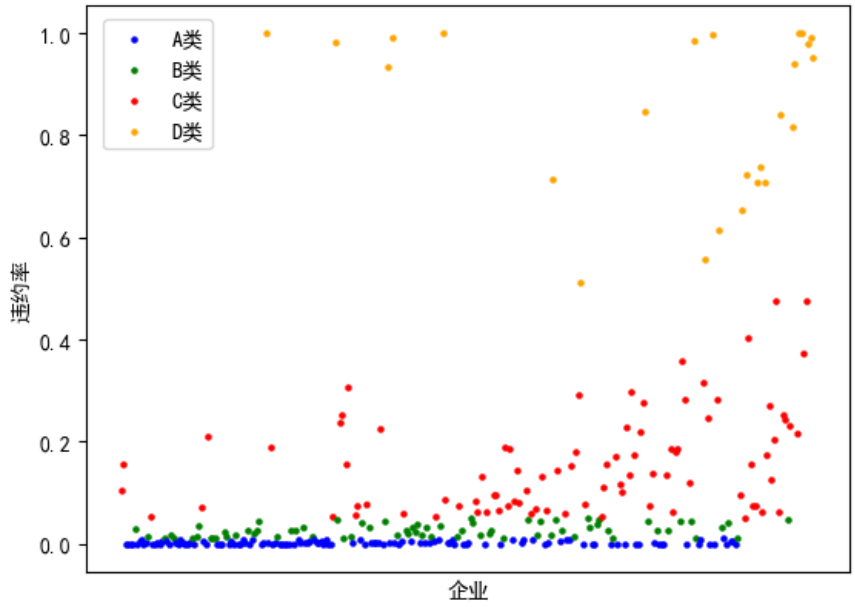

图 **5**　对 **302** 家企业 **logit** 模型的风险预测值

其中，我们基于上一问中计算出的不同信誉等级企业的平均违约率以及问题设计的要求，将违约率低于 1% 的企业设为 A 类企业，低于 5% 的企业记为 B 类企业，低于 50% 的记为 C 类企业，剩余企业为 D 类企业。在此划分标准下，共有 A 类企业 121 家，B 类 70 家，C 类企业 87 家，D 类企业 24 家。可以观察到，较问题一中，A 类企业的数量有非常明显的增加，而 D 类企业的比例有相当显著的减少。而在 B、C 类企业的数量上，其比例则保持相对的稳定。

通过观察我们注意到，附件二中的企业的总体规模较问题一中有较大的提升；同时，问题二中的企业数更多，企业的上下游供应商以及相应的中心性也更高，这让本题中的企业在通过 123 家企业回归得到的 logit 信誉评价模型上拥有更好的整体表现。

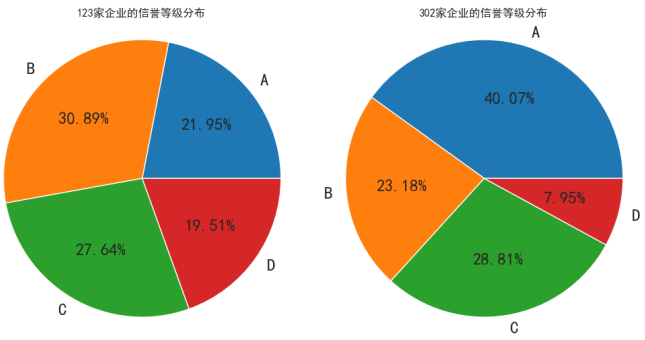

图 6　问题一与问题二中企业的信誉评级占比

而对于本题，其中提供了贷款总额上限。在此基础上可以将优化问题改写为：

$$\max_{i_k, g_l} = \sum_{l=1}^{n} U(i_k, g_l)$$

$$s.t. \sum_{l=1}^{n} g_l \leq 10^8$$

$$0.04 \leq i_k \leq 0.15$$

$$10^5 \leq g_l \leq 10^6$$

而根据问题一中完全一致的寻找初始解以及得到最优解的方法，我们可以得到 302 家公司在无约束情况下的最优信贷策略 (结果在)。其分布形式与问题一的最优策略相近，但对其贷款额求和，发现其总贷款额超过了 2.26 亿元。但本题中要求放贷总额不超过一亿元，则接下来我们将解决如何从无约束条件下的最优解推导出有约束条件下的最优解。基于此，我们设计了两个算法对其进行研究：

4.2 基于梯度下降法有约束最优解

由于本算法的目的并非求出最优解，而是使总贷款额尽快收敛到预算约束之下的同时对银行福利带来最小的影响；因此我们没有选择传统的梯度下降的方式，而是选取出两个梯度最小、在影响总效用程度最低的情况下使贷款额收敛最快的企业，每次减少其贷款额度。数学表示为：

$$\nabla g_x = \min\{\nabla \mathbf{g_t}\}$$

$$\nabla g_y = \min\{\nabla \mathbf{g_t} \backslash \nabla g_x\}$$

$$\nabla \mathbf{g_{t+1}} = \nabla \mathbf{g_t} + s_x * \nabla g_x + s_y * \nabla g_y$$

如此不断迭代，直至 $\sum_{l=1}^{n} g_l \leq 10^8$。其得到的结果在支撑材料中展示。将其中部分结果与最优的信贷决策在图 7 中进行对比，发现其中部分点的有约束贷款额和利率都

没有发生变化，而部分点的两个指标都有极大程度的降低 (直接降至最低利率和 0 贷款额)。

通过对相应企业的信誉评级和违约几率进行研究，发现，最优利率和最优贷款额均保持稳定的企业的违约几率往往较低，且信誉评级几乎都为 A 级 (其中违约度最高的企业为 E163 号，1.15%)。而其余 B、C、D 类点的贷款额几乎都降为了 0。而这个结果是符合我们在问题一中给出的 M<G 时的简化版信贷分配方案原则的。即，优先保证高信用评级、低违约几率企业的信贷供给。

而对于部分被取消信贷资格的 A 类企业，我们认为其原因可能是由于在我们修改的梯度下降法中，梯度的下降遵循一定的顺序，在不同的位置各个信贷方案的梯度大小关系也并非完全一致。而这导致了某些低违约率的企业可能会提前被淘汰。但总体上我们可以认为，虽然信贷资金的分配较为极端，但是梯度下降法得到的结果是符合我们在问题一得到的分配方案的。因而可以被接受。

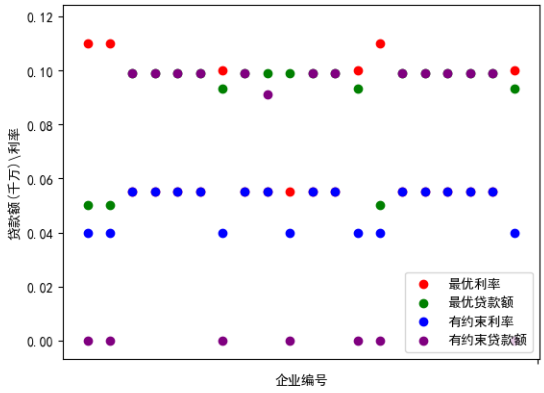

图 7　部分公司有无约束条件下的最优利率与最优贷款额

4.3 基于 PSO 算法的有约束最优解

为了对梯度下降法中存在的时序问题和极端的结果进行验证，并在极高的维度下以更快的速度精确收敛到预算约束边界，我们进一步引入了启发式算法中的 PSO 算法 (Particle Swarm Optimization)，对问题二的优化结果进行进一步的检验和计算并增加一定的随机性。其基本步骤如下：

在本题中，其每个粒子 l 会以以下公式更新自身位置：

$$v_t = \omega * v_{t-1} + c_1 * rand() * (pbest_l - x_{t-1}) + c_2 * rand() * (gbest - x_{t-1}) \tag{14}$$

$$x_{t+1} = x_t + v_t \tag{15}$$

式中 x 代表粒子位置，v 代表粒子更新速度；$pbest_l$ 是粒子 l 搜索到的历史最优位置，而 $gbest$ 是所有粒子搜索到的全局最优位置。式 (1) 表示粒子 l 下一步将朝着历史最

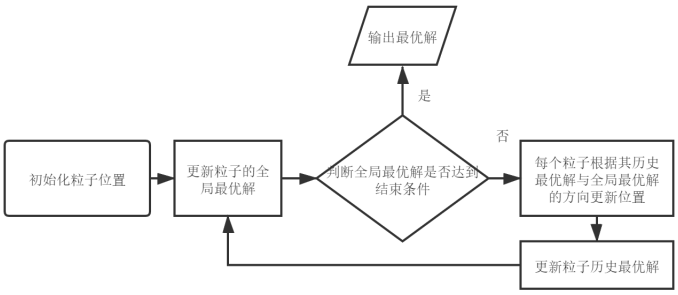

图 **8**　**PSO** 算法流程图

优位置与全局最优位置的线性组合方向移动。$\omega * v_i$ 称为惯性项，表示粒子的移动受到上一次移动的影响。当 ω 较大时，说明“惯性”较大，全局寻优能力较强；反之，则局部寻优能力较强，全局寻优能力较弱。

而对于其面对的预算约束限制，我们在模型进行中通过惩罚函数对其进行限制，即，当某粒子贷款总额超过 1 亿时，对其效用函数进行极大的惩罚，迫使其回到预算约束下。

在我们初始化四个粒子并进化了一百代后，PSO 算在我们规定的范围内以明显较梯度下降法更快的速度寻找到符合条件的解。我们将其中的部分结果与梯度下降法得到的信贷决策策略对比展示如下：

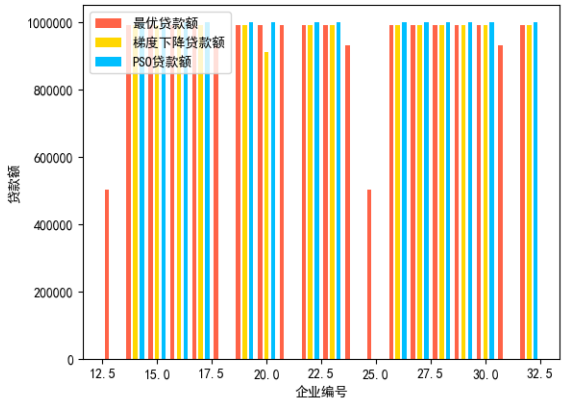

图 **9**　**PSO** 算法与梯度下降法、无约束最优贷款值的比较

可以发现，其中绝大部分的 PSO 算法与梯度下降法得到的预测值和在有约束条件下的变化趋势表现都相近。对于信誉高的企业，两种算法都向相应企业提供全额贷款；而对于低信誉的、最优贷款额不高的企业，银行会在预算约束下优先削减其贷款。只有在部分的样本中体现出一定的差距 (如图中的样本 20)，而这类样本大约处在 1% 的违约率左右，即很有可能处在算法迭代的过程中导致贷款额遭到了部分减少。而总体上可以认为，PSO 算法的预测结果和梯度下降的预测结果在趋势上保持一致，并且都符合了问题一中提出的信贷策略，进一步验证了问题一中贷款策略的准确性。

五、 问题三的求解与分析

5.1 问题分析

本题要求在第二题的基础上，考虑到现实生活中外生冲击对银行信贷政策的影响。由于外生冲击的多源性和突发性，银行很难在事先通过数据调查搜集到相关的信息。这要求银行在贷款之前应当充分考虑各种可能的外生冲击以及其对各个行业企业的偿债能力造成的影响。并对前文中建立的风险评估及偿债模型进行相应的调整，以帮助银行在复杂的现实场景中得到更高的收益。

5.2 外生冲击来源及其影响

对风险模型以及放贷策略进行整理，可以发现其中的对银行的贷款供给及策略产生影响的因素共有以下几类：

$$\text{银行}\begin{cases}\text{违约率}\alpha\\ \text{潜在客户损失比例}\eta(i)\\ \text{现金流/贷款金额上限}M\end{cases}$$

其中，银行的现金流在本题中是外生信息，由银行的总体经营状况决定；潜在客户损失比率 η 是 i 的对数函数，但其中的参数可能会在外生冲击下有所改变；而违约率的评判则来自于银行对贷款客户的了解与调查，决定了小微企业对银行贷款的需求。结合风险度量模型，其中的影响因素共有以下几点：

$$\alpha\rightarrow\text{企业}\begin{cases}\text{销货额}P\\ \text{销购比}r\\ \text{中心性}\Omega\\ \text{退款率}t\end{cases}$$

而对于冲击类别，其中可以简略地分为两类：一类是，对全行业都带来影响的宏观经济冲击；另一类是个别行业伴随内部的技术发展、资本积累、供需关系变化等，对本行业内企业的经营、生产带来的不确定性。下面一部分我们将分别对各种冲击对银行信贷决策的作用机制进行识别和探讨。

5.3 宏观经济冲击的影响机制

以新冠疫情、2008 年次贷危机等为代表的国际宏观经济冲击对国际经济的全产业链都带来冲击。其中对企业最明显的冲击在于两方面：宏观经济的不确定性会通过增加个体收入的不确定性，影响国际市场上全供应链的需求，对企业的销售渠道造成巨大影

响；另一方面，随着宏观经济的不确定性程度不断升高，企业在应对波动时需要更多的流动性资金。而在金融危机中，市场流动性枯竭，这给各个企业、尤其是小微企业的经营造成了更大的压力。[3]

而对于银行来说，其面临的问题则同时来自需求端和供应端：一方面，信贷波动和产出缺口之间具有非常明显的相关性，这样市场对信贷的需求很容易受到实业企业波动的影响 [4]，以带来更多的潜在客户损失; 与此同时，客户的储蓄减少会限制银行的现金流，导致银行可贷款部分的资金减少，从而影响银行贷款的供给端。而由于题目中已经给定了企业的预算约束，在此我们将不再讨论银行的现金流问题。

对题目中的 277 家 (排除了其中的 D 类企业) 企业的行业进行划分，我们共得到了 9 类企业，接下来，我们以宏观冲击中最典型的新冠疫情为例，对其对题目中银行对各行业贷款政策造成的影响进行分析：

首先在企业端，我们认为疫情的冲击主要体现在中心性、销货额和销购比上。而对于行业的冲击在个体层面上可能会体现出差异。故我们将行业 k 在指标 w 上的受到的冲击记作：$\epsilon_{kw} \sim N(\mu_{kw}, \sigma_{kw})$。基于新冠疫情的具体数据 [5]，我们计算出了其中的参数并展示在下表中：

表 **5**　**302** 家企业的行业分布与冲击参数

	个体	服务	零售	运输	科技	文娱	加工	工程	其他
数量	48	59	34	11	28	11	34	43	9
$\mu_P = \mu_r$	-0.4	-0.085	-0.125	-0.134	0.2	-0.025	-0.135	-0.467	-0.15
$\sigma_P = \sigma_r$	0.007	0.008	0.0075	0.0075	0.0071	0.0084	0.0075	0.0046	0.007

(出于数字规模考虑,$\mu_\Omega = 0.1\mu_P, \sigma_\Omega = 0.1\sigma_P$)

将其带入风险评价模型中，可以算出企业的违约率在冲击前后的变化情况。其中部分展示在图 10 中。观察其中结果可以发现，几乎所有企业的违约率在受到新冠疫情的外生冲击后都有明显的升高。唯一的例外出现在科技企业上 (如图中 17 号样本)。由数据特征，可以发现对疫情能够加快相关科技产业的进展，而对相应的企业有明显的裨益。这让科技企业的违约率反而在疫情的冲击下逆势走低。

而进一步考虑到疫情对银行潜在客户流失的影响，我们对 $\eta(i)$ 的关系也进行了修正。我们认为顾客流失的速率不会有随着利率有太大的改变，但是其固定的挤出效应会更强。基于此，我们将 $\eta(i) = \ln(ai + b)$ 修正为 $\eta(i) = \ln(ai + 1.2b)$，以体现外生冲击对企业借贷的限制作用。

整理新冠疫情后的企业特征发现，新冠疫情后，A 类评级企业减少 15 个，D 类评

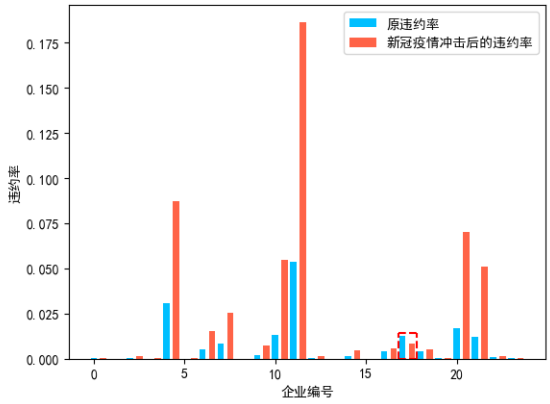

图 **10**　新冠疫情冲击前后的企业违约率变化

级企业增加 5 个，企业的信誉评级集中于 C 类企业。而对于银行的在无约束情况下银行的最优放贷总额较问题二中减少了近一千三百万，平均无约束条件下的放贷额度在 76 万元左右。而当银行总借款约束上限限制在一亿元时，最优借贷策略与问题二类似，但有部分科技企业伴随着本身的违约率降低和其他行业企业违约率的升高而获得了信贷上的支持。而这也是符合直觉的。

5.4 个别行业的供需冲击

除了宏观的经济冲击之外，有更多的冲击是针对个别行业的冲击，而对银行本身的借贷能力不造成影响的。例如，猪瘟的蔓延只会对生猪行业的供给造成影响，进而提高相应的市场价格。虽然在相应行业会造成较大的波动，但是在全行业并不会产生显著的影响。故这只会对相关产业企业的违约率造成影响。

接下来，我们以 277 家企业中的个体企业为例，分析其行业波动对其带来的影响。我们采取上文中同样的冲击形式，但是只对个体企业施加。发现，施加相应的冲击后，个体企业的平均违约率从 3% 上升到了 5.07%。所有企业中 A 类企业共减少了 7 个且全部来自个体企业，D 类企业共增加了 1 个。与此同时，银行将不再向原先几乎得到最大额度贷款的 (E132,E155,E238) 提供贷款，而相应地有另外三家来自其他行业的企业获得贷款。这是由于这三家企业的效用函数的梯度在个体企业受到冲击、违约率升高的情况下表现较好，因而通过相对升高的履约率得到了银行分配的贷款。

由此可以发现，个别行业的供需冲击不会导致全行业的经营情况的普遍恶化，但是会对本行业内的企业的违约风险和信贷情况都造成巨大的影响。而这种影响虽然不会使其他行业企业本身发生太大变化，但是其相对违约风险的降低也能够使其中的一些企业得到银行的贷款。而这不会对银行的放贷策略产生根本性的变化。

六、 模型的检验

6.1 LSTM 模型对评分系统的再检验

在问题二中，我们直接通过问题一中 121 家公司的 logit 回归模型对问题二中 302 家公司的违约几率进行了估计。但由于两组企业在规模以及供应能力上的差异，用前者的模型直接对后者进行衡量可能会带来参数上的巨大偏差。同时，作为有标签的模拟，企业的偿债与否能够体现出企业在响应时间段中的偿债能力。但问题二中的 302 家待贷款的企业并没有相应的标签，将总量带入 logit 模型中也不能体现出跨越三年的时间尺度之后企业当下的经营情况。为了克服这些弊病，下面引入 LSTM(长短时记忆网络) 模型对问题二中的估计结果进行进一步的检验。

在此我们将附件中包含的 128 个公司自 2017 年 1 月以来的财务交易数据分月进行统计和汇总，共得到 38 个月份的数据。随后构建了一个双层的 LSTM 神经网络其基本结构如下图所示：

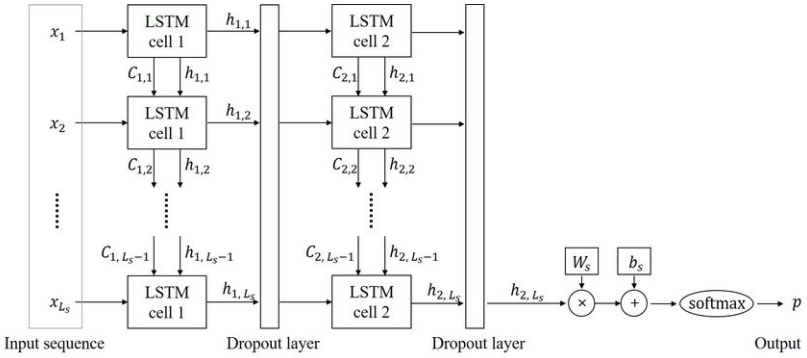

图 **11** **LSTM** 工作原理

其中，我们的输入层包括了销货额、购货额、上游中心性、下游中心性、上游退款率和下游退款率共六类指标，对于 123 家公司共 38 个月份的数据。随后将这些数据输入对 LSTM 网络进行训练，即，输入 123*38*6 的数据，经过第一层网络转化为 123*38*20 作为第二层 LSDM 的输入变为 123*1*20 的数据。最终通过 DenseNet 归一化 123*1*4 的向量，其中包含每一个企业的最终输出的信誉种类。在训练过程中，通过不断地反向传播，得到最优地 LSTM 网络。其在训练集上的准确度达到了 70% 左右。

接下来我们通过训练完成的 LSTM 模型对 302 家公司的信誉评级进行了预测，得到的分类结果如下 (需要注意到的是，考虑到在 logit 模型中划分的违约几率中，D 类企业的违约率较 A 类高数十倍，因此我们对四类信誉评级在 LSTM 中的权重也分别有分别的调整)：

可以发现 LSTM 模型与 logit 模型的预测结果有相似之处。而最大的区别在于，LSTM 模型中预测出的 D 类企业要明显多于 A 类，我们认为这是在训练集中 123 家企业较预测集中的企业供应网络更小、中心性更小的特征导致的。而对于其他类别的公司，其预

表 6　**logit** 模型与 **LSTM** 模型评分系统结果对比

	A 类	B 类	C 类	D 类
logit 模型	121	70	87	24
LSTM 模型	105	47	92	58

测标签占比基本相同。由此，两种方法的结果彼此对照，验证了分类结果的合理性。

6.2 风险规避系数的检验

而在效用函数中，存在另一个始终没有被我们讨论的参数，即银行的风险规避系数 ζ。在初始模型设置中，我们将其设置为 $5*10^{-7}$，将收入函数 E 与风险函数 D 的数量级维持在同一层面上以便于优化。而在进行梯度上升/下降的过程中，我们又将 ζ 调整到了 0.5。但在本题的求解过程中，风险规避系数的值始终来自于我们人为的外生给定，而我们并没有讨论过风险规避系数的变化会给银行的信贷决策带来怎样的影响。下面我们将对其进行讨论。

我们将 ζ 的值扩大十倍调整到了 ζ=5。经过这样的调整之后，在问题一中最终得到的无约束放贷策略和总放贷金额变动较原最优解差距并不大。

而在问题二的梯度下降过程中，我们发现，在 ζ=5 的情况下计算出的总放贷金额仍然高于借贷约束 1 亿元，甚至在无约束条件下的最优值和最优信贷方案都非常接近。但是加入预算约束后，最终的分配方式却存在着较大差别。其中的部分结果展示如下：

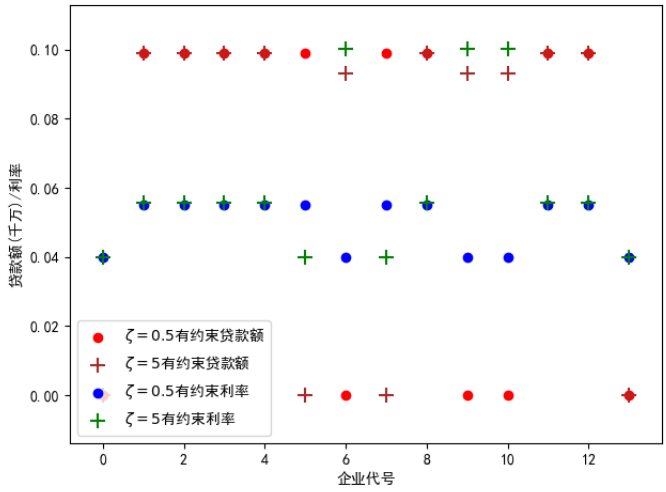

图 **12**　$\zeta = 0.5$ 与 $\zeta = 5$ 时有约束条件下的最优利率与最优贷款额

可以观察到，其中部分违约率较高的企业也得到了较多的贷款，而部分违约率很低

的企业却没有得到贷款。从效用模型上对其进行分析，我们由式 (12)、(13) 可以发现：

$$\frac{\partial g}{\partial \alpha} = \frac{\partial g}{\partial U} * \frac{\partial U}{\partial \alpha} = \frac{-(1+i)g + \zeta(2\alpha-1)(1+i)^2(1-\eta)^2 g^2}{[(1+i)(1-\alpha)-1] - 2\zeta(1+i)^2(1-\eta)^2(1-\alpha)\alpha g} \tag{16}$$

由于银行不会给 $\alpha \geq 0.5$ 的公司提供贷款，则其分母一定为负数。而由式 (12) 可知，分子的第一项一定为正数 (若为负，则银行的收益与效用始终为负数)。但当 ζ 过大时，分子的第二项表现为负数。则此时，$\partial g/\partial \alpha$>0，银行规避风险的倾向过于强，随着违约率升高，向相应企业的放款额度也上升，而这显然是不符合现实逻辑的。因此 ζ 不能设置地过大。

而当 ζ 过小时，此时对风险的规避在银行的效用函数中占比过于低，难以体现银行对风险的厌恶。在这种情况下，银行对满足式 (12) 的任意违约率企业的最优放贷决策都是向其提供最高的贷款额。而这在无约束的优化条件下难以体现风险厌恶机制对不同信誉企业的区分，因此也不适合作为 ζ 的取值。

综上所述，我们认为 $\zeta = 5 * 10^{-7}$(在梯度算法中归一化的条件下转化为 0.5) 的风险规避系数是合适且能够帮助银行做出正确的信贷决策的。

七、 模型的优缺点

7.1 优点

• 模型建立逻辑连贯，在数学上证明严谨，对于银行在不同情况的信贷决策有较为完整的讨论。

• 模型层次丰富，运用了多种模型与思路对问题进行解释与分析，且能够彼此映证。

• 在第三部分的讨论中除了经济学建模之外，还加入了现实数据对相关的冲击机制识别进行了检验。

7.2 缺点

• 评分系统设计较为单一，除了违约率指标之外没有更多考虑到企业经营的其他综合指标。

• 在确定信贷策略时为了简化相应的流程和利用有限的数据，并没有分别讨论不同行业公司的数据特征，并对其信贷策略进行更加精细的分析。

参考文献

[1] H. Markowitz, *Portfolio Selection : Efficient Diversification of Investments.* Yale University Press, 1959.

[2] J. E. Stiglitz and A. Weiss, “Credit rationing in markets with imperfect information,” *The American economic review*, pp. 393–410, 1981.

[3] . 王 and . 宋, “宏观经济不确定性、资金需求与公司投资,” 经济研究, vol. 49, no. 02, pp. 4–17, 2014.

[4] . 许 and . 陈, “银行信贷与中国经济波动:1993—2005,” 经济学 *(*季刊*)*, vol. 8, no. 03, pp. 969–994, 2009.

[5] . 许 and . 张, “新冠肺炎疫情对企业影响调查与一季度经济预测,” 淮海文汇, no. 01, pp. 8–14, 2020.

[6] I. G. Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

附录

附录 1：支撑材料文件列表

结果汇总.xlsx – 各题计算结果；

上游企业 1.xlsx, 上游企业 2.xlsx, 下游企业 1.xlsx, 下游企业 2.xlsx – 上下游企业信息，用来计算邻接矩阵；

123 邻接矩阵.py, 302 邻接矩阵.py – 计算企业的邻接矩阵 (运行时需要附件一放置于本文件夹路径下)；

123 中心性.py, 302 中心性.py – 计算企业的中心性 Ω;

logit.do – logit 回归模型 (需用 stata 打开);

123 家企业总收益与成本.xlsx, 302 家企业总收益与成本.xlsx – 记录企业的相关特征数据以计算 α；

第一题梯度上升.xlsx, 第二题梯度上升.xlsx – 梯度上升求全局最优解的数据准备；

第二题梯度下降.xlsx – 梯度下降求有约束最优解的数据准备；

第一题梯度上升.py, 第二题梯度上升.py, 第二题梯度下降.py – 梯度上升/下降算法；

eta_ 冲击.py – 宏观经济冲击的影响；

单行业冲击.py – 个体经营企业冲击的影响；

textbf 文件夹: PSO: PSO_ 第二小题.py – PSO 算法；

textbf 文件夹: lstm: lstm.py: lstm 训练模型；

predict.py: lstm 预测模型；

ckpy(文件夹)：对模型数据进行存储的中间产物；

data.xlsx, data2.xlsx: 企业的上下游中心性；

expend.xlsx, expend2.xlsx: 购货额；

profit.xlsx, profit2.xlsx: 销货额；

feis-up, feis-up2, feis-down, feis-down: 企业的上下游退货率；

附录 2：LSTM 的原理

LSTM 全称为长短时记忆网络 (Long Short-Term Memory)，是一种改进过的 RNN 网络，用来解决传统循环神经网络中对长时信息处理过程中发生的梯度爆炸等问题。

LSTM 网络在传统 RNN 每时点产生一个输出、隐单元彼此之间循环连接的基础上增加了门限 (遗忘门)，使得在循环中，自循环的权重并非如传统 RNN 中般固定，而每时每刻都随着输入值发生变化，并通过已经积累的信息不断对其做出反应。[6]

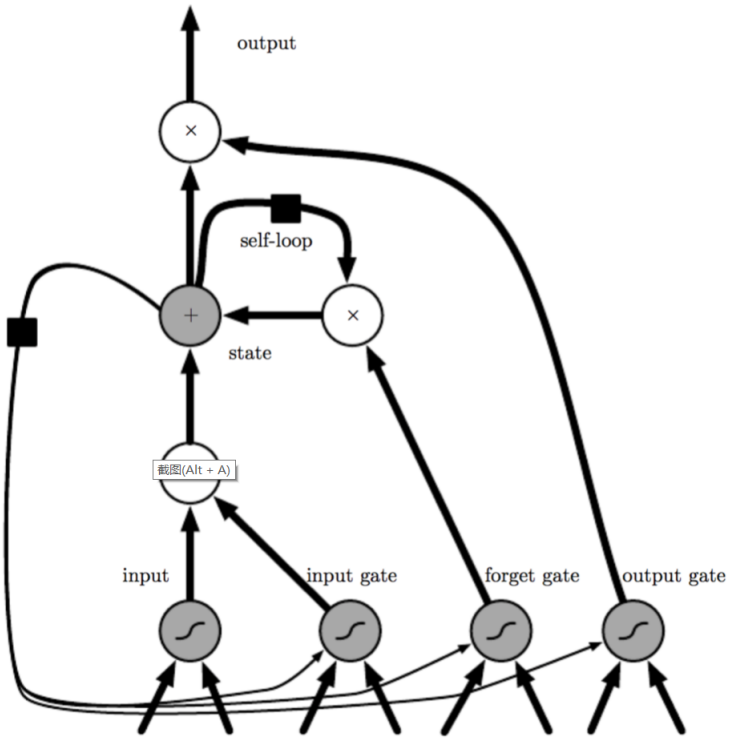

图 **13**　**LSTM** 的概要原理

而在对问题二的检验中，我们采用了两层堆叠的 LSTM 网络形式对输入的 302*38*6 的数据进行处理，直接输出对每个企业包含 A、B、C、D 四个信誉类别的预测结果。这是因为，这样的网络较单层的网络能够在避免过拟合的前提下更好地对时序中的潜藏信息进行捕捉，并对其进行预测。经过第一层网络的输出数据为 302*38*20。将其再次通过第二层 LSTM 网络，并在最后通过全连接层，从而得到最终的预测结果 302*1*4，并与问题二中 logit 模型预测出的结果进行对照。

附录 3：企业邻接矩阵的建立

```
import pandas as pd
import numpy as np



data1 =pd.read_excel(r'附件1: 123家有信贷记录企业的相关数据.xlsx', sheet_name=0)
data2 =pd.read_excel(r'附件1: 123家有信贷记录企业的相关数据.xlsx', sheet_name=1)
data3 =pd.read_excel(r'附件1: 123家有信贷记录企业的相关数据.xlsx', sheet_name=2)
data4 =pd.read_csv(r'上游企业1.csv', encoding='gbk')
```

```
data5 =pd.read_csv(r'下游企业1.csv', encoding='gbk')
data6 =pd.read_excel(r'123家企业总收益与成本.xlsx')
print('读取完成')


matrix_up =np.zeros((123, 13961))
matrix_down =np.zeros((123, 24449))
ups =data4.values
downs =data5.values

for i in range(len(data2)):
up =data2['销方单位代号'][i]
idx =list(ups).index(up)
com =int(data2['企业代号'][i][1:]) -1
matrix_up[com][idx] =data6['购货额'][com]
if i % 10000 ==0:
print(i //10000)
print(matrix_up[0])
dataframe =pd.DataFrame(matrix_up)
dataframe.to_csv('matrix_up', index=False, sep=',')

for i in range(len(data3)):
down =data3['购方单位代号'][i]
idx =list(downs).index(down)
com =int(data3['企业代号'][i][1:]) -1
matrix_down[com][idx] =data6['销货额'][com]
if i % 10000 ==0:
print(i //10000)
print(matrix_down[0])
dataframe =pd.DataFrame(matrix_down)
dataframe.to_csv('matrix_down', index=False, sep=',')
```

附录 4：计算企业的中心性

```
import pandas as pd
import numpy as np

data1 =pd.read_csv('matrix_up.csv')
data2 =pd.read_csv('matrix_down.csv')


print(data1.head())
he =np.zeros(13961)
for i in range(13961):
he[i] =sum(data1[f'{i}'])


result =np.zeros(123)
for k in range(123):
row =data1.iloc[k].values
for l in range(len(row)):
if row[l] !=0:
result[k] +=he[k]
print(k)
```

```
dataframe =pd.DataFrame( result)
dataframe.to_csv('123中心性up', index=False, sep=',')


he =np.zeros(24449)
for i in range(24449):
he[i] =np.log(sum(data2[f'{i}']))


result =np.zeros(123)
for k in range(123):
row =data2.iloc[k].values
for l in range(len(row)):
if row[l] !=0:
result[k] +=he[k]

dataframe =pd.DataFrame( result)
dataframe.to_csv('123中心性down', index=False, sep=',')
```

附录 5：logit 模型的 stata 代码

```
clear all

cd "C:\Users\84710\Desktop\数模\code"
*这里将路径替换为存放do-file和文件的路径
import excel using "123家企业总收益与成本.xlsx",sheet("Sheet1") firstrow
logit sign 对数销货额 上游中心性 下游中心性 上游退款率 下游退款率 lnrati

predict psign
```

附录 6：梯度上升模型求无约束最优信贷方案

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd

def xiugai(a, i, k):
part1 =a[:i]
part2 =tf.constant([k], dtype=tf.float32) # 要修改的元素
part3 =a[i+1:]
new_tensor =tf.concat([part1, part2, part3], axis=0)
return new_tensor

data0 =pd.read_excel('第一题梯度上升.xlsx')
beta =data0['psign'].values
beta =tf.convert_to_tensor(beta, dtype=tf.float32)
I =[]
G =[]
eta =[]
for i in range(99):
hang =data0.iloc[i]
```

```
ping =hang[1]
if ping =='A':
I.append(0.015 /0.11)
G.append(990000 /1000000)
eta.append(np.log(12.876*I[i] *0.11 +0.04+0.697))
elif ping =='B':
I.append(0.06 /0.11)
G.append(930000 /1000000)
eta.append(np.log(12.265*I[i] *0.11 +0.04+0.682))
elif ping =='C':
I.append(0.07 /0.11)
G.append(500000 /1000000)
eta.append(np.log(12.45*I[i] *0.11 +0.04+0.649))

I =tf.convert_to_tensor(I, dtype=tf.float32)
G =tf.convert_to_tensor(G, dtype=tf.float32)
eta =tf.convert_to_tensor(eta, dtype=tf.float32)


for o in range(5000):
with tf.GradientTape() as tape:
tape.watch([G, I])

E =G *(1.0 -eta) *(-beta +I -I *beta)
U =E -0.5 *(1 +I)*(1+I)*beta*(1-beta)*G*G*(1-eta)*(1-eta)
U =tf.reduce_mean(U)

grad_g, grad_i =tape.gradient(U, [G, I])
if o % 100 ==0:
print(o //100)
print(grad_g)
G =0.001 *grad_g +G
I =0.0005 *grad_i +I

for j in range(99):
if (1 +I[j]) *(1 -beta[j]) <1:
G =xiugai(G, j, 0.01)
I =xiugai(I, j, 0.01)
else:
if G[j] >0.999:
G =xiugai(G, j, 0.999)
elif G[j] <0.001:
G =xiugai(G, j, 0.001)
if I[j] >0.999:
I =xiugai(I, j, 0.999)
elif I[j] <0.001:
I =xiugai(I, j, 0.001)
g =[]
i =[]
for k in range(99):
g.append(float(G[k]) *1000000)
i.append(float(I[k]) *0.11 +0.04)

dataframe =pd.DataFrame([g, i])
dataframe.to_csv('第一题梯度上升结果.csv')
```

附录 7：梯度下降模型求有约束最优信贷方案

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd

def xiugai(a, i, k):
part1 =a[:i]
part2 =tf.constant([k], dtype=tf.float32) # 要修改的元素
part3 =a[i+1:]
new_tensor =tf.concat([part1, part2, part3], axis=0)
return new_tensor


data0 =pd.read_excel('第二题梯度下降.xlsx')
beta =data0['logi'].values
beta =tf.convert_to_tensor(beta, dtype=tf.float32)

I =data0['i'].values
G =data0['g'].values
I =(I -0.04) /0.11
G =G /1000000

# I = []
# G = []
eta =[]
for i in range(277):
ping =beta[i]
if ping <0.01:
# I.append(0.015 / 0.11)
# G.append(990000 / 1000000)
eta.append(np.log(12.876*I[i] *0.11 +0.04+0.697))
elif 0.01 <=ping <0.05:
# I.append(0.06 / 0.11)
# G.append(930000 / 1000000)
eta.append(np.log(12.265*I[i] *0.11 +0.04+0.682))
else:
# I.append(0.07 / 0.11)
# G.append(500000 / 1000000)
eta.append(np.log(12.45*I[i] *0.11 +0.04+0.649))

I =tf.convert_to_tensor(I, dtype=tf.float32)
G =tf.convert_to_tensor(G, dtype=tf.float32)
eta =tf.convert_to_tensor(eta, dtype=tf.float32)

o =0
while int(tf.reduce_sum(G)) >100:

with tf.GradientTape() as tape:
tape.watch([G, I])

E =G *(1.0 -eta) *(-beta +I -I *beta)
U =E -0.5 *(1 +I)*(1+I)*beta*(1-beta)*G*G*(1-eta)*(1-eta)
U =tf.reduce_mean(U)
```

```
grad_g, grad_i =tape.gradient(U, [G, I])

for j in range(277):
if (1 +I[j]) *(1 -beta[j]) <1:
G =xiugai(G, j, 0)
I =xiugai(I, j, 0)
grad_g =xiugai(grad_g, j, 100)
grad_i =xiugai(grad_i, j, 100)
else:
if G[j] >0.999:
G =xiugai(G, j, 0.999)
elif G[j] <0:
G =xiugai(G, j, 0)
grad_g =xiugai(grad_g, j, 100)
if I[j] >0.999:
I =xiugai(I, j, 0.999)
elif I[j] <0.001:
I =xiugai(I, j, 0.001)
grad_i =xiugai(grad_i, j, 100)

a1, b1 =np.argpartition(grad_g, 1)[0: 2]
a2, b2 =np.argpartition(grad_i, 1)[0: 2]
G =xiugai(G, a1, -0.01+float(G[a1]))
G =xiugai(G, b1, -0.01+float(G[a2]))
I =xiugai(I, a2, -0.01 *float(grad_i[a2]))
I =xiugai(I, b2, -0.01 *float(grad_i[b2]))

if o % 100 ==0:
print(o //100)
print(sum(G))
# G = 0.001 * grad_g + G
# I = 0.0005 * grad_i + I
o +=1

g =[]
i =[]
for k in range(277):
g.append(float(G[k]) *1000000)
i.append(float(I[k]) *0.11 +0.04)

dataframe =pd.DataFrame([g, i])
dataframe.to_csv('第二题梯度下降结果.csv')
```

附录 8:PSO 算法求解第二小题

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 12 17:20:42 2020
"""



import pandas as pd
import numpy as np
import random
```

```
def utility_fuction(yita_k,i_m,beta_m,g_m):
U =(1-yita_k) *((1+i_m) *(1-beta_m)-1) *g_m -(5e-7)*g_m*g_m*((i_m+1)**2)*beta_m*(1-beta_m)*((1-yita_k)**2)
return U

def Ex(yita_k,i_m,beta_m,g_m):
E =(1-yita_k) *((1+i_m) *(1-beta_m)-1) *g_m
return E

#流失率拟合
def loss_A(i):
A =np.log(12.876*i +0.2307)
return A

def loss_B(i):
B =np.log(12.265*i +0.682)
return B

def loss_C(i):
C =np.log(12.450*i +0.649)
return C

#PSO
def change_speed(local_best, global_best, current_int, current_g, speed_pre):
speed_int =0.8*speed_pre[0] +np.random.rand()*(global_best[0] -current_int) +
                                                    0.5*np.random.rand()*(local_best[0] -current_int)
speed_g =0.8*speed_pre[1] +np.random.rand()*(global_best[1] -current_g) +
                                                    0.5*np.random.rand()*(local_best[1] -current_g)
speed =np.array([[0.0]*230,[0.0]*230])
speed[0] =speed_int
speed[1] =speed_g
return speed

def new_location(location,changespeed):
new_loc =np.array([[0.0]*230,[0.0]*230])
new_loc[0] =location[0] +changespeed[0]
new_loc[1] =location[1] +changespeed[1]
#控制范围
for i in range(len(new_loc[0])):
if new_loc[0][i] <0.04:
new_loc[0][i] =0.04
if new_loc[0][i] >0.15:
new_loc[0][i] =0.15
if new_loc[1][i] <0:
new_loc[1][i] =0
if new_loc[1][i] >0.01:
new_loc[1][i] =0.01
return new_loc


#读取数据

io =r'302家企业总收益与成本.xlsx'
company =pd.read_excel(io, sheet_name=2, header=0, names=None, index_col=None,
usecols=None, squeeze=False,dtype=None, engine=None,
converters=None, true_values=None, false_values=None,
```

```
skiprows=None, nrows=None, na_values=None, parse_dates=False,
date_parser=None, thousands=None, comment=None, skipfooter=0,
convert_float=True)
pingji =company['评级数值']
weiyue =company['违约率']


#随机初始化粒子位置
interest =np.array([[0.0]*230,[0.0]*230,[0.0]*230,[0.0]*230])
g_rate =np.array([[0.0]*230,[0.0]*230,[0.0]*230,[0.0]*230])
a_list =[i for i in range(230)]
for p in range(4):
interest[p] =np.random.rand(230)*0.11+0.04
g_sample =random.sample(a_list,100)
for t in range(230):
if p !=3:
if t in g_sample:
g_rate[p][t] =0.01
# print('粒子',p+1,'的初始位置是',interest[p],g_rate[p])

#初始化局部与全局最优值
g_best =np.array([[0.0]*230,[0.0]*230,[-1e15]]) #g_best初始值必须很低不然更新不了了
l_best=np.array([[[0.0]*230,[0.0]*230,[-1e15]],[[0.0]*230,[0.0]*230,
[-1e15]],[[0.0]*230,[0.0]*230,[-1e15]],[[0.0]*230,[0.0]*230,[-1e15]]])


#计算效用
ut_list=np.array([0.0,0.0,0.0,0.0])location_now=np.array([[[0.0]*230,[0.0]*230],
[[0.0]*230,[0.0]*230],[[0.0]*230,[0.0]*230],[[0.0]*230,[0.0]*230]])
for p in range(4):
ut =ut_list[p]
interest_now =interest[p]
g_now =g_rate[p]
location_now[p][0] =interest_now
location_now[p][1] =g_now
for m in range(230):
if pingji[m] ==0:
ut +=utility_fuction(loss_A(interest_now[m]),
interest_now[m],weiyue[m],g_now[m]*1e8)
if pingji[m] ==1:
ut +=utility_fuction(loss_B(interest_now[m]),
interest_now[m],weiyue[m],g_now[m]*1e8)
if pingji[m] ==2:
ut +=utility_fuction(loss_C(interest_now[m]),
interest_now[m],weiyue[m],g_now[m]*1e8)
ut_list[p]=ut
#更新局部最优
if ut >l_best[p][2]:
l_best[p][2] =ut
l_best[p][0] =interest_now
l_best[p][1] =g_now
#更新全局最优
if ut >g_best[2]:
g_best[2] =ut
g_best[0] =interest_now
g_best[1] =g_now
```

```
print('初始效用',ut_list)

#进行迭代
pre_speed=np.array([[[0.0]*230,[-0.01]*230],[[0.0]*230,[-0.01]*230],
[[0.0]*230,[-0.01]*230],[[0.0]*230,[-0.01]*230]])
for e in range(100):
print('第',e+1,'次迭代')
#每个粒子进行速度的计算
ut1_list =np.array([0.0]*4)
ex1_list =np.array([0.0]*4)
for p in range(4):
change =change_speed(l_best[p],g_best,interest[p],g_rate[p],pre_speed[p])
# print('粒子',p+1,'的速度变化是',change)
location_now[p] =new_location(location_now[p],change)
pre_speed[p] =change
# print('粒子',p+1,'现在的位置是',Location_now[p])
#计算效用
for m in range(230):
if pingji[m] ==0:
ut1_list[p] +=utility_fuction(loss_A(location_now[p][0][m]),location_now[p][0][m],
weiyue[m],location_now[p][1][m]*1e8)
ex1_list[p] +=Ex(loss_A(location_now[p][0][m]),location_now[p][0][m],weiyue[m],location_now[p][1][m]*1e8)
if pingji[m] ==1:
ut1_list[p] +=utility_fuction(loss_B(location_now[p][0][m]),location_now[p][0][m],
weiyue[m],location_now[p][1][m]*1e8)
ex1_list[p] +=
Ex(loss_B(location_now[p][0][m]),location_now[p][0][m],weiyue[m],location_now[p][1][m]*1e8)
if pingji[m] ==2:
ut1_list[p] +=utility_fuction(loss_C(location_now[p][0][m]),location_now[p][0][m],
weiyue[m],location_now[p][1][m]*1e8)
ex1_list[p] +=Ex(loss_C(location_now[p][0][m]),location_now[p][0][m],weiyue[m],location_now[p][1][m]*1e8)
#控制预算范围的惩罚函数
print('粒子',p+1,'的效用是',ut1_list[p])
print('粒子',p+1,'的期望是',ex1_list[p])
#更新局部最优
if ut1_list[p] >l_best[p][2]:
l_best[p][2] =ut1_list[p]
l_best[p][0] =location_now[p][0]
l_best[p][1] =location_now[p][1]
#更新全局最优
for p in range(4):
if ut1_list[p] >g_best[2]:
g_best[2] =ut1_list[p]
g_best[0] =location_now[p][0]
g_best[1] =location_now[p][1]

'''
调用结果
for i in range(len(g_best[0])):
print(g_best[0][i])

for i in range(len(g_best[1])):
print(g_best[1][i]*1e8)
```

附录 9：第三小题宏观经济冲击机制识别

```
import numpy as np
import pandas as pd
np.random.seed(0)

data0 =pd.read_excel('302家企业总收益与成本.xlsx', sheet_name='Sheet2')
com =data0['企业代号'].values
categories =data0['分类'].values
miu =[-0.4, -0.085, -0.125, -0.134, 0.2, -0.025, -0.135, -0.467, -0.15]
sigma =[0.007, 0.008, 0.0075, 0.0075, 0.0071, 0.0084, 0.0075, 0.0046, 0.007]
mai =data0['销货额'].values
up =data0['上游中心性'].values
down =data0['下游中心性'].values
pro =data0['class'].values

sui =[]
sui2 =[]
sui3 =[]
for i in range(9):
sui.append(np.random.normal(loc=miu[i], scale=sigma[i]))
sui2.append(np.random.normal(loc=(miu[i] /10), scale=(sigma[i])/10))
sui3.append(np.random.normal(loc=miu[i], scale=sigma[i]))

mai2 =[]
up2 =[]
down2 =[]
pro2 =[]

for j in range(277):
category =categories[j]
mai2.append(mai[j] *(1 +sui[category]))
up2.append(up[j] *(1 +sui2[category]))
down2.append(down[j] *(1 +sui2[category]))
pro2.append(pro[j] *(1 +sui3[category]))
dataframe =pd.DataFrame([mai2, up2, down2, pro2])
dataframe.to_csv('eta_冲击计算结果.csv')
```

附录 10：第三小题单行业冲击机制识别

```
import numpy as np
import pandas as pd
np.random.seed(0)

data0 =pd.read_excel('302家企业总收益与成本.xlsx', sheet_name='Sheet2')
com =data0['企业代号'].values
categories =data0['分类'].values
miu =-0.4
sigma =0.08
mai =data0['销货额'].values
up =data0['上游中心性'].values
down =data0['下游中心性'].values
pro =data0['class'].values
```

```
sui =np.random.normal(loc=miu, scale=sigma)
sui2 =np.random.normal(loc=(miu/10), scale=(sigma/10))

mai2 =[]
up2 =[]
down2 =[]
pro2 =[]

for j in range(277):
category =categories[j]
if category ==0:
mai2.append(mai[j] *(1 +sui))
up2.append(up[j] *(1 +sui2))
down2.append(down[j] *(1 +sui2))
pro2.append(pro[j] *(1 +sui))
else:
mai2.append(mai[j])
up2.append(up[j])
down2.append(down[j])
pro2.append(pro[j])
dataframe =pd.DataFrame([mai2, up2, down2, pro2])
dataframe.to_csv('单行业.csv')
```

附录 11：模型验证——lstm 模型训练

```
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras import layers, Sequential, optimizers
import pandas as pd
import numpy as np


def f(a):
for k in range(len(a)):
a =list(a)
if a[-1] ==0:
b =a.pop(-1)
a.insert(0, b)
return a


def cross_entropy(y_true, y_pred):
a =y_true *tf.math.log(y_pred)
return -tf.reduce_mean(a)

data0 =pd.read_excel('data.xlsx')
data1 =pd.read_csv('expend.csv')
data2 =pd.read_csv('feis-up.csv')
data3 =pd.read_csv('profits.csv')
data4 =pd.read_csv('feis-down.csv')
x =np.zeros((123, 38, 6))
y =np.zeros((123, 4))
for i in range(123):
x[i, :, 0: 2] =data0.iloc[i][4: 6]
x[i, :, 2] =f(data1.iloc[i][1:])
```

```
x[i, :, 3] =f(data3.iloc[i][1:])
x[i, :, 4] =f(data2.iloc[i][1:])
x[i, :, 5] =f(data4.iloc[i][1:])
label =data0.iloc[i][8]
y[i][label] =1
if label ==0:
y[i][label] =12
if label ==3:
y[i][label] =0.15
if label ==1:
y[i][label] =0.9



x =tf.convert_to_tensor(x, dtype=tf.float32)
y =tf.convert_to_tensor(y, dtype=tf.float32)
db =tf.data.Dataset.from_tensor_slices((x, y)).batch(8)

model =Sequential()
model.add(layers.LSTM(units=20, return_sequences=True)) # 38 x 20
model.add(tf.keras.layers.Dropout(0.4))
model.add(layers.LSTM(units=20, return_sequences=False)) # 1 x 20
model.add(tf.keras.layers.Dropout(0.4))
model.add(tf.keras.layers.Dense(units=4, activation='softmax')) # 1 x 4


model.compile(optimizer=optimizers.Adam(lr=1e-3),
loss=cross_entropy, metrics=['accuracy'])
history =model.fit(db, epochs=100)

model.save_weights('ckpt/weights.cpkt')
```

附录 12：模型验证——lstm 模型预测

```
import tensorflow as tf
from tensorflow.keras import layers, optimizers
import pandas as pd
import numpy as np
data0 =pd.read_excel('data2.xlsx')
data1 =pd.read_csv('expend2.csv')
data2 =pd.read_csv('feis-up2.csv')
data3 =pd.read_csv('profits2.csv')
data4 =pd.read_csv('feis-down2.csv')
x =np.zeros((302, 38, 6))


def f(a):
for k in range(len(a)):
a =list(a)
if a[-1] ==0:
b =a.pop(-1)
a.insert(0, b)
return a
```

```
for i in range(302):
x[i, :, 0: 2] =data0.iloc[i][8: 10]
x[i, :, 2] =f(data1.iloc[i][1:])
x[i, :, 3] =f(data3.iloc[i][1:])
x[i, :, 4] =f(data2.iloc[i][1:])
x[i, :, 5] =f(data4.iloc[i][1:])

x =tf.convert_to_tensor(x, dtype=tf.float32)


model =tf.keras.Sequential()
model.add(layers.LSTM(units=20, return_sequences=True))
model.add(tf.keras.layers.Dropout(0.4))
model.add(layers.LSTM(units=20, return_sequences=False))
model.add(tf.keras.layers.Dropout(0.4))

model.add(tf.keras.layers.Dense(units=4, activation='softmax'))

model.compile(optimizer=optimizers.Adam(lr=1e-3),
loss=tf.losses.CategoricalCrossentropy(), metrics=['accuracy'])
model.load_weights('ckpt/weights.cpkt')

y =model.predict(x)
labels =['A', 'B', 'C', 'D']
idx =tf.argmax(y, axis=1)
result =[]
for i in idx:
result.append(labels[i])


dataframe =pd.DataFrame(result)
dataframe.to_csv('预测结果.csv')
```