Assignment 5

In this assignment, we'll finally work with some nonconjugate models. I will also introduce you to reparameterization techniques.

Instructions

Please complete this Jupyter notebook and **don't** convert it to a `.py` file. Upload this notebook, along with any `.stan` files and any data sets as a `zip` file to Gradescope.

Your work will be manually graded by our TA. There is no autograder for this assignment. For free response questions, feel free to add a markdown cell and type in there. Try to keep the preexisting structure as much as possible, and to be organized and label which cells correspond with which questions.

Problem 1: Poisson Data

In the last assignment, we modeled a vector of counts $y = (y_1, \ldots, y_n)$ using a multinomial distribution.

Unlike last time, all of these counts will now assumed to be independent. Further, we can't reasonably put a bound on what each count could be. So, in this problem, we'll use a **Poisson likelihood**:

$$L(y \mid \theta) = \prod_{i=1}^{n} L(y_i \mid \theta) \propto \prod_{i=1}^{n} e^{-\theta}\theta^{y_i} = e^{-n\theta}\theta^{\sum_i y_i}$$

With this likelihood, $\theta > 0$ is interpreted as a rate or average.

The data can be found in `Road_Casualties_in_Great_Britain_1969___84_434_19.csv` Use the `DriversKilled` column only.

In [2]:
```
import pandas as pd
import numpy as np
dk = np.array(pd.read_csv('Road_Casualties_in_Great_Britain_1969___84_434_19
```

1.

Name a conjugate prior for this likelihood! Write your single-word answer in Gradescope.

Gamma

2.

Suppose that the previous answer does not suite your needs, and that you want to use a lognormal prior! Pick a specific prior distribution (i.e. specify the hyperparameters), and describe a rationale as to why you chose them.

In [3]:
```
dk.mean()
```

Out[3]:
```
122.80208333333333
```

I would use LogNormal(122, 9999) because the mean of the data we have is about 122 and I'm not really sure about it, so I chose an arbitrarily high standard deviation.

3.

Use `stan` to estimate your model for the "DriversKilled" column. Please be sure to

- report an $\hat{R}$ diagnostic and comment on whether it is close to 1
- display trace plots of your samples obtained and comment on whether they look like "fuzzy caterpillars."

Then, after checking diagnostics...

- display a histogram of the posterior for θ
- report estimates of the mean, 5th and 95th percentiles of this posterior
- comment on whether your posterior mean is close to the frequentist estimator of θ (which is the sample mean of your data)

In [5]:
```
import os
from cmdstanpy import CmdStanModel

# bulid model
model_code = os.path.join('.', 'poisson_log_norm.stan')
model = CmdStanModel(stan_file=model_code)
```

```
13:40:06 - cmdstanpy - INFO - compiling stan file /bml24/05/poisson_log_nor
m.stan to exe file /bml24/05/poisson_log_norm
13:40:22 - cmdstanpy - INFO - compiled model executable: /bml24/05/poisson_lo
g_norm
```

In [6]:
```
model
```

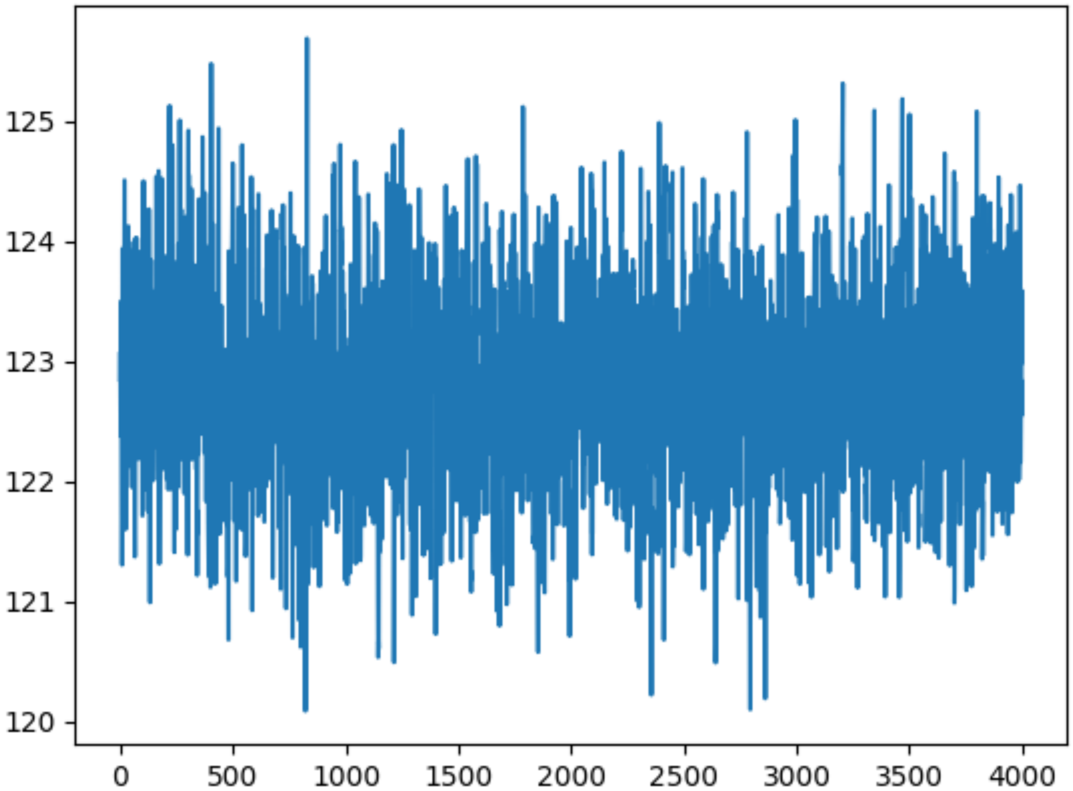
Out[6]:
```
CmdStanModel: name=poisson_log_norm
         stan_file=/bml24/05/poisson_log_norm.stan
         exe_file=/bml24/05/poisson_log_norm
         compiler_options=stanc_options={}, cpp_options={}
```

In [7]:
```
# sample from model
num_samps = 192
normal_data = {'N' : num_samps, 'y': dk}
fit = model.sample(normal_data)
```

```
13:40:26 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

13:40:26 - cmdstanpy - INFO - CmdStan done processing.
```

In [8]:
```
# view diagnostics
fit.draws_pd()['theta'].plot()
details = fit.summary()
details.loc['theta','R_hat']
```

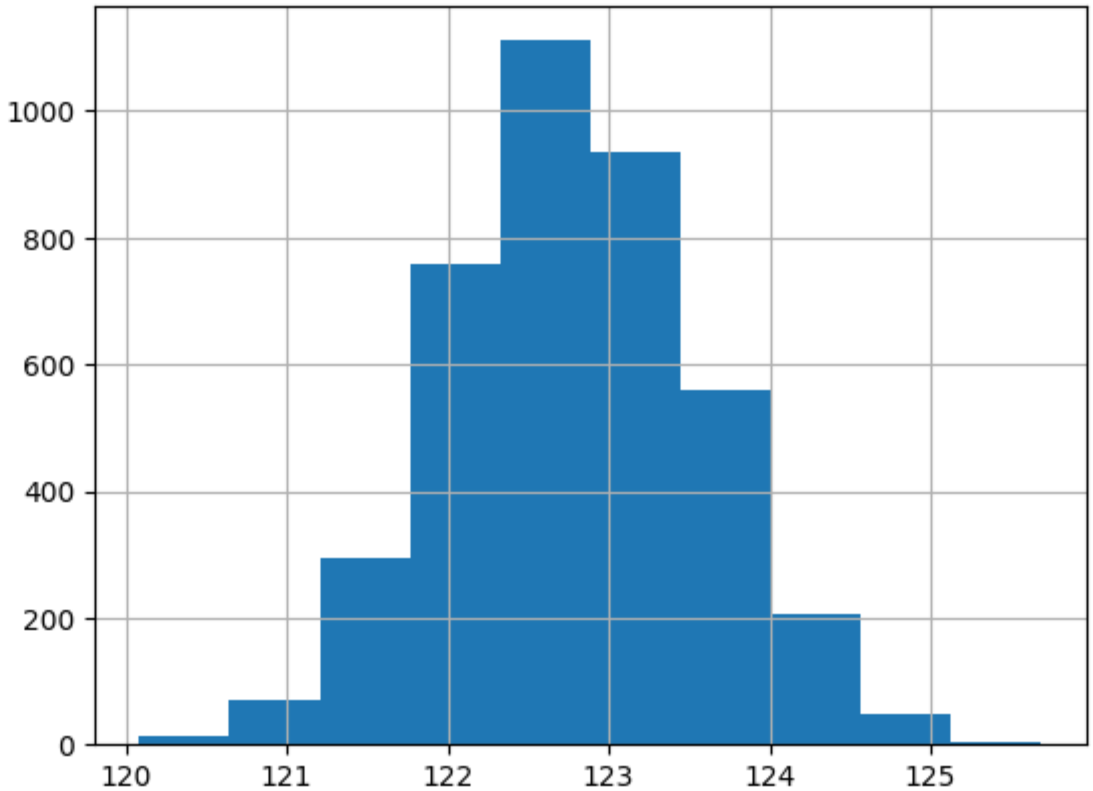
Out[8]: 1.00287



The plot looks like a fuzzy caterpillar, and the R_hat value is very close to 1, which means the MCMC worked well.

In [9]:
```
fit.draws_pd()['theta'].hist()
details.loc['theta',['Mean', '5%', '95%']]
```

Out[9]:
```
Mean     122.790
5%       121.550
95%      124.115
Name: theta, dtype: float64
```



The mean for theta is very close to the frequentist estimate (sample mean).

4.

Now use `stan` to estimate a slightly reparameterized model. Suppose you want to use a normal prior on an unconstrained parameter. Notice that if something is positive, then the (natural) log of it is unconstrained. Similarly, if something is unconstrained, the exponential of it is positive.

Therefore, use the following model

$$\theta \sim \text{Normal}(a, b)$$

and

$$y_i \mid \theta \sim \text{Poisson}(e^{\theta})$$

Use `stan` to estimate your model for the "DriversKilled" column. Please be sure to

- report an $\hat{R}$ diagnostic and comment on whether it is close to 1
- display trace plots of your samples obtained and comment on whether they look like "fuzzy caterpillars."

Then, after checking diagnostics...

- display a histogram of the posterior for θ
- display a histogram of the posterior for the transformed parameter, too.

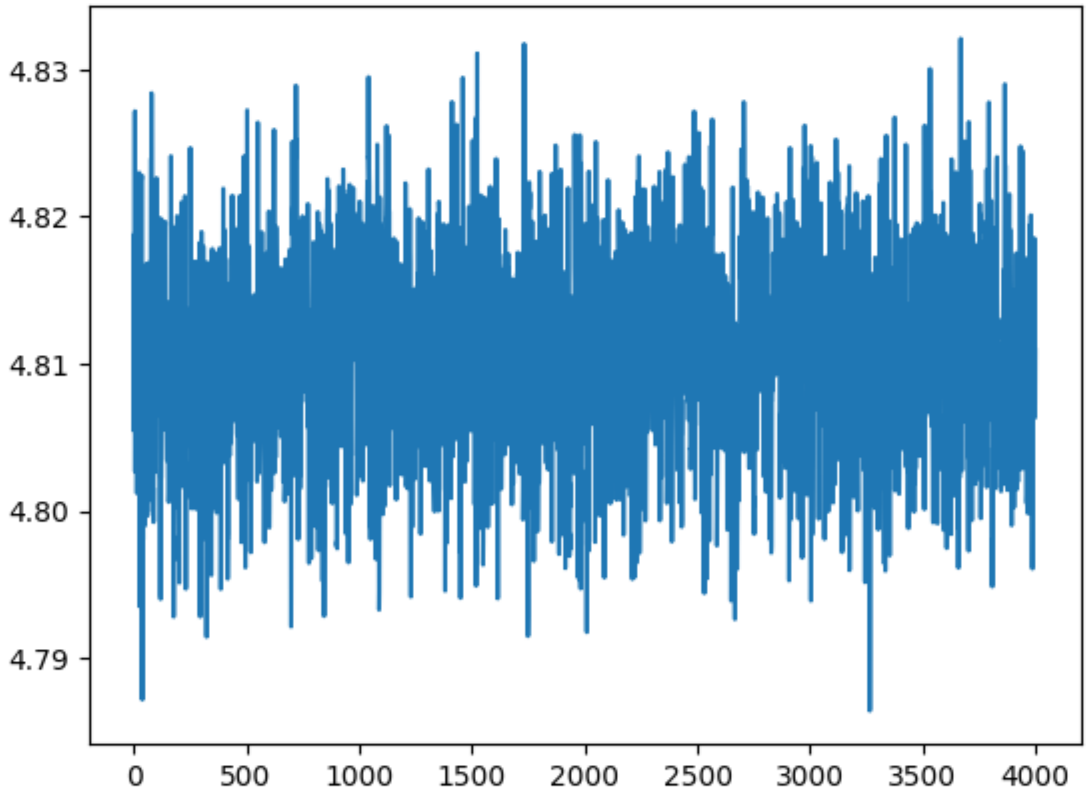
- report estimates of the mean, 5th and 95th percentiles of the posterior of the unconstrained θ
- comment on whether your posterior mean is close to the frequentist estimator (which is the sample mean of your data)

In [10]:

```
# build model
model_code = os.path.join('.', 'poisson_norm.stan')
model = CmdStanModel(stan_file=model_code)

# run sims
num_samps = 192
normal_data = {'N' : num_samps, 'y': dk}
fit = model.sample(normal_data)

# view diagnostics
fit.draws_pd()['theta'].plot()
details = fit.summary()
details.loc['theta','R_hat']
```

```
13:40:39 - cmdstanpy - INFO - compiling stan file /bml24/05/poisson_norm.stan
to exe file /bml24/05/poisson_norm
13:40:55 - cmdstanpy - INFO - compiled model executable: /bml24/05/poisson_no
rm
13:40:55 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

13:40:55 - cmdstanpy - INFO - CmdStan done processing.
```

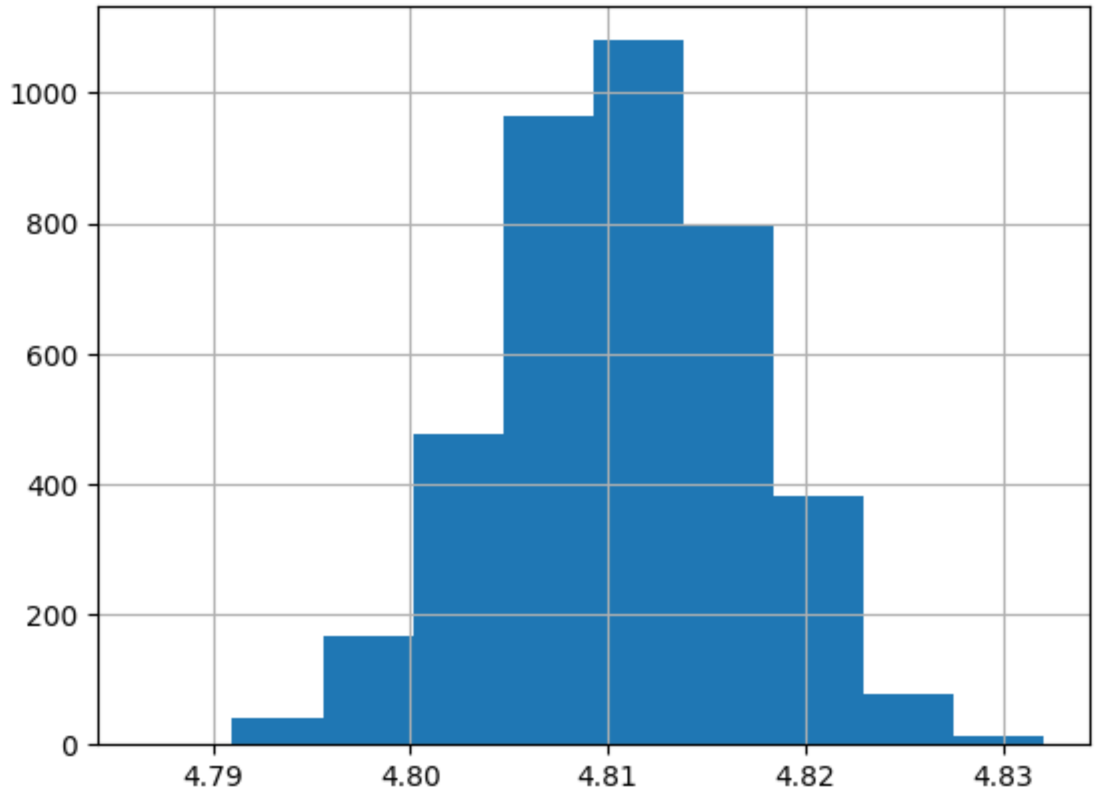
Out[10]: 1.00371



Fuzzy caterpillar - YES! R_hat close to 1 - YES!

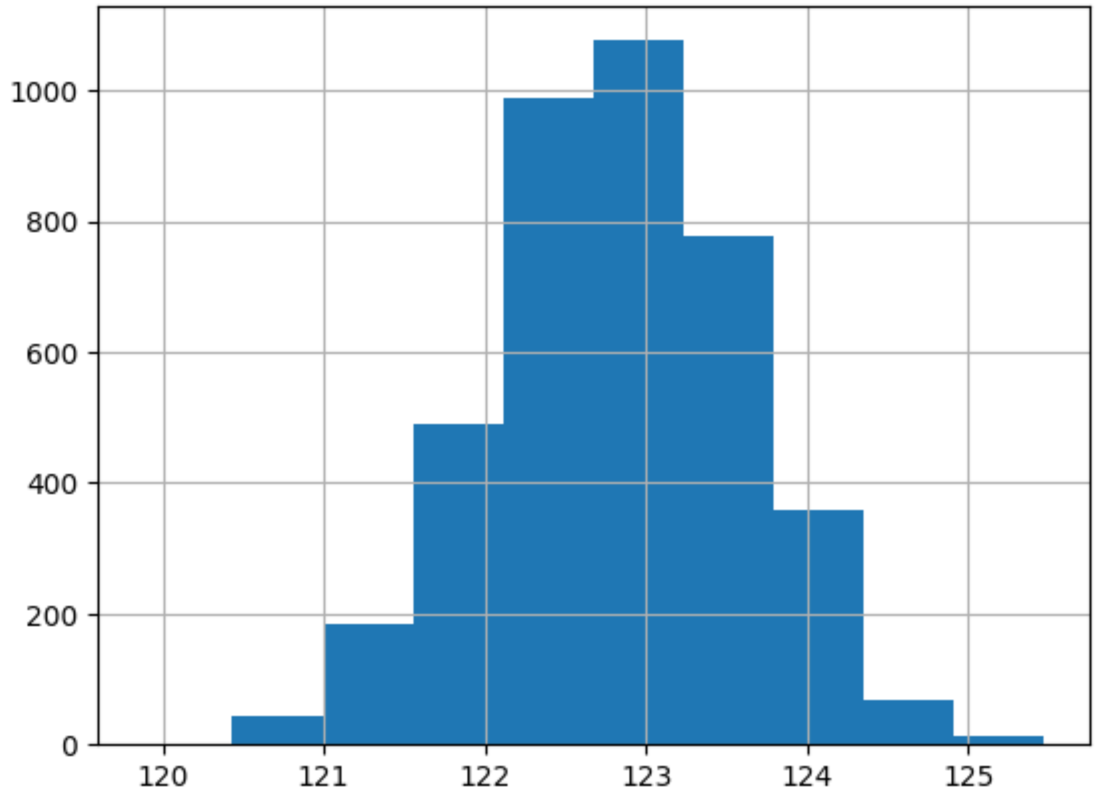
In [11]:
```
# info on unconstrained theta
fit.draws_pd()['theta'].hist()
details.loc['theta',['Mean', '5%', '95%']]
```

Out[11]:
```
Mean     4.81069
5%       4.79994
95%      4.82087
Name: theta, dtype: float64
```



In [12]:
```
# info on the actual parameter used to simulate data
fit.draws_pd()['exp_theta'].hist()
details.loc['exp_theta',['Mean', '5%', '95%']]
```

Out[12]:
```
Mean     122.819
5%       121.503
95%      124.073
Name: exp_theta, dtype: float64
```



In [13]:
```
np.exp(4.81051)
```

Out[13]: 122.79422660615958

I can see that the unconstrained theta averages around 4.81, which makes exp_theta about 122 (very close to the frequentist estimate/sample mean).

Problem 2: Binomial Data (again!)

Suppose that you have $m > 1$ count data points $y_1, \ldots, y_m$, each having a $\text{Binomial}(n, \eta)$ distribution. Assume further that they're all independent.

Here n is the maximum for each data point. m is the number of data points.

In our second homework we used the beta prior for the parameter that was bounded between 0 and 1.

Now, you must use a normal prior for an unconstrained parameter.

If $0 < \eta < 1$, then the *logit* transformation is a way to make $-\infty < \theta < \infty$ (unconstrained). Alternatively, if you have η that's unconstrained, then the `inv_logit` will squash the value to lie between 0 and 1.

`stan` conveniently has a `logit()` and an `inv_logit()` function already made for you.

Use `stan` to estimate your model on any fictitious data you would like. Be sure to

- report an $\hat{R}$ diagnostic and comment on whether it is close to 1
- display trace plots of your samples obtained and comment on whether they look like "fuzzy caterpillars."

Then, after checking diagnostics...

- display a histogram of the posterior for θ
- display a histogram of the posterior for the transformed parameter, too.
- report estimates of the mean, 5th and 95th percentiles of the posterior of the unconstrained θ
- comment on whether your posterior mean is close to the frequentist estimator (which is the sample mean of your data, again).

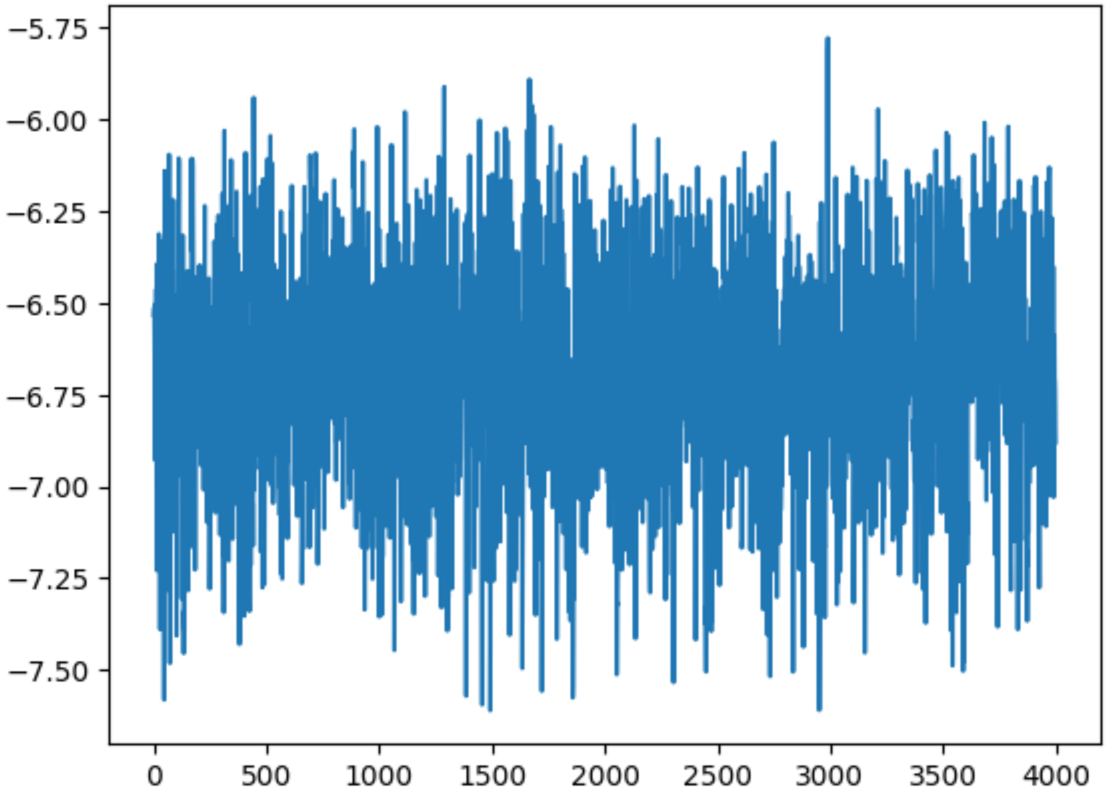
In [14]:
```
some_fake_data = np.random.choice([0, 1], 100, p = [.9, .1])
some_fake_data
```

Out[14]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

In [15]:
```
# build model
model_code = os.path.join('.', 'binom_normal.stan')
model = CmdStanModel(stan_file=model_code)

# run sims
num_samps = 100
mu_prior = 122
sigma_prior = 9999
binom_data = {'N' : num_samps,
              'y': some_fake_data,
              'mu_prior' : mu_prior,
              'sigma_prior' : sigma_prior}
fit = model.sample(binom_data)

# view diagnostics
fit.draws_pd()['theta'].plot()
details = fit.summary()
details.loc['theta','R_hat']
```

```
13:41:09 - cmdstanpy - INFO - compiling stan file /bml24/05/binom_normal.stan
to exe file /bml24/05/binom_normal
13:41:25 - cmdstanpy - INFO - compiled model executable: /bml24/05/binom_norm
al
13:41:25 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status


13:41:26 - cmdstanpy - INFO - CmdStan done processing.
```

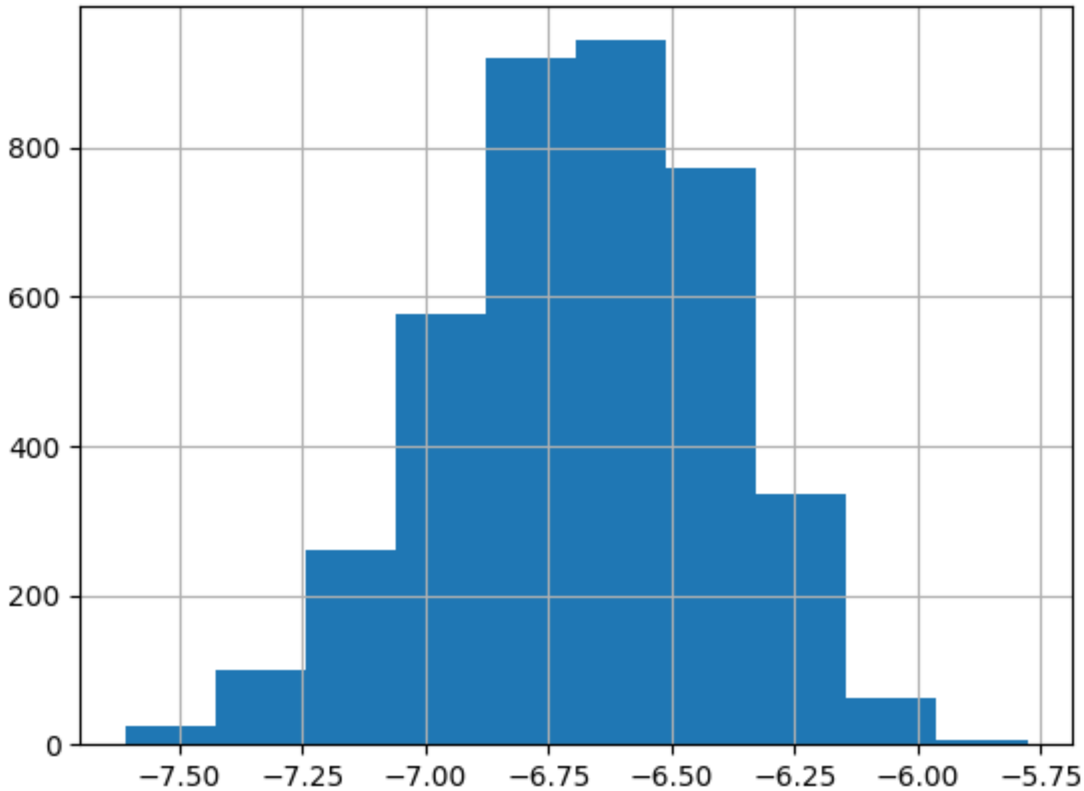
Out[15]: 1.00105



Fuzzy caterpillar - YES! R_hat close to 1 - YES!

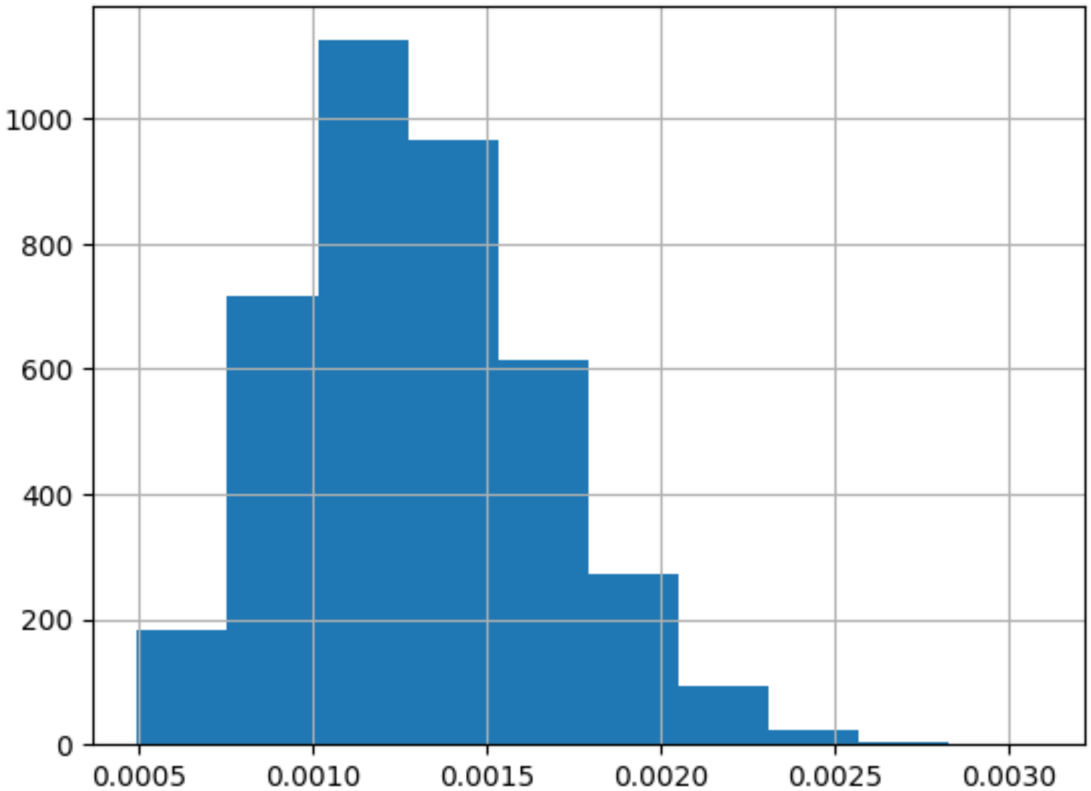
In [16]:
```
# info on unconstrained theta
fit.draws_pd()['theta'].hist()
details.loc['theta',['Mean', '5%', '95%']]
```

Out[16]:
```
Mean    -6.68131
5%      -7.17437
95%     -6.23745
Name: theta, dtype: float64
```



In [17]:
```
# info on the actual parameter used to simulate data
fit.draws_pd()['nu'].hist()
details.loc['nu',['Mean', '5%', '95%']]
```

Out[17]:
```
Mean     0.001303
5%       0.000765
95%      0.001951
Name: nu, dtype: float64
```



The posterior mean of the actual parameter nu ended up being really close to 0.1, which was the true parameter when simulating the data. Cool!