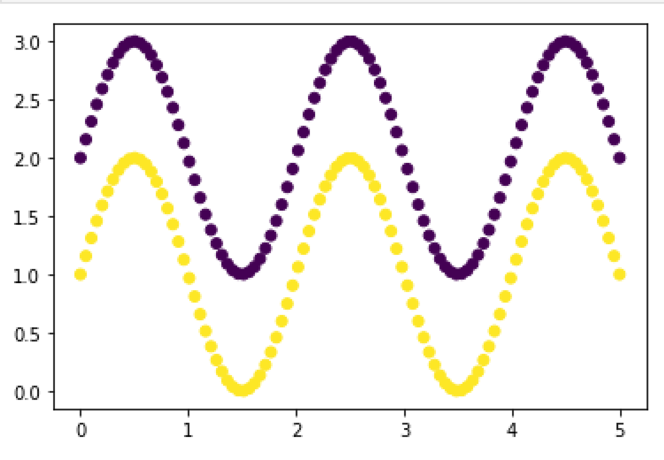
Your work:

- Load this dataset to numpy, with first two columns as features and last as target
- Plot the data using a scatter plot
- Perform the SVM classification using our scratch code

In [13]:
```
dataset = [[3.63636364e+00,1.09036800e+00,0], [4.09090909e+00,2.28173256e+00,0], [1.01010101e-01,1.31203345e+0
```

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt

data = np.array(dataset)

X = data[:, :2]
y = data[:, 2]
```

In [15]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, marker='o')
plt.show()
```



In [16]:
```
#Importing with custom names to avoid issues with numpy / sympy matrix
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers

#Initializing values and computing H. Note the 1. to force to float type
C = 10
m,n = X.shape
y = y.reshape(-1,1) * 1.
X_dash = y * X
H = np.dot(X_dash , X_dash.T) * 1.

#Converting into cvxopt format - as previously
P = cvxopt_matrix(H)
q = cvxopt_matrix(-np.ones((m, 1)))
G = cvxopt_matrix(np.vstack((np.eye(m)*-1,np.eye(m))))
h = cvxopt_matrix(np.hstack((np.zeros(m), np.ones(m) * C)))
A = cvxopt_matrix(y.reshape(1, -1))
b = cvxopt_matrix(np.zeros(1))

#Run solver
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
alphas = np.array(sol['x'])

#==================Computing and printing parameters===============================#
w = ((y * alphas).T @ X).reshape(-1,1)
S = (alphas > 1e-4).flatten()
b = y[S] - np.dot(X[S], w)

#Display results
print('Alphas = ',alphas[alphas > 1e-4])
print('w = ', w.flatten())
print('b = ', b[0])
```

```
Alphas =  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
w =  [2.45403203e-26 3.14744154e-26]
b =  [-1.23556224e-25]
```

In [17]:
```
#w parameter in vectorized form
w = ((y * alphas).T @ X).reshape(-1,1)

#Selecting the set of indices S corresponding to non zero parameters
S = (alphas > 1e-4).flatten()

#Computing b
b = y[S] - np.dot(X[S], w)

#Display results
print('Alphas = ',alphas[alphas > 1e-4])
print('w = ', w.flatten())
print('b = ', b[0])
```

```
Alphas =  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
w =  [2.45403203e-26 3.14744154e-26]
b =  [-1.23556224e-25]
```

2. Scratch: SVM with Soft Margin

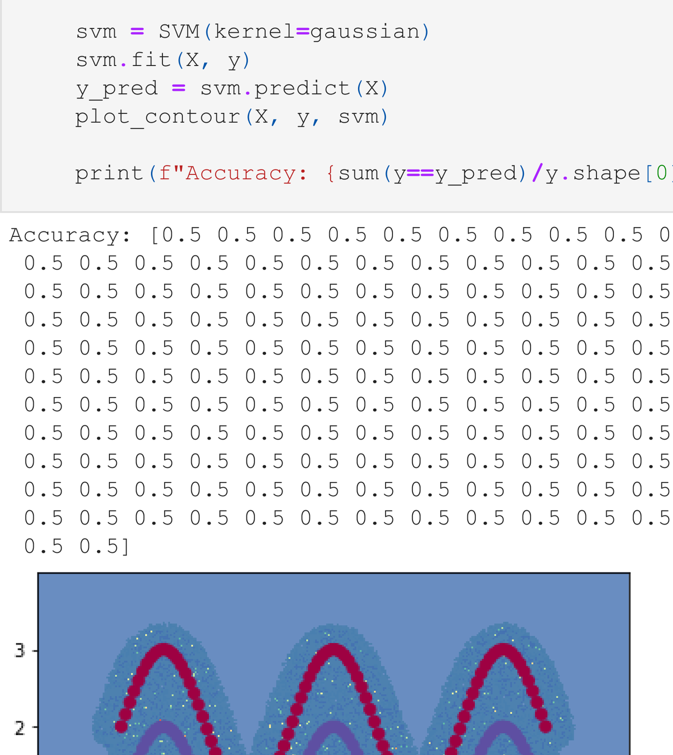
To make the data no longer linearly separable, we shall add a positive point in the middle of the negative cluster:

In [18]:
```
from sklearn.svm import SVC

clf = SVC(C = 10, kernel = 'linear')
clf.fit(X, y.ravel())

print('w = ',clf.coef_)
print('b = ',clf.intercept_)
print('Indices of support vectors = ', clf.support_)
print('Support vectors = ', clf.support_vectors_)
print('Number of support vectors for each class = ', clf.n_support_)
print('Coefficients of the support vector in the decision function = ', np.abs(clf.dual_coef_))
```

```
w =  [[-6.01062521e-06 -1.37116125e+00]]
b =  [2.37090601]
Indices of support vectors =  [  0   1   4   9  12  19  20  21  30  31  35  41  51  52  53  58  64  65
  70  71  77  78  83  88  91  95  96  97  98 100 104 105 106 112 117 118
 123 125 129 131 132 137 138 144 145 146 147 148 150 157 159 162 163 175
 177 178 185 188 189 197   2   3   8  14  17  22  28  29  36  37  38  40
  43  46  56  62  66  67  68  72  74  75  76  79  80  81  82  84  85  86
  92 101 107 109 110 111 115 116 119 120 121 122 128 130 133 141 143 153
 160 165 170 176 180 181 182 183 186 191 192 194]
Support vectors =  [[3.63636364 1.090368  ]
 [4.09090909 2.28173256]
 [1.91919192 1.74885201]
 [1.86868687 1.59906946]
 [2.97979798 2.06342392]
 [1.46464646 1.00616154]
 [4.14141414 2.42979491]
 [1.71717172 1.22385354]
 [3.88888889 1.65797986]
 [0.15151515 2.45822652]
 [3.68686869 1.16743015]
 [1.31313131 1.16743015]
 [3.33333333 1.1339746 ]
 [3.13131313 1.59906946]
 [2.02020202 2.06342392]
 [1.41414141 1.03615784]
 [0.05050505 2.1580014 ]
 [3.03030303 1.90494396]
 [3.08080808 1.74885201]
 [2.87878788 2.37166246]
 [3.58585859 1.03615784]
 [0.1010101  2.31203345]
 [0.90909091 2.28173256]
 [0.95959596 2.12659245]
 [2.12121212 2.37166246]
 [5.         2.        ]
 [2.07070707 2.22031053]
 [3.53535354 1.00616154]
 [1.36363636 1.090368  ]
 [3.38383838 1.06585214]
 [1.06060606 1.81074876]
 [4.94949495 2.1580014 ]
 [1.16161616 1.51380326]
 [3.43434343 1.02119755]
 [3.48484848 1.00113266]
 [3.83838384 1.51380326]
 [4.04040404 2.12659245]
 [1.56565657 1.02119755]
 [0.85858586 2.42979491]
 [3.18181818 1.45935918]
 [3.73737374 1.26540829]
 [4.8989899  2.31203345]
 [3.28282828 1.22385354]
 [1.26262626 1.26540829]
 [0.         2.        ]
 [1.11111111 1.65797986]
 [1.01010101 1.96827207]
 [3.23232323 1.333231  ]
 [2.92929293 2.22031053]
 [1.81818182 1.45935918]
 [3.93939394 1.81074876]
 [1.76767677 1.333231  ]
 [3.98989899 1.96827207]
 [1.66666667 1.1339746 ]
 [1.61616162 1.06585214]
 [3.78787879 1.38184101]
 [1.96969697 1.90494396]
 [1.51515152 1.00113266]
 [1.21212121 1.38184101]
 [4.84848485 2.45822652]
 [0.1010101  1.31203345]
 [0.45454545 1.98982144]
 [2.97979798 1.06342392]
 [2.17171717 1.51367739]
 [0.50505051 1.99987413]
 [4.34343434 1.88145336]
 [2.32323232 1.84972543]
 [4.14141414 1.42979491]
 [2.57575758 1.97181157]
 [4.09090909 1.28173256]
 [2.12121212 1.37166246]
 [4.49494949 1.99987413]
 [2.52525253 1.99685478]
 [2.27272727 1.75574957]
 [4.04040404 1.12659245]
 [2.62626263 1.92235429]
 [2.92929293 1.22031053]
 [0.75757576 1.69007901]
 [2.77777778 1.64278761]
 [2.07070707 1.22031053]
 [0.35353535 1.89599377]
 [2.02020202 1.06342392]
 [0.05050505 1.1580014 ]
 [0.         1.        ]
 [0.65656566 1.88145336]
 [4.24242424 1.69007901]
 [4.6969697  1.81457595]
 [0.90909091 1.28173256]
 [4.54545455 1.98982144]
 [2.87878788 1.37166246]
 [4.5959596  1.95490224]
 [0.4040404  1.95490224]
 [0.85858586 1.42979491]
 [2.22222222 1.64278761]
 [4.7979798  1.59290793]
 [2.82828283 1.51367739]
 [2.37373737 1.92235429]
 [0.25252525 1.71269417]
 [4.8989899  1.31203345]
 [0.95959596 1.12659245]
 [4.94949495 1.1580014 ]
 [0.15151515 1.45822652]
 [4.64646465 1.89599377]
 [4.19191919 1.56705986]
 [0.2020202  1.59290793]
 [2.42424242 1.97181157]
 [2.47474747 1.99685478]
 [2.72727273 1.75574957]
 [4.74747475 1.71269417]
 [5.         1.        ]
 [0.55555556 1.98480775]
 [4.44444444 1.98480775]
 [4.29292929 1.79576184]
 [2.67676768 1.84972543]
 [0.80808081 1.56705986]
 [0.3030303  1.81457595]
 [0.60606061 1.94500082]
 [4.84848485 1.45822652]
 [0.70707071 1.79576184]
 [4.39393939 1.94500082]]
Number of support vectors for each class =  [60 60]
Coefficients of the support vector in the decision function =  [[10.         10.         10.         10.
  10.         10.
  10.         10.         10.          6.11772484 10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.          2.31773569
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.          6.00257485
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.
  10.          2.43288568 10.         10.         10.         10.
  10.         10.         10.         10.         10.         10.        ]]
```

In [19]:
```
import cvxopt
from sklearn import datasets

#here I use z instead of xprime since I don't know how to write prime in code....
def linear(x, z):
    return np.dot(x, z.T)

def polynomial(x, z, p=5):
    return (1 + np.dot(x, z.T)) ** p

def gaussian(x, z, sigma=0.1):
    return np.exp(-np.linalg.norm(x - z, axis=1) ** 2 / (2 * (sigma ** 2)))

def plot_contour(X, y, svm):
    # plot the resulting classifier
    h = 0.01
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    points = np.c_[xx.ravel(), yy.ravel()]

    Z = svm.predict(points)
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)

    # plt the points
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)


class SVM:
    def __init__(self, kernel=gaussian, C=1):
        self.kernel = kernel
        self.C = C

    def fit(self, X, y):
        self.y = y
        self.X = X
        m, n = X.shape

        # Calculate Kernel
        self.K = np.zeros((m, m))
        for i in range(m):
            self.K[i, :] = self.kernel(X[i, np.newaxis], self.X)

        # Solve with cvxopt final QP needs to be reformulated
        # to match the input form for cvxopt.solvers.qp
        P = cvxopt.matrix(np.outer(y, y) * self.K)
        q = cvxopt.matrix(-np.ones((m, 1)))
        G = cvxopt.matrix(np.vstack((np.eye(m) * -1, np.eye(m))))
        h = cvxopt.matrix(np.hstack((np.zeros(m), np.ones(m) * self.C)))
        A = cvxopt.matrix(y, (1, m), "d")
        b = cvxopt.matrix(np.zeros(1))
        cvxopt.solvers.options["show_progress"] = False
        sol = cvxopt.solvers.qp(P, q, G, h, A, b)
        self.alphas = np.array(sol["x"])

    def predict(self, X):  #<----this is X_test
        y_predict = np.zeros((X.shape[0]))
        sv = self.get_parameters(self.alphas)

        for i in range(X.shape[0]):
            y_predict[i] = np.sum(
                self.alphas[sv]
                * self.y[sv, np.newaxis]
                * self.kernel(X[i], self.X[sv])[:, np.newaxis]
            )

        return np.sign(y_predict + self.b)

    def get_parameters(self, alphas):
        threshold = 1e-5

        sv = ((alphas > threshold) * (alphas < self.C)).flatten()
        self.w = np.dot(self.X[sv].T, alphas[sv] * self.y[sv, np.newaxis])
        self.b = np.mean(
            self.y[sv, np.newaxis]
            - self.alphas[sv] * self.y[sv, np.newaxis] * self.K[sv, sv][:, np.newaxis]
        )
        return sv

if __name__ == "__main__":
#     X, y = datasets.make_moons(50)

    #transform our y to be -1 and 1 to meet svm purpose
    y[y==0] = -1

    svm = SVM(kernel=gaussian)
    svm.fit(X, y)
    y_pred = svm.predict(X)
    plot_contour(X, y, svm)

    print(f"Accuracy: {sum(y==y_pred)/y.shape[0]}")
```

```
Accuracy: [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5]
```



In [20]:
```
print("Done")
```

```
Done
```