BC Cancer Agency
CARE + RESEARCH
An agency of the Provincial Health Services Authority

Cancer Surveillance & Outcomes

Solve the Rubik's Cube using Proc IML

Jeremy Hamm
Cancer Surveillance & Outcomes (CSO)
Population Oncology
BC Cancer Agency

Overview

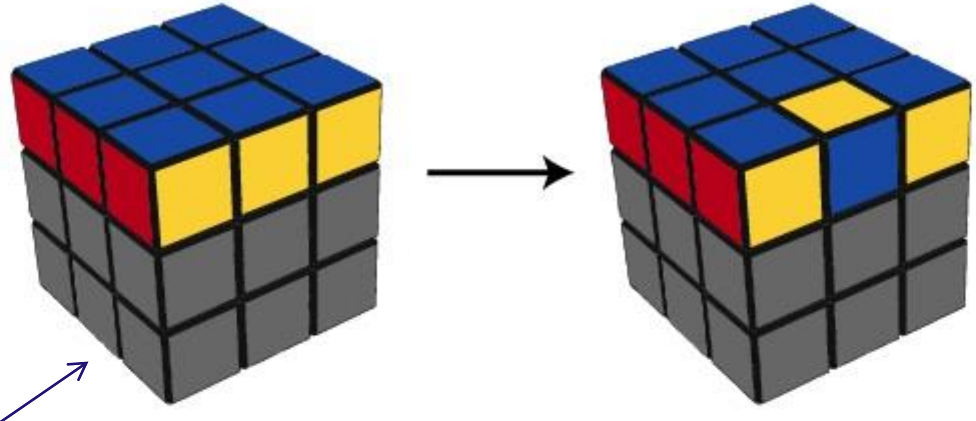
- Rubik’s Cube basics
- Translating the cube into linear algebra
- Steps to solving the cube using proc IML
- Proc IML examples

The Rubik's Cube

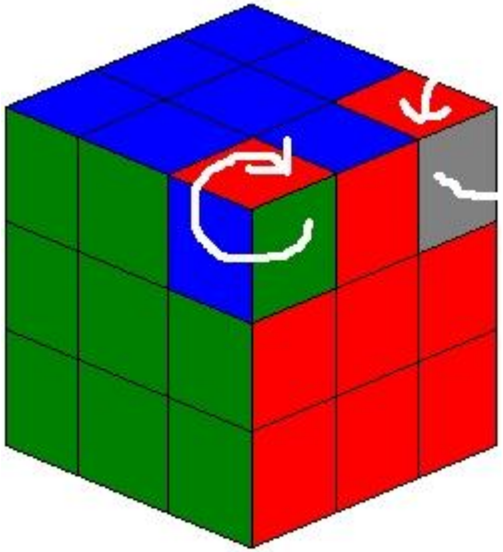
- 3x3x3 cube invented in 1974 by Hungarian Erno Rubik
  - Most popular in the 80's
  - Still has popularity with speed-cubers
- 43×10^{18} permutations of the Rubik's Cube
  - Quintillion
- Interested in discovering moves that lead to permutations of interest
  - or generalized permutations
    - Generalized permutations can help solve the puzzle

The Rubik's Cube

- Made up of
  Edges and corners
- Pieces can permute
  - Squares called facets

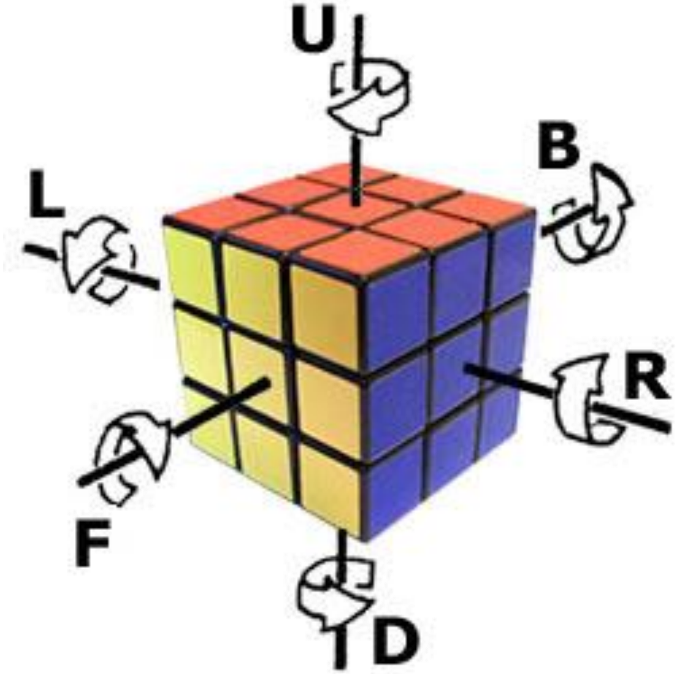


- Edges can flip
- Corners can rotate

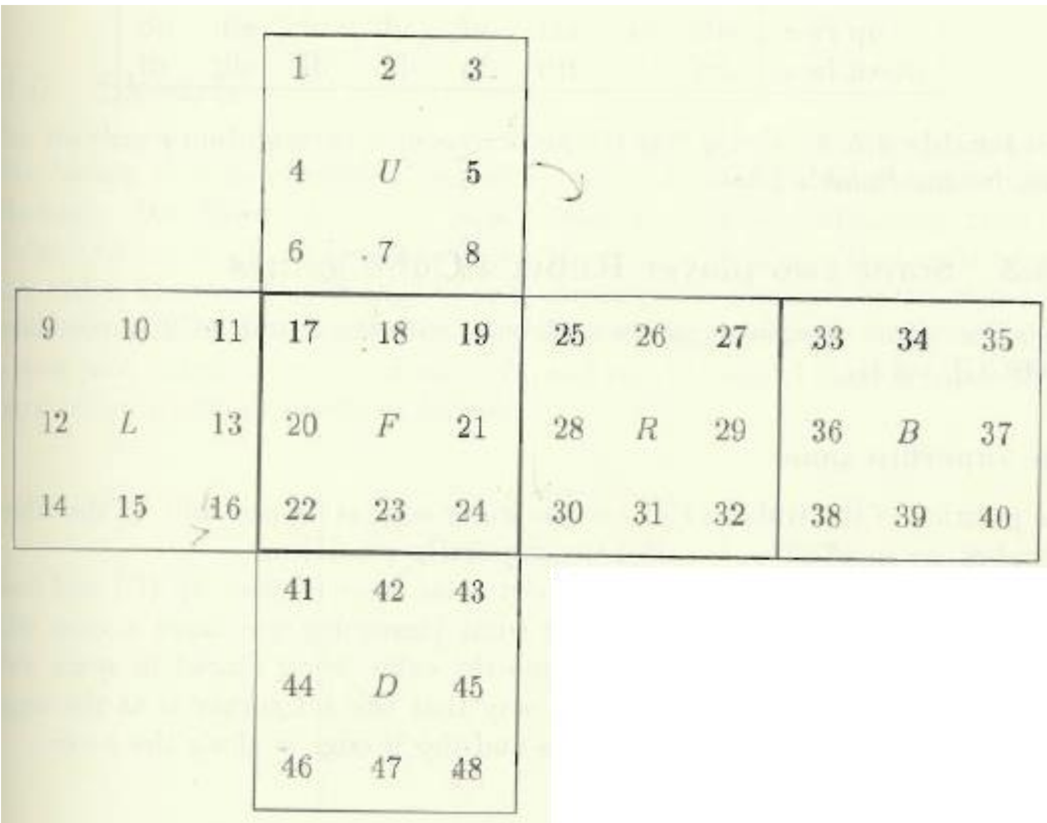


Rubik's Cube Basics

- Each move can be defined as a combination of Basic Movement Generators
  - each face rotated a quarter turn clockwise
- Eg.
  - Movement of front face ¼ turn move F
  - ¼ turn counter clockwise is F^{-1}
  - 2 quarter turns would be F*F or F^2



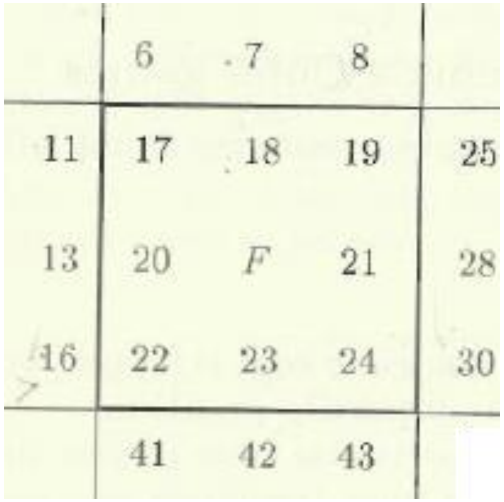
Relation to Linear Algebra



The Rubik's cube can represented by a Vector, numbering each facet 1-48 (excl centres). Permutations occur through matrix algebra where the basic movements are represented by Matrices Ax=b

Relation to Linear Algebra

- Certain facets are connected and will always move together
- Facets will always move in a predictable fashion
  - Can be written as: F= (17,19,24,22)
    (18,21,23,20)
    (6,25,43,16)
    (7,28,42,13)
    (8,30,41,11)



Relation to Linear Algebra

- Therefore, if you can keep track of which numbers are edges and which are corners, you can use a program such as SAS to mathematically determine moves which are useful in solving the puzzle
  - Moves that only permute or flip a few pieces at a time such that it is easy to predict what will happen

Useful Group Theory

- Notes:
  - All moves of the Rubik's cube are cyclical where the order is the number of moves needed to return to the original
    - Eg. Movement F (front face ¼ turn)
    - If done enough times, will return to original position
    - Enough times=4; F is Order 4

Proc IML

- Proc IML (interactive matrix language) can be used to test Rubik's Cube moves using Matrix algebra to determine which moves are useful for solving the puzzle

Intro to Proc IML

- Similar to proc SQL in use

  Proc iml;

  IML code …;

  Quit;

  - code will be able to run while in IML until you exit with a ‘quit;’ statement

  – Useful for row and column calculations/summaries

    - Good at do loops, simulations and linear algebra
    - Not as awesome with character data
    - As always, need to keep track of matrix/vector dimensions

Steps to Solve Cube

- Read in and Create list of moves to test
- Determine Order of each move
  - How many moves in cycle
- Determine during cycle, if at any point:
  - The edges are stable but corners move
    - When and how many?
  - The corners stay stable but edges move
    - When and how many?

Solving in Proc IML

- Read data into proc IML
- Create functions in IML
- Operate on individual matrix cells
- Perform matrix operations
- Output data from IML

Importing Data

- ‘Use’ statement makes a SAS dataset available in proc iml
  - Can specify which variables you wish to import and any ‘where’ statements for filtering
- ‘Read’ statement turns this dataset into a usable matrix
  - Default only includes numeric variables
  - Rows and columns now numbered instead of named as default
    - Can read in names and refer to them

Example

Read in all rows

- Can specify specific row (point 5 = 5th row)

Specify variables to read

```
data test;
    input x y z;
datalines;
1 2 3
4 5 6
7 8 9
;
run;

proc iml;
use test; read all var {x y} into test2 [colname=names];
a=test2[,"x"];
    print a test2;

create test3 from test2 [colname={'w' 'x'}]; append from test2 ;
quit;
```

The SAS System 13:23

a	test2 x	test2 y
1	1	2
4	4	5
7	7	8

Example

```
PROC IML;
RESET DEFLIB=RC;
    use F; read all INTO F; use R; read all INTO R;
    use B; read all INTO B; use L; read all INTO L;
    use D; read all INTO D; use U; read all INTO U;
    use Mu; read all INTO MU; use Mf; read all INTO Mf;
    use Mr; read all INTO MR;
```

- Read in pre-created movement generators in matrix form
- Setup default libname
  - All input and output data will come/go to this library
- Specify rows and columns to import
  - We're using all of them

Functions/macros in IML

- Functions can be created in proc IML
  - Similar to macros
    - Use ‘start’ and ‘end’ statements instead of %macro and %mend
    - Eg. start(variable(s))

      function

      end
  - Function is applied with a
    - Run <function name>(variable(s)) command

Example

```
START FILL(A);
    DO I=1 TO 48;
        IF (A[I ,+]=0 & A[+, I]=0) THEN A[I, I]=1;
    END;
FINISH FILL;
RUN FILL(F); RUN FILL(R);
RUN FILL(B); RUN FILL(L);
RUN FILL(D); RUN FILL(U);
RUN FILL(Mr);RUN FILL(Mu);
RUN FILL(Mf);
```

- This function ‘Fill’ sets the movement generators diagonal values to 1 if there are no values in a row/column combination

Creating and operating on vectors and matrices

- Vectors can be created with () and {} brackets
  - () for continuous style values
    - ST=(1:48)
      - 1 2 3 4 … 48
      - Starting position vector for each face of the cube
  - {} for discrete style
    - POS={2 3 2 3 3 2 …}
      - Position vector for cube faces
        - 2's represent corners; 3's represent edges

Creating and operating on vectors and matrices

- Matrices can be created discretely or with functions
  - A={1 2 3, 4 5 6} 2x3 matrix
- Functions include
  - Identity matrix: I(3) = 3x3 identity matrix
  - All one value: j(4, 3,0) = 4x3 matrix of 0's
    - Useful to create a matrix to fill in with list of permuted faces in cube for each movement in cycle

Matrix Operations

- Matrices can be operated on
  - A*B=Matrix A times Matrix B
    - Eg. F*R creates a single move from 2 movement generators
  - A**n = matrix A to the power of n
    - Eg. F**3
  - A//B = stack A and B (must have same #cols)
    - Stack moves on top of each other to create list of moves as matrices
  - A||B = A beside B (must have same #rows)

Testing a Move

- To determine the order of a move:
  - Isolate Movement matrix G from list as a 48x48 matrix
    - Let d be the number of moves being examined
      - Do i=1 to d by 48 will isolate moves 1 at a time
  - Multiply G by ST vector (1:48) to get permutation (A=G*ST)
    - Re-attach ST to A to identify starting position
      - A=ST||A
- Do while (sum(A=ST)<48) will continue to cycle until every element of A=every element of the starting position vector ST
  - Run a count variable to enumerate the number of moves in the cycle
    - The order

Testing a Move

```
do i=1 to d by 48; *go by movement generating matriix - as they are stacked on each other;
    x=i+47;
    *print G;
    G=moves[i:x,]; *identify a movement generating matrix (eg moves[49:96,]=F);
    ST1=G*ST; *multiply movement generating matrix by position vector;
    A=st||st1; *horizontal concatenation of original position vector and new position vector;
    count=1;
    *print g;
        do while(sum(ST1=ST)<48); *examine the length of the move's cycle with variable count.
        sum(ST1=ST)=48 upon cycle completion;
                ST1=G*ST1;
                A=A||ST1;
            *print count ;
            count=count+1;
        end;
```

Example

- $G=F*R^3$

A=

	COL1	COL2	COL3
1	1	1	1
2	2	2	2
3	3	19	48
4	4	4	4
5	5	21	23
6	6	30	41
7	7	31	29
8	8	32	27
9	9	9	9
10	10	10	10
11	11	24	22
12	12	12	12
13	13	7	31
14	14	14	14

…

COL61	COL62	COL63	COL64
1	1	1	1
2	2	2	2
27	25	38	3
4	4	4	4
18	45	36	5
17	43	16	6
28	42	13	7
19	48	33	8
9	9	9	9
10	10	10	10
6	30	41	11
12	12	12	12
26	28	42	13
14	14	14	14

- Order=63

Summarizing a Move

- Matrix is created for each move which has a 1 or 0 indicating whether a facet has been permuted (compared to starting location)
- Can isolate corners and edges into vectors

```
PERM2=PERM||POS;
last=ncol(perm2);

i1=loc(perm2[,last]=2);*vector of corner positions (i.e. [1,3,6,8,9,11,...]);
i2=loc(perm2[,last]=3);*vector of edge positions (i.e. [2,4,5,6,...]);
a1=2:(ncol(perm2)-1); *2:the end of perm2;
ca=PERM2[i1,a1][+,];
ea=perm2[i2,a1][+,];
ta=ca//ea;/*matrix [2 x number of movements in cycle]
    of number of corners/edges permuted from original-
```

Can specify columns and rows using vectors.
Can summarize columns and/or rows

Example

	COL1	COL2	COL3	COL4
1	1	0	0	0
2	2	0	0	0
3	3	1	1	1
4	4	0	0	0
5	5	1	1	1
6	6	1	1	1
7	7	1	1	1
8	8	1	1	1
9	9	0	0	0
10	10	0	0	0
11	11	1	1	1
12	12	0	0	0
13	13	1	1	1
14	14	0	0	0

…

COL62	COL63	COL64	COL65
0	0	0	2
0	0	0	3
1	1	0	2
0	0	0	3
1	1	0	3
1	1	0	2
1	1	0	3
1	1	0	2
0	0	0	2
0	0	0	3
1	1	0	2
0	0	0	3
1	1	0	3
0	0	0	2

Position vector

Corners permuted

Edges permuted

	COL1	COL2	COL3	COL4
1	18	18	18	18
2	14	14	14	14

Summarizing a Move

- Create 7 column vector that identifies:
  - Do either corners or edges stay stable in cycle (1/0)
  - If edges stable (1/0):
    - What move in cycle does this occur?
    - How many corners move?
  - If corners stable (1/0):
    - What move in cycle does this occur?
    - How many edges move?

Summarizing a Move

Corners permuted →
Edges permuted →

	COL6	COL7	COL8	COL9
1	18	18	18	0
2	14	0	14	14

- Stable edges at move number 7
- Stable corners at move number 9

Summary Vector

	COL1	COL2	COL3	COL4	COL5	COL6	COL7
1	1	1	7	18	1	9	14

Exporting results

- For each move and 7 column vector generating describing the move:
  - Stack vectors to create an Nx7 matrix corresponding to all moves tested
- Can output as SAS dataset for further anaylsis:

```
create study_jh_20141106 from study1;
    append from study1;
```

Questions?

Jeremy Hamm
Cancer Surveillance & Outcomes (CSO)
Population Oncology
BC Cancer Agency