In [124]:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import numpy.matlib
import scipy
%matplotlib inline
```

In [2]:

```
def plotCurrent(X, Rnk, Kmus):
    N, D = np.shape(X)
    K = np.shape(Kmus)[0]

    InitColorMat = np.matrix([[1, 0, 0],
                              [0, 1, 0],
                              [0, 0, 1],
                              [0, 0, 0],
                              [1, 1, 0],
                              [1, 0, 1],
                              [0, 1, 1]])

    KColorMat = InitColorMat[0:K]
    colorVec = Rnk.dot(KColorMat)
    muColorVec = np.eye(K).dot(KColorMat)

    plt.scatter(X[:,0], X[:,1], edgecolors=colorVec, marker='o', facecolors
    plt.scatter(Kmus[:,0], Kmus[:,1], c=muColorVec, marker='D', s=50);
```

In [53]:

```
def calcSqDistances(X, Kmus):
    return scipy.spatial.distance.cdist(X, Kmus, 'sqeuclidean')
```

In [74]:

```
def determineRnk(sqDmat):
    return (sqDmat == np.min(sqDmat,1).reshape(-1, 1)).astype(int)
```

In [106]:

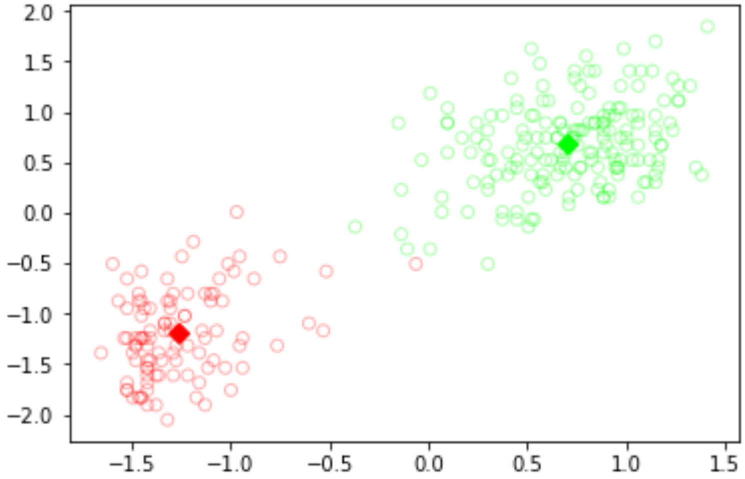
```
def recalcMus(X, Rnk):
    return Rnk.T.dot(X)/np.sum(Rnk, 0).T.reshape(-1,1)
```

In [111]:

```
def runKMeans(K, fileString):
    # Load data file specified by fileStringfrom Bishop book
    X = np.loadtxt(fileString)

    # Determine and store data set information
    N = np.shape(X)[0]
    D = np.shape(X)[1]

    # Allocate space for the K mu vectors
    Kmus = np.zeros((K, D))

    # Initialize cluster centers by randomly picking points from the data
    rndinds = np.random.permutation(N)
    Kmus = X[rndinds[:K]];

    # Specify the maximum number of iterations to allow
    maxiters = 1000;

    for iter in range(maxiters):
        # Assign each data vector to closest mu vector as per Bishop (9.2)
        # Do this by first calculating a squared distance matrix where the
        # contains the squared distance from the nth data vector to the kth

        # sqDmat will be an N-by-K matrix with the n,k entry as specfied ab
        sqDmat = calcSqDistances(X, Kmus);

        # given the matrix of squared distances, determine the closest clus
        # center for each data vector

        # R is the "responsibility" matrix
        # R will be an N-by-K matrix of binary values whose n,k entry is se
        # per Bishop (9.2)
        # Specifically, the n,k entry is 1 if point n is closest to cluster
        # and is 0 otherwise
        Rnk = determineRnk(sqDmat)

        KmusOld = Kmus
        #plotCurrent(X, Rnk, Kmus)
        #plt.show()

        # Recalculate mu values based on cluster assignments as per Bishop
        Kmus = recalcMus(X, Rnk)

        # Check to see if the cluster centers have converged.  If so, break
        if sum(abs(KmusOld.flatten() - Kmus.flatten())) < 1e-6:
            break

    plotCurrent(X,Rnk,Kmus)
```

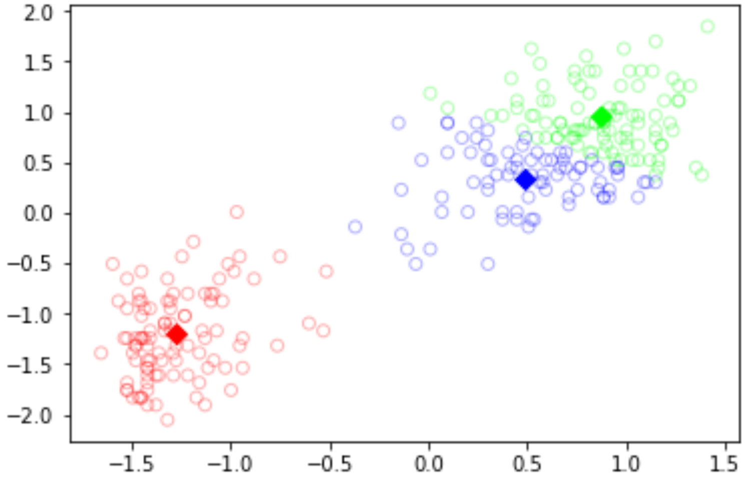
K = 2

In [112]:
```
runKMeans(2, 'scaledfaithful.txt')
```



K = 3

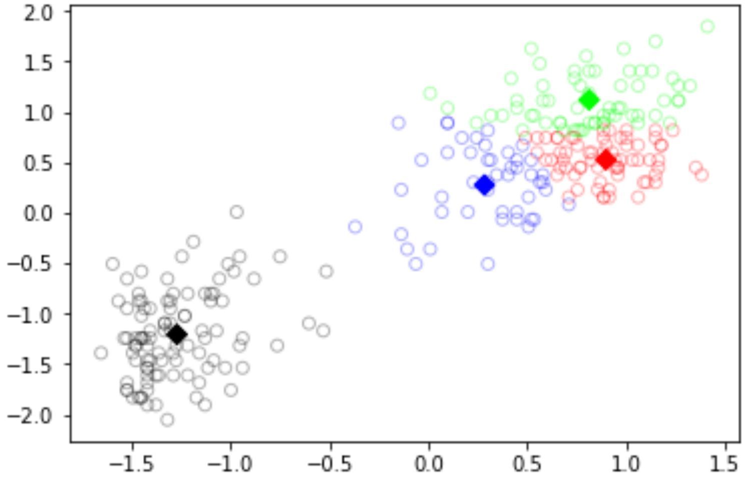
In [114]:
```
runKMeans(3, 'scaledfaithful.txt')
```



K = 4

In [123]:

```
runKMeans(4, 'scaledfaithful.txt')
```



In [ ]:

```
```