Flipping game under the perspective of Linear Algebra
寻找变化中的不变量

俞奕成

2022 年 11 月 20 日

目录

第一部分　Introduction

1　开端：2阶翻棋游戏的具体描述

问题的开始是一个简单的二阶翻棋游戏：

初始状况：

在2×2的棋盘上，随机放置着一些棋子，这些棋子一面是黑色，一面是白色，向上面的颜色完全随机

允许对这些棋子进行如下操作：

每次选中一列或者一行棋子，将这一列或者一行棋子全部翻转。

游戏要求的最终结果：

如果有可能，在进行若干次操作之后，棋盘上的棋子变为全部黑色向上或全部白色向上。

游戏的解法：

一系列允许的操作，使得初始情况经过这些操作后变为最终结果

注意：

并不是所有棋盘都有可能达到最终要求，部分初始状况在操作后是不可能达到最终要求的

2　问题的提出

对于这个翻棋游戏，不难提出如下问题:

1. 有没有方法，在给出任意一种棋盘的初始状况后，快速的判断出该棋盘是否可解，即是否有可能经过允许的操作达到要求的结果（向上面全黑或全白）？

2. 有没有方法，对于给出的任意一种棋盘的初始状况，如果有可能达到要求结果，都能找出一种能该游戏的解法？

3　过程和问题的抽象化

棋盘，黑棋子，白棋子显然不利于我们以线性代数的方式来思考问题。所以，我们应该借助线性代数的概念将问题抽象化：

定义棋盘为一个2×2的矩阵：

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

定义：白棋子为1，黑棋子为-1
由此可知，矩阵的初始情况的所有元素都是随机的1和-1
例如：

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

允许的操作也可以对应到矩阵的变换：将某一行（列）的元素全部乘以1或者-1
不难发现这个操作就是矩阵的一种初等行（列）变换
游戏的要求——最终变为全黑或者全白，也可以对应到元素全为-1或者元素全为1的矩阵，即

$$\begin{pmatrix} -1 & -1 \\ -1 & -1 \end{pmatrix} \quad (\text{全黑}) \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad (\text{全白})$$

4 具体问题的扩展

只要将翻棋游戏的定义中2×2棋盘改为$n \times n$棋盘，而其他定义不变，翻棋游戏的规则就可以很容易地扩展到$n \times n$的棋盘上。
两个问题在$n \times n$棋盘的情况下仍然存在并且可以讨论。

5 抽象定义的扩展

只需将2×2的矩阵扩展为$n \times n$的矩阵：

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

游戏要求的最终结果也进行对应扩展：

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & -1 & \dots & -1 \\ -1 & -1 & \dots & -1 \\ \vdots & \vdots & \vdots & \vdots \\ -1 & -1 & \dots & -1 \end{pmatrix}$$

其余定义不变，即可将抽象定义扩展到$n \times n$

第二部分　初始简单情况的规律与分析

先从2×2的情况开始分析，由于这种情况很简单，初始情况只有$2^4 = 16$种，我们不妨把每种情况都枚举一下，并进行分类：

分类为可以转化的：$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ $\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$ $\dots$

和不能转化的：$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ $\begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$

能转化的矩阵自然可以直接给出方法，而不能给出的矩阵需要进行证明，下面给出一种简易的证明：

每一次操作改变的元素都是偶数个，故不会改变矩阵内所有元素的乘积，初始状况所有元素乘积是-1的矩阵无论经过多少次变化，乘积仍然是-1，而不难发现最后全黑或者全白的话，所有元素的乘积是1，故这些矩阵不可能变为全黑或者全白

我们可以发现以上的证明过程已经给出了一种判断2×2矩阵是否可以达到最终结果的特征：即

1. 若所有元素乘积为1，则可以转化

2. 若所有元素乘积为-1，则不能转化

对这些矩阵的特点进行分析,不难发现另外一些可以用于判断的特征:

设初始矩阵为A,发现

1. 能够转化的矩阵均有行列式为0（$det(A) = 0$）

2. 而不能转化的矩阵均有行列式不为0（$det(A) \neq 0$）

由于2×2的情况全部都枚举出来了，所以上面那些检验标准的正确性都可以直接验证。

第三部分 使用线性代数的方式分析该问题

6 很多2阶标准并不适用于高阶情况

接下来我们要考虑$n \times n$的情况了。
首先可以尝试将前面提出的2阶情况下的两种判断标准直接扩展到高阶。
但是可以发现直接使用元素乘积判断显然是不可行的，因为一次操作会改变奇数个元素
而直接使用行列式判断也不可行，因为可以举出反例：
例如三阶行列式 $\begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}$ 并不能化为需要的最终结果，但是它的行列式却是0
可见前面的两个适用于2阶的标准并不适用于高阶的一般情况。

7 引入更多线性代数的概念和思考方式

前面的第二种采用行列式的思考方式虽然不能直接扩展，但尝试将它推广很容易让我们联想到线性代数中的一个概念——矩阵的秩。这样做有两个好处：

1. 我们允许的操作是初等行（列）变换，这个过程中矩阵的秩具有不变性，对分析有利

2. 2阶情况下，对非零矩阵（这边的初始情况当然不可能是零矩阵）而言，行列式等于0与秩为1是等价的，猜想这可能是更好的推广方式。

第四部分 一般化情况的分析与思考

8 判断标准:矩阵的秩为1

8.1 推断，猜想和尝试

前面已经提到了猜想，秩为1是很有可能是一个正确的判断方法。我们可以列举几个高阶情况，发现这个标准都是正确的。由于变换（操作）过程中矩阵的秩不变，我们希望寻找变化中的不变，这样易于解决问题。
因此对于不变的最终结果——全黑或者全白矩阵进行分析，我们发现这两

个矩阵的秩可能等于1。
于是，下面给出简易证明：

8.2 最终结果矩阵秩为1的证明

由定义可知，在表示全黑（全白）棋盘的$n \times n$矩阵$\mathbf{A} = (a_{ij})_{n \times n}$中，
有$\forall i, j, a_{ij} = 1 (or - 1)$
从中任意取出一个元素a_{ij}，即为该矩阵的一阶子式，显然$a_{ij} \neq 0$，因此存在不为零的一阶子式
另一方面，矩阵$\mathbf{A}$中$\forall$二阶子式$\begin{vmatrix} a_{ij} & a_{in} \\ a_{mj} & a_{mn} \end{vmatrix}$，有

$$\begin{vmatrix} a_{ij} & a_{in} \\ a_{mj} & a_{mn} \end{vmatrix} = a_{ij}a_{mn} - a_{in}a_{mj} = 1 - 1 = 0$$

故由矩阵秩的定义，可知全黑或全白矩阵$r(\mathbf{A}) = 1$

8.3 判断标准必要性证明

在已经证明最终结果矩阵秩为1的基础上，容易证明判断标准的必要性：
翻棋操作是一种初等变换，而初等变换不会改变矩阵的秩
$\Rightarrow$任意次允许的操作都不会改变矩阵的秩
$\Rightarrow$如果给定的一个初始情况棋盘对应的$n \times n$矩阵$\mathbf{M}$经过若干次允许的操作能够化为全黑或全白矩阵$\mathbf{A}$，则必有$r(\mathbf{M}) = 1$
即 一个棋盘能化为全黑或全白$\Rightarrow$对应矩阵的秩$r(\mathbf{M}) = 1$

8.4 判断标准充分性证明

上述证明过程尚不足以说明秩为1这个判断标准的充分性。我们先明确充分性问题：
分析：
$\forall$一个$r(\mathbf{M}) = 1$的$n \times n$初始情况矩阵$\mathbf{M}$
证明：它可以经过若干次$-1c_i$和$-1r_i$初等变换，化为全1或全-1矩阵
经过线性代数的学习，我们知道初等变换$-1r_i$等价于左乘初等矩阵$E(i(-1))$，初等变换$-1c_i$等价于右乘初等矩阵$E(i(-1))$
于是，可以进一步从矩阵的角度来描述这个问题：

存在$n \times n$矩阵$\mathbf{P}, \mathbf{Q}, s.t. \mathbf{PMQ} = \mathbf{A}$ 其中

$$\mathbf{P}, \mathbf{Q} = \begin{pmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{pmatrix} \quad a_{ii} = 1\, or\, -1 \quad i = 1, 2, \ldots, n$$

$\mathbf{M}$为棋盘初始情况对应的$n \times n$矩阵
$\mathbf{A}$为全黑或全白的棋盘最终结果对应的$n \times n$矩阵
然而，接下来，我们发现这个问题似乎难以从矩阵角度继续分析或者得出更多的结论了。
其实上面这个命题一共对应了有限的2^{2n}种情况。如果在有计算机辅助的情况下进行枚举，最终可以找出可行情况，将可行情况中的$\mathbf{P}, \mathbf{Q}$还原成对应的行变换和列变换，再还原为允许的操作，就找出解法了。
然而这种的时间复杂度为$\mathbf{O}(2^n)$，这是很高的一个时间复杂度，而且它过于暴力，很不数学，至少很不线性代数。
所以，我们考虑换个角度看问题，找到更多变化中的不变，引入了线性代数中和矩阵十分相关的概念——向量组。并且把初始矩阵$\mathbf{M}$看作一个列向量组。
即$\mathbf{M} = \alpha_1, \alpha_2, \ldots, \alpha_n$
线性代数中，有关向量组和矩阵的秩的关系，有如下定理:

定理 8.4.1 设$\mathbf{A}$是$m \times n$矩阵，
则$\mathbf{A}$的列向量组$\alpha_1, \alpha_2, \ldots, \alpha_n$的秩等于矩阵$\mathbf{A}$的秩，

由此可知，棋盘初始矩阵$\mathbf{M}$的列向量组的秩都为1。即$r(\alpha_1, \alpha_2, \ldots, \alpha_n) = r(\mathbf{M}) = 1$
又由

性质 8.4.2 如果一个向量组的秩为$r(r > 0)$，则向量组中任意r个线性无关的向量都是它的一个极大无关组。

可知，任取一个$\mathbf{M}$的行向量或者列向量，都是$\mathbf{M}$的列向量组的极大线性无关组。
此处我们取一个列向量为第一列的向量α_1
又由于极大线性无关组的定义中有

定义 8.4.3 向量组中的每个向量都可以被它的极大线性无关组线性表示。

我们可知$\mathbf{M}$的列向量组中每个向量都可以用α_1线性表示
即$\alpha_i = k\alpha_1 \quad i = 2, \ldots, n$，

由定义，矩阵中的元素只能为1或-1$\Rightarrow k = 1\, or\, -1$
因此，只需要将所有列向量都乘以k^{-1}，便可以让所有列向量相等，
也就是在矩阵中对这些列作$c(k^{-1})$变换，或者在翻棋游戏中，对所有与第一列不同的列做一次翻转，就可以得到一个所有列的对应位置的元素都相等的矩阵
经过上述操作之后，我们再从行向量组的角度来看矩阵，会发现每个行向量组中的所有元素都相等。
由于元素只能是1或者-1，不难发现这些行或全为1，或全为-1
因此，再将所有元素全为-1的行向量组乘以-1，或者在翻棋游戏中对所有值是-1的行做一次翻转，则得到了所有元素全为1的矩阵也就是定义的全白矩阵，是符合翻起游戏最终结果的矩阵。
类似地，
对于任意一个$r = 1$初始矩阵，都可以进行这样的操作，使得最终得到全黑或全白矩阵，故充分性得证

9 推论：更直观的判断标准

当矩阵的阶数很高的时候，想要快速的求出一个没有规律的矩阵的秩，或者得知它的秩是否是1是相当复杂而耗时的。但是，对充分性论证的过程进行反思，可以发现，我们已经得知一个很重要的结论：

性质 9.0.1 $\forall$一个可以化为全黑或全白的初始棋盘对应的矩阵
$\mathbf{M} = \alpha_1, \alpha_2, \ldots, \alpha_n$，
有$\alpha_i = k\alpha_1 \quad i = 2, \ldots, n \quad k = 1\, or\, -1$

从直观上来看，就是这个矩阵的所有列的对应位置的元素成比例
这是非常容易在直观上感受或判断的。我们也就得到了一种快速直观的判断翻棋游戏是否可解的方法。

10 快速简洁的一般化解答方法

其实充分性论证的过程就是给出了一种标准化解答方法，此处不再重复。

11 进一步推广到$m \times n$矩阵

发现以上关于$n \times n$矩阵的判断标准：

1. 具有充要性的判断标准：秩为1
2. 具有充要行的快速直观判断标准：列向量对应成比例

可以直接推广到$m \times n$矩阵
关于充分性和必要性的证明都可以直接推广，此处不再赘述。
也就是说，我们推得的判断标准能适用于任意行数，列数的矩阵了。

第五部分　该类问题一般解法的实现与分析

前面的证明可能不太直观，我们使用程序给出一种标准的解法，更直观一些：
为了节约时间，我们使用python来解决这个问题，且使用numpy库来存储和处理矩阵。
我们规定输入为：

第一行 目标结果是全白棋盘还是全黑棋盘

第二行 矩阵的尺寸$m \times n$，逗号分隔

之后的行 按行输入矩阵，逗号分隔元素

例如:

```
To reach All black input -1, All white 1:-1
Please input the size of your chessboard in rows,lines:2,2
Please input line 0 split by , :1,1
Please input line 1 split by , :-1,-1
```

输出为：
若有解，输出一个字符串列表，它是翻棋游戏的解例如：

```
['r1']
```

或者

```
['c1', 'c2', 'r2']
```

其中每个元素为ri或ci(i为一个数字)，分别代表对i行或i列做一次翻转
做完列表中的所有操作，就可以得到目标的全黑或全白棋盘
若无解，输出"NOANSWER"
程序大致分为几个部分:

```
c = chessGame(type)
c.input()
c.columnMatch()
c.rowMatch()
c.output()
```

然后以下是对于每个部分的实现：

```
class chessGame():
def __init__(self,type):
    self.solutionStatus = 1
    self.solution = []
    self.type=type

def input(self):
    self.m,self.n = map(int,input("Please input the size of your chessboard in row
    matrixList = []
    for i in range(0,self.m):
        rowList = input("Please input line " + str(i+1) + " split by , :").split("
        rowList = list(map(int,rowList))
        matrixList.append(rowList)
    #print(matrixList)
    self.M = np.matrix(matrixList)

def columnMatch(self):
    for j in range(0,self.n):
        sign = self.M[0,j] / self.M[0,0]
        for p in range(0,self.m):
            if self.M[p,j] / self.M[p,0] == sign:
                continue;
            else:
                self.solutionStatus = 0
                break
        if(self.solutionStatus == 1) and (sign == -1):
            operation = "c" + str(j+1)
            self.solution.append(operation)

def rowMatch(self):
```

```
        if self.solutionStatus:
            for i in range(0,self.m):
                if self.M[i,0] == -self.type:
                    operation = "r" + str(i+1)
                    #print(operation)
                    self.solution.append(operation)

    def output(self):
        if self.solutionStatus:
            print(self.solution)
        else:
            print("NOANSWER")
```

能够按照要求运行的完整版本源代码在solution.py中，已经open source在
`http://59.78.35.247:3000/TonyYYC/Flipping-Game`
校内使用gitea自行搭建的git服务器，请使用sjtu内网或连接校园vpn访问
技术所限，连接不稳定或无法访问的情况，敬请谅解
如有bug，可以在gitea的页面上直接提issue

第六部分　回顾解决翻棋游戏的过程：对线性代数中不变量的思考

12　不变量——矩阵与向量组中的秩

翻棋游戏的解决过程一直围绕着线性代数中一个核心的不变量——秩来进行。矩阵的秩在初等变换中具有不变性，也就是在左乘或右乘初等矩阵（相当于初等变换）中具有不变性。
而且这种秩的不变性并不局限于矩阵之中，它和向量组的秩是相等的，故在对向量组进进行与矩阵的初等变换等效的操作时，向量组的秩是不会变的。
这些操作包括：

1. 将向量组中某个向量乘以非零常数
2. 将一个向量的k倍加到另一个向量上
3. 将两个向量调换位置

而向量组的秩是其极大线性无关组中的向量个数，结合向量组秩的定义，我们可以推出有关向量组的线性相关性和是否能线性表示（两者在其实是等价的）的特性
而这些特性可以为我们解决特定问题提供支持。

13 从特殊到一般，从具象到抽象

从以上的过程中我们可以看到秩的不变性具有很多用处。但是正如这篇文章所做的，在思考的时候，在某有任何铺垫的情况下直接利用秩去解释一个新的问题，其实相当困难。
毕竟一般化问题本身往往就很复杂，而秩这种特性其实很抽象，不好直观理解，如果直接用抽象的工具去解决一般化问题，往往手足无措。
所以，我们可以先从特殊的，简单的问题开始，从具象的问题开始，例如本文先从2×2的矩阵开始，尝试寻找一些直观的性质，再尝试推广到$n\times n$，按需引入抽象化工具。
有了在特殊和具象的情况下的直观理解，后续解决一般化，抽象的问题的过程，会顺利很多。

第七部分 关于其他学科及生活中不变量的思考

正如寻找和利用线性代数的不变量——秩是这篇文章解决翻棋游戏的核心一样，寻找和利用不变量无论在其他学科还是在生活中都是解决问题的关键。
以下列举几个我能想到的例子。由于这是一个线性代数的文章，以下只进行简述，具体内容可以由读者自行展开：

1. 我们的无机化学中充满着不变量的思想，常用溶液中的质子守恒去写质子平衡式，氧化还原中的得失电子守恒去推导反应方程式中各物质的系数比，解决各种复杂的定量计算问题，这其实就是不变量思想，抓住并充分利用了反应过程中的不变量——质子的传递和电子的得失

2. 在分析化学中，我们利用上述无机化学的概念，建立了一套基于“基本反应单元”的计算方法，进一步将上面的不变量抽象化，让计算更快，步骤更少更简便。

3. 这种建立抽象概念来简化，分析复杂问题的办法，与线性代数中建立秩，极大线性无关组，标准型等等将矩阵，向量组的性质抽象出来的

概念，以及建立抽象性更高的线性空间，维数，坐标等等概念，不无相似之处。

4. 程序设计，尤其是在进行较大规模的软件工程设计的时候，会采用一种面向对象的设计方式，在大量的，复杂的不同类型的需求中先分类，再在每个类别中抽象出几个能够复用的逻辑，几个重要的变量，这些东西在复杂需求实现的过程中是不变的，然后我们将每个类别都建立一个类，再在每个类中声明这些抽象的逻辑，变量，把整个框架搭好，然后再去具体实现这些抽象的object。虽然这个过程可能和线性代数的概念关系不大，但它其实也是在寻找变化中的不变，而这些object其实就是抽象程度更高的一种“不变量”吧。这样先从变化中找不变，建立不变的抽象化概念和框架，再具体实现的基本思路其实对于工程，无论是什么类型的工程，都有很好的作用，因为正如我们在思考理论问题时的情况一样，它也能够简化复杂的工程问题，同时天生具有“低耦合”的特性，避免不必要的重复步骤，提高效率。其实上面解决这个翻棋游戏的程序虽然规模不大，但也较为浅显地运用了这样的思想，先抽象出几个步骤，再去具体的实现某个步骤。

5. 更多的例子，此处不再赘述，留下一些空间，供读者自行类比，思考……