Midterm Astro 410

by Swapnil Dubey

Included Files

- midterm.ipynb
- midterm.py
- midterm.pdf
- midterm.html
- midterm.dat

Introduction

For this midterm we were tasked with using Markov Chain Monte Carlo (MCMC) method to approximate and fit a gaussian function using data provided in the file "midterm.dat". We derive the joint posterior distribution of parameters \mu, \alpha_D and A. This allows us to contruct a Markov Chain to sample from. Running the specified 10,000 iterations of MCMC program over the retrieved sample of our parameters return an approximate value for a fit of the Gaussian model provided in the question.

Experiment

We are using the Metropolis algorithm to be specific

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import *

# Import data as Pandas DataFrame
df = pd.read_csv("midterm.dat", sep="\s+", header=None)

# Setting random seed value for consistent results over multiple runs
seed(4)
```

Gaussian

$$G(x, \mu, \alpha_D, A) = \frac{A}{\alpha_D}\sqrt{\frac{ln2}{\pi}}e^{\frac{-(ln2)(x-\mu)^2}{\alpha_D^2}}$$

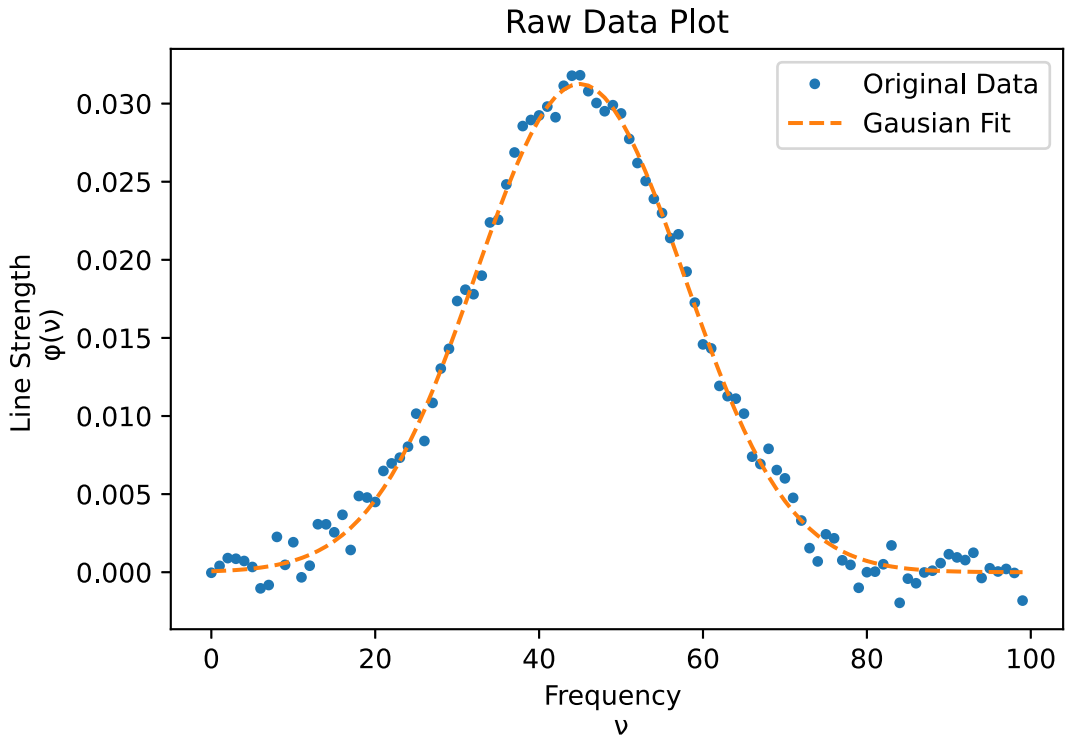
The above equation will be used to fit the data. To use this gaussian model, we need to predict the parameters such that we can reproduce our data from the same inputs. In homework 2 we attempted to do this using the Levenberg-Marquard algorithm. In that assignment our predicted parameters were the following:

$$\mu = 44.91914468 \pm 0.04116516$$
$$\alpha_D = 15.04509641 \pm 0.06398338$$

below is are plots from the previous lab depicting how close fit the new model was. For this midterm, we attempt to achieve the same task but using the MCMC algorithm.

In [2]:
```
#Def Function for our Gaussian
def G(x, mu, alpha_D, A):
    return ((A / alpha_D) * np.sqrt(np.log(2) / np.pi) * np.exp(-1 * np.log(2)
* (x-mu)**2 / alpha_D**2))

def P(y_i, m_i, sigma_i):
    return (q/sigma_i * np.sqrt(2 * np.pi)) * np.exp(-1 * (y_i-m_i)/ sigma_i *
np.sqrt(2))

# Use Guess values to plot and manually choose good starting guess for fitting
curve
# Guess as dict
guess = {'x':43, 'mu':44.954, 'alpha_D':15.027, 'A':1}
# Guess as array; v_0: alpha_L
guess1 = [43, 8]


# Calculating output with our guessed inputs so we can build plot
n = len(df[1])
y = np.empty(n)

for i in range(n):
    y[i] = G(df[0][i],guess['mu'], guess['alpha_D'], guess['A'])

plt.errorbar(df[0], df[1], fmt='.')
plt.title("Raw Data Plot")
plt.errorbar(df[0], y, fmt='--')
plt.ylabel("Line Strength \n \u03C6(\u03BD)")
plt.xlabel("Frequency \n \u03BD")
plt.legend(['Original Data', 'Gausian Fit'])
```

Out[2]: <matplotlib.legend.Legend at 0x158ff57eac8>



MCMC is a class of methods. Metropolis-Hastings is a specific implementation of MCMC. It works well in high dimensional spaces as opposed to Gibbs sampling and rejection sampling. This technique requires a simple distribution called the proposal distribution **Q(θ′/θ)** to help draw samples from an intractable posterior distribution **P(Θ = θ/D)**. Metropolis-Hastings uses **Q** to randomly walk in the distribution space, accepting or rejecting jumps to new positions based on how likely the sample is. This “memoriless” random walk is the “Markov Chain” part of MCMC.

The “likelihood” of each new sample is decided by a function **f** . That’s why **f** must be proportional to the posterior we want to sample from. **f** is commonly chosen to be a probability density function that expresses this proportionality.

To get a new position of the parameter, just take our current one, **θ**, and propose a new one , **θ’**, that is a random sample drawn from **Q(θ′/θ)**. Often this is a symmetric distribution. For instance, a normal distribution with mean **θ** and some standard deviation

$$\sigma : Q(\theta'/\theta) = N(\theta, \sigma)$$

To decide if **θ′** is to be accepted or rejected, the following ratio must be computed for each new proposed **θ’**:

$$\frac{P(\theta'/D)}{P(\theta/D)}$$

Using Bayes’ formula this can be easily re-formulated as:

$$\frac{P(D/\theta')P(\theta')}{P(D/\theta)P(\theta)}$$

(The evidence P(D) is simply crossed out during the division)

In [3]:
```
def gaussian(x_i, p):
    a0 = 1.177410225 #sqrt(2ln(2))
    a1 = 0.3989423 #1/sqrt(2*pi)
    s = p[1]/a0
    numer = (x_i-p[0])/s
    m = a1*p[2]/s * np.exp(-0.5*numer**2)
    return m

#this computes the -ln(likelyhood)
def mlnlikely(p, d):
    xd = d[0]
    yd = d[1]
    sd = d[2]
    m = gaussian(xd, p)
    r = 0.5*((yd-m)/sd)**2 # ignore the factor 1/sqrt(2*pi)/sd, because it doe
s not depend on the model parameters.
    return sum(r)

def mcmc(d, p0, s0, nm):
    np = len(p0)
    nq = np+2
    #first dim of p is parameter index, 2nd dim is the chain iteration index
    #p[0:np,i] store the np parameters of i-th iteration
    #p[np,i] store the -ln(likelyhood) of i-th iteration
    #p[np+1,i] store the rejection probability i-th iteration
    p = zeros((nq,nm))
    #copy the inital parameter into p[:np,0]
    for ip in range(np):
        p[ip,0] = p0[ip]
    #compute the -ln(likelyhood) for the initial parameters
    p[np,0] = mlnlikely(p0, d)
    #iterate along the chain
    for i in range(1,nm):
        #random jump from i-1 iteration to new parameters pnew
        #x is the np random variables uniformly distributed from -1 to 1
        x = 2*random(np)-1.0
        pnew = zeros(np)
        for ip in range(np):
            pnew[ip] = p[ip,i-1] + x[ip]*s0[ip]
        #compute the -ln(likelyhood) for the new parameters, store in p[np,i]
        p[np,i] = mlnlikely(pnew, d)
        if p[np,i] <= p[np,i-1]:
            #if p[np,i] <= p[np,i-1], the new parameters are accepted
            for ip in range(np):
                p[ip,i] = pnew[ip]
        else:
            #if p[np,i] > p[np,i-1], the new parameters are accepted with prob
ability of r = exp(-(p[np,i]-p[np,i-1]))
            r = exp(-(p[np,i]-p[np,i-1]))
            #1-r is the rejection probability, store in p[np+1,i]
            p[np+1,i] = 1-r
            y = random()
            if (y > r):
                #reject pnew, copy p[:,i-1] into p[:,i]
                for ip in range(np+1):
                    p[ip,i] = p[ip,i-1]
```

```
            else:
                #accept pnew, copy pnew into p[:,i]
                for ip in range(np):
                    p[ip,i] = pnew[ip]

        if (i+1)%100 == 0:
            print('%6d %10.3E %10.3E %10.3E %10.3E %10.3E %10.3E'%(i,p[np+1,i
],p[np,i],p[0,i],p[1,i],p[2,i],mean(p[np+1,:i+1])))
    return p
```

After building the gaussian and MCMC funtions, we just have to set the initial parameters (guess) and other inputs, such as the MCMC widths (s0) and how many cycles the algorithm should run (nm). We chose the 'nm' value to be 10,000 hoping to see them converge well before that many iterations. If the algorithm is not able to achieve stability or seems to be stuck in a local minima instead of global minima, we may need to change the input values.

In [4]:
```
# Initial guess for Parameters
#     mu,  alpha_D, A
guess = [40, 15, 2]

# Optimal MCMC widths
# s0 = [0.06, 0.04, 0.0025]
s0 = [0.06, 0.04, 0.0025]

# Number of Iterations
nm = 10_000

# Gives us probability for each Parameter
p = mcmc(df, guess, s0, nm)
```

99	1.000E+00	6.969E+04	3.987E+01	1.589E+01	1.971E+00	5.200E-01
199	0.000E+00	5.708E+04	3.907E+01	1.643E+01	1.947E+00	5.500E-01
299	1.000E+00	4.784E+04	3.848E+01	1.695E+01	1.929E+00	5.533E-01
399	1.000E+00	3.778E+04	3.811E+01	1.751E+01	1.881E+00	5.425E-01
499	0.000E+00	2.990E+04	3.730E+01	1.788E+01	1.841E+00	5.333E-01
599	1.000E+00	2.283E+04	3.655E+01	1.842E+01	1.795E+00	5.377E-01
699	0.000E+00	1.774E+04	3.576E+01	1.913E+01	1.759E+00	5.266E-01
799	0.000E+00	1.359E+04	3.523E+01	1.967E+01	1.697E+00	5.193E-01
899	0.000E+00	1.159E+04	3.495E+01	2.002E+01	1.653E+00	5.236E-01
999	1.000E+00	9.544E+03	3.481E+01	2.014E+01	1.584E+00	5.155E-01
1099	0.000E+00	8.029E+03	3.503E+01	2.028E+01	1.517E+00	5.112E-01
1199	9.998E-01	6.790E+03	3.557E+01	2.014E+01	1.454E+00	5.097E-01
1299	1.000E+00	5.878E+03	3.620E+01	1.992E+01	1.407E+00	5.091E-01
1399	1.000E+00	5.186E+03	3.680E+01	1.964E+01	1.374E+00	5.158E-01
1499	1.000E+00	4.437E+03	3.748E+01	1.918E+01	1.340E+00	5.165E-01
1599	1.000E+00	3.552E+03	3.833E+01	1.875E+01	1.291E+00	5.144E-01
1699	9.991E-01	2.886E+03	3.904E+01	1.844E+01	1.248E+00	5.193E-01
1799	0.000E+00	2.135E+03	3.998E+01	1.783E+01	1.203E+00	5.183E-01
1899	1.000E+00	1.723E+03	4.062E+01	1.746E+01	1.179E+00	5.231E-01
1999	0.000E+00	1.186E+03	4.156E+01	1.707E+01	1.140E+00	5.226E-01
2099	1.000E+00	8.197E+02	4.210E+01	1.678E+01	1.097E+00	5.249E-01
2199	1.000E+00	4.932E+02	4.295E+01	1.625E+01	1.081E+00	5.254E-01
2299	8.921E-01	2.071E+02	4.380E+01	1.571E+01	1.047E+00	5.267E-01
2399	9.993E-01	7.485E+01	4.449E+01	1.530E+01	1.019E+00	5.302E-01
2499	4.559E-01	5.240E+01	4.486E+01	1.506E+01	1.003E+00	5.348E-01
2599	9.498E-01	5.185E+01	4.484E+01	1.508E+01	1.004E+00	5.417E-01
2699	8.876E-01	5.108E+01	4.490E+01	1.506E+01	1.004E+00	5.465E-01
2799	8.407E-01	5.180E+01	4.495E+01	1.501E+01	1.003E+00	5.520E-01
2899	9.153E-01	5.429E+01	4.489E+01	1.508E+01	1.006E+00	5.541E-01
2999	8.340E-01	5.242E+01	4.483E+01	1.511E+01	1.003E+00	5.595E-01
3099	7.473E-01	5.242E+01	4.481E+01	1.512E+01	1.005E+00	5.627E-01
3199	6.338E-01	5.118E+01	4.494E+01	1.504E+01	1.001E+00	5.651E-01
3299	9.995E-01	5.154E+01	4.487E+01	1.507E+01	1.005E+00	5.681E-01
3399	1.264E-01	5.107E+01	4.493E+01	1.504E+01	1.002E+00	5.694E-01
3499	1.000E+00	5.105E+01	4.489E+01	1.506E+01	1.003E+00	5.700E-01
3599	4.084E-01	5.116E+01	4.494E+01	1.503E+01	1.003E+00	5.716E-01
3699	1.000E+00	5.152E+01	4.490E+01	1.507E+01	1.003E+00	5.732E-01
3799	8.189E-01	5.168E+01	4.493E+01	1.502E+01	1.004E+00	5.737E-01
3899	5.140E-01	5.361E+01	4.481E+01	1.512E+01	1.007E+00	5.754E-01
3999	0.000E+00	5.297E+01	4.480E+01	1.512E+01	1.004E+00	5.773E-01
4099	9.608E-01	5.274E+01	4.487E+01	1.509E+01	1.001E+00	5.807E-01
4199	8.564E-01	5.254E+01	4.495E+01	1.505E+01	9.997E-01	5.818E-01
4299	9.788E-01	5.130E+01	4.496E+01	1.502E+01	1.002E+00	5.845E-01
4399	8.030E-01	5.239E+01	4.497E+01	1.503E+01	9.982E-01	5.855E-01
4499	0.000E+00	5.228E+01	4.484E+01	1.511E+01	1.003E+00	5.872E-01
4599	1.000E+00	5.211E+01	4.482E+01	1.510E+01	1.004E+00	5.899E-01
4699	6.884E-01	5.217E+01	4.493E+01	1.502E+01	1.000E+00	5.931E-01
4799	8.930E-01	5.334E+01	4.490E+01	1.508E+01	1.004E+00	5.930E-01
4899	4.578E-01	5.253E+01	4.482E+01	1.509E+01	1.006E+00	5.942E-01
4999	0.000E+00	5.193E+01	4.487E+01	1.509E+01	1.002E+00	5.955E-01
5099	6.370E-01	5.236E+01	4.495E+01	1.504E+01	9.986E-01	5.961E-01
5199	5.053E-01	5.305E+01	4.488E+01	1.505E+01	1.001E+00	5.969E-01
5299	4.945E-01	5.261E+01	4.492E+01	1.502E+01	1.003E+00	5.983E-01
5399	8.538E-01	5.170E+01	4.498E+01	1.502E+01	9.996E-01	6.002E-01
5499	2.816E-01	5.263E+01	4.498E+01	1.502E+01	1.003E+00	6.011E-01
5599	8.555E-02	5.253E+01	4.485E+01	1.507E+01	1.003E+00	6.028E-01
5699	0.000E+00	5.124E+01	4.494E+01	1.504E+01	1.002E+00	6.039E-01

5799	3.134E-01	5.475E+01	4.486E+01	1.511E+01	9.999E-01	6.052E-01
5899	0.000E+00	5.296E+01	4.497E+01	1.503E+01	9.976E-01	6.061E-01
5999	0.000E+00	5.319E+01	4.479E+01	1.512E+01	1.005E+00	6.071E-01
6099	2.651E-02	5.302E+01	4.481E+01	1.512E+01	1.005E+00	6.076E-01
6199	0.000E+00	5.531E+01	4.504E+01	1.498E+01	1.003E+00	6.084E-01
6299	7.037E-01	5.339E+01	4.496E+01	1.504E+01	1.003E+00	6.082E-01
6399	3.854E-01	5.232E+01	4.488E+01	1.509E+01	1.001E+00	6.094E-01
6499	9.995E-01	5.098E+01	4.493E+01	1.505E+01	1.002E+00	6.111E-01
6599	0.000E+00	5.100E+01	4.492E+01	1.504E+01	1.003E+00	6.110E-01
6699	9.800E-01	5.156E+01	4.494E+01	1.504E+01	1.003E+00	6.114E-01
6799	8.173E-01	5.158E+01	4.491E+01	1.504E+01	1.002E+00	6.124E-01
6899	8.849E-01	5.307E+01	4.479E+01	1.513E+01	1.005E+00	6.133E-01
6999	8.846E-01	5.162E+01	4.491E+01	1.506E+01	1.004E+00	6.144E-01
7099	9.803E-01	5.131E+01	4.489E+01	1.506E+01	1.004E+00	6.148E-01
7199	9.550E-01	5.230E+01	4.501E+01	1.498E+01	9.988E-01	6.151E-01
7299	9.958E-01	5.410E+01	4.478E+01	1.514E+01	1.007E+00	6.153E-01
7399	2.988E-01	5.246E+01	4.490E+01	1.508E+01	1.004E+00	6.160E-01
7499	8.983E-01	5.107E+01	4.495E+01	1.503E+01	1.001E+00	6.157E-01
7599	1.000E+00	5.196E+01	4.496E+01	1.500E+01	1.001E+00	6.162E-01
7699	4.251E-01	5.242E+01	4.495E+01	1.501E+01	1.000E+00	6.168E-01
7799	0.000E+00	5.126E+01	4.492E+01	1.504E+01	1.004E+00	6.172E-01
7899	2.070E-02	5.174E+01	4.496E+01	1.500E+01	1.002E+00	6.173E-01
7999	1.000E+00	5.139E+01	4.493E+01	1.504E+01	1.003E+00	6.179E-01
8099	9.319E-01	5.374E+01	4.506E+01	1.496E+01	9.987E-01	6.188E-01
8199	7.421E-01	5.227E+01	4.489E+01	1.506E+01	1.000E+00	6.201E-01
8299	9.909E-01	5.170E+01	4.489E+01	1.507E+01	1.001E+00	6.207E-01
8399	9.885E-01	5.217E+01	4.483E+01	1.509E+01	1.006E+00	6.210E-01
8499	1.866E-01	5.239E+01	4.494E+01	1.506E+01	1.001E+00	6.208E-01
8599	9.065E-01	5.097E+01	4.491E+01	1.505E+01	1.003E+00	6.212E-01
8699	1.905E-01	5.280E+01	4.481E+01	1.510E+01	1.004E+00	6.211E-01
8799	8.092E-01	5.340E+01	4.488E+01	1.505E+01	1.000E+00	6.209E-01
8899	9.691E-01	5.319E+01	4.496E+01	1.500E+01	1.004E+00	6.215E-01
8999	0.000E+00	5.153E+01	4.491E+01	1.504E+01	1.002E+00	6.214E-01
9099	8.417E-01	5.436E+01	4.488E+01	1.511E+01	1.003E+00	6.221E-01
9199	1.000E+00	5.136E+01	4.486E+01	1.508E+01	1.003E+00	6.229E-01
9299	0.000E+00	5.105E+01	4.494E+01	1.503E+01	1.002E+00	6.238E-01
9399	0.000E+00	5.172E+01	4.499E+01	1.500E+01	9.994E-01	6.238E-01
9499	6.097E-01	5.309E+01	4.480E+01	1.511E+01	1.005E+00	6.239E-01
9599	1.000E+00	5.553E+01	4.501E+01	1.501E+01	1.003E+00	6.242E-01
9699	5.978E-01	5.602E+01	4.495E+01	1.504E+01	9.950E-01	6.250E-01
9799	5.494E-01	5.295E+01	4.490E+01	1.504E+01	1.006E+00	6.254E-01
9899	8.736E-01	5.330E+01	4.482E+01	1.511E+01	1.008E+00	6.261E-01
9999	1.816E-01	5.239E+01	4.501E+01	1.500E+01	9.982E-01	6.264E-01

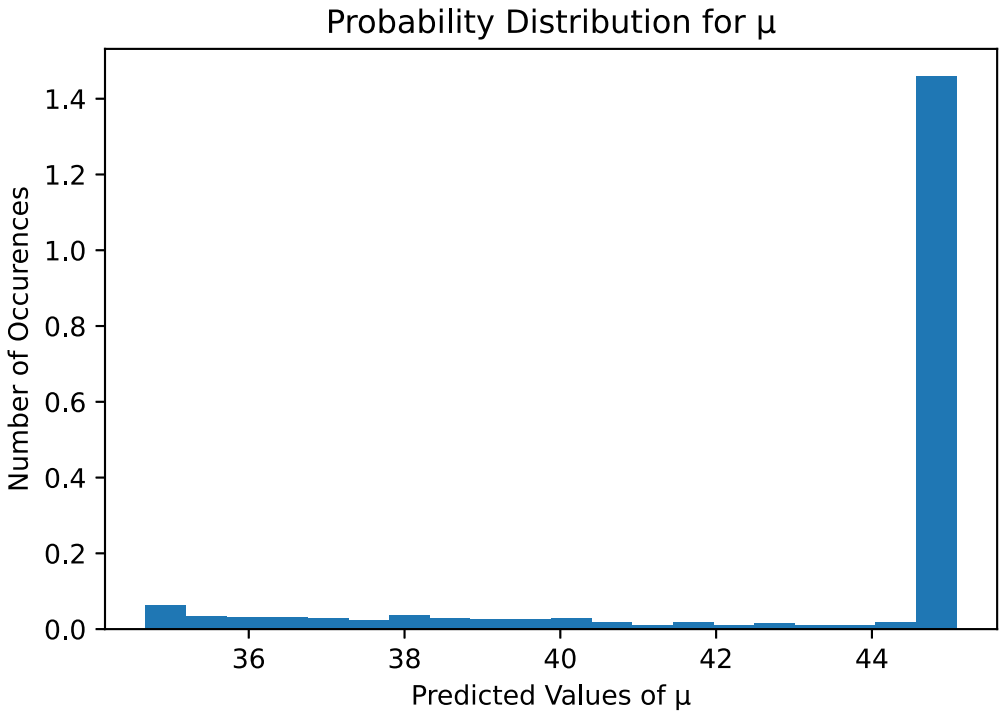
Probability Distribution

Below are histograms that plot the number of occurances of a certain number in each bin. This allows us to view the highly skewed probability distribution for each of the parameters that we estimated using MCMC Metropolis sampling. In all three plots, there is a huge spike in the number of frequency of the predicted value while other values have little to no height indicating a lack of probability.

PS: I realise that these are not probability distribution plots, but for some reason the 'normed=True' parameter in hist() is not working for me and giving an error. Had it worked, sum of the bins would be 1 and therefore represent a proper probability distribution.

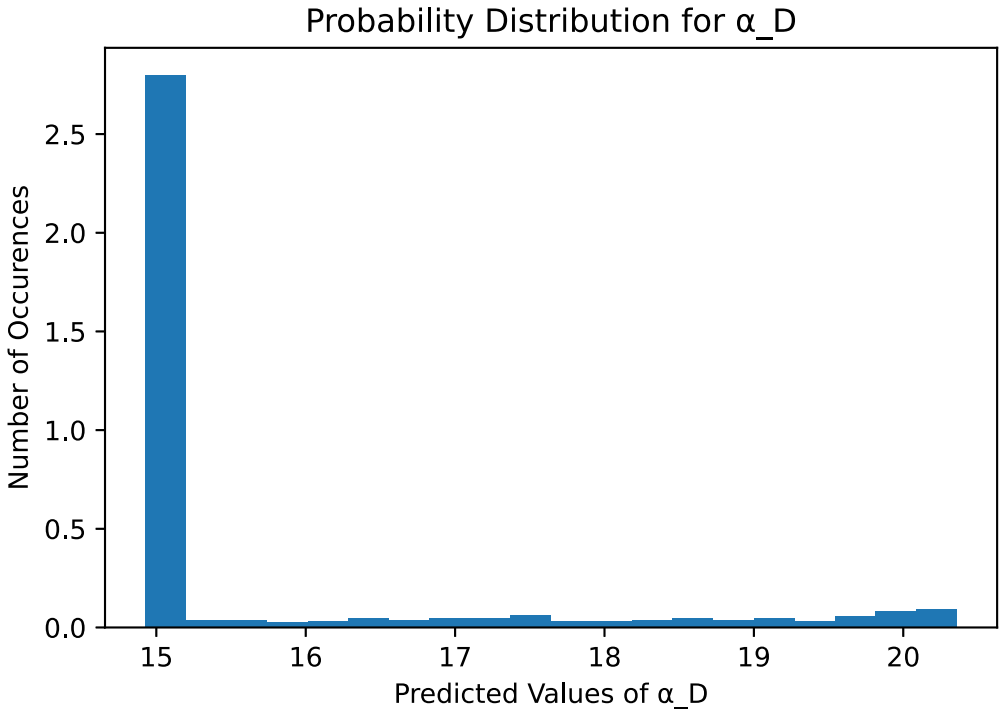
In [5]:
```
# Histogram showing the frequency of occurance for a given value range in the
 predicted markov chain
plt.hist(p[0], bins=20, density=True, stacked=True)
plt.title("Probability Distribution for \u03BC")
plt.ylabel("Number of Occurences")
plt.xlabel("Predicted Values of \u03BC")
```

Out[5]:
```
Text(0.5, 0, 'Predicted Values of μ')
```



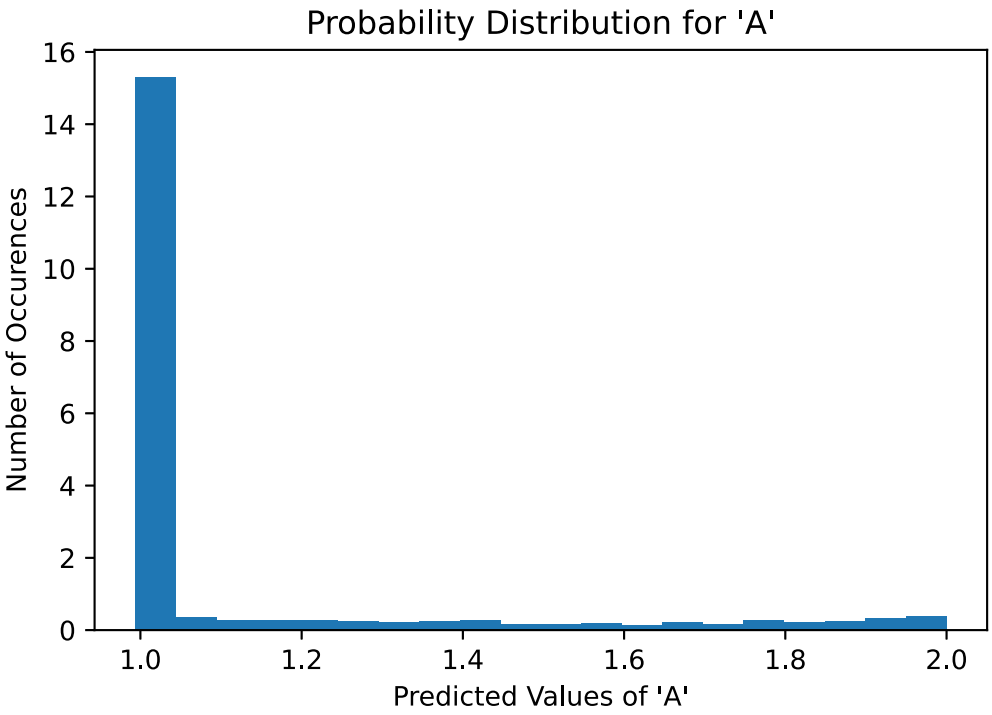
In [6]:
```
plt.hist(p[1], bins=20, density=True, stacked=True)
plt.title("Probability Distribution for \u03B1_D")
plt.ylabel("Number of Occurences")
plt.xlabel("Predicted Values of \u03B1_D")
```

Out[6]: Text(0.5, 0, 'Predicted Values of α_D')



In [7]:
```
plt.hist(p[2], bins=20, density=True, stacked=True)
plt.title("Probability Distribution for 'A'")
plt.ylabel("Number of Occurences")
plt.xlabel("Predicted Values of 'A'")
```

Out[7]:
```
Text(0.5, 0, "Predicted Values of 'A'")
```

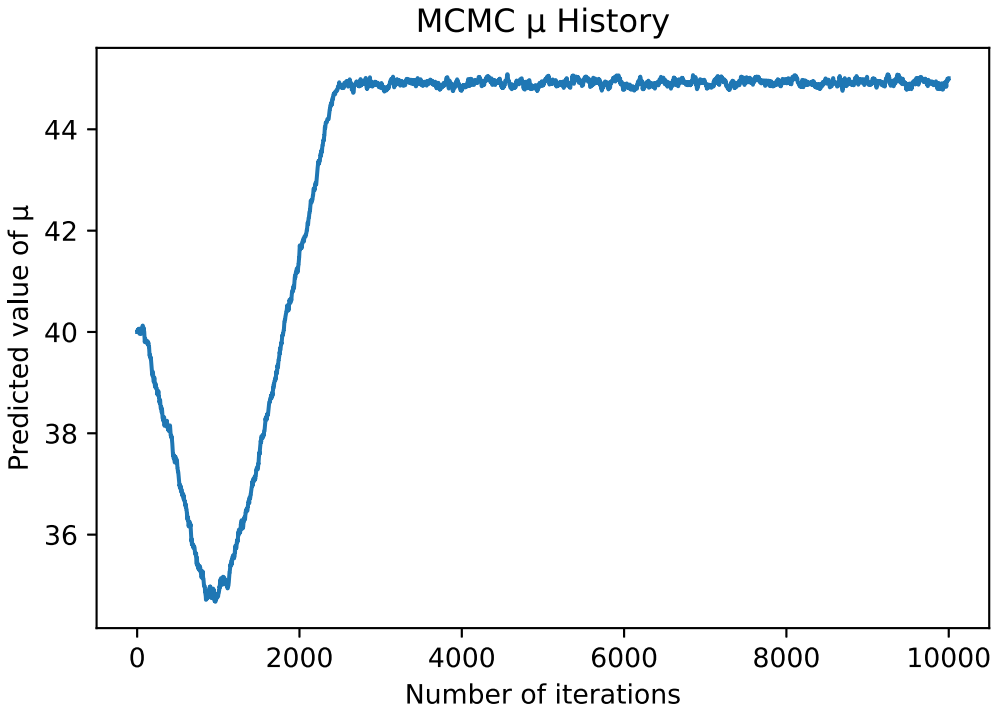


MCMC History Plots

Each plot below shows the predicted value of it's corresponding parameter at every instance of the iteration. In our case we iterated for 10,000 cycles, so the graphs plot over 10,000 x values.

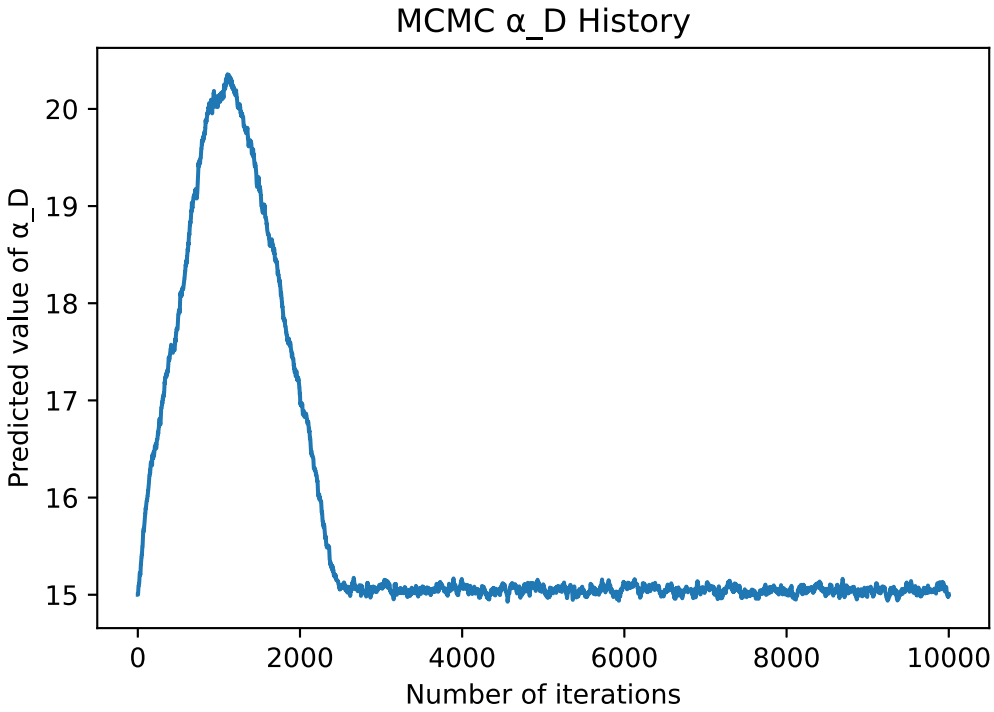
In [8]:
```
# Graph Estimating parameter mu
plt.errorbar(range(0,10000), p[0], fmt='')
plt.title("MCMC \u03BC History")
plt.ylabel("Predicted value of \u03BC")
plt.xlabel("Number of iterations")
```

Out[8]:
```
Text(0.5, 0, 'Number of iterations')
```



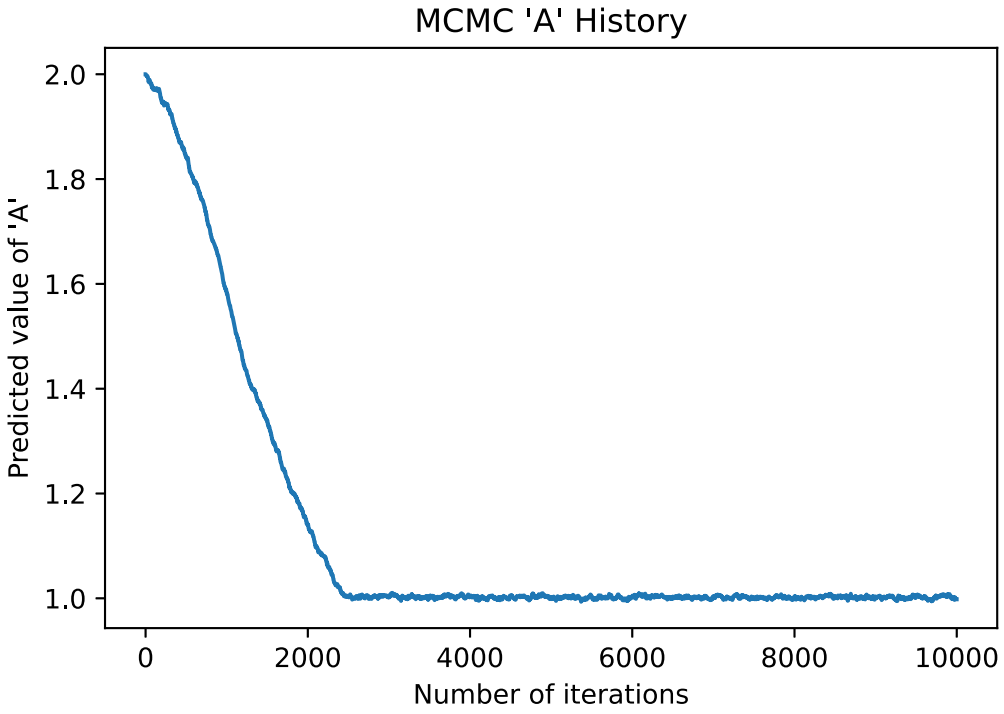
In [9]:
```
# Graph Estimating parameter Alpha_D
plt.errorbar(range(0,10000), p[1], fmt='')
plt.title("MCMC \u03B1_D History")
plt.ylabel("Predicted value of \u03B1_D")
plt.xlabel("Number of iterations")
```

Out[9]:
```
Text(0.5, 0, 'Number of iterations')
```



In [10]:
```
# Graph Estimating parameter A
plt.errorbar(range(0,10000), p[2], fmt='')
plt.title("MCMC 'A' History")
plt.ylabel("Predicted value of 'A'")
plt.xlabel("Number of iterations")
```

Out[10]: Text(0.5, 0, 'Number of iterations')



Viewing the plots above, it is obvious that the burn-in period is upto at least 2000 cycles/iterations, after which all 3 parameters begin to converge to a value similar to what we observed for our Homework 2, using Levenberg-Marquard algorithm. From the plots above, we can approximate the parameters as:

$$\mu = 44.9$$
$$\alpha_D = 15.1$$
$$A = 1.006$$

Conclusion

In conclusion, the MCMC algorithm can produce very accurate results but is very sensitive and can take many iterations to reach a stable value. Even in the plots above, the beginning parameter guess are not too far off from the final values and yet the burn-in period is at least 2,000 cycles. Also by just changing one of the parameters by a small value can affect the burn-in period for all other parameters by a lot. For example, changing the starting point for parameter A to 2 drastically increases the burn-in period, especially if the specified widths in the variable "s0" are not the optimal widths (0.06, 0.04, 0.0025).

Due to the above mentioned inconviniences, I personally would recommend using Levenberg-Marquard algorithm to fit this specific model. It is much faster to compute but will unfortunately not be sufficient for models more complex than in this midterm with many more dimensions/parameters.