Bayesian Inference with Markov Chain Monte Carlo (MCMC) algorithm in python from scratch

- 1) Metropolis Hastings
- 2) Metropolis-adjusted Langevin

The MCMC method is a class of techniques for sampling from a probability distribution and can be used to estimate the distribution of parameters given a set of observations and a prior belief.

For the purpose of learning, the following is implment from scratch with *numpy* (a gerneral purupose package providing high-performance mulitdimensional arrays objects), *scipy* (for validating density functions) and visualisation library *matplotlib*.

With the discrete-state-space Markov chain theory, let the sequence $\{X^{(t)}\}$ denote a Markov chain for t = 0,1,2.. where $X^{(t)} = \left(X_1^{(t)}, \ldots, X_p^{(t)}\right)$ and the state space is either continuous or discrete.

Metropolis Hastings

According to (Givens, 2013) a general method for construction of MCMC is the Metropolis-Hastings algorithm. The method starts with t = 0 with the selection of $X^{(0)} = x^{(0)}$ drawn at random from statring distribution *g*, with the reqirement that $f(x^{(0)}) > 0$. Given $X^{(t)} = x^{(t)}$, the algorithm generates $X^{(t+1)}$ as follows:

1. Sample a candidate value X^* from a *proposal distribution* g $\left(. \,|x^{(t)}\right)$
2. Compute the *Metropolis-Hastings ratio* R$\left(x^{(t)}, X^*\right)$,where

$$R(u, v) = \frac{f(v)g(u|v)}{f(u)g(u|v)}$$

1. Sample a value for $X^{(t+1)}$ given the following:

$$X^{(t+1)} = \begin{cases} X^*, \text{ with probability } \min\left\{R\left(x^{(t)}, X^*\right), 1\right\}, \\ x^{(t)}, \text{ otherwise} \end{cases}$$

1. Increment *t and return to step 1.

Bayesian Process

For step 1. the proposal distribution g. (coded as 'transition model') helps draw samples from intractable posterior distribution. Metropolis-Hastings uses g to randomly walk in the distribution space, accepting or rejecting new positions based on how likely the sample (describe in step 2&3).

For new position of paramter θ' ,take current parameter θ. Randomly sample drawn from $g(\theta'|\theta)$. Using Bayes formula the ratio θ' : $\frac{P(\theta'|D)}{P(\theta|D)}$) can be shown as $\frac{P(D|\theta')P(\theta')}{P(D|\theta)P(\theta)}$) = $\frac{\prod_i^n f(d_i|\Theta=\theta')P(\theta')}{\prod_i^n f(d_i|\Theta=\theta)P(\theta)}$

$$P(accepted) = \begin{cases} \frac{\prod_i^n f(d_i|\Theta=\theta')P(\theta')}{\prod_i^n f(d_i|\Theta=\theta)P(\theta)}, \prod_i^n f(d_i|\Theta=\theta)P(\theta) > \prod_i^n f(d_i|\Theta=\theta')P(\theta') \\ 1\ , \prod_i^n f(d_i|\Theta=\theta)P(\theta) \leq \prod_i^n f(d_i|\Theta=\theta')P(\theta') \end{cases}$$

Therefore, θ' is more likely than current θ.

Algorithm Process

For simplicity we will initially adopt a normal distribution for both target and proposed distribution. However will attempt to build single function to adapt to paramters/distribution and MH/MALA algorithm.

Model

We first simulate this model with the intial parameters:

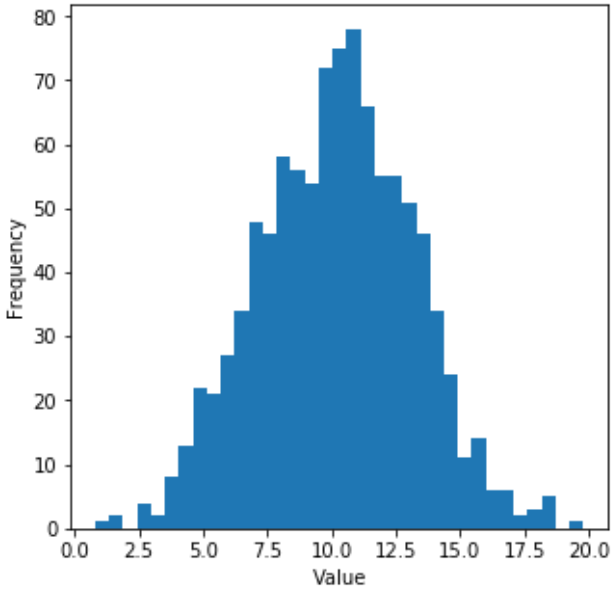
- μ = 50
- σ = 5

Normal data generation from 30,000 samples observing 1,000. For the observed paramters:

- μ ~ 10
- σ ~ 3

Algorithm iterates 50000 times, where accepted and rejected values are recorded for burn-in.

Figure 1: Distribution of 1000 observations sampled from a population of 30,000 with μ=10, σ=3

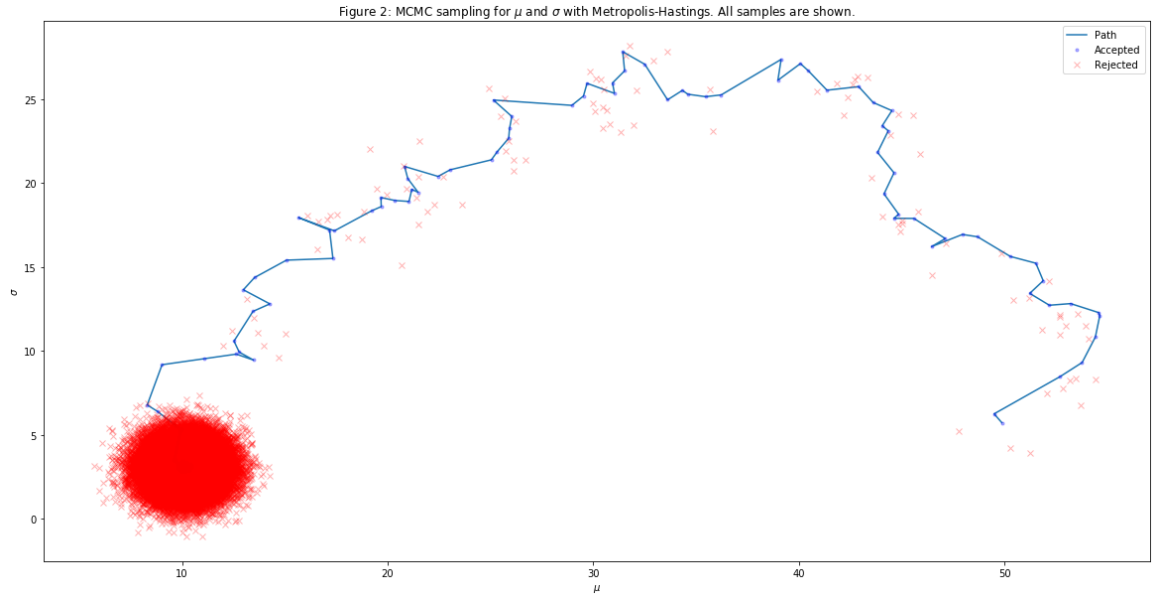


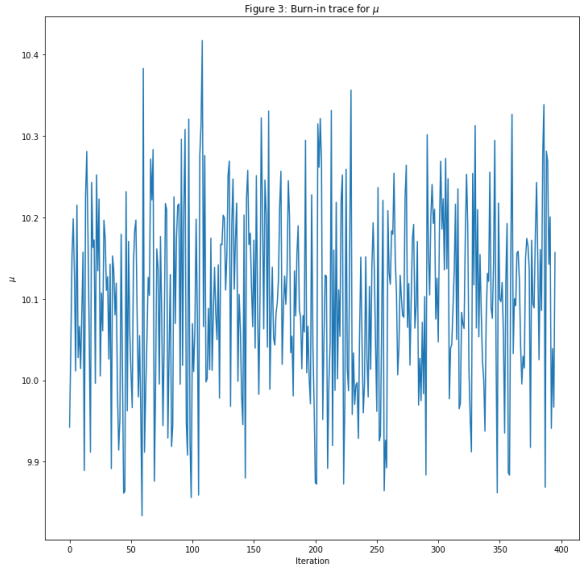
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Runti
meWarning: invalid value encountered in log
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:57: Runti
meWarning: divide by zero encountered in log
```

```
10.099499147803549 3.086705659568563
```

Out[238]:

```
<matplotlib.legend.Legend at 0x242080100c8>
```





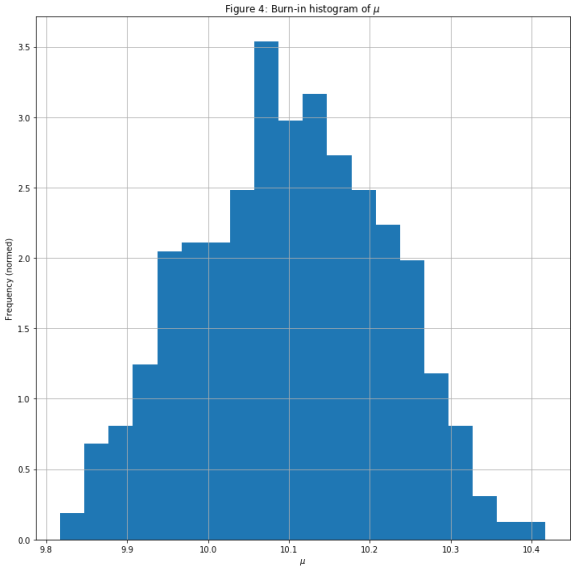


Figure 5: Burn-in trace for σ

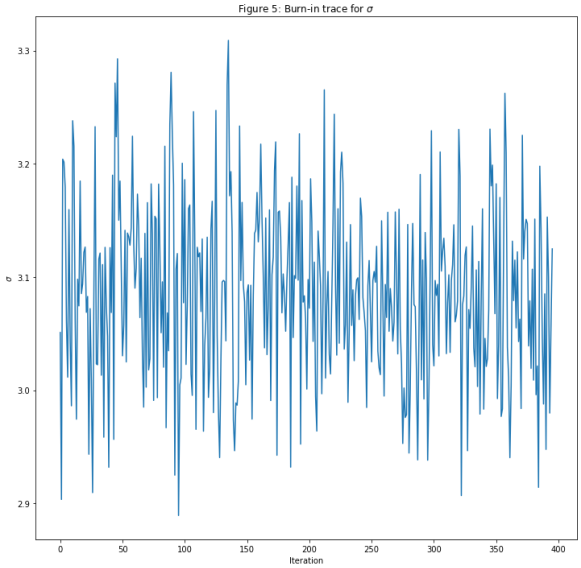


Figure 6: Burn-in histogram of σ

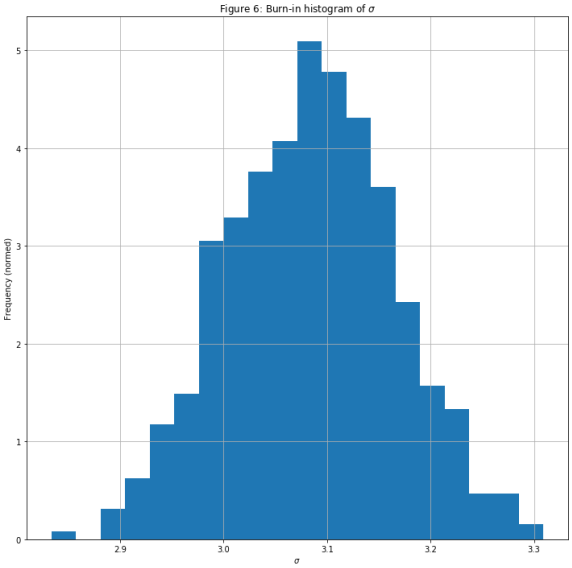


Figure 7: 2D histogram showing the joint distribution of μ and σ

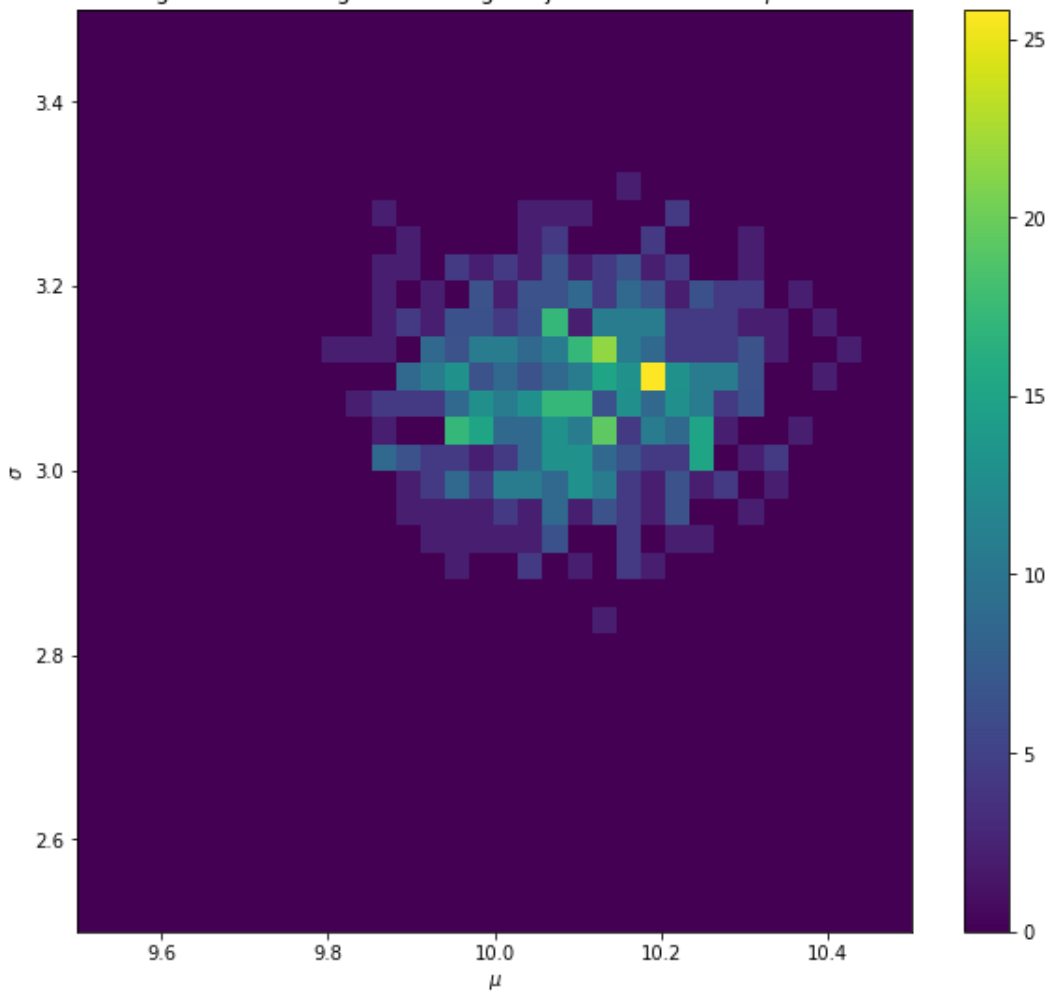
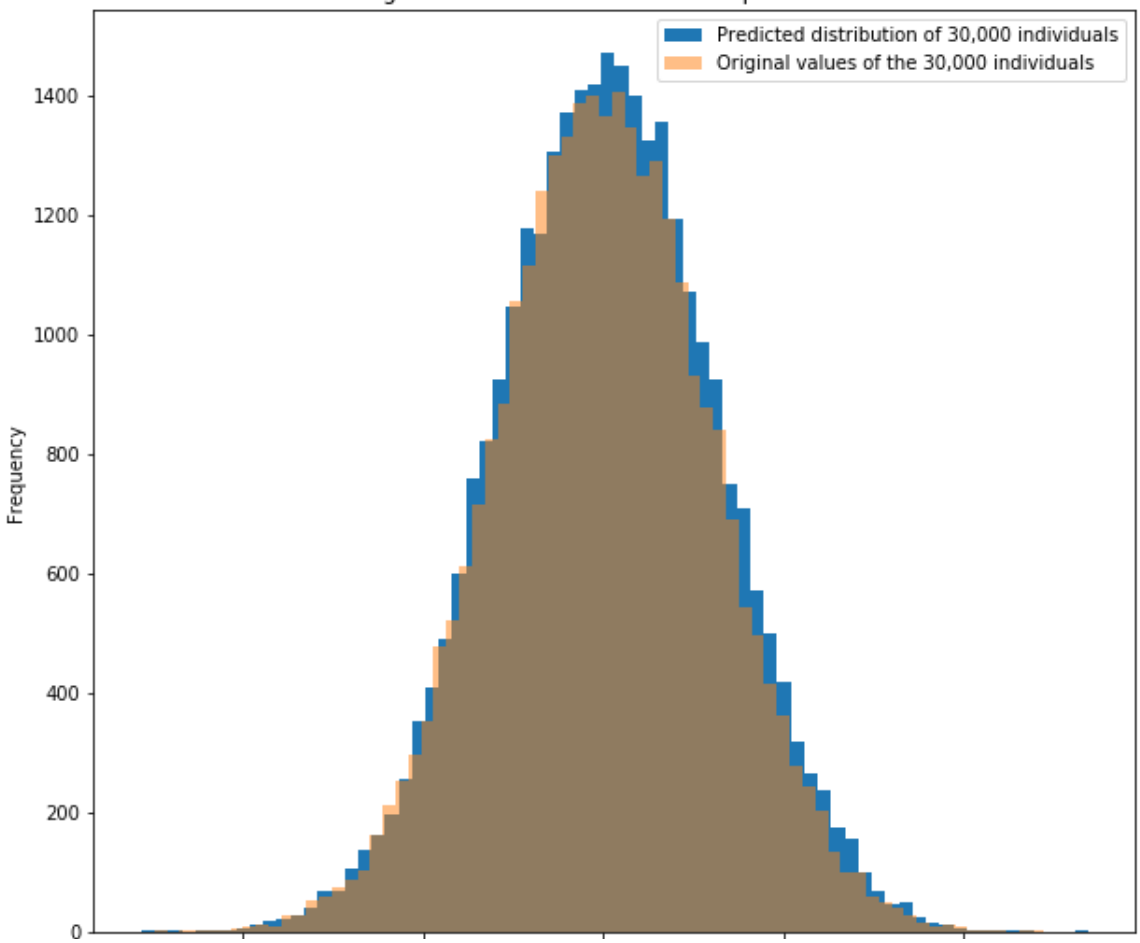


Figure 8: Posterior distribution of predicitons



Results

So starting from intial μ = 50 and σ = 5, the algorithm quickly converges to expected μ appx 10 and σ appx 3. Shown in 2d space, figure2.

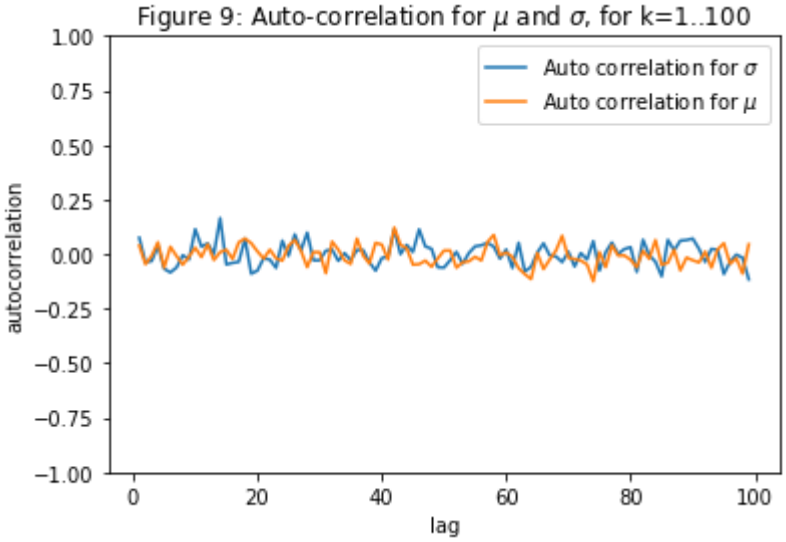
We have chosen to drop the first 25% (μ,σ) to be the burn-in. Figure 3-6 display trace/histogram plots. The converged space is exaimed in figure 7, where joint distibution frequency(histogram) of μ and σ.

Averaging the accepted 75% samples paramters we generate 30,000 random values (Predictions) and compared against the original observed (Figure 8).

```
10.099499147803549 3.086705659568563
```

```
Out[221]:
```

```
[(-1, 1), Text(0, 0.5, 'autocorrelation'), Text(0.5, 0, 'lag')]
```



Evaluating proposal distribution

The autocorrelation of accepted samples for μ and σ from 1 to 100 is low. A lag of k=0 means that we are measuring the correlation of a sample with itself, so we expect it to be equal to 1.This is not shown here, with above figure 9 starting at apprx 0.1 - further investigation required. However the higher k goes, the lower that correlation ought to be, which is what is depicted.

Future works such as Hamiltonian MC are able to incoprate a reducing correlation between successive sample states, aiming to achieve stationary distribution quicker.

We will now explore the MALA implementation, that hopes to drive th random walk towards regions of high probability in the manner of a gradient flow [3].

Metropolis-adjusted Langevin algorithm (MALA)

A more involved version of the random walk used in Metropolis algorithm is the inclusion of drift. This can be generated from the following proposal

$$X^* = x^{(t)} + d^{(t)} + \sigma\epsilon^{(t)},$$

where

$$d^{(t)} = \left(\frac{\sigma^2}{2}\right)\frac{\partial log f(x)}{\partial x}\Big|_{x=x^t}$$

Step 1. Using previous Metropolis algorithm

Step 2. Update new states to include Langevin dynamics, using gradient evaluations of the gradient of the target probability density function.

Out[167]:

```
(49993, 1)
```

Figure 2: MCMC sampling for μ with MALA. First 100 samples are shown.

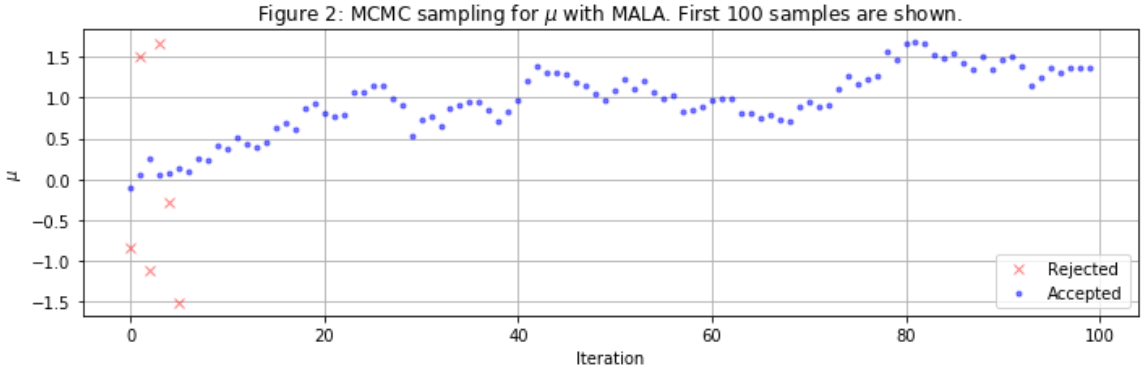


Figure 3: MCMC sampling for μ with MALA. Last 100 samples are shown.

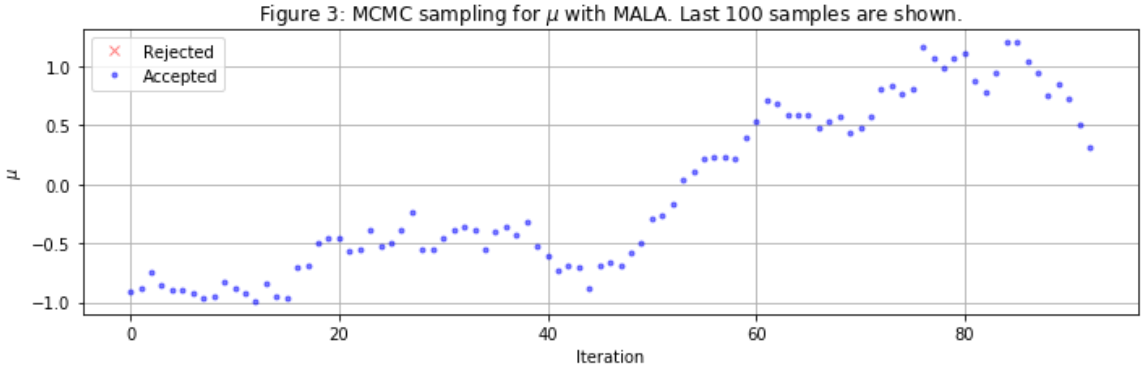
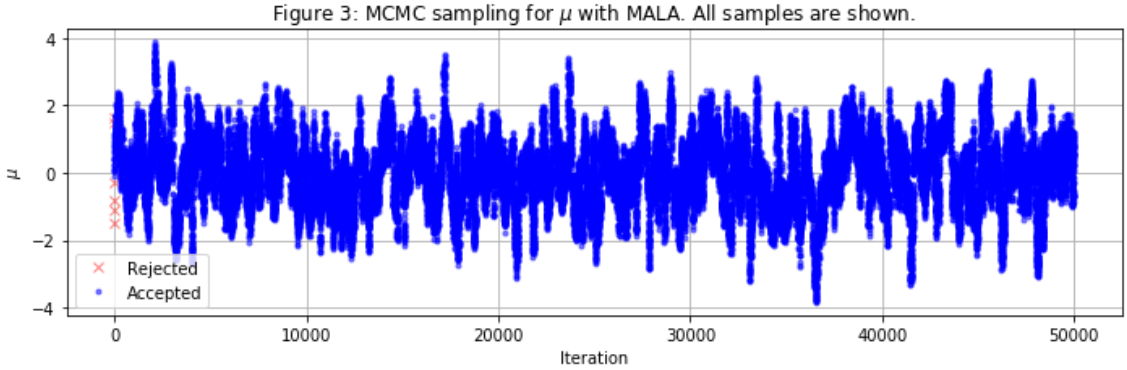
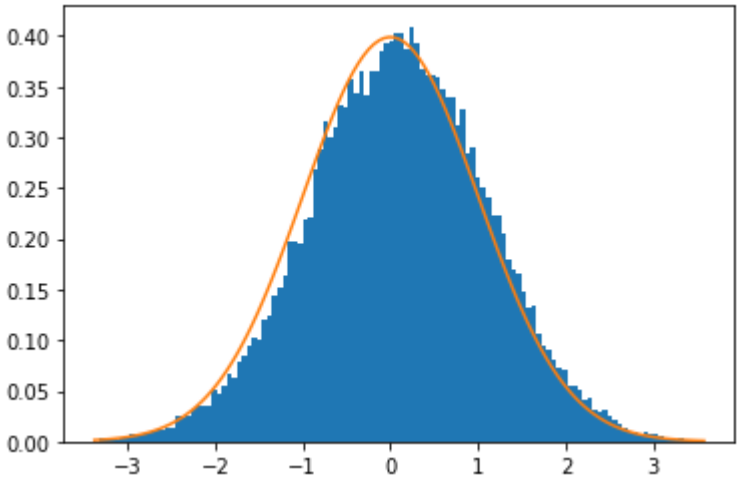


Figure 3: MCMC sampling for μ with MALA. All samples are shown.



Out[151]:

```
[<matplotlib.lines.Line2D at 0x2421669cb88>]
```



References:

[1] https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo (https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo)

[2] https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm (https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm)

[3] https://en.wikipedia.org/wiki/Metropolis-adjusted_Langevin_algorithm (https://en.wikipedia.org/wiki/Metropolis-adjusted_Langevin_algorithm)

[4] Givens,G 2013, Computational Statistics, Second Edition