Train a single perceptron and SVM to learn an AND gate with two inputs x1 and x2. Assume that all the weights of the perceptron are initialized as 0. Show the calculation for each step and also draw the decision boundary for each updation.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
from sympy.plotting import plot_implicit
from sympy import symbols
```

In [ ]:

```
# Append 1 to vector and make it -ve for one class (say 0)

def append_vector(df,class_1,class_2,pos_to_append): #assumption class label is last co
lumn , two features
    ext_vect=[]
    for _,row in df.iterrows():
        if row[df.columns[-1]]==int(class_1):
            ext_vect.append(-1)
            row[df.columns[0]]=-1* row[df.columns[0]]
            row[df.columns[1]]=-1* row[df.columns[1]]
        else:
            ext_vect.append(1)

    df.insert(pos_to_append,"bias",ext_vect,True)
```

In [ ]:

```
def move_sympyplot_to_axes(p, ax):
    backend = p.backend(p)
    backend.ax = ax
    backend._process_series(backend.parent._series, ax, backend.parent)
    backend.ax.spines['right'].set_color('none')
    backend.ax.spines['bottom'].set_position('zero')
    backend.ax.spines['top'].set_color('none')
    plt.close(backend.fig)
```

In [ ]:

```
def plot_eqn_and_points(df_train,weights): #works only for 2-feature # plot the origina
l points
#     df_train.plot.scatter(x=df_train.columns[0],y=df_train.columns[1])

    y,x = symbols('y x')
    eqn=y*weights[0]+x*weights[1]+weights[2]
    graph=plot_implicit(eqn, (x, -3, 3), (y, -3, 3),show=False)
    fig, ax = plt.subplots()

    move_sympyplot_to_axes(graph, ax)
    plt.setp(ax.yaxis.get_label(), 'rotation', 0)


    plt.scatter(df_train[df_train.columns[0]], df_train[df_train.columns[1]],c=df_train
[df_train.columns[-1]])
    plt.show()
#     plt.show()
#     plt.scatter(df_versicolor['petal.length'], df_versicolor['petal.width'],label="Ve
rsicolor")
```

In [ ]:

```
def perceptron_iter(df_train,learning_rate=1,weights=None):
    n=len(df_train)
    k=0
    count=0
    all_weights=[]
    if weights == None:
        weights=np.zeros(df_train.shape[1])
    weights=np.array(weights)
    while count != n:
        if weights.T @ np.array(df_train.iloc[k]) > 0:
            count+=1
        else:
            count=0
            weights += learning_rate * df_train.iloc[k]
            count+=1

        k=(k+1)%n
        print(weights)
        all_weights.append(weights.copy())
        # new_list = old_list.copy()
        # plot_eqn_and_points(df_train,weights)
    return weights,all_weights
```

In [ ]:

```
df_list=[[0,0,0],[0,1,0],[1,0,0],[1,1,1]]
```

In [ ]:

```
df_org=pd.DataFrame(df_list,columns=['x1','x2','output'])
```

In [ ]:

```
df=pd.DataFrame(df_list,columns=['x1','x2','output'])
```

In [ ]:

```
df
```

Out[ ]:

	x1	x2	output
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

In [ ]:

```
append_vector(df,0,1,2)
```

In [ ]:

```
df
```

Out[ ]:

	x1	x2	bias	output
0	0	0	-1	0
1	0	-1	-1	0
2	-1	0	-1	0
3	1	1	1	1

In [ ]:

```
df_train=df[df.columns[:-1]]
```

In [ ]:

```
df_train
```

Out[ ]:

	x1	x2	bias
0	0	0	-1
1	0	-1	-1
2	-1	0	-1
3	1	1	1

In [ ]:

```
w,w_a=perceptron_iter(df_train,1,weights=None)
```

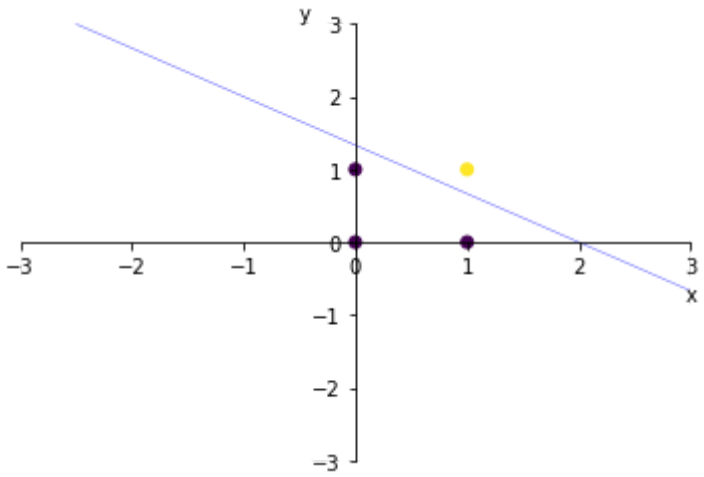
```
x1      0.0
x2      0.0
bias   -1.0
Name: 0, dtype: float64
x1      0.0
x2      0.0
bias   -1.0
Name: 0, dtype: float64
x1      0.0
x2      0.0
bias   -1.0
Name: 0, dtype: float64
x1      1.0
x2      1.0
bias    0.0
Name: 0, dtype: float64
x1      1.0
x2      1.0
bias   -1.0
Name: 0, dtype: float64
x1      1.0
x2      0.0
bias   -2.0
Name: 0, dtype: float64
x1      1.0
x2      0.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -1.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -1.0
Name: 0, dtype: float64
x1      2.0
x2      0.0
bias   -2.0
Name: 0, dtype: float64
x1      1.0
x2      0.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -2.0
Name: 0, dtype: float64
x1      1.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
```

```
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      3.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      3.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      3.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
```

```
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
```

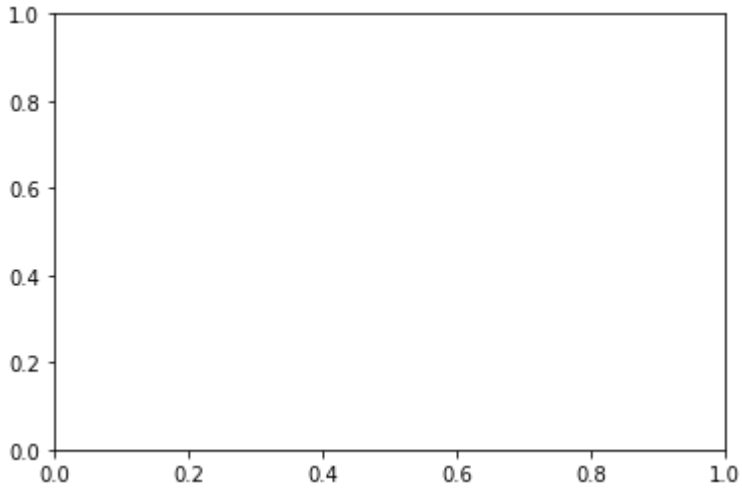
In [ ]:

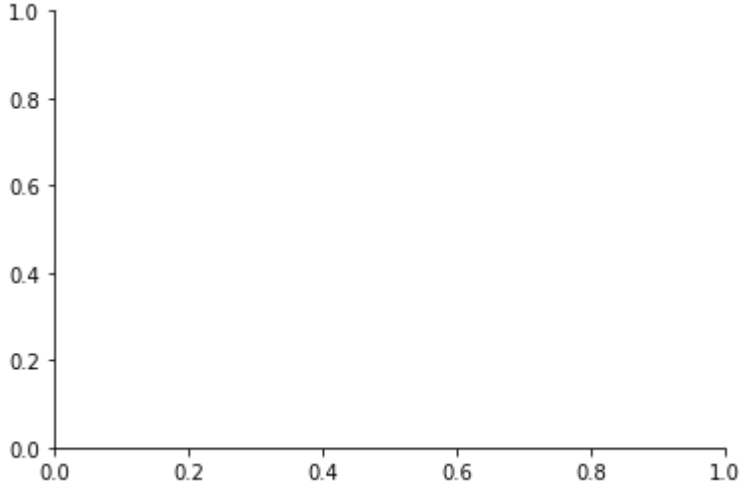
```
plot_eqn_and_points(df_org,w)
```

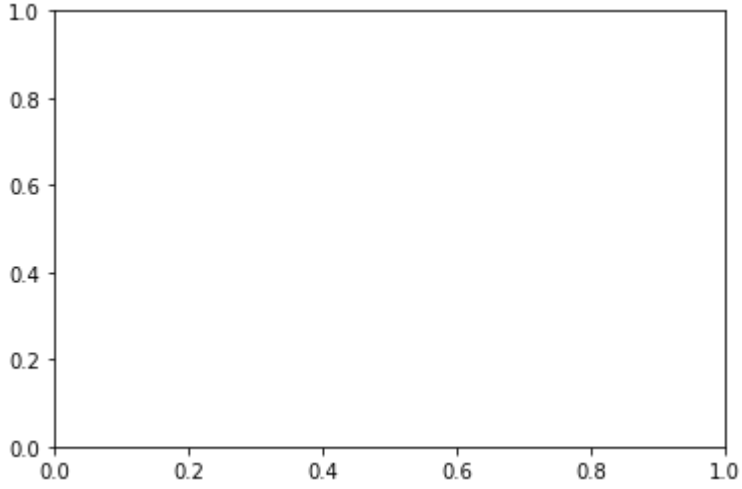


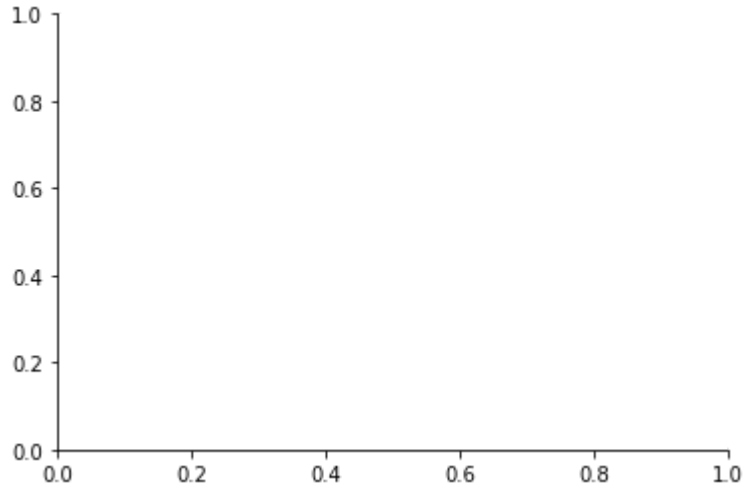
In [ ]:

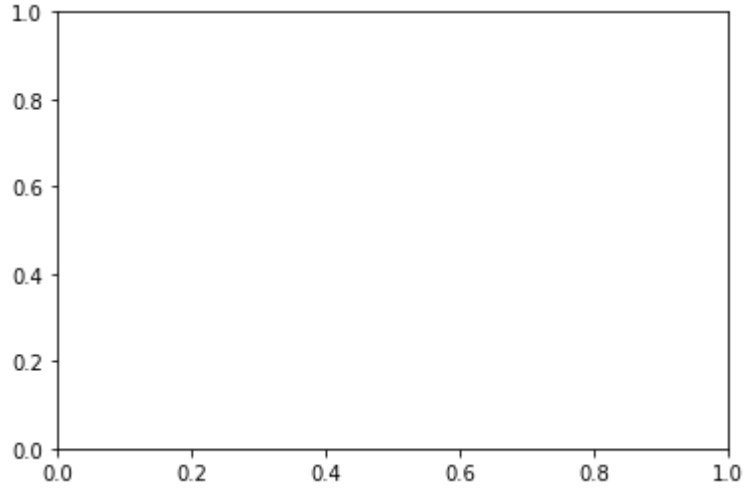
```
for i in w_a:
  try:
    plot_eqn_and_points(df_org,i)
  except:
    pass
```

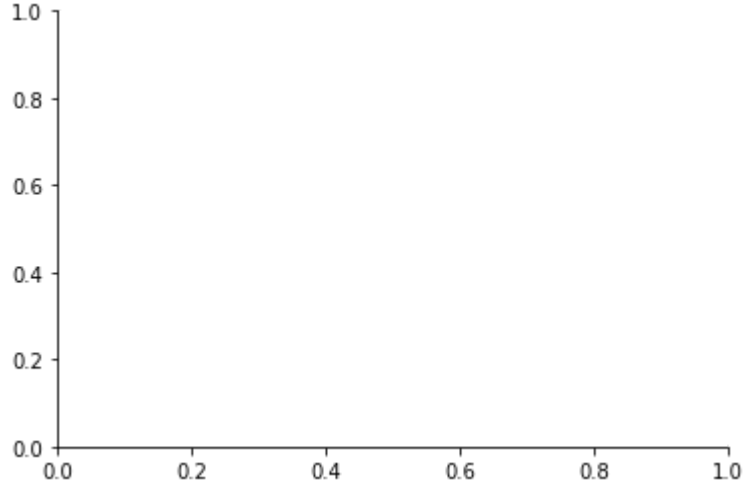

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

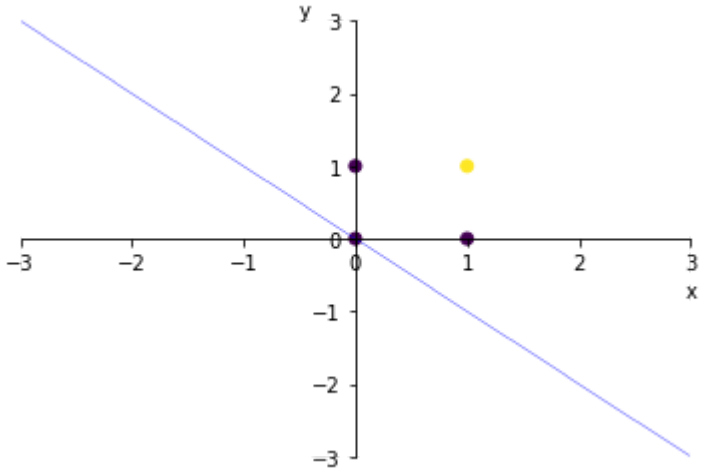

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

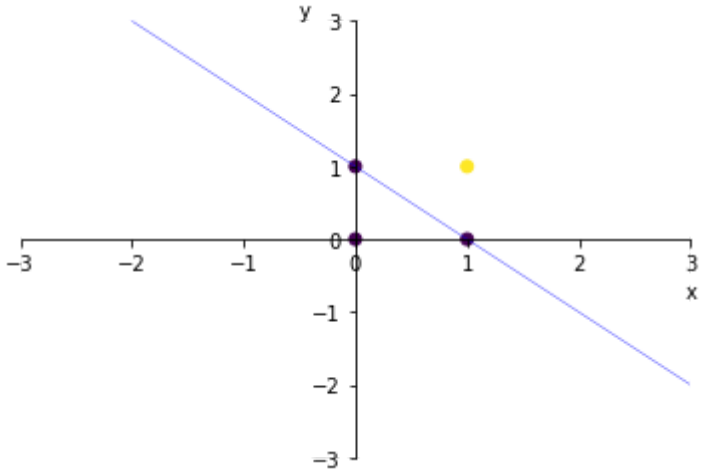

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

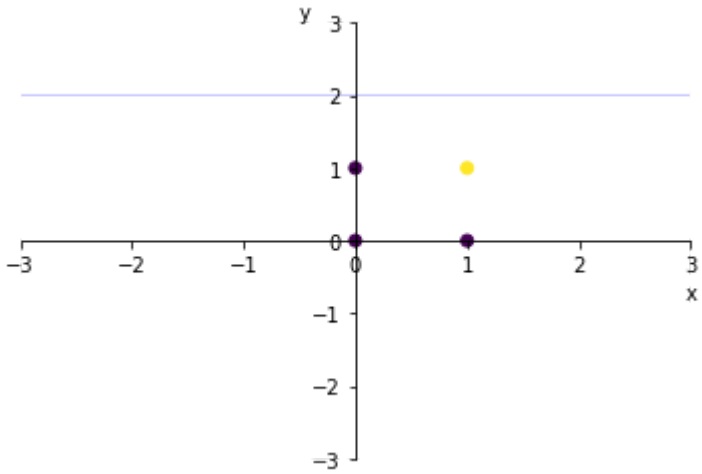

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

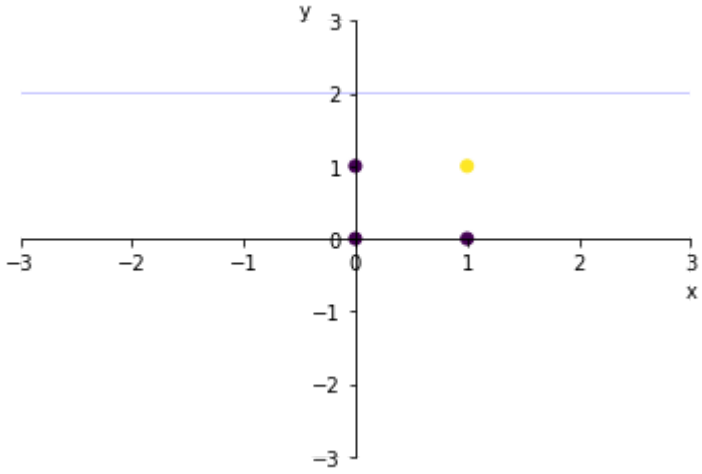

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

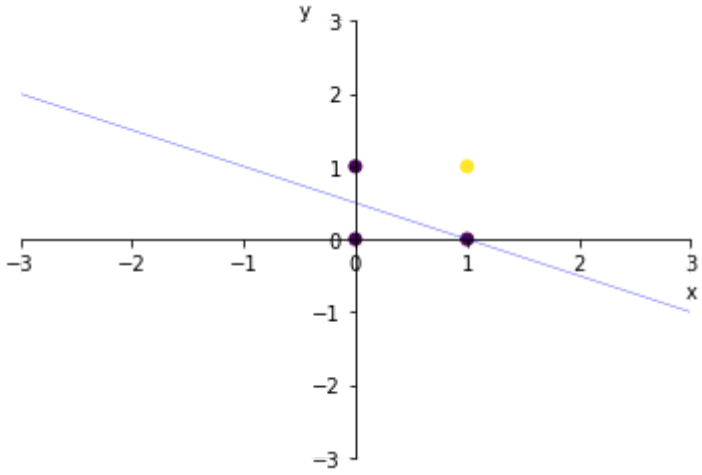

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

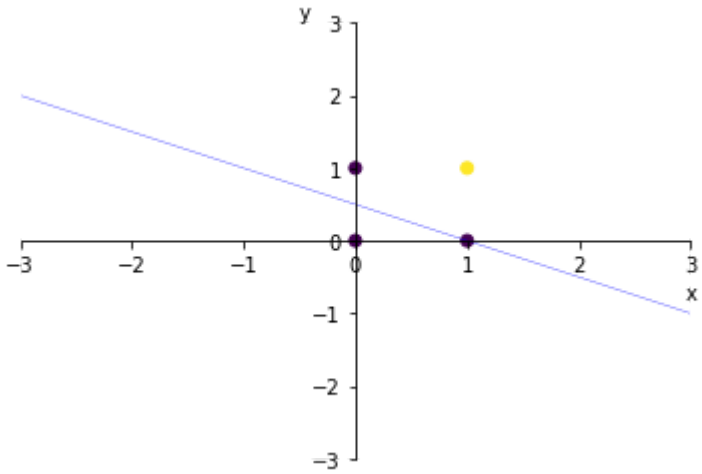

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

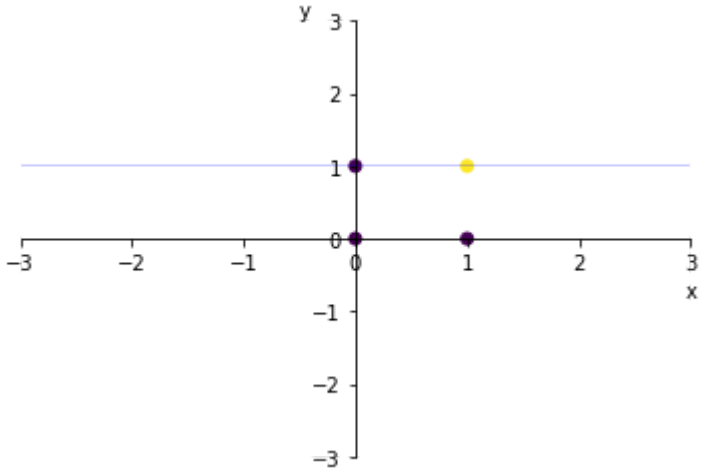

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

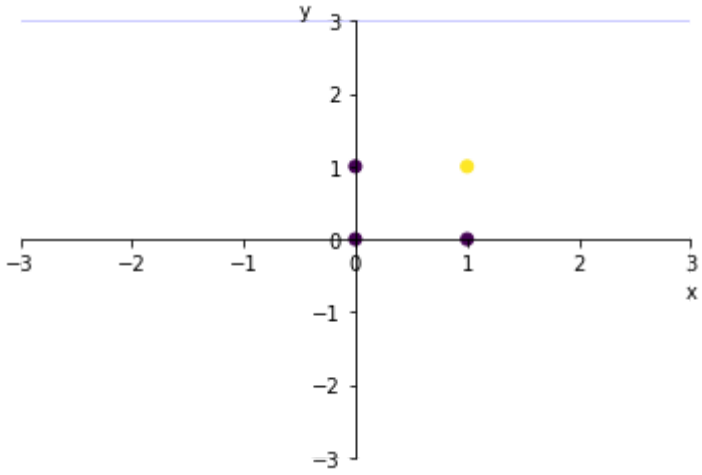

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

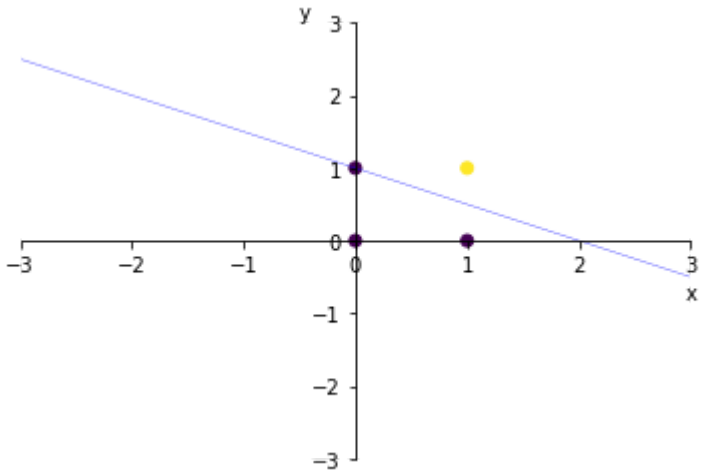

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

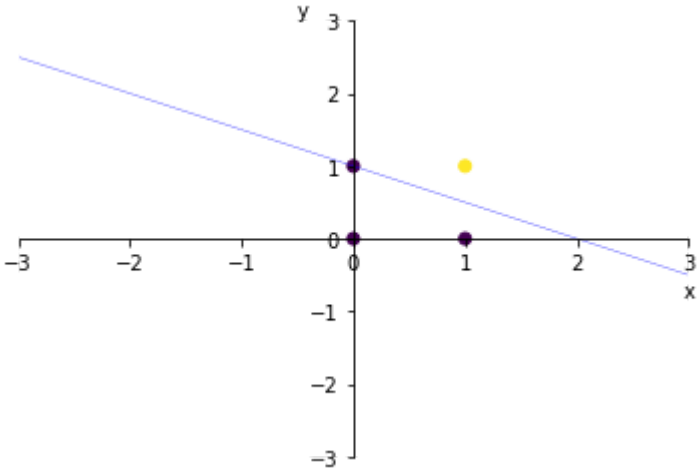

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

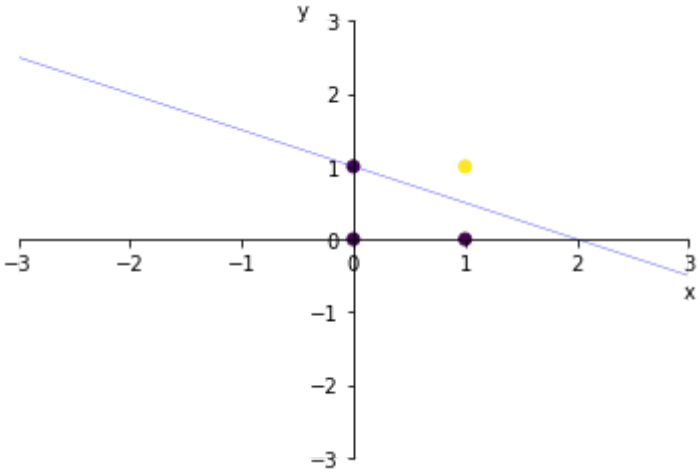

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

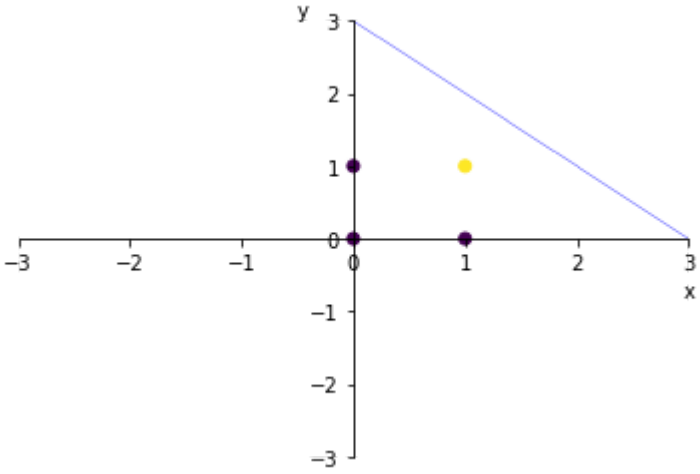

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

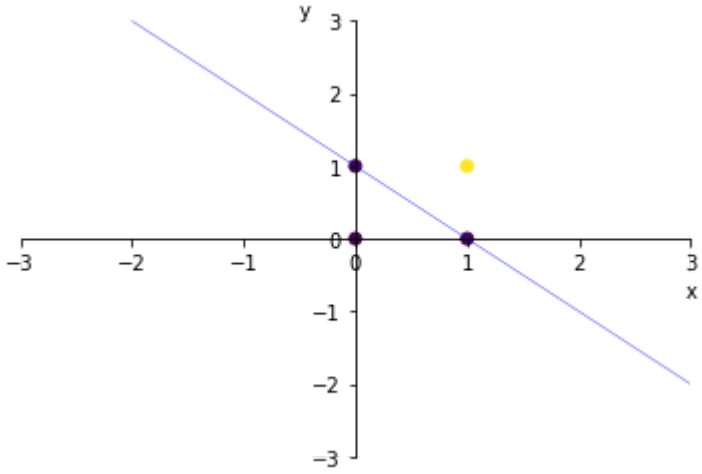

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

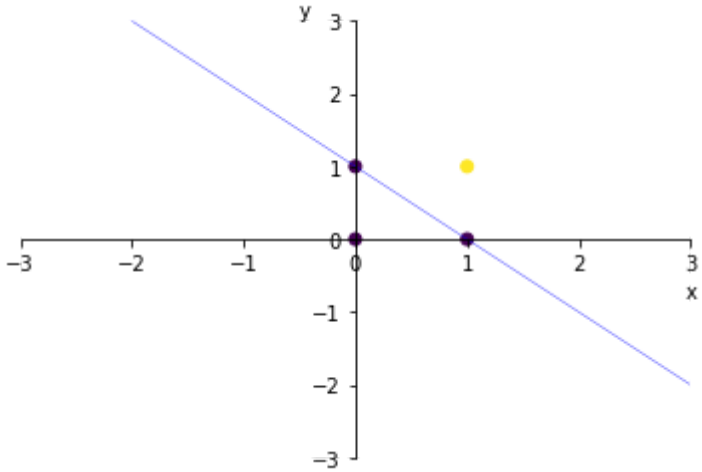

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

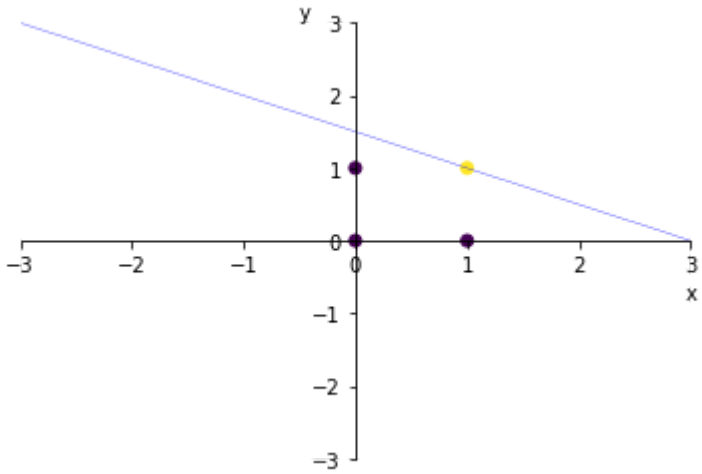

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

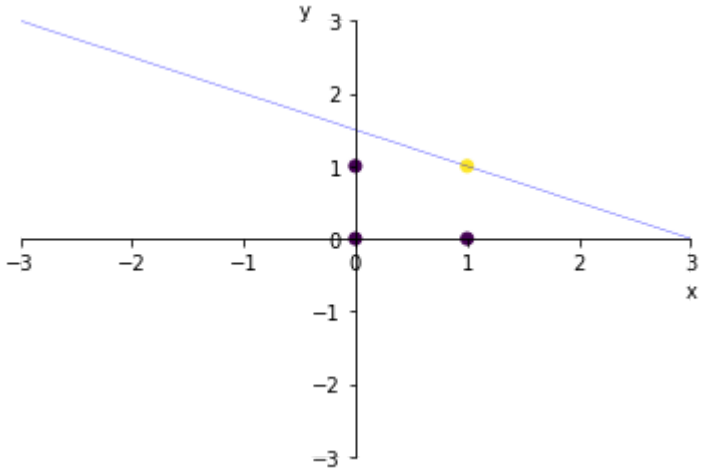

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

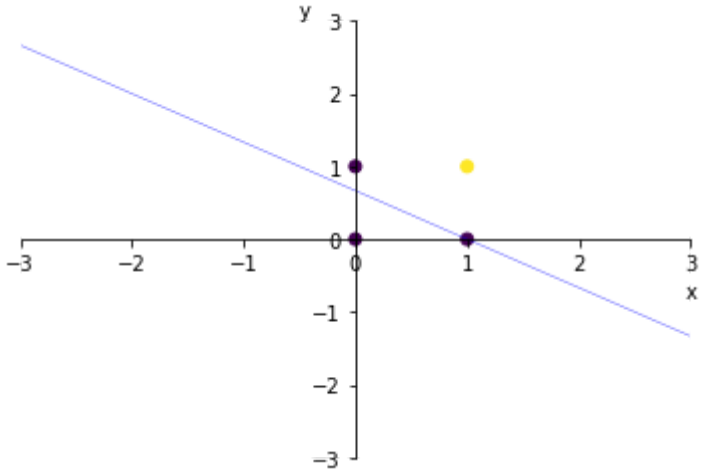

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

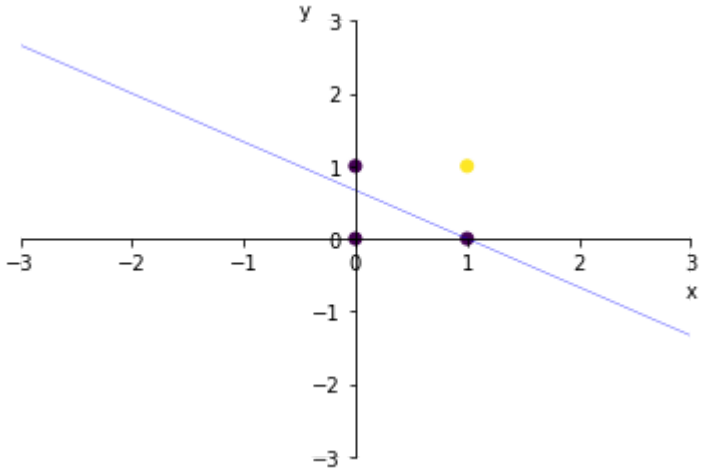

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

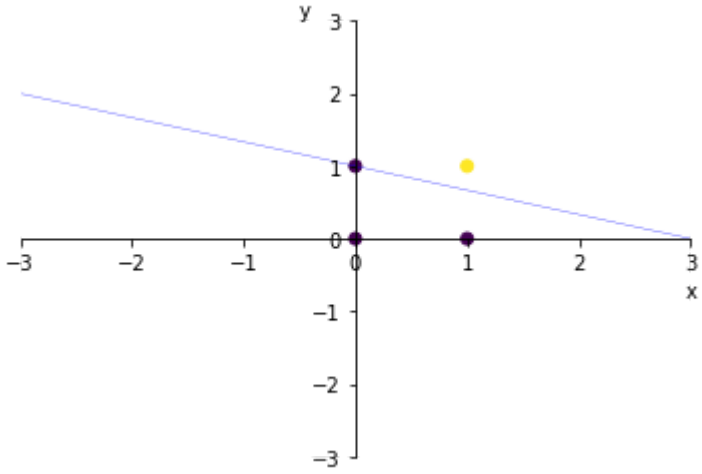

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

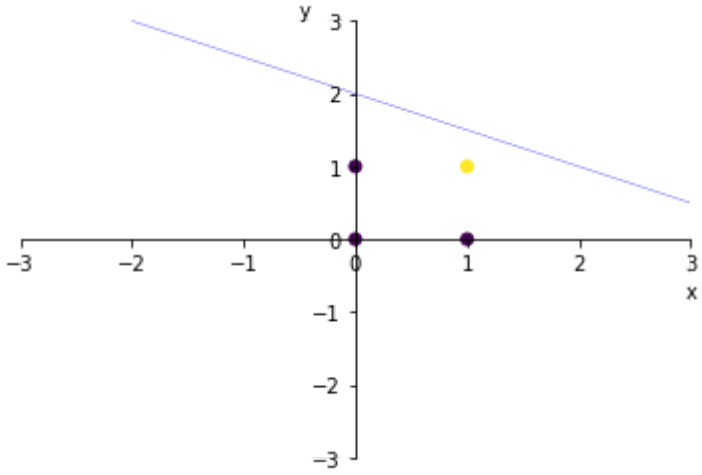

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

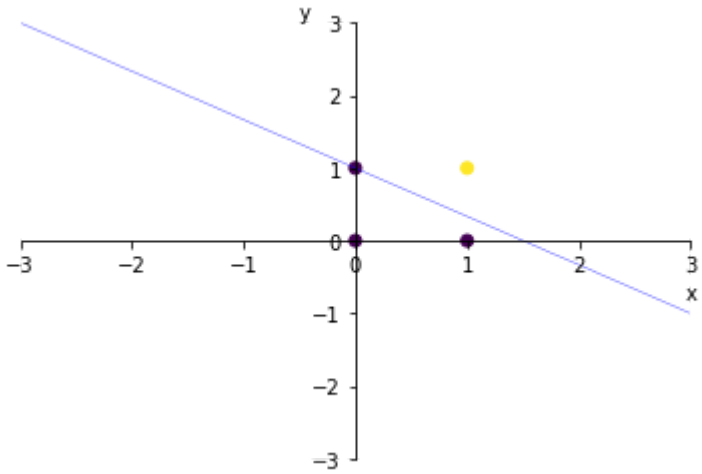

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

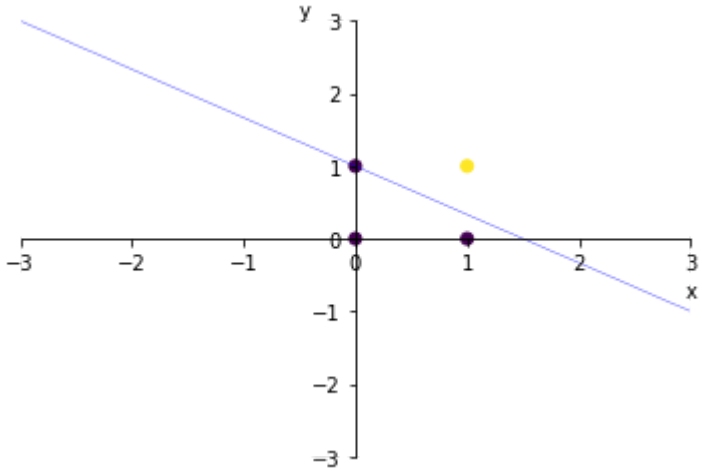

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

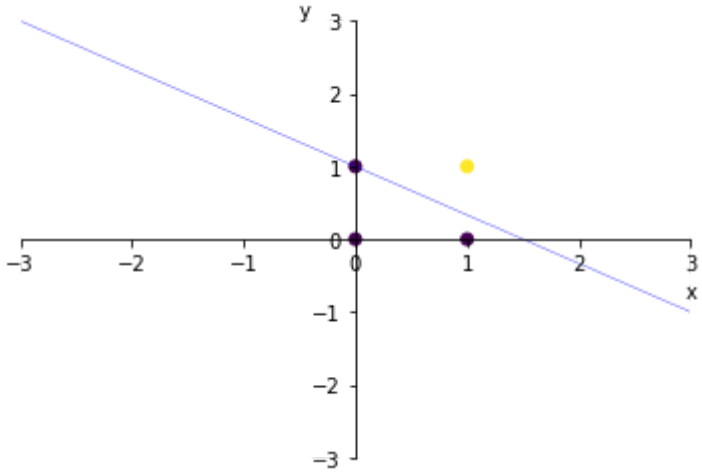

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

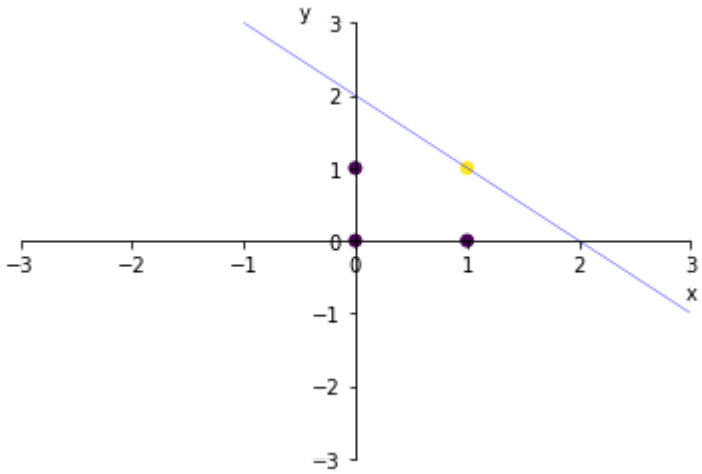

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

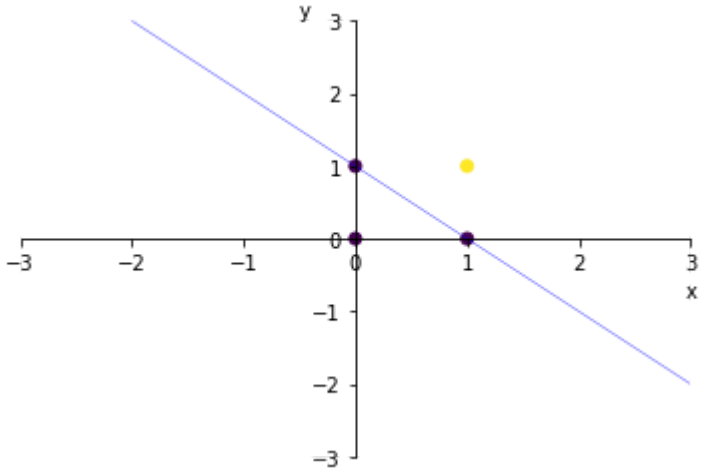

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

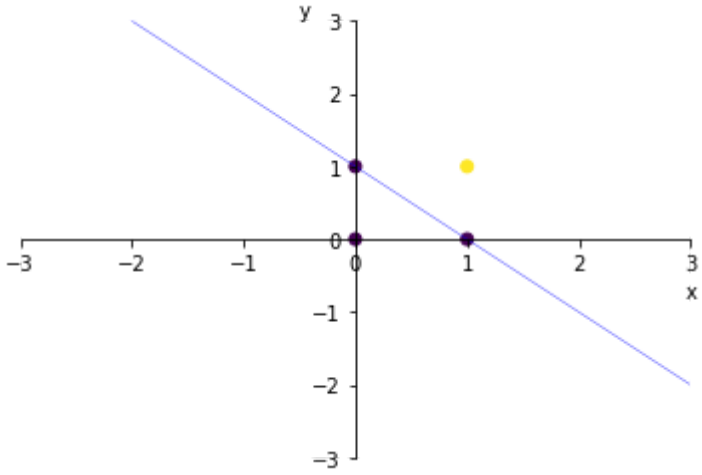

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

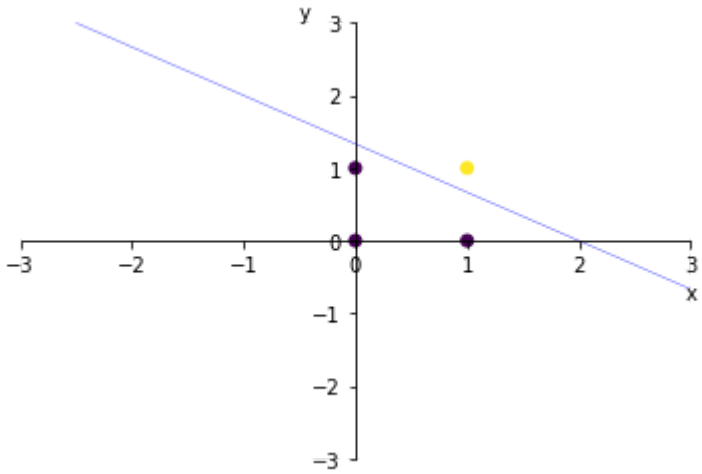

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

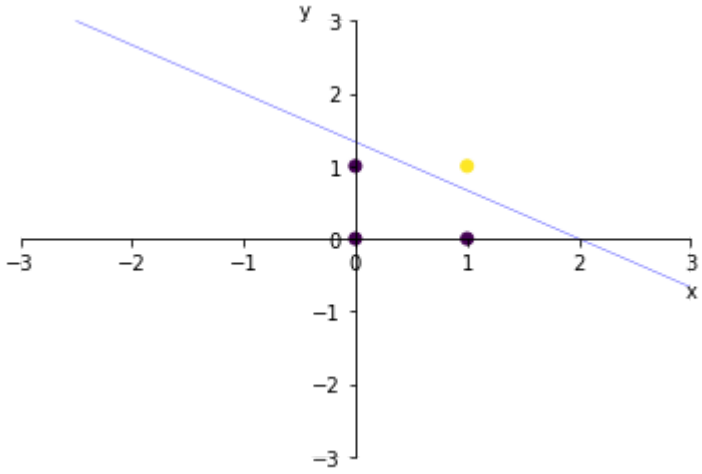

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

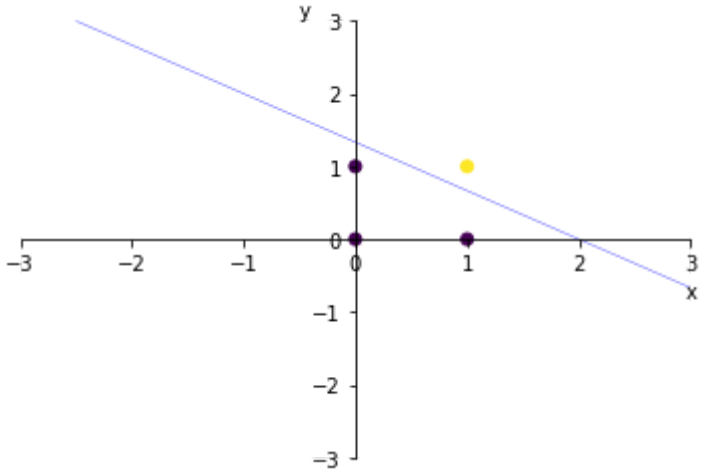

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

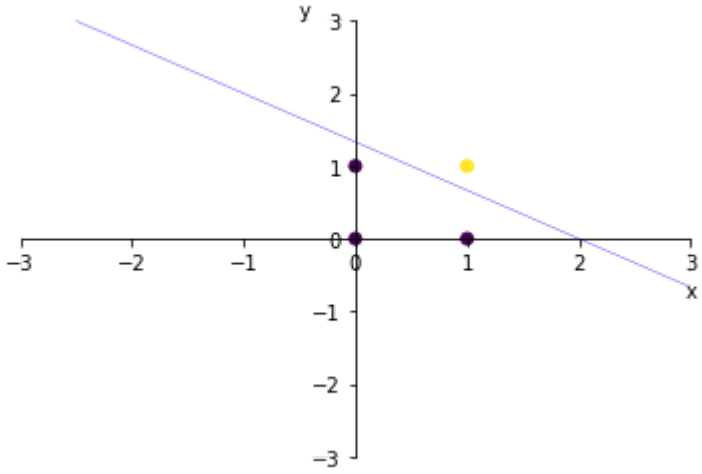

y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

In [ ]:

```
w
```

Out[ ]:

```
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
```