**James Ash**

**Homework 1**

In [37]:
```
# setup packages
import numpy as np
from numpy.linalg import norm, det, inv, solve
from numpy import arccos, dot, transpose
import sympy
```

## Question 1

**Problem:** Consider the game we described in class that involved clicking on points on a grid. There are three colors (red, green, blue), and clicking on a square in the grid cycles the colors of that square and every other square in its row and column. Suppose the game was played on a 3 × 3 grid instead of a 2 × 2. Can you start with an all red configuration and end with a configuration where the first column is blue, the second column is red, and the third column is green?

*Answer:* Let B be a 3X3 matrix with the elements in each cell being one of the colors red (0), green (1), or blue (2) and each cell indexed as indexed as $a_1 - a_9$. For question 1, B is a matrix of containing only 0 because all elements start as red.

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

and $B$'s index...

$$\text{Index of} B = \begin{pmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{pmatrix}$$

We then construct a matrix A from the 9 column vectors, $a_1 - a_9$, each representing what would happen if we "pressed" cells $b_1 - b_9$ once. An example of the first column of matrix A is given below...

$$a_1 = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

In [3]:
```
A = sympy.Matrix([[1,1,0,1,0,0,0,0,0],
                  [1,1,1,0,1,0,0,0,0],
                  [0,1,1,0,0,1,0,0,0],
                  [1,0,0,1,1,0,1,0,0],
                  [0,1,0,1,1,1,0,1,0],
                  [0,0,1,0,1,1,0,0,1],
                  [0,0,0,1,0,0,1,1,0],
                  [0,0,0,0,1,0,1,1,1],
                  [0,0,0,0,0,1,0,1,1]])
A
```

Out[3]:

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Vector **b**, of $Ax = b$, represents the desired end state of $B$, where the first column is blue (2), the second column is red (0), and the third column is green (1). The vector **x** represents the number of times to "press" each element of B. We construct **b** as follows...

In [4]:
```
b = sympy.Matrix([[2],
                  [0],
                  [1],
                  [2],
                  [0],
                  [1],
                  [2],
                  [0],
                  [1]])
b
```

Out[4]:

$$\begin{bmatrix} 2 \\ 0 \\ 1 \\ 2 \\ 0 \\ 1 \\ 2 \\ 0 \\ 1 \end{bmatrix}$$

We will find **b** using the inverse of $A$. I'm using the sympy packages inv_mod(x) function to do so.

First I'm checking that the code works by performing $AA^{-1} = I$.

In [9]:

```
# just a check
I = A * A.inv_mod(3)
#for mod 3. In matlab the % symbol begins a comment, but in python it puts ari
I % 3
```

Out[9]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

**Solving the Ax = b equation using the inverse of A**

In [14]:

```
x = (A.inv_mod(3) * b) % 3
x
```

Out[14]:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

**Cheking the solution by recaluclating b using** $Ax = b$

```
In [16]: b = A*x %3
         b
```

Out[16]:
$$\begin{bmatrix}2\\0\\1\\2\\0\\1\\2\\0\\1\end{bmatrix}$$

Looks good to me.

**Answer.** By pressing $b_4$ twice and $b_6$ once of matrix $B$, we go from an an all red configuration and end with a configuration where the first column is blue, the second column is red, and the third column is green.

## Question 2

**Problem:** A game is played on a $7 \times 7$ board. The board wraps around i.e. if a piece is situated on the rightmost column and moves one to the right, it ends up on the leftmost column. A piece on the top row that moves one up ends up on the bottom row. A piece is placed in the (1, 1) square. The pieces uses the knight move set from Chess (a knight can move in L shapes - two to the right and one down, two to to the right and one up, two up and one left, etc). Can the piece reach the (2, 2) square? If so, describe the moves it needs to do so. Hint: this is a linear algebra problem that requires solving a system of equations over the field $F_7$.

```
In [27]: A = sympy.Matrix([[2, 1, -1, -2],
                           [1, 2,  2,  1]])
         A
```

Out[27]:
$$\begin{bmatrix}2 & 1 & -1 & -2\\1 & 2 & 2 & 1\end{bmatrix}$$

I create vector **b** of $Ax =$ **b** by noting how many spaces are needed to move in the right and left direction traveling from space (1, 1) to space (2, 2). That is one poistive move in both the up/down direction and the righ/left direction, so **b** $= (1, 1)$. Left and down are negative, right and up are positive.

```
In [28]: b = sympy.Matrix([1, 1])
         b
```

Out[28]:
$$\begin{bmatrix}1\\1\end{bmatrix}$$

$$A = \begin{pmatrix} 2 & 1 & -1 & -2 \\ 1 & 2 & 2 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 0 & 4 & 1 & 3 \end{pmatrix}$$

$$A = \begin{pmatrix} 2 & 0 & 3 & 6 \\ 0 & 4 & 1 & 3 \end{pmatrix}$$

Doing the same to **b**...

$$b = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

Then using back substitution we find $m_1 = 5$ and $m_2 = 5$, without the use of the other move types (setting them to 0). So move (1, 2) five times and move (2, 1) five times to land on coordinate (2,2) form coordinate (1,1). I checked this by walking throught he move and it works. Similarly here's a quick calculation...

```
In [29]: (np.array([[1], [1]]) + 5*np.array([[2], [1]]) + 5 * np.array([[1], [2]])) % 7

Out[29]: array([[2],
                [2]])
```

**Answer.** Anyway the answer is yes, the knight can move from space (1,1) to space (2,2) with a number of different solutions. It can be done with the two moves, (-1, 2) and (2,-1) or it can be done with the ten moves described above in the solution.

## Question 3

**Problem:** A circle passes through the points (2, 6), (−1, 7), (−4, −2). Find an equation for the circle (Hint: every circle can be written in the form $x^2 + y^2 + ax + by + c = 0$. Your variables should be a, b, c.)

This is similar to how I would set up the equation to solve for a linear regression only now the equation is $x^2 + y^2 + ax + by + c = 0$ instead of $y = mx + b$. To set up $Ax = b$, I sum $x^2$ and $y^2$, then carry them over to **b**, so that **x** = $(a, b, c)$.

Setting up matrix $A$ from points $(x_1, y_1)$, $(x_2, y_2)$, $(x_3, y_3)$...

$$A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 6 & 1 \\ -1 & 7 & 1 \\ -4 & -2 & 1 \end{pmatrix}$$

now settin uping vector **b** from the sum of $x^2$ and $y^2$...

$$b = \begin{pmatrix} -(x_1^2 + y_1^2) \\ -(x_2^2 + y_2^2) \\ -(x_3^2 + y_3^2) \end{pmatrix}$$

vector **x** is a vector of the cooeficients to solve for, $a, b$ and $c$.

$$x = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Putting it into code to solve...

```
In [25]: A = np.array([[ 2,  6, 1],
                       [-1,  7, 1],
                       [-4, -2, 1]])

         b = np.array([[-40],
                       [-43],
                       [-20]])

         x = np.array(["a", "b", "c"])
```

$A$ is now invertable and I can use $A^{-1}$ to solve $Ax = b$. I'm transitioning from R to python (I used to use matlab), so I do it a few different ways that are close to the same just to make sure the code is right. Now I'm using the numpy package, which is suposed to be similar to matlab. For the previous questions I was using the sympy package.

```
In [26]: # using multiplication of inverse A
         x = np.matmul(inv(A), b)
         x

Out[26]: array([[  0.13333333],
                [ -2.6       ],
                [-24.66666667]])
```

```
In [27]: # using the solve() function
         x = solve(A, b)
         x

Out[27]: array([[  0.13333333],
                [ -2.6       ],
                [-24.66666667]])
```

Checking the solution to see if $Ax = b$.

```
In [28]: # just multiplying A by x to get b back
         np.matmul(A , x)
```

```
Out[28]: array([[-40.],
                [-43.],
                [-20.]])
```

**Answer.** Looks good. There is only one solution to this problem so, even the numbers are kinda gross looking I get $a = 0.13, b = -2.6$, and $c = -24$ from the vector **x**.

$$x = \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0.13 \\ -2.6 \\ -24.6 \end{pmatrix}$$

## Question 4

A marketplace trades four different commodities: abacuses, bananas, crys- tals, and drums. You can trade 3 abacuses, 8 bananas, and four crystals for five drums. You can trade one crystal for three abacuses and four drums. You can trade one abacus and one drum for two bananas and a crystal. You can trade one abacus for two bananas and three crystals. A person comes to the market with one abacus. They left with five drums. What were the trades that they made?

I set up matrix $A$ so that each column represents a trade, denoted by the gain and lost of abacus (a), bananas (b), crystals (c), and drums (d) such that a trade vector **t** and trade coeficient $t_i$, looks like this...

$$trade = t_i \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

and the four trades, starting with one abacus and leaving with five drums looks like this...

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + t_1 \begin{pmatrix} -3 \\ -8 \\ -4 \\ 5 \end{pmatrix} + t_2 \begin{pmatrix} 3 \\ 0 \\ -1 \\ 4 \end{pmatrix} + t_3 \begin{pmatrix} -1 \\ 2 \\ 1 \\ -1 \end{pmatrix} + t_4 \begin{pmatrix} -1 \\ 2 \\ 3 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \end{pmatrix}$$

moving the starting abacus to the other side we get...

$$t_1 \begin{pmatrix} -3 \\ -8 \\ -4 \\ 5 \end{pmatrix} + t_2 \begin{pmatrix} 3 \\ 0 \\ -1 \\ 4 \end{pmatrix} + t_3 \begin{pmatrix} -1 \\ 2 \\ 1 \\ -1 \end{pmatrix} + t_4 \begin{pmatrix} -1 \\ 2 \\ 3 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 5 \end{pmatrix}$$

So then the matrix $A$ and vectors **b**, and **x** in $Ax = b$ are...

$$A = \begin{pmatrix} -3 & 3 & -1 & -1 \\ -8 & 0 & 2 & 2 \\ -4 & -1 & 1 & 3 \\ 5 & 4 & -1 & 0 \end{pmatrix}, b = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 5 \end{pmatrix}, x = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{pmatrix}$$

putting it into code to solve...

$$A = \begin{pmatrix} -3 & 3 & -1 & -1 \\ -8 & 0 & 2 & 2 \\ -4 & -1 & 1 & 3 \\ 5 & 4 & -1 & 0 \end{pmatrix}$$

In [29]:
```
A = np.array([[-3,  3, -1, -1],
              [-8,  0,  2,  2],
              [-4, -1,  1,  3],
              [ 5,  4, -1,  0]])
```

In [30]:
```
b = np.array([[-1],
              [0],
              [0],
              [5]])
```

In [34]:
```
x = solve(A, b)
print(x)
```

```
[[0.56521739]
 [0.98550725]
 [1.76811594]
 [0.49275362]]
```

In [36]:
```
# checking the solution by recalculating b
print(np.matmul(A, x))
```

```
[[-1.]
 [ 0.]
 [ 0.]
 [ 5.]]
```

**Answer.** So my solution is in fractions, shown below. I would also like to note that trades can be done in reverse, and debt can be taken on. The later is to ensure that addition is colmunative, and the former is to ensure that vectors can be subtracted. I can't spell to save my life.

$$x = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{pmatrix} = \begin{pmatrix} 0.56521739 \\ 0.98550725 \\ 1.76811594 \\ 0.49275362 \end{pmatrix}$$