Q1.

$$J(u) = u^T S u + \lambda (1 - u^T u)$$

$$\frac{\partial J(u)}{\partial u} = \frac{\partial (u^T S u)}{\partial u} + \lambda \frac{\partial (1 - u^T u)}{\partial u}$$

$$= (S + S^T) u + \lambda \left(0 - \frac{\partial (u^T u)}{\partial u} \right)$$

$$= (S + S^T) u + \lambda (0 - 2u)$$

$$= (S + S^T) u - 2 \lambda u$$

$$= (S + S^T - 2\lambda) u$$

Q2.

$$p(\{x^{(1)}, x^{(2)}, \dots, x^{(N)}\} \mid \mu, \Sigma) = \frac{1}{(2\pi)^{ND/2}} \frac{1}{|\Sigma|^{N/2}} \exp\left\{ -\frac{1}{2} \sum_{n=1}^{N} (x^{(n)} - \mu)^T \Sigma^{-1} (x^{(n)} - \mu) \right\}$$

$$l = \ln p = -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln|\Sigma| + \left(-\frac{1}{2} \sum_{n=1}^{N} (x^{(n)} - \mu)^T \Sigma^{-1} (x^{(n)} - \mu) \right)$$

$$\frac{\partial l}{\partial u} = -\frac{1}{2} \sum_{n=1}^{N} \left(\frac{\partial x^{(n)} \Sigma^{-1} x^{(n)}}{\partial u} - \frac{\partial x^{(n)} \Sigma^{-1} \mu}{\partial u} - \frac{\partial \mu^T \Sigma^{-1} x^{(n)}}{\partial u} + \frac{\partial \mu^T \Sigma^{-1} \mu}{\partial u} \right) = 0$$

$$\sum_{n=1}^{N} (\Sigma^{-1} x^{n} + \Sigma^{-1} x^{(n)} - 2u \Sigma^{-1}) = 0$$

$$\sum_{n=1}^{N} \Sigma^{-1} x^{(n)} = N \mu \Sigma^{-1}$$

$$\mu_{ML} = \frac{1}{N} \sum_{n=1}^{N} x^{(n)} \quad \checkmark$$

Q3

In [1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy.stats import beta
%matplotlib inline
```

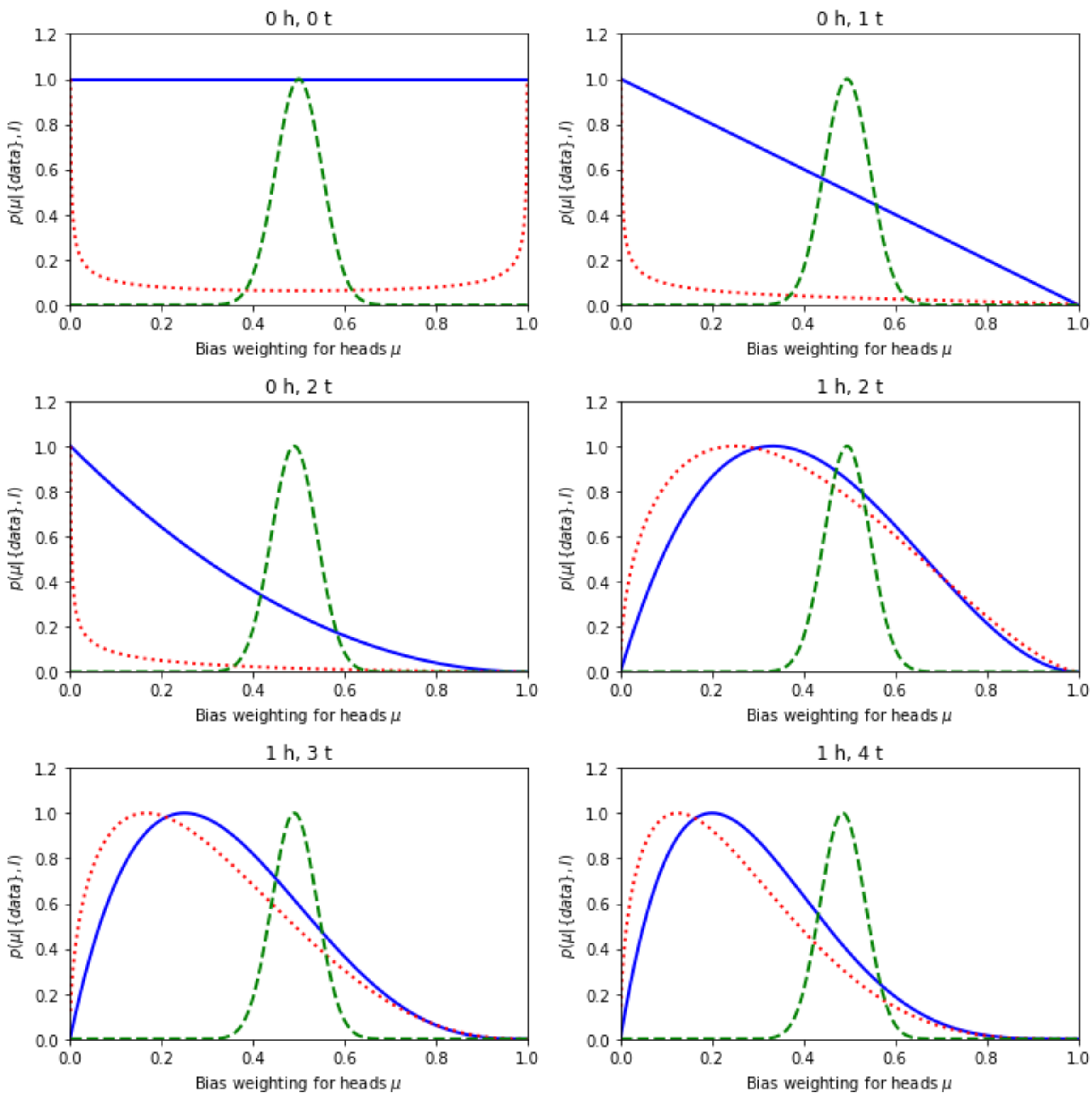
In [2]:
```
def plotbetapdfs(ab, sp_idx, tally):
    # ab is a 3-by-2 matrix containing the a,b parameters for the
    # priors/posteriors
    # Before the first flip: ab = [[1, 1], [0.5, 0.5], [50, 50]]
    #
    # sp_idx is a 3-tuple that specfies in which subplot to plot the curren
    # distributions specified by the (a,b) pairs in ab.
    #
    # tally is a 2-tuple (# heads, # tails) containing a running count of t
    # observed number of heads and tails.
    # Before the first flip: tally=(0,0)

    num_rows = np.shape(ab)[0]
    mark = ['b-','r:','g--'];

    if 'axes' not in globals():
        global fig
        global axes
        fig, axes = plt.subplots(sp_idx[0], sp_idx[1])
        fig.set_figheight(10)
        fig.set_figwidth(10)
    elif np.shape(axes)[0] != sp_idx[:2][0] or np.shape(axes)[1] != sp_idx[
        print(sp_idx[:2])
        print(list(np.shape(axes)))
        fig, axes = plt.subplots(sp_idx[0], sp_idx[1])
        fig.set_figheight(10)
        fig.set_figwidth(10)

    for row in range(num_rows):
        a = ab[row][0]
        b = ab[row][1]

        x = np.linspace(0.001, 0.999, num=999)
        y = beta.pdf(x, a, b)
        norm_y = y / max(y)

        marker = mark[row]
        ax = axes[sp_idx[2]//sp_idx[1], sp_idx[2]%sp_idx[1]]

        ax.plot(x, norm_y, mark[row], lw=2)
        ax.set_xlim([0, 1])
        ax.set_ylim([0, 1.2])
        ax.set_title(str(tally[0])+' h, '+str(tally[1])+' t')
        ax.set_xlabel('Bias weighting for heads $\mu$')
        ax.set_ylabel('$p(\mu|\{data\},I)$')

    fig.tight_layout()
    plt.close()
    return fig
```

(a)

In [3]:
```python
mu = 0.25
ab = [[1, 1], [0.5, 0.5], [50, 50]]
tally=[0,0]
sp_idx = [3,2,0]
plot = plotbetapdfs(ab, sp_idx, tally)
for i in range(5):
    sp_idx[2] = i + 1
    flip = np.random.choice([1,0], p = [mu, 1-mu])
    if flip == 0:
        tally[1] += 1
    else:
        tally[0] += 1
    ab_new = (np.array(np.transpose(np.matrix([np.array(np.transpose(np.mat
              np.array(np.transpose(np.matrix(ab)))[1] + tally[1]]))))
    plot = plotbetapdfs(ab_new,sp_idx,tally)
```

In [4]:
```python
plot
```

Out[4]:



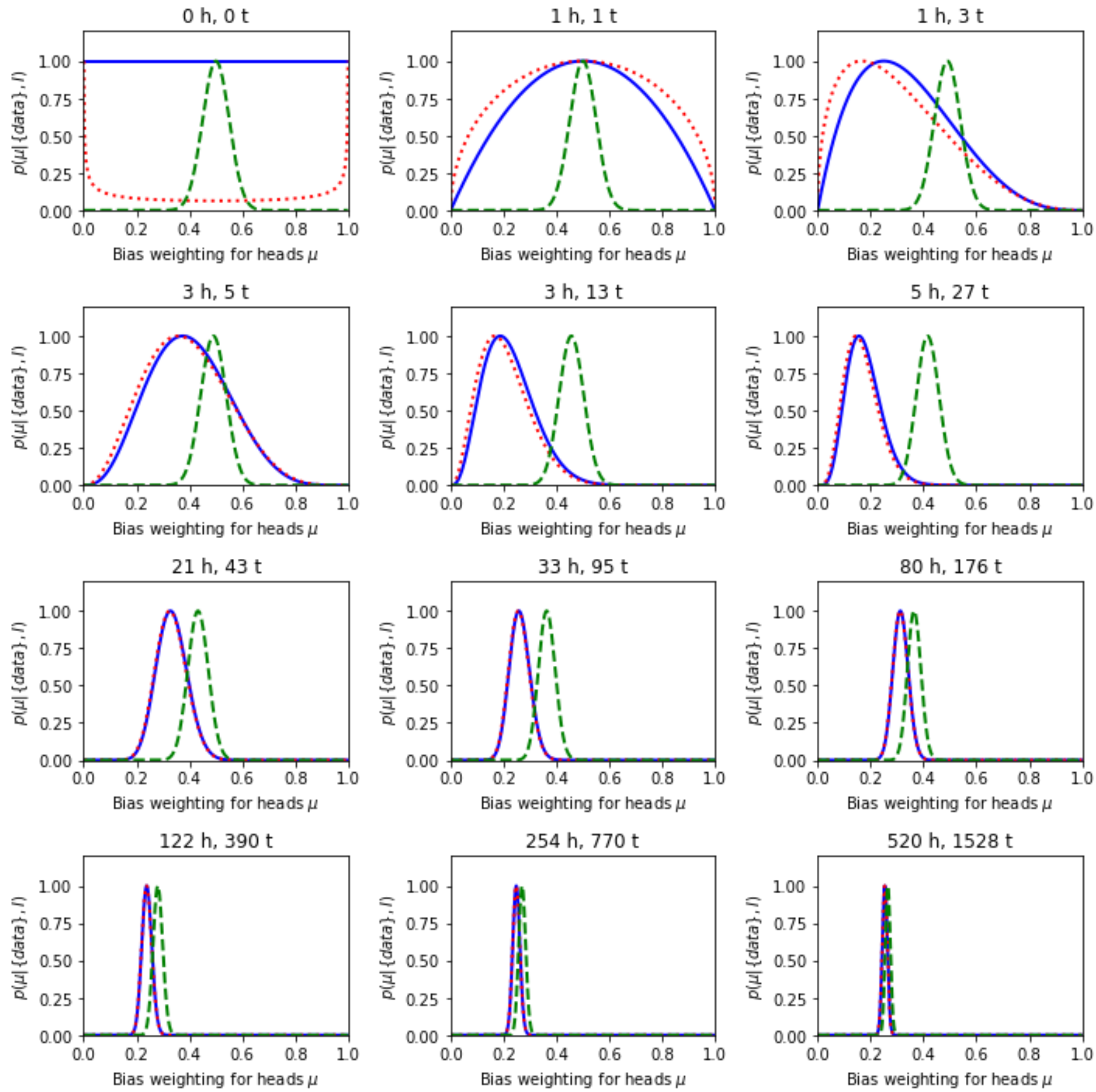
(b)

In [3]:

```
mu = 0.25
ab = [[1, 1], [0.5, 0.5], [50, 50]]
tally=[0,0]
sp_idx = [4,3,0]
plot2 = plotbetapdfs(ab, sp_idx, tally)
for i in range(11):
    sp_idx[2] = i + 1
    tally_i = [0,0]
    flip = np.random.choice([1,0], size = 2**(i+1),p = [mu, 1-mu])
    for num in flip:
        if num == 0:
            tally_i[1] += 1
        else:
            tally_i[0] += 1
    ab_new = (np.array(np.transpose(np.matrix([np.array(np.transpose(np.mat
              np.array(np.transpose(np.matrix(ab)))[1] + tally_i[1]]))))
    plot2 = plotbetapdfs(ab_new,sp_idx,tally_i)
```

In [4]:
```
plot2
```

Out[4]:



(c)

As we can see from the plots, the distributions where a and b are both set to 50 are centered at mu = 0.25, while the other two priors do not produce similar distributions. This is due to the fact that as we set both a and b to 50, we are setting much more fake data than 5, which is our sample size. With a large amount of fake data provided in the case of Beta(50,50), our sample could not easily overcome it, and mu will stay centered around 0.25. On the other hand, with Beta(1,1) or Beta(0.5,0.5), although they have the same mu, it is much easier to overcome by 5 true sample.

(d)

As we obtain large amount of true observations, the relatively small amount of fake data we set before hardly affects resulting distribution. Thus, after thousands of flips, the center of distribution will approach the true mean that is observed from the thousands of true data.

In [ ]:

Q4

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')
```

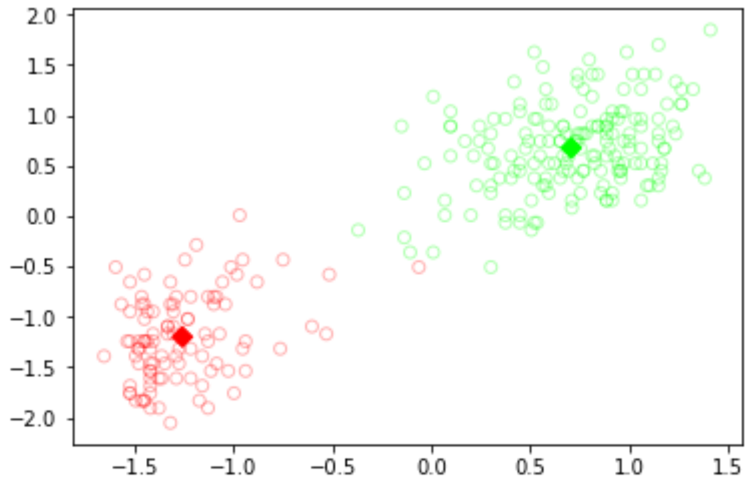
In [74]:
```
def determineRnk(sqDmat):
    return (sqDmat == np.min(sqDmat,1).reshape(-1, 1)).astype(int)
```

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)
```

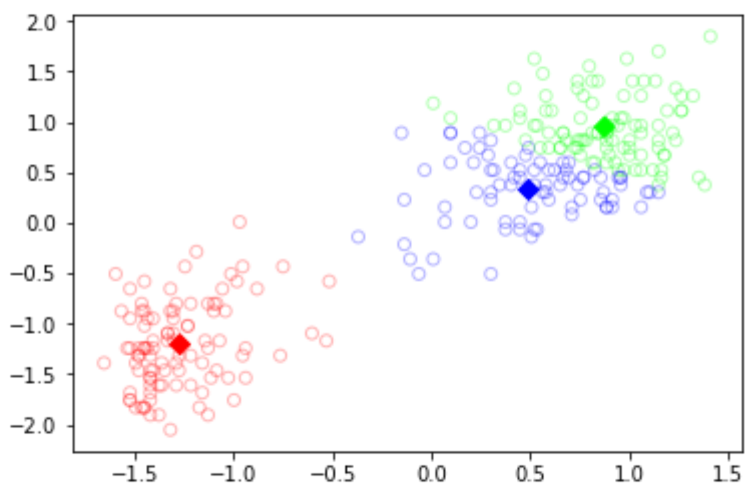
K = 2

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



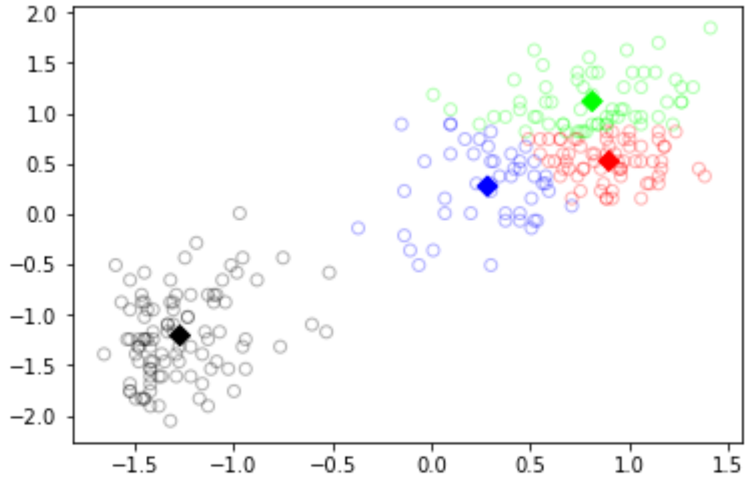
K = 3

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



K = 4

In [123]:
```
runKMeans(4, 'scaledfaithful.txt')
```



In [ ]:

Q5.

$$\frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

(a) expected value for Gaussian where $\mu = 3$, $\sigma = 1$, times $\sqrt{2\pi}$

$\Rightarrow \quad 3\sqrt{2\pi}$

(b) $\sqrt{2\pi}\, E[(x-3)^2] = \sqrt{2\pi}\, Var(x) = \sqrt{2\pi}$

(c) $\sqrt{2\pi}\, E[x^2] = \sqrt{2\pi}\, (Var(x) + E[x]^2) = 10\sqrt{2\pi}$

Q6.

$P = X + a$

$$\begin{aligned} E[P] &= E[X + a] \\ &= \sum [x_i + a]\, P(x_i) \\ &= \sum x_i\, p(x) + a \sum P(x_i) \\ &= E[x] + a \\ &= m + a \end{aligned}$$

$$\begin{aligned} Var[P] &= Var[X + a] \\ &= \sum (x_i + a - E[x + a])^2 P(x_i) \\ &= \sum (x_i + a - (m + a))^2 P(x_i) \\ &= \sum (x_i - m)^2 p(x_i) \\ &= Var[X] \\ &= \sigma^2 \end{aligned}$$

$Q = bX$

$$\begin{aligned} E[Q] &= E[bx] \\ &= \sum b x_i\, P(x_i) \\ &= b \sum x_i\, P(x_i) \\ &= b\, E[x] \\ &= bm \end{aligned}$$

$$\begin{aligned} Var[Q] &= Var[bX] \\ &= \sum (b x_i - E[bx])^2 P(x_i) \\ &= \sum b^2 (x_i - m)^2 P(x_i) \\ &= b^2\, Var[X] \\ &= b^2 \sigma^2 \end{aligned}$$