MATHEMATICAL REPRESENTATION AND FORMAL PROOFS OF CARD TRICKS

A PREPRINT

Boro Sitnikovski
Faculty of Informatics
UTMS
Skopje, North Macedonia
buritomath@gmail.com

May 15, 2020

ABSTRACT

Card tricks can be entertaining to audiences. Magicians apply them, but an in-depth knowledge of why they work the way they do is necessary, especially when constructing new tricks. Mapping a trick to its corresponding mathematical operations can be helpful in analysis, and the vice-versa process can help create new tricks and make them accessible to magicians.

***K*eywords** Card tricks · Matrix operations · Formal proofs · Python

1 Introduction

We will present a simple card trick - the card sandwith trick - and then analyze it to represent it mathematically in terms of matrices. Further, the computational code will be provided in Python together with a formal proof by cases.

Different variants of the trick are possible, however, we will use a simpler variant with only 9 cards.

A magician starts by picking 9 random cards. Then, another person is asked to pick a card, remember it, and put it back into the deck without the magician knowing which card it is. The magician starts dealing the deck into 3 piles. Then the person is asked which pile their card appeared in. They can answer 1, 2, or 3. Whatever the answer, it is important to keep that pile as a "sandwich" between the other two piles when the card piles are grabbed back together. For example, if they said 1, then the magician would first grab either pile 2 or 3, then pile 1, then either pile 3 or 2. Repeat this process one more time, and the cards will be ordered in a way such that the 5th card will be the card picked by the person.

For example, the available cards are $(1, 2, 3, 4, 5, 6, 7, 8, 9)$. A person picks card 4 without the magician knowing this. The magician lays these cards in 3 piles: $((1, 4, 7), (2, 5, 8), (3, 6, 9))$. The person answers pile 1. The magician grabs the second pile, then the first pile, then the third pile. The next lay on the table will be these 3 piles: $((2, 1, 3), (5, 4, 6), (8, 7, 9))$. The person answers pile 2. The magician grabs the first pile, then the second pile, then the third pile. The fifth card will be 4.

After trying a couple of more cases manually, it will become obvious that matrix operations with rotations and column/row swaps are involved. We will present these operations, together with an example calculation program in Python and a proof of correctness.

The piles can be seen as columns in a matrix. The gist of the trick is getting the (x, y) coordinate by asking two questions about columns' position, rather than asking one question about x (row position) and one about y (column position).

2 Mathematical representation

In this section, we will provide the mathematical representation of the card trick, as well as proof by cases.

2.1 Operations

For simplicity we focus on a 2D 3-by-3 matrix, but these operations can be extended to larger square matrices.

Definition 2.1. Operation `SwapCol1-2`: Let

$$\mathbf{S_{12}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To swap column 1 with 2 in a matrix, we just multiply it by $\mathbf{S_{12}}$ from the right:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \mathbf{S_{12}} = \begin{bmatrix} x_{12} & x_{11} & x_{13} \\ x_{22} & x_{21} & x_{23} \\ x_{32} & x_{31} & x_{33} \end{bmatrix}$$

Example 2.1. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Thus,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 3 \\ 5 & 4 & 6 \\ 8 & 7 & 9 \end{bmatrix}$$

Definition 2.2. Operation `SwapCol2-3`: Let

$$\mathbf{S_{23}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

To swap column 2 with 3 in a matrix, we just multiply it by $\mathbf{S_{23}}$ from the right:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \mathbf{S_{23}} = \begin{bmatrix} x_{12} & x_{13} & x_{11} \\ x_{22} & x_{23} & x_{21} \\ x_{32} & x_{33} & x_{31} \end{bmatrix}$$

Example 2.2. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Thus,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 6 & 5 \\ 7 & 9 & 8 \end{bmatrix}$$

Note: Swapping column 2 with 2 is the identity matrix, i.e. `SwapCol2-2` is defined as $\mathbf{S_{22}} = \mathbf{I}$.

Definition 2.3. Operation `FlipUD`: Let

$$\mathbf{D}' = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

That is, $\mathbf{D}'$ is the anti-diagonal matrix. To get the upside-down representation of a matrix, we just multiply by $\mathbf{D}'$ from the left:

$$\mathbf{D}' \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} = \begin{bmatrix} x_{31} & x_{32} & x_{33} \\ x_{21} & x_{22} & x_{23} \\ x_{11} & x_{12} & x_{13} \end{bmatrix}$$

Example 2.3. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Thus,

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

Definition 2.4. Operation `RightRotate`:

Let $\mathbf{D}'$ be the anti-diagonal matrix. To get the rotated representation of a matrix, we just take its transpose, and multiply by $\mathbf{D}'$ from the right:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}^T \mathbf{D}' = \begin{bmatrix} x_{31} & x_{21} & x_{11} \\ x_{32} & x_{22} & x_{12} \\ x_{33} & x_{23} & x_{13} \end{bmatrix}$$

Example 2.4. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Thus,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$$

2.2 Proof by cases

We will show that if we apply these operations in a specific order for specific inputs, it is possible to extract every element from the matrix. Motivated by the example showed earlier, we will come up with a general algorithm.

Based on an input, the algorithm should first apply either `SwapCol1-2`, `SwapCol2-3`, or $\mathbf{I}$ (`SwapCol2-2`). Next, `FlipUD` should be applied to flip the matrix upside-down, and then finally `RightRotate` to rotate the matrix. These operations should be applied twice, which means there are two inputs. Let:

$$f(\mathbf{A}, \mathbf{S}) = (\mathbf{D}'\mathbf{A}\mathbf{S})^T\mathbf{D}'$$

This function represents only one step, an operation on $\mathbf{A}$ with corresponding swap $\mathbf{S}$. Thus, to capture the whole process, we need to apply f twice, and then return the center element:

$$g(\mathbf{A}, \mathbf{S_1}, \mathbf{S_2}) = f(f(\mathbf{A}, \mathbf{S_1}), \mathbf{S_2})_{1,1}$$

We wish to prove that g provides combinations for extracting all elements in a matrix. Using proof by cases, it is trivial to do the calculations on $\mathbf{A}$ to derive the table 1, as we will show in the next section. This proves that every element in a 3-by-3 matrix can be extracted based on the inputs.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Table 1: Numbers mapping

Operation	Result
$g(\mathbf{A}, \mathbf{S_{12}}, \mathbf{S_{12}})$	1
$g(\mathbf{A}, \mathbf{I}, \mathbf{S_{12}})$	2
$g(\mathbf{A}, \mathbf{S_{23}}, \mathbf{S_{12}})$	3
$g(\mathbf{A}, \mathbf{S_{12}}, \mathbf{I})$	4
$g(\mathbf{A}, \mathbf{I}, \mathbf{I})$	5
$g(\mathbf{A}, \mathbf{S_{23}}, \mathbf{I})$	6
$g(\mathbf{A}, \mathbf{S_{12}}, \mathbf{S_{23}})$	7
$g(\mathbf{A}, \mathbf{I}, \mathbf{S_{23}})$	8
$g(\mathbf{A}, \mathbf{S_{23}}, \mathbf{S_{23}})$	9

3 Implementation and proof in Python

Python is a general-purpose programming language[1]. We will be using the Numpy[2] library which provides functions for scientific computations.

```
import numpy as np
```

Further, Numpy does not provide out of the box functions for swapping columns in a matrix, so we will define one. We will also provide the definitions for the different columns swapping combinations.

```
[I, S12, S23] = [[1,1], [0,1], [1,2]]

def swap_cols(arr, frm, to):
  arr[:,[frm,to]] = arr[:,[to,frm]]
```

The `FlipUD` operation is defined in Numpy as simply `np.flipud(A)`. Similarly, the `RightRotate` operation is `np.rot90(A, 3)`. Given this, we can implement `f` and `g` as follows:

```
def f(A, S):
  swap_cols(A, S[0], S[1])
  A = np.flipud(A)
  A = np.rot90(A, 3)
  return A

def g(A, S1, S2):
  A = np.array(A)
  return f(f(A, S1), S2)[1][1]
```

For the formal proof these functions can be used with various inputs, and we can use assertions that these combinations produce the numbers $\{1, 2, \ldots, 9\}$.

```
cards = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]]

assert(1 == g(cards, S12, S12))
assert(2 == g(cards, I, S12))
assert(3 == g(cards, S23, S12))
assert(4 == g(cards, S12, I))
assert(5 == g(cards, I, I))
assert(6 == g(cards, S23, I))
assert(7 == g(cards, S12, S23))
assert(8 == g(cards, I, S23))
assert(9 == g(cards, S23, S23))
```

In which all asserted cases will pass.

4 Conclusion

We showed how to get an in-depth understanding of a ”magical” trick, using mathematics and methods for formal verification. This research can be useful since analyzed tricks can be further researched to create new tricks. For example, if we show that the defined operations apply to larger square matrices we can create new variants of the same trick, or adjust the variants and create a completely new trick.

References

[1] Guido van Rossum Python tutorial. *Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam*, 1995.

[2] Travis E. Oliphant. A guide to NumPy. *USA: Trelgol Publishing*, 2006.

[3] Daniel J Velleman. How to Prove It: A Structured Approach. *Cambridge University Press*, 2006.

[4] Gilbert Strang. Linear Algebra and its Applications. *Harcourt Brace Jovanovich*, 1980.