Astro518-HW3

September 25, 2014

1 Astro518 Homework 3

1.1 Markov Chain Monte Carlo

In[1]:
```
from random import randint, uniform
import matplotlib.pyplot as plt
```

In[2]:
```
(data_id, x, y, dy, dx, rho_xy) = np.loadtxt('sample_data.txt', comments = '%', unpack = True) ## readin
```

1.2 Write the code and Run

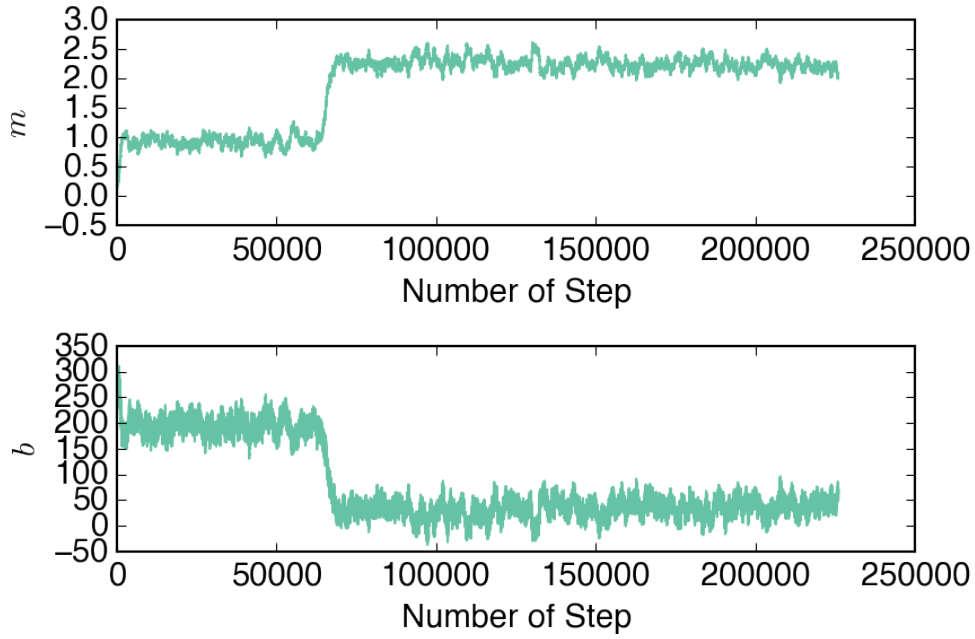
ML_MCMC.py program calls for 7 command parameters, which are intial values for: 1. m 2. b 3. Pb 4. Vb 5. Yb and the step length and step number. The 'base length' for every step is coded in the script, which are 0.01 for m, 10 for b, 0.1 for Pb, 400 for Vb and 70 for Yb. The real step in every Monte Carlo trial is the command line step multiplying the baselength.

For the first run, the m and b are set to values which make sure they are far away from the convergent values to see how many step needed for burning in. Here m0 equals to 0 and b0 equals to 200.

In[150]:
```
%%time
%run -i ML_MCMC.py 0 200 0.5 2000 300 0.5 1000000
np.savetxt("MCMC_First_run.dat", zip(m_list, b_list, Pb_list, Vb_list, Yb_list), fmt = '%.8f')
```

```
1000000
CPU times: user 1min 10s, sys: 265 ms, total: 1min 11s
Wall time: 1min 10s
```

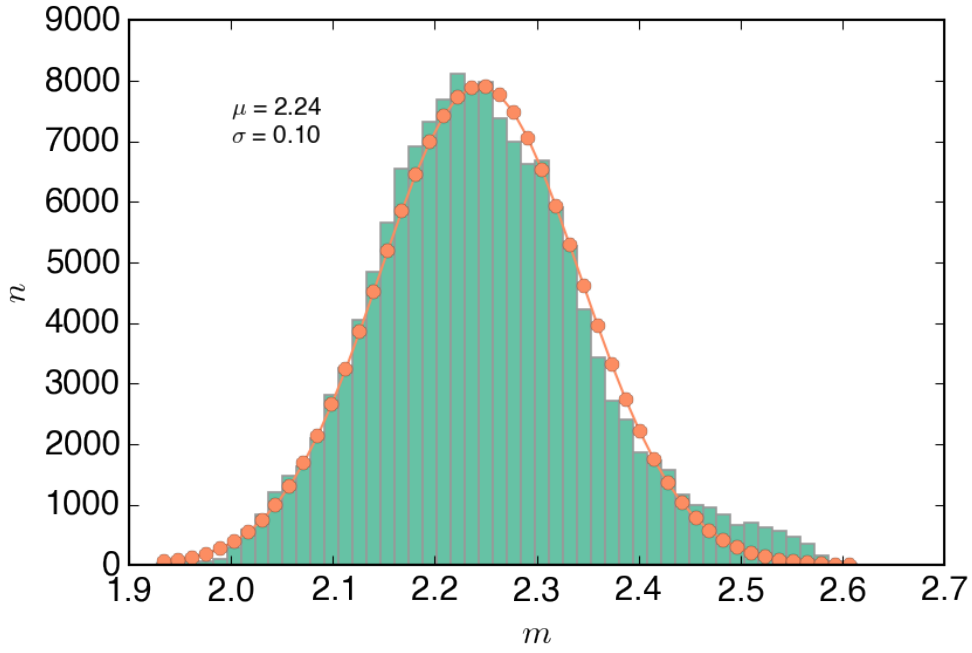
In[155]:
```
## How many points are needed to converge
fig = plt.figure()
m_ax = fig.add_subplot(211)
m_ax.plot(m_list)
m_ax.set_xlabel('Number of Step')
m_ax.set_ylabel('$m$')

b_ax = fig.add_subplot(212)
b_ax.plot(b_list)
b_ax.set_xlabel('Number of Step')
b_ax.set_ylabel('$b$')

fig.tight_layout()
```

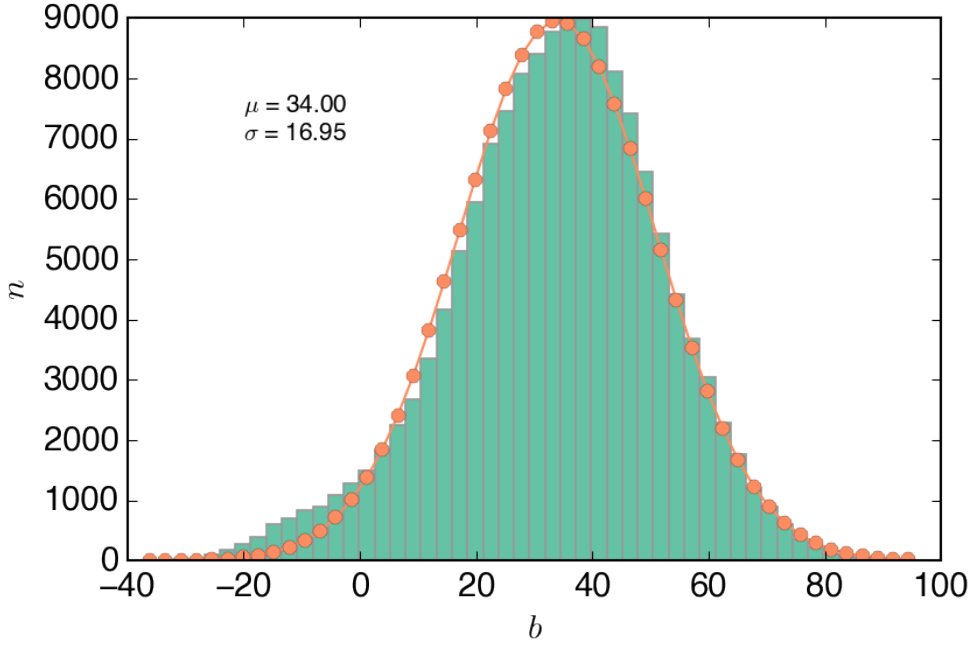


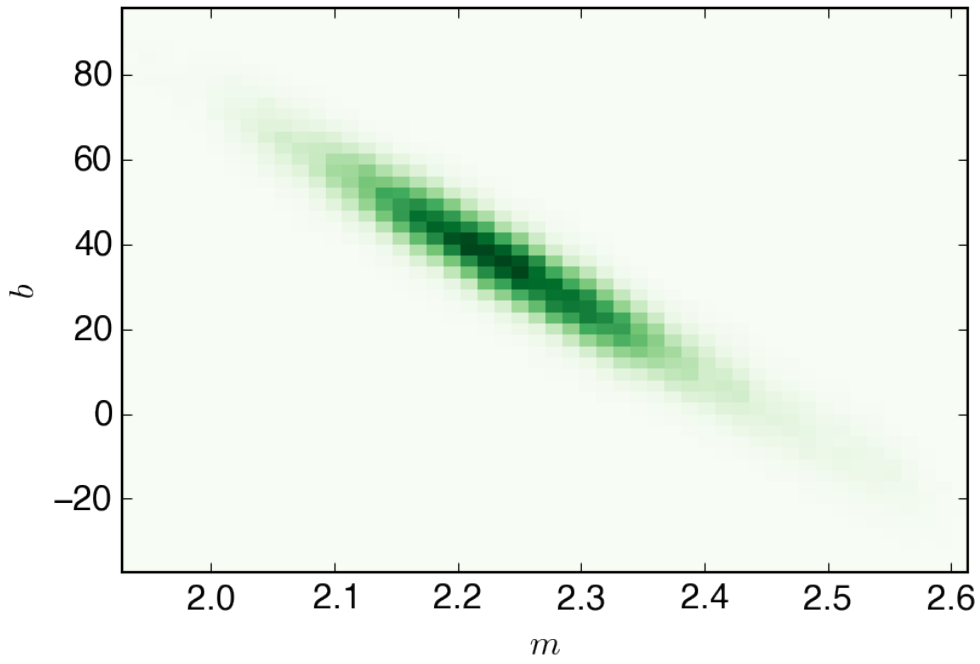
In[151]:
```
%run plot_hist.py
```

In[172]:
```
m_fig, m_gaussian = plotHist(m_list[80000:], [20000, 2.2, 0.5], 50)
m_ax = m_fig.axes[0]
m_ax.set_xlabel('$m$')
m_ax.set_ylabel('$n$')
mu_m = m_gaussian[1]
sigma_m = m_gaussian[2]
m_ax.text(2.0, 7000, '$\mu$ = {0:.2f}\n$\sigma$ = {1:.2f}'.format(mu_m, sigma_m))

b_fig, b_gaussian = plotHist(b_list[80000:], [20000, 30, 20], 50)
b_ax = b_fig.axes[0]
b_ax.set_xlabel('$b$')
b_ax.set_ylabel('$n$')
mu_b = b_gaussian[1]
sigma_b = b_gaussian[2]
b_ax.text(-20, 7000, '$\mu$ = {0:.2f}\n$\sigma$ = {1:.2f}'.format(mu_b, sigma_b))

## a 2D histogram
fig = plt.figure()
ax = fig.add_subplot(111)
bars = ax.hist2d(m_list[80000:], b_list[80000:], bins = 50)
ax.set_xlabel('$m$')
ax.set_ylabel('$b$')
```

Out[172]:
```
<matplotlib.text.Text at 0x1216f0190>
```

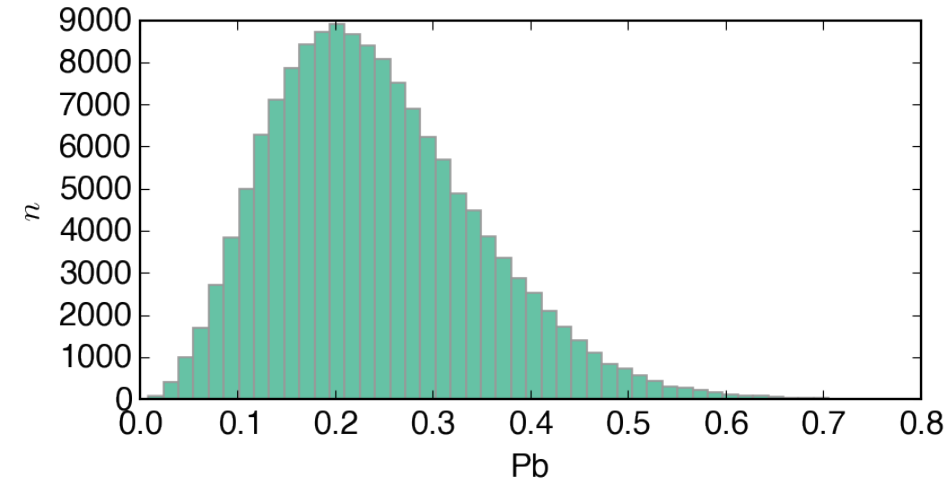

μ = 2.24
σ = 0.10
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
n
1.9
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
m

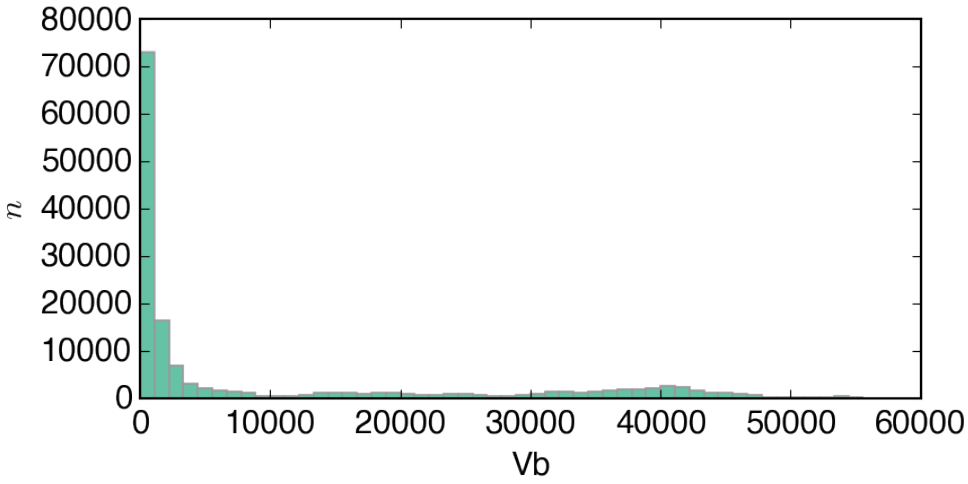

μ = 34.00
σ = 16.95
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
n
–40
–20
0
20
40
60
80
100
b

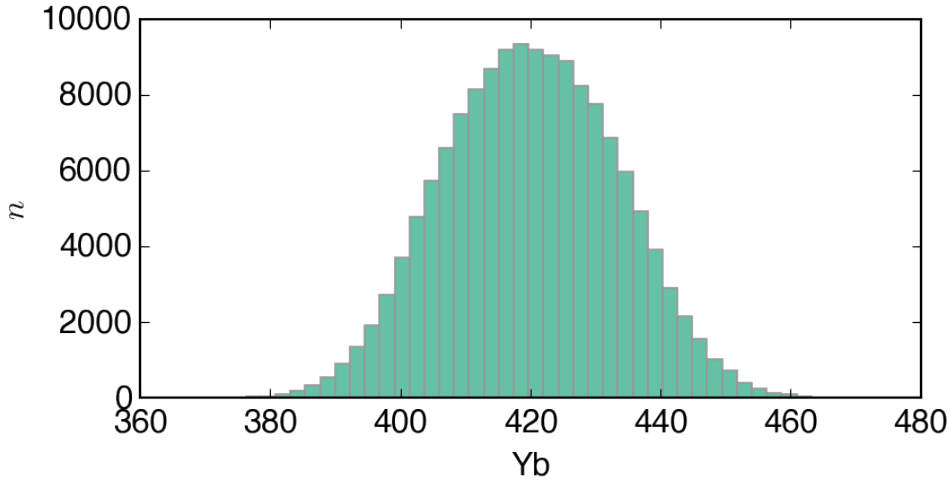


In[171]:

```
## Hist gram of Pb, Vb, Yb
fig = plt.figure(figsize = (6, 9))
ax_pb = fig.add_subplot(311)
bars = ax_pb.hist(Pb_list[80000:], bins = 50)
ax_pb.set_xlabel('Pb')
ax_pb.set_ylabel('$n$')

ax_Vb = fig.add_subplot(312)
bars = ax_Vb.hist(Vb_list[80000:], bins = 50)
ax_Vb.set_xlabel('Vb')
ax_Vb.set_ylabel('$n$')
ax_Yb = fig.add_subplot(313)
bars = ax_Yb.hist(Yb_list[80000:], bins = 50)
ax_Yb.set_xlabel('Yb')
ax_Yb.set_ylabel('$n$')
fig.tight_layout()
```







In[176]:
```
def confidInterval(data_list, interval_range = 0.95):
    """
    calculate interval limit with given interval range
```

```
    """
    n_data = len(data_list)
    data_list.sort()
    lower_id = int(n_data * (1 - interval_range)/2)
    upper_id = n_data - lower_id
    return data_list[lower_id], data_list[upper_id]

## for m
m_low, m_high = confidInterval(m_list[80000:])
print 'for m'
print '95% confident interval is {0:.3f} to {1:.3f}'.format(m_low, m_high)
print 'while 2 sigma interval is {0:.3f} to {1:.3f}\n'.format(mu_m- 2 * sigma_m, mu_m + 2 * sigma_m)
b_low, b_high = confidInterval(b_list[80000:])
print 'for b'
print '95% confident interval is {0:.3f} to {1:.3f}'.format(b_low, b_high)
print 'while 2 sigma interval is {0:.3f} to {1:.3f}\n'.format(mu_b- 2 * sigma_b, mu_b + 2 * sigma_b)
```

```
for m
95% confident interval is 2.055 to 2.490
while 2 sigma interval is 2.048 to 2.441

for b
95% confident interval is -6.508 to 66.249
while 2 sigma interval is 0.093 to 67.898
```

- The advantage of using 95% trust interval is that we do not need the distribution to be gaussian. For

1.3 Malmquist Bias

1.3.1 Generate supernovae and calculate mean distance

- To simplify the script, the following function uniformly generate supernovae in a box, and remove supernovae who have larger distance to the center than 2000 Mpc
- Analytically

$$\begin{aligned}\bar{d} &= \frac{\int_0^{d_0} r \cdot n \cdot r^2 \sin\theta dr d\theta d\phi}{\int_0^{d_0} n \cdot r^2 \sin\theta dr d\theta d\phi} \\ &= \frac{\int_0^{d_0} r^3 dr}{\int_0^{d_0} r^2 dr} \\ &= \frac{3d_0}{4}\end{aligned} \tag{1}$$

In[1]:
```
def SNgenesis(N, r = 2000):
    """
    generate supernovae uniformly distributed within a given distance
    """
    N = np.int(N * 6./np.pi) #generate more SN because we have to drop some
    x = np.random.uniform(-r, r, size = N)
    y = np.random.uniform(-r, r, size = N)
    z = np.random.uniform(-r, r, size = N)
    dis = np.sqrt(x**2 + y**2 + z**2)
    return dis[[dis < 2000]]

sn_dis = SNgenesis(100000)
print 'number of SN generated:', len(sn_dis)
print 'mean distance: {0:.2f}'.format(np.mean(sn_dis))
```

```
number of SN generated: 100090
mean distance: 1499.94
```

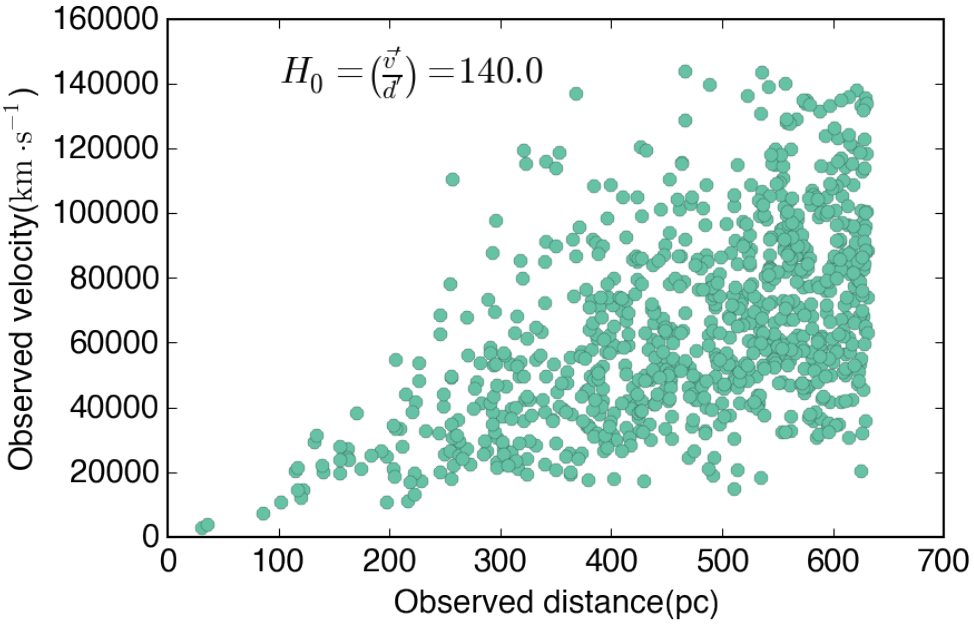
- distance calculated wiht Monte Carlo method agrees well with analytical result

In[10]:
```
def simObservation(samplesize = 10000):
    """
    carry out the simulated observation
    return the apparent magnitude, and velocity that measured
    """

    dis = SNgenesis(samplesize)
    velocity = dis * 72 # Hubble's law
    Mag = np.random.normal(0, 1, size = len(dis)) - 19
    mag = Mag + 5 * np.log10(dis * 1e6/10)

    return mag[mag < 20], velocity[mag < 20] # only return supernovae that have apparent magnitudes less

mag_ob, velocity_ob = simObservation(10000)
dis_ob = 10 * 10**((mag_ob - (-19))/5.) /1e6 #Mpc

print 'Observed Hubble constant:', np.mean(velocity_ob/dis_ob)
```

```
Observed Hubble constant: 140.004811072
```

In[14]:
```
plt.plot(dis_ob, velocity_ob, 'o')
ax = plt.gca()
ax.set_xlabel('Observed distance(pc)')
ax.set_ylabel('Observed velocity($\mathrm{km\cdot s^{-1}}$)')
ax.text(100, 140000, r"$H_0 = \bar{\left(\frac{v'}{d'}\right)} = 140.0$",fontsize = 16)
```

Out[14]: <matplotlib.text.Text at 0x114868c90>



In order to eliminate bias, we can use Bayesian analysis. We can set a prior distribution for the absolute magnitude of supernova, and calculate the maximum likelihood directly using grid or using Markov Chain Monte Carlo.

In[]: