Extended Application of Burnside's Lemma to Higher Dimension using Matrix and Linear Transformation

楊記綱

2022 年 9 月 11 日

概 要

```
module Burnside where
```

針對清大課程 Burnside's Lemma 去從線性轉換與矩陣、向量的角度去做延伸。令嘗試將此想法推廣到更高維度、非正多面體與較複雜之正多面體。

目錄

How to Read

1 Brief Introduction to Matrix and Linear Transformation

So what does matrix in linear transformation means? Well, for any $n \times n$ matrix, you could think of each column as representing each axis' unit vector's position after the transformation. So for example:

$$\text{Let } T = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{1}$$

$$\text{means, shifting } \hat{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \hat{j} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \hat{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ to} \tag{2}$$

$$\hat{i}' = \begin{bmatrix} a \\ d \\ g \end{bmatrix}, \hat{j}' = \begin{bmatrix} b \\ e \\ h \end{bmatrix}, \hat{k}' = \begin{bmatrix} c \\ f \\ i \end{bmatrix} \tag{3}$$

Hence by these definitions we could conclude that:

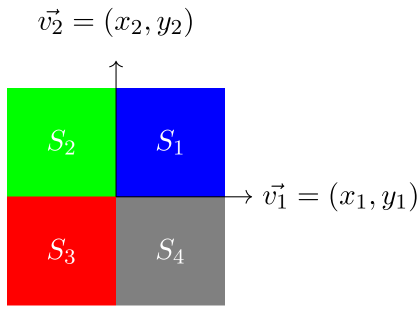
{data}

定理 1 (最少資料量)
For any object within n-Dimensional space, by specifying n-Surfaces' location could unique identify an object's orientation.

2 Describing Object's State

To descibe an object's current orientation, we'll use a set of vectors (which is collected into a matrix). For example, a 2×2 square could be denoted by two 2-D vectors (aka a 2×2 matrix) like following:

$$M = \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \tag{4}$$



in which we could clearly see by giving two 2-D vectors, we could rigidly define our 4 different squadron (in counterclockwise fashion).
Now we can apply transformations onto these vectors. But what could be counted as a valid transformation? Well, those whom preformed a closed transformation.

2

3 Describing Operations

4 Properties

當我們在談到旋轉一物體的時候我們可以將其總結成：

$$\mathbb{T} = \{T \mid T \cdot x \in \mathbb{X}, x \in \mathbb{X}\} \tag{5}$$

其中 $\mathbb{T}$ 代表所有合法的 $n \times n$ 矩陣使得作用於 $\mathbb{X}$ 中任意元件會得結果 $T \cdot x = x' \ni x, x' \in \mathbb{X}$。那也就是換成講義裡的記號 $|G| = \#\mathbb{T}$。

4.1 Calculate $\#\mathbb{T}$

還記得在定理 1 中所說的最少資料量嗎？現在對我們的 n 維物體來標示各種 orientation 的話，舉 $\mathbb{R}^3$ 物體為例，可靠標記其中三個面的方位即可。所以（為了後續計算方便）我們就永遠選相鄰的三個邊，以數對 (A, B, C) 表示[1]。那現在在令所有可能的數對之集合：

$$\mathbb{P} = \{(D, E, F) \mid \angle DOE = \angle AOB, \angle DOF = \angle AOC, \angle EOF = \angle BOC\} \tag{6}$$

其中我們叫這保角性質我們所選之數對 (A, B, C) 的 Structure，而單純允許旋轉而不允許映射的情況下，這是一個該被確保的 Structure。另外須注意這 Structure 是有方向之分的，通常 A, B, C 採逆時鐘排列（從原點向外指，右手方向)。
那現在定義完 $\mathbb{P}$，要計算旋轉方式就簡單多了。首先我們知道 $\forall P \in \mathbb{P}, \exists T \in \mathbb{T} \ni T \cdot (A, B, C) = P$ (其中 $T \cdot (A, B, C) = (TA, TB, TC)$ 所以 $\#\mathbb{P} = \#\mathbb{T} = |G|$。而至於 $\mathbb{P}$ 則可用排列組合推出，對立方體舉例：

$$\text{先任意選一面 } A$$
$$\text{再選一面相鄰 } A \text{ 的面 } B$$
$$\text{最後再選唯一一個在這兩面逆時鐘方向的面 } C$$
$$\text{得 } \#\mathbb{P} = C^6_1 \times C^4_1 \times C^1_1 = 24$$

對正四面體亦同：

$$\text{先任意選一面 } A$$
$$\text{再選一面相鄰 } A \text{ 的面 } B$$
$$\text{最後再選唯一一個在這兩面逆時鐘方向的面 } C$$
$$\text{得 } \#\mathbb{P} = C^4_1 \times C^3_1 \times C^1_1 = 12$$

或者嘗試將它 Generalized 對任意正多面體：

$$|G| = \text{（面數）} \times \text{（一面的邊數、一面相鄰的面數）} \tag{7}$$

4.2 Calculate $\mathbb{T}$

那算完他的數量，我們有沒有從它回推 $\mathbb{T}$ 裡面的內容呢？有的（至少用電腦算式簡單的），我們甚至能算出他的組數（那個 k^n 的 n）一樣拿立方體舉例：

$$\text{令所有面之向量之列表 } L = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{8a}$$

[1]本文中所有表示面之數，皆為向量，且皆由原點指向該面之重心

$$\text{舉例取轉換 } T = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8b}$$

$$\text{得 } TL = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{8c}$$

其中我們可看到，縱列 1∼4 向右位移了一格，而 5,6 不變。因此可得對變換 T（z 軸左手旋轉 90 度）可分成三組，也就是對 L, TL 做 disjoint set 後數他的數量。
就此我們就可以算出他的塗色可能性了。

```haskell
disjointSetNum :: Matrix -> Matrix -> Int
disjointSetNum (Matrix a) (Matrix b) = sum $ map fromEnum (map eqPair (zip a b))
```

5 Extend Research Topics（未寫）

5.1 2x2 魔術方塊

The way I model 2x2 Rubik's Cube is by first giving a position vector p_i and then a facing vector f_i which tells you which direction is the first face facing.

$$P = \begin{bmatrix} p_1 & \dots & p_8 \end{bmatrix}, \text{ each component of } p_i \text{ is either 1 or -1} \tag{9a}$$

$$F = \begin{bmatrix} f_1 & \dots & f_8 \end{bmatrix}, \; f_i \text{ is one of the 6 different (directional) unit vector} \tag{9b}$$

And know with (p_i, f_i) we could denote any block we want, where we can then give a list of 3 colours (order-sensitive) which will be coloured counterclockwise.

```

```

5.1.1 Proof of Completeness

A Matrix

```haskell
instance Show Matrix where
    show (Matrix [x:xs])
                    | null xs = "[" ++ show x ++ "]"
                    | otherwise = "[" ++ show x ++ "]\n" ++ show (Matrix [xs])
    show (Matrix rows)
           | length (head rows) > 1  = "[" ++ (intercalate ", " (map (show.head) rows)) ++ "]\n"
                                        ++ show (Matrix (map tail rows))
           | otherwise = "[" ++ (intercalate ", " (map (show.head) rows)) ++ "]"
```

```haskell
a = Matrix [[1,2],[2,0]]
b = Matrix [[2,4],[2,0]]
```

```haskell
instance Num Matrix where
    (Matrix [xs]) + (Matrix [ys]) = Matrix [zipWith (+) xs ys]
    (Matrix (x:xs)) + (Matrix (y:ys)) = concatM (Matrix [zipWith (+) x y]) ((Matrix xs)+(Matrix ys))
    (Matrix []) + a@(Matrix _) = a
    a@(Matrix _) + (Matrix []) = a
    x * (Matrix ys) = Matrix $ map f ys
                     where f y = map (sum.(zipWith (*)) y) xs'
                           (Matrix xs') = diagFlip x
```

B Sage Graphics

B.1 2x2 Rubik's Cube

```python
def colorRect3D(x, c, l, f): # f: 0 (xy), 1 (xz), 2 (yz)
    x = vector(x)
    if f == 0:
        sv1 = vector((-1, 0, 0))
        sv2 = vector((0, -1, 0))
    elif f == 1:
        sv1 = vector((-1, 0, 0))
        sv2 = vector((0, 0, -1))
    elif f == 2:
        sv1 = vector((0, -1, 0))
        sv2 = vector((0, 0, -1))
```

```
    Gph = Graphics()
    Gph += polygon3d([x, x+sv1, x+sv1+sv2, x+sv2])
    return Gph

def colorBlock(p, f, c):
    Gph = Graphics()
    baseVector = p * 0.5
    for cl in c:
        if f[2]!=0:
            Gph += colorRect3D(baseVector, c, 1, 0)
    return Gph

def plotRubiks2x2(P, F, cs):
    Gph = Graphics()
    for p, f, c in zip(P, F, cs):
        Gph += colorBlock(p, f, c)
    # save3D(Gph) # Comment out if you don't want auto-export
    return Gph

def save3D(g, n="plot"):
    filename = "/tmp/"+n+".html"
    g.save(filename)
    os.system("sed -i 's/\/usr\/share/..\/usr\/share/g' "+filename)
    print("Plot3D saved to: "+filename)
```

LICENSE

Codes

Documentation