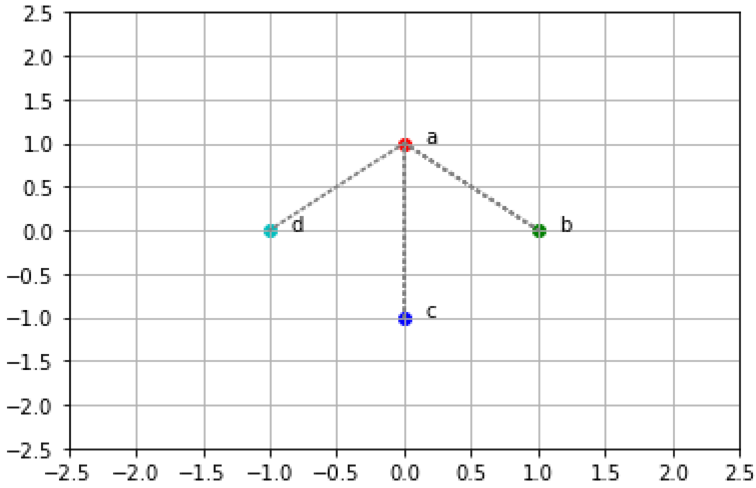
Exercise 1

Question 1

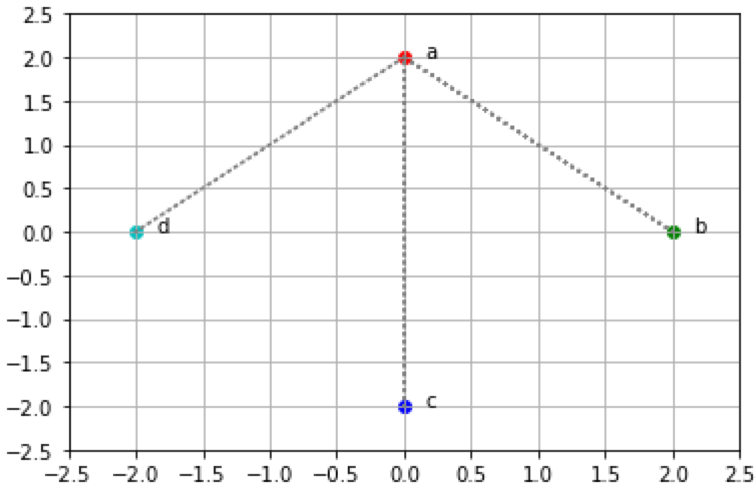
In [36]:

```
import matplotlib.pyplot as plt
import numpy as np
import string

# 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])

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

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

def print_transform(t_matrix, override = False):
    fig = plt.figure()
    ax = plt.gca()
    xs = []
    ys = []

    for row in A:
        row = row.copy()
        # for translate matrix
        if override:
            row[2] = 1
        output_row = t_matrix @ 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()

print_transform(I)
```

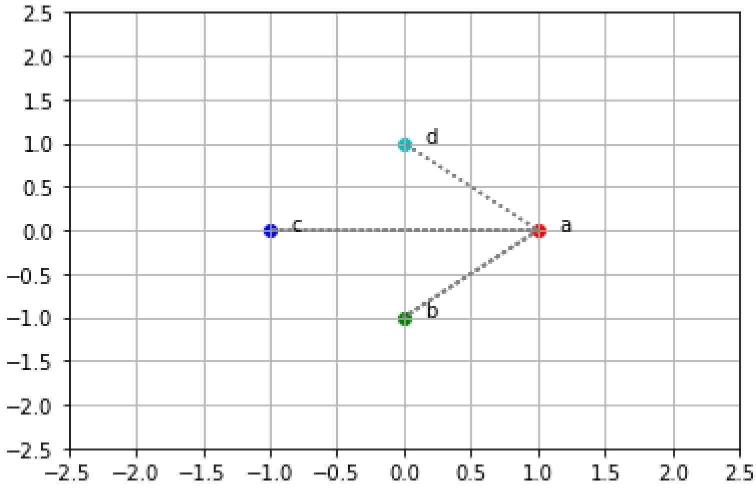


Question 2

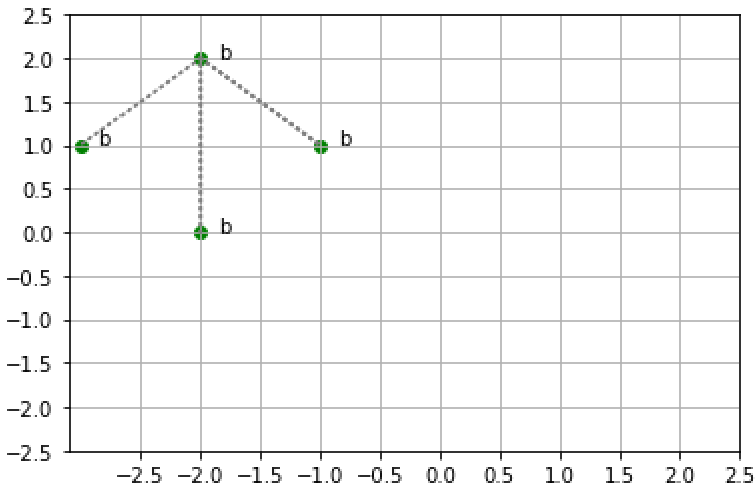
In [37]:
```
# 3x3 2x scaling transformation matrix
S = np.array([[2,0,0],
              [0,2,0],
              [0,0,1]])

print_transform(S, False)
```



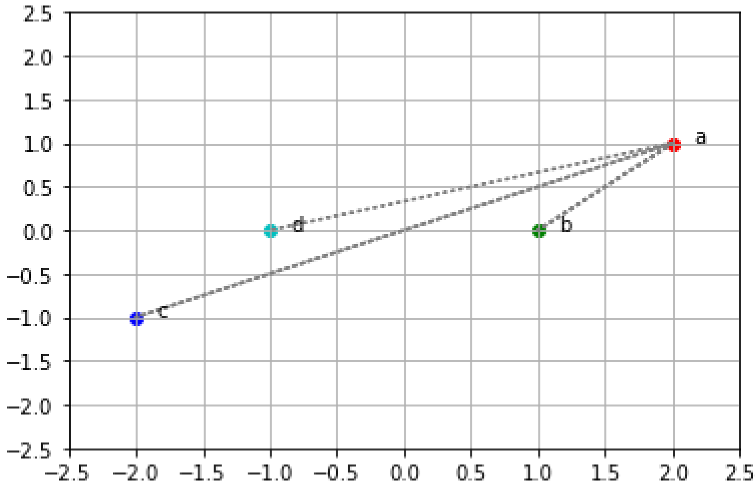
In [38]:
```
# 3x3 90 degree rotation matrix
R = np.array([[0, 1,0],
              [-1,0,0],
              [0, 0,1]])

print_transform(R, False)
```



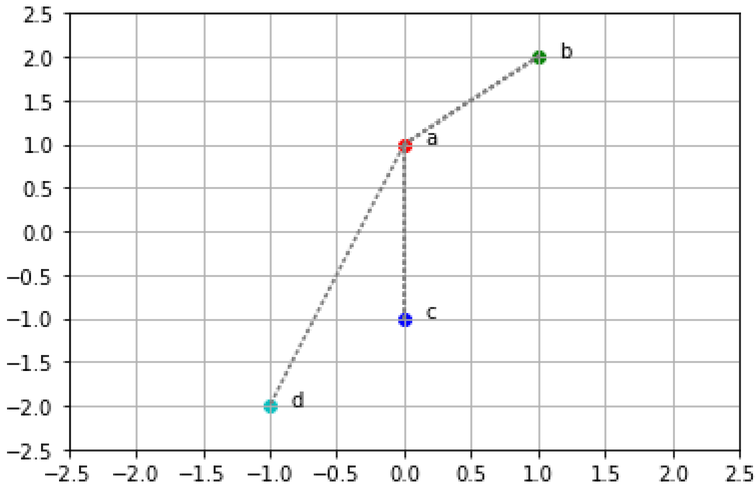
In [39]:
```
# 3x3 translation matrix
# shifts x by -2, y by 1
T = np.array([[1,0,-2],
              [0,1, 1],
              [0,0, 1]])

# translations are affected by the last item of vector, added override argument
# all points are 'b' and green because all vectors end with '1'
print_transform(T, True)
```



In [40]:
```
# 3x3 horizontal shear matrix
HS = np.array([[1,2,0],
               [0,1,0],
               [0,0,1]])

print_transform(HS, False)
```



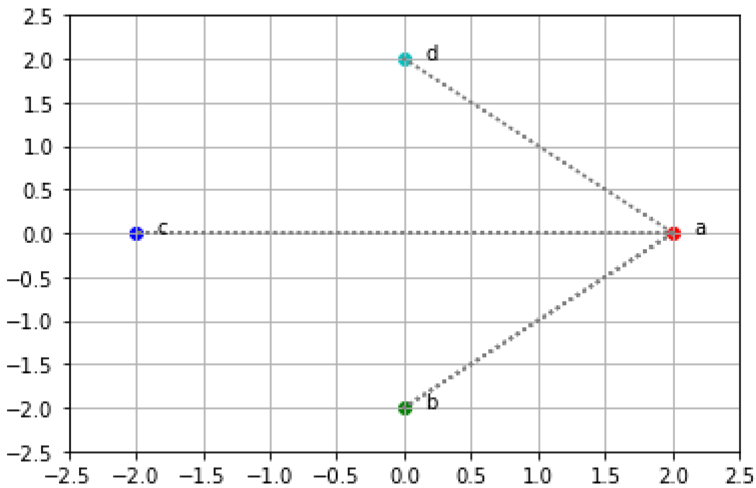
In [41]:
```
# 3x3 vertical shear matrix
VS = np.array([[1,0,0],
               [2,1,0],
               [0,0,1]])

print_transform(VS)
```



Question 3

In [49]:
```
def print_transforms(t_matrices):
    fig = plt.figure()
    ax = plt.gca()
    xs = []
    ys = []

    # multiplies matrices from right->left until 1 is left
    while len(t_matrices) > 1:
        t_matrices[-2] = np.matmul(t_matrices[-2], t_matrices[-1])
        del t_matrices[-1]

    for row in A:
        output_row = t_matrices[0] @ 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()

# rotate 90 degrees right followed by 2x scale up
ts = [S,R]
print_transforms(ts)
```



Exercise 2

Question 4

In [50]:

```
'''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 [51]:

```
from numpy import linalg

# either x6 or decrease tol value in rref function
L = np.array([[ -1, 1/3, 1/3, 1/2, 0],
              [1/2,  -1, 1/3,   0, 0],
              [1/2, 1/3,  -1, 1/2, 0],
              [  0, 1/3, 1/3,  -1, 0]])

print("Reduced L:\n", rref(6*L))
```

```
Reduced L:
 [[ 1.      0.      0.     -1.5    -0.    ]
 [ 0.      1.      0.     -1.3125 -0.    ]
 [ 0.      0.      1.     -1.6875 -0.    ]
 [ 0.      0.      0.      0.      0.    ]]
```

from the rref, we deduce that ra = 1.5rd, rb = 1.3125rd and rc = 1.6875rd

popularity vector r = [1.5, 1.3125, 1.6875, 1], setting rd = 1 as a free variable

we can confirm this by computing Lr = r and (L-I)r = 0 below

In [52]:

```
L = np.array([[ 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]])
I = np.identity(4)
r = np.stack([1.5, 1.3125, 1.6875, 1])

calc_r = np.matmul(L, r)
print("Calculated Lr:\n", calc_r, "\n")

calc_r = np.matmul((L-I), r)
print("Calculated (L-I)r:\n", calc_r)
```

```
Calculated Lr:
 [1.5    1.3125 1.6875 1.    ]

Calculated (L-I)r:
 [0. 0. 0. 0.]
```

Question 5

In [53]:

```
#                        A,     B,    C,   D,    E
L = 12 * np.array([[-1,   1/2, 1/4,   1, 1/3, 0],    # ra = 1/2rb + 1/4rc + rd + 1/3re
```

```
                    [1/3,  -1, 1/4,  0,   0, 0],   # rb = 1/3ra + 1/4rc
                    [1/3, 1/2,  -1,  0, 1/3, 0],   # rc = 1/3ra + 1/2rb + 1/3re
                    [1/3,   0, 1/4, -1, 1/3, 0],   # rd = 1/3ra + 1/4rc + 1/3re
                    [  0,   0, 1/4,  0,  -1, 0]])  # re = 1/4rc

rref_L = rref(L)

print("Reduced L:\n", rref_L)
```

```
Reduced L:
 [[ 1.          0.          0.          0.         -6.33333333 -0.        ]
 [ 0.          1.          0.          0.         -3.11111111 -0.        ]
 [ 0.          0.          1.          0.         -4.         -0.        ]
 [ 0.          0.          0.          1.         -3.44444444 -0.        ]
 [ 0.          0.          0.          0.          0.          0.        ]]
```

from the rref, ra = 19re/3, rb = 28re/9, rc = 4re, rd = 31re/9

popularity vector r = [19/3, 28/9, 4, 31/9, 1], setting re = 1 as a free variable

In [54]:

```
'''
# multiply both sides by (L-I)^-1
L_I = L - np.identity(5)
L_I_inv = np.linalg.inv(L_I)
zero = np.stack([0,0,0,0,0])
x = np.matmul(L_I_inv, zero)
x
'''

L = L = 12 * np.array([[-1,  1/2, 1/4,  1, 1/3],
                       [1/3,  -1, 1/4,  0,   0],
                       [1/3, 1/2,  -1,  0, 1/3],
                       [1/3,   0, 1/4, -1, 1/3],
                       [  0,   0, 1/4,  0,  -1]])
rref_L = rref(L)
rref_L_I = rref_L - np.identity(5)
rref_L_I
```

Out[54]:

```
array([[ 0.        ,  0.        ,  0.        ,  0.        , -6.33333333],
       [ 0.        ,  0.        ,  0.        ,  0.        , -3.11111111],
       [ 0.        ,  0.        ,  0.        ,  0.        , -4.        ],
       [ 0.        ,  0.        ,  0.        ,  0.        , -3.44444444],
       [ 0.        ,  0.        ,  0.        ,  0.        , -1.        ]])
```

Exercise 3

Question 6

In [55]:

```
# works with transpose x
x_t = np.array([0.75, 0.1, 0.1, 0.05]).T

#               daily change matrix
#                S,      I,     R,  D
P = np.array([[0.95,  0.04,  0,  0],
              [0.05,  0.85,  0,  0],
              [0,     0.10,  1,  0],
              [0,     0.01,  0,  1]])
# apply daily change to population
x_t1 = np.matmul(P, x_t)
print("Day after xt: ", x_t1)
```

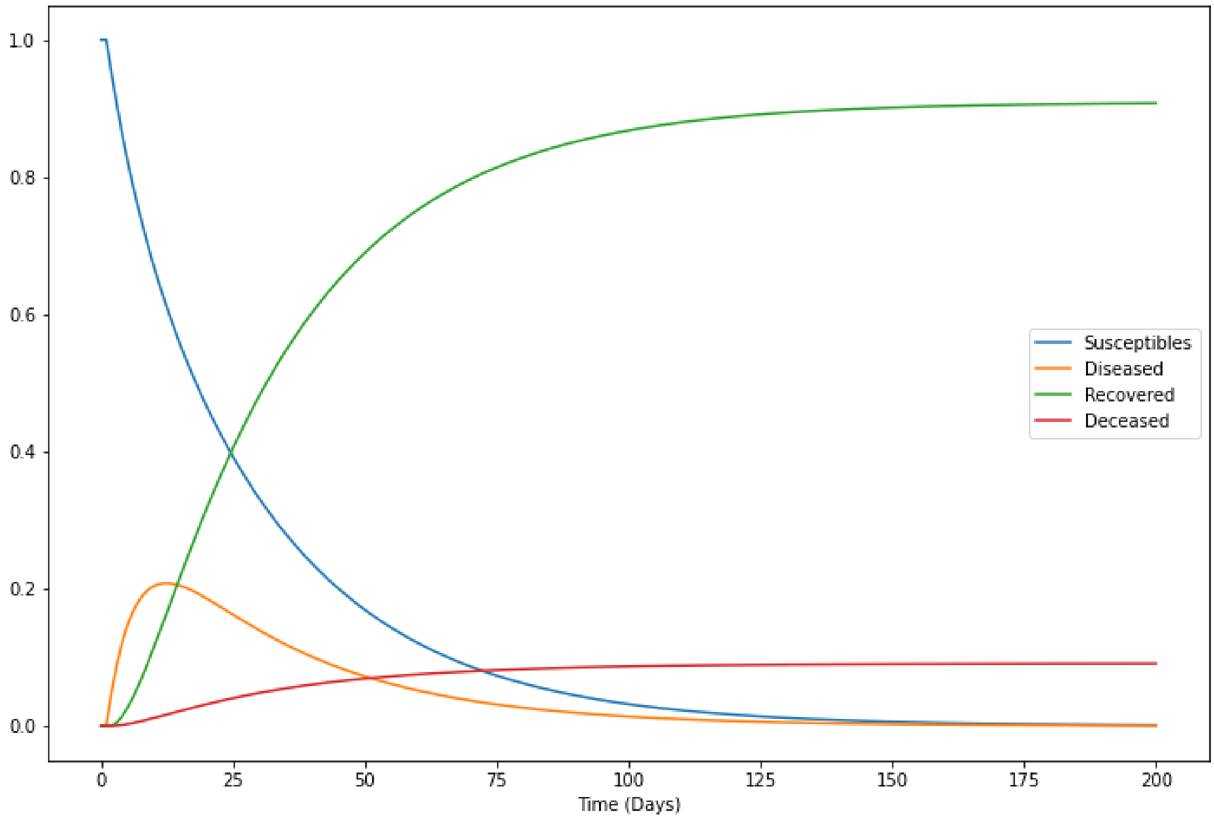
```
Day after xt:  [0.7165 0.1225 0.11   0.051 ]
```

Question 7

In [56]:
```
# array containing arrays of x_n
# init with x0 and x1
x = [[1,0,0,0],[1,0,0,0]]

# days 2-200
for i in range(2, 201):
    # applies day change with previous day to get current day
    x_i = np.matmul(P, x[i-1])
    x.append(x_i)

x = np.array(x)
print("Day 2: ", x[2], "\nDay 200: ", x[200])
```

```
Day 2:  [0.95 0.05 0.   0.  ]
Day 200:  [1.11700273e-03 4.77017115e-04 9.07641800e-01 9.07641800e-02]
```

In [57]:
```
f = plt.figure(figsize = (12, 8))

''' transpose so each row contains all values of each category
instead of data of 1 day '''
for line in x.T:
    plt.plot(line)

plt.legend(["Susceptibles", "Diseased", "Recovered", "Deceased"])
plt.xlabel("Time (Days)")
plt.show()
```



In [ ]: