CS 132 – Spring 2020, Assignment 4

Answer 1.

If we start by simplifying our matrix, we get that:

$$\begin{bmatrix} 1 & 3 & 9 & 2 \\ 1 & 0 & 3 & -4 \\ 0 & 1 & 2 & 3 \\ -2 & 3 & 0 & 5 \end{bmatrix} \equiv \begin{bmatrix} 1 & 3 & 9 & 2 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With this we know that we have 1 free variable, x_3 and a variable that always has to be 0, x_4, so if we solve the system bellow with $x_3 = 1$ then:

$$\begin{cases} x_1 = -3x_2 - 9x_3 - 2x_4 \\ x_2 = -3x_3 + 4x_4 \\ x_4 = 0 \end{cases} \equiv \begin{cases} x_1 = 0 \\ x_2 = -3 \\ x_4 = 0 \end{cases}$$

Hence, all $x = x_3 \times \begin{bmatrix} 0 \\ -3 \\ 1 \\ 0 \end{bmatrix}$ *would map to the zero vector.*

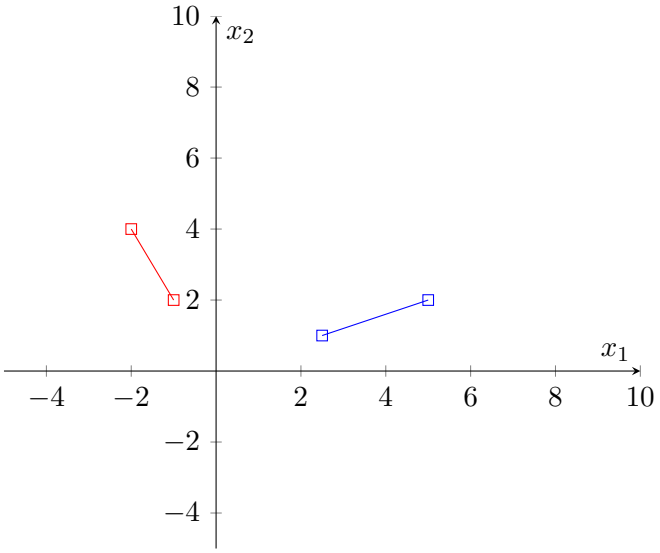
Answer 2.

If b is in said range, then the following matrix must be consistent:

$$\begin{bmatrix} 1 & 3 & 9 & 2 & -1 \\ 1 & 0 & 3 & -4 & 3 \\ 0 & 1 & 2 & 3 & -1 \\ -2 & 3 & 0 & 5 & 4 \end{bmatrix} \equiv \begin{bmatrix} 1 & 3 & 9 & 2 & -1 \\ 0 & 1 & 2 & 3 & -1 \\ 0 & 0 & 0 & 1 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

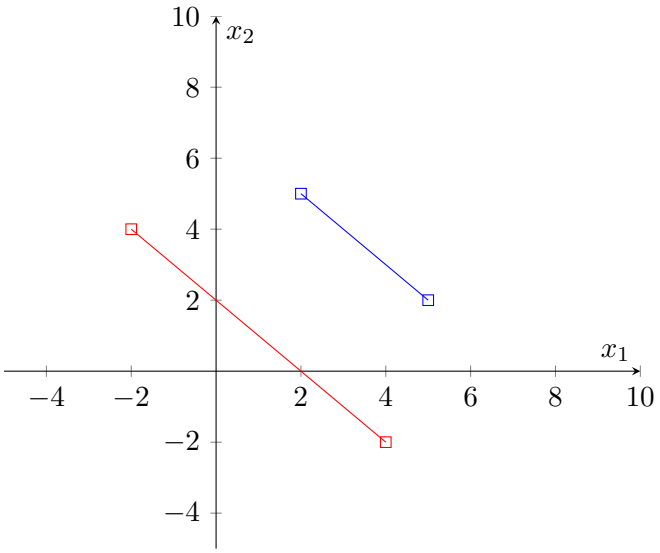
However, as we can clearly see, we have reached an impossible system, since 0 + 0 + 0 + 0 cannot equal 1. As such, b is not in the range.

Answer 3.



A crop both vertically horizontally by half

Answer 4.



Reflection on the line $y = x$

Answer 5.

$$A = \begin{bmatrix} -2 & 7 \\ 5 & -3 \end{bmatrix}$$

Answer 6.

Firstly, we know A will have to be a 3x2 matrix, and so, the result of each multiplication by a unit vector will give us a column, meaning that $A = \begin{bmatrix} 1 & 4 & -5 \\ 3 & -7 & 4 \end{bmatrix}$

Answer 7.

Knowing that $\frac{-\pi}{4}$ *radians is -45°, we can just apply this transformation to the two unit vectors, with the first one being* $A(e_1) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$ *and the second one being* $A(e_2) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$

From here we can construct the matrix $A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$

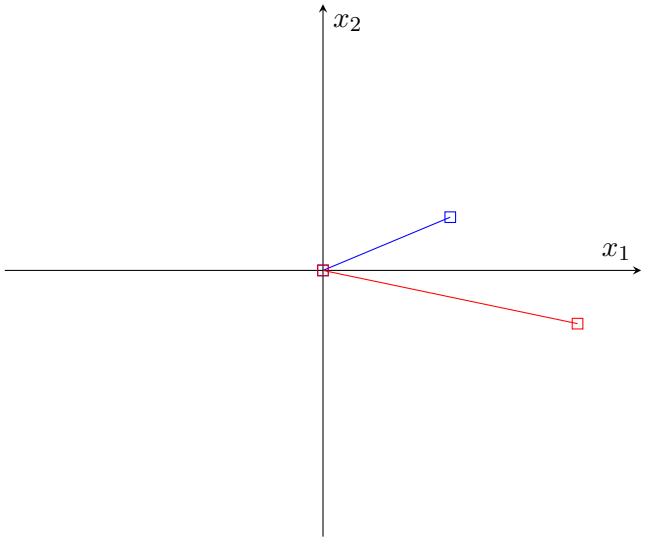
Answer 8.

Applying the same logic as before, we know that $A(e_1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ *and that* $A(e_2) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$

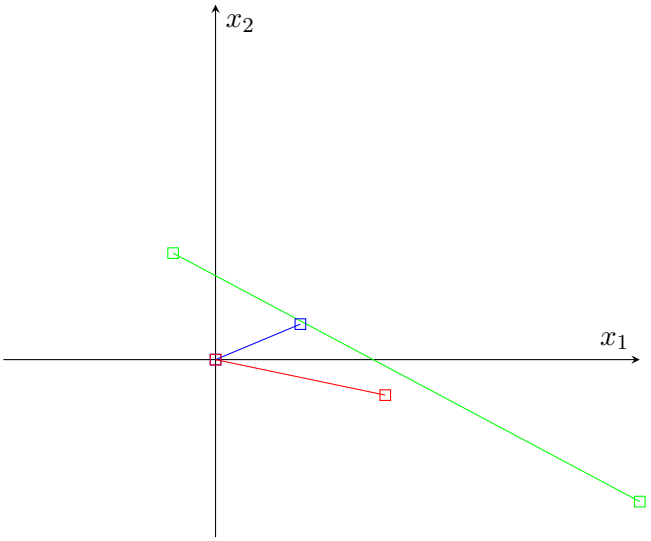
From here we can construct the matrix $A = \begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix}$

Answer 9.

Given the following picture as a representation of the one given to us, as we don't actually have that one:



If we apply this to $\begin{bmatrix} -1 \\ 3 \end{bmatrix}$ *we get that:*



Answer 10.

Knowing that the matrix will be a 2x3, we can construct it from each transformation and get that: $A = \begin{bmatrix} -3 & 2 \\ 1 & -4 \\ 0 & 1 \end{bmatrix}$

Answer 11.

Knowing that the matrix will be a 4x1, we can construct it from each transformation and get that:

$$A = \begin{bmatrix} 2 & 0 & 3 & -4 \end{bmatrix}$$

Answer 12.

First lets build the matrix: $A = \begin{bmatrix} 1 & -2 \\ -1 & 3 \\ 3 & -2 \end{bmatrix}$

Now, lets complete the matrix and solve it:

$$\begin{bmatrix} 1 & -2 & -1 \\ -1 & 3 & 4 \\ 3 & -2 & 9 \end{bmatrix} \equiv \begin{bmatrix} 1 & -2 & -1 \\ 0 & 1 & 3 \\ 0 & 0 & -2 \end{bmatrix}$$

This is an inconsistent system, so there is no x for that solution.

Answer 13.

In [2]:
```
def getPopulation(val):
    lst = [0,0]

    lst[0] = 0.94 * val[0] + 0.04 * val[1]
    lst[1] = 0.06 * val[0] + 0.96 * val[1]

    return lst

val2015 = [10000000, 800000]

val2016 = getPopulation(val2015)

val2017 = getPopulation(val2016)

print(val2015, val2016, val2017)
```

```
[10000000, 800000] [9432000.0, 1368000.0] [8920800.0, 1879200.0]
```

Answer 14.

```
In [7]: import math

        def getCars(val):
            lst = [0,0,0]

            lst[0] = round(0.97 * val[0]) + round(0.05 * val[1]) + round(0.10 * val[2])
            lst[1] = round(0.00 * val[0]) + math.floor(0.90 * val[1]) + round(0.05 * val[2])
            lst[2] = round(0.03 * val[0]) + round(0.05 * val[1]) + round(0.85 * val[2])

            return lst

        valM = [295, 55, 150]

        valT = getCars(valM)

        valW = getCars(valT)

        print(valM, valT, valW)
```

```
[295, 55, 150] [304, 57, 140] [312, 58, 131]
```

Answer 15.

Since the resulting matrix has the amount of rows of A and columns of B, we are working with a 2x2 zero matrix, here we have two options, find the inverse numbers so that each row adds up to 0, or simply find the solution that zeros every element on multiplication.

Here, by observation we can see that a matrix $\begin{bmatrix}2\\1\end{bmatrix}$ *will zero a single column of the resulting matrix, so, if we apply this again, we get a zero matrix, with:*

$$B = \begin{bmatrix}2 & 2\\1 & 1\end{bmatrix}$$

Answer 16.

A *False*
B *True*
C *False*
D *True*
E *False*

Answer 17.

If $B = [b_1 b_2 b_3 ...]$ *and* $b_1 = b_2$ *then:*
$AB = [Ab_1 Ab_2 Ab_3 ..]$ *so, the first two columns will be equal in AB.*

Answer 18.

Applying the same logic as before, if $B = [b_1 b_2 b_3 ...]$ *and* $b_2 = 0$ *then:*
$AB = [Ab_1 Ab_2 Ab_3 ..] = [Ab_1 0 Ab_3 ..]$ *so, the second column of AB is all 0s.*

Answer 19.

If all the columns of B are linearly dependent, then you have, at least, a row of 0s, and, as such, a free variable. Applying the same logic as before where $B = [b_1 b_2 b_3 ...]$ *and* $AB = [Ab_1 Ab_2 Ab_3 ...]$*, then for each column of AB, the row that is zeroed in B, will be zeroed, meaning AB will also have a free variable, making it linearly dependent.*

Answer 20.

For the first comparison, we have by theorem 3c, that $r(A)^T = (rA)^T$*, this property can be transferred to vectors, as they are simply mx1 matrices, and same for the scalar, as the inner product works as a scalar value, so*
$u^T v = v^T u$.
On the other hand, uv^T *and* vu^T *are transposes of one another.*

Answer 21.

In [1]:
```
import numpy as np

a = np.zeros((5,6))

b = np.ones((3,5))

c = np.identity(6)

d = np.zeros((5, 5))
np.fill_diagonal(d, [3,5,7,2,4])

print(a)
print()
print(b)
print()
print(c)
print()
print(d)
print()
```

```
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]

[[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[3. 0. 0. 0. 0.]
 [0. 5. 0. 0. 0.]
 [0. 0. 7. 0. 0.]
 [0. 0. 0. 2. 0.]
 [0. 0. 0. 0. 4.]]
```

Answer 22.

Using the Numpy random function, we get values between 0 and 1.

In [2]:
```
a = np.random.rand(6,4)

b = np.random.randint(-9, 10, size=(3,3))

print(a)
print()
print(b)
```

```
[[0.85686825 0.09540811 0.63970254 0.08333437]
 [0.6164653  0.23919012 0.12196163 0.76965196]
 [0.30556139 0.69166998 0.04790988 0.35300464]
 [0.41213625 0.99896008 0.32030461 0.0401705 ]
 [0.08930268 0.98105096 0.51016964 0.17446602]
 [0.04921934 0.05253567 0.43313012 0.88371839]]

[[-6  1  9]
 [ 4  9 -5]
 [ 3  5 -7]]
```

Answer 23.

Despite the fact that, when using the identity matrix, we get a zero matrix, if we use a random one, we do not get the same thing.

In [3]:
```
def computeVal(A,B):
    C = np.add(A,B)
    D = np.subtract(A,B)

    E = np.matmul(C, D)

    F = np.matmul(A,A)
    G = np.subtract(F,B)

    return np.subtract(E,G)

def computeVal2A(A,B):
    C = np.add(A,B)
    D = np.subtract(A,B)

    return np.matmul(C, D)

def computeVal2B(A, B):
    C = np.matmul(A, A)
    D = np.subtract(B,B)

    return np.subtract(C,D)

def compareVal(A,B):
    C = computeVal2A(A,B)
    D = computeVal2B(A,B)
    return np.array_equal(C,D)

A = np.random.randint(0, 100, size=(4,4))
B = np.random.randint(0, 100, size=(4,4))
C = np.random.randint(0, 100, size=(4,4))
I = np.identity(4)

g = computeVal(A,I)
h = computeVal(B,I)
i = computeVal(C,I)

D1 = np.random.randint(0, 100, size=(4,4))
D2 = np.random.randint(0, 100, size=(4,4))
E1 = np.random.randint(0, 100, size=(4,4))
E2 = np.random.randint(0, 100, size=(4,4))
F1 = np.random.randint(0, 100, size=(4,4))
F2 = np.random.randint(0, 100, size=(4,4))

j = compareVal(D1, D2)
k = compareVal(E1, E2)
l = compareVal(F1, F2)

print(g)
print()
print(h)
print()
print(i)
print()
print(j, k, l)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

False False False
```