MeiBau_Sheet08

Unknown Author

December 10, 2013

Exercise Sheet 07

Expectation Maximization

In this assignment we will be using the Expectation Maximization method to estimate the parameters of the three coin experiment. We will examine the results of the method for various combinations of λ, p_1 and p_2.

In [91]:
```
import numpy as np
import random as rand
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='svg'
```

Part 1: Generating the Data

Implement a function which generates the data for the three coin experiment.

The parameters are:

- λ := The probability of heads on the secret coin S
- p_1 := The probability of heads on coin A
- p_2 := The probability of heads on coin B

N samples are collected the following way:

- The secret coin (S) is tossed
- If the result was heads, coin A is tossed m times and the results are recorded
- If the result was 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 m**x**N matrix, where each of the N columns contains the results of the corresponding sample (generated eiher by coin A or by coin B).

In [3]:
```
def generateData(lam, p1, p2, N, M):
    """
    returns: An mxN matrix, containing 1 for heads and 0 for tails.
    """
    prob = [p2,p1]
    p = 0

    data = np.zeros((M,N))
    for j in range(N):
        # toss S
        # if true the index p of propabilities is set to 1 and we choose coin A
        # otherwise its B. thats why the propabilities array is sorted [p2,p1]
        p = rand.random() < lam
```

```
        for i in range(M):
            # toss A or B := prob[p]
            data[i,j] = rand.random() < prob[p]

    return data
```

Part 2: Implementing EM for the model

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:

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

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

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

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

The update rules for the remaining parameters are:

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

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

Apply the update rule while $|\lambda^{new} - \lambda|$ + $|p_1^{new} - p_1|$ + $|p_2^{new} - p_2|$ > t, where t is some small threshold.

In [83]:

```
def EM(lam,p1,p2,X,N,M):
    threshhold = 1e-2
    iterations = 0
    max_iterations = 1000

    lam_new = 0
    p1_new = 0
    p2_new = 0

    # number of heads for every N
    h = sum(X,0)
    # number of tails is the number of
    # tosses minus the number of heads for every series.
    t = M - h

    # break condition further down
    while iterations < max_iterations:

        tmp1 = (lam*(p1**h) * ((1-p1)**t))
        den = (tmp1 + (1 - lam)*(p2**h) * ((1-p2)**t))
        R1 = tmp1 / den
        lam_new = sum(R1) / N
        tmp2 = ((1-lam)*(p2**h) * ((1-p2)**t))
        R2 = tmp2 / den
        p1_new = sum(R1 * h) / (M * sum(R1))
        p2_new = sum(R2 * h) / (M * sum(R2))

        iterations += 1
        if (abs(lam_new - lam) + abs(p1_new - p1) +
            abs(p2_new - p2) < threshhold):
            break
        else:
            lam = lam_new
            if not lam > 0: lam = 0.0000001
```

```
            p1 = p1_new
            if not p1 > 0: p1 = 0.0000001
            p2 = p2_new
            if not p2 > 0: p2= 0.0000001

    if iterations == max_iterations:
        return False
    return lam, p1,p2
```

Part 3: Testing the Solution

Examine how the method behaves w.r.t varying parameters of the generated data. For each combination you test, generate the data once and run EM 20 times with different random initialization values (the values you feed into the EM function). Then show the following in one plot for every estimated parameter separately (total of 3 plots) (λ,p_1,p_2):

- the true value (mark it with a star)
- the mean (mark it with a circle)
- mean + standard deviation
- mean - standard deviation
- minimum
- maximum

Add the hyperparameters to the x axis (for example using xtick).

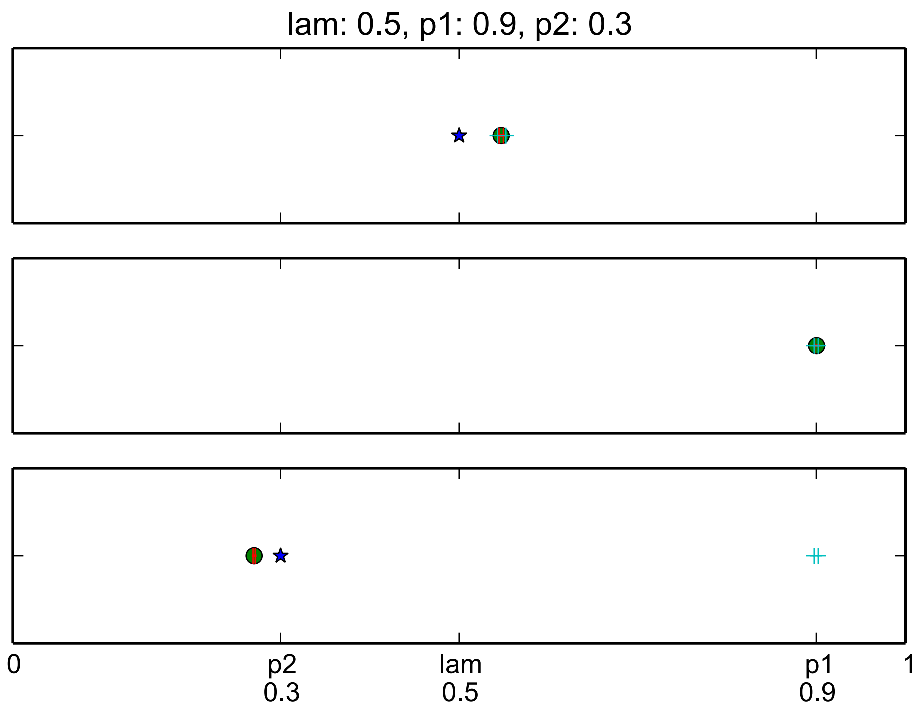
Take care that the following may occur: if the true parameters are (λ,p_1,p_2) = (0.3,0.7,0.4) the method may estimate (0.7,0.4,0.7), i.e. determine (1-λ) and swap the values of coins A and B. Account for this in your solution

In [92]:

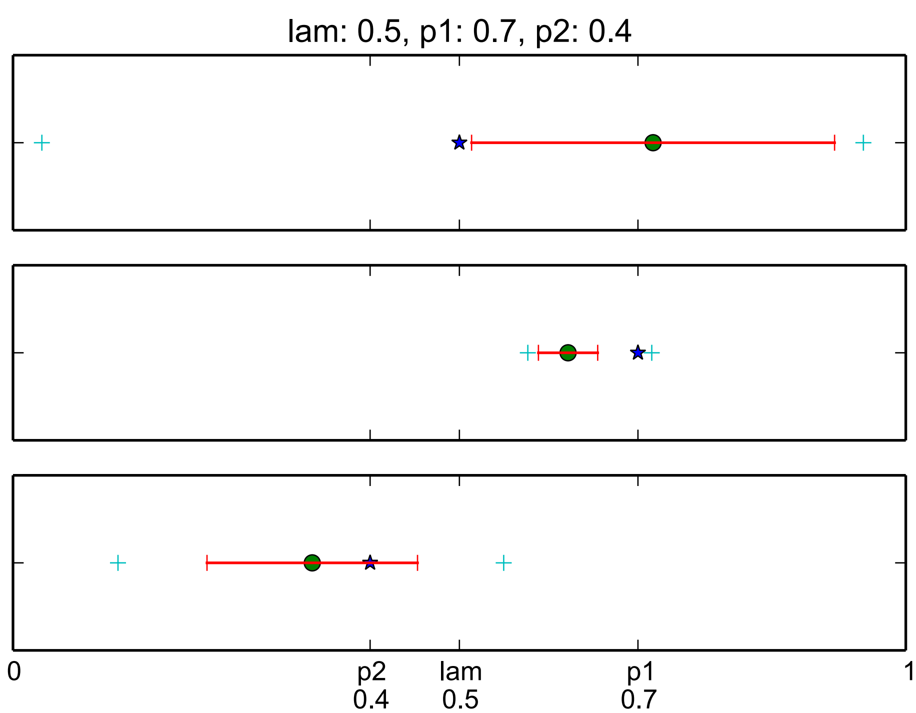
```
def run(N,M):
    sets = np.array([[0.5, 0.9, 0.3],[0.5,0.7,0.4], [0.5,0.6,0.5], [0.5,0.55,0.45], [0
                    [0.8,0.6,0.4],[0.8,0.5,0.45], [0.8, 0.51,0.49], [0.6, 0.5, 0.3], [0.1
    set_labels = ['lam', 'p1', 'p2']

    for sample in sets:
        results = []
        lam = sample[0]
        p1 = sample[1]
        p2 = sample[2]
        # generate the data
        old_res = (0,0,0)
        data = generateData(lam, p1, p2, N, M)
        # run EM 20 times
        for i in range(20):
            # generate random initialization values
            while True:
                # run EM
                init_lam = rand.random()
                init_p1 = rand.random()
                init_p2 = rand.random()
                res = EM(init_lam,init_p1,init_p2,data,N,M)
                if res:
                    break
            res = list(res)
            # maybe the results switched p1 and p2
            dist1 = abs(old_res[1] - res[1]) + abs(old_res[2] - res[2])
            dist2 = abs(old_res[1] - res[2]) + abs(old_res[2] - res[1])
            if dist1 > dist2:
                res[1], res[2] = res[2], res[1]
                res[0] = 1 - res[0]
            results.append(res)
            old_res = res
```

```
        results = (np.array(results)).T

        #std_deviation
        std_dev = np.std(results,1)

        # maybe the results switched p1 and p2
        means = sum(results.T)/len(results[0])

        dist1 = abs(means[1] - sample[1]) + abs(means[2] - sample[2])
        dist2 = abs(means[1] - sample[2]) + abs(means[2] - sample[1])
        if dist1 > dist2:
            # exchange the results if so
            print "exchanged results to fit the samples"
            means[1], means[2] = means[2], means[1]
            means[0] = 1-means[0]

            results[0] = 1 - results[0]
            # copy arrays
            results[1] = np.array(results[2])
            results[2] = np.array(results[1])

        print "results: "+str(means)
        print "min, max: "+str(np.min(results,1))+", "+str(np.max(results,1))
        print "standard deviation: "+str(std_dev)

        # plot the EM generated parameters
        f, ax = plt.subplots(3, sharex=True)
        ax[0].set_title("lam: "+str(sample[0])+", p1: "+
                        str(sample[1])+", p2: "+str(sample[2]))
        for i in range(3):
            # - the true value (mark it with a star)
            ax[i].plot(sample[i], 0, marker = '*')
            # - the mean (mark it with a circle)
            ax[i].plot(means[i], 0, marker = 'o')
            # - mean + standard deviation
            # - mean - standard deviation
            ax[i].plot([means[i]-std_dev[i],means[i]+std_dev[i]],
                       [0,0], marker = '|')
            # - minimum
            # - maximum
            ax[i].plot([min(results[i]),max(results[i])],
                       [0,0], marker = '+', linestyle='None')

            # Add the hyperparameters to the x axis (for example using xtick).
            ax[i].set_xticks([0,sample[0],sample[1],sample[2],1])
            ax[i].set_xticklabels(['0',str(set_labels[0])+
                                   '\n'+str(sample[0]),str(set_labels[1])+
                                   '\n'+str(sample[1]),str(set_labels[2])+
                                   '\n'+str(sample[2]),'1'])
            ax[i].set_yticks([0])
            ax[i].set_yticklabels(' ')
            ax[i].set_autoscalex_on(False)
            ax[i].set_xlim([0,1])
        plt.show()

run(20,10)
```

```
exchanged results to fit the samples
results: [ 0.54705957  0.90034955  0.27035143]
min, max: [ 0.54299374  0.89764145  0.89764145], [ 0.55233469
0.90217848  0.90217848]
standard deviation: [ 0.00239482  0.00206167  0.00121206]
```

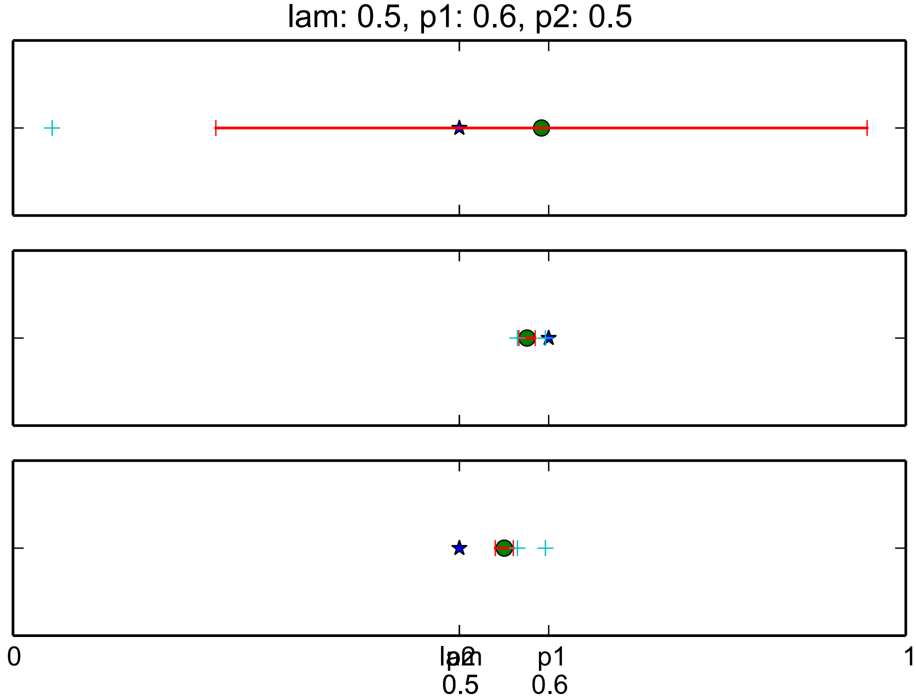



```
results: [ 0.71695273  0.62172099  0.33520313]
min, max: [ 0.03235165  0.57665619  0.11761126], [ 0.95249649
0.71547872  0.54963467]
standard deviation: [ 0.20328578  0.03318301  0.11789715]
```

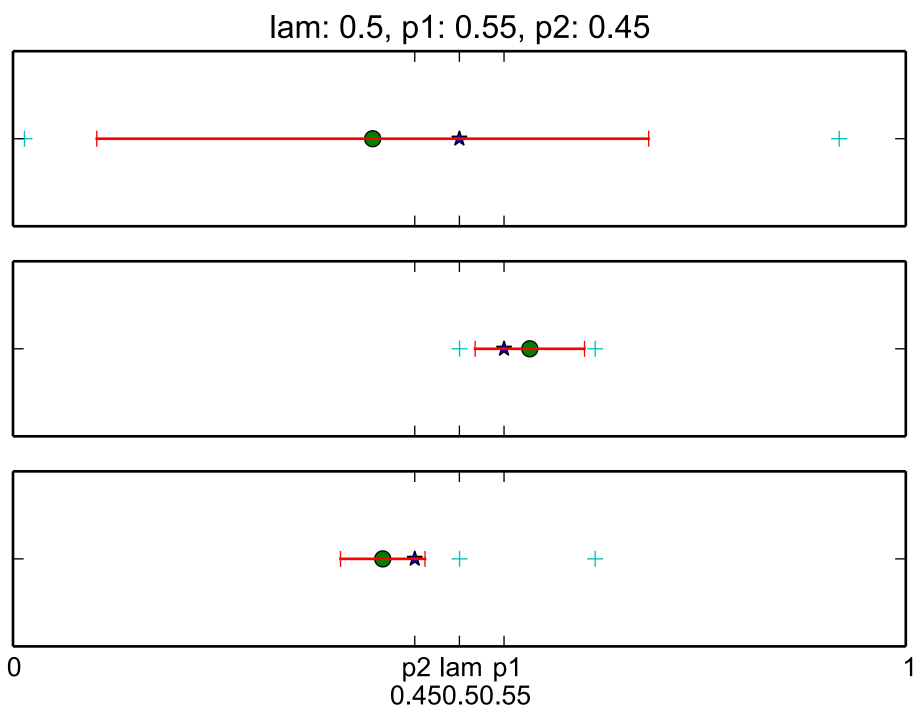



```
exchanged results to fit the samples
results: [ 0.59198372  0.57573408  0.55024848]
min, max: [ 0.04391916  0.56500076  0.56500076], [ 0.99996926
```

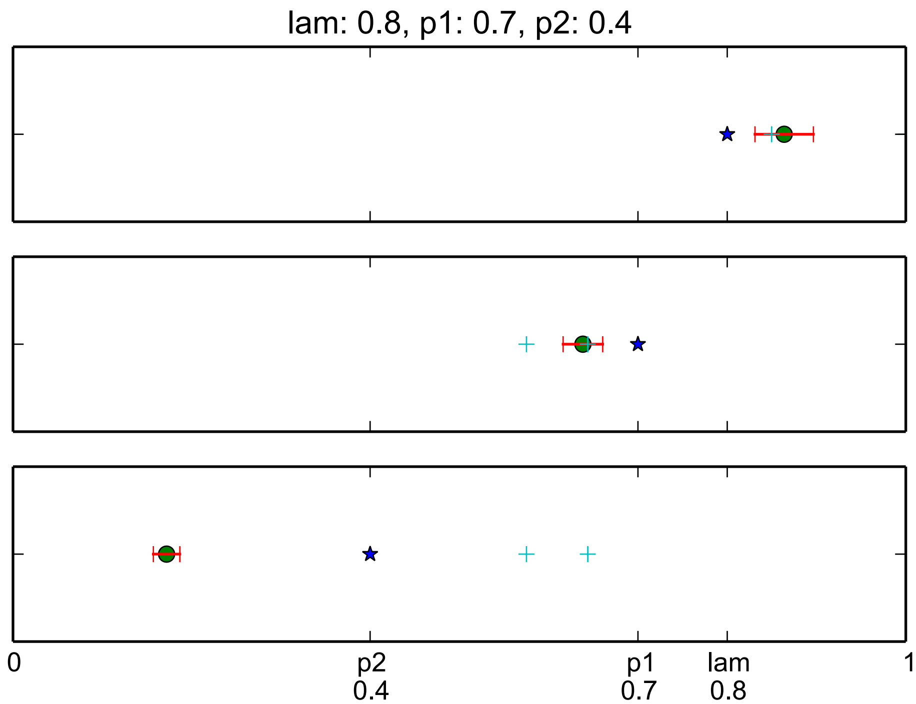
```
0.59637005  0.59637005]
standard deviation: [ 0.36475533  0.00911388  0.01007897]
```



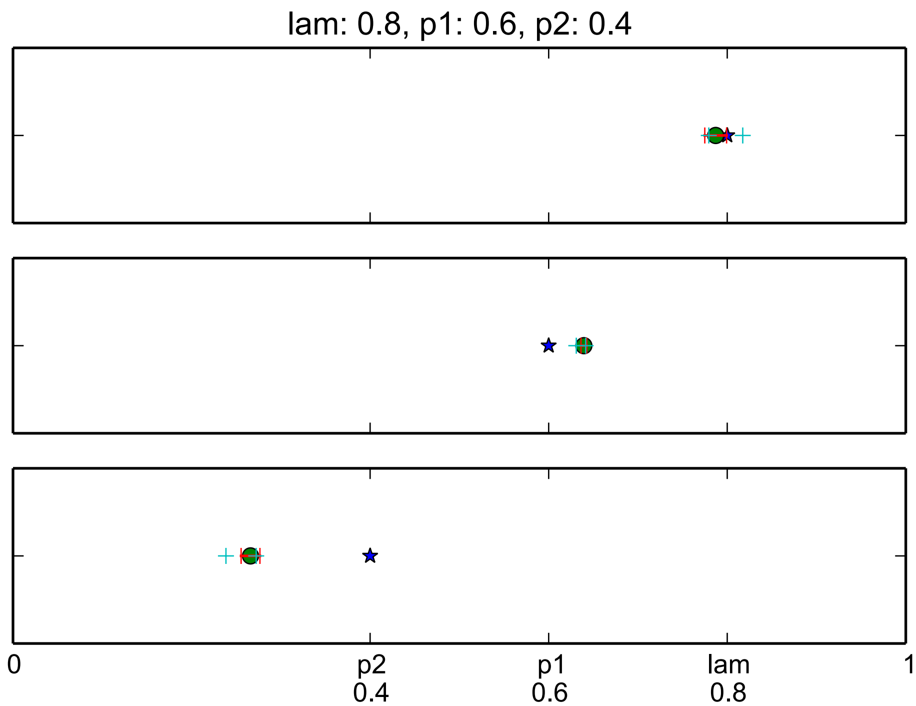

```
exchanged results to fit the samples
results: [ 0.40289539  0.57886023  0.41415375]
min, max: [ 0.01282471  0.50015825  0.50015825], [ 0.9253783
0.65209098  0.65209098]
standard deviation: [ 0.30907445  0.0612452   0.04734729]
```



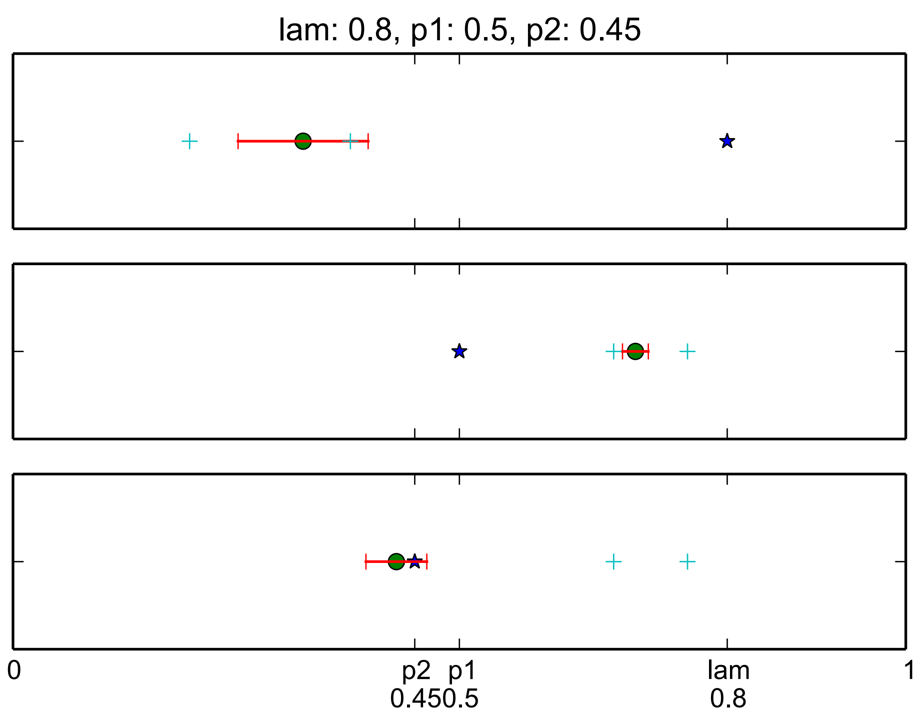
```
exchanged results to fit the samples
results: [ 0.86379277  0.63833512  0.17203381]
min, max: [ 0.84971564  0.57512668  0.57512668], [ 0.99973338
0.64382165  0.64382165]
standard deviation: [ 0.03272278  0.02217488  0.01483723]
```




```
results: [ 0.78707318  0.63941284  0.26604343]
min, max: [ 0.77935548  0.63093149  0.23852825], [ 0.81737627
0.64142265  0.27240078]
standard deviation: [ 0.01220957  0.00322437  0.0105453 ]
```

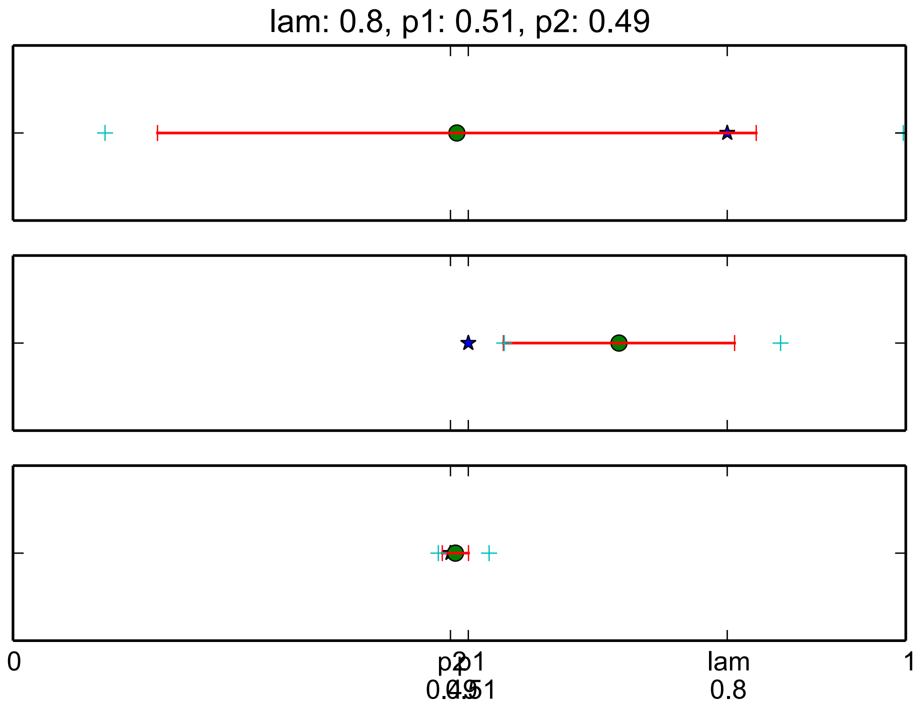



```
exchanged results to fit the samples
results: [ 0.32495672  0.69715281  0.42943367]
min, max: [ 0.19785331  0.67280151  0.67280151], [ 0.37790272
0.75550874  0.75550874]
standard deviation: [ 0.07283907  0.01445641  0.03409282]
```

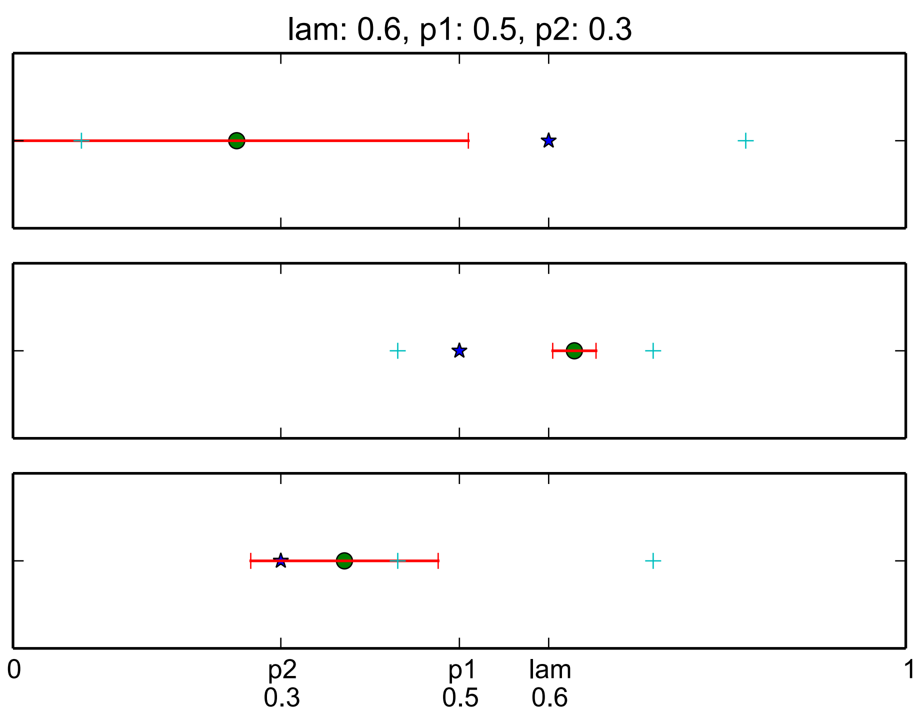



```
results: [ 0.49716523  0.67875153  0.49552945]
min, max: [ 0.10309333  0.55018916  0.4764514 ], [ 0.99726088
```

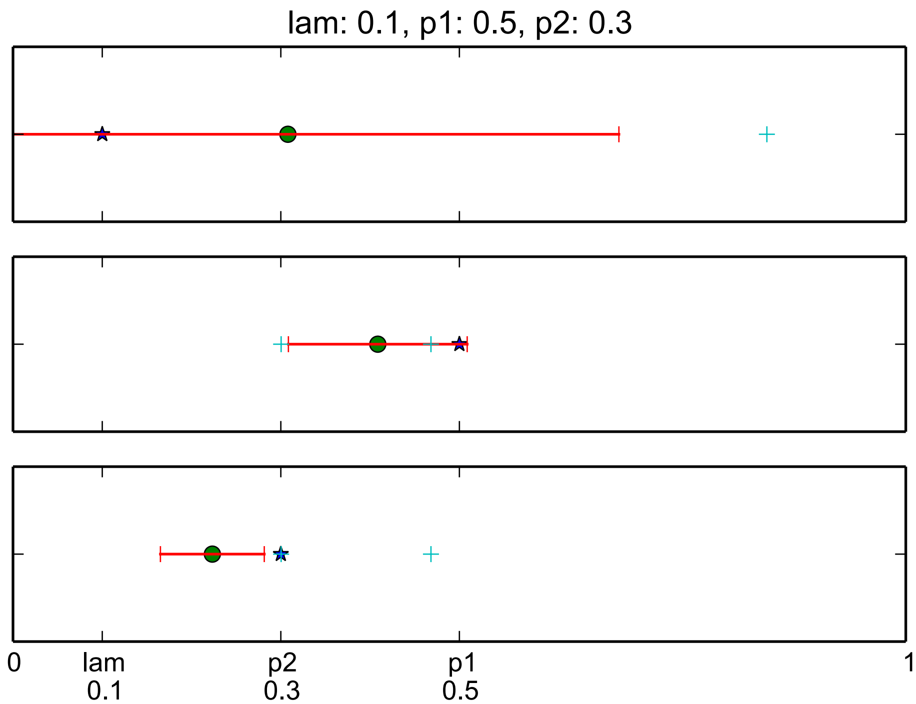
0.85976376 0.53330363]
standard deviation: [0.33530872 0.12948868 0.01468398]



exchanged results to fit the samples
results: [0.25059936 0.62878696 0.37126136]
min, max: [0.07666295 0.43094905 0.43094905], [0.82073953
0.71705328 0.71705328]
standard deviation: [0.25944025 0.02425805 0.10496458]



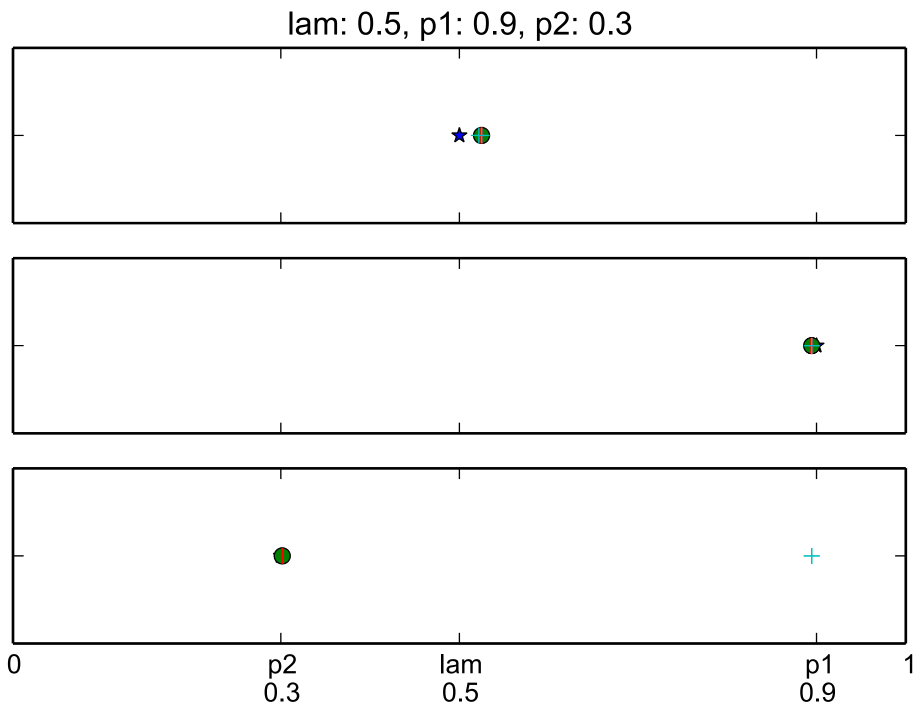
```
exchanged results to fit the samples
results: [ 0.30792513  0.4086096   0.22329365]
min, max: [  1.12008155e-06   3.00319100e-01   3.00319100e-01], [
0.84441485  0.46815757  0.46815757]
standard deviation: [ 0.37075209  0.10012825  0.05815594]
```



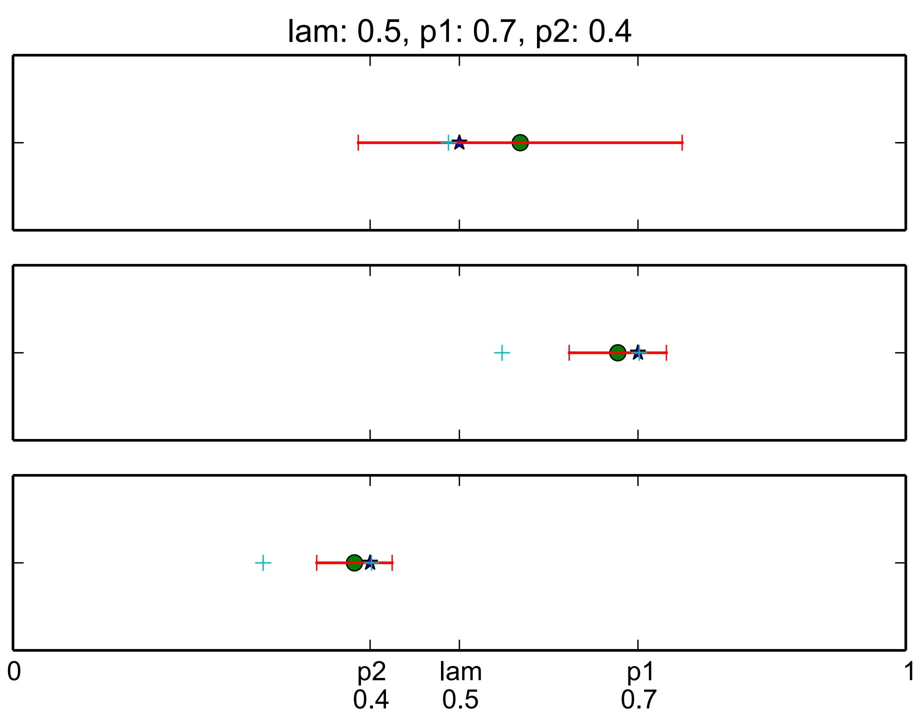
In [93]:

```
#Run the same code for N = 200 and M = 100
run(200,100)
```

```
exchanged results to fit the samples
results: [ 0.52480898  0.89458551  0.30159015]
min, max: [ 0.52176909  0.89456803  0.89456803], [ 0.52500382
0.89475302  0.89475302]
standard deviation: [  7.05326700e-04   8.30529580e-04
4.27723323e-05]
```

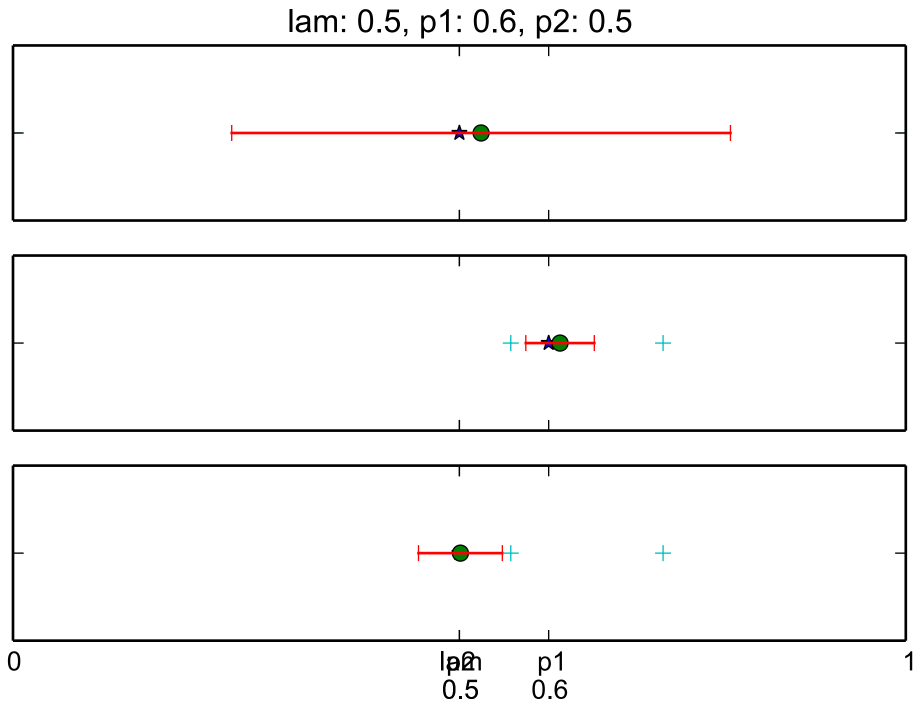


results: [0.56817077 0.67744331 0.3824592]
min, max: [0.48777686 0.54785   0.28022583], [1.
0.70148832 0.40154419]
standard deviation: [0.18141684 0.05444442 0.0422626]

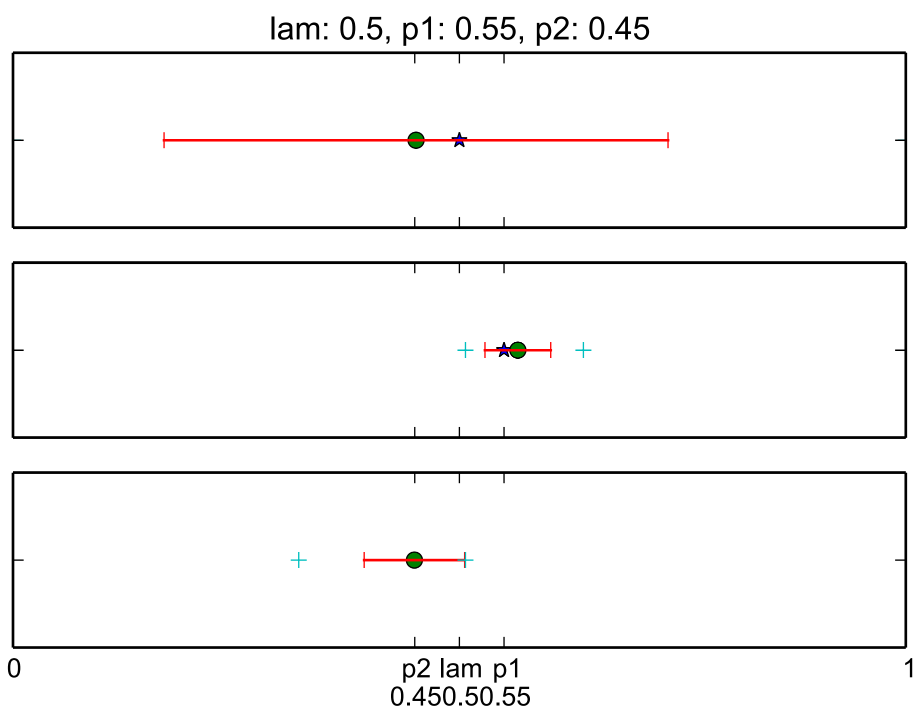


exchanged results to fit the samples
results: [0.52429285 0.61274847 0.50115477]
min, max: [2.55351296e-15   5.57550000e-01   5.57550000e-01], [1.

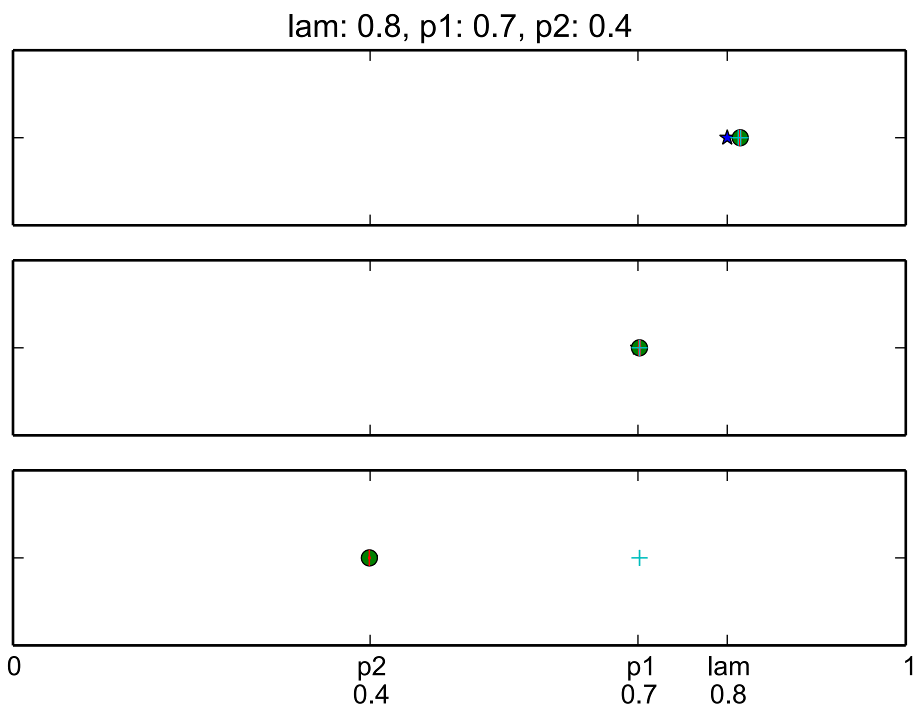
```
0.72814723  0.72814723]
standard deviation: [ 0.2793054   0.0383552   0.04696678]
```



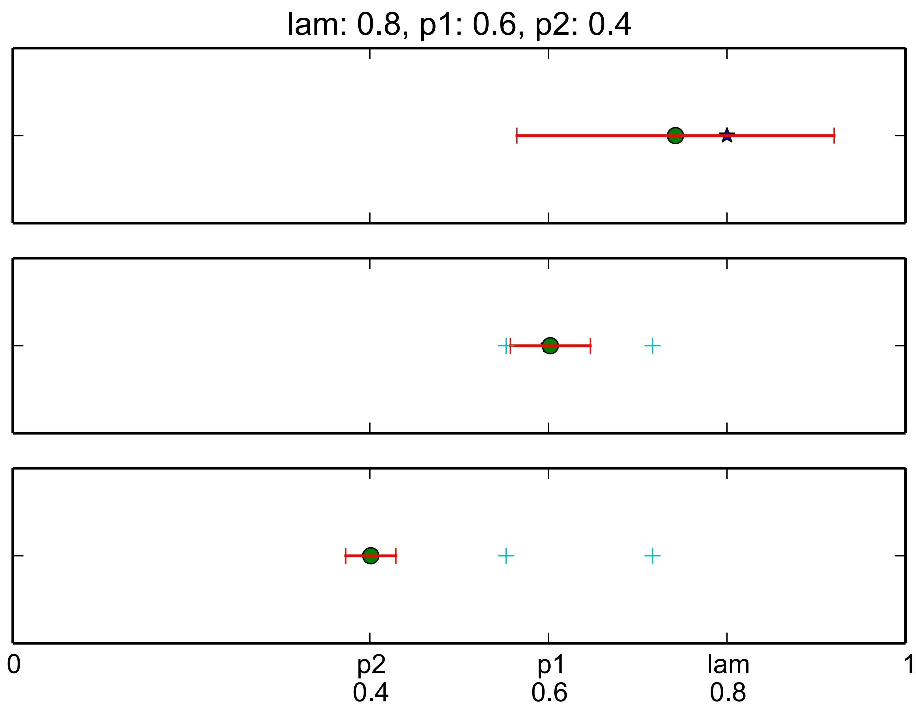

```
results: [ 0.45144779  0.5654731   0.44960829]
min, max: [  5.02251574e-11   5.06800000e-01   3.20048164e-01], [ 1.
0.63890047  0.5068    ]
standard deviation: [ 0.28223829  0.03682168  0.05626111]
```



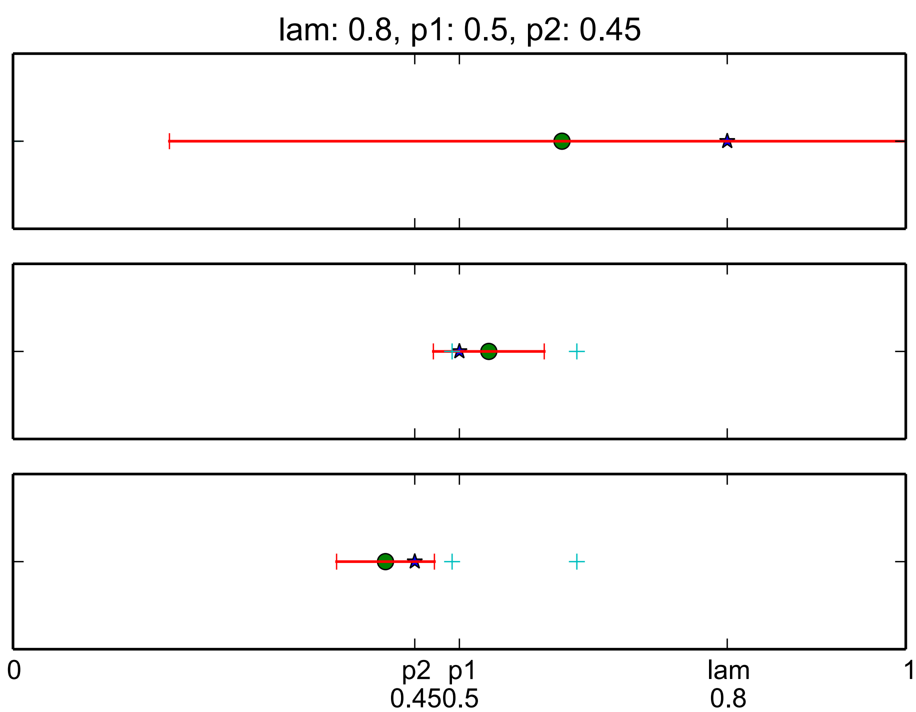
```
exchanged results to fit the samples
results: [ 0.8145601   0.70169891  0.39917723]
min, max: [ 0.81235374  0.70158244  0.70158244], [ 0.8150457
0.70197003  0.70197003]
standard deviation: [  8.22332047e-04   9.26881612e-04
9.55269289e-05]
```




```
exchanged results to fit the samples
results: [ 0.74230059  0.60208693  0.40105224]
min, max: [  1.21756807e-08   5.52650004e-01   5.52650004e-01], [
0.99999999  0.71663939  0.71663939]
standard deviation: [ 0.1776502   0.04482089  0.02812354]
```

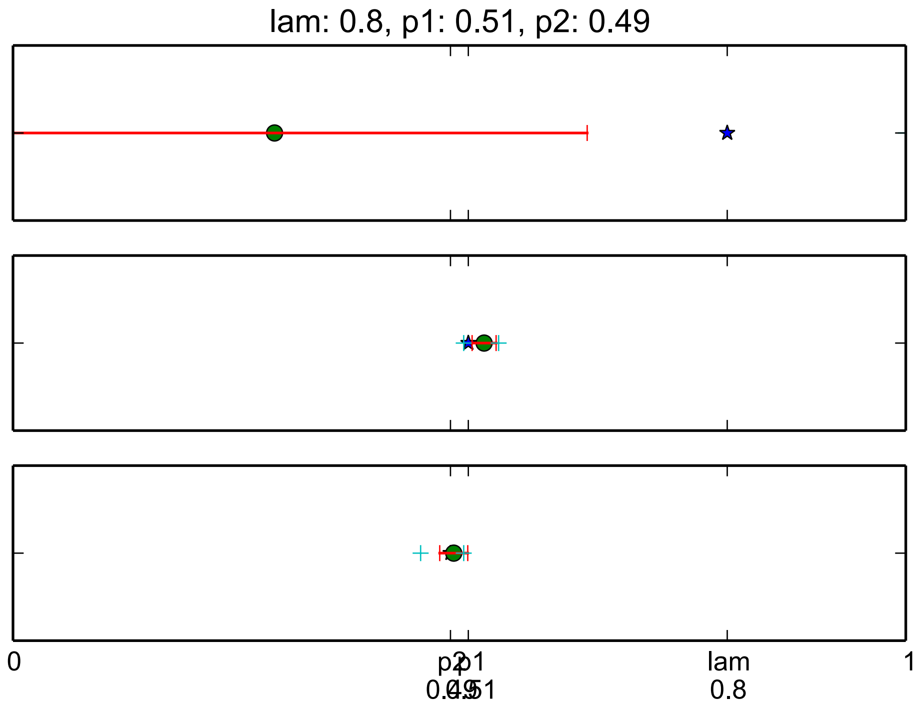



```
exchanged results to fit the samples
results: [ 0.61492451  0.53294746  0.4172309 ]
min, max: [  1.00315312e-10   4.91850000e-01   4.91850000e-01], [ 1.
0.63157856  0.63157856]
standard deviation: [ 0.43968915  0.06209084  0.05478564]
```

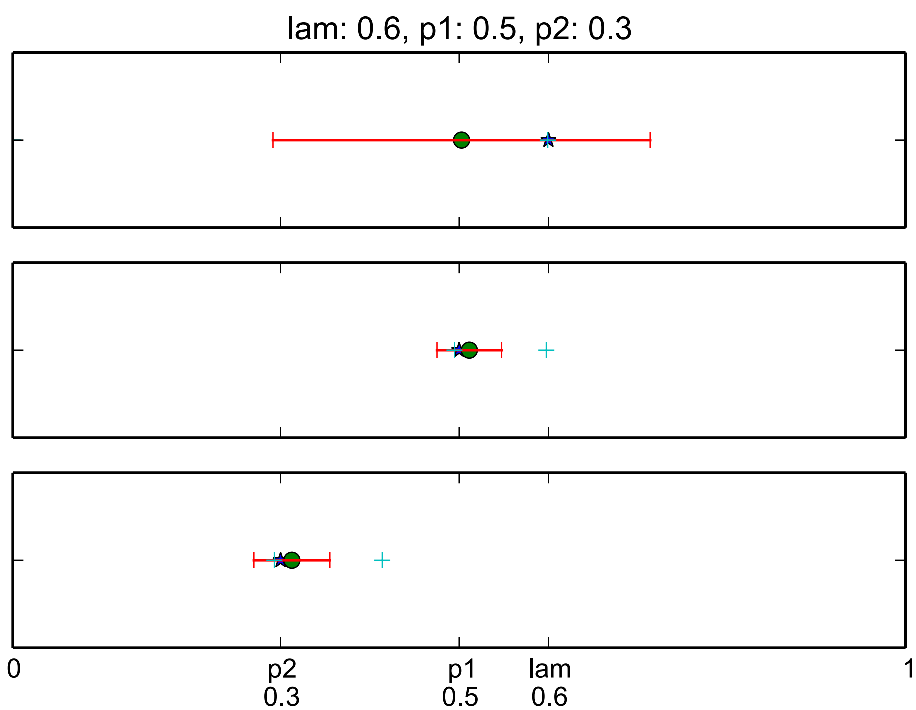



```
results: [ 0.29297362  0.52767916  0.49367237]
min, max: [  1.81239420e-39   5.04850000e-01   4.56568232e-01], [ 1.
```

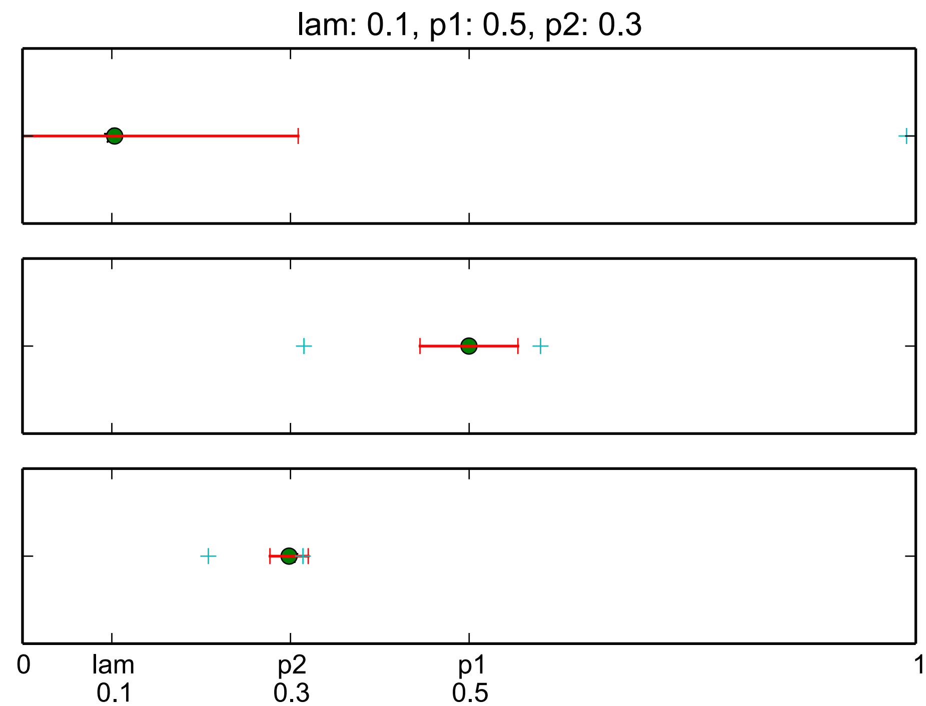
```
0.54392532  0.50485   ]
standard deviation: [ 0.35015903  0.01344003  0.01569763]
```




```
results: [ 0.50270959  0.51138382  0.31264763]
min, max: [  7.47233345e-28   4.94746222e-01   2.93034717e-01], [
0.59919876  0.59765646  0.4139    ]
standard deviation: [ 0.21122359  0.0361745   0.04254594]
```



```
results: [ 0.10319493  0.49978505  0.29841928]
min, max: [  5.73439560e-41   3.15055352e-01   2.08048447e-01], [
0.98967027  0.57992362  0.31395   ]
standard deviation: [ 0.20541646  0.05481445  0.02143763]
```



Analysis

** When does the method provide estimates close to the true values? When are the estimates further away from the true values? In general, when would you expect the method to behave well with low variance? **

Analysis

If p1 and p2 are close to the same value, lambda varies a lot. this is because if both coin's propabilities are nearly the same, it is hard to distiguish between them. i.e. for [0.8, 0.51,0.49] the standard derivation for lambda is big because even for lambda = 0.5 we would have data that looks nearly the same. The method will behave well if p1 and p2 are some well distingushable values. The results are getting better if N and M are as big as possible of course.Export your ipynb file as a pdf

Make sure **all** of your code can be read within the pdfs and the individual plots aren't split across pages.