1) Using one or both of the above rules, for $J(u) = u^T Su + \lambda(1 - u^T u)$ determine the value of the derivative $\frac{dj(u)}{du}$.

$$
\begin{aligned}
&\frac{d}{du}(u^T Su + \lambda(1 - u^T u)) \\
&\frac{d}{du}(u^T Su) + \frac{d}{du}(\lambda(1 - u^T u)) \\
&= 2Su - \frac{d}{du}(\lambda u^T u)) \\
&= 2Su - 2\lambda u
\end{aligned}
$$

2)

- set ln of the function

$$ln\ p(X|\mu,\Sigma) = -\frac{ND}{2}\ ln(2\pi) - \frac{N}{2}\ ln|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)$$

- take derivative

$$\frac{d}{du}\left[-\frac{ND}{2}\ ln(2\pi) - \frac{N}{2}\ ln|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right]$$

$$\frac{d}{du}\left[-\frac{ND}{2}\ ln(2\pi)\right] - \frac{d}{du}\left[\frac{N}{2}\ ln|\Sigma|\right] - \frac{d}{du}\left[\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right]$$

$$0 + 0 - \frac{d}{du}\left[\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right]$$

$$\sum_{n=1}^{N}(\Sigma+\Sigma^T)^{-1}(x_n-\mu)$$

- set equal to zero & solve for mu

$$\sum_{n=1}^{N}\frac{(x_n-\mu)}{(\Sigma+\Sigma^T)} = 0$$

$$\sum_{n=1}^{N}(x_n-\mu) = 0$$

$$\sum_{n=1}^{N}x_n - \sum_{n=1}^{N}\mu = 0$$

$$\sum_{n=1}^{N}x_n - N\mu = 0$$

$$\sum_{n=1}^{N}x_n = N\mu$$

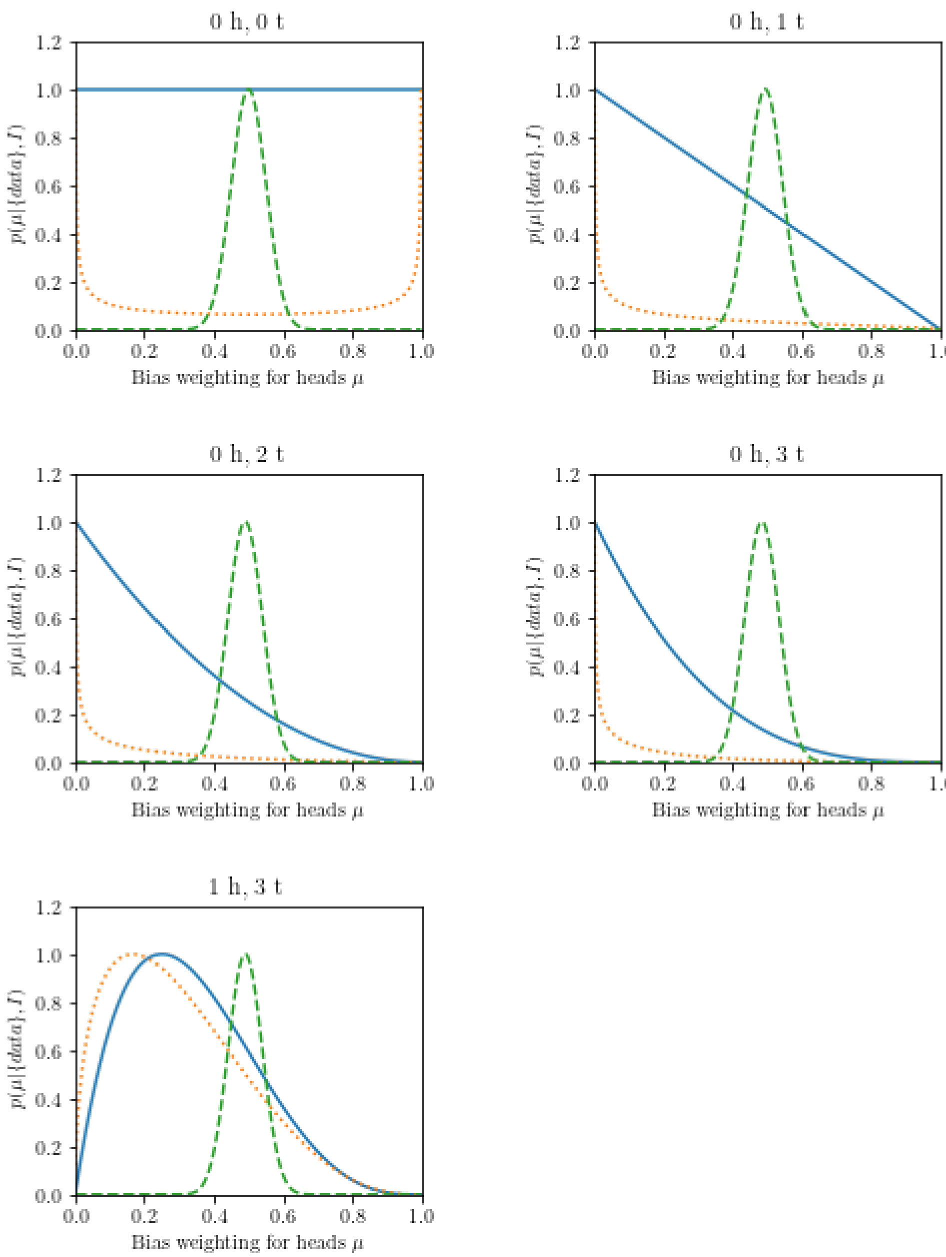
$$\mu = \frac{1}{N}\sum_{n=1}^{N}x_n$$

In [9]:
```
#3)
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta
from scipy.spatial import distance
import time
```

In [10]:
```
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[0,:]=[1   1];
    #                       ab[1,:]=[.5 .5];
    #                       ab[2,:]=[50 50];

    #sp_idx is a 3-element array that specfies in which subplot to plot
 the
    #current distributions specified by the (a,b) pairs in ab.

    #tally is a 2-element array (# heads, # tails) containing a running
 count
    #of the observed number of heads and tails.
    #Before the first flip: tally=[0 0]
    num_rows = ab.shape[0]
    xs = np.arange(.001, 1, .001)

    plt.subplots_adjust(left=0, right=1, bottom=0, top=2, wspace=0.5, hs
pace=0.5)
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    plt.subplot(sp_idx[0], sp_idx[1], sp_idx[2])

    mark = ['-',':','--']

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

        vals = beta.pdf(xs, a, b)
        norm_vals = vals/np.amax(vals, axis=0)
        plt.plot(xs, norm_vals, marker)
        axes = plt.gca()
        axes.set_xlim([0, 1])
        axes.set_ylim([0, 1.2])
        plt.title('{:d} h, {:d} t'.format(*tally))
        plt.xlabel(r'Bias weighting for heads $\mu$')
        plt.ylabel(r'$p(\mu|\{data\},I)$')
```

In [11]:
```
n = 1
p = .25
flips = np.random.binomial(n, p, 5)

print("\ns shape:" + str(flips.shape))
print("sp_index type: " + str(type(flips)))
print(flips)

#num_ones
num_heads = (flips == 1).sum()
print("num_ones: " + str(num_heads))

#num_zeros
num_tails = (flips == 0).sum()
print("num_zeros: " + str(num_tails))

#tally = [head, tail]
tally = [num_heads, num_tails]

#print("\ns tally:" + str(tally.shape))
print("\ntally type: " + str(type(tally)))
print("[heads, tails]: " + str(tally))


#ab: 3-by-2 matrix containing the a,b parameters
# ___ Represents [1, 1]
# ... Represents [0.5, 0.5]
# --- Represents [50, 50]
ab = np.matrix([[1, 1], [0.5, 0.5], [50, 50]])
print("\nab type: " + str(type(ab)))
print("ab shape: " + str(ab.shape[0]))
print("Original ab: ")
print(ab)

newTally = [0, 0]

#sp_idx is a 3-element array that specfies in which subplot to plot the
 current
#distributions specified by the (a,b) pairs in ab.
sp_idx = [3, 2, 1]

#Loop through the array of the simulation and add on to the collection o
f subplot matrix.
for i in flips:
    plotbetapdfs(ab, sp_idx, newTally)
    if(i == 1):
        newTally[0] = newTally[0] + 1
        ab[:, 0] = ab[:, 0] + np.ones((3,1))
    if(i == 0):
        newTally[1] = newTally[1] + 1
        ab[:, 1] = ab[:, 1] + np.ones((3,1))
    sp_idx[2] = sp_idx[2] + 1
```

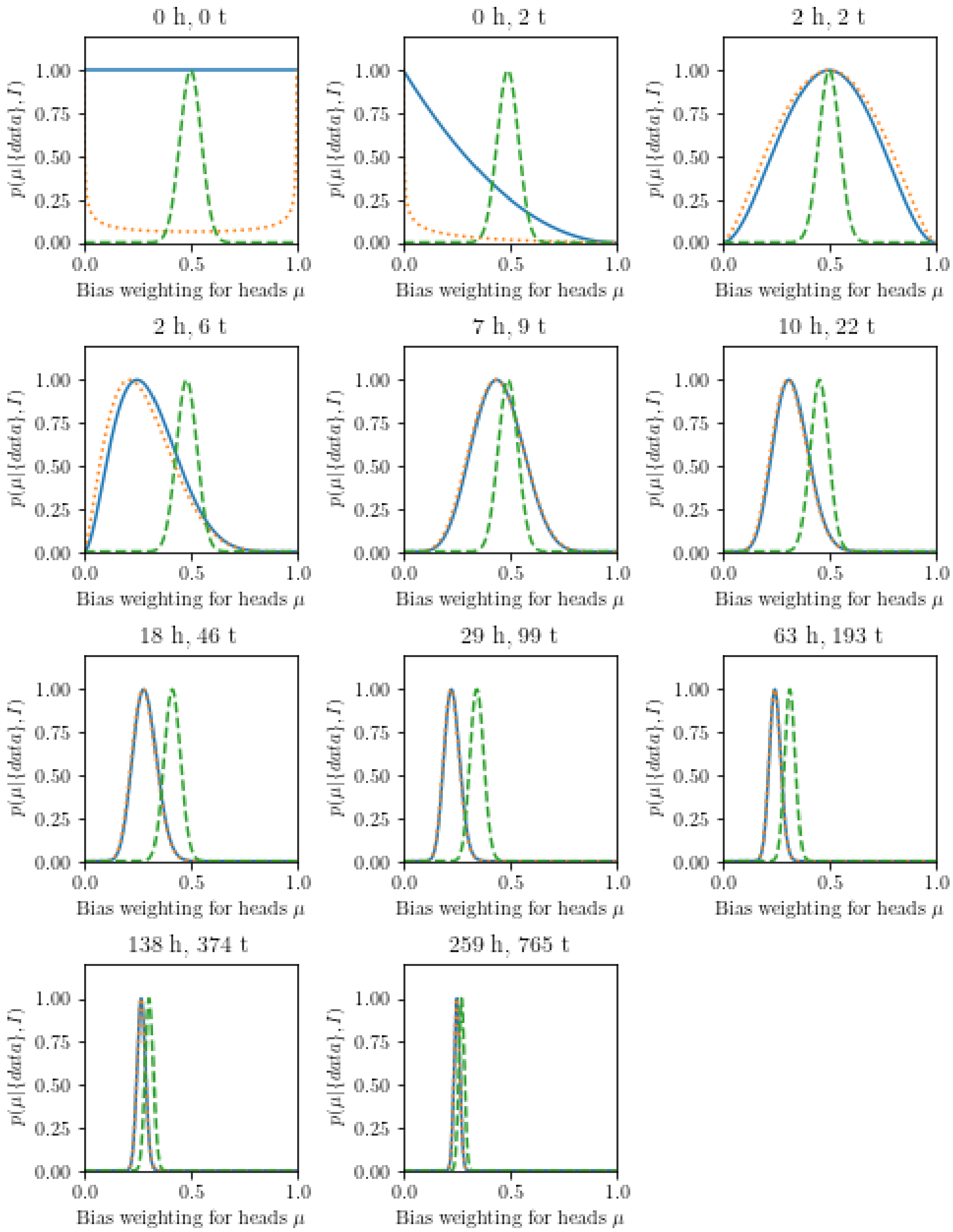
```
s shape:(5,)
sp_index type: <class 'numpy.ndarray'>
[0 0 0 1 0]
num_ones: 1
num_zeros: 4

tally type: <class 'list'>
[heads, tails]: [1, 4]

ab type: <class 'numpy.matrixlib.defmatrix.matrix'>
ab shape: 3
Original ab:
[[ 1.   1. ]
 [ 0.5  0.5]
 [50.  50. ]]
```



In [12]:
```
n = 1
p = .25
flips = np.random.binomial(n, p, 2048)

print("\nflips shape:" + str(flips.shape))
print(flips)

#num_ones
num_heads = (flips == 1).sum()
print("num_ones: " + str(num_heads))

#num_zeros
num_tails = (flips == 0).sum()
print("num_zeros: " + str(num_tails))

#tally = [head, tail]
tally = [num_heads, num_tails]

#print("\ns tally:" + str(tally.shape))
print("\ntally type: " + str(type(tally)))
print("[heads, tails]: " + str(tally))


#ab: 3-by-2 matrix containing the a,b parameters
# ___ Represents [1, 1]
# ... Represents [0.5, 0.5]
# --- Represents [50, 50]
ab = np.matrix([[1, 1], [0.5, 0.5], [50, 50]])
print("\nab type: " + str(type(ab)))
print("ab shape: " + str(ab.shape[0]))
print("Original ab: ")
print(ab)

newTally = [0, 0]

#sp_idx is a 3-element array that specfies in which subplot to plot the
 current
#distributions specified by the (a,b) pairs in ab.
sp_idx = [4, 3, 1]
print("\nsp_index type: " + str(type(sp_idx)))

pow2List = [0, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]
count = 0
countPow = 0

print("ENTERING LOOP")
#Loop through the array of the simulation and add on to the collection o
f subplot matrix.
for i in flips:
    if(count == pow2List[countPow]):
        print("countPow: " + str(countPow))
        print("count: " + str(count))
        print("sp_idx: " + str(sp_idx))
        #print("i: " + str(i))
        countPow = countPow + 1
        sp_idx[2] = countPow
```

```
        plotbetapdfs(ab, sp_idx, newTally)
    count = count + 1
    if(i == 1):
        #print("add to left side")
        newTally[0] = newTally[0] + 1
        ab[:, 0] = ab[:, 0] + np.ones((3,1))
    if(i == 0):
        #print("add to right side")
        newTally[1] = newTally[1] + 1
        ab[:, 1] = ab[:, 1] + np.ones((3,1))
```

```
flips shape:(2048,)
[0 0 1 ... 1 0 0]
num_ones: 508
num_zeros: 1540

tally type: <class 'list'>
[heads, tails]: [508, 1540]

ab type: <class 'numpy.matrixlib.defmatrix.matrix'>
ab shape: 3
Original ab:
[[ 1.   1. ]
 [ 0.5  0.5]
 [50.  50. ]]

sp_index type: <class 'list'>
ENTERING LOOP
countPow: 0
count: 0
sp_idx: [4, 3, 1]
countPow: 1
count: 2
sp_idx: [4, 3, 1]
countPow: 2
count: 4
sp_idx: [4, 3, 2]
countPow: 3
count: 8
sp_idx: [4, 3, 3]
countPow: 4
count: 16
sp_idx: [4, 3, 4]
countPow: 5
count: 32
sp_idx: [4, 3, 5]
countPow: 6
count: 64
sp_idx: [4, 3, 6]
countPow: 7
count: 128
sp_idx: [4, 3, 7]
countPow: 8
count: 256
sp_idx: [4, 3, 8]
countPow: 9
count: 512
sp_idx: [4, 3, 9]
countPow: 10
count: 1024
sp_idx: [4, 3, 10]
```



3c.

The parameters a and b going into the beta distribution are fake (posterior) coin flips. These posterior coin flips have a stronger belief when the values of a and b are higher. The the probability of the coin flip was biased with a p = 0.25, the convergence should be faster for smaller values of a and b.

3d.

After thousands of flips the Bayes will converge to the 0.25 probability of heads due to the law of large numbers. Large values of a and b from the beta distribution don't have much of an impact.

In [13]:
```
#4)
def plotCurrent(X, Rnk, Kmus):
    N, D = X.shape
    K = Kmus.shape[0]

    InitColorMat = np.array([[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 = np.dot(Rnk, KColorMat)
    muColorVec = np.dot(np.eye(K), KColorMat)
    plt.scatter(X[:,0], X[:,1], c=colorVec)

    plt.scatter(Kmus[:,0], Kmus[:,1], s=200, c=muColorVec, marker='d')
    plt.axis('equal')
    plt.show()
```

In [14]:
```
def runKMeans(K,fileString):
    #load data file specified by fileString from Bishop book
    X = np.loadtxt(fileString, dtype='float')
    print(X)
    #determine and store data set information
    N, D = X.shape

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

    #initialize cluster centers by randomly picking points from the data
    rand_inds = np.random.permutation(N)
    Kmus = X[rand_inds[0: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 th
e n,k entry
        #contains the squared distance from the nth data vector to the k
th mu vector

        #sqDmat will be an N-by-K matrix with the n,k entry as specfied
 above
        sqDmat = calcSqDistances(X, Kmus)
        print(sqDmat)
        #given the matrix of squared distances, determine the closest cl
uster
        #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
 set as
        #per Bishop (9.2)
        #Specifically, the n,k entry is 1 if point n is closest to clust
er k,
        #and is 0 otherwise
        Rnk = determineRnk(sqDmat)

        KmusOld = Kmus
        plotCurrent(X, Rnk, Kmus)
        time.sleep(1)

        #recalculate mu values based on cluster assignments as per Bisho
p (9.4)
        Kmus = recalcMus(X, Rnk)

        #check to see if the cluster centers have converged.  If so, bre
ak.
        if np.sum(np.abs(KmusOld.reshape((-1, 1)) - Kmus.reshape((-1, 1
)))) < 1e-6:
            print(iter)
            break
```

```
    plotCurrent(X, Rnk, Kmus)
```

In [15]:
```
def recalcMus(X,rank):
    N = size(X,1)
    K = size(rank, 2)
    D = size(X, 2)
    sum_of_cluster = dot(rank.T,X)
    num_of_cluster = rank.sum().T
    normalMat = np.tile(num_of_cluster, np.ones(1),D)
    Kmus = divide(sum_of_cluster,normalMat)
```

In [6]:
```
def calcSqDistances(X, Kmus):
    dists = -2 * np.dot(X, np.transpose(Kmus)) + np.sum(Kmus**2,axis=1)
+ np.sum(X**2, axis=1)[:, np.newaxis]
    return dists
```

In [7]:
```
def determineRnk(sqDmat):

    # calculate the label for each cluster
    # 1 for belong, 0 for not belong

    N = sqDmat.shape[0]
    print(N)
    K = sqDmat.shape[1]
    print(K)

    RnKMat = np.zeros((N, K))
    print(RnKMat)

    print("RnkMat type: " + str(type(RnKMat)))
    print("RnkMat size: " + str(RnKMat.shape))

    result1 = [min(row) for row in sqDmat]
    print (result1)
    positionVec = range(n)

    for i in positionVec:
        idxVec = N * (result1 - 1) + i

    RnKMat[idxVec] = 1
    return RnkMat
```

In [8]:
```
runKMeans(3,"./faithful.txt")
```

```
[[ 3.6    79.    ]
 [ 1.8    54.    ]
 [ 3.333 74.    ]
 [ 2.283 62.    ]
 [ 4.533 85.    ]
 [ 2.883 55.    ]
 [ 4.7    88.    ]]
[[  36.870489  576.514089    0.        ]
 [ 968.469289    2.172889  628.24     ]
 [ 122.44      361.2025     25.071289]
 [ 534.0625     49.36      290.734489]
 [   0.        902.7225     36.870489]
 [ 902.7225      0.        576.514089]
 [   9.027889 1092.301489   82.21     ]]
7
3
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
RnkMat type: <class 'numpy.ndarray'>
RnkMat size: (7, 3)
[0.0, 2.1728889999994863, 25.071288999999524, 49.36000000000013, 0.0,
0.0, 9.027889000000869]

--------------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-8-90c46fb45b9e> in <module>()
----> 1 runKMeans(3,"./faithful.txt")

<ipython-input-4-6735975a8274> in runKMeans(K, fileString)
     32         #Specifically, the n,k entry is 1 if point n is closest
to cluster k,
     33         #and is 0 otherwise
---> 34         Rnk = determineRnk(sqDmat)
     35 
     36         KmusOld = Kmus

<ipython-input-7-24b31854ad5a> in determineRnk(sqDmat)
     17     result1 = [min(row) for row in sqDmat]
     18     print (result1)
---> 19     positionVec = range(n)
     20 
     21     for i in positionVec:

NameError: name 'n' is not defined
```

5)

- $\int xe^{\frac{-(x-3)^2}{2}}\,dx$

we know that:

$$E(x) = \int \frac{x}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}\,dx$$
$$\rightarrow \int xe^{\frac{-(x-3)^2}{2}}\,dx = 3\sqrt{2\pi}$$

- $\int (x-3)^2 e^{\frac{-(x-3)^2}{2}}\,dx$

we know that:

$$E(x-3)^2 = \int \frac{(x-3)^2}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}\,dx$$
$$\rightarrow \int (x-3)^2 e^{\frac{-(x-3)^2}{2}}\,dx = \sqrt{2\pi}$$

- $\int x^2 e^{\frac{-(x-3)^2}{2}}\,dx$

we know that:

$$E(x^2) = \int \frac{x^2}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}\,dx$$

$$\rightarrow \int x^2 e^{\frac{-(x-3)^2}{2}}\,dx = 10\sqrt{2\pi}$$

6)

$P = X + a$

$Q = bX$

$E(P) = E(X+a) = E(X) + a = m + a$ $E(P - E(P))^2 = E(X + a - (m + a))^2 = E(X - m)^2$
$E(X - m)^2 = \sigma^2$

$var(P) = \sigma^2$

$E(Q) = E(bX) = bE(X) = bm$
$E(Q - E(Q))^2 = E(bX - bm)^2 = E(b^2(X - m)^2) = b^2 E(X - m)^2 = b^2\sigma^2$ $var(Q) = b^2\sigma^2$

```
In [ ]:
```