1 Metropolis: The Bounded Normal Mean

We choose uniform distribution on $[-2, 2]$ as the proposal distribution. The algorithm for generating samples from posterior is shown as below.

- **Step 1.** Start with arbitrary θ_0
- **Step 2.** At stage n, generate proposal θ' from $Unif(-2, 2)$.
- **Step 3.** Determine θ_{n+1} as
  - $\theta_{n+1} = \theta'$ with probability $\rho(\theta_n, \theta')$;
  - $\theta_{n+1} = \theta'$ with probability $1 - \rho(\theta_n, \theta')$.
- **Step 4.** Set $n = n + 1$ and go to **Step 2**.

We derive the acceptance ratio as follows.
As we know that the observations are taken from normal distribution with precision $\tau = 1/4$, that is $f(y|\theta) \propto \frac{1}{2}\exp\{-\frac{1}{8}(y-\theta)^2\}$, then the target density function $\pi(\theta|X)$ is proportional to

$$\begin{aligned}\pi(\theta|\mathbf{y}) &\propto f(\mathbf{y}|\theta)\pi(\theta)\\ &\propto \prod_{i=1}^{n} \exp\Big\{-\frac{1}{8}(y_i-\theta)^2\Big\}\frac{1}{2}\cos^2\Big(\frac{\pi\theta}{4}\Big)\\ &= \exp\Bigg\{-\frac{1}{8}\sum_{i=1}^{n}(y_i-\theta)^2\Bigg\}\frac{1}{2}\cos^2\Big(\frac{\pi\theta}{4}\Big)\end{aligned}$$

As we choose independent proposal distribution, then the proposed θ' and θ_n are independent. In this case, we have

$$q(\theta_n|\theta') = \frac{1}{4}, \quad q(\theta'|\theta_n) = \frac{1}{4}.$$

Thus we compute the acceptance ratio as

$$\rho(\theta_n, \theta') = \min\left\{1, \frac{\pi(\theta')}{\pi(\theta_n)}\frac{q(\theta_n|\theta')}{q(\theta'|\theta_n)}\right\} = \min\left\{1, \frac{\exp\Big\{-\frac{1}{8}\sum_{i=1}^{n}(y_i-\theta')^2\Big\}\frac{1}{2}\cos^2\Big(\frac{\pi\theta'}{4}\Big)}{\exp\Big\{-\frac{1}{8}\sum_{i=1}^{n}(y_i-\theta_n)^2\Big\}\frac{1}{2}\cos^2\Big(\frac{\pi\theta_n}{4}\Big)}\right\}.$$

(a) We choose the proposal as the uniform distribution over $[-2, 2]$, which means we have a Metropolis random walk. By simulation, we obtain Figure 1.

(b) Bayesian estimator is -0.397 and the 95% credible set is $[-1.442, 0.750]$.

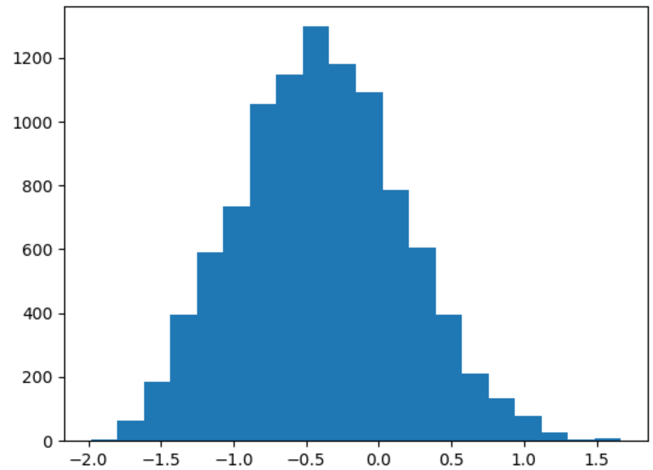

Figure 1: One simulation for the posterior.

2 Gibbs Sampler and High/Low Protein Diet in Rats

First, we denote that $\mathbf{y}_1 = (y_{1i})_{i=1}^{12\top}$ and $\mathbf{y}_2 = (y_{2i})_{i=1}^{7\top}$ as vectors representing the 12 observations under high protein and 7 observations under lower protein, respectively.

(a) For θ_1, we have that

$$\begin{aligned}\pi(\theta_1, \tau_1, \mathbf{y}_1) &= f(\mathbf{y}_1|\theta_1, \tau_1)\pi(\theta_1)\pi(\tau_1)\\ &\propto \exp\left(-\frac{\tau_1}{2}\sum_{i=1}^{12}(y_{1i}-\theta_1)^2 - \frac{1}{200}(\theta_1-\theta_{10})^2 - 4\tau_1\right)\tau_1^{5.01}.\end{aligned}$$

Therefore, for θ_1, we have

$$\pi(\theta_1|\tau_1, \mathbf{y}_1) \propto \exp\left(-\frac{12\tau_1 + 1/100}{2}\left(\theta_1 - \frac{1440\tau_1 + 110/100}{12\tau_1 + 1/100}\right)^2\right),$$

which means $[\theta_1|\tau_1, \mathbf{y}_1] \sim \mathcal{N}(\frac{1440\tau_1+110/100}{12\tau_1+1/100}, \frac{1}{12\tau_1+1/100})$. Similarly, we obtain $[\tau_1|\theta_1, \mathbf{y}_1] \sim \mathcal{G}a(6.01, 4 + \frac{\sum_{i=1}^{12}(y_{1i}-\theta_1)^2}{2})$.

Furthermore, we obtain the sampler for the $[\theta_2|\tau_2, \mathbf{y}_2] \sim \mathcal{N}(\frac{707\tau_2+110/100}{7\tau_2+1/100}, \frac{1}{7\tau_2+1/100})$ and $[\tau_2|\theta_2, \mathbf{y}_2] \sim \mathcal{G}a(6.01, 4 + \frac{\sum_{i=1}^{7}(y_{2i}-\theta_2)^2}{2})$.

(b) By simulation, we obtain that $\theta_1 - \theta_2 = 19$ and the proportion of positive differences equals to 1.

(c) The credible set is $[18.989, 19.010]$, which does not contain 0.

A Code for Problem 1

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

matplotlib.use('TkAgg')
rand.seed(100)

class Metrolis_hasting(object):
    def post(self,a,x):
        #if not isinstance(x,float):
        #     raise Exception("Input Value Error!")
        f = np.exp(-1/8*np.sum((a-x)**2)) * 1/2 * np.cos(np.pi*x/4)**2
        return f

    def proposal(self,x,y):
        return 1

    def Update(self,a,x):
        y = rand.uniform(-2,2,1)
        ratio = min(1,self.post(a,y)*self.proposal(x,y)
                    / (self.post(a,x)*self.proposal(y,x)))
        accept_ratio = rand.uniform(0,1,1)
        if accept_ratio <= ratio:
            return y
        return x

a = np.array([-3,-3,4,-7,0,4])
x=0
Sampler = Metrolis_hasting()

## warm_up
for _ in range(500):
    x = Sampler.Update(a,x)

## statistics
result = np.array([])
for _ in range(10000):
    x = Sampler.Update(a,x)
```

```
    result = np.append(result,np.array(x),0)

estimate = np.mean(result)
result_sort = np.sort(result)
print(estimate)
print(result_sort[249],result_sort[9749])
plt.hist(result,bins=20)
plt.show()
```

B Code for Problem 2

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import numpy.random as rand
matplotlib.use('TkAgg')

rand.seed(10)

class Gibbs(object):
    def normal(self,a,tau):
        #if not isinstance(x,float):
        #    raise Exception("Input Value Error!")
        p = len(a)
        f = (np.sum(a)*tau+110/100)/(p*tau+1/100) + np.sqrt(1/(p*tau+1/100))*
        return f

    def gamma(self,a,x):
        par = np.sum((a-x)**2)/2
        f = rand.gamma(6.01,4+par)
        return f

a_1 = np.array([134,146,104,119,124,161,107,83,113,129,97,123])
a_2 = np.array([70,118,101,85,107,132,94])
theta_1=0
theta_2=0
tau_1 = 10
tau_2 = 10
Sampler = Gibbs()
## warm_up

for _ in range(500):
    theta_1 = Sampler.normal(a_1,tau_1)
    theta_2 = Sampler.normal(a_2,tau_2)
    tau_1 = Sampler.gamma(a_1, theta_1)
    tau_2 = Sampler.gamma(a_2, theta_2)

result_1 = np.array([])
result_2 = np.array([])
result_diff = np.array([])
```

```
count=0
## statistics
for _ in range(10000):
    theta_1 = Sampler.normal(a_1, tau_1)
    theta_2 = Sampler.normal(a_2, tau_2)
    tau_1 = Sampler.gamma(a_1, theta_1)
    tau_2 = Sampler.gamma(a_2, theta_2)
    if theta_1>theta_2:
        count += 1
    result_1 = np.append(result_1, np.array(theta_1), 0)
    result_2 = np.append(result_2, np.array(theta_2), 0)
    result_diff = np.append(result_diff,np.array(theta_1-theta_2),0)

print(count/10000)

estimate_1 = np.mean(result_1)
estimate_2 = np.mean(result_2)
result_sort = np.sort(result_diff)

print(estimate_1)
print(estimate_2)
print(result_sort[249],result_sort[9749])
```