Machine Learning 1 WS18/19

Submission for Exercise Sheet 5

Hendrik Makait 384968
Michael Hoppe 362514
Wai Tang Victor Chan 406094
Rudi Poepsel Lemaitre 373017
Jonas Piotrowski 399334
Aki Saksala 399293

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.

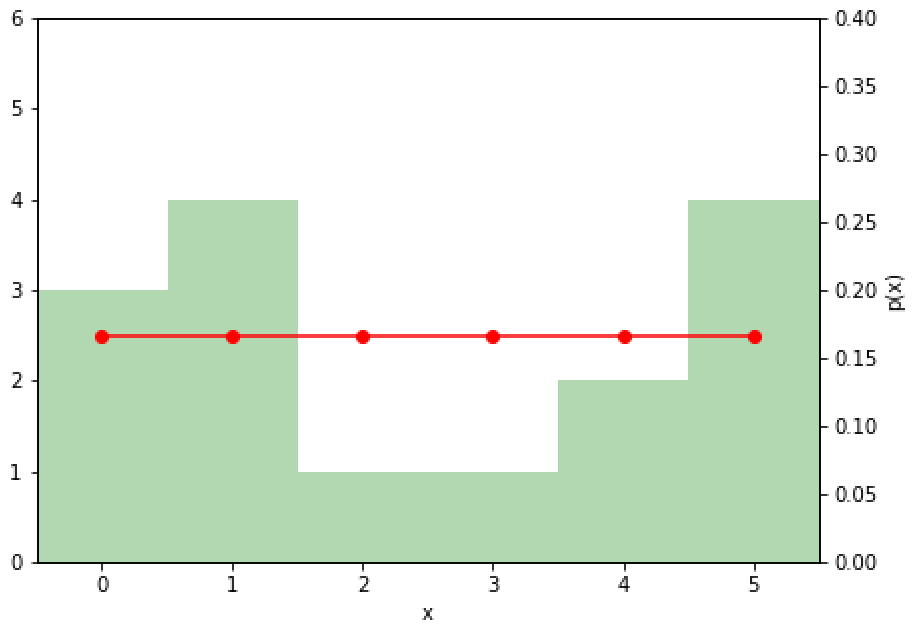
- **`utils.plot(data,distribution)` :** Plot a histogram of the number of heads per trial along with the probability distribution. This function will be used to visualize the progress of the EM algorithm at every iteration.

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

In [2]:

```
%matplotlib inline
import numpy,utils

# Print the data matrix as a result of the three coins experiment with parameter 0.5,
 0.8 and 0.2.
data = utils.generateData(0.5,0.8,0.2,15,5)
print(data)

# Print the data histogram along with a uniform probability distribution.
utils.plot(data,numpy.ones([data.shape[1]+1])/(data.shape[1]+1))
```

```
[[1 1 1 1 1]
 [0 0 1 1 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 0 0 0]
 [1 1 1 1 1]
 [1 0 0 0 0]
 [0 0 0 0 0]
 [1 1 1 1 1]
 [1 0 0 1 1]
 [1 0 1 1 1]
 [0 0 0 1 0]
 [1 1 1 1 0]
 [0 0 0 0 0]
 [0 0 0 0 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:

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

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

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 [3]:

```
def h(xi):
    return numpy.sum(xi)

def t(xi):
    return len(xi) - numpy.sum(xi)

def log_P(xi, lam, p1, p2):
    coin_a = lam * numpy.power(p1, h(xi)) * numpy.power((1 - p1), t(xi))
    coin_b = (1 - lam) * numpy.power(p2, h(xi)) * numpy.power((1 - p2), t(xi))
    return numpy.log(coin_a + coin_b)

def loglikelihood(data, lam, p1, p2):
    return numpy.sum([(1.0 / data.shape[0]) * log_P(x, lam, p1, p2) for x in data])

print(loglikelihood(data, 0.5, 0.8, 0.2))
```

```
-2.696651522967214
```

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

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

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

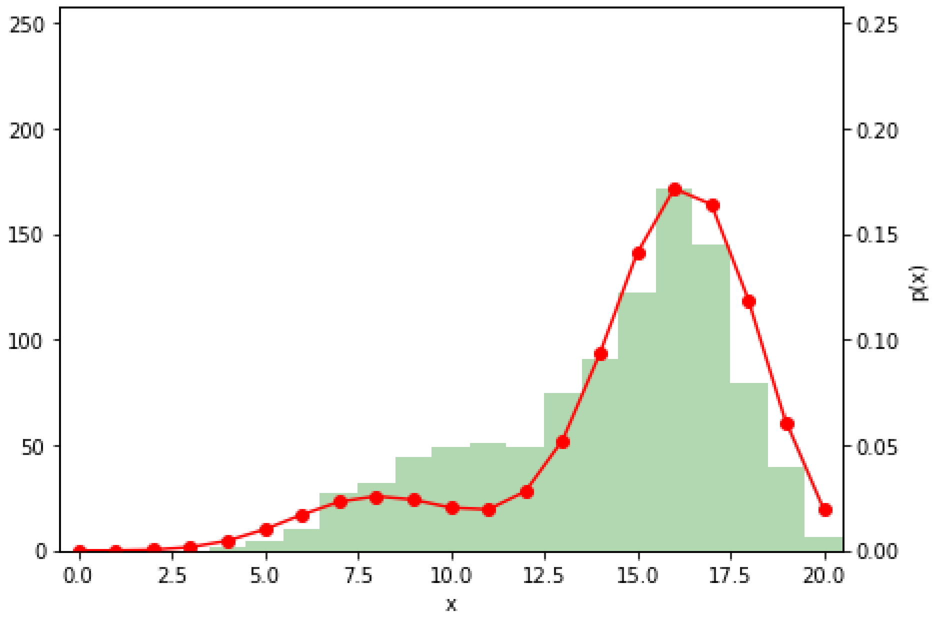
TODO:

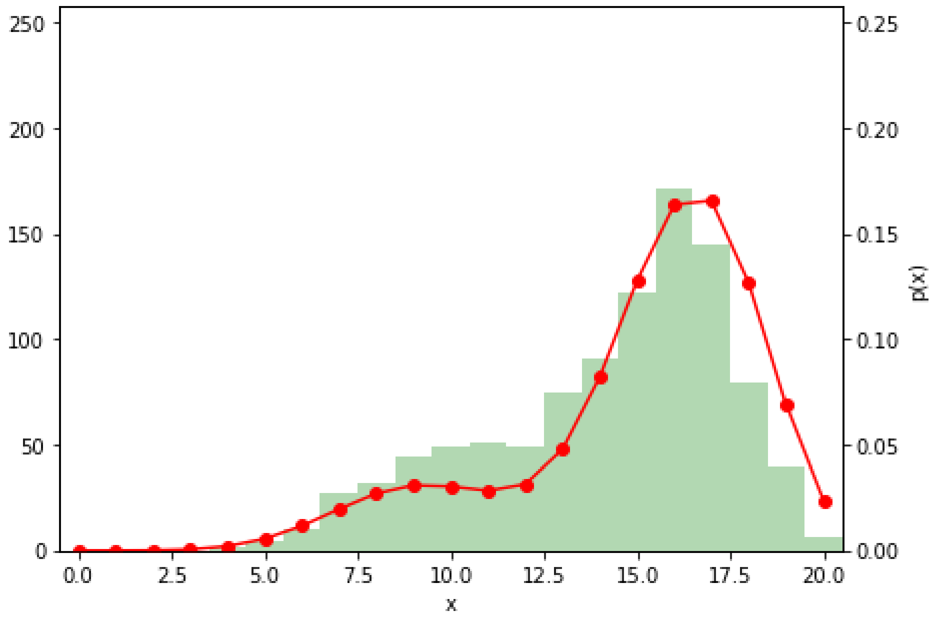
- **Implement the EM learning procedure.**
- **Use as stopping criterion the improvement of log-likelihood between two iterations to be smaller than 0.001.**
- **Run the EM procedure on the data returned by function `utils.unknownData()`. Use as an initial solution for your model the parameters $\lambda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

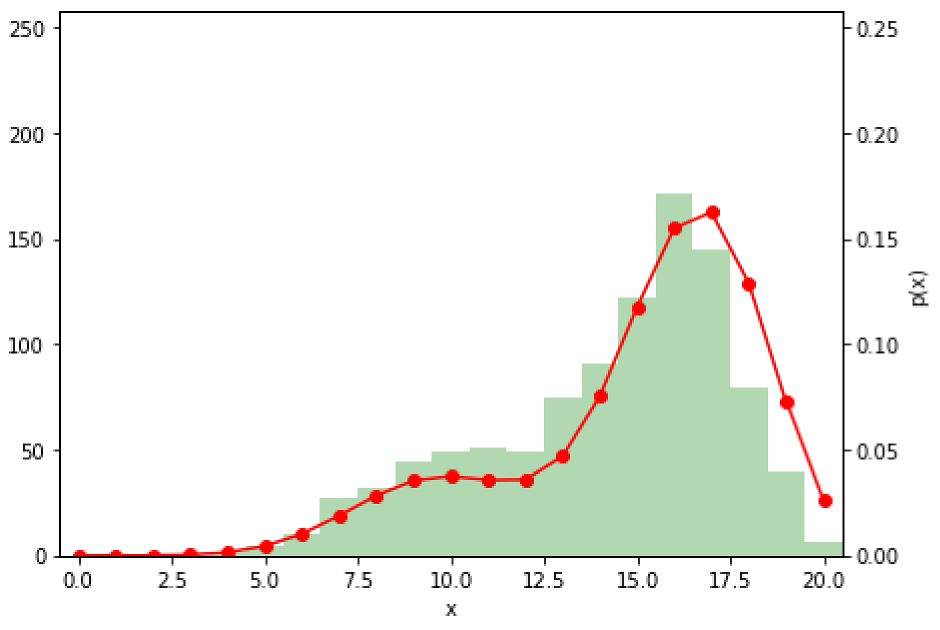
In [4]:

```python
import utils
from scipy import stats
%matplotlib inline

def R1(xi, lam, p1, p2):
    t1 = lam * numpy.power(p1, h(xi)) * numpy.power((1 - p1), t(xi))
    t2 = t1 + (1 - lam) * numpy.power(p2, h(xi)) * numpy.power((1 - p2), t(xi))
    return t1 / t2

def R2(xi, lam, p1, p2):
    t1 = (1 - lam) * numpy.power(p2, h(xi)) * numpy.power((1 - p2), t(xi))
    t2 = lam * numpy.power(p1, h(xi)) * numpy.power((1 - p1), t(xi)) + t1
    return t1 / t2

def p1_new(data, lam, p1, p2):
    t1 = numpy.sum([R1(xi, lam, p1, p2) * h(xi) for xi in data])
    t2 = data.shape[1] * numpy.sum([R1(xi, lam, p1, p2) for xi in data])
    return t1 / t2

def p2_new(data, lam, p1, p2):
    t1 = numpy.sum([R2(xi, lam, p1, p2) * h(xi) for xi in data])
    t2 = data.shape[1] * numpy.sum([R2(xi, lam, p1, p2) for xi in data])
    return t1 / t2

def lam_new(data, lam, p1, p2):
    return numpy.sum([1.0 / data.shape[0] * R1(xi, lam, p1, p2) for xi in data])

def head_probability(head_count, throw_count, lam, p1, p2):
    coin_a = stats.binom.pmf(head_count, throw_count, p1)
    coin_b = stats.binom.pmf(head_count, throw_count, p2)
    return (lam * coin_a) + ((1 - lam) * coin_b)

def EM(data, lam, p1, p2, plot_all=True):
    criterion = False
    loglike = loglikelihood(data, lam, p1, p2)

    it = 0

    # Iterate until the stopping criterion is satisfied
    while (criterion == False):

        #TODO:
        # - Perform one step of EM
        # - Print the log-likelihood and the model parameters
        # - Plot data histogram and the learned probability distribution
        # - Determine if stopping criterion is satisfied

        lam = lam_new(data, lam, p1, p2)
        p1 = p1_new(data, lam, p1, p2)
        p2 = p2_new(data, lam, p1, p2)

        old_loglike = loglike
        loglike = loglikelihood(data, lam, p1, p2)
        criterion = loglike - old_loglike < 0.001

        print('it:%2d  lambda: %.2f  p1: %.2f  p2: %.2f  log-likelihood: %.3f'%(
                it, lam, p1, p2, loglike))
        it += 1
        if plot_all:
            utils.plot(data, [head_probability(i, data.shape[1]+1, lam, p1, p2) for i in range(data.shape[1]+1)])
    if not plot_all:
        utils.plot(data, [head_probability(i, data.shape[1]+1, lam, p1, p2) for i in range(data.shape[1]+1)])

data = utils.unknownData()
lam = 0.5
p1  = 0.25
p2  = 0.75
EM(data, lam, p1, p2)
```

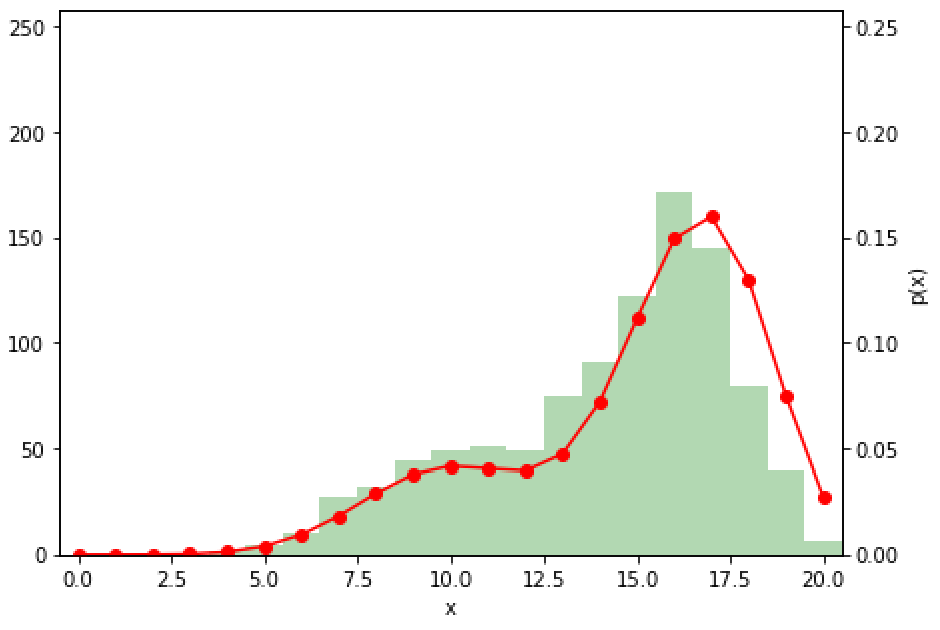
```
it: 0  lambda: 0.15  p1: 0.39  p2: 0.76  log-likelihood: -11.720
it: 1  lambda: 0.18  p1: 0.44  p2: 0.78  log-likelihood: -11.682
it: 2  lambda: 0.21  p1: 0.46  p2: 0.78  log-likelihood: -11.669
it: 3  lambda: 0.24  p1: 0.47  p2: 0.79  log-likelihood: -11.664
it: 4  lambda: 0.25  p1: 0.48  p2: 0.79  log-likelihood: -11.662
it: 5  lambda: 0.26  p1: 0.49  p2: 0.79  log-likelihood: -11.662
```



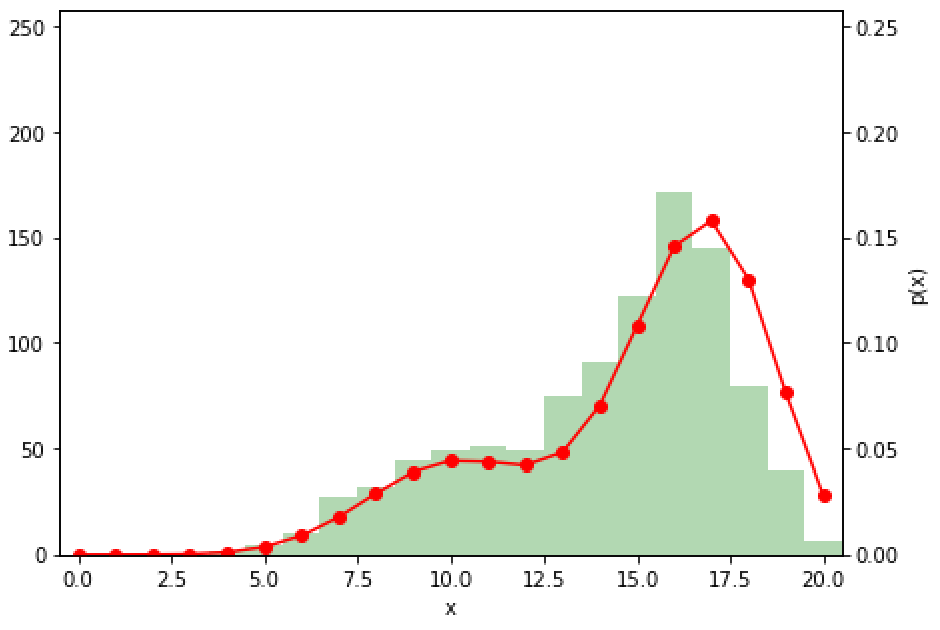




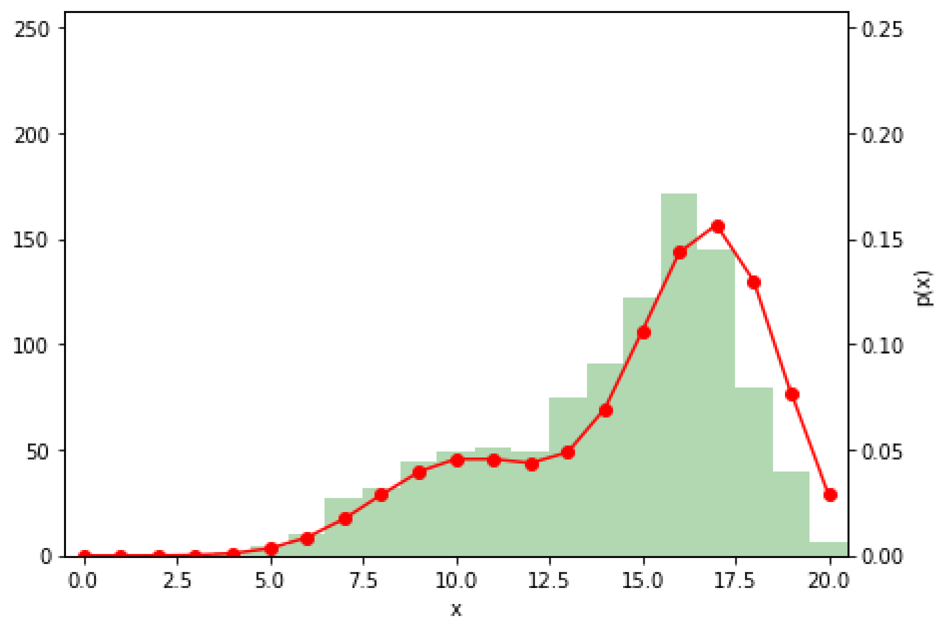
250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x



250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x



250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
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:

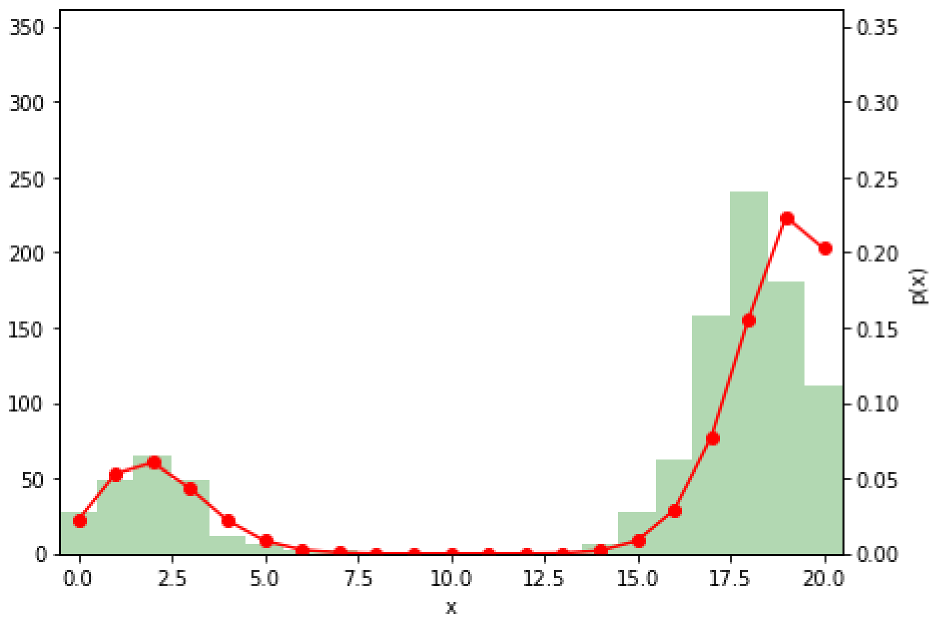
- The role of coins A and B are permuted between the data generating model and the learned model (i.e. $\hat{p}_1 \approx p_2$, $\hat{p}_2 \approx p_1$ and $\hat{\lambda} \approx 1 - \lambda$).
- The EM procedure takes a long time to converge.

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

```
In [5]:
```

```
data = utils.generateData(0.2, 0.1, 0.9, 1000, 20)

lam = 0.8
p1 = 0.8
p2 = 0.2
EM(data, lam, p1, p2, plot_all=False)
```

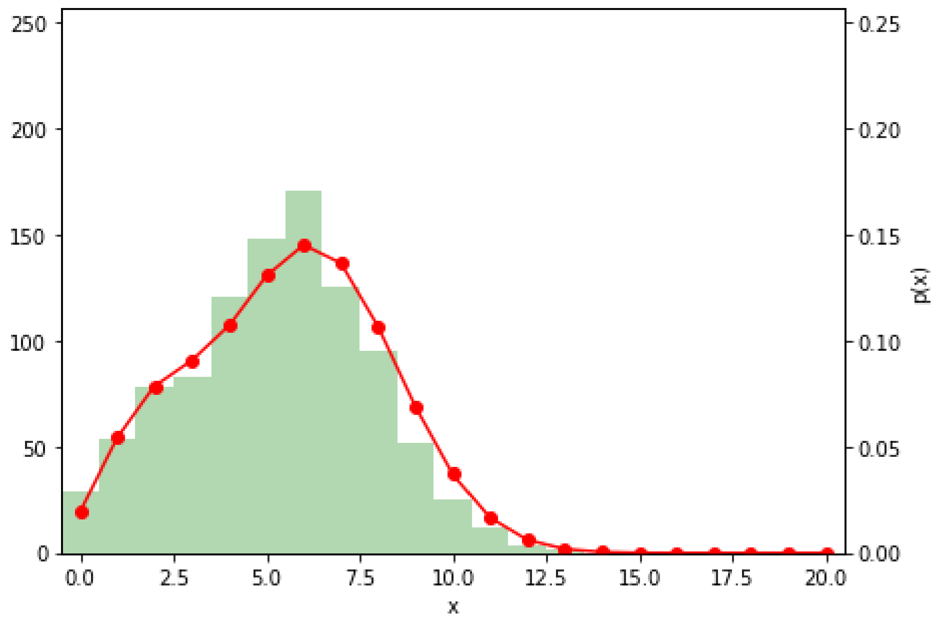
```
it: 0  lambda: 0.79  p1: 0.90  p2: 0.10  log-likelihood: -7.015
it: 1  lambda: 0.79  p1: 0.90  p2: 0.10  log-likelihood: -7.015
```



In [12]:

```
data = utils.generateData(0.2, 0.1, 0.3, 1000, 20)

lam = 0.1
p1 = 0.000002
p2 = 0.99999
#utils.plot(data, [head_probability(i, data.shape[1]+1, lam, p1, p2) for i in range(dat
a.shape[1]+1)])
EM(data, lam, p1, p2, plot_all=False)
```

```
it: 0  lambda: 0.96  p1: 0.25  p2: 0.75  log-likelihood: -11.574
it: 1  lambda: 0.99  p1: 0.26  p2: 0.63  log-likelihood: -11.538
it: 2  lambda: 0.99  p1: 0.26  p2: 0.57  log-likelihood: -11.536
it: 3  lambda: 0.99  p1: 0.26  p2: 0.54  log-likelihood: -11.534
it: 4  lambda: 0.98  p1: 0.26  p2: 0.52  log-likelihood: -11.532
it: 5  lambda: 0.97  p1: 0.26  p2: 0.50  log-likelihood: -11.529
it: 6  lambda: 0.97  p1: 0.26  p2: 0.49  log-likelihood: -11.527
it: 7  lambda: 0.96  p1: 0.25  p2: 0.48  log-likelihood: -11.525
it: 8  lambda: 0.95  p1: 0.25  p2: 0.46  log-likelihood: -11.522
it: 9  lambda: 0.94  p1: 0.25  p2: 0.46  log-likelihood: -11.520
it:10  lambda: 0.93  p1: 0.25  p2: 0.45  log-likelihood: -11.518
it:11  lambda: 0.92  p1: 0.25  p2: 0.44  log-likelihood: -11.515
it:12  lambda: 0.90  p1: 0.24  p2: 0.43  log-likelihood: -11.513
it:13  lambda: 0.89  p1: 0.24  p2: 0.42  log-likelihood: -11.511
it:14  lambda: 0.88  p1: 0.24  p2: 0.42  log-likelihood: -11.509
it:15  lambda: 0.86  p1: 0.24  p2: 0.41  log-likelihood: -11.506
it:16  lambda: 0.85  p1: 0.24  p2: 0.41  log-likelihood: -11.504
it:17  lambda: 0.83  p1: 0.23  p2: 0.40  log-likelihood: -11.502
it:18  lambda: 0.81  p1: 0.23  p2: 0.40  log-likelihood: -11.499
it:19  lambda: 0.80  p1: 0.23  p2: 0.39  log-likelihood: -11.497
it:20  lambda: 0.78  p1: 0.23  p2: 0.39  log-likelihood: -11.495
it:21  lambda: 0.76  p1: 0.22  p2: 0.38  log-likelihood: -11.493
it:22  lambda: 0.74  p1: 0.22  p2: 0.38  log-likelihood: -11.490
it:23  lambda: 0.72  p1: 0.22  p2: 0.38  log-likelihood: -11.488
it:24  lambda: 0.70  p1: 0.22  p2: 0.37  log-likelihood: -11.485
it:25  lambda: 0.68  p1: 0.21  p2: 0.37  log-likelihood: -11.483
it:26  lambda: 0.66  p1: 0.21  p2: 0.36  log-likelihood: -11.481
it:27  lambda: 0.64  p1: 0.21  p2: 0.36  log-likelihood: -11.478
it:28  lambda: 0.61  p1: 0.20  p2: 0.36  log-likelihood: -11.476
it:29  lambda: 0.59  p1: 0.20  p2: 0.35  log-likelihood: -11.473
it:30  lambda: 0.57  p1: 0.19  p2: 0.35  log-likelihood: -11.470
it:31  lambda: 0.55  p1: 0.19  p2: 0.35  log-likelihood: -11.468
it:32  lambda: 0.53  p1: 0.19  p2: 0.35  log-likelihood: -11.465
it:33  lambda: 0.50  p1: 0.18  p2: 0.34  log-likelihood: -11.462
it:34  lambda: 0.48  p1: 0.18  p2: 0.34  log-likelihood: -11.459
it:35  lambda: 0.46  p1: 0.17  p2: 0.34  log-likelihood: -11.457
it:36  lambda: 0.44  p1: 0.17  p2: 0.33  log-likelihood: -11.454
it:37  lambda: 0.41  p1: 0.16  p2: 0.33  log-likelihood: -11.451
it:38  lambda: 0.39  p1: 0.16  p2: 0.33  log-likelihood: -11.448
it:39  lambda: 0.37  p1: 0.15  p2: 0.33  log-likelihood: -11.446
it:40  lambda: 0.35  p1: 0.15  p2: 0.32  log-likelihood: -11.443
it:41  lambda: 0.33  p1: 0.14  p2: 0.32  log-likelihood: -11.441
it:42  lambda: 0.32  p1: 0.14  p2: 0.32  log-likelihood: -11.439
it:43  lambda: 0.30  p1: 0.13  p2: 0.32  log-likelihood: -11.437
it:44  lambda: 0.28  p1: 0.13  p2: 0.32  log-likelihood: -11.435
it:45  lambda: 0.27  p1: 0.12  p2: 0.31  log-likelihood: -11.433
it:46  lambda: 0.26  p1: 0.12  p2: 0.31  log-likelihood: -11.432
it:47  lambda: 0.25  p1: 0.12  p2: 0.31  log-likelihood: -11.431
it:48  lambda: 0.24  p1: 0.11  p2: 0.31  log-likelihood: -11.430
```


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x

```
In [ ]:
```