Expectation-Maximization

In this assignment we will be using the Expectation Maximization method to estimate the parameters of the same three coin experiment as in the theoretical part. We will examine the behavior of the algorithm for various combinations of parameters.

Description of the Experiment

The following procedure generates the data for the three coin experiment.

The parameters are:

- λ := The probability of heads on the hidden coin H.
- p_1 := The probability of heads on coin A.
- p_2 := The probability of heads on coin B.

Each of the N samples is collected the following way:

- The secret coin (H) is tossed.
- If the result is heads, coin A is tossed M times and the results are recorded.
- If the result is tails, coin B is tossed M times and the results are recorded.

Heads are recorded as 1.

Tails are recorded as 0.

The data is returned as an $N \times M$ matrix, where each of the N rows correspond to the trials and contains the results of the corresponding sample (generated either by coin A or by coin B).

Description of Provided Functions

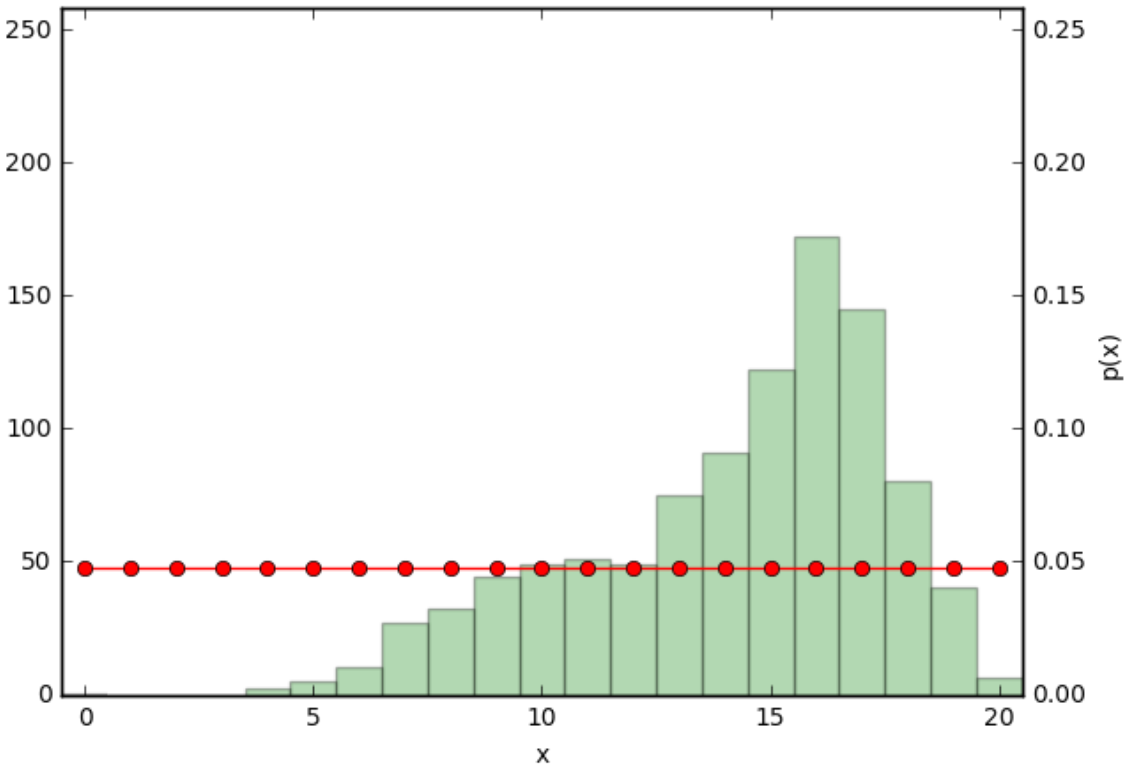
Three functions are provided for your convenience:

- **`utils.generateData(lambda,p1,p2,N,M)`:** Performs the experiment N times with coin parameters specified as argument and returns the results in a $N \times M$ matrix.
- **`utils.unknownData()`** Returns a dataset of size $N \times M$ where generation parameters are unknown.
- **`utils.plot(data,distribution)`:** Plot a histogram of the number of heads per trial along with the probability distribution. This function will be used to visualize the progress of the EM algorithm at every iteration.

An example of use of these two functions is given below:

In [72]:

```
%matplotlib inline
import numpy as np,utils
data = utils.generateData(0.5,0.8,0.2,15,5)
data = utils.unknownData()
N = data.shape[0]
M = data.shape[1]
utils.plot(data,np.ones([data.shape[1]+1])/(data.shape[1]+1))
```



Calculate the Log-Likelihood (10 P)

Implement a function which calculates the log likelihood for a given dataset and parameters. The log-likelihood is given by:

$$LL = \frac{1}{N}\sum_{i=1}^{N} \log \sum_{z \in \{\text{heads,tails}\}} P(X = x_i, Z = z \mid \theta)$$

$$= \frac{1}{N}\sum_{i=1}^{N} \log \Big[\lambda \cdot p_1^{h(x_i)} \cdot (1-p_1)^{t(x_i)} + (1-\lambda) \cdot p_2^{h(x_i)} \cdot (1-p_2)^{t(x_i)}\Big]$$

where $h(x_i)$ and $t(x_i)$ denote the number of heads and tails in sample i, respectively. Note that we take the averaged log-likelihood over all trials, hence the multiplicative term $\frac{1}{N}$ in front.

In [73]:

```
import utils
%matplotlib inline
import math
from operator import mul
from fractions import Fraction
criterion = False # (to be set to True)
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value

def getbinom(n):
    def binom(k):
        if (k > n-k):
            k = n-k
        return reduce(mul, (Fraction(n-i,i+1) for i in range(k)),1)
    return binom

def loglikelihood(data,lam,p1,p2):
    numHeads = np.sum(data, axis=1)
    tail = np.power(p1,numHeads) * np.power(1-p1,M-numHeads)
    notTail = np.power(p2,numHeads) * np.power(1-p2,M-numHeads)
    numComb = np.array(list(map(float, map(getbinom(M), numHeads))))
    LL = np.log(lam * numComb * tail + (1-lam) * numComb * notTail)
    return np.mean(LL, axis=0)
```

Implementing and Running the EM Algorithm (30 P)

Implement a function which iteratively determines the values of λ, p_1 and p_2. The function starts with some initial estimates for the parameters and returns the results of the method for those parameters.

In each iteration, the following update rules are used for the parameters:

$$\lambda^{new} = \frac{E(\#heads(coin\_H))}{\#throws(coin\_H)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

$$p_1^{new} = \frac{E(\#heads(coin\_A))}{E(\#throws(coin\_A))} = \frac{\sum_{i=1}^{N} R_1(i)h(x_i)}{M\sum_{i=1}^{N} R_1(i)}$$

$$p_2^{new} = \frac{E(\#heads(coin\_B))}{E(\#throws(coin\_B))} = \frac{\sum_{i=1}^{N} R_2(i)h(x_i)}{M\sum_{i=1}^{N} R_2(i)}$$

where $h(x_i)$ and $t(x_i)$ denote the number of heads and tails in sample i, respectively, and

$$R_1(i) = \frac{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

$$R_2(i) = \frac{(1-\lambda) p_2^{h(x_i)}(1-p_2)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

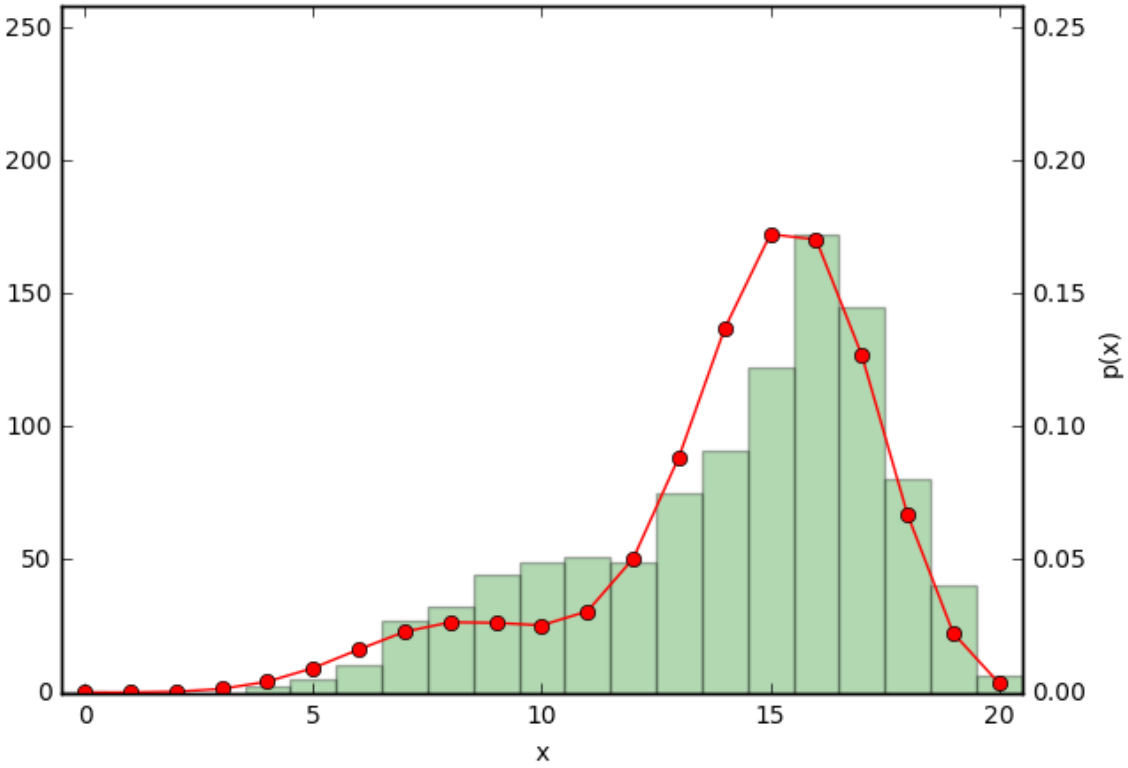
TODO:

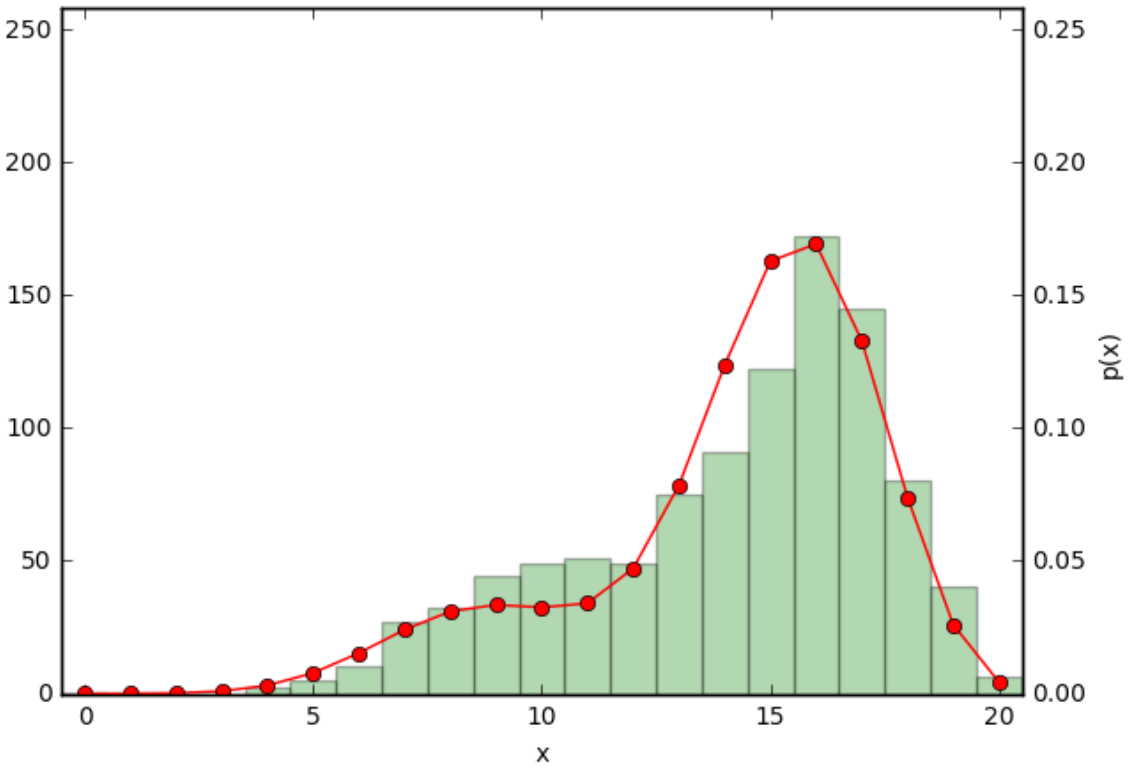
- **Implement the EM learning procedure.**
- **Use as stopping criterion the improvement of log-likelihood between two iterations to be smaller than 0.001.**
- **Run the EM procedure on the data returned by function `utils.unknownData()`. Use as an initial solution for your model the parameters $\lambda = 0.5, p_1 = 0.25, p_2 = 0.75$.**
- **At each iteration of the EM procedure, print the log-likelihood and the value of your model parameters, and plot the learned probability distribution using the function utils.plot().**

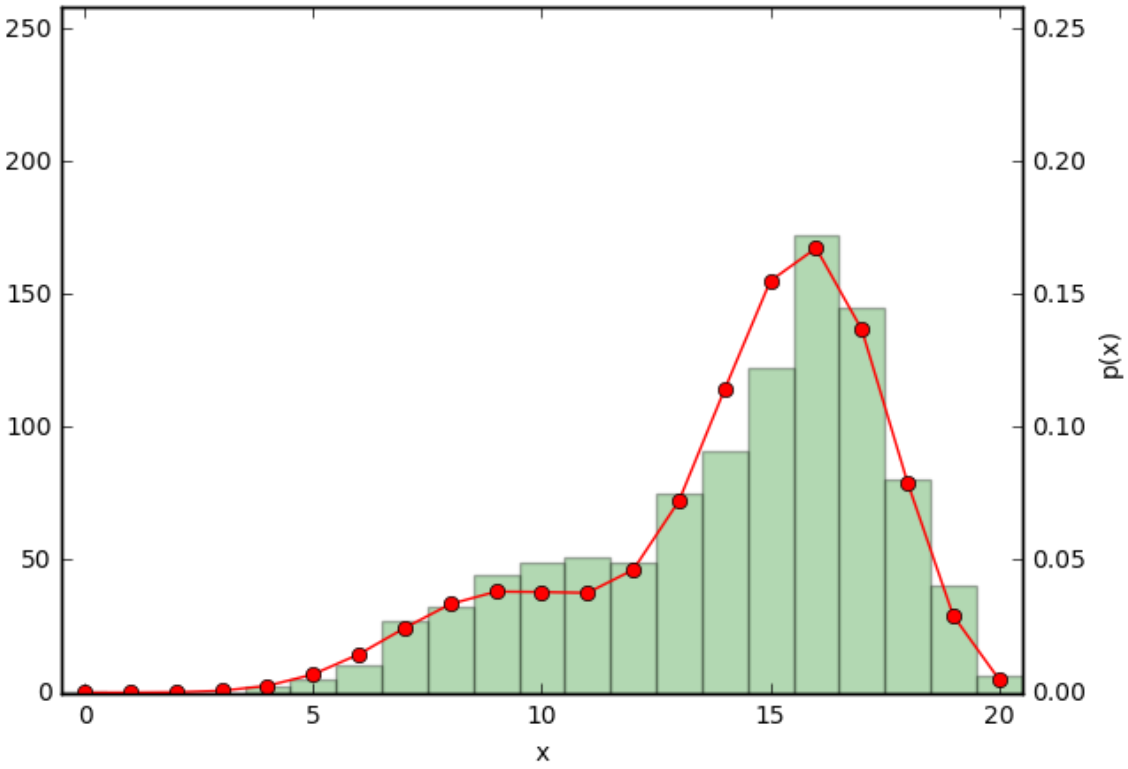
In [75]:

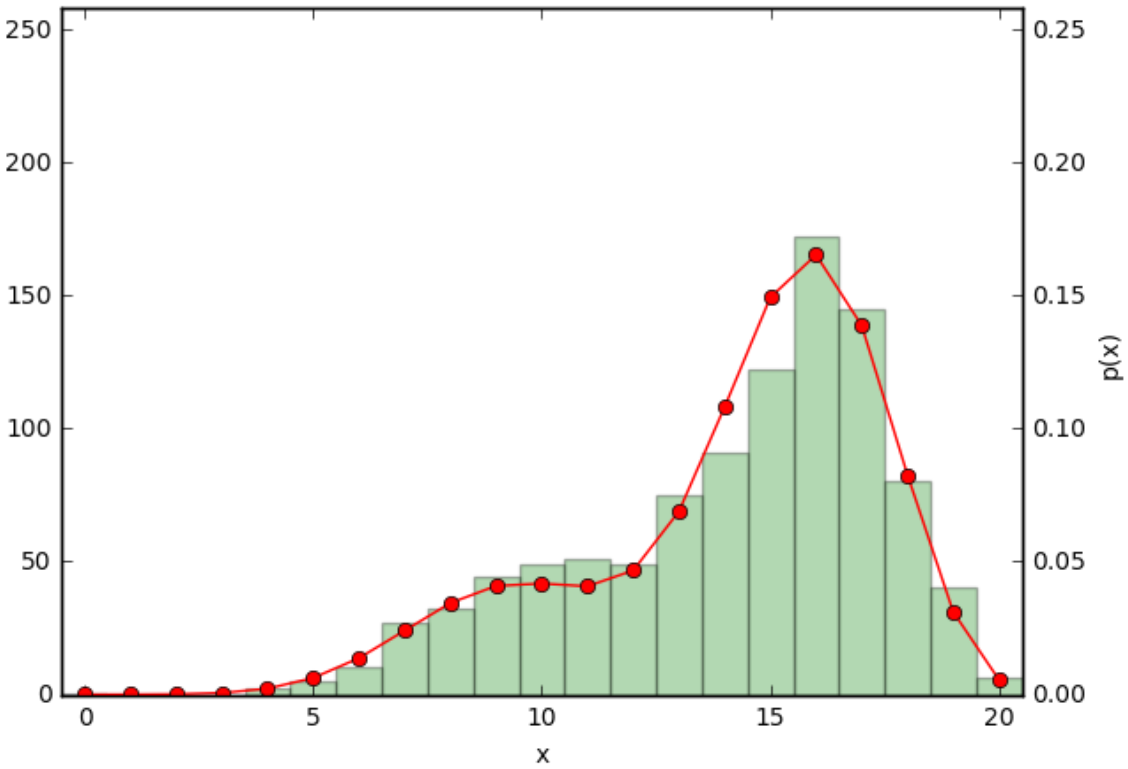
```
def computedist(lam,p1,p2,M):
    x = np.arange(M+1)
    tails = np.power(p1,x)*np.power(1-p1,M-x)
    notTails = np.power(p2,x)*np.power(1-p2,M-x)
    numcomb = list(map(float, map(getbinom(M), x)))
    numcomb = np.array(numcomb)
    dist = lam*numcomb*tails + (1-lam)*numcomb * notTails
    return dist

def CalculateLambda(lam,p1,p2,hx,tx):
    return (lam*pow(p1,hx)*(pow(1-p1,tx)))/((lam*pow(p1,hx)*(pow(1-p1,tx)))+((1-lam)*po
w(p2,hx)*(pow(1-p2,tx))))

def CalculateR1R2(lam,p1,p2,hx,tx):
    denominator = ((lam* pow(p1,hx)*pow((1-p1),tx))+((1-lam)* pow(p2,hx)*pow((1-
p2),tx)))
    R1 = (lam*(p1**hx)*((1-p1)**tx))/denominator
    R2 = ((1-lam)* pow(p2,hx)*pow((1-p2),tx))/denominator
    return R1,R2

def EMLearning(lam,p1,p2,criterion,printlastone):
    LLPrev = 0
    LL = -1
    it = 0
    while (criterion == False):
        it+=1
        if(abs(LL-LLPrev) <= 0.001):
            criterion = True
            if(printlastone==True):
                utils.plot(data,computedist(lam, p1, p2, M))
        else:
            R1hx = 0
            R2hx = 0
            R1i = 0
            R2i = 0
            lami = 0
            for i,row in enumerate(data):
                hxi = row.nonzero()[0].shape[0]
                txi = M - hxi
                R1,R2 = CalculateR1R2(lam,p1,p2,hxi,txi)
                R1hx+= R1*hxi
                R2hx+=R2*hxi
                R1i+=R1
                R2i+=R2
                lami+=CalculateLambda(lam,p1,p2,hxi,txi)
            p1 = R1hx/(M*R1i)
            p2 = R2hx/(M*R2i)
            lam = lami/N
            LLPrev = LL
            LL = loglikelihood(data,lam,p1,p2)
            print ('it:%2d  lambda: %.2f  p1: %.2f  p2: %.2f  log-likelihood: %.3f'%
(it, lam, p1, p2, LL))
            if(printlastone ==False):
                utils.plot(data,computedist(lam, p1, p2, M))

lam = 0.5
p1  = 0.25
p2  = 0.75
EMLearning(lam,p1,p2,criterion,False)
```

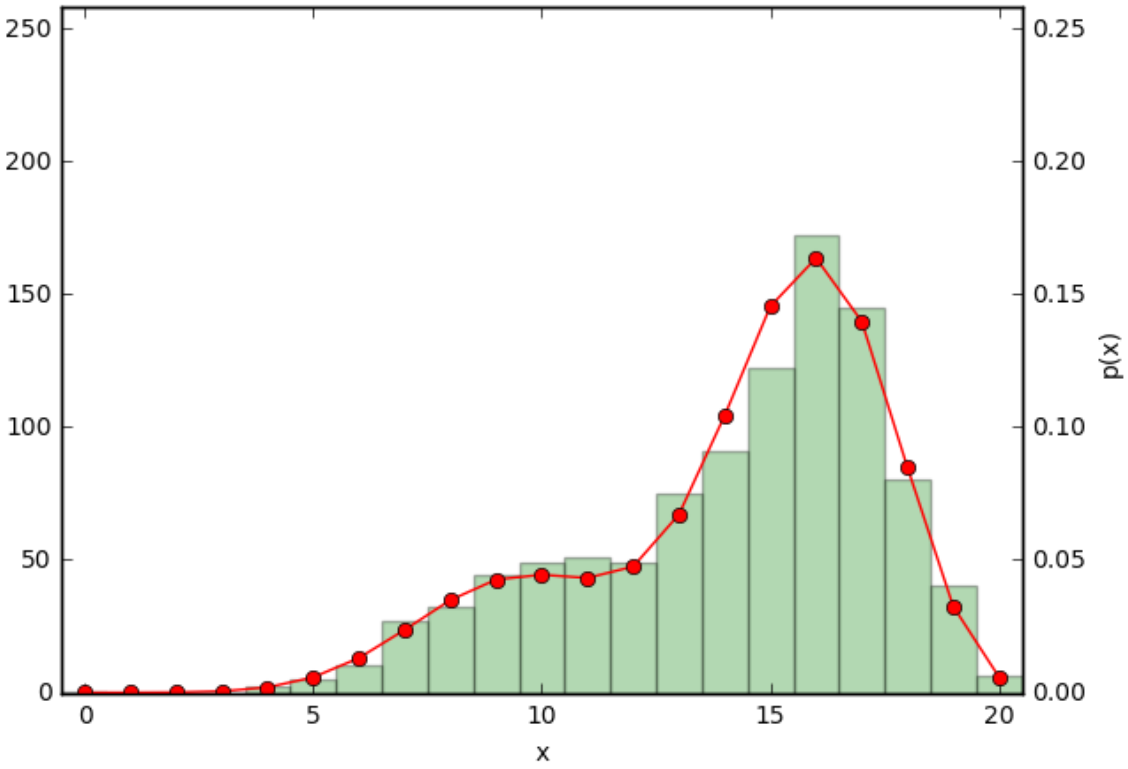
```
it: 1  lambda: 0.15  p1: 0.41  p2: 0.76  log-likelihood: -2.543
it: 2  lambda: 0.18  p1: 0.44  p2: 0.77  log-likelihood: -2.518
it: 3  lambda: 0.21  p1: 0.46  p2: 0.78  log-likelihood: -2.506
it: 4  lambda: 0.23  p1: 0.47  p2: 0.78  log-likelihood: -2.500
it: 5  lambda: 0.24  p1: 0.48  p2: 0.78  log-likelihood: -2.498
it: 6  lambda: 0.25  p1: 0.48  p2: 0.79  log-likelihood: -2.496
it: 7  lambda: 0.26  p1: 0.48  p2: 0.79  log-likelihood: -2.496
```

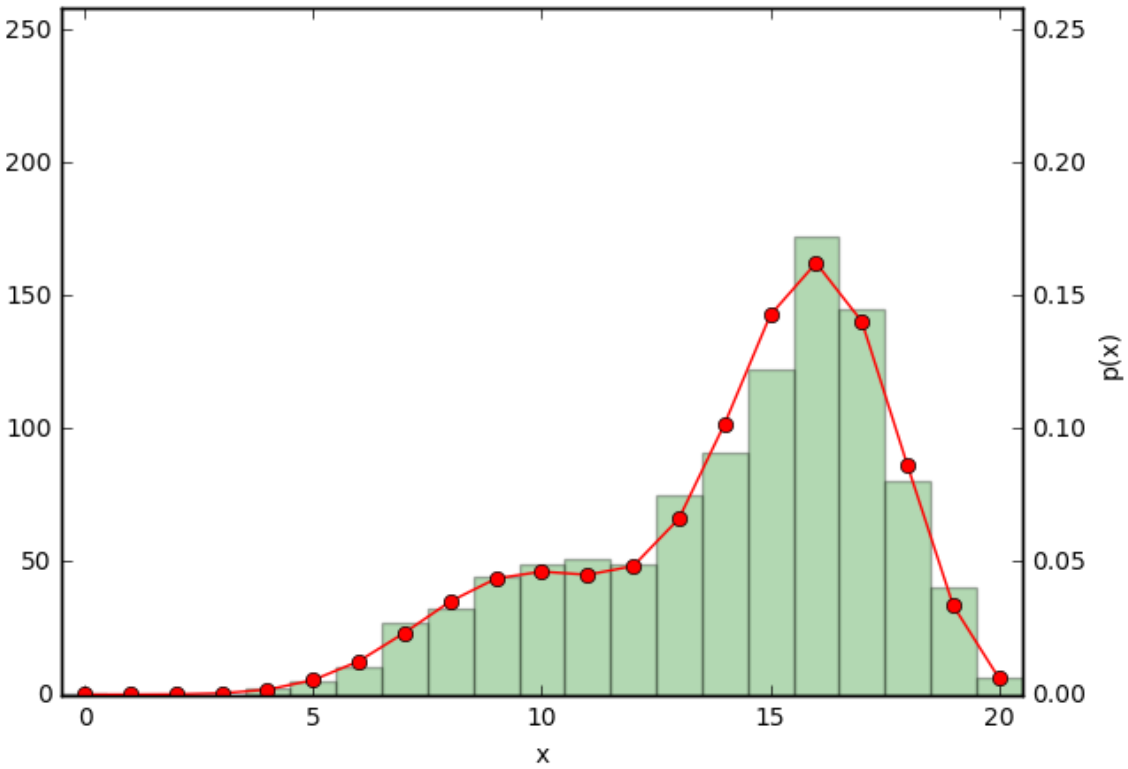


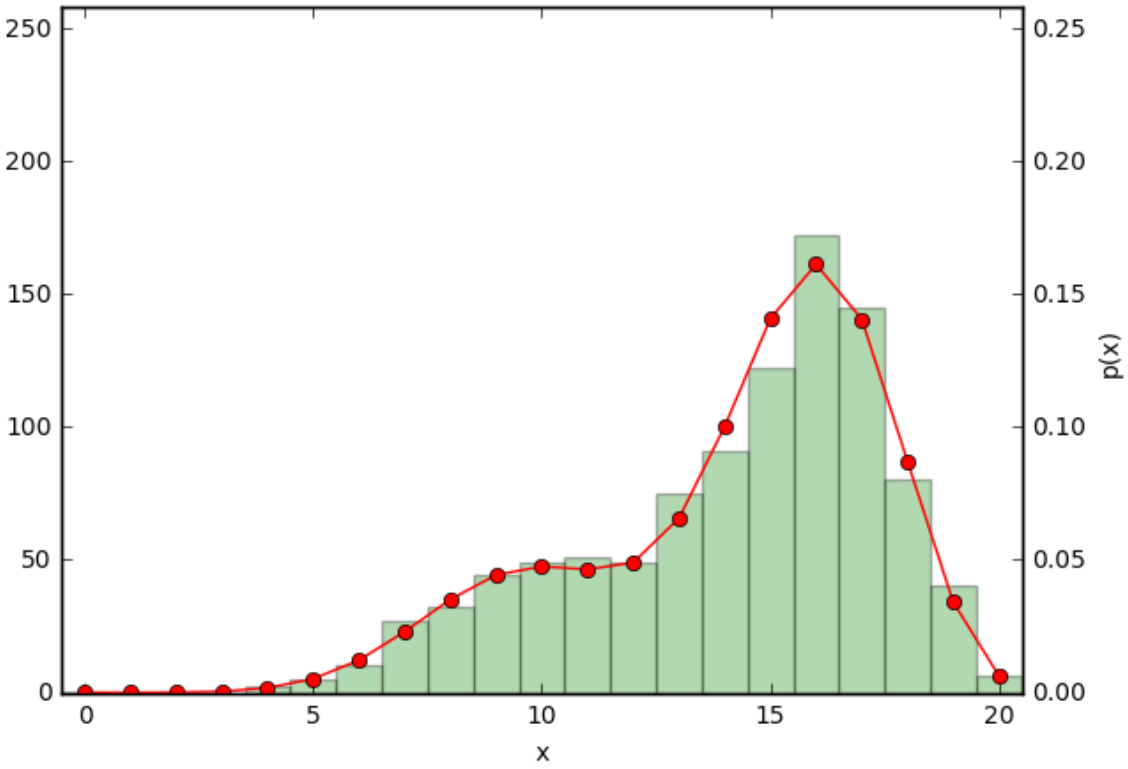



250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0
5
10
15
20
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0
5
10
15
20
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0
5
10
15
20
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0
5
10
15
20
x



More Experiments (10 P)

Examine the behaviour of the EM algorithm for various combinations of data generation parameters and initializations (for generating various distributions, use the method `utils.generateData(...)`). In particular, find settings for which:

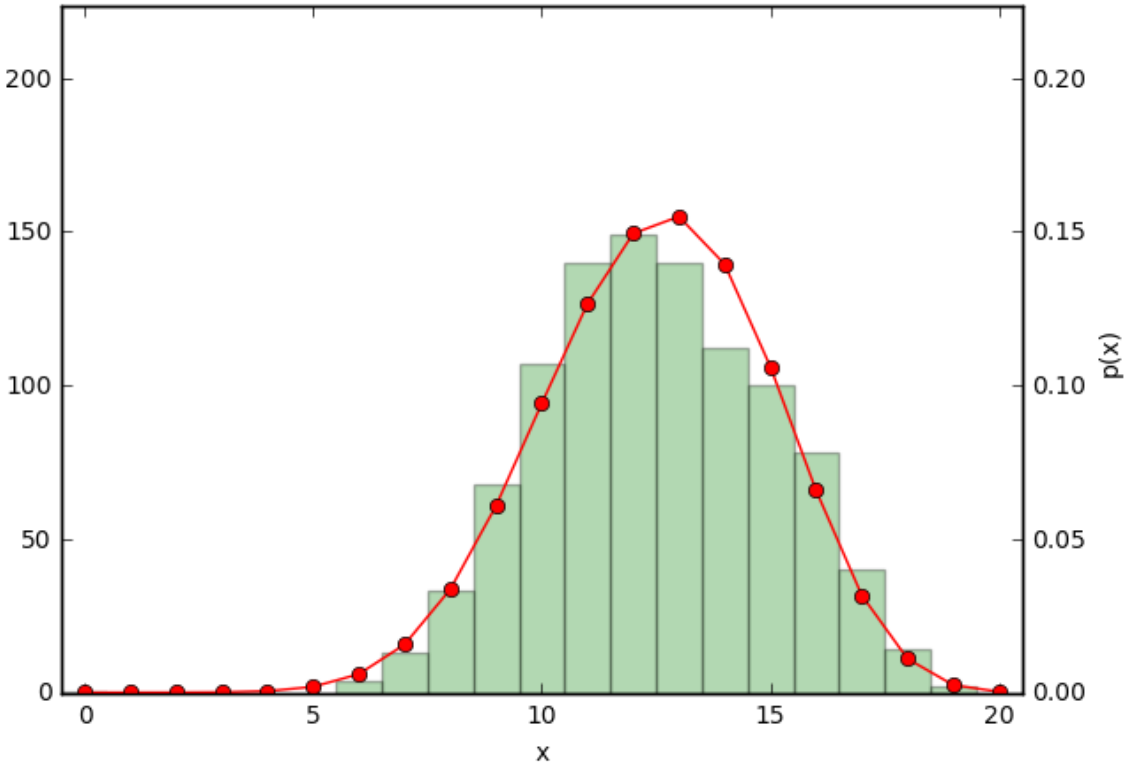
- The role of coins A and B are permuted between the data generating model and the learned model (i.e. $\hat{p}_1 \approx p_2$, $\hat{p}_2 \approx p_1$ and $\hat{\lambda} \approx 1 - \lambda$).
- The EM procedure takes a long time to converge.

Print the parameters and log-likelihood objective at each iteration. Only display the plot for the converged model.

In [76]:

```
lam = 0.50
p1 = 0.49
p2 = 0.51
data = utils.generateData(0.25,0.75,0.6,1000,20)
EMLearning(lam,p1,p2,False,True)
```

```
it: 1  lambda: 0.45  p1: 0.61  p2: 0.64  log-likelihood: -2.358
it: 2  lambda: 0.45  p1: 0.61  p2: 0.64  log-likelihood: -2.357
it: 3  lambda: 0.45  p1: 0.60  p2: 0.64  log-likelihood: -2.355
it: 4  lambda: 0.45  p1: 0.59  p2: 0.65  log-likelihood: -2.351
it: 5  lambda: 0.45  p1: 0.59  p2: 0.66  log-likelihood: -2.347
it: 6  lambda: 0.46  p1: 0.58  p2: 0.66  log-likelihood: -2.343
it: 7  lambda: 0.46  p1: 0.57  p2: 0.67  log-likelihood: -2.339
it: 8  lambda: 0.46  p1: 0.57  p2: 0.67  log-likelihood: -2.336
it: 9  lambda: 0.46  p1: 0.56  p2: 0.68  log-likelihood: -2.335
it:10  lambda: 0.47  p1: 0.56  p2: 0.68  log-likelihood: -2.334
```



In [ ]: