Expectation Maximization with Coin Flips

Expectation Maximization is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The best introductory example I've come across, which considers a series of coin flips, is from the paper, "What is the expectation maximization algorithm?", and is also covered in one of University of Michigan's Machine Learning course's lectures and discussion section notes.

While the original paper and course coverage of this example are good, I found that a few key details were glossed over; in this notebook I aim to lay everything out in its entirety, and hope understanding this example in detail will provide intuition for how EM works, laying the foundation to study its theory and more complex examples further.

A coin experiment

Suppose your friend has posed a challenge: estimate the bias of two coins in her possession. They might be fair coins, be more heavily weighted towards heads; you don't know. Here's the clue she's provided: a piece of paper with 5 records of an experiment where she's:

- Chosen one of the two coins at random.
- Flipped that same coint 10 times.

How can you provide a reasonable estimate of each coin bias? Let's refer to these coins as coin A and coin B and their bias as θ_A and θ_B.

We see which coin is flipped

Let's first imagine that this piece of paper shows which coin was chosen for each trial:

coin	flips	# coin A heads	# coin B heads
B	HTTTHHTHTH	0	5
A	HHHHTHHHHH	9	0
A	HTHHHHHTHH	8	0
B	HTHTTTHHTT	0	4
A	THHHTHHHTH	7	0

In this case it's easy, and boils down to estimating each independently. Let's start with coin A: across three trials of 10 flips, there are 24 heads. So a reasonable estimate of the coin bias would be 24/30 or 0.8.

To estimate the bias of coin B, we have 9 heads across 2 sets of 10 flips for an estimated bias of 9/20 = 0.45.

So when we know everything, in *the complete data case*, our problem is pretty easy.

We don't see which coin is flipped

Now let's make the problem harder: what if we are shown the same trials as above, but do not know which coin was chosen for each set of 10 flips? We only know that each coin has an equal chance of being chosen each time.

In this scenario, the coin is not observed, and could be considered a hidden or latent variable. EM comes in handy for all sorts of so called "latent variable" models, including Gaussian Mixture Models and Hidden Markov Models. This is merely a contrived example to provide as simple a latent variable model as possible.

coin	flips	# coin A heads	# coin B heads
?	HTTTHHTHTH	?	?
?	HHHHTHHHHH	?	?

Code implementation

Let's now implement the algorithm described above and work through the same example.

In [9]:
```
import numpy as np

def coin_em(rolls, theta_A=None, theta_B=None, maxiter=10):
    # Initial Guess
    theta_A = theta_A or random.random()
    theta_B = theta_B or random.random()
    thetas = [(theta_A, theta_B)]
    # Iterate
    for c in range(maxiter):
        print("#%d:\t%0.2f %0.2f" % (c, theta_A, theta_B))
        heads_A, tails_A, heads_B, tails_B = e_step(rolls, theta_A, theta_B)
        theta_A, theta_B = m_step(heads_A, tails_A, heads_B, tails_B)

    thetas.append((theta_A,theta_B))
    return thetas, (theta_A,theta_B)

def e_step(rolls, theta_A, theta_B):
    """Produce the expected value for heads_A, tails_A, heads_B, tails_B
    over the rolls given the coin biases"""

    heads_A, tails_A = 0,0
    heads_B, tails_B = 0,0
    for trial in rolls:
        likelihood_A = coin_likelihood(trial, theta_A)
        likelihood_B = coin_likelihood(trial, theta_B)
        p_A = likelihood_A / (likelihood_A + likelihood_B)
        p_B = likelihood_B / (likelihood_A + likelihood_B)
        heads_A += p_A * trial.count("H")
        tails_A += p_A * trial.count("T")
        heads_B += p_B * trial.count("H")
        tails_B += p_B * trial.count("T")
    return heads_A, tails_A, heads_B, tails_B

def m_step(heads_A, tails_A, heads_B, tails_B):
    """Produce the values for theta that maximize the expected number of heads/tails"""

    # Replace dummy values with your implementation
    theta_A = heads_A / (heads_A + tails_A)
    theta_B = heads_B / (heads_B + tails_B)
    return theta_A, theta_B

def coin_likelihood(roll, bias):
    # P(X | Z, theta)
    numHeads = roll.count("H")
    flips = len(roll)
    return pow(bias, numHeads) * pow(1-bias, flips-numHeads)
```

Example from paper

Completing the example above until convergence:

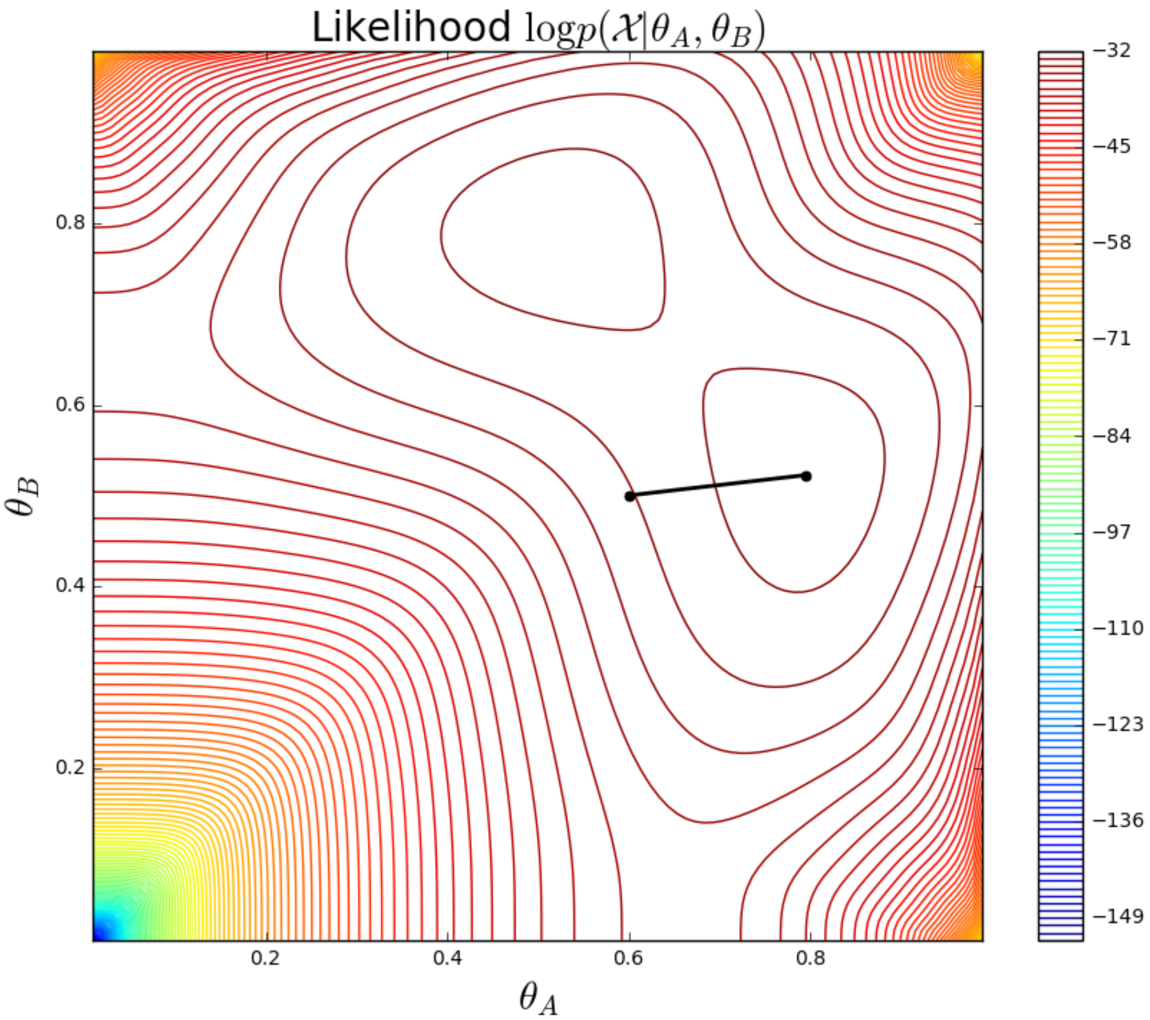
In [2]:
```
rolls = [ "HTTTHHTHTH", "HHHHTHHHHH", "HTHHHHHTHH",
          "HTHTTTHHTT", "THHHTHHHTH" ]
thetas, _ = coin_em(rolls, 0.6, 0.5, maxiter=6)
```

```
#0:     0.60 0.50
#1:     0.71 0.58
#2:     0.75 0.57
#3:     0.77 0.55
#4:     0.78 0.53
#5:     0.79 0.53
```

Plotting convergence

In [7]:
```
%matplotlib inline
from matplotlib import pyplot as plt
import matplotlib as mpl


def plot_coin_likelihood(rolls, thetas=None):
    # grid
    xvals = np.linspace(0.01,0.99,100)
    yvals = np.linspace(0.01,0.99,100)
    X,Y = np.meshgrid(xvals, yvals)

    # compute likelihood
    Z = []
    for i,r in enumerate(X):
        z = []
        for j,c in enumerate(r):
            z.append(coin_marginal_likelihood(rolls,c,Y[i][j]))
        Z.append(z)

    # plot
    plt.figure(figsize=(10,8))
    C = plt.contour(X,Y,Z,150)
    cbar = plt.colorbar(C)
    plt.title(r"Likelihood $\log p(\mathcal{X}|\theta_A,\theta_B)$", fontsize=20)
    plt.xlabel(r"$\theta_A$", fontsize=20)
    plt.ylabel(r"$\theta_B$", fontsize=20)

    # plot thetas
    if thetas is not None:
        thetas = np.array(thetas)
        plt.plot(thetas[:,0], thetas[:,1], '-k', lw=2.0)
        plt.plot(thetas[:,0], thetas[:,1], 'ok', ms=5.0)


def coin_marginal_likelihood(rolls, biasA, biasB):
    # P(X | theta)
    trials = []
    for roll in rolls:
        h = roll.count("H")
        t = roll.count("T")
        likelihoodA = coin_likelihood(roll, biasA)
        likelihoodB = coin_likelihood(roll, biasB)
        trials.append(np.log(0.5 * (likelihoodA + likelihoodB)))
    return sum(trials)
```

In [4]:
```
plot_coin_likelihood(rolls, thetas)
```



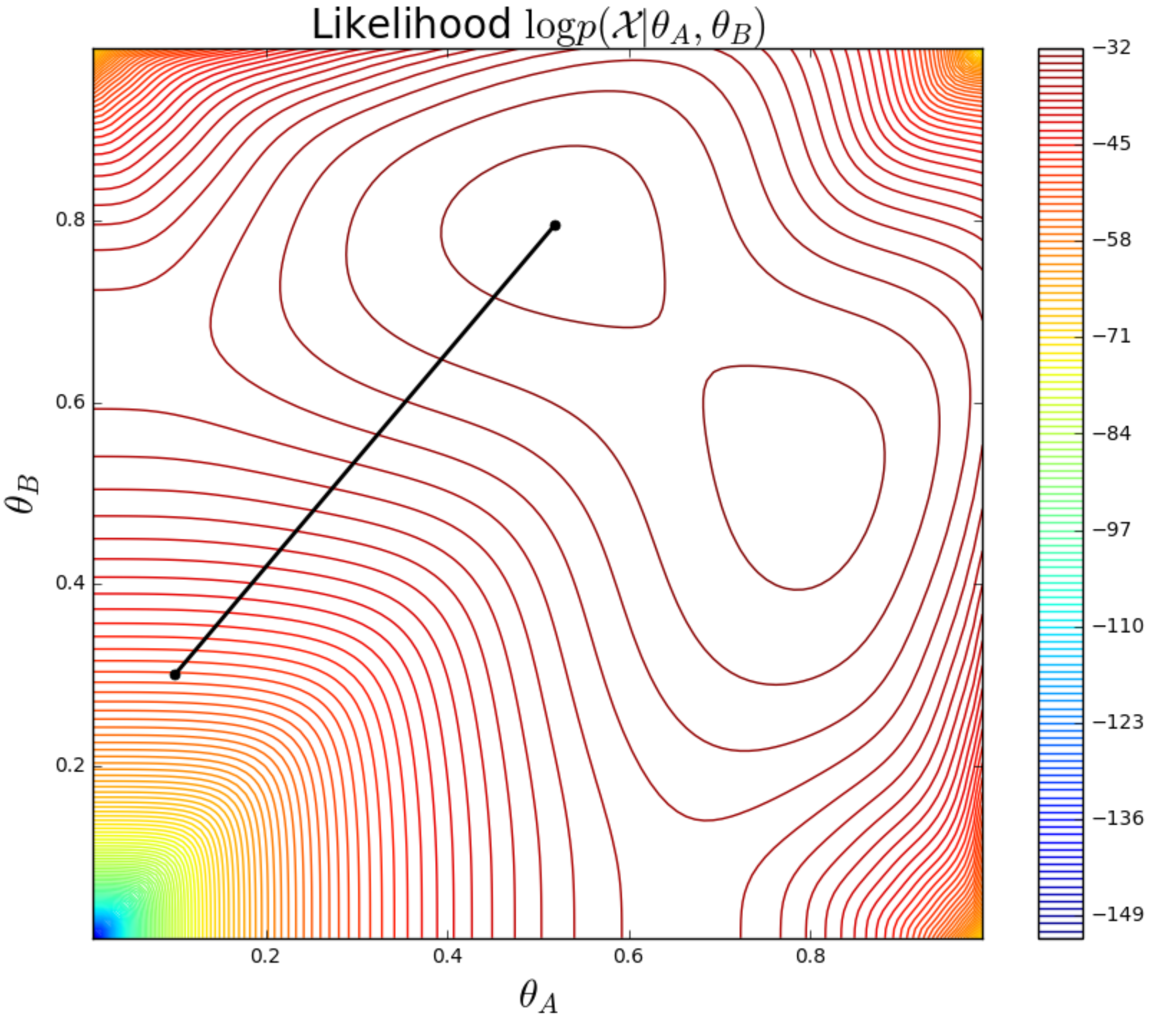
Another example

Let's run it again with different initial biases. Notice it converges to a different local optima.

In [5]:
```
thetas2, _ = coin_em(rolls, 0.1, 0.3, maxiter=6)
```

```
#0:	 0.10 0.30
#1:	 0.43 0.66
#2:	 0.50 0.75
#3:	 0.51 0.78
#4:	 0.52 0.79
#5:	 0.52 0.79
```

In [6]:
```
plot_coin_likelihood(rolls, thetas2)
```



Theory and wrapup

I hope working through this example painstakingly will help you gain some intuition for how EM works. It's also important to understand the theory behind how these steps map to maximizing the liklihood of a latent variable model, and to prove why each EM step increases the liklihood. For that, I refer you to the aforementioned University of Michigan lecture and discussion section notes.