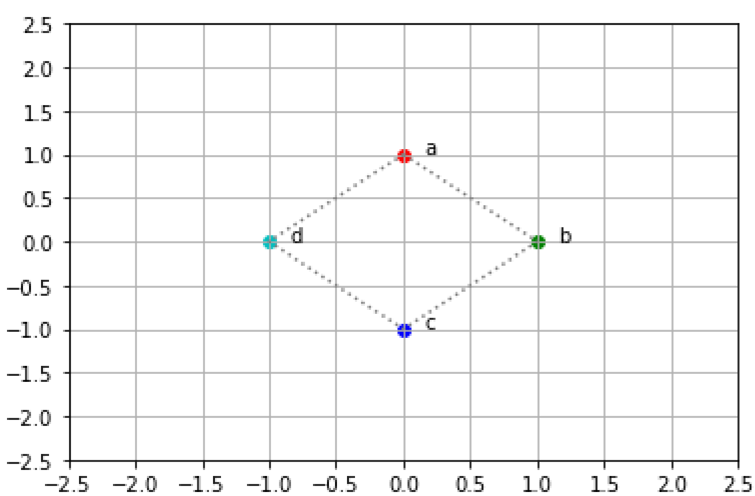
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import string
```

Question 1

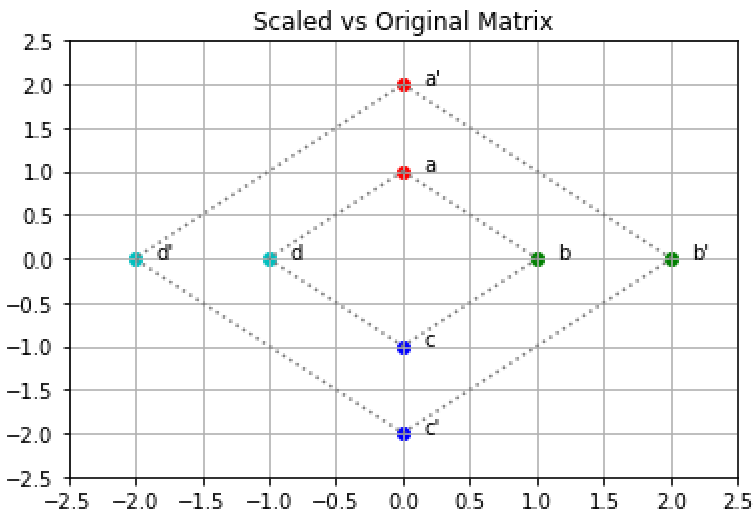
In [2]:
```
# points a, b, c, and d
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

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

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

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()
```



Question 2

part (i) scale by 2

In [3]:
```
# 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 2 Scale transformation matrix
scale_matrix = np.array([[2, 0, 0], [0, 2, 0], [0, 0, 1]])

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs_s = [] #scaled matrix
ys_s = [] #scaled matrix
xs = [] #og matrix
ys = [] #og matrix

for row in A:
    output_row = scale_matrix @ row
    x, y, i = row
    x_s, y_s, i_s = output_row

    xs_s.append(x_s)
    ys_s.append(y_s)
    xs.append(x)
    ys.append(y)

    i, i_s = int(i), int(i_s) # convert float to int for indexing
    c, c_s = color_lut[i], color_lut[i_s] #make seperate colour variable for easier cod

    plt.scatter(x, y, color=c)
    plt.scatter(x_s, y_s, color=c_s)
    plt.text(x + 0.15, y, f"{string.ascii_letters[int(i)]}")
    plt.text(x_s + 0.15, y_s, f"{string.ascii_letters[int(i_s)]}'")

xs_s.append(xs_s[0])
ys_s.append(ys_s[0])
xs.append(xs[0])
ys.append(ys[0])

plt.title('Scaled vs Original Matrix')
plt.plot(xs, ys, color="gray", linestyle='dotted')
plt.plot(xs_s, ys_s, 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()
```

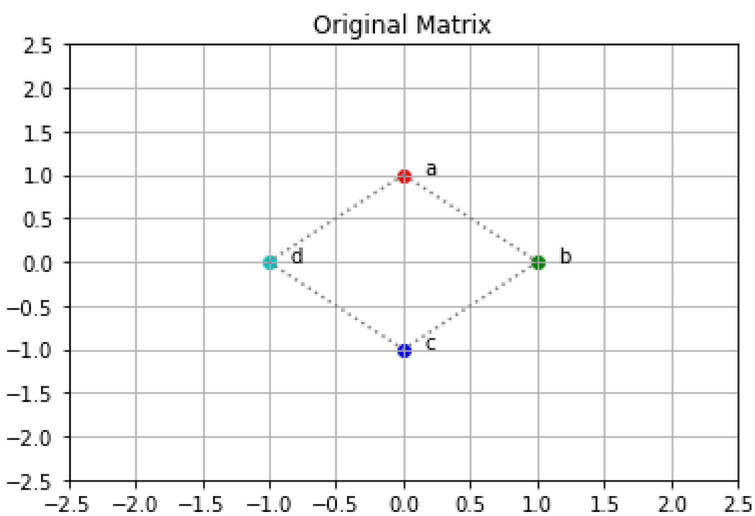


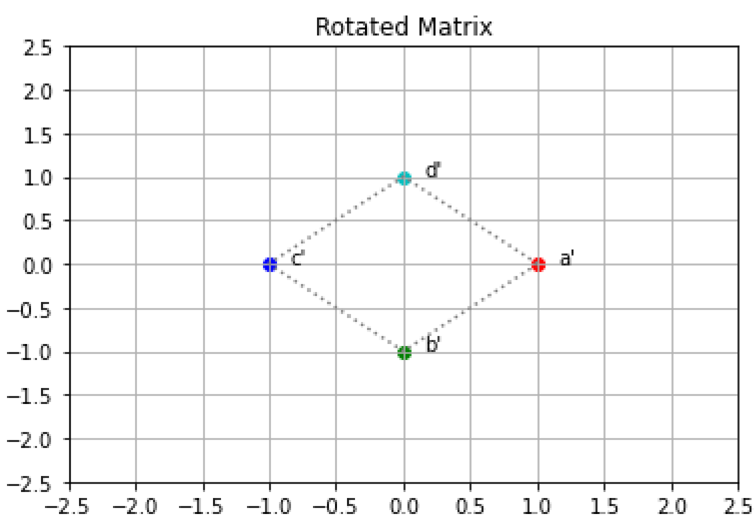
part II rotate by 90 degrees

In [4]:

```
#original matrix plot

# 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

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.title('Original Matrix')
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()


# create the rotation transformation matrix
rotate_matrix = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 1]])

fig = plt.figure()
```

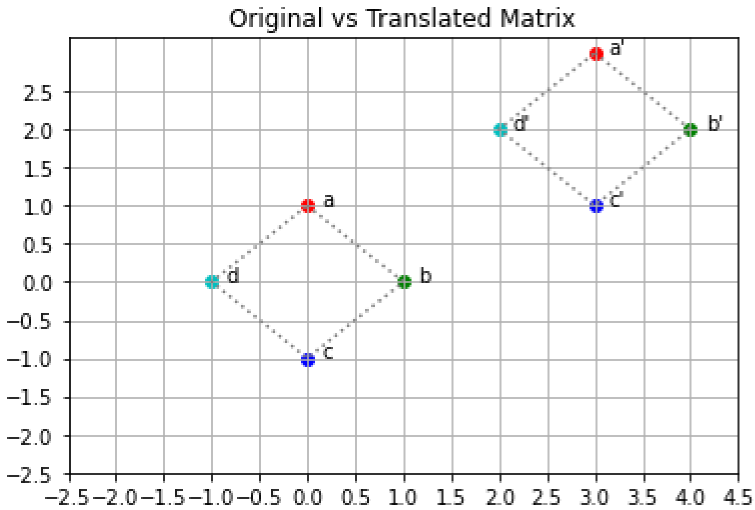
```
ax = plt.gca()
xr = []
yr = []

for row in A:
    output_row = rotate_matrix @ row
    x_r, y_r, i_r = output_row
    i_r = int(i_r) # convert float to int for indexing
    c_r = color_lut[i_r]
    xr.append(x_r)
    yr.append(y_r)
    letter_r = string.ascii_letters[i_r]
    plt.scatter(x_r, y_r, color=c_r)
    plt.text(x_r + 0.15, y_r, f"{letter_r}'")

xr.append(xr[0])
yr.append(yr[0])

plt.title('Rotated Matrix')
plt.plot(xr, yr, 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()
```





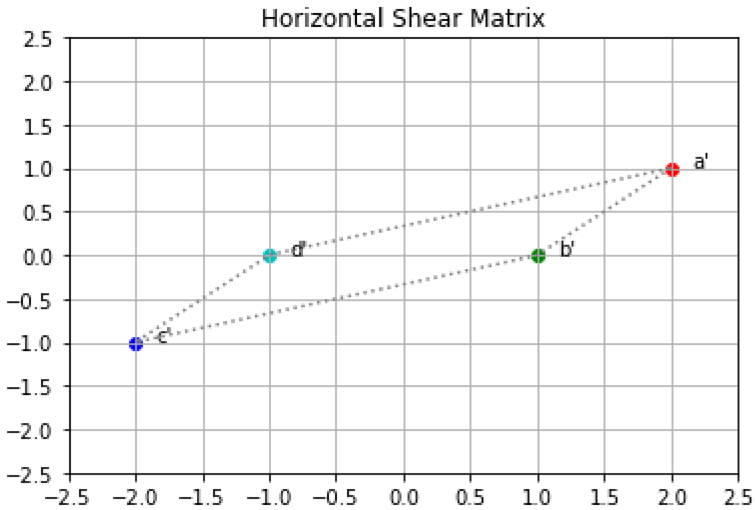
part III translation , horizontal , vertical shear

In [5]:
```
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])

# create the rotation transformation matrix
translate_matrix = np.array([[1, 0, 3], [0, 1, 2], [0, 0, 1]])

fig = plt.figure()
ax = plt.gca()
xt = [] #translated matrix
yt = []
xs = [] #og matrix
ys = []


for row in A:
    x_t = row[0] + translate_matrix[0][2]
    y_t = row[1] + translate_matrix[1][2]
    i_t = row[2] * translate_matrix[2][2]
    x, y, i = row
    i_t, i  = int(i_t), int(i) # convert float to int for indexing
    c_t, c = color_lut[i_t], color_lut[i]

    xs.append(x)
    ys.append(y)
    xt.append(x_t)
    yt.append(y_t)

    letter_t = string.ascii_letters[i_t]
    plt.scatter(x, y, color=c)
    plt.scatter(x_t, y_t, color=c_t)
    plt.text(x_t + 0.15, y_t, f"{letter_t}'")
    plt.text(x + 0.15, y, f"{string.ascii_letters[int(i)]}")

xt.append(xt[0])
yt.append(yt[0])
xs.append(xs[0])
ys.append(ys[0])

plt.title('Original vs Translated Matrix')
plt.plot(xs, ys, color="gray", linestyle='dotted')
plt.plot(xt, yt, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 5, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



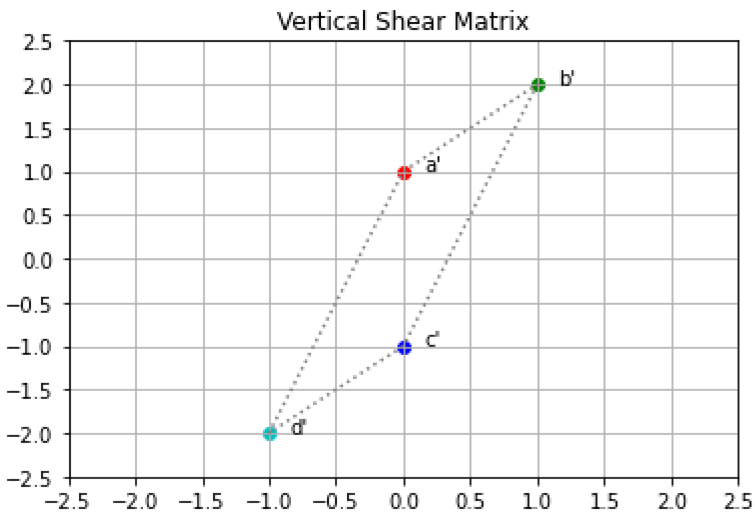
In [6]:

```
# HORIZONTAL SHEAR

# 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])

# create the rotation transformation matrix
sh_matrix = np.array([[1, 2, 0], [0, 1, 0], [0, 0, 1]])

fig = plt.figure()
ax = plt.gca()
xs_sh = []
ys_sh = []

for row in A:
    output_row = sh_matrix @ row
    x_sh, y_sh, i_sh = output_row
    i_sh = int(i_sh) # convert float to int for indexing
    c_sh = color_lut[i_sh]

    xs_sh.append(x_sh)
    ys_sh.append(y_sh)
    plt.scatter(x_sh, y_sh, color=c_sh)
    plt.text(x_sh + 0.15, y_sh, f"{string.ascii_letters[int(i_sh)]}'")

xs_sh.append(xs_sh[0])
ys_sh.append(ys_sh[0])

plt.title('Horizontal Shear Matrix')
plt.plot(xs_sh, ys_sh, 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()
```



In [7]:

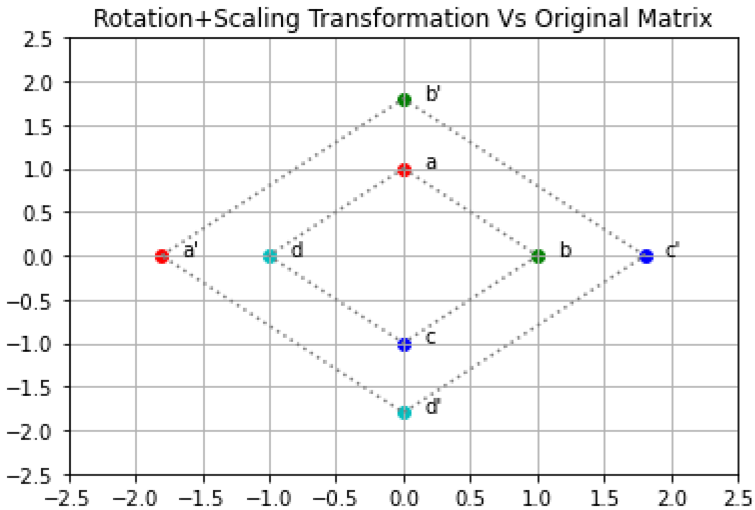
```
# VERTICAL SHEAR

# 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])

# create the rotation transformation matrix
sh_matrix = np.array([[1, 0, 0], [2, 1, 0], [0, 0, 1]])

fig = plt.figure()
ax = plt.gca()
xs_sh = []
ys_sh = []

for row in A:
    output_row = sh_matrix @ row
    x_sh, y_sh, i_sh = output_row
    i_sh = int(i_sh) # convert float to int for indexing
    c_sh = color_lut[i_sh]

    xs_sh.append(x_sh)
    ys_sh.append(y_sh)
    plt.scatter(x_sh, y_sh, color=c_sh)
    plt.text(x_sh + 0.15, y_sh, f"{string.ascii_letters[int(i_sh)]}'")

xs_sh.append(xs_sh[0])
ys_sh.append(ys_sh[0])

plt.title('Vertical Shear Matrix')
plt.plot(xs_sh, ys_sh, 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()
```



Question 3

In [8]:
```
# 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])

# create the rotation transformation matrix by 270 degrees
rotate_matrix = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])
scale_matrix = np.array([[1.8, 0, 0], [0, 1.8, 0], [0, 0, 1]])
combined_matrix = rotate_matrix @ scale_matrix

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs_c = [] #combined matrix
ys_c = []
xs = [] #og matrix
ys = [] #og matrix

for row in A:
    output_row = combined_matrix @ row #order doesn't matter, @ automatically arrange i
    x, y, i = row
    x_s, y_s, i_s = output_row

    xs_c.append(x_s)
    ys_c.append(y_s)
    xs.append(x)
    ys.append(y)

    i, i_s = int(i), int(i_s) # convert float to int for indexing
    c, c_s = color_lut[i], color_lut[i_s] #make seperate colour variable for easier cod

    plt.scatter(x, y, color=c)
    plt.scatter(x_s, y_s, color=c_s)
    plt.text(x + 0.15, y, f"{string.ascii_letters[int(i)]}")
    plt.text(x_s + 0.15, y_s, f"{string.ascii_letters[int(i_s)]}'")

xs_c.append(xs_c[0])
```

```
ys_c.append(ys_c[0])
xs.append(xs[0])
ys.append(ys[0])

plt.title('Rotation+Scaling Transformation Vs Original Matrix')
plt.plot(xs, ys, color="gray", linestyle='dotted')
plt.plot(xs_c, ys_c, 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()
```



Question 4

In [9]:
```
#reduced row echelon form

#Function to transform a matrix to reduced row echelon form
def rref(A):
    tol = 1e-14
    #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_abov
                ## 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_belo
                r += 1
        ## Check if done
        if r == rows:
            break;
    return A
```

In [10]:

```
from scipy.linalg import solve
from fractions import Fraction as frac

A = (frac(0),frac(1,3),frac(1,3),frac(1,2))
B = (frac(1,2),frac(0),frac(1,3),frac(0))
C = (frac(1,2),frac(1,3),frac(0),frac(1,2))
D = (frac(0),frac(1,3),frac(1,3),frac(0))
L = np.array([A,B,C,D], dtype=float)
I = np.eye(4)
zero = np.zeros((4,1))

LI = np.subtract(L,I) #(L-I)r=0 == Ax=b where b=0
rref(LI)
print(LI,"\n")

print("ra - 1.5rd = 0\nrb - 1.3125rd = 0\nrc - 1.6875rd = 0\nrd = free variable\n")
print("ra = 1.5rd\nrb = 1.3125rd\nrc = 1.6875rd\nrd = free variable\n")
print("Let rd = 1, therefore\nra = 1.5\nrb = 1.3125\nrc = 1.6875\n")

print("(a) The equation will always have a solution as there are infinite solutions\n")
print("(b) No, it will not have non negative entries\n")
print("(c) Yes, As long as rD is unique, the solution will be unique.\n")
```

```
[[ 1.      0.      0.     -1.5   ]
 [ 0.      1.      0.     -1.3125]
 [ 0.      0.      1.     -1.6875]
 [ 0.      0.      0.      0.    ]]

ra - 1.5rd = 0
rb - 1.3125rd = 0
rc - 1.6875rd = 0
rd = free variable

ra = 1.5rd
rb = 1.3125rd
rc = 1.6875rd
rd = free variable

Let rd = 1, therefore
ra = 1.5
rb = 1.3125
rc = 1.6875

(a) The equation will always have a solution as there are infinite solutions
```

```
(b) No, it will not have non negative entries

(c) Yes, As long as rD is unique, the solution will be unique.
```

Question 5

In [11]:

```
A = (frac(0,1),frac(1,2),frac(1,4),frac(1),frac(1,3))
B = (frac(1,3),frac(0,1),frac(1,4),frac(0,1),frac(0,1))
C = (frac(1,3),frac(1,2),frac(0,1),frac(0,1),frac(1,3))
D = (frac(1,3),frac(0,1),frac(1,4),frac(0,1),frac(1,3))
E = (frac(0,1),frac(0,1),frac(1,4),frac(0,1),frac(0,1))

zero = np.zeros((5,1))
L = np.array([A,B,C,D,E], dtype=float)
I = np.eye(5)
LI = np.subtract(L,I) #(L-I)r=0 == Ax=b where b=0
rref(LI)

print("ra - 6.3333re = 0\nrb - 3.1111re = 0\nrc - 4re = 0\nrd - 3.4444re = 0\nre = free
print("X: \nra = 6.3333re\nrb = 3.1111re\nrc = 4re\nrd = 3.4444re\nre = free variable\n
print("There are infinite solutions to the matrix equation (L - I)x = 0")
```

```
ra - 6.3333re = 0
rb - 3.1111re = 0
rc - 4re = 0
rd - 3.4444re = 0
re = free variable

X:
ra = 6.3333re
rb = 3.1111re
rc = 4re
rd = 3.4444re
re = free variable

There are infinite solutions to the matrix equation (L - I)x = 0
```

Question 6

In [12]:

```
s_dict = []
i_dict = []
r_dict = []
t_dict = []

s = (19/20,1/25,0,0)
i = (1/20,17/20,0,0)
r = (0,1/10,1,0)
t = (0,1/100,0,1)

P = np.array([s,i,r,t], dtype=float)

x = (0.75, 0.1,0.1,0.05)
x = np.array(x)
x = x.transpose()

Px = np.dot(P, x)
Px = Px.transpose()
```

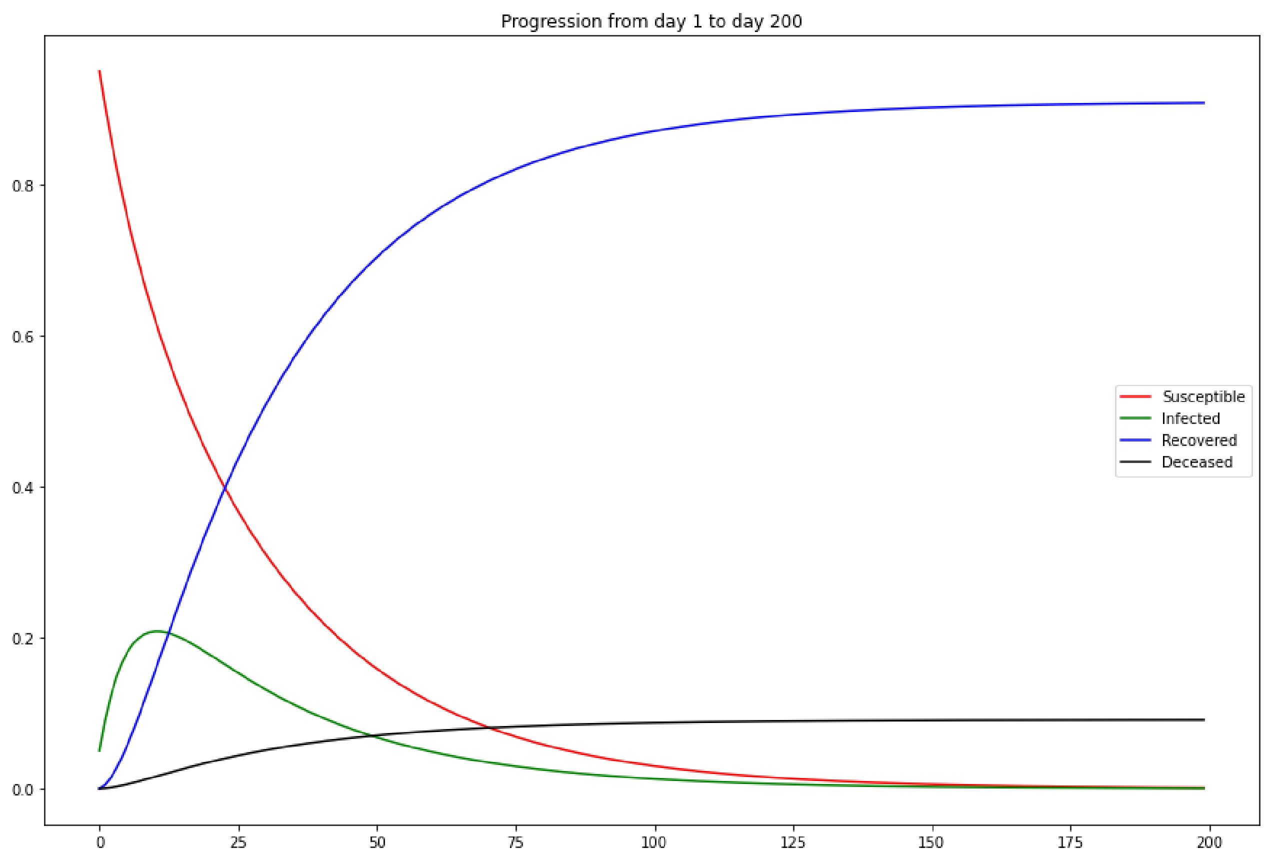
```
print(f"The state of the disease the next day is Px [susceptible , infected, recovered,
```

```
The state of the disease the next day is Px [susceptible , infected, recovered, disease
d] respectively
 = [0.7165 0.1225 0.11   0.051 ]
```

Question 7

In [13]:

```
counter = 0

x = (1, 0,0,0)
x = np.array(x) #1x4 array
x = x.transpose() #4x1 array


while (counter <200): #Repeat modelling everyday with previous day's result
    x = P @ x #Px(subscript t)
    x_t = x.transpose() #Reverse back to original 1x4

    s_dict.append(x_t[0])
    i_dict.append(x_t[1])
    r_dict.append(x_t[2])
    t_dict.append(x_t[3])

    counter += 1

plt.figure(figsize=(15,10))
plt.title('Progression from day 1 to day 200')
plt.plot(s_dict,'r', i_dict,'g', r_dict,'b', t_dict,'k'), plt.legend(['Susceptible','In
plt.show()
```



```
In [ ]:
```