Linear Regression

In [157]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [158]:

```
iris = load_iris()
```

In [159]:

```
df = pd.DataFrame(iris.data, columns = iris.feature_names)
```

In [160]:

```
df.head()
```

Out[160]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [161]:

```
X = df['petal length (cm)'].values.reshape(-1,1)
y = df['petal width (cm)']
```

In [162]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

In [163]:

```
print(X_train.shape,X_test.shape)
```

```
(120, 1) (30, 1)
```

In [164]:

```
lr = LinearRegression()
```

In [165]:

```
lr.fit(X_train,y_train)
```

Out[165]:

```
▾ LinearRegression
LinearRegression()
```

In [166]:

```
lr.predict(X_test)
```

Out[166]:

```
array([1.74413389, 1.74413389, 1.99204765, 1.3309443 , 1.86809077,
       1.08303055, 1.99204765, 1.70281493, 0.33928928, 0.25665136,
       1.57885806, 1.62017701, 1.95072869, 0.33928928, 1.95072869,
       1.3309443 , 0.25665136, 0.17401344, 1.90940973, 0.09137552,
       0.25665136, 0.25665136, 1.70281493, 0.33928928, 2.48787516,
       1.57885806, 1.16566846, 0.2153324 , 2.28128036, 2.40523724])
```

In [167]:

```
import numpy as np

input = np.array([[1.4]]).reshape(-1,1)
lr.predict(input)
```

Out[167]:

```
array([0.2153324])
```

In [167]:

```

```

Logistic Regression

In [168]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [169]:

```
iris = load_iris()
```

In [170]:

```
df = pd.DataFrame(iris.data, columns = iris.feature_names)
```

In [171]:

```
df.head()
```

Out[171]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [172]:

```
df['target'] = iris.target
```

In [173]:

```
df.head()
```

Out[173]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [174]:

```
X = df.drop('target',axis=1)
y = df['target']
```

In [175]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

In [176]:

```
print(X_train.shape,X_test.shape)
```

```
(120, 4) (30, 4)
```

In [177]:

```
lr = LogisticRegression()
```

In [178]:

```
lr.fit(X_train,y_train)
```

Out[178]:

```
▾ LogisticRegression
LogisticRegression()
```

In [179]:

```
lr.predict(X_test)
```

Out[179]:

```
array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 2, 1, 2, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 2, 0, 2, 1])
```

In [180]:

```
import numpy as np

input = np.array([[5.1, 3.5, 1.4, 0.2 ]]).reshape(-1,4)
lr.predict(input)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have
valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

Out[180]:

```
array([0])
```

In [180]:

```

```

SVM

In [181]:

In [181]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

In [182]:

```
iris = load_iris()
```

In [183]:

```
df = pd.DataFrame(iris.data, columns = iris.feature_names)
```

In [184]:

```
df.head()
```

Out[184]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [185]:

```
df['target'] = iris.target
```

In [186]:

```
df.head()
```

Out[186]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [187]:

```
X = df.drop('target',axis=1)
y = df['target']
```

In [188]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

In [189]:

```
print(X_train.shape,X_test.shape)
```

```
(120, 4) (30, 4)
```

In [190]:

```
svm = SVC()
```

In [191]:

```
svm.fit(X_train,y_train)
```

Out[191]:

```
▾ SVC
SVC()
```

In [192]:

```
svm.predict(X_test)
```

Out[192]:

```
array([2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 1, 0, 0, 2, 1, 1, 1, 1, 2, 1, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1])
```

In [193]:

```
import numpy as np

input = np.array([[5.1, 3.5, 1.4, 0.2 ]]).reshape(-1,4)
svm.predict(input)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have
valid feature names, but SVC was fitted with feature names
  warnings.warn(
```

Out[193]:

```
array([0])
```

In [193]:

```

```

Hebbian Learning

In [194]:

```
def hebbian(sample):

  #step 1 = weights and bias = 0
  w1,w2,b = 0,0,0
  print('original weights')
  print(f"({w1:2},{w2:2},{b:2})")
  #step 2 = looping the formulas
  for x1,x2,y in sample:
    w1 = w1 + x1*y # w1(new) = w1(old) + x1*y
    w2 = w2 + x2*y
    b = b +y # b(new) = b(old) + y

    print(f'({x1:2},{x2:2}){y:2}|({x1*y:2},{x2*y:2},{y:2})|({w1:2},{w2:2},{b:2})')
    print()
    print()

sample = {
    'binary_input':[
        [1,1,1],
        [1,0,0],
        [0,1,0],
        [0,0,0]
    ],
    'input_binary_output_bipolar':[
        [1,1,1],
        [1,0,-1],
        [0,1,-1],
```

```
        [0,0,-1]
    ],
    'bipolar_input':[
        [1,1,1],
        [1,-1,-1],
        [-1,1,-1],
        [-1,-1,-1]
    ]
}

print("sample with binary input")
hebbian(sample['binary_input'])
print("sample with binary input bipolar output")
hebbian(sample['input_binary_output_bipolar'])
print("sample with bipolar input")
hebbian(sample['bipolar_input'])
```

```
sample with binary input
original weights
( 0, 0, 0)
( 1, 1) 1|( 1, 1, 1)|( 1, 1, 1)


( 1, 0) 0|( 0, 0, 0)|( 1, 1, 1)


( 0, 1) 0|( 0, 0, 0)|( 1, 1, 1)


( 0, 0) 0|( 0, 0, 0)|( 1, 1, 1)


sample with binary input bipolar output
original weights
( 0, 0, 0)
( 1, 1) 1|( 1, 1, 1)|( 1, 1, 1)


( 1, 0)-1|(-1, 0,-1)|( 0, 1, 0)


( 0, 1)-1|( 0,-1,-1)|( 0, 0,-1)


( 0, 0)-1|( 0, 0,-1)|( 0, 0,-2)


sample with bipolar input
original weights
( 0, 0, 0)
( 1, 1) 1|( 1, 1, 1)|( 1, 1, 1)


( 1,-1)-1|(-1, 1,-1)|( 0, 2, 0)


(-1, 1)-1|( 1,-1,-1)|( 1, 1,-1)


(-1,-1)-1|( 1, 1,-1)|( 2, 2,-2)
```

McCulloch pitts Algorithm

In [195]:

```
import numpy as np
```

In [196]:

```
matrix = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])
print(matrix)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

In [197]:

```
weight = np.array([1,1])
```

In [198]:

```
dot_product = matrix @ weight
print(dot_product)
```

```
[0 1 1 2]
```

In [199]:

```
def fire(dot: int, T: float):
  if dot >= T:
    return 1
  else:
    return 0
```

In [200]:

```
T = 2
#And
for i in range(4):
  activation = fire(dot_product[i],T)
  print(activation)

print()
# or
T=1
for i in range(4):
  activation = fire(dot_product[i],T)
  print(activation)
```

```
0
0
0
1

0
1
1
1
```

In [201]:

```
weight = np.array([-1,-1])
dot_product = matrix @ weight
print(dot_product)
T = 0
for i in range(4):
  activation = fire(dot_product[i],T)
  print(activation)
```

```
[ 0 -1 -1 -2]
1
0
0
```

Expectation-Maximization Algorithm

In [202]:

```
import numpy as np
# from  numpy.linalg import inv
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
```

In [203]:

```
m1 = [1,1]
m2 = [7,7]
cov1 = [[4,2],[3,4]]
cov2 = [[1,-1],[-1,2]]

x = np.random.multivariate_normal(m1,cov1,size=(200,))
y = np.random.multivariate_normal(m2,cov2,size=(200,))
d = np.concatenate((x,y),axis=0)
```

```
<ipython-input-203-3610678144bf>:6: RuntimeWarning: covariance is not symmetric positive-semidefinite.
  x = np.random.multivariate_normal(m1,cov1,size=(200,))
```

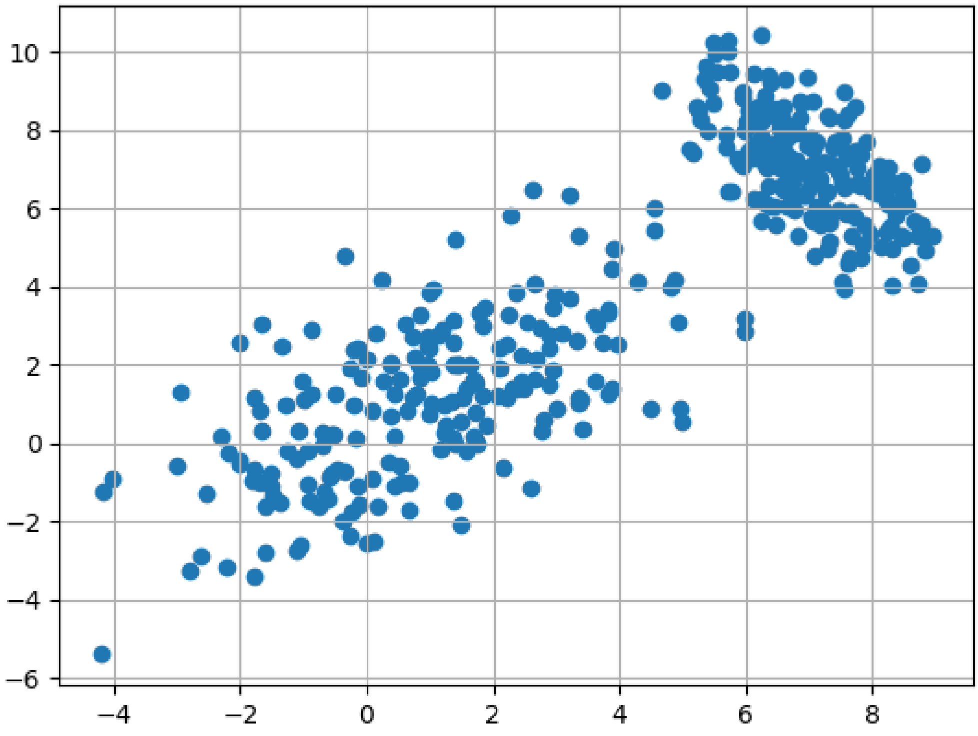
In [204]:

```
# print(x[0])
# print(y[0])
# print(d[0])
# print(d[200])
```

```
[1.25982344 0.45277717]
[7.05866964 8.74298645]
[1.25982344 0.45277717]
[7.05866964 8.74298645]
```

In [205]:

```
# ground truth plot
plt.scatter(d[:,0],d[:,1], marker='o')
plt.grid()
```

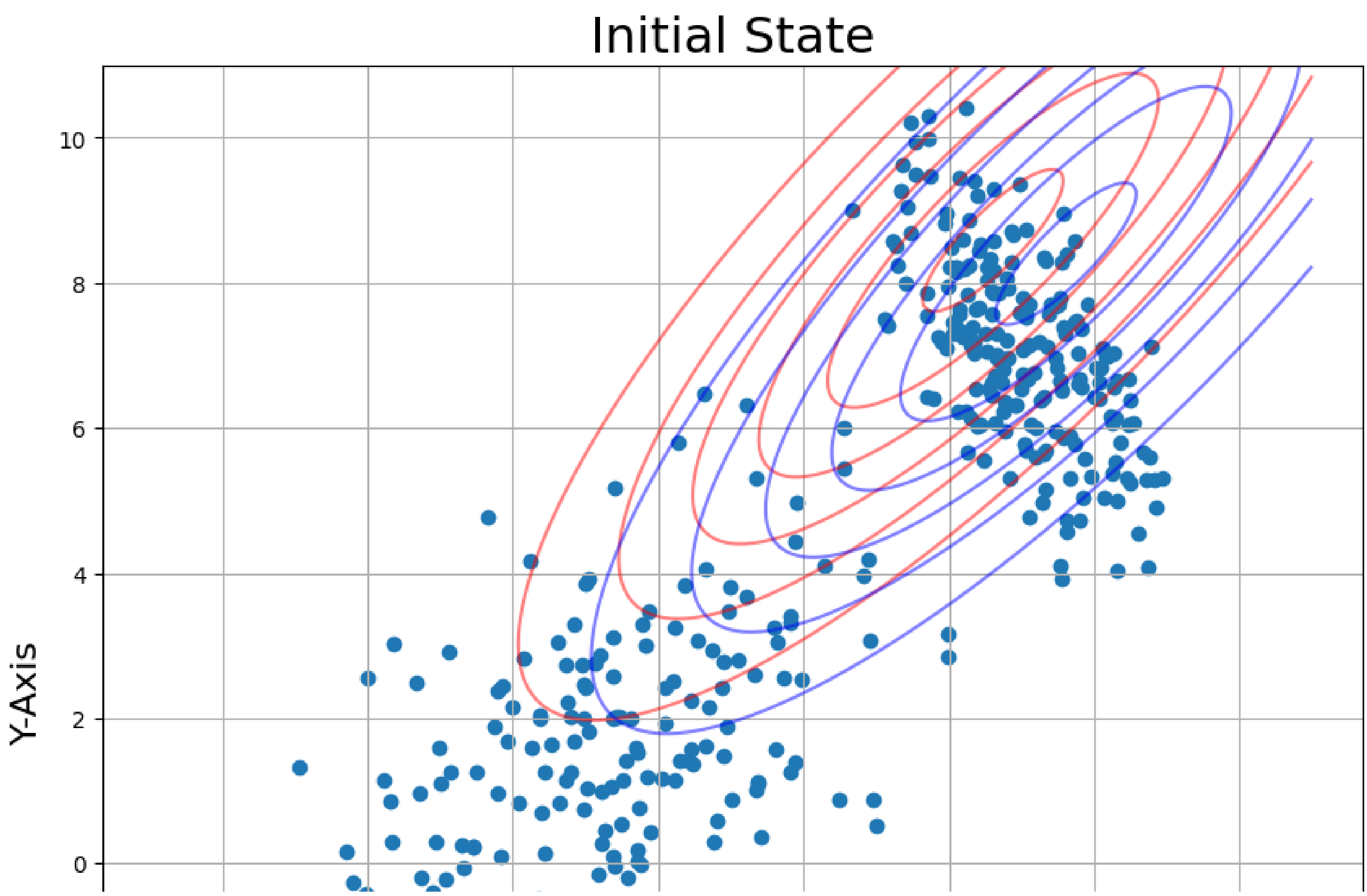


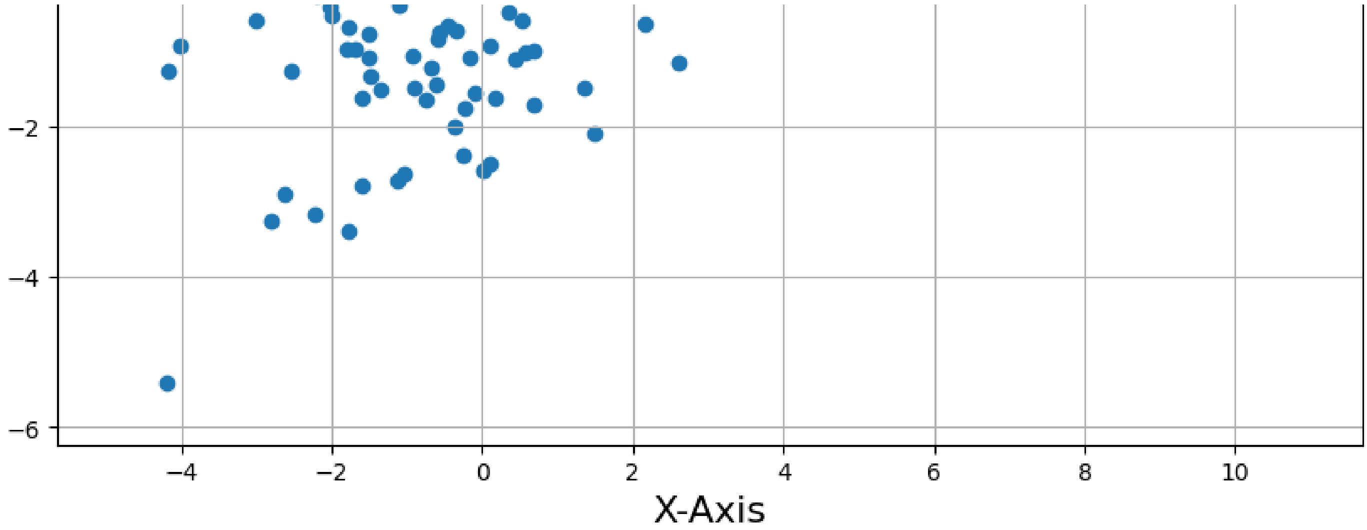
In [206]:

```
# print(np.transpose(d))
```

In [207]:

```
import random
m1 = random.choice(d)
m2 = random.choice(d)
cov1 = np.cov(np.transpose(d))
cov2 = np.cov(np.transpose(d))
pi = 0.5

#Plotting Initial State
x1 = np.linspace(-4,11,200)
x2 = np.linspace(-4,11,200)
X, Y = np.meshgrid(x1,x2)

Z1 = multivariate_normal(m1, cov1)
Z2 = multivariate_normal(m2, cov2)

pos = np.empty(X.shape + (2,))                # a new array of given shape and type, wit
hout initializing entries
pos[:, :, 0] = X; pos[:, :, 1] = Y

plt.figure(figsize=(10,10))                                                          #
creating the figure and assigning the size
plt.scatter(d[:,0], d[:,1], marker='o')
plt.contour(X, Y, Z1.pdf(pos), colors="r" ,alpha = 0.5)
plt.contour(X, Y, Z2.pdf(pos), colors="b" ,alpha = 0.5)
plt.axis('equal')                                                                  # ma
king both the axis equal
plt.xlabel('X-Axis', fontsize=16)                                                  # X-
Axis
plt.ylabel('Y-Axis', fontsize=16)                                                  # Y-
Axis
plt.title('Initial State', fontsize=22)                                            # Tit
le of the plot
plt.grid()                                                                         # di
splaying gridlines
plt.show()
```





In [208]:

```
#Expectation Step
##Expectation step
def Estep(lis1):
    m1=lis1[0]
    m2=lis1[1]
    cov1=lis1[2]
    cov2=lis1[3]
    pi=lis1[4]

    pt2 = multivariate_normal.pdf(d, mean=m2, cov=cov2)# probabilities
    pt1 = multivariate_normal.pdf(d, mean=m1, cov=cov1)# probabilities
    w1 = pi * pt2# weighted probabilities
    w2 = (1-pi) * pt1
    eval1 = w1/(w1+w2)# represents the probability of culster for each datapoint

    return(eval1)
```

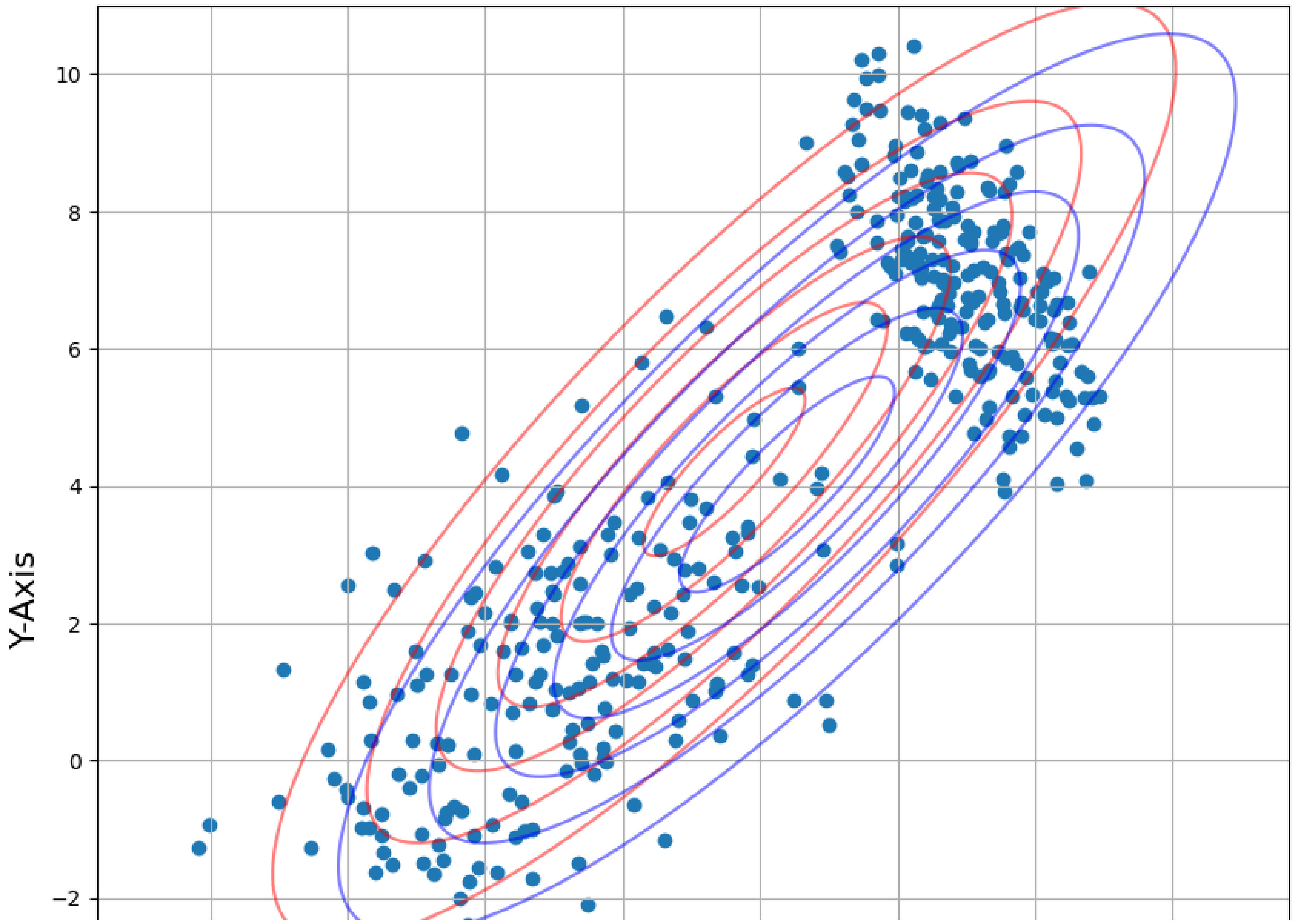
In [209]:

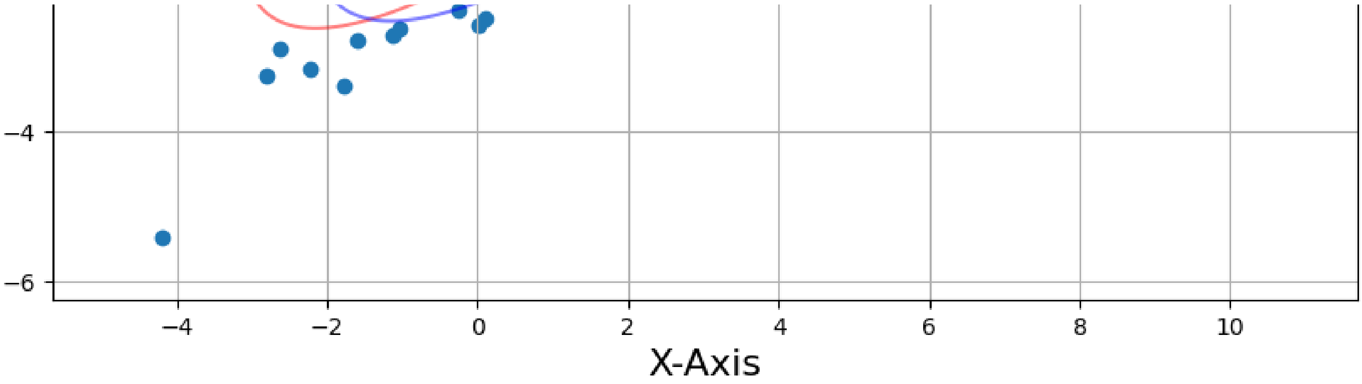
```
## Maximization step
def Mstep(eval1):
    num_mu1,din_mu1,num_mu2,din_mu2=0,0,0,0# weighted_sum, responsibilities

    for i in range(0,len(d)):
        num_mu1 += (1-eval1[i]) * d[i]# update the weighted sum
        din_mu1 += (1-eval1[i])# update the responsibilities

        num_mu2 += eval1[i] * d[i]
        din_mu2 += eval1[i]

    mu1 = num_mu1/din_mu1# for getting new clusters
    mu2 = num_mu2/din_mu2# //

    num_s1,din_s1,num_s2,din_s2=0,0,0,0
    for i in range(0,len(d)):

        q1 = np.matrix(d[i]-mu1)
        num_s1 += (1-eval1[i]) * np.dot(q1.T, q1)
        din_s1 += (1-eval1[i])

        q2 = np.matrix(d[i]-mu2)
        num_s2 += eval1[i] * np.dot(q2.T, q2)
        din_s2 += eval1[i]

    s1 = num_s1/din_s1
    s2 = num_s2/din_s2

    pi = sum(eval1)/len(d)

    lis2=[mu1,mu2,s1,s2,pi]
    return(lis2)
```

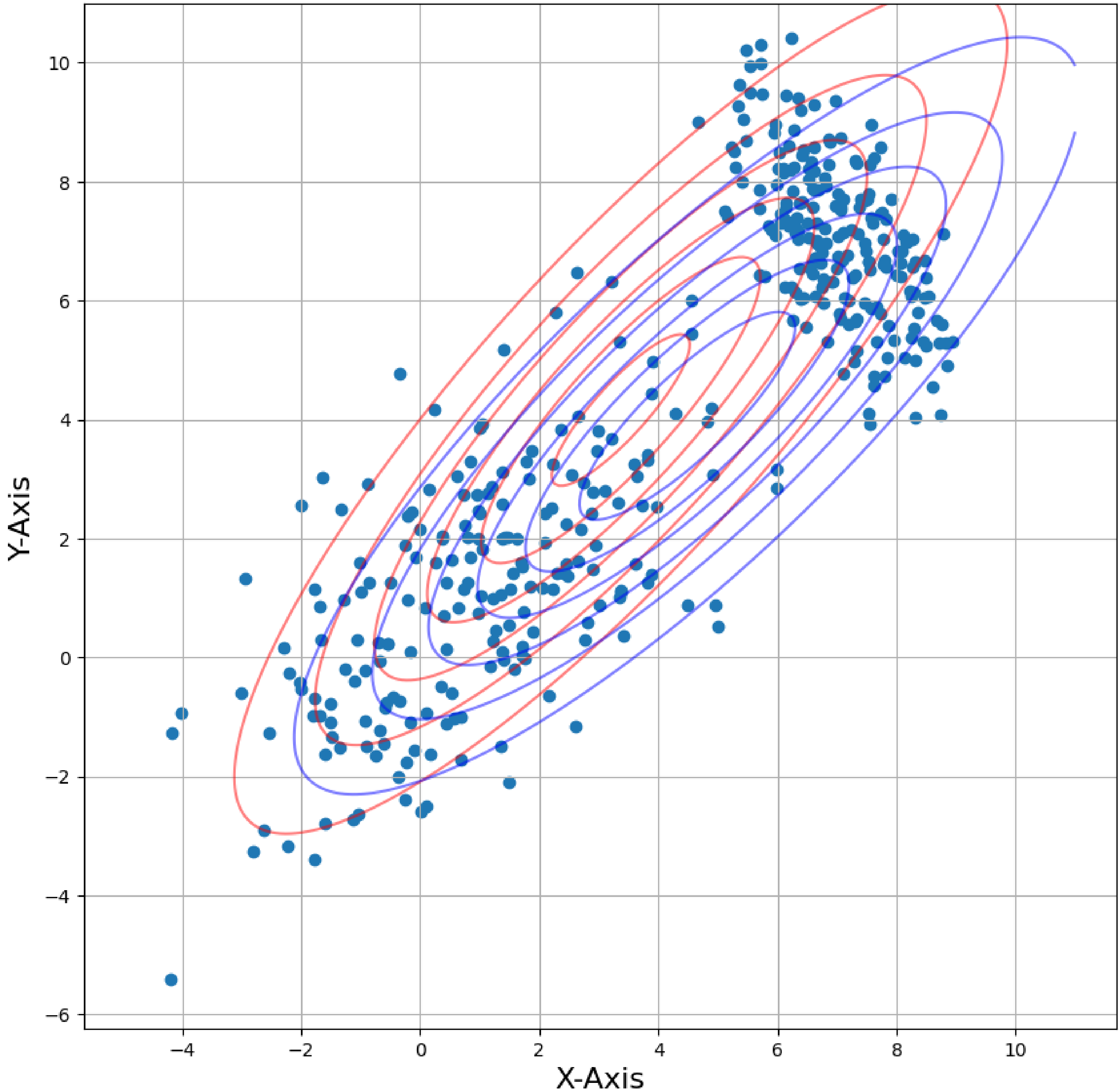
```python
#Function to plot the EM algorithm
def plot(lis1):
    mu1=lis1[0]
    mu2=lis1[1]
    s1=lis1[2]
    s2=lis1[3]
    Z1 = multivariate_normal(mu1, s1)
    Z2 = multivariate_normal(mu2, s2)

    pos = np.empty(X.shape + (2,))                # a new array of given shape and type,
without initializing entries
    pos[:, :, 0] = X; pos[:, :, 1] = Y

    plt.figure(figsize=(10,10))
# creating the figure and assigning the size
    plt.scatter(d[:,0], d[:,1], marker='o')
    plt.contour(X, Y, Z1.pdf(pos), colors="r" ,alpha = 0.5)
    plt.contour(X, Y, Z2.pdf(pos), colors="b" ,alpha = 0.5)
    plt.axis('equal')
# making both the axis equal
    plt.xlabel('X-Axis', fontsize=16)
# X-Axis
    plt.ylabel('Y-Axis', fontsize=16)
# Y-Axis
    plt.grid()
# displaying gridlines
    plt.show()
```

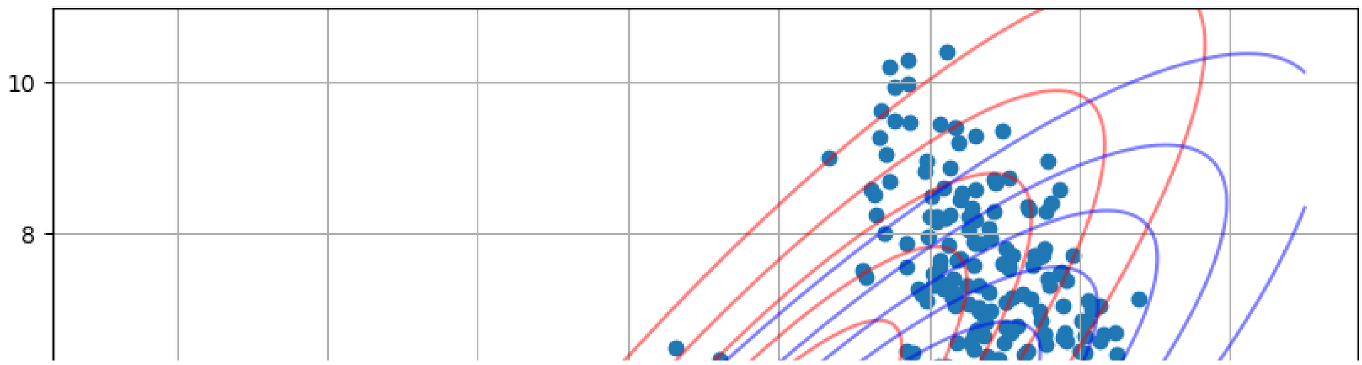
In [210]:

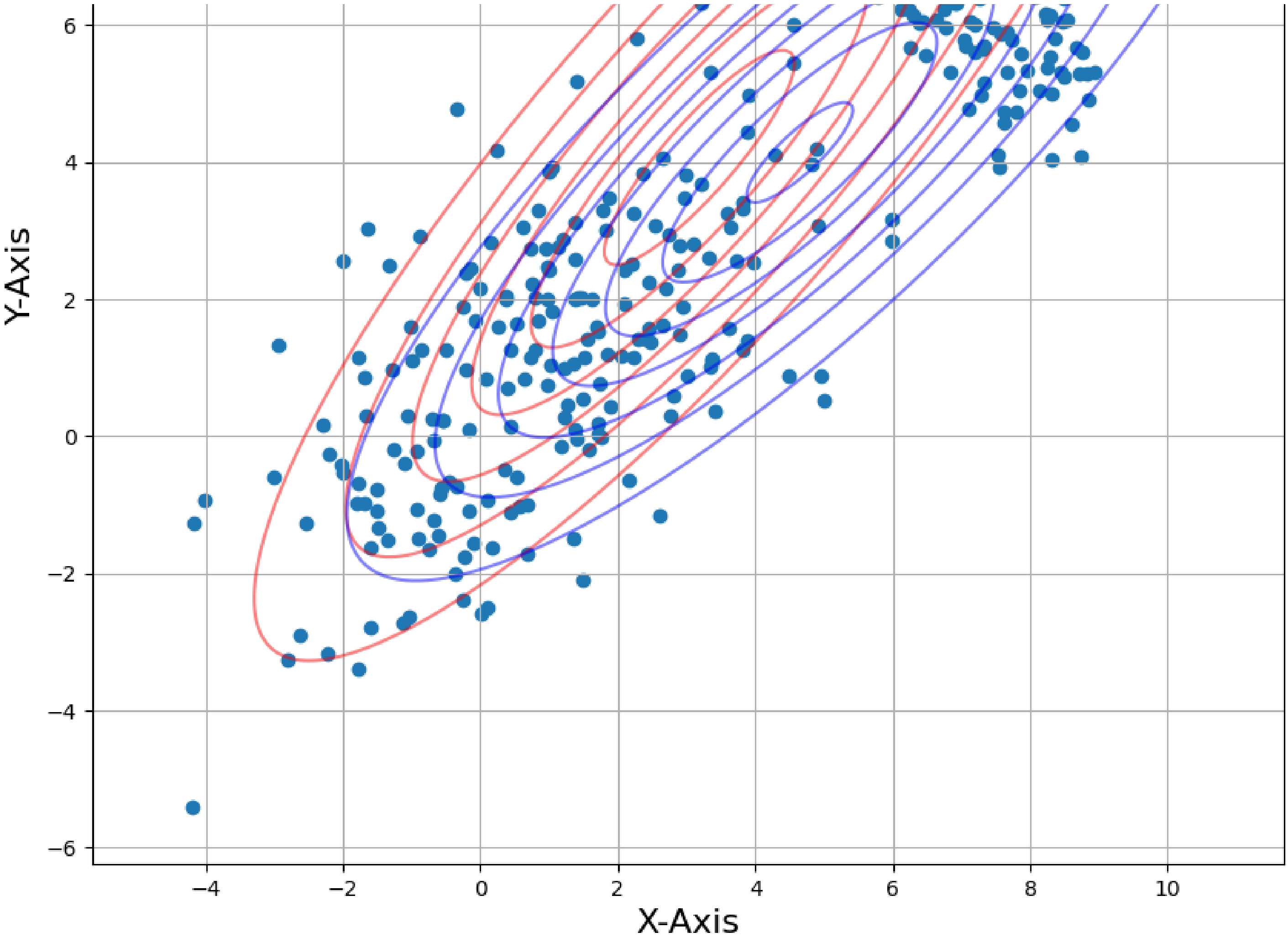
```python
iterations = 20
lis1=[m1,m2,cov1,cov2,pi]
for i in range(0,iterations):
    lis2 = Mstep(Estep(lis1))
    lis1=lis2
    if(i==0 or i == 4 or i == 9 or i == 14 or i == 19):
        plot(lis1)
```

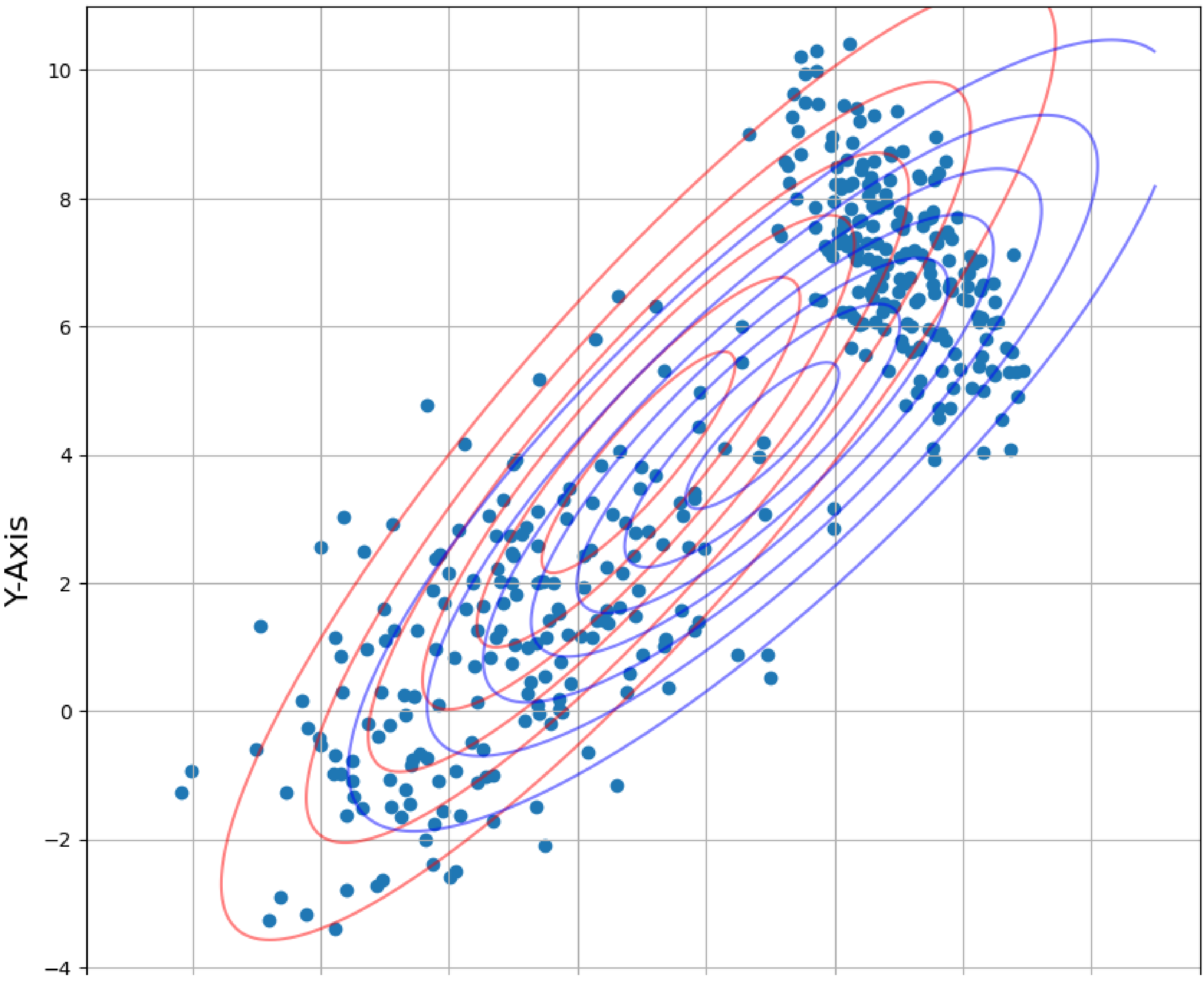


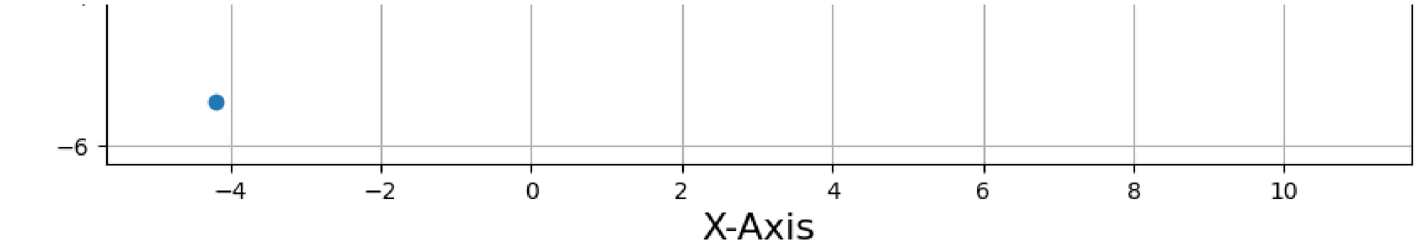

−4
−6
−4
−2
0
2
4
6
8
10
X-Axis

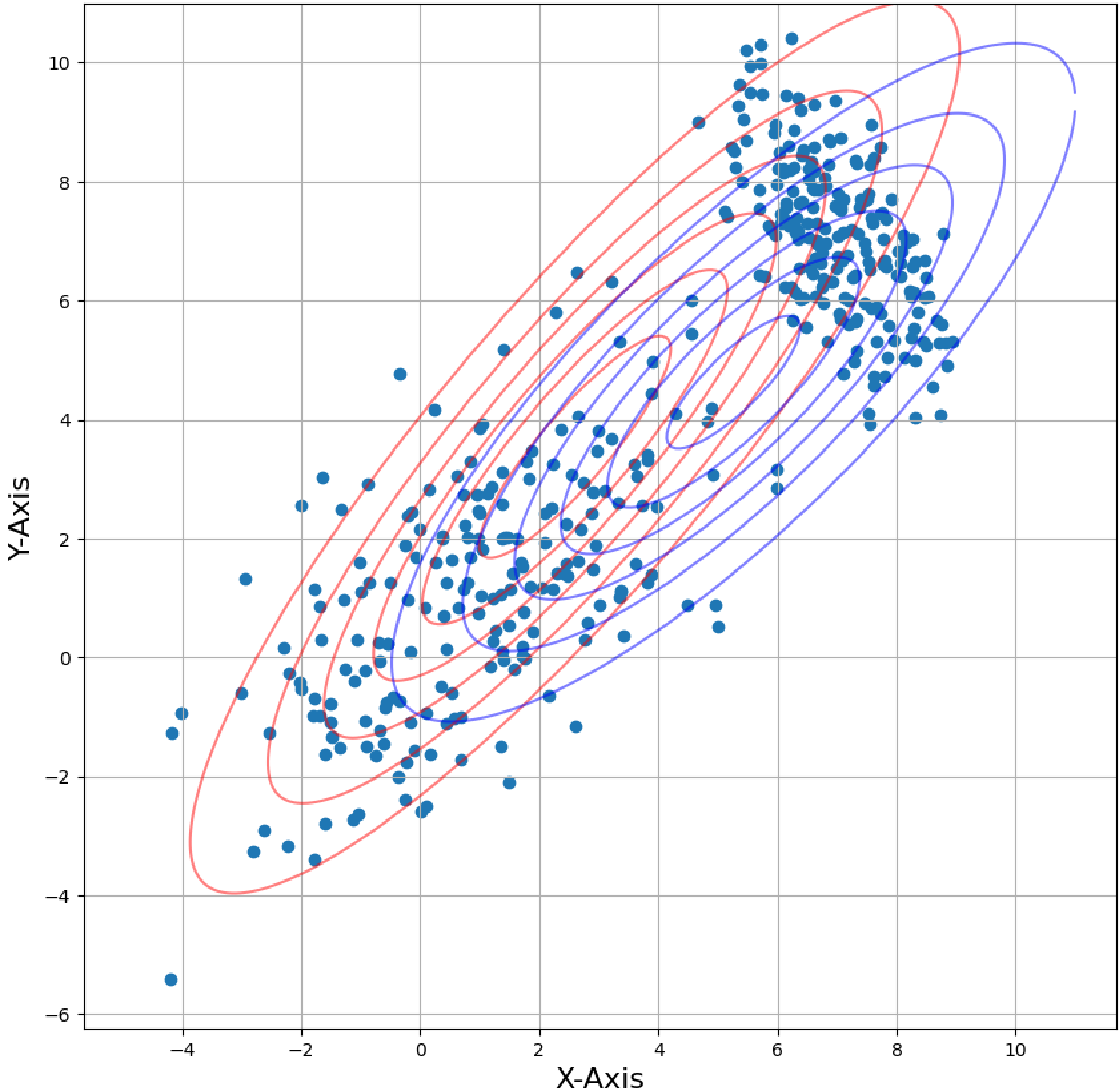

10
8
6
4
2
0
−2
−4
−6
Y-Axis
−4
−2
0
2
4
6
8
10
X-Axis


10
8


Y-Axis
X-Axis


Y-Axis





In [210]: