Bayes Method - Practice

Tony RuiKang OuYang

June 2022

Abstract

This is a report for practice in the course - Bayes Method. In this report, we're going to use the methods taught in the course to tackle with the problem of approximating the social rank, which could be solved using the Plackett-Luce model[**?**]. Likewise, we're going to simulate samples based on the Plackett-Luce model via the MCMC based methods, as well as inferring the probability of the 7^{th} player rank 1^{st} in the 68^{th} game. We've used the normal prior and uniform prior respectively in our experiments, and the MCMC algorithms under both of these two priors would converge quickly. Furthermore, we've implemented the model averaging algorithm, which is based on the Reversible-Jump MCMC. And all the codes are implemented using python3.9.

Keywords— B ayesian Inference, MCMC, Reversible-Jump MCMC, model averaging

All of my codes could be found in this GitHub link

1 Model

To solve this problem, we use the Plackett-Luce model to parameterise. Then, we simulate a sequence of samples using MCMC algorithm, as well as performing the model averaging method to find better models.

1.1 The Plackett-Luce model

Following the problem settings, we have two kinds of model parameterisations:

$$Pr(O = o|\lambda) = \prod_{i=1}^{m} \frac{exp(\lambda_{o_i})}{\sum_{j=1}^{m} exp(\lambda_{o_j})} \tag{1}$$

$$Pr(O = o|\lambda, \beta) = \prod_{i=1}^{m} \frac{exp(\lambda_{o_i} + \beta_{e_i})}{\sum_{j=1}^{m} exp(\lambda_{o_j} + \beta_{e_i})} \tag{2}$$

Let $\theta = (\lambda, \beta)$, where $\lambda = (\lambda_1, ..., \lambda_m)$ and $\beta = (\beta_1, ..., \beta_m)$. Then we could use the following notions:

$$Pr(O = o|\theta, M = 1) = \prod_{i=1}^{m} \frac{exp(\lambda_{o_i})}{\sum_{j=1}^{m} exp(\lambda_{o_j})} \tag{3}$$

$$Pr(O = o|\theta, M = 2) = \prod_{i=1}^{m} \frac{exp(\lambda_{o_i} + \beta_{e_i})}{\sum_{j=1}^{m} exp(\lambda_{o_j} + \beta_{e_i})} \tag{4}$$

1.2 Metropolis Hasting Algorithm

To simulate samples via MCMC, we first assign prior for the parameters. Let $\theta = (\lambda, \beta)$, where $\lambda = (\lambda_1, ..., \lambda_m)$ and $\beta = (\beta_1, ..., \beta_m)$. Denote the prior of θ as $\pi_\theta(\cdot)$, and simply assume that the parameters are independent and have the same parameter space $\Omega_m = \mathbf{R}$, i.e. $\pi_\theta(d\theta) = \prod_{i=1}^{2m} \pi_{\theta_i}(d\theta_i)$.

Thus, the posterior is:

$$\pi(\theta|o) \propto Pr(O = o|\lambda, \beta) \prod_{i=1}^{2m} \pi_{\theta_i}(\theta_i)$$

Using the independent proposal $\pi_i(\cdot)$ for each parameter, the MH algorithm is described as following:

Algorithm 1 Metropolis Hasting Algorithm

Require: X_t

for i=1,2...,2m **do**

Simulate $\theta_i' \sim \pi_i(\cdot)$, $\theta' \leftarrow (\theta_1, ..., \theta_{i-1}, \theta_i', \theta_{i+1}, ..., \theta_{2m})$

$\alpha(\theta'|\theta) \leftarrow \min\{1, \prod_{j=1}^{n} \frac{Pr(O=o_j|\theta')}{Pr(O=o_j|\theta)}\}$

with probability $\alpha(\theta'|\theta)$ accept $X_{t+1} = \theta'$; Otherwise, $X_{t+1} = \theta$.

end for=0

1.3 Model Averaging via Reversible-Jump MCMC

To average the model 1 and model 2, we use the model averaging and RJ-MCMC methods introduced in chapter 6 and 7 to simulate the parameters. To formulate the acceptance rate, we first review the general acceptance rate with 2 models in total. Let $\rho_{12} = \rho_{21} = 1$, given the model prior $\theta_M(\cdot)$, then

1. if $m' = m + 1$, set

$$\alpha(\theta', m'|\theta, m) = \min\{1, \frac{\pi(y|\theta', m')\pi_\theta(\theta')\pi_M(m')}{\pi(y|\theta, m)\pi_\theta(\theta)\pi_M(m)g_{m,m'}(u)} J_\phi(\theta, u)\}$$

2. if $m' = m - 1$, set

$$\alpha(\theta', m'|\theta, m) = \min\{1, \frac{\pi(y|\theta', m')\pi_\theta(\theta')\pi_M(m')g_{m',m}(u)}{\pi(y|\theta, m)\pi_\theta(\theta)\pi_M(m)} J_\phi(\theta, \emptyset)\}$$

Then, we assign an independent normal prior $N(0, 2)$ for each parameter(λ_i andβ_i). And we set $u \sim N(0, 2I)$, $\theta' = u$ and $u' = \theta'$. Then $\phi(\cdot, \cdot)$ is an identical map, therefore $J_\phi(\theta, u) = J_\phi(\theta, \emptyset) = 1$. So the RJ-MCMC algorithm is described as following pseudo-code:

Algorithm 2 Model Average via RJ-MCMC

Require: X_t, m
  if $m = 1$ **then**
    $m' \leftarrow m + 1$
    $u \sim N(\beta; 0, 2I)$
    $\theta' \leftarrow (\theta, u)$
    $\alpha(\theta', u|\theta, u) = \min\{1, \frac{\pi(y|\theta', m')\pi_M(m')}{\pi(y|\theta, m)\pi_M(m)}\}$
  else
    $m' \leftarrow m - 1$
    $\theta' \leftarrow \theta[1 : m]$
    $\alpha(\theta', u|\theta, u) = \min\{1, \frac{\pi(y|\theta', m')\pi_M(m')}{\pi(y|\theta, m)\pi_M(m)}\}$
  end if
  with probability $\alpha(\theta'|\theta)$ accept $X_{t+1} = \theta'$; Otherwise, $X_{t+1} = \theta$.
  for i=1,2...,m **do**
    Simulate $\theta_i' \sim N(0, 2)$, $\theta' \leftarrow (\theta_1, ..., \theta_{i-1}, \theta_i', \theta_{i+1}, ..., \theta_m)$
    $\alpha(\theta'|\theta) \leftarrow \min\{1, \prod_{j=1}^{n} \frac{Pr(O=o_j|\theta')}{Pr(O=o_j|\theta)}\}$
    with probability $\alpha(\theta'|\theta)$ accept $X_{t+1} = \theta'$; Otherwise, $X_{t+1} = \theta$.
  end for=0

2 Experiments

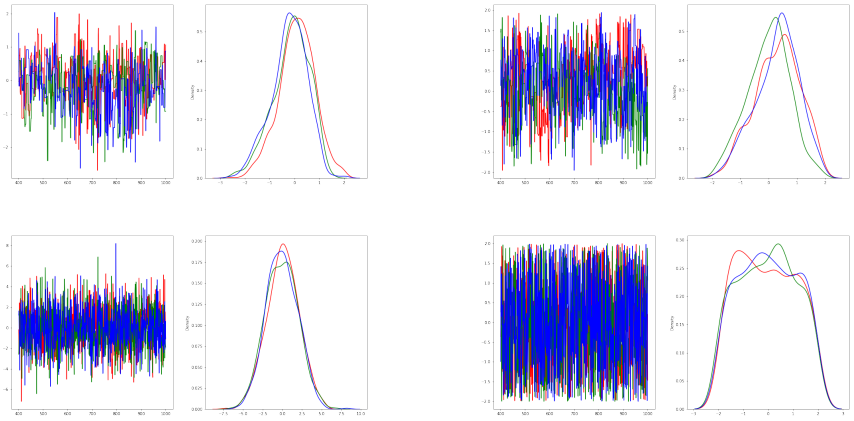
In this section, we choose two different priors to do simulation under model-1 and model-2, respectively. The priors are chosen as: $\pi_1(\theta) = N(\theta; 0, 2)$ and $\pi_2(\theta) = U(\theta; -2, 2)$. To compare the difference of priors, we first analyze the convergence of the MCMC. Then, we calculate the Effective Sample Size(ESS) of each model under different priors. Finally, we estimate the probability of the 7^{th} player ranking 1^{st} in the 68^{th} game by Monte Carlo.

2.1 Convergence Analysis

Firstly, we simulate the parameter θ for three times with 1000 steps for each time, and draw the posterior distribution to check the convergence.

It is shown that, under the two different priors: $N(0, 2)$ and $U(-2, 2)$, the MCMC would converge quickly. However, the shape of sampled density would be a little bit different with these priors. And I guess the main reason is the region of parameter is limited by uniform prior, which could only simulate random variables between -2 and 2.

To further visualize the simulation results, we also provide the traces plot and acf plot for each model under different prior, which would be shown in the appendix A.



(a) normal prior (b) uniform prior

Figure 1: The diagnostic plots for convergence checking: (a)Convergence checking under normal prior $N(0, 2)$. (Upper) Model 1; (Lower) Model 2. (Left) MCMC traces for λ_1; (Right) Density plots of the sampled λ_1. (b) Convergence checking under uniform prior $U(-2, 2)$.

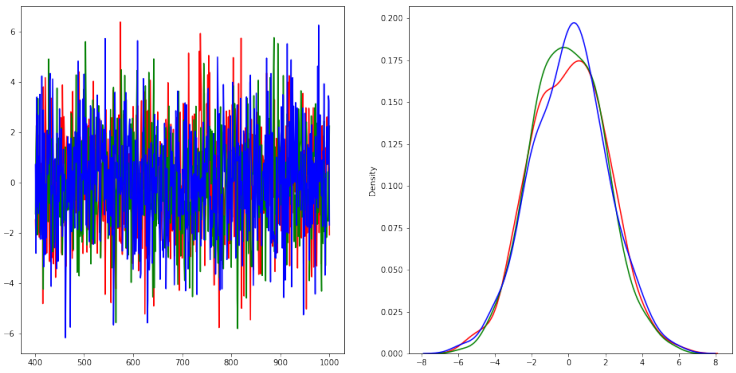


Figure 2: The diagnostic plots for averaging model: (Left) RJ-MCMC traces; (Right) Density plot of the samples.

2.2 Estimation

After simulating a series of samples of the parameters, we're now going to estimate the probability of the 7^{th} player rank 1^{st} in the

68^{th} game using Monte Carlo Method.

Since $Pr(O = o|\theta, M)$ is permutation invariant, then

$$\begin{aligned} Pr(\textbf{the } 7^{th} \textbf{ player rank } 1^{st} \textbf{ in the } 68^{th} \textbf{ game}) &= \sum_{o_2,...o_9} Pr(O = (7, 0_2, ..., 0_9)|\theta, M) \\ &= \frac{exp(\theta_7)}{\sum_{j=1}^{9} exp(\theta_j)} \end{aligned}$$

Thus, we are going to estimate the probability as following:

1. First, we simulate T samples of theta: $\theta^{(1)}, ..., \theta^{(T)}$
2. Then, the estimated prob is:

$$\hat{p} = \frac{1}{T} \sum_{t=1}^{T} \frac{exp(\theta_7^{(t)})}{\sum_{j=1}^{9} exp(\theta_j^{(t)})}$$

We've experimented on both the model 1 and model 2 under different priors. In our settings, we run the MCMC algorithm for $T = 1000$ steps, and set the "burn in" time as 400. Then the results are shown in the following table:

	normal	uniform
model 1	0.1090	0.1171
model 2	0.1109	0.1184
averaging model	0.1151	N/A

Table 1: The estimated prob with different models and priors.

It is shown that the probability of the 7^{th} player rank 1^{st} in the 68^{th} game is approximately 0.11.

2.3 Further Analysis - ESS

The Effective Sample Size(ESS) is usually used to compare the difference between two priors. The larger the ESS is, the more efficient the prior is. Thus, we've calculated the ESS of the normal prior and uniform prior for both the model 1 and model 2. The results are shown in the following table:

	normal	uniform
model 1	107.72	197.97
model 2	1000	1000

Table 2: The ESS of model 1 and model 2 under different priors.

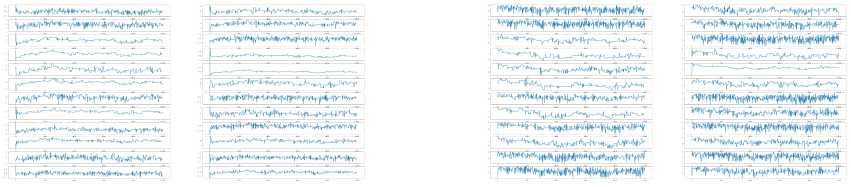
Thus, model 2 is quite efficient, ,no matter what the prior is. And, even if the ESS of uniform prior is larger than that of the normal prior in model 1, we're still in favor of the normal prior, who has the same region as the parameter space, while the region of uniform prior is restricted in $[-2, 2]$ for each parameter.

3 Conclusion and Discussion

Following the methods like Bayesian Inference, MCMC, RJ-MCMC and model averaging that introduced in the course - Bayesian Methods, we've implemented the relevant codes using Python, as well as experimenting and analyzing under this social rank approximation case. Under different priors, we found the MCMC algorithm does converge after some steps, and perform well. Besides, the model averaging via RJ-MCMC is efficient and more practical than the theories in the notes. Though there is quite a long proof and settings in the lecture notes, it's implementation is concise and straight forward. And finally, we've estimated the target under different models and priors, which show that the probability of the 7^{th} player rank 1^{st} in the 68^{th} game is approximately 0.11.

Appendix A Diagnostic Plots

A.1 Trace plots

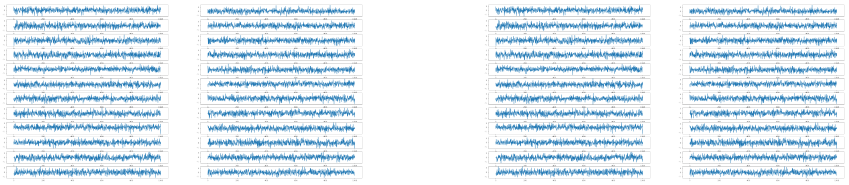


(a) normal prior (b) uniform prior

Figure 3: (Left) Traces plots of $\lambda_1, ..., \lambda_{24}$ of model 1 with normal prior; (Right) Traces plots of $\lambda_1, ..., \lambda_{24}$ of model 1 with uniform prior.

(a) normal prior (b) uniform prior

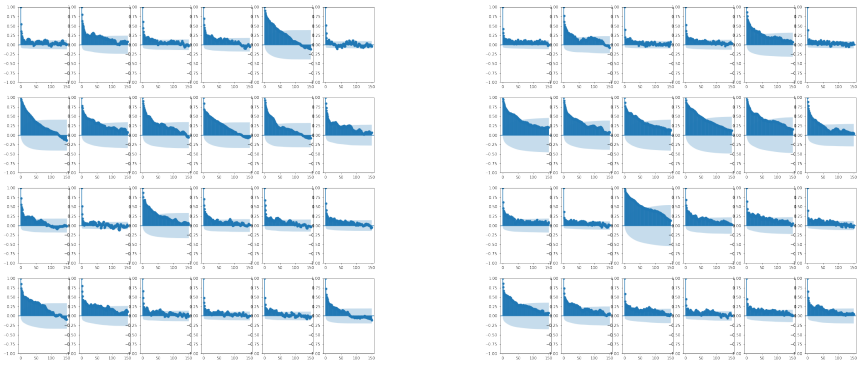
Figure 4: (Upper Left) Traces plots of $\lambda_1, ..., \lambda_{24}$ of model 2 with normal prior; (Lower Left)Traces plots of $\beta_1, ..., \beta_{24}$ of model 2 with normal prior;(Upper Right) Traces plots of $\lambda_1, ..., \lambda_{24}$ of model 2 with uniform prior; (Lower Right) Traces plots of $\beta_1, ..., \beta_{24}$ of model 2 with uniform prior.



(a) λ (b) β

Figure 5: (Left) Traces plot of $\lambda_1, ..., \lambda_{24}$ of averaging model; (Right) Traces plot of $\beta_1, ..., \beta_{24}$ of averaging model.

A.2 ACF plots



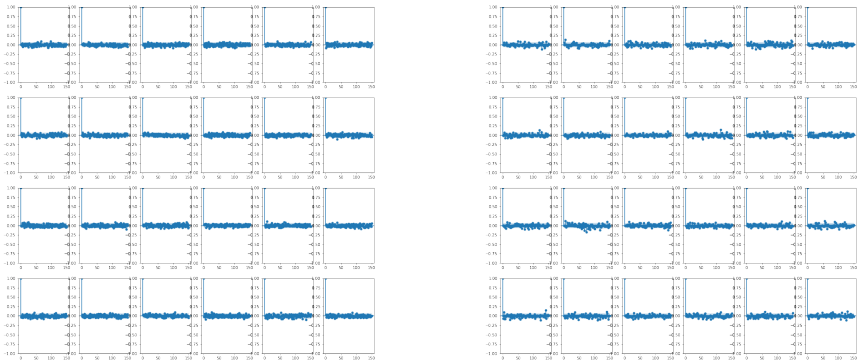
(a) normal prior (b) uniform prior

Figure 6: (Left) ACF plot of $\lambda_1, ..., \lambda_{24}$ of model 1 with normal prior; (Right) ACF plot of $\lambda_1, ..., \lambda_{24}$ of model 1 with uniform prior

(a) normal prior

(b) uniform prior

Figure 7: (Upper Left) ACF plot of $\lambda_1, ..., \lambda_{24}$ of model 2 with normal prior; (Lower Left)ACF plot of $\beta_1, ..., \beta_{24}$ of model 2 with normal prior;(Upper Right) ACF plot of $\lambda_1, ..., \lambda_{24}$ of model 2 with uniform prior; (Lower Right) ACF plot of $\beta_1, ..., \beta_{24}$ of model 2 with uniform prior.



(a) λ (b) β

Figure 8: (Left) ACF plot of $\lambda_1, ..., \lambda_{24}$ of averaging model; (Right) ACF plot of $\beta_1, ..., \beta_{24}$ of averaging model.

Appendix B Python Codes

B.1 utils.py

```
import re
import numpy as np


def dataloader():
    raw_data = ''
    with open('CompRank22.txt', 'r') as f:
        for line in f:
            raw_data += line.strip()
    raw_data = raw_data[5:-1].split('list')[1:]
    data = []
    for d in raw_data:
        tmp = re.findall(r'\d+', d)
        ind = int(len(tmp) / 2)
        data.append(np.array([tmp[: ind], tmp[ind:]], dtype=int) - 1)

    data_o = [tmp[0] for tmp in data]
    data_oe = [np.concatenate([tmp[0], tmp[1] + 24]) for tmp in data]
```

```
    return data_o, data_oe


def prob(index, theta):
    tmp = np.exp(theta[index] - np.max(theta[index]))
    return np.prod(tmp / np.sum(tmp))


def likelihood(data, theta):
    l = 1
    for d in data:
        l *= prob(d, theta)
    return l
```

B.2 plots.py

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from statsmodels.graphics import tsaplots

N = 24


def convergence_plot(trails, trail_time, burn_in):
    plt.subplots(figsize=(16, 8))
    plt.subplot(121)
    index = np.arange(burn_in, trail_time) + 1
    plt.plot(index, trails[0][burn_in:, 0], 'r')
    plt.plot(index, trails[1][burn_in:, 0], 'g')
    plt.plot(index, trails[2][burn_in:, 0], 'b')

    ax = plt.subplot(122)
    sns.kdeplot(trails[0][burn_in:, 0], ax=ax, color='r')
    sns.kdeplot(trails[1][burn_in:, 0], ax=ax, color='g')
    sns.kdeplot(trails[2][burn_in:, 0], ax=ax, color='b')

    plt.savefig('convergence_plot.png')
```

```
#     plt.show()

def trace_plot(samples):
    T = len(samples)
    x = np.arange(T) + 1
    plt.figure()
    plt.subplots(figsize=(32, 16))
    for i in range(24):
        plt.subplot(12, 2, i + 1)
        plt.plot(x, samples[:, i])
    plt.savefig('lambda_trace.png')

    if len(samples[0]) == 2 * N:
        plt.figure()
        plt.subplots(figsize=(32, 16))
        for i in range(24):
            plt.subplot(12, 2, i + 1)
            plt.plot(x, samples[:, i + N])
        plt.savefig('beta_trace.png')


def acf_plot(samples, lags=150):
    T = len(samples)
    plt.figure()
    plt.subplots(figsize=(16, 16))
    for i in range(24):
        ax1 = plt.subplot(4, 6, i + 1)
        tsaplots.plot_acf(samples[:, i], lags=lags, ax=ax1)
        plt.title('')
    plt.savefig('lambda_acf.png')

    if len(samples[0]) == 2 * N:
        plt.figure()
        plt.subplots(figsize=(16, 16))
        for i in range(24):
            ax2 = plt.subplot(4, 6, i + 1)
            tsaplots.plot_acf(samples[:, i + N], lags=lags, ax=ax2)
            plt.title('')
```

```
        plt.savefig('beta_acf.png')
```

B.3 MCMC.py

```
import numpy as np
import tqdm
from utils import likelihood
import copy
from scipy.stats import multivariate_normal


def MH(data, n, sampler, T=1000):
    samples = []
    sig2 = 5
    theta = sampler(n)
    samples.append(theta)
    for t in tqdm.tqdm_notebook(range(T - 1)):
        for i in range(n):
            theta_new = copy.deepcopy(theta)
            s = sampler(1)
            theta_new[i] = s
            tmp1 = likelihood(data, theta_new)
            tmp2 = likelihood(data, theta)
            alpha = min(1, tmp1 / tmp2)
            u = np.random.uniform(0, 1, 1)
            if u <= alpha:
                theta = theta_new
        samples.append(theta)
    return np.stack(samples)


def RJMCMC(data, T=1000):
    model_prior = [0.1, 0.9]
    m = np.random.choice([1, 2], 1, p=model_prior)
    theta = np.random.normal(0, 2, 48)
    if m == 1:
        theta[24:] = np.zeros(24)
    samples = [theta]
    var = multivariate_normal(mean=np.zeros(24), cov=np.eye(24) * 2)
```

```
    for t in tqdm.tqdm_notebook(range(T - 1)):
        if m == 1:
            beta_new = np.random.normal(0, 2, 24)
            theta_new = copy.deepcopy(theta)
            theta_new[24:] = beta_new
            tmp1 = likelihood(data, theta_new)
            tmp2 = likelihood(data, theta)
            alpha = min(1, tmp1 * 0.9 / (tmp2 * 0.1))
            m += 1
        else:
            theta_new = copy.deepcopy(theta)
            deleted = theta_new[24:]
            theta_new[24:] = np.zeros(24)
            tmp1 = likelihood(data, theta_new)
            tmp2 = likelihood(data, theta)
            alpha = min(1, tmp1 * 0.1 / (tmp2 * 0.9))
            m -= 1
        u = np.random.uniform(0, 1, 1)
        if u <= alpha:
            theta = theta_new

        for i in range(24):
            theta_new = copy.deepcopy(theta)
            s = np.random.normal(0, 2, 1)
            theta_new[i] = s
            tmp1 = likelihood(data, theta_new)
            tmp2 = likelihood(data, theta)
            alpha = min(1, tmp1 / tmp2)
            u = np.random.uniform(0, 1, 1)
            if u <= alpha:
                theta = theta_new
        samples.append(theta)
    return np.stack(samples)
```

B.4 main.py

```
from utils import dataloader, prob
from plots import convergence_plot, acf_plot, trace_plot
import matplotlib.pyplot as plt
```

```
from MCMC import MH
import os
import pickle
from itertools import permutations
import numpy as np


N = 24


def inference(data, samples):
    index = data[67]
    l = int(len(index) / 2)
    T = samples.shape[0]
    f = []
    print(samples.shape)
    for sample in samples:
        if samples.shape[1] == 24:
            tmp = sample[index]
        else:
            tmp = sample[index][:l] + sample[index][l:]
        f.append(np.exp(tmp[5]) / np.sum(np.exp(tmp)))
    print('Prob of 7-th player win in the 68-th game: {}'.format(np.mean(f)))
    return f


def pipeline(data, trail_time, burn_in, file_path, MH, sampler, is_scratch=Tru
    if not os.path.exists(file_path):
        os.mkdir(file_path)
    os.chdir(file_path)
    if is_scratch:
        trails = []
        trail_time = trail_time
        burn_in = burn_in
        for i in range(3):
            trails.append(MH(data, N * mode, sampler, T=trail_time))
        with open('param.pkl', 'wb') as f:
            dic = {'T': trail_time, 'burn_in': burn_in, 'trails': trails}
            pickle.dump(file=f, obj=dic)
```

```
    else:
        with open('param.pkl', 'rb') as f:
            dic = pickle.load(f)
            trail_time, burn_in, trails = dic['T'], dic['burn_in'], dic['trail
    samples = trails[-1]
    convergence_plot(trails, trail_time, burn_in)
    acf_plot(samples, lags=150)
    trace_plot(samples)
    os.chdir('..')
    os.chdir('..')

    return samples
```

B.5 run-model12.ipynb

```
from main import pipeline, inference
from utils import dataloader
from MCMC import MH
import numpy as np
import warnings

warnings.filterwarnings('ignore')

data_o, data_oe = dataloader()
trail_time = 1000
burn_in = 400
sampler1 = lambda x: np.random.normal(0, 2, size=x)
sampler2 = lambda x: np.random.uniform(-2, 2, size=x)
samples = []
for i, data in enumerate([data_o, data_oe]):
    tmp1 = pipeline(data, trail_time, burn_in, 'normal/model{}/'.format(i + 1)
    tmp2 = pipeline(data, trail_time, burn_in, 'uniform/model{}/'.format(i + 1
    samples.append([tmp1, tmp2])

f = []
for i, data in enumerate([data_o, data_oe]):
    f1 = inference(data, samples[i][0])
    f2 = inference(data, samples[i][1])
    f.append([f1, f2])
```

```
from mcmc_diagnostics.effective_sample_size import estimate_ess

for i, tmp in enumerate(f):
    ess1 = estimate_ess(np.array(tmp[0]))
    ess2 = estimate_ess(np.array(tmp[1]))
    print('Model-{}\nESS-normal: {}\nESS-uniform: {}'.format(i + 1, ess1, ess2
```

B.6 run-model3.ipynb

```
from main import pipeline, inference
from utils import dataloader
from MCMC import RJMCMC
import numpy as np
import warnings
import matplotlib.pyplot as plt
from plots import convergence_plot, acf_plot, trace_plot

warnings.filterwarnings('ignore')

data_o, data_oe = dataloader()
trail_time = 1000
burn_in = 400
trails = []

for i in range(3):
    trails.append(RJMCMC(data_oe, T=trail_time))
samples = trails[-1]
convergence_plot(trails, trail_time, burn_in)
acf_plot(samples, lags=150)
trace_plot(samples)
plt.show()

f = inference(data_oe, samples)

from mcmc_diagnostics.effective_sample_size import estimate_ess
ess1 = estimate_ess(np.array(f))
ess2 = estimate_ess(np.array(f))
print('Model-{}\nESS-normal: {}\nESS-uniform: {}'.format(i + 1, ess1, ess2))
```