STAT 203 SOFTWARE TUTORIAL

PYTHON IN BAYESIAN ANALYSIS

YING LIU

Some facts about Python

- An open source programming language
- Have many IDE to choose from (for R? Rstudio!)
- A powerful language; it can do many different things
- Popular among developers *Click!*

Diasorin example

```
import numpy as np
import matplotlib.pyplot as plt

# Data
low = np.array([91, 46, 95, 60, 33, 410, 105, 43, 189, 1097, 54, 178, 114, 137, 233, 101, 25, 70,
357])
log_low = np.log(low)
n_low = len(log_low)
low_bar = np.mean(log_low)
low_s = np.var(log_low)**0.5

# Initialize random number generator
np.random.seed(123)

# MCMC size
N=10000

# Initialize mu and tau
mu_l_all = np.zeros(N + 1)
tau_l_all = np.zeros(N + 1)
mu_l_all[0] = 5
tau_l_all[0] = 1

# Priors
a_low = 4.87
b_low = 0.00288
c_low = 1.0376
d_low = 0.001
```

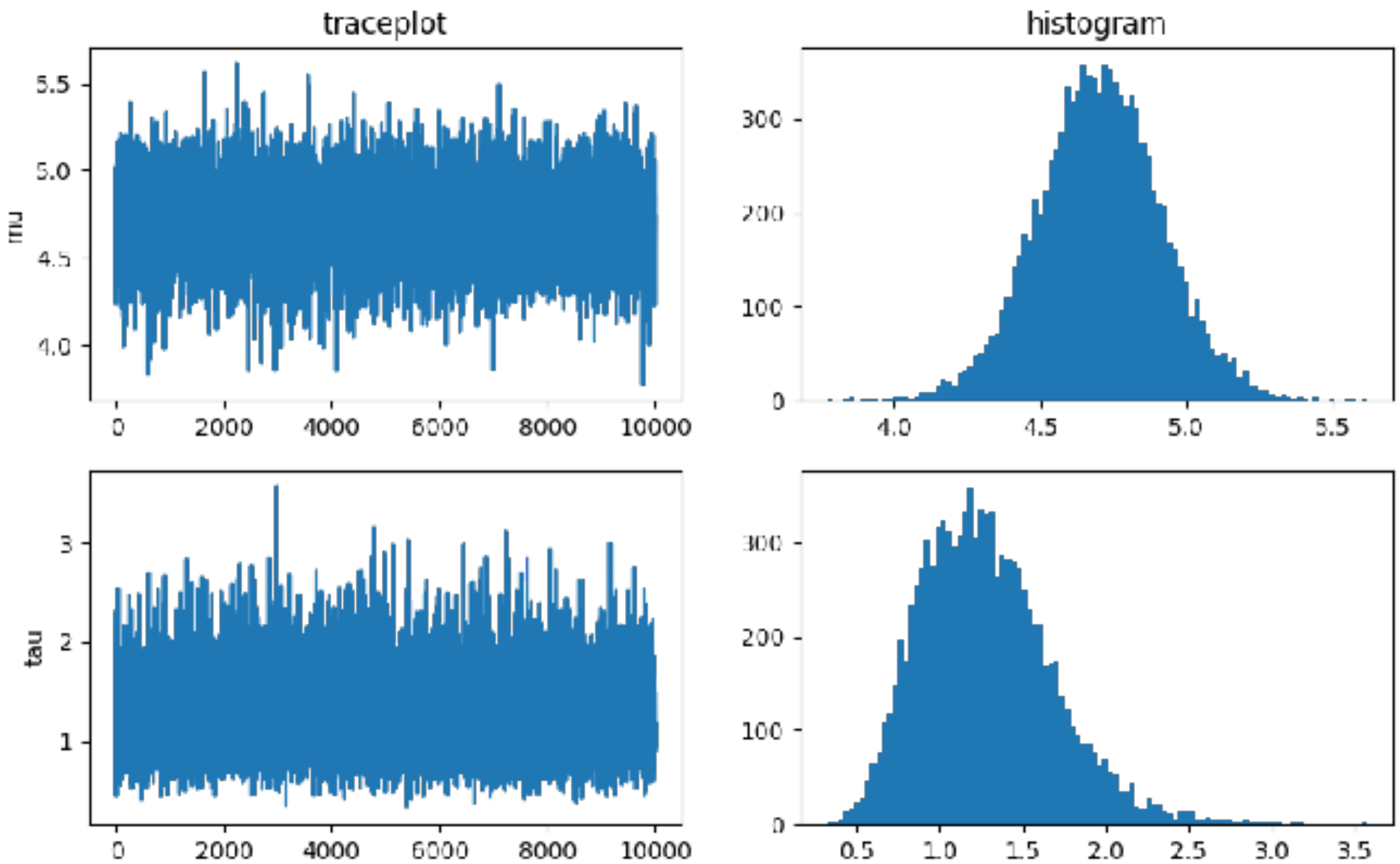
Diasorin example

```
# Run sampler
for i in range(1, N+1):
    mu_hat_l = n_low * tau_l_all[i-1] / (n_low * tau_l_all[i-1] + b_low) * low_bar + b_low /
(n_low * tau_l_all[i-1] + b_low) * a_low
    sd_norm_l = 1 / np.sqrt(n_low * tau_l_all[i-1] + b_low)
    mu_l_all[i] = np.random.normal(mu_hat_l, sd_norm_l)

    a_gamma_l = c_low + n_low / 2
    b_gamma_l = d_low + ((n_low - 1) * low_s + (low_bar - mu_l_all[i])**2) / 2
    tau_l_all[i] = np.random.gamma(a_gamma_l, 1 / b_gamma_l)

# plot
ax0.plot(mu_l_all)
ax0.set_title("traceplot")
ax0.set(ylabel="mu")
ax1.hist(mu_l_all, bins=100)
ax1.set_title("histogram")
ax2.plot(tau_l_all)
ax2.set(ylabel="tau")
ax3.hist(tau_l_all, bins=100)

plt.show()
```



Summary Plots

*the summary statistics agree with R results

PyMC3

- PyMC3 is a Python package for Bayesian statistical modeling and Probabilistic Machine Learning which focuses on advanced Markov chain Monte Carlo and variational fitting algorithms

- <u>Sampling algorithms</u>: Metropolis, No U-Turn Sampler, Slice, Hamiltonian Monte Carlo.

- Variational inference: ADVI for fast approximate posterior estimation as well as mini-batch ADVI for large data sets.

Diasorin example - PyMC3

```
import pymc3 as pm

diasorin_model = pm.Model()

with diasorin_model:

    # Priors for unknown model parameters
    mu_l = pm.Normal('mu_l', mu=4.87, sd=np.sqrt(0.00288))
    mu_n = pm.Normal('mu_n', mu=5.39, sd=np.sqrt(0.00280))
    tau_l = pm.Gamma('tau_l', alpha=1.0376, beta=0.001)
    tau_n = pm.Gamma('tau_n', alpha=1.04653, beta=0.001)

    # Likelihood (sampling distribution) of observations
    low_obs = pm.Normal('low_obs', mu=mu_l, sd=tau_l**-0.5, observed=log_low)
    nor_obs = pm.Normal('nor_obs', mu=mu_n, sd=tau_n**-0.5, observed=log_nor)

with diasorin_model:

    # instantiate sampler
    step = pm.Metropolis()

    # draw 5000 posterior samples
    trace = pm.sample(5000, step=step, chains=1)

print(pm.summary(trace))
pm.traceplot(trace)
pm.autocorrplot(trace)
plt.show()
```

Diasorin example - PyMC3

Sequential sampling (1 chains in 1 job)

CompoundStep

>Metropolis: [tau_n_log__]

>Metropolis: [tau_l_log__]

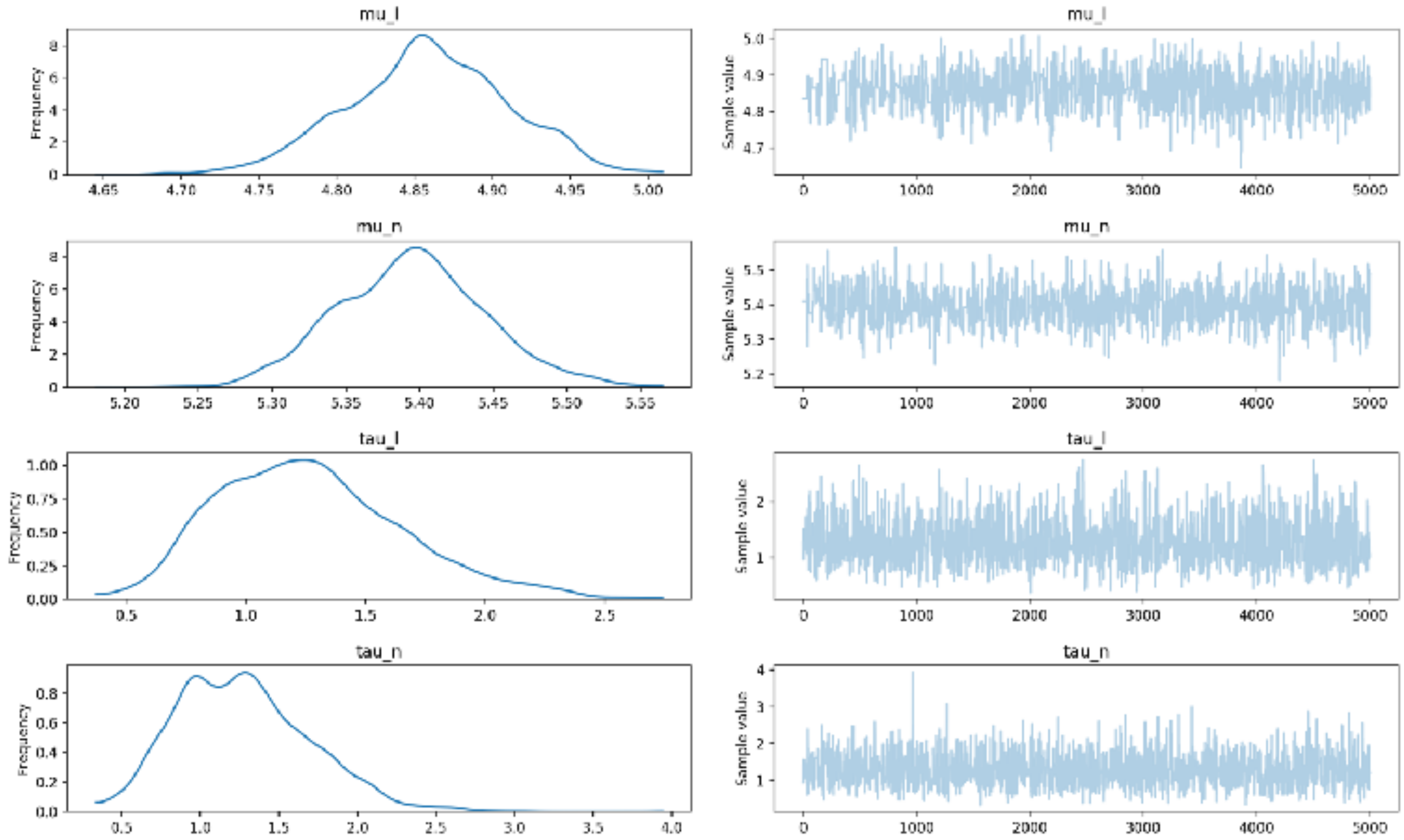
>Metropolis: [mu_n]

>Metropolis: [mu_l]

100%|██████████| 5500/5500 [00:03<00:00, 1611.02it/s]

Only one chain was sampled, this makes it impossible to run some convergence checks

	mean	sd	mc_error	hpd_2.5	hpd_97.5
mu_l	4.857972	0.051716	0.001683	4.758402	4.953345
mu_n	5.394712	0.049998	0.001543	5.296662	5.493079
tau_l	1.274592	0.386698	0.010289	0.579774	2.058619
tau_n	1.270912	0.420986	0.010174	0.593396	2.142391

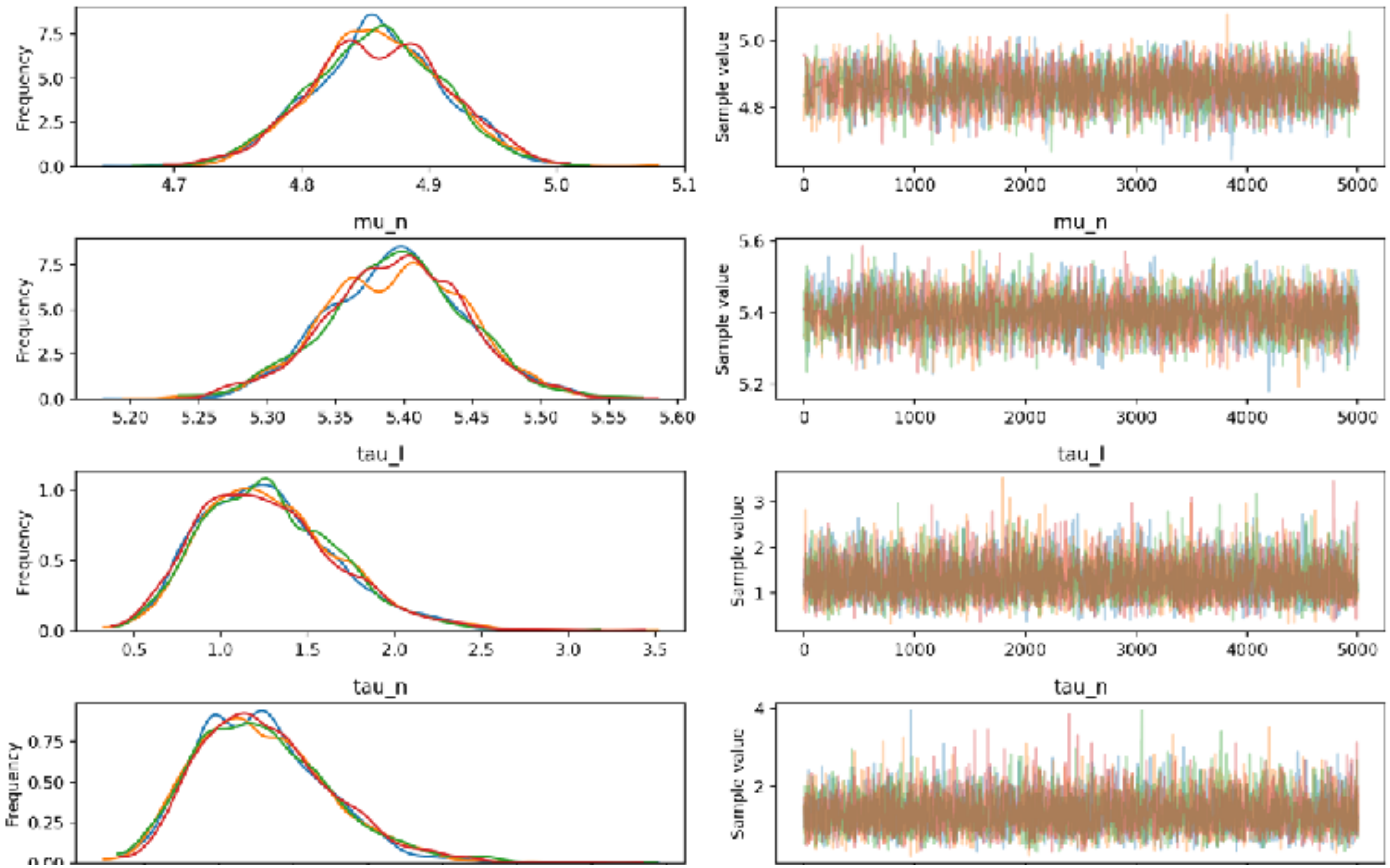


Summary plots

Diasorin example - PyMC3

- We can also run 4 chains simultaneously and get effective sample size and R_hat

	mean	sd	mc_error	hpd_2.5	hpd_97.5	n_eff	Rhat
mu_l	4.858695	0.051844	0.000897	4.757479	4.957881	3100.0	1.000299
mu_n	5.394198	0.050610	0.000769	5.290099	5.489188	3815.0	0.999937
tau_l	1.283721	0.395592	0.006016	0.571324	2.058619	4535.0	1.000327
tau_n	1.296577	0.446363	0.006509	0.477823	2.142391	4016.0	1.000944



Summary plots

Oring example - PyMC3

```
import matplotlib.pyplot as plt
import pandas as pd
import pymc3 as pm

data = pd.read_table("http://www.ics.uci.edu/~wjohnson/BIDA/Ch8/OringData.txt")
data_std = data
data_std['Temperature'] = data['Temperature'] - np.mean(data['Temperature'])

with pm.Model() as oring_model:
    pm.glm.GLM.from_formula('Failure ~ Temperature', data_std,
family=pm.glm.families.Binomial())
    step = pm.Metropolis()
    trace_oring_model = pm.sample(2000, step=step, chains=1, tune=1000)

print(data_std)

print(pm.summary(trace_oring_model))
pm.traceplot(trace_oring_model)
pm.autocorrplot(trace_oring_model)

plt.show()
```

Modified Gibbs code

```
# MCMC size
N = 10000000

# Initialize mu and tau
mu_l_all = [5] + [0] * N
tau_l_all = [1] + [0] * N

# Data
n_low = len(log_low)
low_bar = stats.mean(log_low)
low_s = stats.stdev(log_low)

a_gamma_l = c_low + n_low / 2

m = mu_l_all[0]
t = tau_l_all[0]

# Run sampler
for i in range(1, N+1):
    mu_hat_l = n_low * tau_l_all[i-1] / (n_low * tau_l_all[i-1] + b_low) * low_bar + b_low /
(n_low * tau_l_all[i-1] + b_low) * a_low
    sd_norm_l = (n_low * tau_l_all[i-1] + b_low)**(-0.5)
    mu_l_all[i] = random.normalvariate(mu_hat_l, sd_norm_l)

    b_gamma_l = 2 / (d_low+(n_low - 1) * low_s + (low_bar - mu_l_all[i])**2)
    tau_l_all[i] = random.gammavariate(a_gamma_l, b_gamma_l)
```

Running time

Consider three cases:

- Simple Iteration (print “hi” for 1000000 time)
- Diasorin (Gibbs) with 10000000 draws
- Diasorin (Metropolis) with 100000 draws
- Diasorin (Metropolis) with 1000000 Simulated data and 5000 draws

Running Time	R	Python
Simple Iteration	27.44s	3.88s
Diasorin (Gibbs)	42s	27s/72s
Diasorin (Metropolis)	29s	56.2s
Diasorin (Metropolis, simulated data)	245s	76.9s

Python v.s. R

Just look at speed is not enough.

Use what is best for what you want to do, not whatever is "fastest"

- Packages?

R has huge number of packages while Python is still growing

- Users?

In general, Python has more supporters of developers and programmers but statisticians and researchers may choose R.

- Large data set?

- Complicated statistics model?

- Visualization?

… and more

Python v.s. R

Consider the situations:

1. Very simple but relatively routine process (eg. Generate random numbers)

2. Small scale tasks (eg. Compare samplers)

3. Pure data analysis

4. Not just data analysis: pull the data from website and clean it, do the analysis and create a website or embed it into larger programs

Resources

1. Getting started!
2. IDEs for Python
3. PyMC3 documentation
4. AM207 course material