CE/CZ 1104 Linear Algebra for Computing

Lab 2

Instructions: There are 3 exercises in this lab with questions for each exercise. All questions require answers to be in the form of outputs to your Python script running in Jupyter notebook. Convert the notebook to pdf format (File -> Print Preview) and upload at Content -> Part 1 -> Lab -> Lab 2 -> your lab group folder.

Exercise 1: Computer Graphics – Linear Transformations

In the lectures, we have seen that matrices represent linear transformations such as scaling, translation, rotation and shear. The following table shows the transformations and the corresponding matrices when the input is pixel location is given by the vector $[x \quad y \quad 1]^T$. The 1 in the last row of the vector arises from the concept of 'homogeneous co-ordinates', which facilitate the transformations to be represented as matrices. You will study homogeneous co-ordinates in the computer graphics course.

Transformation type	Transformation matrix	Pixel mapping
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x$ $y' = y$
Scaling	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = c_x \times x$ $y' = c_y \times y$
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x\cos\theta + y\sin\theta$ $y' = -x\sin\theta + y\cos\theta$
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x + t_x$ $y' = y + t_y$
Horizontal shear	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x + s_h \times y$ $y' = y$
Vertical shear	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x$ $x' = s_v \times x + y$

We will implement the above transformations by applying them on the (x, y) co-ordinates of four points represented by the vectors $\mathbf{a} = [0 \quad 1 \quad 0]^T$, $\mathbf{b} = [1 \quad 0 \quad 1]^T$, $\mathbf{c} = [0 \quad -1 \quad 2]^T$, $\mathbf{d} = [-1 \quad 0 \quad 3]^T$, where the last component of each vector is an ascii character index.

Question 1:

Use the code available in NTULearn to plot these points. Note the application of Identity transformation in the code.

Question 2:

Modify the above code to implement and the display the results of the following transformations: (i) scaling transformation with scale of 2, (ii) rotation transformation with 90°, (iii) translation, horizontal shear and vertical shear using your own parameters.

Question 3:

Modify the code to implement a combination of the rotation and scaling transformations, i.e., a rotation followed by scaling. Note that since the transformations are linear, a combination of transformations is represented simply as a product of the matrices representing the individual transformation.

Exercise 2: Web Search – PageRank (not quite, but almost)

Google's search algorithm to rank web pages, also called PageRank algorithm, is based on the following question: in a particular group of people, who is the most popular? Let us say there are 4 people – Alpha, Bravo, Charlie, Delta – who we will call A, B, C and D. Everyone in the group is asked to list their friends in this group. The outcome is as follows: A lists B and C; B lists A, C and D; C lists A, B and D; D lists A and C. This list can be represented as a 4 x 4 matrix as follows:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Some people might list every one they ever met and others only list closest friends. This is compensated by normalizing the matrix, i.e., by dividing each list by the number of people in it to obtain the **linking matrix**:

$$L = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/3 & 1/2 \\ 1/2 & 0 & 1/3 & 0 \\ 1/2 & 1/3 & 0 & 1/2 \\ 0 & 1/3 & 1/3 & 0 \end{bmatrix} \end{array}.$$

Next, we associate a non-negative number to each person that reflects that person's **popularity** and collect all the numbers into a **popularity vector** $\mathbf{r} = \begin{bmatrix} r_A \\ r_B \\ r_C \\ r_D \end{bmatrix}$. Let *a person's popularity be the weighted sum of the popularity of people who reference that person,* for example, $r_A = \frac{1}{3}r_B + \frac{1}{3}r_C + \frac{1}{2}r_D$. The equations can be written as

$$L\mathbf{r} = \mathbf{r}, \text{ which is the same as } (L - I)\mathbf{r} = \mathbf{0}.$$

The above equation can be solved using reduced row echelon form (use the code for reduced row echelon form available in NTULearn) by setting $r_D = 1$ (arbitrarily) to get the values of r_A, r_B, r_C.

Question 4:

Find the values of r_A, r_B, r_C from the above data. Ensure that you use fractions in the linking matrix so that the sum of the columns in the matrix is 1.

PageRank essentially looks at a webpage *w* and determines which other web pages *w* links to, thus associating to each webpage *w* a linking matrix of 0's and 1's, and then normalizes it. Rest is just as we did in the popularity problem.

However, there are a few questions to be answered:

(a) Does the equation L**r** = **r** above always have a solution?

(b) Will a solution have entries that are nonnegative?

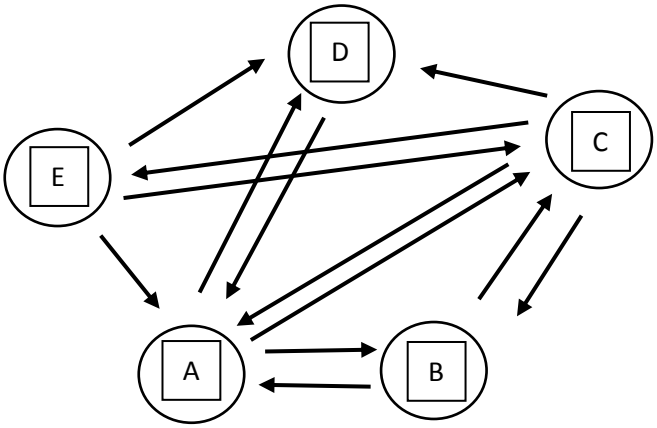
(c) Is the solution unique? If not, we will have conflicting rankings.

To the rescue comes the Perron-Frobenius Theorem:

For any matrix *L* having all entries nonnegative and each column summing to 1, the equation $L\boldsymbol{r} = \boldsymbol{r}$ has a nonnegative solution ***r***.

Question 5:

Suppose that we have five websites A, B, C, D, and E. Let's also suppose that the links between the various sites are given by the graph below. The arrow pointing from C to D means that C's site links to D's, etc



- Create a linking matrix L which contains the information of which site links to which just as you did in the popularity example. Remember to normalize, and be sure that your input is exact (for example, make sure you enter 1/3 instead of .333 since our columns must sum to 1).
- Find all solutions **x** to the matrix equation (L - I)**x** = **0**.

Exercise 3: Epidemic Dynamics – SIR model

The SIR model is a simple mathematical description of the spread of a disease in a population. It divides the (fixed) population of *N* individuals into four "compartments" which may vary as a function of time, *t*:

- Susceptible *S(t)*: not yet infected but can acquire the disease the next day;
- Infected *I(t)*: have the disease;
- Recovered *R(t)*: had the disease, have recovered, and now have immunity to it.
- Deceased *D(t)*: had the disease, and unfortunately, died.

The above information could be put together in a normalized vector, e.g., $\mathbf{x}_t = [0.75 \quad 0.1 \quad 0.1 \quad 0.05]$.

Question 6:

Suppose the progression of the disease over each day is as follows:

- Among susceptible population
  - 5% acquires the disease
  - 95% remains susceptible
- Among infected population
  - 1% dies
  - 10% recovers with immunity
  - 4% recover without immunity (i.e., remain susceptible)
  - 85% remain infected
- 100% of immune and dead people remain in their state

Write a matrix *P* containing the above information. Using the state of the disease given by vector $\mathbf{x}_t$ above, what is the state of the disease the next day? Note that $\mathbf{x}_{t+1} = P\mathbf{x}_t$.

Question 7:

Assume $\mathbf{x}_1 = [1 \quad 0 \quad 0 \quad 0]$. Find the progression of the disease for each day from $t = 2$ till $t = 200$. For each of the compartments – *S*, *I*, *R* and *D* – plot the progression from day 1 to day 200 on the same plot.

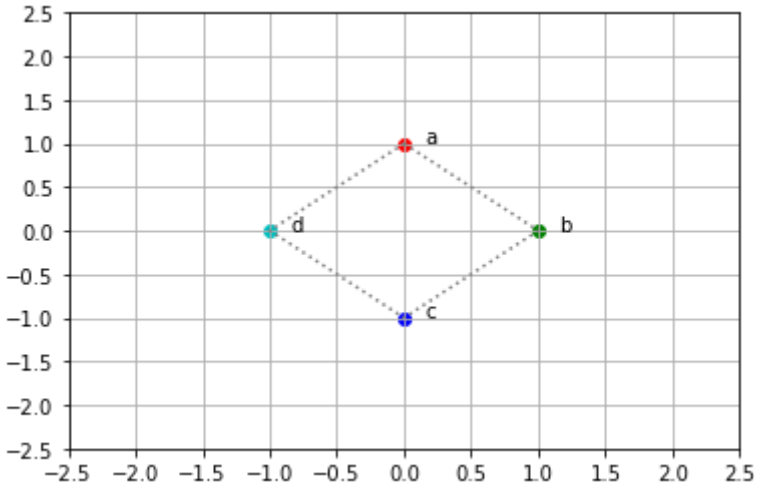
CE/CZ 1104 Linear Algebra for Computing

Lab 2

Exercise 1: Computer Graphics – Linear Transformations

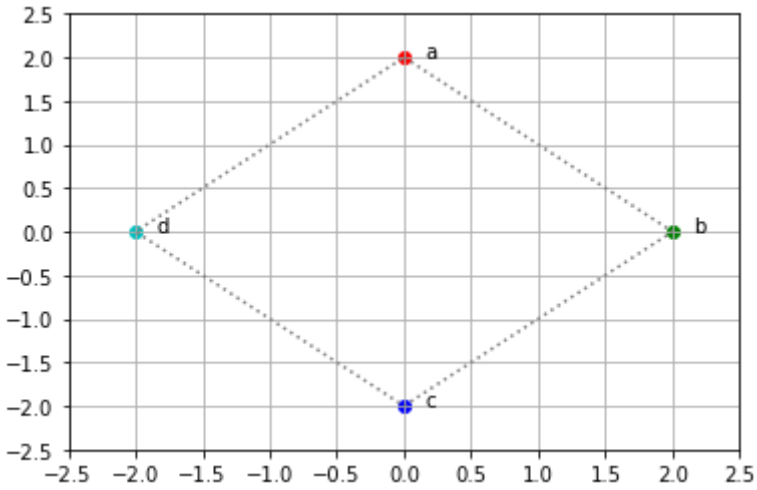
Question 1

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import string

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# 3x3 Identity transformation matrix
I = np.eye(3) #float

def transform(A,I):
    color_lut = 'rgbc' #4 colors to represent 4 points
    fig = plt.figure()
    ax = plt.gca()
    xs = []
    ys = []
    for row in A:
        output_row = I @ row
        x, y, i = output_row
        xs.append(x)
        ys.append(y)
        i = int(i) # convert float to int for indexing
        c = color_lut[i]
        plt.scatter(x, y, color=c)
        plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
    xs.append(xs[0])
    ys.append(ys[0])
    plt.plot(xs, ys, color="gray", linestyle='dotted')
    ax.set_xticks(np.arange(-2.5, 3, 0.5))
    ax.set_yticks(np.arange(-2.5, 3, 0.5))
    plt.grid()
    plt.show()

transform(A,I)
```



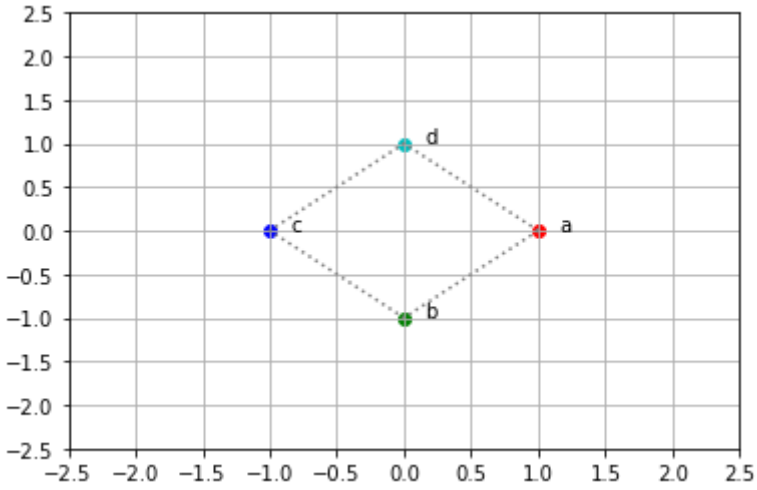
Question 2 (i)

In [2]:
```
# points a, b and, c
A_scale = np.array([(0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)])
# create the scaling transformation matrix
I_scale = np.array([(2, 0, 0), (0, 2, 0), (0, 0, 1)])

transform(A_scale, I_scale)
```



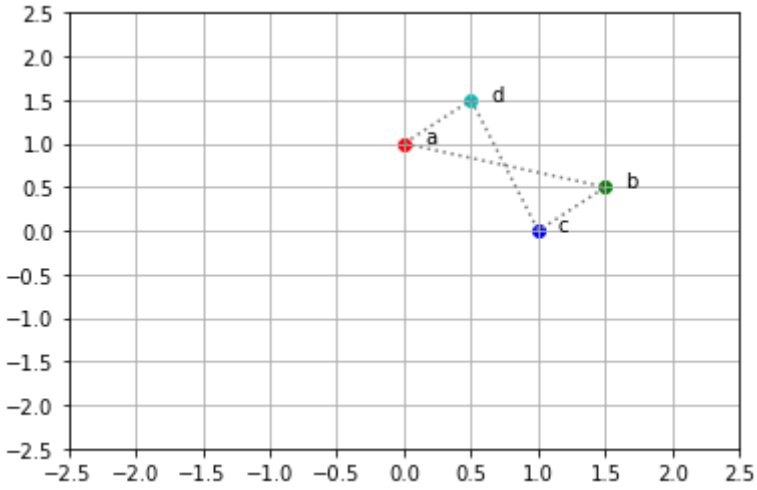
Question 2 (ii)

In [3]:
```
import math

A_rotate = np.array([(0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)])
I_rotate = np.vstack([(np.cos(np.pi/2), np.sin(np.pi/2),0),(-(np.sin(np.pi/2)), np.c
transform(A_rotate, I_rotate)
```



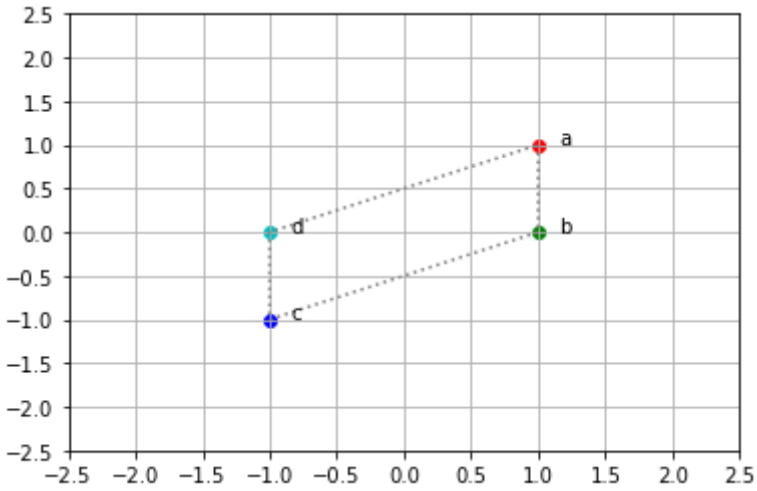
Question 2 (iii) - Translation

In [4]:
```
# x = 0.5 units, y = 0.5 units
A_translate = np.array([(0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)])
I_translate = np.array([(1, 0, 0.5), (0, 1, 0.5), (0, 0, 1)])
transform(A_translate, I_translate)
```



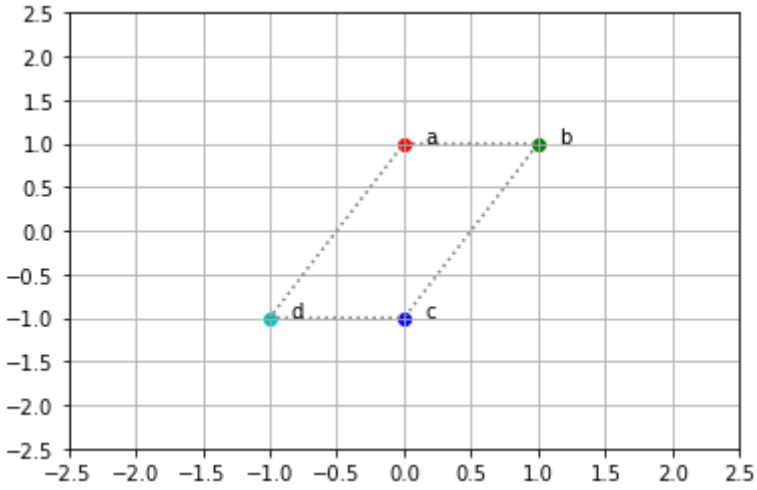
Question 2 (iii) - Horizontal Shear

In [5]:
```
# 1 unit horizontal shear
A_hShear = np.array([(0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)])
I_hShear = np.array([(1, 1, 0), (0, 1, 0), (0, 0, 1)])
transform(A_hShear, I_hShear)
```



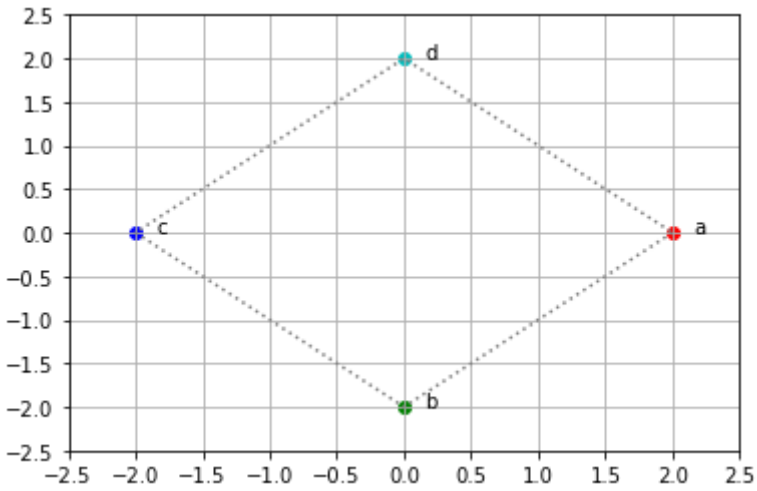
Question 2 (iii) - Vertical Shear

In [6]:
```
# 1 unit vertical shear
A_vShear = np.array([(0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)])
I_vShear = np.array([(1, 0, 0), (1, 1, 0), (0, 0, 1)])
transform(A_vShear, I_vShear)
```



Question 3

In [7]:
```
# scale + rotate
A_scale_rotate = np.array([(0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)])
I_rotate = np.vstack([(np.cos(np.pi/2), np.sin(np.pi/2),0),(-(np.sin(np.pi/2)), np.c
I_scale = np.array([(2, 0, 0), (0, 2, 0), (0, 0, 1)])
# @ combines
I_scale_rotate = I_rotate @ I_scale
transform(A_scale_rotate, I_scale_rotate)
```



---------- *end of excercise 1* ----------

Exercise 2: Web Search – PageRank (not quite, but almost)

Question 4

In [8]:
```
web_graph = {
    'a': {'b', 'c'}, # A lists B and C
    'b': {'a', 'c', 'd'}, # B lists A, C and D
    'c': {'a', 'b', 'd'}, # C lists A, B and D
    'd': {'a', 'c'}} # D lists A and C

pages = sorted(web_graph.keys())
```

```
web_graph
```

Out[8]:
```
{'a': {'b', 'c'}, 'b': {'a', 'c', 'd'}, 'c': {'a', 'b', 'd'}, 'd': {'a', 'c'}}
```

In [9]:
```
import pandas as pd

def get_adj_matrix(adj_list):
  pages = sorted(adj_list.keys())
  A = pd.DataFrame(0, index = pages, columns= pages)
  for source in adj_list:
    for sink in adj_list[source]:
      A.loc[sink, source] = 1
  return A

web_matrix = get_adj_matrix(web_graph)

web_matrix
```

Out[9]:

	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	1
d	0	1	1	0

In [10]:
```
def get_transition_matrix(web_matrix):
  result = web_matrix / web_matrix.sum(axis = 0)
  return result

L = get_transition_matrix(web_matrix)

L
```

Out[10]:

	a	b	c	d
a	0.0	0.333333	0.333333	0.5
b	0.5	0.000000	0.333333	0.0
c	0.5	0.333333	0.000000	0.5
d	0.0	0.333333	0.333333	0.0

In [11]:
```
# create an identity matrix in dataframe form
I = pd.DataFrame(np.eye(4), index=list('abcd'), columns=list('abcd'))

# (L-I)r = 0
aug_matrix = L.subtract(I)

# popularity vector r
r = [0, 0, 0, 0]

aug_matrix['r'] = r

aug_matrix
```

Out[11]:

	a	b	c	d	r
a	-1.0	0.333333	0.333333	0.5	0

	a	b	c	d	r
b	0.5	-1.000000	0.333333	0.0	0
c	0.5	0.333333	-1.000000	0.5	0
d	0.0	0.333333	0.333333	-1.0	0

In [12]:
```
'''Function to transform a matrix to reduced row echelon form'''
def rref(A):
    tol = 1e-16
    #A = B.copy()
    rows, cols = A.shape
    r = 0
    pivots_pos = []
    row_exchanges = np.arange(rows)
    for c in range(cols):
        ## Find the pivot row:
        pivot = np.argmax (np.abs (A[r:rows,c])) + r
        m = np.abs(A[pivot, c])
        if m <= tol:
        ## Skip column c, making sure the approximately zero terms are
        ## actually zero.
            A[r:rows, c] = np.zeros(rows-r)
        else:
            ## keep track of bound variables
            pivots_pos.append((r,c))

            if pivot != r:
                ## Swap current row and pivot row
                A[[pivot, r], c:cols] = A[[r, pivot], c:cols]
                row_exchanges[[pivot,r]] = row_exchanges[[r,pivot]]

            ## Normalize pivot row
            A[r, c:cols] = A[r, c:cols] / A[r, c];

            ## Eliminate the current column
            v = A[r, c:cols]
            ## Above (before row r):
            if r > 0:
                ridx_above = np.arange(r)
                A[ridx_above, c:cols] = A[ridx_above, c:cols] - np.outer(v, A[ridx_a
                ## Below (after row r):
            if r < rows-1:
                ridx_below = np.arange(r+1,rows)
                A[ridx_below, c:cols] = A[ridx_below, c:cols] - np.outer(v, A[ridx_b
                r += 1
        ## Check if done
        if r == rows:
            break;
    return A
```

In [13]:
```
# convert dataframe to np array
aug_matrix_array = np.array(aug_matrix * 6)
rref(aug_matrix_array)
```

Out[13]:
```
array([[ 1.    ,  0.    ,  0.    , -1.5   , -0.    ],
       [ 0.    ,  1.    ,  0.    , -1.3125, -0.    ],
       [ 0.    ,  0.    ,  1.    , -1.6875, -0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ]])
```

==========

Final working:

- rA - 1.5 = 0
- rB - 1.3125 = 0
- rC - 1.6875 = 0
- rD is a free variable

Therefore, the values are: rA = 1.5, rB = 1.3125, rC = 1.6875

Questions to be answered:

- (a) Does the equation Lr = r above always have a solution?

Yes, the equation Lr = r will always have a solution.

- (b) Will a solution have entries that are nonnegative?

Yes, there will be a solution with nonnegative entries.

- (c) Is the solution unique? If not, we will have conflicting rankings.

No, the solution is not unique. This is because of the augmented matrix where rD is a free variable. Hence, there is an infinite number of solutions.

==========

Question 5

In [14]:
```
link_matrix = np.array([(0, 1/2, 1/4, 1, 1/3), (1/3, 0, 1/4, 0, 0), (1/3, 1/2, 0, 0,
identity_matrix = np.eye(5)
zero_matrix = np.array([[0],[0],[0],[0],[0]])
link_identity_matrix = np.vstack([3 * (link_matrix - identity_matrix)])
```

In [15]:
```
aug_matrix2 = np.hstack([link_identity_matrix, zero_matrix])
rref_aug_matrix = rref(aug_matrix2)

rA = -(rref_aug_matrix[0,4])
rB = -(rref_aug_matrix[1,4])
rC = -(rref_aug_matrix[2,4])
rD = -(rref_aug_matrix[3,4])

print("All solutions x to the matrix equation (L - I)x = 0:")
print("• rA = ", rA, " * rE")
print("• rB = ", rB, " * rE")
print("• rC = ", rC, " * rE")
print("• rD = ", rD, " * rE")
```

```
All solutions x to the matrix equation (L - I)x = 0:
• rA =  6.333333333333334  * rE
• rB =  3.1111111111111116  * rE
• rC =  4.0  * rE
• rD =  3.4444444444444446  * rE
```

---------- *end of excercise 2* ----------

Exercise 3: Epidemic Dynamics – SIR model

In [16]:
```
P = np.array([[0.95, 0.04, 0, 0], [0.05, 0.85, 0, 0], [0, 0.1, 1, 0], [0, 0.01, 0, 1
```

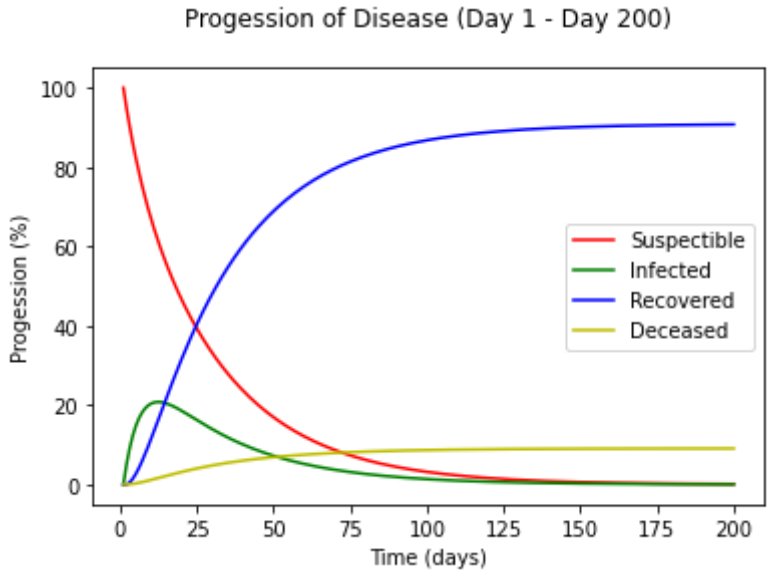
```
x_t = np.array([0.75, 0.1, 0.1, 0.05])
print("matrix P:")
print(P)
```

```
matrix P:
[[0.95 0.04 0.   0.  ]
 [0.05 0.85 0.   0.  ]
 [0.   0.1  1.   0.  ]
 [0.   0.01 0.   1.  ]]
```

In [17]:
```
x_tplus1 = P @ x_t
print("x_t+1 = Px_t =", x_tplus1)
```

```
x_t+1 = Px_t = [0.7165 0.1225 0.11   0.051 ]
```

In [18]:
```
x_1 = np.array([1,0,0,0])

st = []
it = []
rt = []
dt = []

def progress(x,y):
    current = x
    for i in range(1, 201):
        st.append(current[0]*100)
        it.append(current[1]*100)
        rt.append(current[2]*100)
        dt.append(current[3]*100)
        current = y @ current

progress(x_1,P)

z = np.linspace(1, 200, 200)
plt.xlabel("Time (days)")
plt.ylabel("Progession (%)")
plt.plot(z, st, 'r', label = "Suspectible")
plt.plot(z, it, 'g', label = "Infected ")
plt.plot(z, rt, 'b', label = "Recovered")
plt.plot(z, dt, 'y', label = "Deceased")
plt.title("Progession of Disease (Day 1 - Day 200)\n")
plt.legend()
plt.show()
```



---------- *end of excercise 3* ----------