STAT6011/7611/6111/3317
COMPUTATIONAL STATISTICS (2016 Fall)
Midterm Examination

Kei Ikegami (u3535947)

October 29, 2016

1

The code is below.

Listing 1: code

```
# import packages
from scipy.stats import invgamma
from scipy.stats import norm
from multiprocessing import Pool
from datetime import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline


# define functions
def integral(estimate, ts):
    elements = np.ones(len(ts) - 1)
    for i in range(len(ts) - 1):
        elements[i] = (ts[i+1] - ts[i])*(estimate[i+1] + estimate[i])/2
    return sum(elements)

def sum1(beta):
    return np.sum(p[:, 0] - beta * p[:, 3])

def sum2(alpha):
    return np.sum(p[:, 3] * (p[:, 0] - alpha))

def sum3(alpha, beta):
    l = (p[:, 0] - alpha - beta*p[:, 3])
    return np.sum(l * l )

def loglike(alpha, beta, sigma):
    return N*np.log(1/(np.sqrt(2*np.pi*sigma))) - sum3(alpha, beta) / (2*sigma)

def function1(w):

    if w < m:
        np.random.seed(datetime.now().microsecond)

        for i in range(n+1):
            t = ts[i]

            if i == 0:
                alphas[0] = 3000
                betas[0] = 185
                sigmas[0] = 90000

            else:
                alphas[0] = np.mean(alpha_sample)
                betas[0] = np.mean(beta_sample)
                sigmas[0] = np.mean(sigma_sample)

```

```python
            for j in range(sample_iter -1):

                location_alpha = (sigma_alpha*t*sum1(betas[j]) + sigmas[j]*mu_alpha) / (sigma_alpha * N*t + sigmas[j])
                scale_alpha = np.sqrt((sigma_alpha * sigmas[j]) / (sigma_alpha * N*t + sigmas[j]))
                alphas[j+1] = norm.rvs(loc = location_alpha, scale = scale_alpha)

                location_beta = (sigma_beta * t * sum2(alphas[j+1]) + sigmas[j] * mu_beta) / (sigma_beta *t* ssx + sigmas[j])
                scale_beta = np.sqrt((sigmas[j] * sigma_beta) / (sigma_beta *t* ssx + sigmas[j]))
                betas[j+1] = norm.rvs(loc = location_beta, scale = scale_beta)

                shape = N*t/2 + a
                invrate = 2*b / (b*t*sum3(alphas[j+1], betas[j+1]) + 2)
                sigmas[j+1] = invgamma.rvs(a = shape, scale = 1/ invrate)

            alpha_sample = alphas[burn_in:]
            beta_sample = betas[burn_in:len(betas)]
            sigma_sample = sigmas[burn_in:len(sigmas)]

            box = np.ones(len(alpha_sample))
            for k, l in enumerate(alpha_sample):
                box[k] = loglike(l, beta_sample[k], sigma_sample[k])

            estimates[i] = np.average(box)

        return estimates

def sum4(beta):
    return np.sum(p[:, 0] - beta * p[:, 4])

def sum5(alpha):
    return np.sum(p[:, 4] * (p[:, 0] - alpha))

def sum6(alpha, beta):
    l = (p[:, 0] - alpha - beta*p[:, 4])
    return np.sum(l * l )

def loglike2(alpha, beta, sigma):
    return N*np.log(1/(np.sqrt(2*np.pi*sigma))) - sum6(alpha, beta) / (2*sigma)

def function2(w):
    if w < m:
        np.random.seed(datetime.now().microsecond)

        for i in range(n+1):
            t = ts[i]

            if i == 0:
                gammas[0] = 3000
                deltas[0] = 185
                taus[0] = 90000

            else:
                gammas[0] = np.mean(gamma_sample)
                deltas[0] = np.mean(delta_sample)
                taus[0] = np.mean(tau_sample)

            for j in range(sample_iter - 1):

                location_alpha = (sigma_alpha*t*sum4(deltas[j]) + taus[j]*mu_alpha) / (sigma_alpha * N*t + taus[j])
                scale_alpha = np.sqrt((sigma_alpha * taus[j]) / (sigma_alpha * N*t + taus[j]))
                gammas[j+1] = norm.rvs(loc = location_alpha, scale = scale_alpha)

                location_beta = (sigma_beta * t * sum5(gammas[j+1]) + taus[j] * mu_beta) / (sigma_beta *t* ssz + taus[j])
                scale_beta = np.sqrt((taus[j] * sigma_beta) / (sigma_beta *t* ssz + taus[j]))
                deltas[j+1] = norm.rvs(loc = location_beta, scale = scale_beta)

                shape = N*t/2 + a
                invrate = 2*b / (b*t*sum6(gammas[j+1], deltas[j+1]) + 2)
                taus[j+1] = invgamma.rvs(a = shape, scale = 1/ invrate)

            gamma_sample = gammas[burn_in:]
```

```
            delta_sample = deltas[burn_in:len(deltas)]
            tau_sample = taus[burn_in:len(taus)]

            box2 = np.ones(len(gamma_sample))
            for k, l in enumerate(gamma_sample):
                box2[k] = loglike2(l, delta_sample[k], tau_sample[k])

            estimates[i] = np.average(box2)

        return estimates


# setting
pine = pd.read_table("pine.txt", delim_whitespace = True)
p = pine.values
pine['ave_x'] = pine['x'] - np.average(p[:, 1])
pine['ave_z'] = pine['z'] - np.average(p[:, 2])
p = pine.values

mu_alpha = 3000
sigma_alpha = 10**6
mu_beta = 185
sigma_beta = 10**4
a = 3
b = 1/(2*300**2)

N = np.shape(p)[0]
ssx = np.sum(p[:, 3] * p[:, 3])
ssz = np.sum(p[:, 4] * p[:, 4])

n = 10
c = 2
ts = [(i/n)**c for i in range(n+1)]
estimates = np.ones(n+1)


sample_iter = 100000
burn_in = 30000
m = 100
core = 4

alphas = np.ones(sample_iter)
betas = np.ones(sample_iter)
sigmas = np.ones(sample_iter)

gammas = np.ones(sample_iter)
deltas = np.ones(sample_iter)
taus = np.ones(sample_iter)


# model 1's marginal likelihood
if __name__ == '__main__':
    w = Pool(core)
    result1 = w.map(function1, range(m))

expect1 = np.ones(10)
for i in range(10):
    expect1[i] = integral(result1[i], ts)


# model 2's marginal likelihood
if __name__ == '__main__':
    result2 = w.map(function2, range(m))

expect2 = np.ones(10)
for i in range(10):
    expect2[i] = integral(result2[i], ts)


# computing BF_21
bf_21 = []
for a,b in zip(expect1, expect2):
    bf_21.append(np.exp(b-a))

# the upper left cell in Table1
```

```
bias = np.average(bf_21) - 4862
std = np.sqrt(np.var(bf_21))
```

The result is .

The main idea of this paper is the below identity.

$$
\begin{aligned}
\log\{p(\mathbf{y})\} &= \log\left\{\frac{z(\mathbf{y}|t=1)}{z(\mathbf{y}|t=0)}\right\} = \left[\log\{z(\mathbf{y}|t)\}\right]_0^1 = \int_0^1 \frac{1}{z(\mathbf{y}|t)}\frac{\mathrm{d}}{\mathrm{d}t} z(\mathbf{y}|t)\mathrm{d}t \\
&= \int_0^1 \frac{1}{z(\mathbf{y}|t)}\left(\int_\theta \frac{\mathrm{d}}{\mathrm{d}t} p(\mathbf{y}\theta)^t p(\theta)\theta\right)\mathrm{d}t = \int_0^1 \frac{1}{z(\mathbf{y}|t)}\left(\int_\theta \log\{p(\mathbf{y}|\theta)\}\, p(\mathbf{y}|\theta)^t p(\theta)\mathrm{d}\theta\right)\mathrm{d}t \\
&= \int_0^1 \int_\theta \log\{p(\mathbf{y}|\theta)\}\frac{p(\mathbf{y}|\theta)^t p(\theta)}{z(\mathbf{y}|t)}\mathrm{d}\theta\mathrm{d}t = \int_0^1 \int_\theta \log\{p(\mathbf{y}|\theta)\}\, p_t(\theta|\mathbf{y})\mathrm{d}\theta\mathrm{d}t \\
&= \int_0^1 \mathrm{E}_{\theta|\mathbf{y},t}\left[\log\{p(\mathbf{y}|\theta)\}\right]\mathrm{d}t
\end{aligned}
$$