人工智能编程语言实验报告

1 对比法

1.1 变量与数据结构

	C	C++	Python	MATLAB
变量定义方式	需要声明类型，类型不可变	需要声明类型，类型不可变	不需要声明类型，类型可变	不需要声明类型，类型可变
默认输入类型	无	无	str	double
数据结构：数组和列表（序列）结构	数组，大小在定义时确定。动态大小的数组需要通过动态内存分配函数（如 malloc、realloc）来实现。	和 C 语言类似，同时 C++提供了动态数组 vector 类，可以自动管理内存，方便地进行元素的插入、删除和访问。	列表，一种非常灵活的序列结构。可以包含不同类型的元素，长度可以动态变化，还可以使用切片操作获取子列表。	数组是 MATLAB 的核心数据结构，可以是一维、二维或多维的。MATLAB 的数组操作非常丰富，包括矩阵运算、元素级运算等，可以方便地对数组进行 reshape 操作。
数据结构：映射（字典）结构	没有内置的字典类型，通常可以通过结构体数组和哈希函数等方式自己实现类似字典的功能。	有 map 和 unordered_map，map 是基于红黑树实现的有序映射，unordered_map 是基于哈希表实现的无序映射。	字典（dict）是内置的数据结构，它是一种无序的键-值对集合。	可以使用结构体（struct）来模拟字典的功能。
数据结构：集合结构	没有内置的集合类型。	有 set 和 unordered_set。set 是基于红黑树实现的有序集合，unordered_set 是基于哈希表实现的无序集合。	集合（set）是内置的数据结构，无序，不包含重复元素。	可以通过 unique 函数对数组进行处理得到类似集合的结果。
数据结构的存储和内存管理方面	需要手动管理内存。例如使用 malloc 函数分配内存，使用 free 函数释放内存。如果忘记释放内存，会导致内存泄漏。	对于 vector、map 等标准库中的数据结构，内存管理是自动的。但对于自定义的数据结构，如果涉及到动态内存分配，则需要在适当的地方释放内存。	自动管理内存。	自动管理内存。

1.2 运算符、标识符与程序控制语句

	C	C++	Python	MATLAB
赋值	可以合并赋值，但不可交换赋值和按顺序赋值	同C	都可以	都不可以
%	取模	取模	取模	注释(取模 mod)
//	注释	注释	取整除	无效
#	声明	声明	注释	无效
注释	// /**/	同C	# ‘’’	%
求幂	pow	pow	**	^
^	按位异或	按位异或	按位异或	求幂
矩阵乘法			@矩阵乘法 *按位乘(numpy)	*矩阵乘法 .*按位乘
与	&&	同C	and:布尔值与 &:按位与	&&:逻辑与,可短路 &:对应位置与，不短路
或	||	同C	or:布尔值或 &:按位与	||:逻辑或,可短路 |:对应位置或，不短路
非	!	同C	not:布尔值非 ~:按位取反	~
选择结构	if else if else	同C	if elif else	if elseif else
代码块分隔标志	{}	同C	缩进	end
语句结束标志	;	同C	无	;(不添加会输出数据)
遍历	无直接遍历功能	for(auto& x:vec)	for i in range(10) for i in list	for i = 1:10
下标	0开始,按行方向递增	同C	0开始 多维数组用单个索引输出整行	1开始，按列方向递增
全局参数定义	extern	同C	global 函数内声明，只需一次	global 函数内声明，需要所有函数声明

1.3 函数定义与调用

	C	C++	Python	MATLAB
函数定义	返回值类型 函数名(参数列表){函数体}	基本形式和C语言类似。另外C++支持函数重载。	函数定义使用def关键字，形式为： def 函数名(参	函数定义在.m文件中，基本形式是： function [输出

			数列表):函数体	参数列表] = 函数名(输入参数列表)函数体
参数传递	值传递、指针传递	值传递、引用传递	可以传递可变参数/不定长参数	同 Python。同时可传递句柄（处理图形对象时）。
返回值	返回一个值	返回一个值	可返回多个值	可返回多个值
匿名函数	没有真正的匿名函数语法。	Lambda 表达式，基本形式为 [捕获列表](参数列表) -> 返回值类型 {函数体}	使用 lambda 关键字定义，形式为 lambda 参数列表:表达式	使用@符号定义，形式为 handle = @(参数列表)表达式
输出函数	printf	cout<<	print	disp
输入函数	scanf	cin>>	input	input

1.4 文件操作

	C	C++	Python	MATLAB
文件打开和关闭	fopen fclose	ifstream/ofstream close	open close(或上下文管理器)	fopen fclose
文件读写	fprintf/fwrite fscanf/fread	<< >>	write read/readline	fprintf fscanf

2 组合投资问题

2.1 选择成本价格最高的前 10 种股票的编码

2.1.1 Python

我们定义了获得成本价最高的 loc 支股票编码的函数 findtop(loc)，在函数中使用 np.argsort 对股票索引按照成本价排序，并获取成本价最高的 loc 支股票索引，将全零数组的对应索引位置变为 1，以此得到成本价最高的 loc 支股票的编码。运行结果如下：

```
--------------------选择成本价最高的前10种股票的编码--------------------
[0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

2.1.2 MATLAB

同样的思路，只不过使用 sort 函数对股票索引进行排序。运行结果如下：

```
选择成本价最高的前10种股票的编码：
  列 1 至 19

    0     0     0     1     0     0     0     0     0     1     0     0     0     0     1     0     0     0     0

  列 20 至 38

    1     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0

  列 39 至 57

    0     0     0     0     1     0     0     0     0     1     0     0     0     0     0     0     0     0     0

  列 58 至 76

    1     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  列 77 至 95

    0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  列 96

    0
```

2.2 交叉操作

2.2.1 Python

调用前面定义的 findtop 函数获取选择成本价最高的 30 种股票的编码。然后使用同样的方法获取选择收益率最低的 30 种股票的编码。使用 random.randint 随机选择交叉位点，将两个编码进行重组拼接。运行结果如下：

```
------------------crossover test------------------
选择成本价最高的前30种股票的编码：
[1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0
 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1]
选择收益率最低的前30种股票的编码：
[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0
 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0
 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1 1]
交叉后的子代1：
[1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0
 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0
 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1 1]
交叉后的子代2：
[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0
 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1]
```

2.2.2 MATLAB

思路同 Python，只不过使用 randi 函数随机选择交叉位点。运行结果如下：

选择成本价最高的前30种股票的编码：

列 1 至 19

1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0

列 20 至 38

1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 1

列 39 至 57

0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0

列 58 至 76

1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

列 77 至 95

1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0

列 96

1

选择收益率最低的前30种股票的编码：

列 1 至 19

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

列 20 至 38

1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1

列 39 至 57

0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1

列 58 至 76

1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0

列 77 至 95

0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1

列 96

1

交叉后的子代1：

列 1 至 19

1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0

列 20 至 38

1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 1

列 39 至 57

0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0

列 58 至 76

1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

列 77 至 95

1 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1

列 96

1

```
交叉后的子代2：
  列 1 至 19

    0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  列 20 至 38

    1     0     0     0     1     0     0     0     1     0     0     0     0     0     0     1     0     0     1

  列 39 至 57

    0     0     1     0     0     0     1     1     0     1     0     0     0     0     0     0     1     0     1

  列 58 至 76

    1     0     0     1     0     0     0     0     0     0     1     1     1     1     1     0     0     1     0

  列 77 至 95

    0     0     0     1     0     0     0     1     0     0     0     1     0     1     1     1     0     0     0

  列 96

    1
```

2.3 变异操作

2.3.1 Python

定义变异函数。随机选择变异位点，将该位置的二进制数取反（1-dna[pos]）。运行结果如下：

```
-------------------variation test-------------------
变异后的子代1:
[1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0
 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0
 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1 0 1 1]
变异后的子代2:
[0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0
 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1]
```

2.3.2 MATLAB

定义变异函数。给定变异概率，使用 randi 随机选择 5 个变异位点，当 rand 生成的随机数小于变异概率时，将该位编码取反变异。代码与运行结果如下：

```
function mutated_encoding = mutate(encoding, mutation_rate)
mutate_point = randi(length(encoding), [1,5]);  % 随机选择5个变异位点
for i = mutate_point
    if rand < mutation_rate
        encoding(i) = 1 - encoding(i);  % 若随机数小于变异概率，则进行变异
    end
end
mutated_encoding = encoding;
end
```

```
变异后的子代1：
  列 1 至 19

    1     0     0     1     0     0     0     0     0     0     1     0     0     0     1     0     0     0     0

  列 20 至 38

    1     1     0     0     1     0     0     0     1     1     0     0     0     1     0     0     0     0     1

  列 39 至 57

    0     0     1     0     1     0     0     1     0     1     0     0     0     0     0     1     1     0     0

  列 58 至 76

    1     1     1     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0

  列 77 至 95

    1     0     0     1     1     0     1     1     0     0     0     1     1     1     0     0     1     0     1

  列 96

    1

变异后的子代2：
  列 1 至 19

    0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  列 20 至 38

    1     0     0     0     1     0     0     0     1     0     0     0     0     0     0     1     0     0     1

  列 39 至 57

    0     0     1     0     0     0     1     1     0     1     0     0     0     0     0     0     1     0     1

  列 58 至 76

    1     0     0     1     0     0     0     0     0     0     1     1     1     1     1     0     0     1     0

  列 77 至 95

    0     0     0     1     0     0     0     1     0     0     0     1     0     1     1     1     0     0     0

  列 96

    1
```

2.4 进化算法的实现与组合投资问题的优化

2.4.1 基本实现

1、定义适应度函数：

我们根据年初至今涨跌幅定义适应度。当该组合的总花费大于 700 时，适应度为 0，否则适应度为该组合的总涨跌幅。

```python
# 适应度函数（根据年初至今涨跌幅）
def fitness(individual):
    total_cost = np.sum(individual * costs)
    if total_cost > max_investment:  # 成本总价大于700，适应度为0
        return 0
    return np.sum(individual * chg)
```

2、定义初始化种群函数：

随机生成包含 P0 个组合且每个组合选中 15 支股票的种群。并将组合中成本价大

于 700 的股票去掉（置 0）。

```
# 初始化种群函数
def initialize_population(pop_size, num_stocks):  1 usage
    population = np.zeros( shape: (pop_size, num_stocks), dtype=int)  # 初始化全为 0 的种群
    for i in range(pop_size):
        # 随机选择20只股票
        selected_indices = np.random.choice(num_stocks, size=15, replace=False)
        # 确保选中的股票成本价不超过700元
        valid_indices = [j for j in selected_indices if costs[j] <= max_investment]
        population[i, valid_indices] = 1  # 将选中的股票置为1
    return population
```

3、定义交叉函数：

我们选择了两种交叉方式：单点交叉和均匀交叉。单点交叉为当 random()生成的随机数小于交叉概率时，随机选择 1 个交叉位点进行交叉产生子代。均匀交叉为遍历编码的每一个位点，当生成的随机数小于交叉概率时，进行交叉产生子代。希望在实验中看到两种交叉方式带来的结果异同。

```
# 交叉函数（单点交叉）
def crossover_p(parent1, parent2, pc):  1 usage
    if random.random() < pc:
        point = random.randint( a: 0, num_stocks-1)
        child1 = np.concatenate((parent1[:point], parent2[point:]))
        child2 = np.concatenate((parent2[:point], parent1[point:]))
        return child1, child2
    return parent1, parent2
# 交叉函数（均匀交叉）
def crossover_avg(parent1, parent2, pc):
    child1, child2 = parent1, parent2
    for i in range(num_stocks):
        if random.random() < pc:
            child1[i] = parent2[i]
            child2[i] = parent1[i]
    return child1, child2
```

4、定义变异函数：

当生成的随机数小于变异概率时，随机选择 1 个位点进行变异。

```
# 变异函数（基本位变异）
def mutate(individual, pm):  1 usage
    if random.random() < pm:
        point = random.randint( a: 0, num_stocks - 1)
        individual[point] = 1 - individual[point]
    return individual
```

5、定义选择函数：

选出适应度最高的 num_parents 个组合作为亲代产生子代。

```
# 选择函数
def select(population, fitnesses, num_parents):
    selected_indices = np.argsort(fitnesses)[-num_parents:]  # 取出适应度最高的num_parents个个体的索引
    return population[selected_indices]
```

6、定义进化函数：

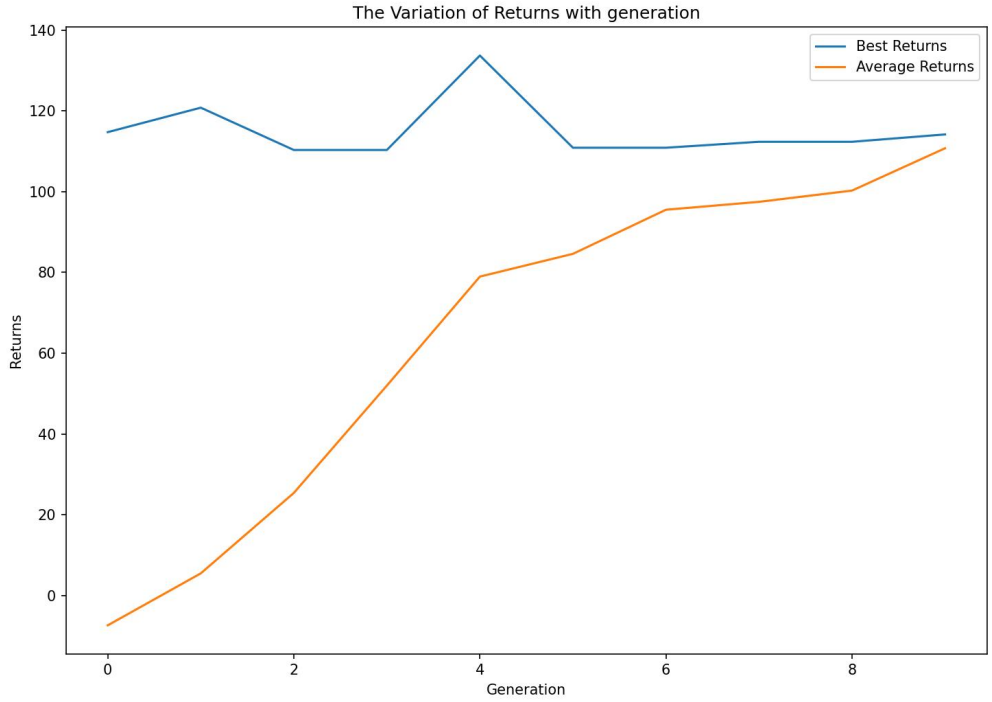
在函数中首先初始化种群。进行 Gen 次迭代：在每次迭代中，调用适应度函数计算每个组合的适应度，并调用选择函数选择适应度最高的 P0 个组合作为亲代。将选中的亲代进行交叉产生子代，使种群规模达到 P，然后对新种群进行小规模变异。最后记录下进化后的最佳收益和平均收益。

```
# 进化算法
def evolutionary_algorithm(Gen, pc, pm, P0, P):  1 usage
    population = initialize_population(P0, num_stocks)
    best_returns = []
    avg_returns = []

    for generation in range(Gen):
        fitnesses = np.array([fitness(ind) for ind in population])
        fitnesses = fitnesses / np.sum(fitnesses)  # 适应度

        parents = select(population, fitnesses, P0)  # 选择亲代
        next_population = list(parents)

        while len(next_population) < P:
            parent1, parent2 = random.choices(parents, k=2)
            child1, child2 = crossover_p(parent1, parent2, pc)  # 交叉产生子代
            next_population.append(child1)
            next_population.append(child2)

        for i in range(len(next_population)):
            next_population[i] = mutate(next_population[i], pm)  # 变异

        returns_arr = np.array([np.sum(ind*returns) for ind in next_population])
        best_returns.append(np.max(returns_arr))
        avg_returns.append(np.mean(returns_arr))

        population = np.array(next_population[:P])

    return best_returns, avg_returns
```

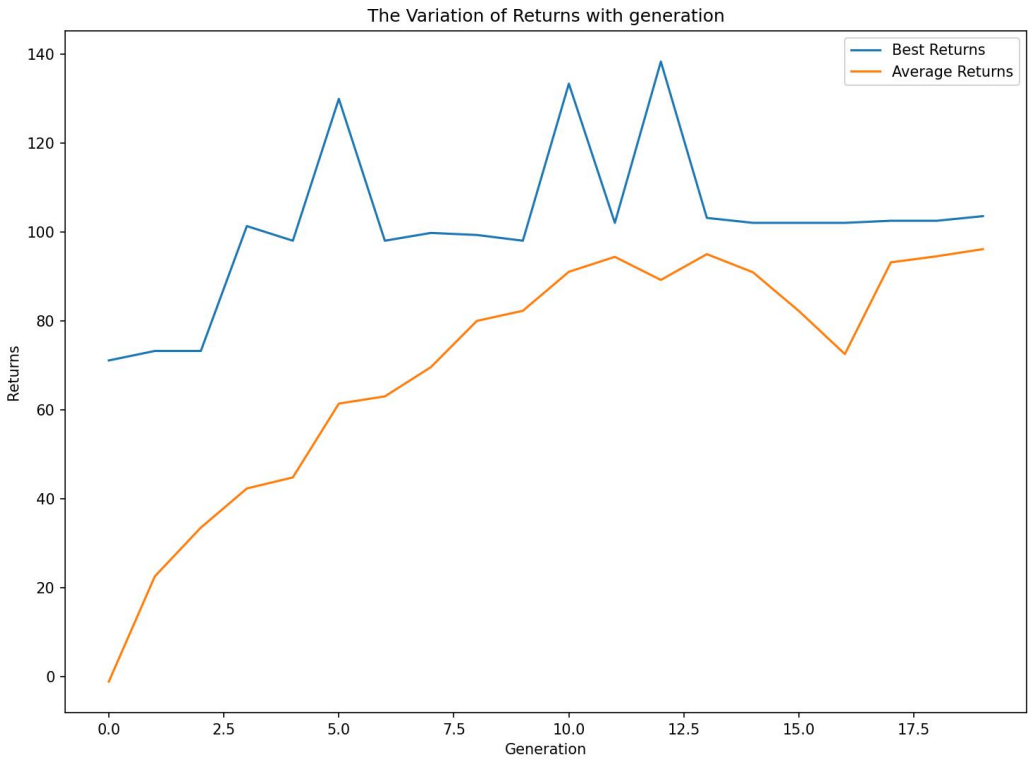
7、设置参数，调用进化函数得到每一代的最佳收益和平均收益。并使用 plot 画出收益随代数的变化曲线。

以下是不同参数下的可视化结果：

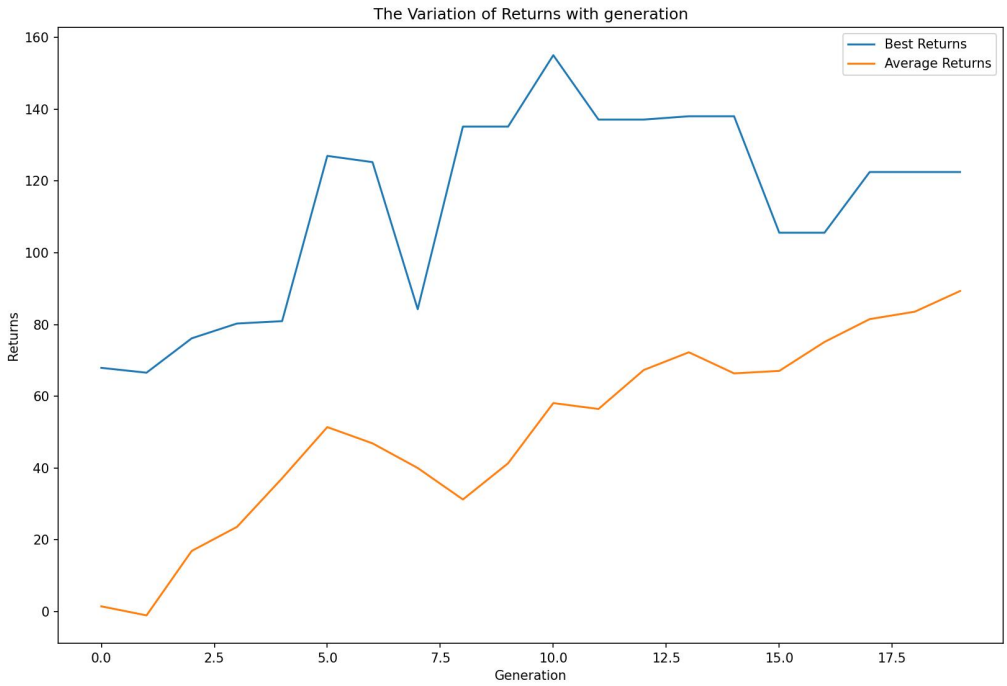
Gen = 10, pc = 0.6, pm = 0.2, P0 = 10, P = 15



Gen = 20, pc = 0.6, pm = 0.2, P0 = 10, P = 15



Gen = 30, pc = 0.6, pm = 0.2, P0 = 20, P = 30



总结：

可以看到，在迭代过程中，平均收益基本稳定上升，但最佳收益波动很大，且最终的最佳收益与初始相差不大（实际上在我们的多次运行结果中，也会出现最佳收益下降的情况）。我们试图改善这一情况。我们初始尝试改变交叉产生子代的方式，将单点交叉变为均匀交叉，但收效甚微。

2.4.2 算法改进

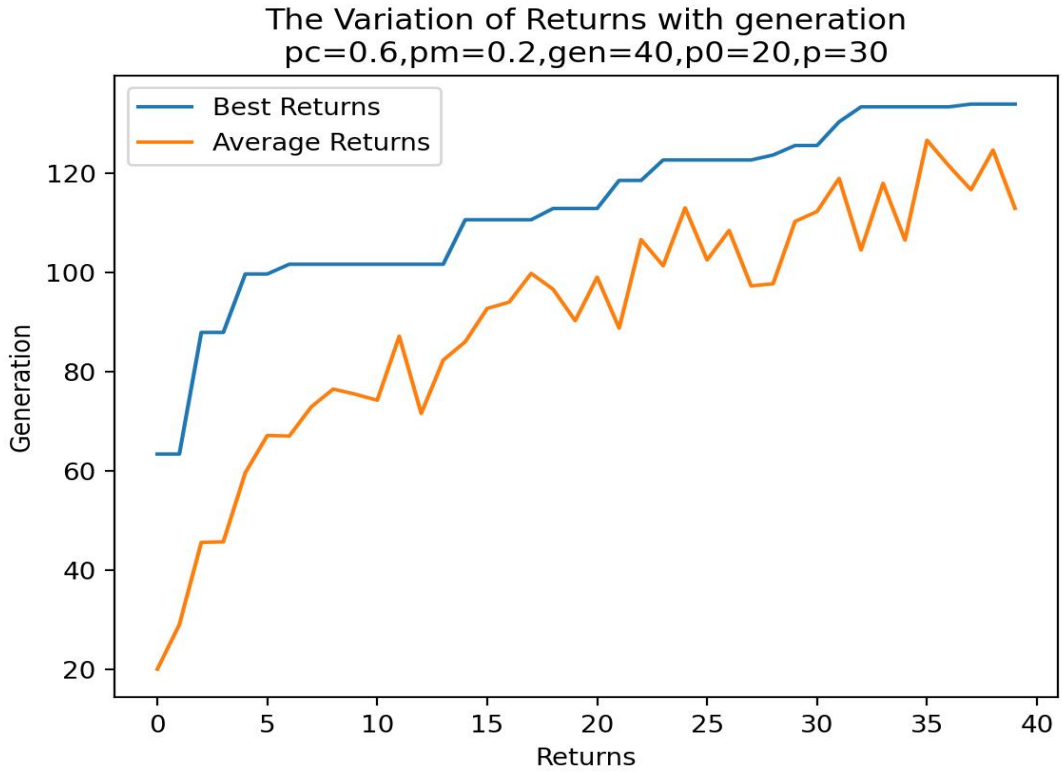
1、我们改用 sigmoid(收益/200)作为适应度。

```
# 适应度函数
def fitfun(dna):  5 usages
    dnaarray = np.array(dna)
    price = originprice.dot(dnaarray)  # 计算总成本价
    if price < 700:
        gain = (aftprice-originprice).dot(dnaarray)/200  # 使用sigmoid(收益/200)作为适应度
        fit = math.exp(gain)/(1+math.exp(gain))
    else:  # 如果总成本超过了限额700就把适应度改成零
        fit = 0
    return fit
```

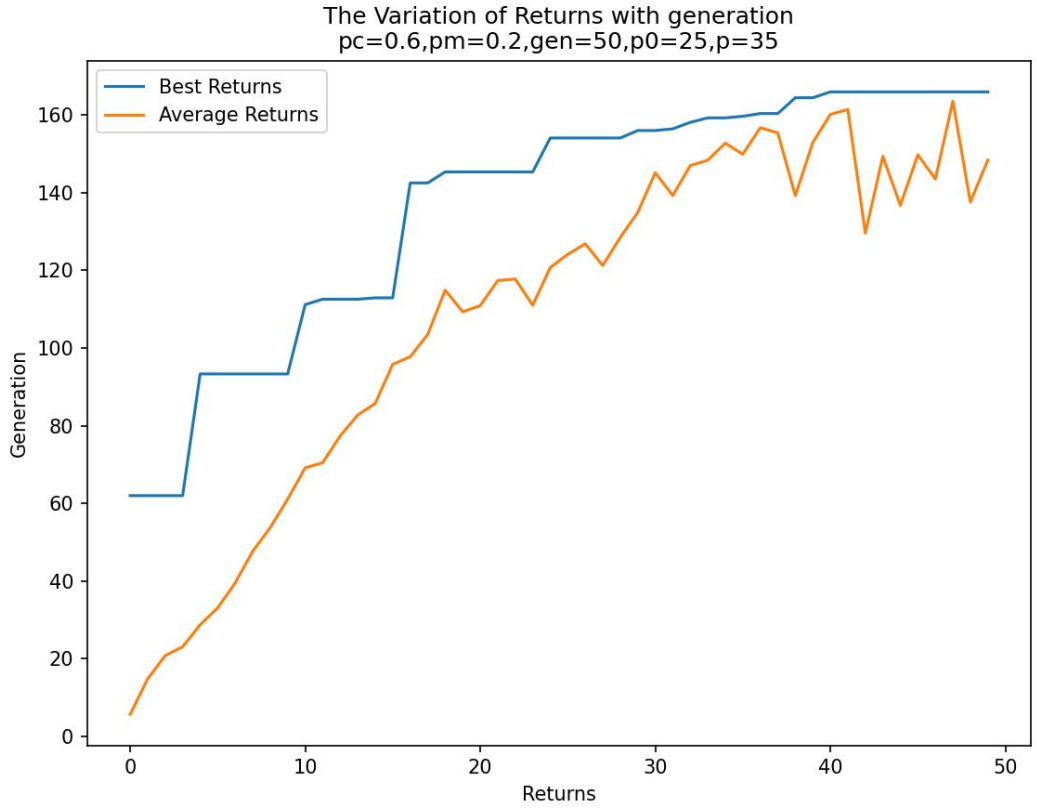
2、在变异过程中，保留每代的最佳组合不进行变异（类似生物学中保留好的基因）。

最终结果如下：

Gen = 40, pc = 0.6, pm = 0.2, P0 = 20, P = 30



Gen = 50, pc = 0.6, pm = 0.2, P0 = 25, P = 35



总结：

1、可以看到改进之后的最佳收益和平均收益都基本呈上升趋势。且最终最佳收益的值也较大（160）。

2、不同的参数设置会产生不同的结果。当迭代次数和种群规模较小时，虽然收敛快，但经常出现最终收益不够高的问题（早熟收敛）。而当迭代次数和种群规模较大时，收益波动较大，但最终收益的值较高，更容易找到全局最优解。交叉率和变异率的选择不宜过大——过大会产生结果波动大、不易收敛的问题。

2.4.3 参数调整

1、针对 pm 的调整（Gen=50, pc=0.6）

基于改进后的算法，对 pm 进行调整。由于随机性较大，对每个 pm 进行 100 次实验（将主程序写成函数，循环 100 次），取最后十代的平均收益，与最后一代的最佳收益，并计算最后十代的平均值和方差。

pm = 1:

```
最后一代最佳收益平均数: 151.37142499378245
最后一代最佳收益总体方差: 90.26036643030324
最后十代平均平均收益平均数: 90.45112858178874
最后十代平均平均收益总体方差: 67.63204066602181
```

pm = 0.5：

```
最后一代最佳收益平均数: 154.24083261361702
最后一代最佳收益总体方差: 121.77650564244276
最后十代平均平均收益平均数: 114.32655294204348
最后十代平均平均收益总体方差: 87.93220205048932
```

pm = 0.2:

```
最后一代最佳收益平均数: 151.18102609199562
最后一代最佳收益总体方差: 183.96581149631103
最后十代平均平均收益平均数: 130.80333232666197
最后十代平均平均收益总体方差: 169.88659958863767
```

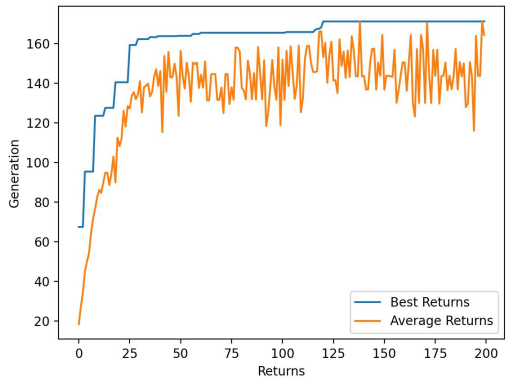
pm = 0.1:

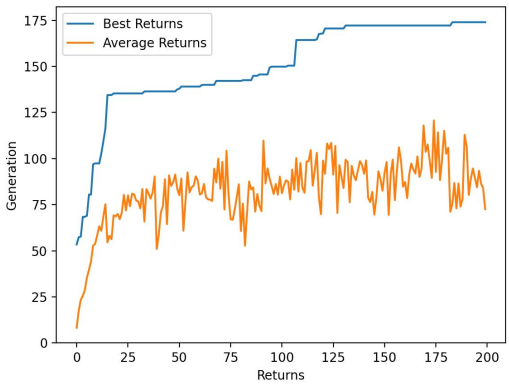
```
最后一代最佳收益平均数: 141.70809224729567
最后一代最佳收益总体方差: 217.89925626355577
最后十代平均平均收益平均数: 129.71906697244498
最后十代平均平均收益总体方差: 185.47965729673274
```

可以看出，在限定 50 代范围内，随着变异概率 pm 减小,最后一代最佳收益和最后十代平均平均收益的总体方差均会增加，也就是不稳定性增加，推测这是因为变异量小，结果更加依赖于样本的初始化结果。

pm 的减小对最后一代的最佳收益，在前期没有观察到显著影响，但在 pm 减小到 0.1 后会导致最后一代的最佳收益减小。

pm 的适度减小可以使最后十代平均收益增大，但有限迭代次数内过度减小 pm 可能会适得其反。并且，减小 pm 会导致平均收益收敛变慢，这或许就是有限代数内不宜过度减小 pm 的原因。





左：pm = 0.2(Gen = 200, pc = 0.6)　右：pm = 1(Gen = 200, pc = 0.6)

2、对 pc 的调整（Gen = 50, pm = 0.2）

pc = 1:

```
最后一代最佳收益平均数: 154.6498371310799
最后一代最佳收益总体方差: 123.86882403026567
最后十代平均平均收益平均数: 133.25351335110037
最后十代平均平均收益总体方差: 103.74613051595718
```

Pc=0.75:

```
最后一代最佳收益平均数: 149.92397892317626
最后一代最佳收益总体方差: 182.64340669552803
最后十代平均平均收益平均数: 129.04044545803524
最后十代平均平均收益总体方差: 153.97190951615204
```

0.5:

```
最后一代最佳收益平均数: 146.73335187707985
最后一代最佳收益总体方差: 191.94170310403774
最后十代平均平均收益平均数: 127.11482622121054
最后十代平均平均收益总体方差: 185.75705324356736
```

0.25:

```
最后一代最佳收益平均数: 140.42881021140866
最后一代最佳收益总体方差: 200.90979554775265
最后十代平均平均收益平均数: 121.81692106615748
最后十代平均平均收益总体方差: 177.71985366731676
```

可以看出，最后一代最佳收益平均数随交叉率降低而减小，最后十代的平均平均收益随交叉率降低而减小，并且这两个参数的稳定性也随交叉几率降低呈减小趋势。因此可以推断，充分的杂交有利于提升遗传算法的性能。

3 心得体会

3.1 小组合作方面

在本次实验中，我们在初始算法策略的选定（包括适应度函数的定义依据、交叉和变异的方式等）方面遇到了疑惑和意见分歧，且在实际算法实现中也遇到了许多困难，例如初始化选择股票太多导致适应度全为 0、收益下降或不收敛等。但经过讨论和不断思索，我们最终找到了较好的解决方案，在合作中学习新的观点，共同成长。

3.2 个人成长方面

在遇到问题时不退缩不气馁，多尝试新的方法，设置不同的参数，深入挖掘数据隐含的规律，本次实验锻炼了我们的编程能力和算法设计与改进能力。

3.3 对算法的理解方面

进化算法与生物学中的“物竞天择，适者生存”十分相似。每次选择最好的“基因”进行交叉变异产生子代。算法具有较大的随机性，应多次实验，选择不同的适应度函数与交叉变异方式，设置不同参数，观察结果，以找到更好的方案。

致谢

感谢金老师和助教师兄一学期来的辛苦付出。老师的认真上课和师兄的耐心答疑让我在本门课程中受益良多，逐渐认识 Python 和 MATLAB 编程语言的便利性和优点，并决定继续学习这两门编程语言。预祝新年快乐！