Note to grader: Each question consists of parts, e.g. Q1(i), Q1(ii), etc. Each part must be graded on a 0-4 scale, following the standard NJIT convention (A:4, B+: 3.5, B:3, C+: 2.5, C: 2, D:1, F:0). The total score must be re-scaled to 100 -- that should apply to all future assignments so that Canvas assigns the same weight on all assignments.

Assignment 2

This assignment walks you through the basics of the perceptron. You will also do some first experiments on a toy data set, and check the effect of hyperparameters. The intended goal of the assignment is to familiarize you further with the Jupyter/Colab environment and help you acquire some tools that we will later use to experiment with 'professional-grade' data sets and algorithms.

Note: You must run/evaluate all cells. **Order of cell execution is important.**

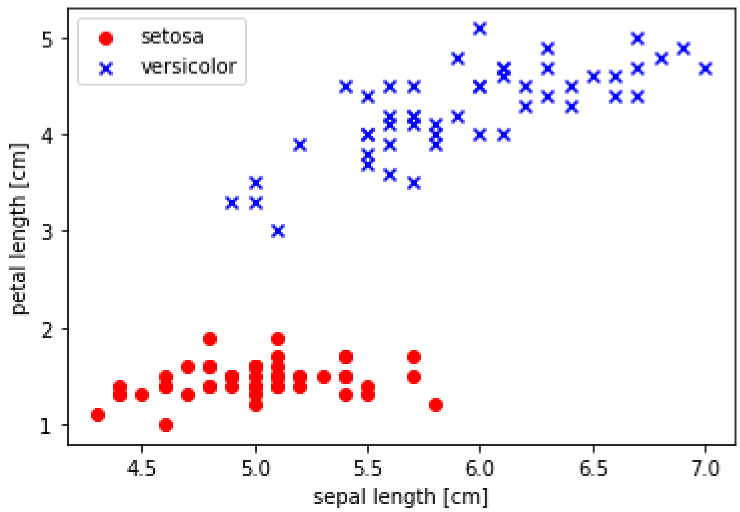
Preparation Steps

In [1]:
```
# Import all necessary python packages
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

In [2]:
```
# ### Reading-in the Iris data

s = os.path.join('https://archive.ics.uci.edu', 'ml',
                 'machine-learning-databases', 'iris','iris.data')
s = s.replace("\\","/");
print('URL:', s)
df = pd.read_csv(s,header=None,encoding='utf-8')
```

```
URL: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
```

In [3]:
```
# select setosa and versicolor
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

# extract sepal length and petal length
X = df.iloc[:100, [0, 2]].values

# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')

plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')


# plt.savefig('images/02_06.png', dpi=300)
plt.show()
```



Question 0. Manual Perceptron Training

Below you can see the first 5 data points of the data set, all labeled as 'setosa'.

In [4]:
```
X[0:5], y[0:5]
```

Out[4]:
```
(array([[5.1, 1.4],
        [4.9, 1.4],
        [4.7, 1.3],
        [4.6, 1.5],
        [5. , 1.4]]),
 array([-1, -1, -1, -1, -1]))
```

Suppose the initial weights of the perceptron are w0=0.1, w1=0.2, w2=-0.1 . Here w_0 is the bias.

In the following space (Double click this text), write the weights after processing data points 0,1,2, and show your calculations (with $\eta = 0.1$):

w0 + w1 x1 + w2 x2 = 0

w0 + w1 5.1 + w2 x2 = 0.1 + 0.2*5.1 - 0.1*1.4

w2 := w2 - (y - y_hat)*x2 = -0.1 -(-2)*1.4

w1 := w1 - (y - y_hat)*x1

w0 := w0 - (y - y_hat)

Computing Weights by passing 0,1,2 points 1) Passing point (x1,x2,y1) = (5.1,1.4,-1) y_hat1 = 0.1 + 0.2*5.1 -0.1*1.4

y_hat1 = 0.98

Weights updtation

w0 = 0.1 - (-1-0.98)*0.1

w0 = 0.299

w1 = 0.2 - 0.1*(-1 -0.98)*5.1

w1 = 0.2 + 1.0098

w1 = 1.2098

w2 = -0.1 - 0.1*(-1-0.98)*1.4

w2 = -0.1 + 0.2772

w2 = 0.1772

2) Passing point (x1,x2,y1) = (4.9,1.4,-1)

y_hat2 = 0.299 + 1.2098*4.9 + 0.1772*1.4

y_hat2 = 6.4751

Weights updtation

w0 = 0.299 -(-1-6.4751)*0.1

w0 = 1.04651

w1 = 1.2098 - 0.1*(-1-6.4751)*4.9

w1 = 4.87251

w2 = 0.1772 - 0.1*(-1-6.4751)*1.4

w2 = 1.2237

3) Passing point (x1,x2,y1) = (4.7,1.3,-1)

y_hat2 = 1.04651 + 4.87251*4.7 + 1.2237*1.3

y_hat2 = 25.538

Weights updtation

w0 = 1.04651 - (-1 - 25.538)*0.1

w0 = 3.70031

w1 = 4.87251 - 0.1*(-1 - 25.538)*4.7

w1 = 17.345

w2 = 1.2237 - 0.1*(-1 - 25.538)*1.3

w2 = 4.6736

In [ ]:

In [5]:

```
# Grader's area
import numpy as np
M = np.zeros([10,10])
maxScore = 0
```

```
maxScore = maxScore +4
# M[0,1] =
```

Question 1. Perceptron Code Modification

The following code is a perceptron implementation (with three do-nothing lines 59-61).

In [6]:
```
import numpy as np


class Perceptron(object):
    """Perceptron classifier.

    Parameters
    ------------
    eta : float
      Learning rate (between 0.0 and 1.0)
    n_iter : int
      Passes over the training dataset.
    random_state : int
      Random number generator seed for random weight
      initialization.

    Attributes
    -----------
    w_ : 1d-array
      Weights after fitting.
    errors_ : list
      Number of misclassifications (updates) in each epoch.

    """
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter  # Attribute for iterations
        self.weights = []  # Attribute for weights
        self.random_state = random_state

    def fit(self, X, y):
        """Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_examples, n_features]
          Training vectors, where n_examples is the number of examples and
          n_features is the number of features.
        y : array-like, shape = [n_examples]
          Target values.

        Returns
        -------
        self : object

        """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []
```

```
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.weights.append(self.w_) # storing weights at each iteration to main
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
            if errors == 0: # stops when no more iterations are necessary
                break
            # my do-nothing code
            IK = 2020
            # my do-nothing code

        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

Work on the above cell and modify the code so that:

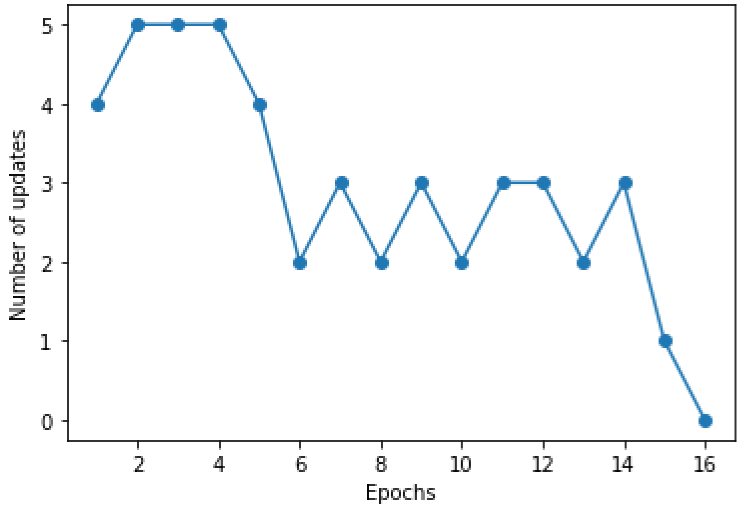
(i) The fit function stops when no more iterations are necessary.
(ii) The trained perceptron contains as an attribute not only its weights, but also the number of iterations it took for training
(iii) The perceptron maintains a history of its weights, i.e. the set of weights after each point is processed.

To modify the code please insert your code with clear comments surrounding it, similarly to "my do-nothing code". Make sure you evaluate the cell again, so that following cells will be using the modified perceptron.

In [7]:
```
# Grader's area

maxScore = maxScore +4
# M[1,1] =
# M[1,2] =
# M[1,3] =
```

Question2: Experimenting with hyperparameters

In [8]:
```
ppn = Perceptron(eta=0.0001, n_iter=20, random_state=1)
ppn.fit(X, y)

plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
```



Running the above code, you can verify if your modification in question 1 works correctly. The point of this question is to experiment with the different hyperparameters. Here are some specific questions:
(i) Find the largest value of η for which the process takes more than 20 iterations to converge. Explain how you found that η
(ii) Are you able to find $\eta > 1$ for which the process fails to converge in less than 30 iterations?
(iii) Find two different settings for the random state, that give different convergence patterns, for the same η.

Please give your answers in the cell below.

In [9]:
```
#(i) the largest value of eta for which the process takes more than 20 iterations to con
eta = 1
while eta <= 1 and eta >= 0:
    ppn = Perceptron(eta=eta, n_iter=21, random_state=1)
    ppn.fit(X, y)
    eta -= 0.0001
    if ppn.errors_[-1] == 0:
        break
print(eta)
```

```
0.9999
```

(i)η is 0.9999 I have intialized the η as 1 and trained perceptron for 20 iteraions for different values of η and found converging at 0.9999

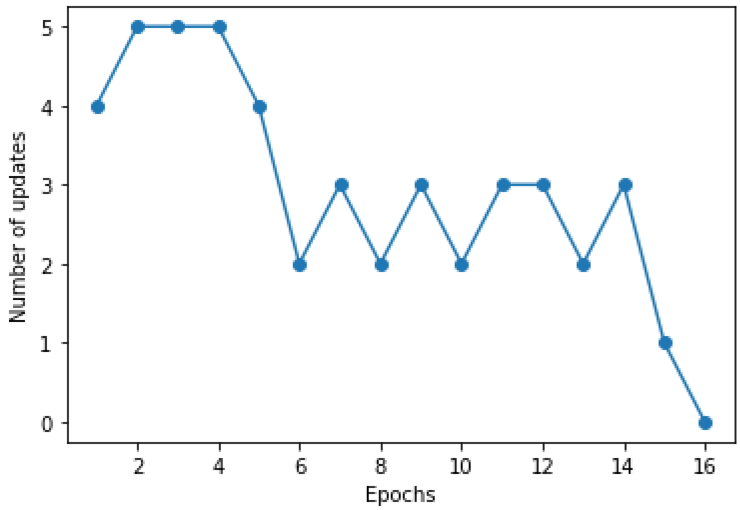
In [10]:
```
# (ii) Are you able to find $\eta>1$ for which the process fails to converge in less tha
eta = 1
loop = False
while eta >= 1 and eta <= 10:
    for iter in range(20, 30):
        ppn = Perceptron(eta=eta, n_iter=iter, random_state=1)
        ppn.fit(X, y)
        eta += 0.0001
        if ppn.errors_[-1] == 0:
            loop = True
            break
    if loop:
        break
eta
```

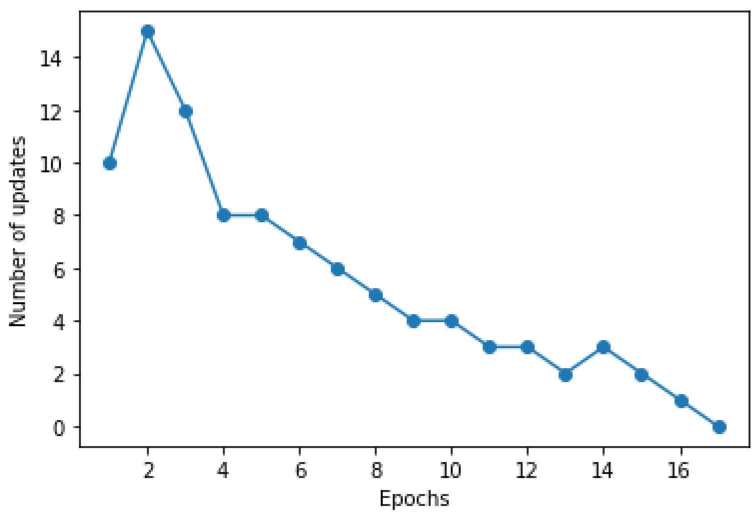
Out[10]:
```
1.0001
```

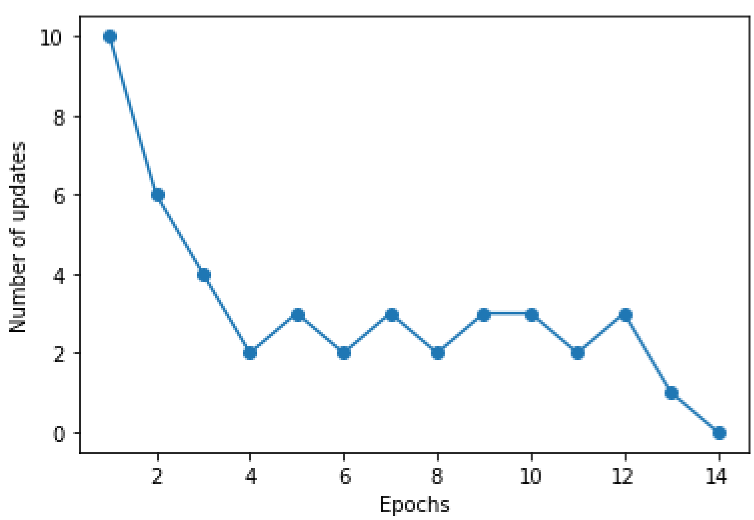
```
# (iii)Find two different settings for the random state, that give different convergence
```

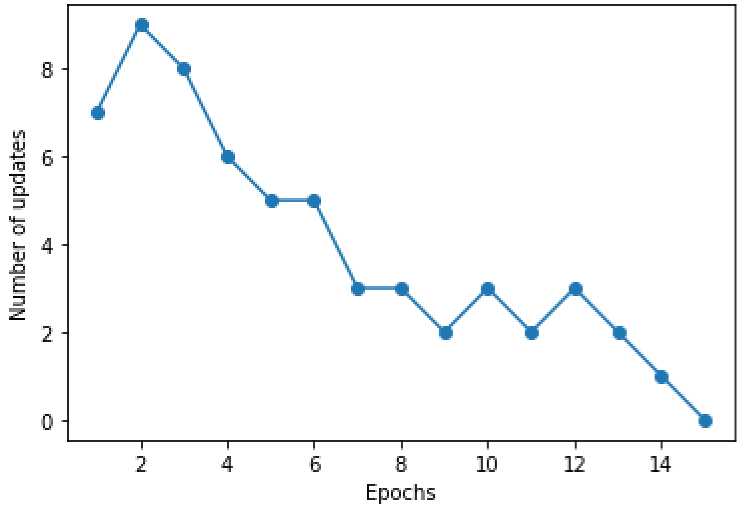
In [11]:

```
for r in range(1, 20):
    ppn = Perceptron(eta=0.0001, n_iter=20, random_state=r)
    ppn.fit(X, y)
    if ppn.errors_[-1] == 0:
        plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
        plt.xlabel('Epochs')
        plt.ylabel('Number of updates')
        plt.show()
```

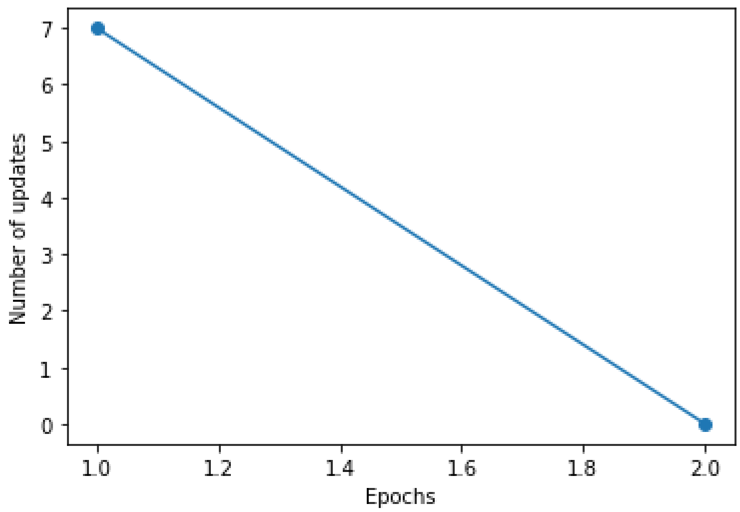




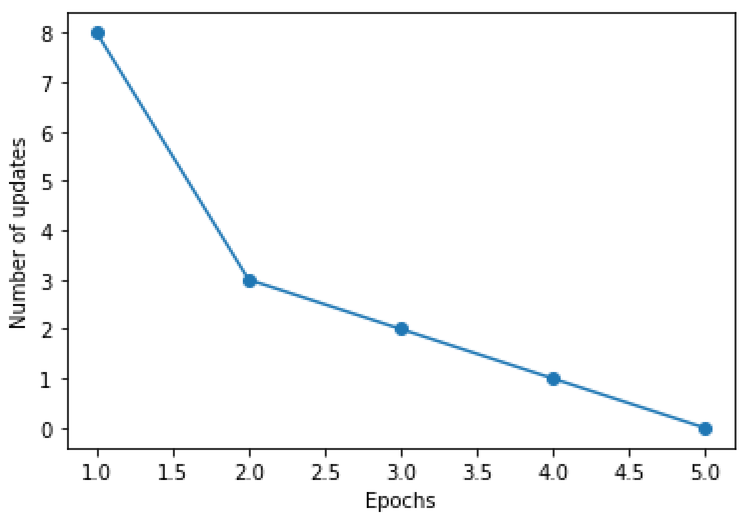




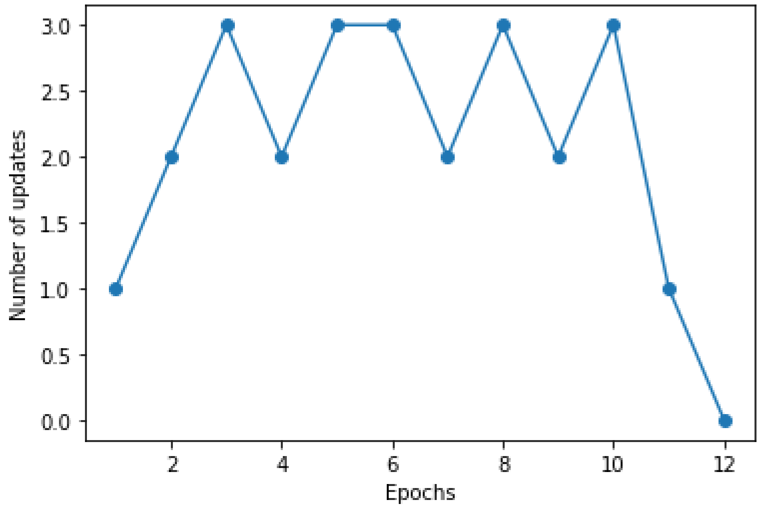
Number of updates
Epochs



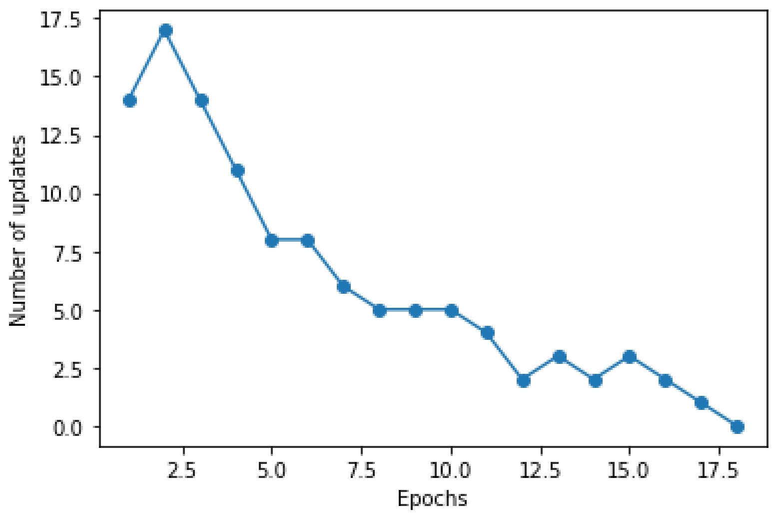
Number of updates
Epochs



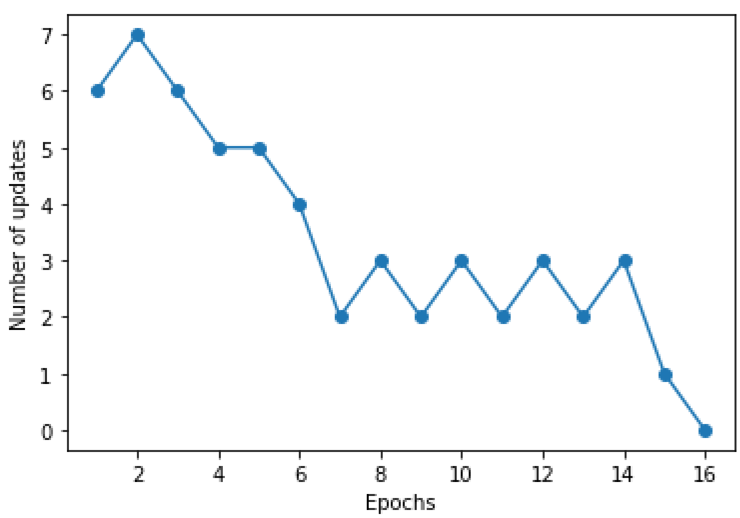
Number of updates
Epochs



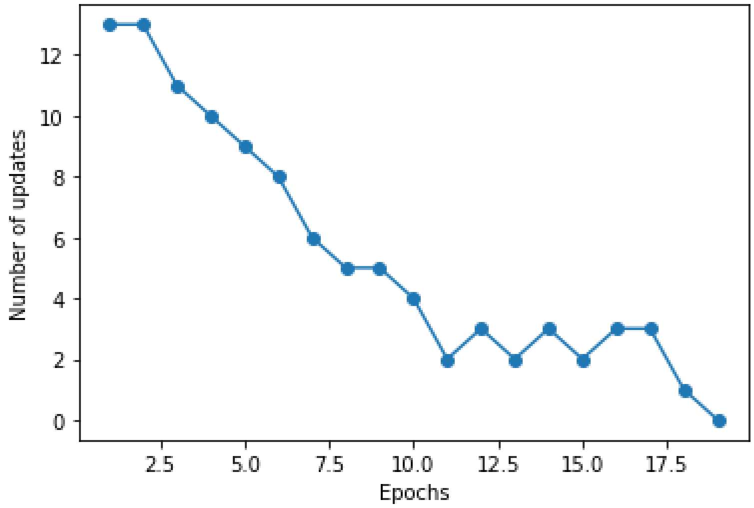
Number of updates
Epochs

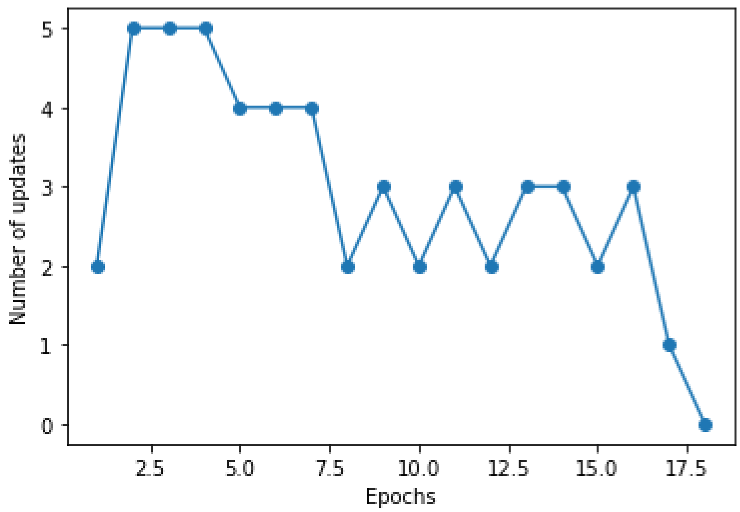


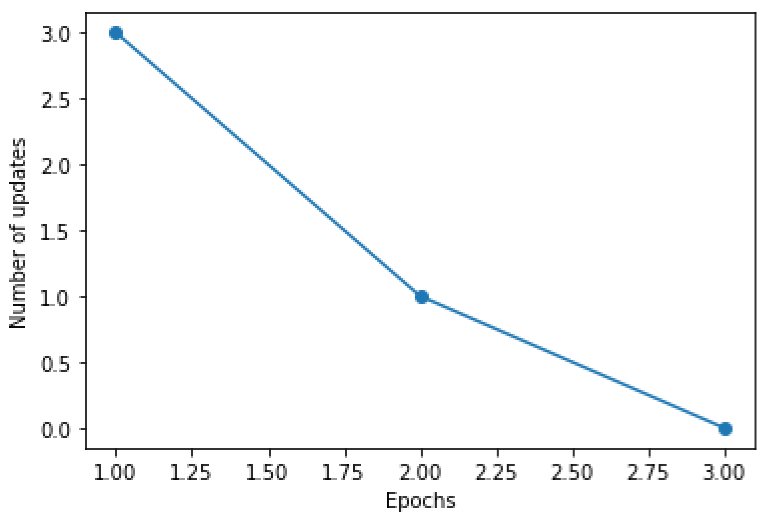
Number of updates
Epochs

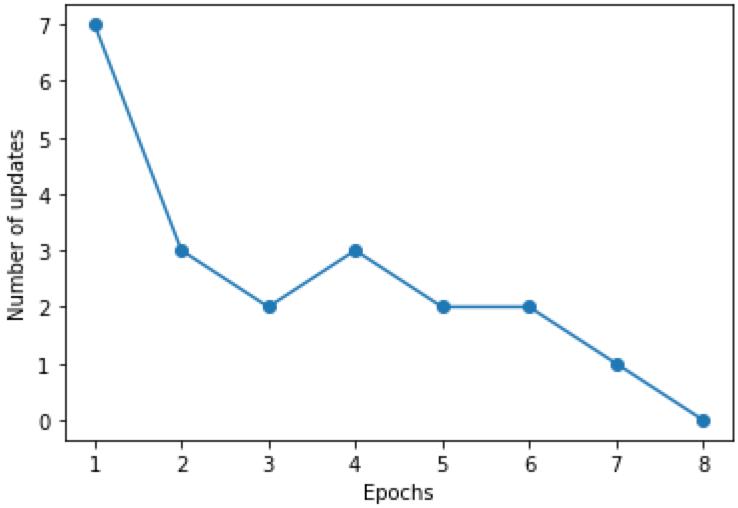


Number of updates
Epochs

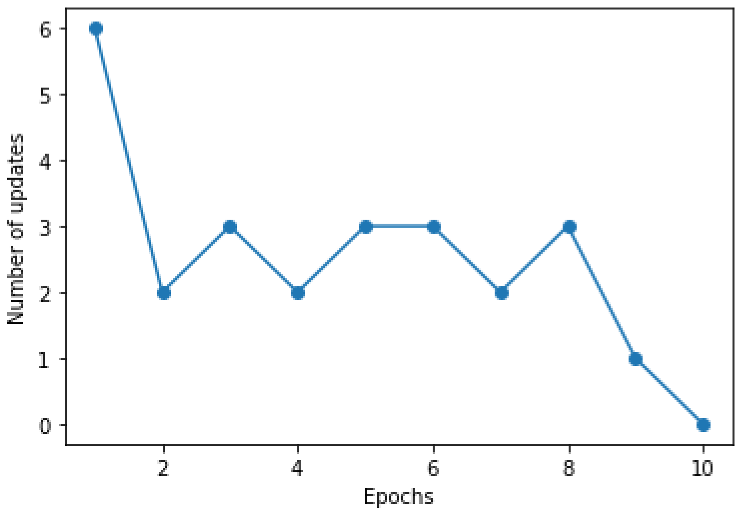

Number of updates
Epochs
0
2
4
6
8
10
12
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Number of updates
Epochs
0
1
2
3
4
5
2.5
5.0
7.5
10.0
12.5
15.0
17.5

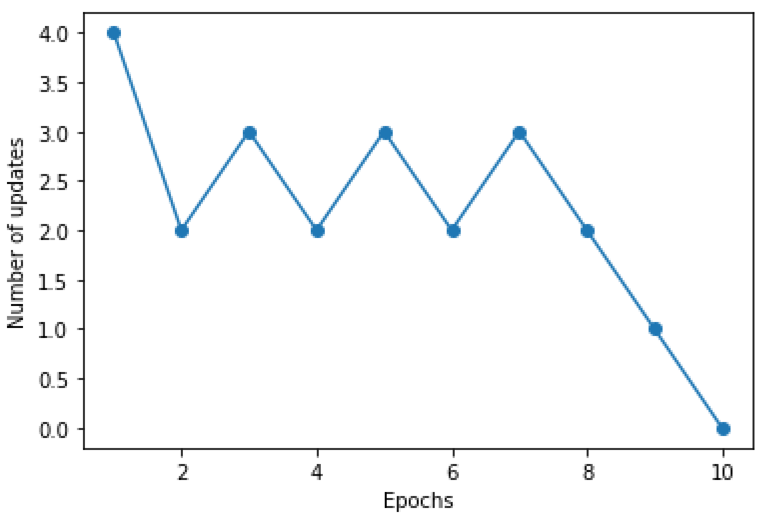

Number of updates
Epochs
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00



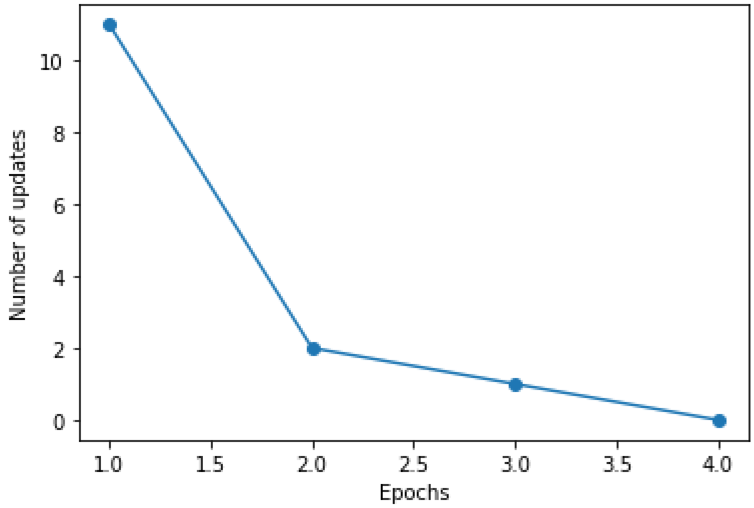
Number of updates
Epochs

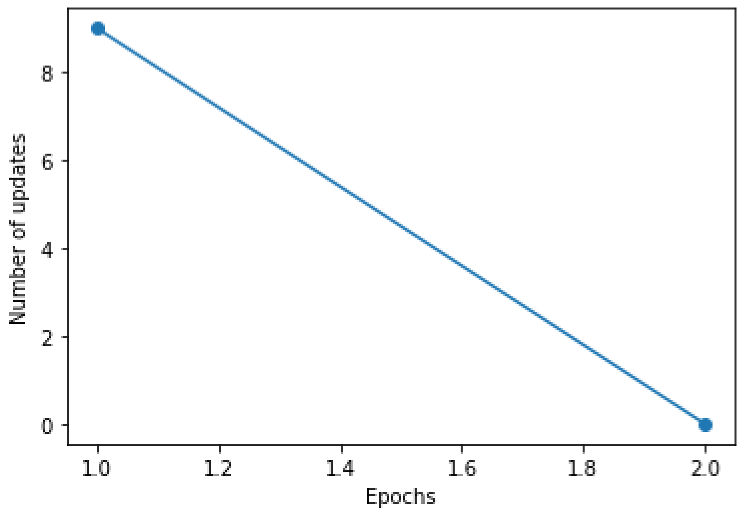


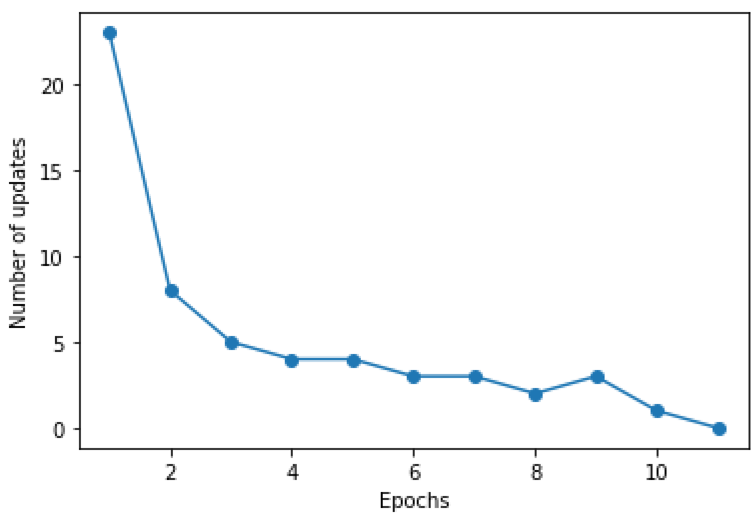
Number of updates
Epochs

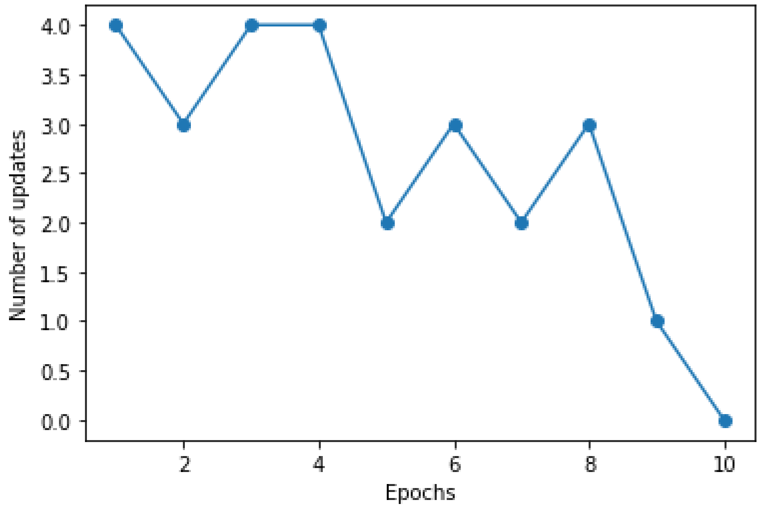


Number of updates
Epochs


Number of updates
0
2
4
6
8
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs


Number of updates
0
2
4
6
8
1.0
1.2
1.4
1.6
1.8
2.0
Epochs


Number of updates
0
5
10
15
20
2
4
6
8
10
Epochs



In [12]:
```
# Grader's area

maxScore = maxScore +4
# M[2,1] =
# M[2,2] =
# M[2,3] =
```

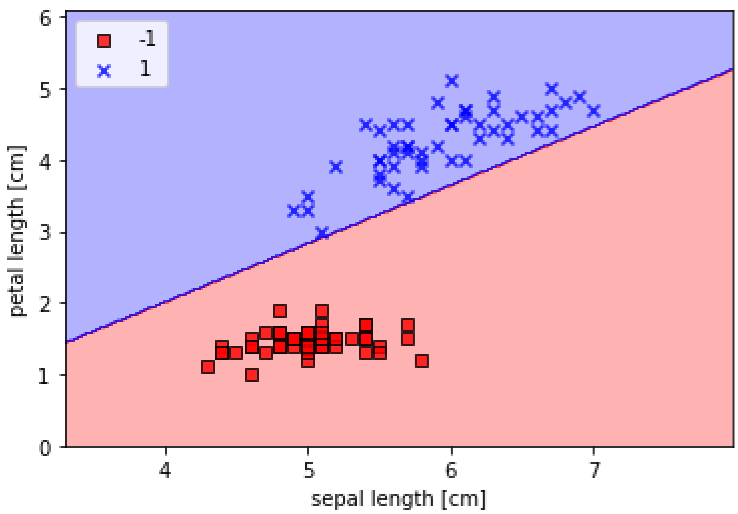
Question 3: Visualizing multiple decision regions over time

Here is the function for visualizing decision regions

In [13]:
```
from matplotlib.colors import ListedColormap


def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class examples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')
```

In [14]:
```
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')


# plt.savefig('images/02_08.png', dpi=300)
plt.show()
```

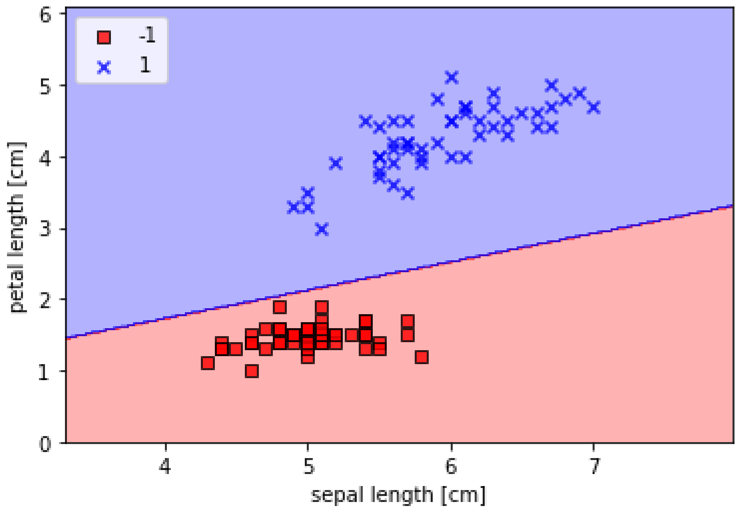
```
C:\Users\vamsi\AppData\Local\Temp\ipykernel_3120\1032177424.py:24: UserWarning: You pass
ed a edgecolor/edgecolors ('black') for an unfilled marker ('x').  Matplotlib is ignorin
g the edgecolor in favor of the facecolor.  This behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
```



Using the above, give code that plots the decision regions for the first 5 epochs. Use learning rate = 0.01 and random seed = 1 when applicable.

In [15]:
```
# Perceptron with eta = 0.01, 5 epochs, random state 1
ppn = Perceptron(eta=0.01, n_iter=5, random_state=1)
ppn.fit(X, y)

plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()

x
```


```
C:\Users\vamsi\AppData\Local\Temp\ipykernel_3120\1032177424.py:24: UserWarning: You pass
ed a edgecolor/edgecolors ('black') for an unfilled marker ('x').  Matplotlib is ignorin
g the edgecolor in favor of the facecolor.  This behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
```



Out[15]:
```
array([[5.1, 1.4],
       [4.9, 1.4],
       [4.7, 1.3],
       [4.6, 1.5],
       [5. , 1.4],
       [5.4, 1.7],
       [4.6, 1.4],
       [5. , 1.5],
       [4.4, 1.4],
       [4.9, 1.5],
       [5.4, 1.5],
       [4.8, 1.6],
       [4.8, 1.4],
       [4.3, 1.1],
       [5.8, 1.2],
       [5.7, 1.5],
       [5.4, 1.3],
       [5.1, 1.4],
       [5.7, 1.7],
       [5.1, 1.5],
       [5.4, 1.7],
       [5.1, 1.5],
       [4.6, 1. ],
       [5.1, 1.7],
       [4.8, 1.9],
       [5. , 1.6],
       [5. , 1.6],
       [5.2, 1.5],
       [5.2, 1.4],
       [4.7, 1.6],
       [4.8, 1.6],
       [5.4, 1.5],
       [5.2, 1.5],
       [5.5, 1.4],
       [4.9, 1.5],
       [5. , 1.2],
       [5.5, 1.3],
       [4.9, 1.5],
       [4.4, 1.3],
       [5.1, 1.5],
       [5. , 1.3],
       [4.5, 1.3],
       [4.4, 1.3],
       [5. , 1.6],
       [5.1, 1.9],
       [4.8, 1.4],
       [5.1, 1.6],
       [4.6, 1.4],
       [5.3, 1.5],
```

```
       [5. , 1.4],
       [7. , 4.7],
       [6.4, 4.5],
       [6.9, 4.9],
       [5.5, 4. ],
       [6.5, 4.6],
       [5.7, 4.5],
       [6.3, 4.7],
       [4.9, 3.3],
       [6.6, 4.6],
       [5.2, 3.9],
       [5. , 3.5],
       [5.9, 4.2],
       [6. , 4. ],
       [6.1, 4.7],
       [5.6, 3.6],
       [6.7, 4.4],
       [5.6, 4.5],
       [5.8, 4.1],
       [6.2, 4.5],
       [5.6, 3.9],
       [5.9, 4.8],
       [6.1, 4. ],
       [6.3, 4.9],
       [6.1, 4.7],
       [6.4, 4.3],
       [6.6, 4.4],
       [6.8, 4.8],
       [6.7, 5. ],
       [6. , 4.5],
       [5.7, 3.5],
       [5.5, 3.8],
       [5.5, 3.7],
       [5.8, 3.9],
       [6. , 5.1],
       [5.4, 4.5],
       [6. , 4.5],
       [6.7, 4.7],
       [6.3, 4.4],
       [5.6, 4.1],
       [5.5, 4. ],
       [5.5, 4.4],
       [6.1, 4.6],
       [5.8, 4. ],
       [5. , 3.3],
       [5.6, 4.2],
       [5.7, 4.2],
       [5.7, 4.2],
       [6.2, 4.3],
       [5.1, 3. ],
       [5.7, 4.1]])
```

In [16]:

```
# Grader's area

maxScore = maxScore +4
# M[3,1] =
```

Question 4: Changing the data order in training

The data arrays (X,y) currently in the memory are organized so that the all data points with a given label (e.g. 'Setosa') lie in a contiguous part of the arrays (X,y). In this question we will check the impact of changing the order of the data in the number of iterations required to learn a correct perceptron.

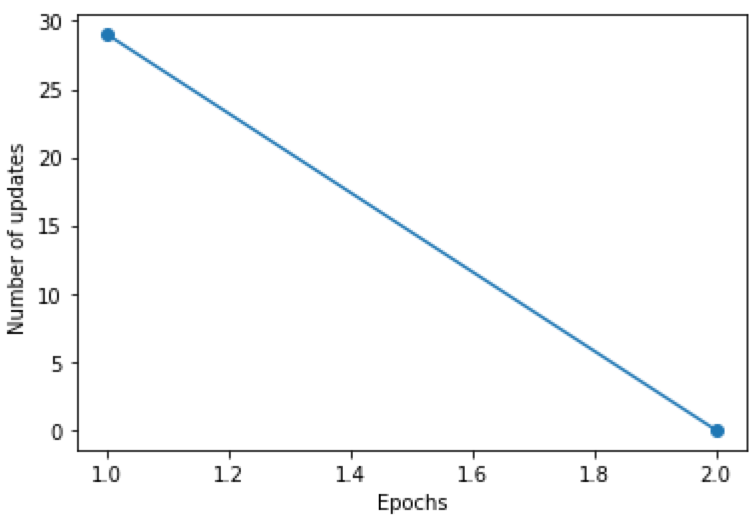
The commented code below needs a small change in order to generate a random shuffle (also called permutation) of the data). Please look up the particular functions of the code, see how they work and then do the required modification and uncomment/evaluate the code.

In [17]:
```
# establish a random shuffle
s = np.arange(len(X))
np.random.shuffle(s)
# shuffle sample
X_shuffle = X[s];
y_shuffle = y[s];
```

In [ ]:

In [18]:
```
ppn1 = Perceptron(eta=0.0001, n_iter=20, random_state=r)
ppn1.fit(X_shuffle, y_shuffle)

plt.plot(range(1, len(ppn1.errors_) + 1), ppn1.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates ')
```

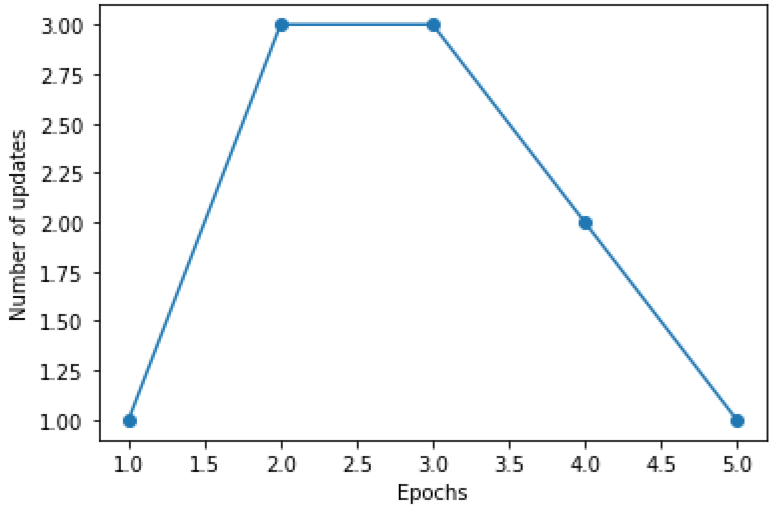
Out[18]:
```
Text(0, 0.5, 'Number of updates ')
```



Modify the code below as follows:
(i) Pick a sufficiently small η so that convergences takes 20 iterations or more
(ii) Add an extra line that fits the perceptron on the shuffled data
(iii) Plot the error for both training processes (the original, and the shuffled ata)

What do you observe?

In [19]:
```
ppn.fit(X, y)

plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
# plt.savefig('images/02_07.png', dpi=300)
plt.show()
```



-- Give your answers here

There is linear convergence for shuffled data and non linear convergence for unshuffled data

In [20]:
```
# Grader's area
maxScore = maxScore +4
# M[4,1] =
# M[4,2] =
# M[4,3] =
```

Question 5: Understanding linear transformations

Suppose we have a 2-dimensional data set. Then we transform each data point $X_j = (X_{j,1}, X_{j,2})$ as follows: $\tilde{X}_j = (aX_{j,1} - c, bX_{j,2} - c)$, where a, b, c are constant numbers. This is a linear transformation, because our transformed data comes from simple operations that use 'first powers' of the original data.

If our given data set is linearly separable, is the same true for the transformed one? In the following cells you can plot a transformed version of the Iris dataset, so that you see how it behaves (for your choice of a, b, c.) But you should also try and justify your answer in a theoretical way: if there exists a 'good' perceptron for the original data set, what would be the weights for the perceptron that works on the transformed set?

Q5 Answer)

Our original dataset is linearly seperable.Our transformed data may or may not be linearly seperable.

In [ ]:
```

```

In [21]:
```
# Transforming dataset
X_transformed = []
```

```python
a,b,c = 10,45,78
for i in X:
    Xj = (a*i[0] - c,b*i[1] - c)
    X_transformed.append(Xj)
X_transformed = np.array(X_transformed)
X_transformed
```

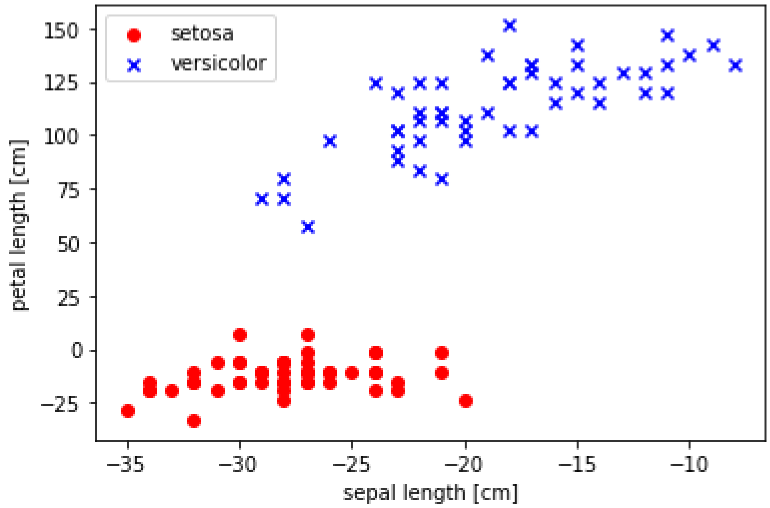
Out[21]:

```
array([[-27. , -15. ],
       [-29. , -15. ],
       [-31. , -19.5],
       [-32. , -10.5],
       [-28. , -15. ],
       [-24. ,  -1.5],
       [-32. , -15. ],
       [-28. , -10.5],
       [-34. , -15. ],
       [-29. , -10.5],
       [-24. , -10.5],
       [-30. ,  -6. ],
       [-30. , -15. ],
       [-35. , -28.5],
       [-20. , -24. ],
       [-21. , -10.5],
       [-24. , -19.5],
       [-27. , -15. ],
       [-21. ,  -1.5],
       [-27. , -10.5],
       [-24. ,  -1.5],
       [-27. , -10.5],
       [-32. , -33. ],
       [-27. ,  -1.5],
       [-30. ,   7.5],
       [-28. ,  -6. ],
       [-28. ,  -6. ],
       [-26. , -10.5],
       [-26. , -15. ],
       [-31. ,  -6. ],
       [-30. ,  -6. ],
       [-24. , -10.5],
       [-26. , -10.5],
       [-23. , -15. ],
       [-29. , -10.5],
       [-28. , -24. ],
       [-23. , -19.5],
       [-29. , -10.5],
       [-34. , -19.5],
       [-27. , -10.5],
       [-28. , -19.5],
       [-33. , -19.5],
       [-34. , -19.5],
       [-28. ,  -6. ],
       [-27. ,   7.5],
       [-30. , -15. ],
       [-27. ,  -6. ],
       [-32. , -15. ],
       [-25. , -10.5],
       [-28. , -15. ],
       [ -8. , 133.5],
       [-14. , 124.5],
       [ -9. , 142.5],
       [-23. , 102. ],
       [-13. , 129. ],
       [-21. , 124.5],
       [-15. , 133.5],
       [-29. ,  70.5],
       [-12. , 129. ],
```

```
       [-26. ,  97.5],
       [-28. ,  79.5],
       [-19. , 111. ],
       [-18. , 102. ],
       [-17. , 133.5],
       [-22. ,  84. ],
       [-11. , 120. ],
       [-22. , 124.5],
       [-20. , 106.5],
       [-16. , 124.5],
       [-22. ,  97.5],
       [-19. , 138. ],
       [-17. , 102. ],
       [-15. , 142.5],
       [-17. , 133.5],
       [-14. , 115.5],
       [-12. , 120. ],
       [-10. , 138. ],
       [-11. , 147. ],
       [-18. , 124.5],
       [-21. ,  79.5],
       [-23. ,  93. ],
       [-23. ,  88.5],
       [-20. ,  97.5],
       [-18. , 151.5],
       [-24. , 124.5],
       [-18. , 124.5],
       [-11. , 133.5],
       [-15. , 120. ],
       [-22. , 106.5],
       [-23. , 102. ],
       [-23. , 120. ],
       [-17. , 129. ],
       [-20. , 102. ],
       [-28. ,  70.5],
       [-22. , 111. ],
       [-21. , 111. ],
       [-21. , 111. ],
       [-16. , 115.5],
       [-27. ,  57. ],
       [-21. , 106.5]])
```

In [22]:

```
# ploting transformed data
plt.scatter(X_transformed[:50, 0], X_transformed[:50, 1],
            color='red', marker='o', label='setosa')
plt.scatter(X_transformed[50:100, 0], X_transformed[50:100, 1],
            color='blue', marker='x', label='versicolor')

plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')


# plt.savefig('images/02_06.png', dpi=300)
plt.show()
```



In [23]:

```python
# Finding perceptron weights of transformed set
ppn1 = Perceptron(eta=0.0001, n_iter=20, random_state=r)
ppn1.fit(X_transformed, y)

ppn1.weights
```

Out[23]:

```
[array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
```

```
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
       array([0.00181003, 0.00919535, 0.01552251]),
```

```
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
array([0.00181003, 0.00919535, 0.01552251]),
```

```
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
        array([0.00181003, 0.00919535, 0.01552251]),
```

```
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251]),
 array([0.00181003, 0.00919535, 0.01552251])]
```

In [24]:

```
# Grader's area

maxScore = maxScore +4
# M[5,1] =
```

In [ ]:

```

```

Question 6: Linear regression with numpy 1-liners

In [25]:

```
# here we initialize a random data matrix X and random numerical labels y
import numpy as np

X = np.random.randn(10,3)
y = np.random.randn(10,1)

# we also initialize a hypothetical hyperplane defined by w and b
w = np.random.randn(1,3)
b = -1
```

In [26]:

```
# (i) find the numerical labels predicted by the model (w,b) for the points in X
#     your code should be a single numpy line
#     hint: we wrote this equation for a single point x in class
#           try to generalize it by expressing everying in terms of matrices

# your code goes here

y_predicted = np.dot(X, w.T) + b
```

In [27]:

```
# (ii) find the updated weights after one application of gradient descent with lr = 0.1
#      your code should be a single numpy line

y_ = np.random.randn(10,1)
update = 0.1 * (y_ - y_predicted)
```

In [ ]:

```

```

In [28]:

```
# Grader's area

maxScore = maxScore +8
# M[6,1] =
# M[6,2] =
```

In [29]:

```
#Grader's area

rawScore = np.sum(M)
score = rawScore*100/maxScore
```