摘　要

在现代市场经济中，风险成为了金融市场的属性之一，没有风险便没有金融活动。对于绝大多数的投资者来说，金融决策的核心关注点便是收益和风险的权衡。卢佐冬[1] 在 2020 年的一项研究中表明，投资者没有绝对的理性人且投资者具有风险厌恶特征。但是在现实金融环境的生产生活中，往往需要承受一定的风险才可以实现盈利，因为金融投资活动本身是具有成本的。基于风险和收益需要得到合理衡量的背景下，Markowitz 的均值-方差模型(下文简称 M-V 模型)诞生了，使得风险和收益在数学上的量化具有了指导。而基于均值-方差模型的投资组合优化策略被证实是 NP-hard 问题，难以在有限的时间内通过非线性规划的方法求解最优解。而金融领域中的数据走向则大多是典型的时间序列模型，所以此时，在有限的时间内，求解投资组合优化中的较优解则具有了意义。粒子群优化算法（Particle Swarm Optimization, PSO）是由 J. Kennedy 和 R. C. Eberhart[2] 在 1995 年所提出的启发式进化算法，由于其具有快速收敛，实现简单的特点，所以周佳莉[3] 提出了粒子群算法在工程领域，经济领域以及数学领域等不同方面的应用。本文针对投资组合优化策略的问题，基于现实生产生活的限制对普通 PSO 进行了改进，提出了 NAPSO(Natural Adaptive PSO)。本文的主要工作如下：

1) 针对投资组合优化问题，并考虑到算法效率的限制，结合金融市场低风险与高收益不可兼得的“无免费午餐”经验性定论[4]，基于 M-V 模型，提出了合适的适应度评估函数，将多目标问题合理地降维为单目标问题，从而避免了天文数字的迭代次数和浮点运算，提高了算法效率，从而使 PSO 算法在对于实时性相当严格的投资优化组合领域上具有了可行性。
2) 针对投资组合优化问题，使用 1)中提出的适应度评估函数进行了经典 PSO 算法的实现并使用投资组合优化问题进行编码仿真实现，试验结果表明经典 PSO 算法对投资组合优化的较优解求解的准确性较高，但是仍存在运行效率问题和易收敛到局部最优点的缺陷。
3) 基于 2)面临的实际工程问题，吸收其他的启发式进化算法的优点，选取合适的加速和变异算子，在经典 PSO 算法的基础上使用新的粒子位置更新机制，并加入了从退火算法中的领域搜索变异的机制，使粒子在收敛过程中保持全局粒子的多样性，增加收敛过程的随机性，避免了早陷于局部最优点的情况
4) 运用正态分布随机模拟金融市场对经典 PSO 和 NAPSO 算法进行测试，并比较测试结果，用实验数据论证 NAPSO 相对于经典 PSO 的优越性。

关键词：投资组合优化；均值-方差(M-V 模型)；经典 PSO；NAPSO；

ABSTRACT

In the modern market economy, risk has become one of the attributes of financial markets, and there is no financial activity without risk. For most investors, the core focus of financial decisions is the trade-off between returns and risks.

Lu Zuodong [1] showed in a 2020 study that investors do not have absolute rational people and that investors have risk aversion characteristics. However, in the production and life of the real financial environment, it is often necessary to bear certain risks in order to achieve profit, because financial investment activities themselves have costs. Based on the background that risks and returns need to be reasonably measured, the Markowitz mean-variance model (hereinafter referred to as the M-V model) was born, so that the mathematical quantification of risks and returns has guidance. However, the portfolio optimization strategy based on the mean-variance model proved to be an NP-hard problem, and it was difficult to solve the optimal solution by nonlinear programming in a limited time. The data trend in the financial field is mostly a typical time series model, so at this time, in a limited time, it is meaningful to solve the better solution in portfolio optimization. Particle Swarm Optimization (PSO) is a heuristic evolutionary algorithm proposed by J. Kennedy and RC Eberhart [2] in 1995. Due to its fast convergence and simple features, Zhou Jiali [3] proposed The applications of particle swarm optimization in engineering, economics, and mathematics are discussed. Aiming at the problem of portfolio optimization strategy, this paper improves ordinary PSO based on the constraints of real production and life, and proposes NAPSO (Natural Adaptive PSO). The main work of this article is as follows:

1) Aiming at the problem of portfolio optimization and taking into account the limitation of algorithm efficiency, combining the empirical conclusion of "no free lunch" that cannot combine both low risk and high return in the financial market [4], based on the MV model, a suitable fitness level is proposed The evaluation function reduces the multi-objective problem to a single-objective problem reasonably, thereby avoiding the number of iterations of astronomical numbers and floating-point operations, and improving the efficiency of the algorithm, so that the PSO algorithm can be used in the field of investment optimization portfolios with strict real-time performance It is feasible.

2) For the problem of portfolio optimization, use the fitness evaluation function proposed in 1) to implement the classic PSO algorithm and use the portfolio optimization problem to implement coding simulation. The test results show that the classic PSO algorithm has a better solution for portfolio optimization. The accuracy of the solution is high, but there are still problems of operating efficiency and easy convergence to the local best.

3) Based on the actual engineering problems facing 2), absorb the advantages of other heuristic evolutionary algorithms, select appropriate acceleration and mutation operators, use the new particle position update mechanism based on the classic PSO algorithm, and add Search for the mutation mechanism from the field in the annealing algorithm, so that the particles maintain the diversity of the global particles during the convergence process, increase the randomness of the convergence process, and avoid the situation of being trapped in the local best advantage.

4) Test the classical PSO and NAPSO algorithms by randomly simulating the financial market with a normal distribution, and compare the test results. Use experimental data to demonstrate the superiority of NAPSO over classic PSO.

Keywords: portfolio optimization; mean-variance (M-V model); classic PSO; NAPSO;

目　　录

第一章　绪论

1.1 研究背景以及业内研究概况

二战结束后，欧美经济发展，金融市场迅速复苏，投资需求日益增大。投资的本质便是在收益和风险之间做出选择。在此之前的投资方法论中，投资者权衡收益和风险多半依靠的是投资者自身结合外部信息所做出的主观判断，并没有一个量化收益和风险的手段。

在此背景下，经济学家马科维兹在论文《资产组合的选择-投资的有效分散化》一文中，提出了均值方差分析模型和投资组合有效边界模型，标志着现代投资组合理论的正式诞生。这套理论在实践中被证明是有效的，是现代金融学的开端。

投资是通过节省现在的消费，期望获得未来更高的消费能力的一种行为。收益是获得未来消费能力中高出现在的部分，风险代表了获取收益的不确定性。投资时要首先要考虑的两个因素就是收益和风险。马科维茨的均值方差分析方法就描述了如何测定和平衡这两个因素。

未来的收益是不确定的，是一个随机变量，因此不能用一个确定的数值来描述收益率。一般通过概率分布来描述随机变量。概率分布的两个重要数字特征是均值和方差。

均值方差分析方法在描述收益率时，使用概率分布的均值。由于均值也称数学期望，所以这个描述被称为期望收益率，记做 E(R)。在描述风险时，使用概率分布的方差，记做 Var(E)。方差代表了分布的一种离散程度，而离散程度正好可以刻画不确定性。根据概率分布知识可知：

$$\mathrm{Var(E)=E[R-E(R)]^2}$$

如果进行多个投资，那么就得到了一个投资组合。那怎么根据单个投资去进行最优的组合？投资组合和各个投资的收益和风险有什么关系呢？

根据前边的均值方差分析方法，可以把问题转换一下，转换为研究联合分布和单个分布之间的关系。马科维茨投资组合有效边界模型就是通过这个研究得出的。

马科维茨观察不同投资的组合，并推导出各投资在不同权重下的风险-收益曲线，也就是联合分布的方差（标准差）-均值曲线。最终，他发现，存在一条最优组合曲线，对于给定收益率（均值），这条曲线上的投资组合有最小的风险（方差），这条曲线就是有效边界，也称有效前沿。

这个过程需要强大的计算能力，因为计算组合的方差时，根据单个投资权重和投资间协方差进行加和。曲线上每一个点都需要进行这个计算。例如，有 n 个投资的投资组合，就有 n(n-1)/2 个协方差需要计算，计算量和投资个数是平方关系。这个均值和方差的计算过程可以用以下公式描述：

$$E(R_P)=\sum_{i=1}^{n} w_i E(R_i)$$

$$\sigma_P^2=\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j Cov(R_i,R_j)$$

而投资者在面临投资组合优化问题的时候，最为关心的两个问题便是风险与收益。而 M-V 模型的理论基础则让量化在一定收益下的风险具有了数学基础。对于基本 M-V 模型的求解，Markowitz 本人提出了传统的二次规划方法。但是对于投资组合优化这种 NP-hard 的问题，想在有限的时间内求解出最优解是非常难的。从现实中抽象出其数学模型，这个问题的实质便是有约束多目标规划问题。随着问题维数的增多，计算的复杂度会呈指数型上升。

当前常用的投资组合选择策略则是从 Pareto 最优解集中根据投资者自身的需求选择合理的投资组合方案。Pareto 最优在经济学的定义如下[5]：在不令任何一个目标变差的情况下，无法通过改变资源分配的方式令至少令一个目标变好。而具体到投资组合优化问题这种双目标问题中，则是在不让风险提高，收益降低的情况下，无法通过改变投资比例的方式，达到至少实现让收益上升或风险降低的目标。

目前的主流研究方向分别有三个方向：

首先定义最小化处理：若一个目标为评估函数值 f(x)越大，解越具有优越性，则 此时用 1/f(x)替代其原评估函数

a) 通过有约束多目标非线性规划优化的方法求解 Pareto 最优解集

首先确定非线性规划的一般数学形式如下：

(此处对目标评价函数做了最小化处理)

$$minf(x)\begin{cases} A_x \le B \\ Aeq.x = Beq \\ C(x) \le 0 \\ Ceq(x) = 0 \end{cases}$$

若在多目标情况下则求解的目标则为 **min{$f_1(x)$,$f_2(x)$……$f_i(x)$}**，其中 i 为目标的 个数。由于非线性规划本身便是对计算机算力要求非常高的 NP-hard 问题，再加 上若干个互相制约的目标函数，一个目标的变优可能会导致另一个目标的变差。

对于此类问题，工业上常用的方法有：约束法，分层序列法，功效系数法，评价 函数法。

约束法的主要原理为从众多的目标中找出一个最主要的目标，然后给让其他的 目标的适应度处于某个及格线之上的区间。最后以最主要的目标的最优值为主要求解目标，其他的目标则作为约束条件，数学描述如下：根据投资者自身的需求 设定 i-1 个目标的及格临界值 $f_1^o, f_2^o, f_3^o, f_4^o \ldots\ldots f_{i-1}^o$，令约束条件的集合为 g(x)，做 最小化处理。此时，求解方程等价于：

$$\begin{cases} f_k(x) < f_k^o(x) \\ \min f_i(x) \\ g(\mathrm{x}) \le 0 \end{cases}$$

此处 **g(x)**为约束条件，**k∈[1，i-1]**

分层序列法是根据多目标的目标重要程度进行排序，对每一个目标进行单独 的非线性规划处理，并且每一个目标的约束条件依赖于前一个目标的约束条件，形成一条马尔科夫链，数学描述形式如下：

f_1 ，f_2 ，f_3 ，f_4 ……f_{i} 将 i 个目标按照重要程度排序，假设经过由次要到重要的顺序排序后的的目标的符号为 $F_1, F_2, F_3, F_4 \ldots\ldots F_{\mathrm{i}}$，先求解以下方程

$$\begin{cases} \min F_1(x) = F_{1*} \end{cases}$$

对于余下的$F_2, F_3, F_4 \ldots\ldots F_{\mathrm{i}}$，则进行以下求解步骤

$$\begin{cases} \min F_{\mathrm{k}}(x) = F_{\mathrm{k}*} \\ x \in R \cup \{x | F_{\mathrm{j}}(x) \leq F_{\mathrm{j}*}, \mathrm{j} = 1,2,3, \ldots\ldots, \mathrm{k}-1\} \end{cases}$$

最终取 k=i 时的解集作为结果解集

功效系数法为将不同量纲的多目标最优问题转化为求解相应功效系数问题的 方法。

评价函数法是将多目标最优问题转化为求一个评价函数最优解的问题，而将 多目标如何集成在一个评价函数内则在多目标领域中有独立的多目标转化为单目标问题的研究。

b) 通过多目标进化算法求解 Pareto 最优解集

对于单目标的进化算法，由于适应度评估函数只有一个，所以通过函数值直 接比较解的优劣。但对于多目标进化算法，则需要引入以下两个概念：

b.1) 支配

若对于解 X_1,X_2，存在以下关系(对于所有的目标做最小化处理)

$$\forall j \in [1, \mathrm{i}], f_{\mathrm{j}}(x_1) \leq f_{\mathrm{j}}(x_2)$$

则称解 X_1 支配解 X_2，若有一个解支配其余所有的解，则称该解为支配解；若在全局解集中存在解 a 被另一个解 b 所支配，则解 a 称为被支配解。

若一个解为被支配解，则会对该解进行惩罚，令它对后续解的迭代更新造成更小的影响

b.2) 拥挤距离

拥挤距离是用来衡量全局解集多样性的标准，具体计算流程如下：

若一个解的拥挤距离越小，则直观说明在解空间中，该解附近的解非常少，　故该解可以对全局解集的解的多样性作出较大贡献，有较好的保留理由。会对其进行奖励，使其对后续解的迭代更新造成更大的影响。

makeGroupDistance(N):　　输入：N(当前全局解集)

作用：　计算当前全局解集中的各个解的拥挤距离

```
1.  for i in N:
2.      if i is N.first() or i is N.last():   //假如当前是遍历的开头或末尾
3.          i.groupDistance = +∞
4.      i.groudDistance = 0    //遍历全局解集，令一个全局解集的拥挤距离为 0
5.  retValue = 0    //将需要求解返回的值初始化为 0
6.  for i in range(len(N[0].objects)):   //进入第二个循环，循环次数为多目标中目标的个数减 2
7.      calGroupDistance(i+1，N)
```

calGroupDistance(j,N):　输入:j(当前进行拥挤距离计算的粒子的序号),N(当前全局解集)

作用：计算当前粒子的拥挤距离

```
1.  objectsNum = len(N[j].objects)
2.  for i in range(objectsNum):
3.      N[j].groupDistance += |N[j+1].fi()-N[j-1]. fi()| //其序号相邻的两个解
```

而当前解集的相互支配关系和拥挤距离的应用则要视乎具体的进化算法，在目前工业上比较常用的多目标进化算法是 K. Deb, A. Pratap, S. Agarwal and T. Meyarivan[6]提出的 NSGA-II 算法则是根据快速支配排序分层，拥挤度估计和精英粒子策略计算全局较优解集的多目标进化算法。

1.2 各个主流方向面临的难题

1.2.1 对于有约束多目标非线性规划的难题

由 1.1 中的提及的约束法，分层序列法，功效系数法，评估函数法都具有一个非常难以解决的问题，那便是客观的量化过程前都必须先用一次极为主观的目标重要性评估才可以。故如此不仅具有难复现性，而且更加会因为人的主观评价而影响解的质量。对于本问题的投资者来说具有的实际意义相对较小。

此外，即便是对于单目标的非线性规划算法，在运算过程中使用到的牛顿迭代法，梯度下降，矩阵乘法在金融领域庞大的数据量下需要巨额的算力才具有生产价值。

1.2.2 对于多目标进化算法的难题

对于多目标进化算法，问题也是相似的。由于设计到支配和拥挤度估计的概念，在未出现新的多目标评估机制时，$O(n^3+n^3)$的算法复杂度是该机制下无法避免的算法复杂度，而这种计算效率对于金融领域的实时性来说实用性有待商酌。并且对于

投资者来说，在帕累托解集中还需要自行选择需要的解，而这显然增加了投资者的投资成本，所以在实际投资组合优化的问题中还有巨大的需要改进的空间。

1.3 本文研究内容

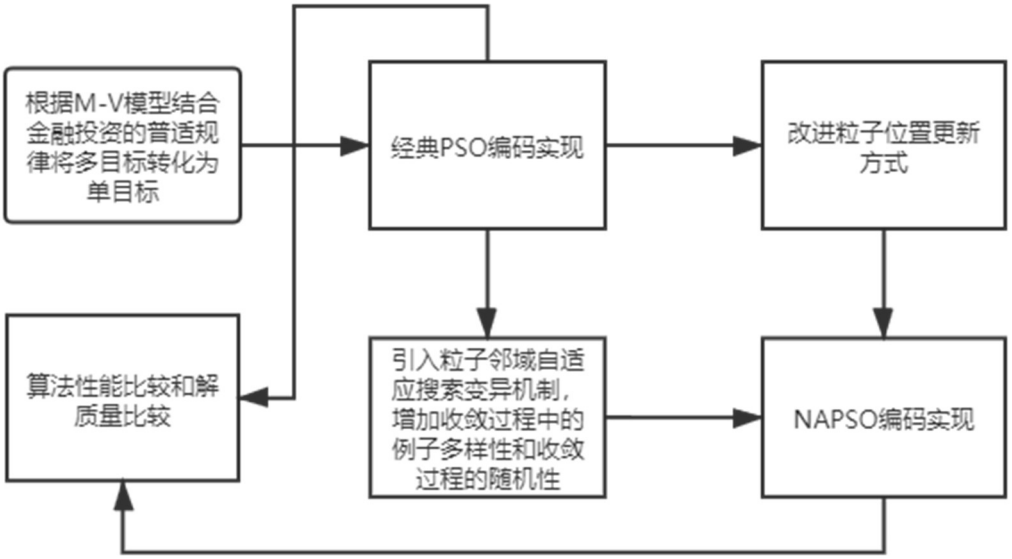


图 1.1　本文主要研究脉络

本文基于目前的主流研究方向，运用主流的研究方向的思路，结合了评价系数法和进化算法的思路。

当目标个数越少的时候，多目标在合理的维度上归一为单目标难度越小。考虑到 M-V 模型只有两个目标，此时运用线性规划中常用的评价系数法将多目标问题化为单目标问题具有可行性。

当问题变为单目标问题后，此时再使用进化算法，便可避免 NP-hard 的最优解求解难度。并且由于为单目标问题，进化算法的算法复杂度在可接受范围之内，此时便可求解对于投资者来说局部的较优解去换取时间上的便捷。

第二章　合理结合 M-V 模型进行多目标归化单目标

2.1 M-V 模型的数学形式

在 Markowitz 提出的 M-V 模型中，对于投资组合的评估是从风险和收益两方面进行评估的。该模型基于以下假设条件：

1) 投资者在考虑每一次投资选择时，其依据是某一持仓时间内的证券收益的概率分布。

2) 投资者是根据证券的期望收益率估测证券组合的风险。

3) 投资者的决定仅仅是依据证券的风险和收益。

4) 在一定的风险水平上，投资者期望收益最大；相对应的是在一定的收益水平上，投资者希望风险最小。

5) 证券市场是有效的，证券的价格反应证券的内在价值，且不存在交易费用和税收。

6) 投资者是根据证券的期望收益率的方差衡量收益率的波动情况

7) 证券收益率的分布符合正态分布

8) 证券之间并不是两两独立的，证券之间的收益率具有一定的相互影响关系

9) 投资者允许卖空和买空

该模型的数学形式描述如下：

在尽可能满足该条件的情况下：$\min(x_i x_j \sigma_{ij})$

尽可能地令$\sum_{i=1}^{N} x_i u_i$的值尽可可能大

约束条件为：**1)** $\sum_{i=1}^{N} x_i = 1$

2) $x_i \geq 0$

其中 $\mathbf{X_i}$，$\mathbf{X_j}$ 分别为第 i 支证券和第 j 支证券的各自的投资比例，σ_{ij}则是第 i 支证券和第 j 支证券在该时间段内的不同时间点记录的多个收益率的方差。μ_i则是第 i 支证券在一定时间段内的平均收益率。

可见 M-V 模型的原始数学形式是一个多目标问题，并没有在数学形式上将风险和收益归入同一方程处理。故直接使用是具有实现难度的，且带有投资者主观意识的影响。故本文将基于 M-V 模型将提出一种多目标归化单目标的评价函数。

2.2 多目标归化为单目标适应度评估函数

单独衡量风险和收益两个目标的评估函数，可显然由 M-V 模型可得：

风险评估函数：$x_i x_j \sigma_{ij}$

收益评估函数：$\sum_{i=1}^{N} x_i u_i$

从数学和金融领域的角度上建立将两者统一在同一方程里是非常具有难度的。但是根据进化算法在解空间内随机搜索的特点，可以在保证趋势大致正确的

基础上在一定程度忽略某些数学上的不严谨，由进化算法自主的寻优性产生较优解。

根据投资理论中的经验结论，资产价格的变化是一个布朗运动。即资产价格在指定时间内，可能的变化服从正态分布。总而言之，资产上升和下降的概率是相等的，价格上升意味着收益，反之则意味着亏损。

在实际的生产生活中也有可能出现极端偶然情况，假设存在一个好的证券资产具有收益高，风险低的特点。但由于市场的有效性和当代社会信息的高度流通性，其价格回迅速上涨逐渐以致于收益降低。反之，若存在一个坏的证券资产，具有风险高，收益低的特点，同上述理由，最终也会向均值水平回落，证券资产价格逐渐降低，收益逐步增大。

因此，尽管难以确切量化风险和收益的数学关系，但由现实的投资组合模型经过简化分析后，至少是可以确定风险和收益至少是呈正比例的关系。

在进化算法中，为了避免数据的溢出，一般会对优化目标的评估函数做最小化处理。且在多目标问题中为了统一目标的优化方向，一般都会对目标值向大而优的目标做最小化处理。而在本问题中，风险值是值小而优，而收益值则是值大而优，此时应该对收益值做最小化处理。

对收益值做最小化处理之后，风险与处理后的收益值便成了反比例关系。即有以下结论：

$$\frac{1}{\sum_{i=1}^{N} x_i \mu_i} \text{与} \quad (x_i x_j \sigma_{ij}) \text{ 呈反比例关系}$$

至此，由于目标只有两个，所以用二维空间便可表示全局解集。且由风险和收益至少是呈正比例的结论可确定帕累托曲线大致如下：

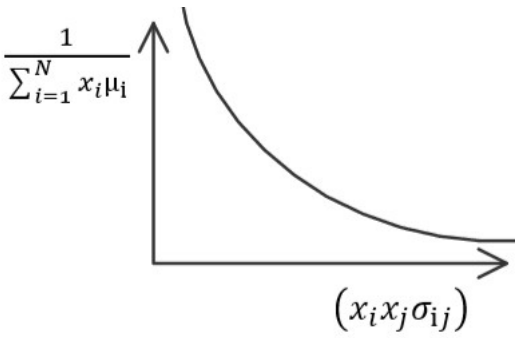


图 2.1　大致帕累托曲线

而在金融领域中，极端地追求低风险意味着收益低，而投资活动本身就具有成本，此时收益小于成本的话对于投资者来说便是亏损。极端地追求高收益则意味着高风险，违背了大多数投资者风险厌恶的心理。所以在上述坐标系的近似中，接近风险轴或收益轴的解应该被放弃，而输出的理想解应该为没有明显的极低风险或极高收益的倾向。并且变量间呈反比关系，故帕累托曲线应该是个凹函数，对于凹函数曲线中在第一象限中不接近两坐标轴的曲线上的点应具有接近相对接近原点的特征。

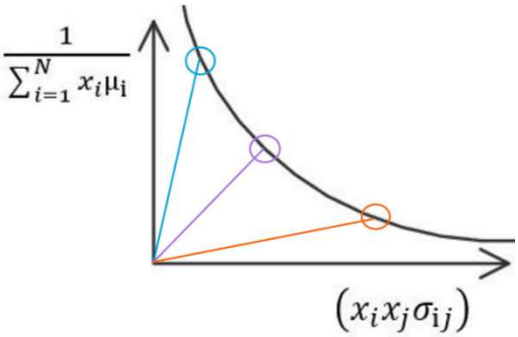


图 2.2　帕累托曲线上的解的优劣性比较

如图 2.2 所示，从帕累托曲线上随机选取三个解，根据颜色分别命名为蓝解，紫解，和橙解。显然可见蓝解和橙解距离原点的距离大于紫解。从定性角度分析蓝解和橙解，蓝解具有低风险低收益的特点，橙解具有高风险高收益的特点，皆不符合投资者的需要。而紫解所在的位置则是对风险和投资进行了折中，从长远发展来看对投资者更有利。

经过分析可得，可根据与原点的距离判度解的优劣性，故有适应度函数如下：

$$Fitness(X) = \sqrt{(\frac{1}{\sum_{i=1}^{N} x_i \mu_i})^2 + (x_i x_j \sigma_{ij})^2}$$

此处的 X 为解，为某个确定的投资组合，即投资者在不同证券上的投资比例。其他符号的含义同 2.1 中所述。

由于适应度函数仅用于比较解的优劣性，所以此处不必严格地按照物理意义上的距离。为了提高算法的效率，可将适应度函数稍作改进，如下：

$$Fitness(X) = (\frac{1}{\sum_{i=1}^{N} x_i \mu_i})^2 + (x_i x_j \sigma_{ij})^2$$

至此，多目标问题被转化为单目标问题。

第三章　经典 PSO 对投资组合优化问题的解决

3.1 PSO 算法原理

粒子群算法是通过模拟鸟群捕食行为设计的一种群智能算法。区域内有大大小小不同的食物源，鸟群的任务是找到最大的食物源（全局最优解），鸟群的任务是找到这个食物源。鸟群在整个搜寻的过程中，通过相互传递各自位置的信息，让其他的鸟知道食物源的位置最终，整个鸟群都能聚集在食物源周围，即我们所说的找到了最优解，问题收敛。此处将鸟群抽象为忽略质量和体积的粒子群，从而有了粒子群算法。学者受自然界的启发开发了诸多类似智能算法，如蚁群算法、布谷鸟搜索算法、鱼群算法、捕猎算法等等。

粒子群算法相对于其他的群智能算法具有收敛速度快，实现简单的特点。算法的基本数学形式有如下：

$$V_{id} = wV_{id} + C_1 random(0,1)(P_{id} - X_{id}) + C_2 random(0,1)(P_{gd} - X_{id})$$

$$X_{id} = X_{id} + V_{id}$$

此处 $\mathbf{V_{id}}$ 为粒子的速度，w为惯性因子，即粒子保持原来的运动状态的趋势。$\mathbf{C_1}$ 为粒子的自身学习因子，$\mathbf{C_2}$ 为粒子的全局学习因子。分别衡量粒子学习过去历史中自身最优位置和当前最优位置的程度。**random(0,1)**为(0,1)区间中的随机数。$\mathbf{P_{id}}$ 为粒子的自身历史最优位置。$\mathbf{P_{gd}}$ 全局的历史最优位置。$\mathbf{X_{id}}$ 则为粒子当前位置。上述式子分别代表粒子速度更新公式和粒子位置更新公式。

对于参数的设定的经验结论为：$\mathbf{C_1} = \mathbf{C_2} = 2$，$w = 0.9$ 。

还有的经典 PSO 会根据具体问题在不同的迭代次数中使用动态变化的各个参数，而其中惯性因子的线性权重递减法是最为常用的。

Shi. Y 的一项研究[8]表明若想提高解的质量，那么粒子群算法在迭代的不同阶段，对于粒子位置更新的轨迹的需要是不同的。在迭代的初期，粒子的初始解为随机分布的，故此时粒子的更新轨迹应该具有更大的随机性从而达到粒子能快速地对评估函数的定义域内尽可能多的空间探索。而到了迭代的后期，粒子已经初步寻找了数个较优的解，此时为了保证收敛速度，随机性应缓慢下降，逐步使粒子集中。但是此时也不能让粒子的位置更新轨迹随机性太小，否则有可能会让粒子陷于局部最优解。Shi. Y 为了做到在不同的迭代阶段产生不同随机性的迭代轨迹，在此处使用了线性递减权重法，数学描述形式如下：

$$w = w_{max} - (t) \times (w_{max} - w_{min})/M$$

w_{max}是最大惯性因子，t 是当前迭代次数，w_{min}是最小惯性因子，M 为最大迭代次数。

本文使用的经典 PSO 的惯性因子便是使用线性递减权重法产生的。

算法实现伪代码如下：

3.2 经典 PSO 在投资组合优化问题中的具体实现方法

PSO (Num,IterTimes): 输入：Num(粒子群的粒子数量)，IterTimes(迭代次数)

输出：最优解

```
1. particles = randomParticles(Num) //随机生成位置随机速度随机的 Num 个粒子
2. for i in range(IterTimes): //进行 IterTimes 次迭代
3.     for j in range(Num): //每一次迭代对每一个粒子进行遍历
4.         Pid = particles[j].findPid() //寻找第 j 个粒子的历史最优位置
5.         Gid = particles.findGid() //寻找全局解集中的历史最优位置
6.         updateParticle(particles[j],Pid,Gid) //更新第 j 个粒子的位置和速度
7.         if particles[j].calFitness() > Gid.calFitness():
            //若粒子更新完之后优于全局最优位置
8.             updateGid(particles[j]) //将当前粒子的位置更新为全局最优位置
9.             updatePid(j,particles[j]) //更新当前粒子的历史最优位置，并记录历史位置
10.        else if particles[j].calFitness() > Pid.calFitness():
           //若粒子更新完之后优于当前粒子的历史最优位置
11.            updatePid(j,particles[j])   //更新当前粒子的历史最优位置
12.ret particles.findGid() //返回历史最优位置
```

3.2.1 粒子位置的具体编码方法

最终投资者感兴趣的解答是对于每个证券的投资比例，所以此处的投资组合使用数组或者列表结构存储 investScale 对于各个证券的投资比例，如下所示

$$[x_1,x_2\cdots\cdots x_n]$$

假设有 n 支证券，x_i 表时对第 i 支证券的投资比例，**(1<=i<=n)**

由证券的比例和为 1 且每个证券的投资比例大于等于 0 可知该结构的内部数据其中具有以下限制：

$$\sum_{i=1}^{n} x_i = 1$$

$$1 \geq x_i \geq 0，\ i\in[1,n]$$

3.2.2 粒子速度的具体编码方法

此处的例子速度使用数组或者列表结构 vector 存储粒子在各个维度上的分速度，如下所示

$$[v_1,v_2\cdots\cdots v_n]$$

粒子的速度向量在理论上是不应该具有任何限制的，本文对于粒子速度没有限制。但是经过实测可得，当分速度的大小没有得到限制的时候，会令 3.1.1 中描述的限制条件容易被违背，从而过多地出现了在更新后需要对粒子位置进行修正的情况，此处为了避免这种情况，若在工业实现上做出以下限制，以增加算法效率

$$1 \geq V_i \geq -1，\ i\in[1,n]$$

3.2.3 粒子群的具体编码方法

利用面向对象的方法进行代码的编写，给单独的粒子建立一个类 **particle**，再根据 3.2.2 和 3.2.1 中提到的编码方法对其添加位置和位置的成员。

随后再建立一个粒子群类 **particles**，将一个以 **particle** 为元素，大小为粒子群中粒子数量的数组或列表添加为 **particles** 的一个核心成员。本文为方便表示，伪代码中直接用 **particles[j]**表时 **particles** 类中的第 j 个 **particle** 粒子

3.2.4 particles 类的其他需要成员

a)定义一个以 **particle** 为元素，且大小为粒子群的粒子数量的列表或数组 Pid，作为个体粒　子历史位置的记录容器。

b)定义一个整数变量 iterTimes，作为用户需要的迭代次数

c)定义一个 **particle** 变量 Gid，作为记录历史全局最优解

d)定义一个以浮点数变量为元素，大小为粒子群的粒子数量的列表或者数组 Pid_fit, 作为　记录每一个粒子的个体历史最优位置对应的适应度评估函数值。

e)定义一个以浮点数变量 Gid_fit，作为记录历史全局最优解的适应度评估函数值

f)定义一个以整数变量 nowTimes，记录当前进行迭代的次数

g)定义一个大小为 m*n 的矩阵 stock，元素为浮点数，用于记录 m 支证券 n 天来的日收益　率。

h)定义一个大小为 m*m 的矩阵，元素为浮点数，作为 stock 的协方差矩阵 **cov，cov[i][j]**　是指第 i 支证券和第 j 支证券的 n 天日收益率的方差。

3.2.5 粒子位置或速度更新后数据超出上下界时的修正方法

根据 3.1 中的位置更新公式中我们可以看到粒子的新位置是由粒子的旧位置与粒子的新速度向量相加而得的。在该过程中并没有考虑到 3.2.1 和 3.2.2 中给出的限制。所以此处需要给出数据超出限制后的修正方法，算法流程如下：

amend(List):　　输入：List（需要进行数据修正的列表或数组）

作用：对超出上下界的数据进行修正

```
1. surplus = 0 //定义两个数据记录缺少和溢出
  //速度需要做出总额限制
2. disMatch = sum(List)-1
3. surplus += disMatch
3. for i in List: //  遍历需要进行数据修正的列表或数组
4.    if List[i]<0:
5.        surplus += List[i] //加上负数维度的绝对值
6.        List[i] = 0 //将为负数的维度归 0
7.if surplus<0: //
8.    randomAdd(List,|surplus|) //将缺少的部分随机加到各个维度
9.else: randomMinus(List,surplus) //将多余的部分随机从各个维度减去
```

randomMinus(List,surplus): 输入：List（需要进行数据修正的列表或数组），surplus 多出的需要减去的部分)

作用：将 surplus 分成若开份额按照随机份额随机作为到 List 的各个减量

```
1.  while surplus>0:
2.      for i in range(len(List)):
3.          temp=random(0,1)
4.          temp = (temp if surplus>temp else surplus) if List[i]-surplus>0 else List[i]
5.          List[i] -= temp
6.          surplus -= temp
```

randomAdd (List,surplus): 输入：List(需要修正的列表或数组)，surplus(不足的部分)

作用：将 surplus 分成若开份额按照随机份额随机加到 List 的各个维度

```
1.while temp>0:
2.    for i in range(len(List)):
3.        temp = random(0,1)      // 生成一个(0,1)间的随机数
4.        temp = temp if surplus>random else surplus    // 控制增量小于 surplus
5.        List[i] += temp    // 在该维度上加上增量
6.        surplus -= temp    //增量总额减去该轮修正所消耗的增量
7.        if surplus == 0:    //增量消耗完即结束循环
8.            break
```

3.2.6 PSO 的伪代码实现

而在总体的流程的实现中则与经典 PSO 的实现并无实质性的不同，只是根据实际问题的约束要在粒子位置更新后进行一次修正操作，流程如下：

PSO (Num,IterTimes，Stock): 输入：Num(粒子群的粒子数量)，IterTimes(迭代次数)，Stock(证券数据)

输出：最优解

```
1. particles = randomInitParticles(Num，Stock,iterTimes) //随机生成位置随机速度随机的 Num 个粒子
2. for i in range(IterTimes): //进行 IterTimes 次迭代
3.    for j in range(Num): //每一次迭代对每一个粒子进行遍历
4.        Pid = particles[j].findPid() //寻找第 j 个粒子的历史最优位置
5.        Gid = particles.findGid() //寻找全局解集中的历史最优位置
6.        updateParticle(particles[j],Pid,Gid) //更新第 j 个粒子的位置和速度
7.        amend(particles[j]) //修正第 j 个粒子的位置
8.        if particles[j].calFitness() > Gid.calFitness():   //适应度评估公式使用 2.2 中推导结果
          //若粒子更新完之后优于全局最优位置
9.            updateGid(particles[j]) //将当前粒子的位置更新为全局最优位置
10.           updatePid(j,particles[j]) //更新当前粒子的历史最优位置，并记录历史位置
11.       else if particles[j].calFitness() > Pid.calFitness():
          //若粒子更新完之后优于当前粒子的历史最优位置
12.           updatePid(j,particles[j])   //更新当前粒子的历史最优位置
13.return particles.findGid() //返回历史最优位置
```

第四章　PSO 算法的改进——NAPSO

4.1 NAPSO 原理描述

NAPSO(Natural Adaptive PSO)是基于精英粒子引导的正态分布位置更新更新策略和退火算法领域随机扰动搜索结合经典 PSO 算法而来的。其放弃了在经典 PSO 中用到的速度向量机制，引入了一种随机性强的位置更新策略，如下所示：

$$x_{id} = N(\frac{P_{id}+g_{id}}{2}, |P_{id} - g_{id}|)\cdots\cdots(1)$$

按照线性递减概率进行x_{id}邻域随机扰动……(2)

此处的符号含义仍然沿用 3.1 中的定义。粒子的更新使用了正态分布的策略，方差为个体历史最优位置加上全局历史最优位置的向量和的二分之一，而方差为两者向量差的值。该方法参考自 BBPSO[10]

在进行正态分布更新后，有一定概率进行一次变异，变异的算法引入了退火算法中在退火过程的随机扰动搜索机制。其中变异的概率引用线性递减法进行动态变化

而使用这种产生新粒子位置的机制则是基于以下理由：

1. 粒子群算法是模仿鸟群觅食活动的仿生学算法，而在实际当中，鸟群个体的当前速度对　个体的下一次位置几乎没有影响。惯性的概念更适合用于非生物，对于生物来说，惯性是 可以靠个体的主动活动所克服的，所以在该算法中摒弃了速度的概念，只保留历史最优位置对于鸟群个体位置更新的影响。

而自然界的生物绝大多数都不具有强思考能力，所以选择时大多会根据自身经验和群体信息折中后进行具有随机性的选择，故此处使用均值为两位置坐标相加之和的二分之一作为正态分布的均值。

而方差则遵循正态分布的概念，方差越大，位置选择的波动性越大。而方差越小，则　位置选择的波动性越小。此处则是对鸟群个体进行定性分析所得的方差值，个体自身历史最优与全局历史最优接近时，鸟群个体会更相信在自身历史和全局历史的附近就存在自身需要的食物来源。而相差更远的时候，鸟群个体更有可能去探索之前没有探索过的区域。抽象到粒子群算法中便是用个体最优位置和全局最优位置做相差求值可得。

2. 在经典 PSO 算法中具有早熟易收敛，易陷于局部最优点的特点。后引入了线性权重递 减法等各种方法增加粒子位置更新轨迹的随机性。但此处摒弃了速度的机制，故此处引入了退火算法过程中的领域随机搜索机制，用一定概率的变异增加随机性，防止陷于局部最优点。

3. 采用这种速度更新机制相对于经典 PSO 进行了更少的浮点数矩阵运算，大

大提高了算法的运行效率。

4.2 NAPSO 在投资组合优化问题中的具体实现方法

4.2.1 对于多维变量实现正态分布的编码方法

本文为了保证实现的简洁性和在不增加算法复杂度的情况下增加粒子的多样性，此时可以对各个维度单独进行一次正态分布随机选值。

updateParticle(Particle，Gid，Pid): 输入：Particle(需要进行正态更新的粒子),Gid(历史最优位置)，Pid(个体历史最优位置)

作用：更新 Particle 的位置

```
1. for i in range(len(Particle.investScale)):
2.    loc = (Pid[i]+Gid[i])/2   //第 i 维数据相加并除以二求出均值
3.    scale = |Pid[i]-Gid[i]|    //第 i 维数据之差的绝对值作为方差
4.    Particle.investScale = normalDistribution(loc,scale)
     //利用均值和方差返回一个正态分布值
5.neighbourSearch(Particle) //对粒子进行领域搜索操作
6.amend(Particle) //修正粒子，此处沿用 3.2.5 中的沿用函数
7. if particles[j].calFitness() > Gid.calFitness():   //适应度评估公式使用 2.2 中推导结果
        //若粒子更新完之后优于全局最优位置
8.          updateGid(particles[j]) //将当前粒子的位置更新为全局最优位置
9.          updatePid(j,particles[j]) //更新当前粒子的历史最优位置，并记录历史位置
10.else if particles[j].calFitness() > Pid.calFitness():
       //若粒子更新完之后优于当前粒子的历史最优位置
11.         updatePid(j,particles[j])   //更新当前粒子的历史最优位置
```

4.2.2 粒子进行领域搜索的编码方法

neighbourSearch(Particle): 输入：Particle(需要进行领域搜索的粒子)

作用：更新 Particle 的位置

```
1. prob = (Particle.maxTimes-Particle.iterTimes)/Particle.maxTimes
2. if random(0,1)>prob:   //按照一定概率进行变异，概率按照线性递减的方式进行动态变化
3.     return
4. pos1 = random(0,1)*len(Particle.investScale) //变异的维度 1
5. pos2 = random(0,2)*len(Particle.investScale) //变异的维度 2
6. difference1 = Particle.investScale[po1]*random(0,1) //变异量
7. Particle.investScale[pos1] -= difference1
8. difference2 =    Particle.investScale[po2]*random(0,1)//变异量
9. Particle.investScale[pos2] -= difference2
10. randomAdd(Particle.investScale,difference1+difference2) //保持投资比例总和为 1
```

此处设置的发生变异的维度为两个，如果有增大随机性的需要，此处可将变异

维度增多，但是变异维度增多后会导致算法后期的收敛性相对不优秀。

4.2.3 NAPSO 的实现伪代码

NAPSO(Num,iterTimes，stock): 输入：Num(粒子群的粒子数量) iterTimes(迭代次数) stock(证券数据)
作用：输出最优投资比例 **1. particles = randomInitParicles(Num,stock,iterTimes) //初始化粒子群** **2. for i in range(iterTimes):** **4. for j in range(Num):** **5. updateParticle(particles[j])** **6.return particles.Gid** //返回历史最优位置

从该总体代码框架中使用的算法基于 4.2.1 和 4.2.2，从改进后的 NAPSO 的算法流程可直观地看到无论是实现的便捷性还是算法效率都得到优化。

在经典 PSO 中，速度的更新运用到了三次矩阵运算，位置的更新使用了一次矩阵运算。对于 NAPSO，是使用了两次矩阵运算决定了方差和均值之后直接根据正态分布决定粒子的新位置。并且在后期的变异操作是常数 O(1)的算法复杂度。

且粒子更新位置的随机波动性在正态分布下更符合现实中的金融领域，更具有统计学意义。

所以 NAPSO 算法相对于经典 PSO 算法在投资组合优化问题中具有更优越的性能。

第五章 NAPSO 和 PSO 算法的性能比较

5.1 相同测试数据下 NAPSO 和 PSO 的表现

此处主要从两个方面进行比较：效率和优越性。效率是由算法开始运行到迭代结束产生最优解的时间衡量。优越性是由最终输出的历史全局最优解的评估函数值决定。

此处用正态分布来模拟十只金融证券在六天内的日收益率，将同一组模拟证券数据分别作为 PSO 和 NAPSO 的输入数据。

在初始的测试中先进行了一次压力测试，去分别决定两种算法的迭代次数。进行压力测试的测试集如下：

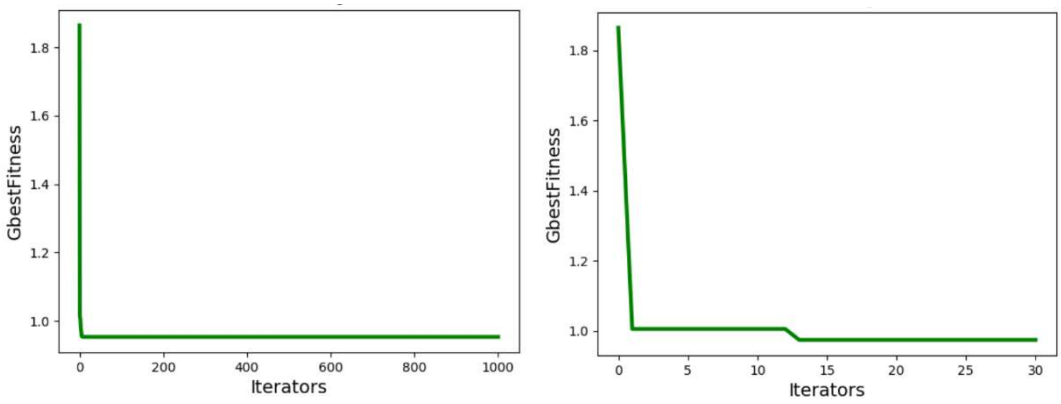
表 5.1 压力测试测试数据

统计 / 时间 / 证券序号(%)	第一天	第二天	第三天	第四天	第五天	第六天
证券 1	1.810	0.612	1.070	2.035	0.522	0.553
证券 2	-0.041	0.10	-0.377	0.097	-0.215	-0.051
证券 3	0.931	0.662	1.142	0.375	0.678	0.823
证券 4	1.407	-0.099	0.551	-0.577	-0.413	-0.045
证券 5	0.755	0.791	0.877	0.880	0.779	0.799
证券 6	0.698	0.283	-0.154	0.005	-0.304	0.311
证券 7	1.748	-0.313	0.665	0.001	-1.12	-0.232
证券 8	0.373	1.628	0.776	1.048	0.579	1.072
证券 9	0.741	0.300	1.137	0.182	0.534	1.082
证券 10	0.462	1.486	0.301	0.606	0.837	0.558

表格内[i，j]的数据代表证券 j 在第 i 天的日收益率百分比值。

压力测试结果如下：

图 5.1 压力测试结果



其中左图为 NAPSO 的最优历史位置计算出的适应度函数值随着迭代次数的

统计图，而右图则是经典 PSO 的相同含义坐标轴的统计图。

压力测试中的其余测试数据为：

NMPSO 迭代经 1000 次，耗费 28.7 秒，输出最优解的评估函数值为 0.95

PSO 迭代经 30 次，耗费 967.6 秒，输出最优解的评估函数值为 0.97

以上测试结果的运行条件为：

环境：Python3.7

CPU：Inter(R) Pentium(R) CPU 4415Y @ 1.60GHz 1.61GHz

RAM：4GB

由结果可得，对于 30 次的迭代经典 PSO 耗费的时间过多，不利于实验的进行。故在后续的测试中调整经典 PSO 的迭代次数为 15 次。而 NAPSO 的收敛速度极快，在不到 100 次的迭代中已寻求到了全局最优解，此时将 NAPSO 的迭代次数调整为 200 次。

后续再次用正态分布的方法模拟证券测试集进行测试，后续测试集规模及形式同压力测试集，测试次数为 120 次，测试结果如下：

算法 / 比较方面	经典 PSO	NAPSO
平均用时	64.9 秒	2.7 秒
用均最优解评估函数值	1.001	0.841

由测试结果可得， NAPSO 解的优越性略优于经典 PSO，而在运行效率方面则远优于经典 PSO。结果表明，本文对于经典 PSO 做出的改进方法在投资组合优化领域是具有有效性的。

结束语

目前在学术界中 PSO 算法是解决非线性连续优化问题，组合优化问题和混合整数非线性优化问题的有效优化工具，因为这些问题都是 NP-hard 的问题。将算法落地到实际的前沿产品中则是在函数优化，神经网络参数训练，模糊系统控制以及其他作为其他进化算法的算子应用。

而在工业界中 PSO 具有潜力的应用包括系统设计，多目标优化，分类，模式识别，信号处理，机器人实时路径，车间作业调度，物流路线调度，时频分析等具体的应用。

总而言之，PSO 算法的应用十分广泛，而且正有机会应该变得更加广泛。随着当前 5G 网络的普及，万物互联开始进入现实。有更多的实时自动化实体设备可以突破在网络技术方面的瓶颈而诞生，此时高效率的优化决策算法就显得尤为重要。此时 PSO 算法作为一种收敛迅速，应用成熟，不需要超精确建模的进化算法就可以作为一种合适的技术解决在现实世界中具有多样性的人机交互问题。

未来对于 PSO 算法的展望应该着重于三个方面：

1) 研究 PSO 算法的适用范围

没有一种进化算法可以适用于所有问题，优秀如 PSO 也必定具有自身的的局限性，故对于各个领域上的难以通过线性规划或非线性规划精确解决的问题，可以考虑使用 PSO，在逐步的探索中总结出 PSO 算法适用的问题的特征。

2) 研究 PSO 算法的数学原理

数学是最严谨高效的语言，只有在群智能算法的理论上得到突破，才可以从量的突破变为质的突破。很多时候在工业上不要求严谨，只要求效果。但是如果没有理论上的完善的话，应用的瓶颈很快就会达到，就像如今的深度学习。故应该着手于研究群智能理论的扩展和定义，在数学层面上解释 PSO 算法的工作原理，只有这样才可以有针对性地改进粒子更新机制和学习机制。并且正确的 PSO 算法收敛性分析方法可以帮助科研人员在改进 PSO 算法时顾及到算法效率的问题，使技术更容易落地于工业应用中。

3) 多目标规划问题的三个方向

a) 将多目标归一化为单目标，Zhang. Q 在一项研究中曾经有提出了 MOEA/D 的多目标归一化方法。具体的方法是把多目标每一个子目标归一化，给每个目标赋予一个权值，通过不断调整这些权值，从而获得一组不同权值的单目标问题，再去同时解决每一组权值下这一组问题。比如只考虑 2 个目标，权值则可以分解为(10,0),(9,1)……(0,10)11 组权值，若对精确度的要求比较高，则可以将权值的增量设定得更小。除此之外还有其他的研究在论证多目标归化为单目标的可行性。本文认为多目标归化为单目标是在牺牲求解精度从而换取工业实现的简便性和算法性能的效率，在某些问题中为非常可取的措施，也非常值得继续研究。

b) 继续研究多目标机制下的粒子优劣性比较

目前在多目标规划中最常见的 PSO 算法解的优劣性的比较用到的便是支配关系和拥挤度计算，而能否研究出新的机制做到更好的比较多目标问题，则是非常值得研究的一个问题。目前的方法具有研究精度也非常地高，但是常用的精英粒子存档和拥挤度估计造成算力要求还是太高了，无法大量地技术落地成产品。故多目标问题中优劣性比较需要更优秀的比较机制。

c) 分治思想是否可以运用在 PSO 算法中

a)和 **b)**提到的方面本文作者并不认为其是对立或互斥的，而是可议并存的。多目标归化为单目标问题中，若目标个数越少，则规划难度越低。就像本文对投资组合优化问题进行了定性分析，且该问题的目标只有风险和收益故选择使用多目标归化为单目标的思路进行 PSO 算法的改进。如若目标个数太多，则此时归化为单目标的难度就会指数型上升。但是如果使用分而治之的思想，将多目标问题进行定性分析，将关系较大的目标两两之间进行组合，再运用多目标归化为单目标的方法，将两目标归化为单一目标，重复此过程，直至剩余目标无法再进行归化。则采用多目标进化算法进行解决是否可取。这也是一个非常值得思考的问题，并且处于学术界研究的空白领域，故非常地具有研究潜力。

1),2),3)三个方向都是未来 PSO 算法可以继续突破瓶颈的地方，希望 PSO 算法可以为更多的优化问题带来思路。

致　谢

本论文是在我的导师×××老师的指导下完成的。导师渊博的专业知识，严肃的科学态度，严谨的治学精神，诲人不倦的高尚师德都对我产生了深远的影响。从课题的选择到项目的最终完成，×××老师都始终给予了我悉心的指导，不仅使我树立了远大的学术目标，还使我明白了许多为人处世的道理，同时还在精神和生活上给了我以无微不至的关怀。在此，谨向×××老师致以我最诚挚的谢意和最衷心的感谢！

此外，本论文的顺利完成，离不开各位老师的关心和帮助。在此还要感谢×××教授、×××老师、×××老师的指导和帮助，他们对本课题做了不少工作，帮助我克服了一个个困难，非常感谢他们！

然后，感谢学术界中研究 PSO 算法的各位科研人员，是你们的努力，才给了后辈学生们如此丰富的研究成果，我们才可以站在巨人的肩膀上看这个世界。

参考文献（三号、黑体、顶格）

[1] 卢佐冬. 个体投资者心理与行为偏差研究[D].武汉大学,2012.

[2] Kennedy, J.; Eberhart, R. Particle swarm optimization. Neural Networks, 1995 Proceedings., IEEE International Conference on (IEEE). 1995, 4: 1942–1948. ISBN 0-7803-2768-3. doi:10.1109/ICNN.1995.488968.

[3] 周佳莉.浅谈粒子群算法的应用[J].计算机产品与流通,2019(11):139.

[4] 周忠宝,任甜甜,肖和录,金倩颖,吴士健.资产收益序列相依下的多阶段投资博弈模型[J].管理科学学报,2019,22(07):66-88.

[5] 徐德云,徐海俊.对生产帕累托最优条件充分性的质疑与改进[J].统计研究,2008(07):52-57.

[6] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002.

[7] 李苏.投资组合 M-V 模型及其解的相关问题分析[J].固原师专学报,2006(03):60-63.

[8] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), Anchorage, AK, USA, 1998, pp. 69-73.

[9] Zhang Q, Li H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition[J]. IEEE Transactions on evolutionary computation, 2007, 11(6): 712-731.

[10]J. Kennedy, "Bare bones particle swarms," Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706), Indianapolis, IN, USA, 2003, pp. 80-87.