STAT 52900
Applied Decision Theory & Bayesian Statistics
Homework Assignments #06

By – Aditya Gaitonde

Q2)

7.1)

Code –

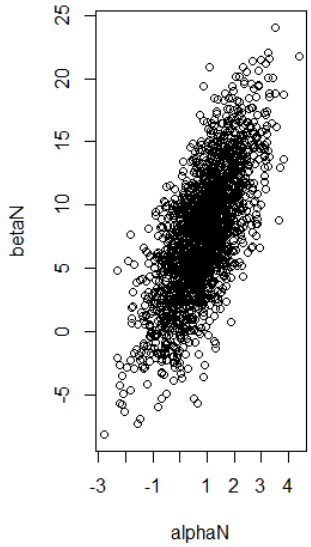
```
library("ggplot2")
library('dplyr')
df1 <- data.frame(
  x = c(-0.86, -0.30, -0.05, 0.73),
  n = c(5, 5, 5, 5),
  y = c(0, 1, 3, 5))
dose<-c(-0.863,-0.296,-0.053,0.727)
mu <- c(0.8732163,7.9110577)
Sigma <- matrix(data=c(1.032047^2,0.7251766*1.032047*4.98319,
                       0.7251766*1.032047*4.98319, 4.98319^2), nrow=2,ncol=2)
Sample from the approximating Normal distribution and zscatterplot
alphaN <- c(2000,0,0)
betaN <- c(2000,0,0)
for (i in 1:2000) {
  x <- mu + t(chol(Sigma)) %*% rnorm(2)
  alphaN[i] <- x[1]
  betaN[i] <- x[2]
```

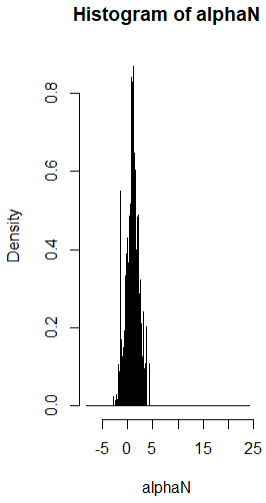
}

plot(alphaN,betaN)

hist(alphaN,betaN)

Output –





Code –

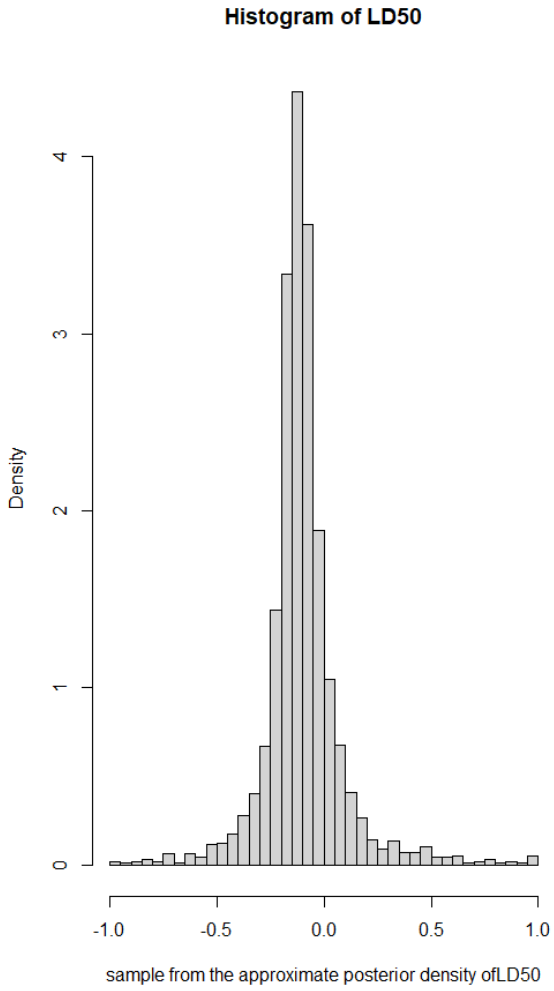
LD50 <- -alphaN/betaN

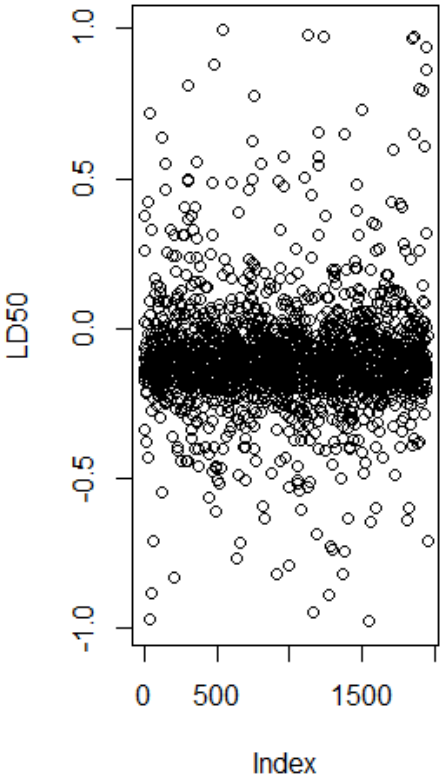
LD50 <- LD50[LD50>-1.0 & LD50 <1.0] # to avoid extreme sample # values of LD50 that distort the histogram

hist(LD50, xlab="sample from the approximate posterior density ofLD50", nclass=30,prob=TRUE)

plot(LD50)

Output –





Code –

Limits for the graph

```
xl <- c(-1.5, 7)
yl <- c(-5, 35)
A = seq(-1.5, 7, length.out = 100)
B = seq(-5, 35, length.out = 100)
```

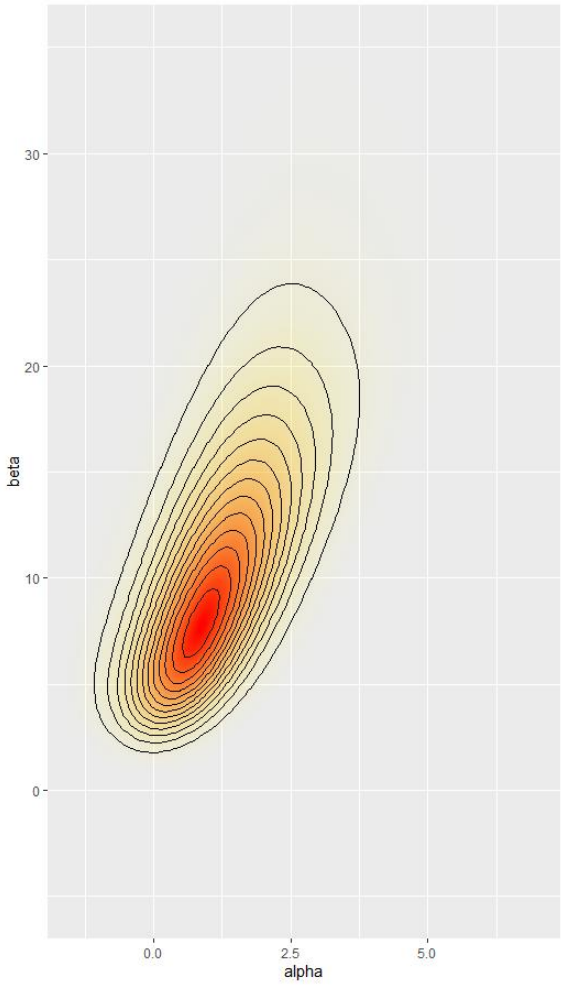
```
Making the vectors that contain all pairwise combinations of A and B
cA <- rep(A, each = length(B))
cB <- rep(B, length(A))

Helper function to calculate the log likelihood
logl <- function(df, a, b)
df['y']*(a + b*df['x']) - df['n']*log1p(exp(a + b*df['x']))

Calculate likelihoods: apply logl function for each observation
p <- apply(df1, 1, logl, cA, cB) %>% rowSums() %>% exp()

Plot for posterior density
pos <- ggplot(data = data.frame(cA ,cB, p), aes(x = cA, y = cB)) +
  geom_raster(aes(fill = p, alpha = p), interpolate = T) +
  geom_contour(aes(z = p), colour = 'black', size = 0.2) +
  coord_cartesian(xlim = xl, ylim = yl) +
  labs(x = 'alpha', y = 'beta') +
  scale_fill_gradient(low = 'yellow', high = 'red', guide = F) +
  scale_alpha(range = c(0, 1), guide = F)
plot(pos)
```

Plot –



10.3)

Based on the information provided, the significance level is $\alpha = 0.05$, and the critical value for a right tailed test $z_c = 1.64$. The rejection region for this right tailed test is $R = \{z: x > 1.6449\}$

The z statistic is computed as follows –

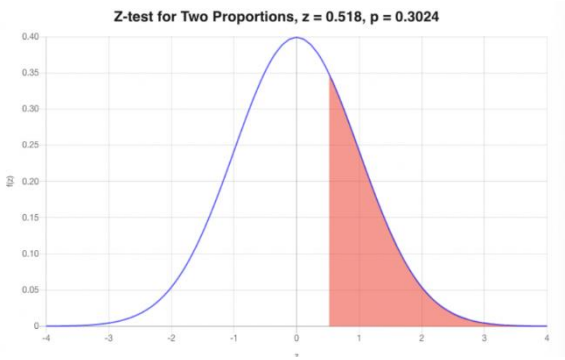
$$z = \frac{\widehat{p_1} - \widehat{p_2}}{\sqrt{\bar{p}(1-\bar{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

$$z = \frac{0.6 - 0.5}{\sqrt{0.5333(1-0.5333)\left(\frac{1}{10} + \frac{1}{20}\right)}} = 0.518$$

Since it is observed that $z = 0.5175 \leq z_c = 1.6449$, it is then concluded that the null hypothesis is not rejected.

Using the P value approach: The p – value is p = 0.3024 and since $p = 0.3024 \geq 0.05$ it is concluded that the null hypothesis is not rejected. Therefore, there is not enough evidence to claim that the population p_1 is greater than p_2 at the $\alpha = 0.05$ significance level

The 95% confidence interval for $p_1 - p_2$ is $-0.2745 < p_1 - p_2 < 0.474$



Question 4)

Code –

y<- c(0,1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1)

In the above y there are 9 failures and 6 successes.

Assume the uniform prior Beta(1,1).

From the Beta Binomial Conjugation,

$$\alpha = 1, \qquad \beta = 1$$

For the likelihood Binomial Distribution,

$$f(y|\theta) = \binom{n}{y} \theta^y (1-\theta)^{n-y}$$

Posterior distribution, Beta(a+y, b+n-y)

n = 15

y = 6

n-y=9

Hence, we get a posterior distribution of the form

Beta (1+6; 1+9)= Beta(7,10)

Before seeing the data, point estimate of posterior mean was –

$$\frac{\alpha}{\alpha + \beta} = \frac{1}{2} = 0.5$$

After seeing the data (y), point estimate of posterior mean =

$$\frac{\alpha}{\alpha + \beta} = \frac{7}{10 + 7} = \frac{7}{17} = 0.412$$

Mode –

$$\frac{\alpha - 1}{\alpha + \beta - 2} = \frac{7 - 1}{17 - 2} = \frac{6}{15} = 0.4$$

Hypothesis:

$$H_0: p < 0.5 \; VS \; H_1: p > 0.5$$

$$\Pr(\theta < 0.5) = \int_0^{0.5} \pi(\theta|y)d\theta = CDF(\theta|y)|_{\theta=0.5}$$

Code –

pbeta(0.5,7,10)

Output –

[1] 0.7727509

This means that we are 77.27% confident that coin is biased towards tail.

$$\Pr(\theta > 0.5) = \int_{0.5}^{1} \pi(\theta|y)d\theta = 1 - CDF(\theta|y)|_{\theta=0.5}$$

Code –

1-pbeta(0.5,7,10)

Output –

[1] 0.2272491

Bayes Factor(K)

$$Bayes\ Factor(K) = \frac{p(H_0)}{p(H_1)} = \frac{p(\theta < 0.5)}{p(\theta > 0.5)} = \frac{0.7727509}{0.2272491} = 2.179353$$

Since, the Bayes factor-2.18 which is greater than 1, hence Ho is true.

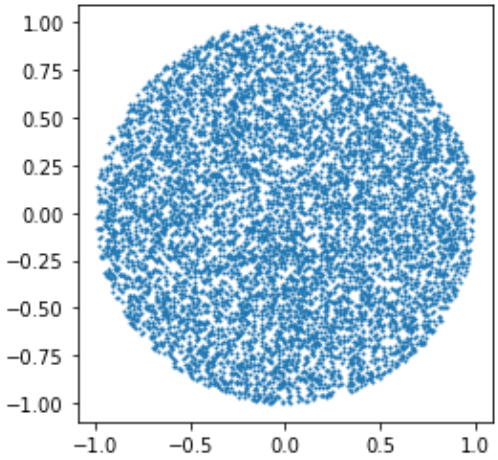
The posterior probability of $H_0 = CDF(\theta|y)|_{\theta=0.5} = pbeta(0.5,7,10)$.

Question 5)

Code –

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
x = np.random.uniform(-1, 1, (10000, 2))
x = x[np.sum(x**2, axis=1) < 1]
plt.scatter(x[:, 0], x[:,1], s=1)
plt.axis('square')
pass
```

Plot –

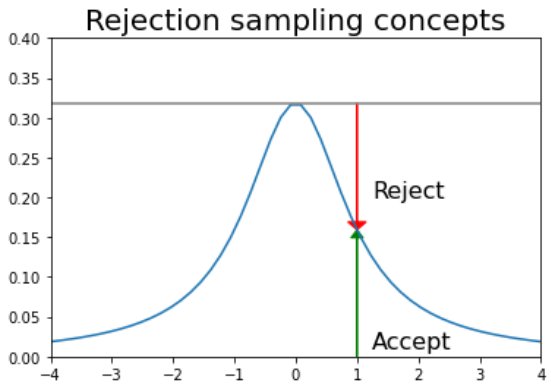


Example: Rejection sampling from uniform distribution

Code –

```
x = np.linspace(-4, 4)
df = 10
dist = stats.cauchy()
upper = dist.pdf(0)
plt.plot(x, dist.pdf(x))
plt.axhline(upper, color='grey')
px = 1.0
plt.arrow(px,0,0,dist.pdf(1.0)-0.01, linewidth=1,
        head_width=0.2, head_length=0.01, fc='g', ec='g')
plt.arrow(px,upper,0,-(upper-dist.pdf(px)-0.01), linewidth=1,
        head_width=0.3, head_length=0.01, fc='r', ec='r')
plt.text(px+.25, 0.2, 'Reject', fontsize=16)
plt.text(px+.25, 0.01, 'Accept', fontsize=16)
plt.axis([-4,4,0,0.4])
plt.title('Rejection sampling concepts', fontsize=20)
pass
```

Plot –

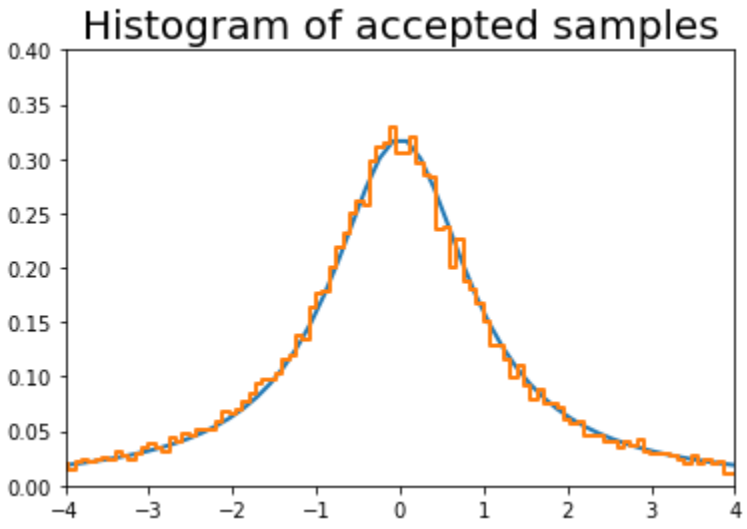


Simple Monte Carlo Integration and intuition behind it,

Code –

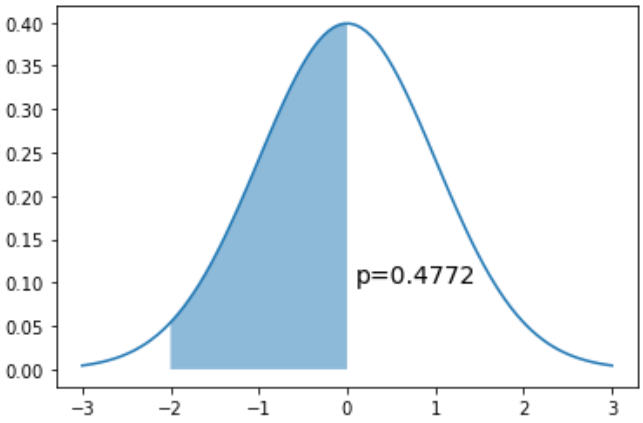
```
n = 100000
u = np.random.uniform(-4, 4, n)
r = np.random.uniform(0, upper, n)
v = u[r < dist.pdf(u)]
plt.plot(x, dist.pdf(x), linewidth=2)
# Plot scaled histogram
factor = dist.cdf(4) - dist.cdf(-4)
hist, bin_edges = np.histogram(v, bins=100, normed=True)
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2.
plt.step(bin_centers, factor*hist, linewidth=2)
plt.axis([-4,4,0,0.4])
plt.title('Histogram of accepted samples', fontsize=20)
pass
```

Plot –



Code –

```
from scipy import stats
x = np.linspace(-3,3,100)
dist = stats.norm(0,1)
a = -2
b = 0
plt.plot(x, dist.pdf(x))
plt.fill_between(np.linspace(a,b,100), dist.pdf(np.linspace(a,b,100)), alpha=0.5)
plt.text(b+0.1, 0.1, 'p=%.4f' % (dist.cdf(b) - dist.cdf(a)), fontsize=14)
pass
```



Code –

```
from scipy.integrate import quad
y, err = quad(dist.pdf, a, b)
y
```

Output –

0.47724986805182085

If we can sample directly from the target distribution N(0,1)

Code –

```
n = 10000
x = dist.rvs(n)
np.sum((a < x) & (x < b))/n
```

Output –

0.4768

Code –

```
n = 10000
x = np.random.uniform(a, b, n)
np.mean((b-a)*dist.pdf(x))
```

Output –

0.4738861418973744

Monte Carlo Integration

Code –

```
for n in 10**np.array([1,2,3,4,5,6,7,8]):
    x = np.random.uniform(0, 1, n)
    sol = np.mean(np.exp(x))
    print('%10d %.6f' % (n, sol))
```

Output –

10 1.657157

100 1.729237

1000 1.730363

10000 1.718365

100000 1.713790

1000000 1.717539

10000000 1.718316

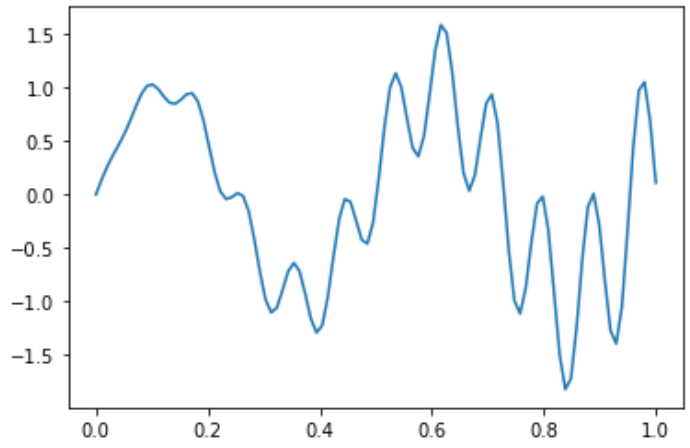
100000000 1.718217

Monitoring variance in Monte Carlo integration

Code –

```
def f(x):
    return x * np.cos(71*x) + np.sin(13*x)
x = np.linspace(0, 1, 100)
plt.plot(x, f(x))
pass
```

Plot –



Single MC Integration Estimate

Code –

```
n = 100
x = f(np.random.random(n))
y = 1.0/n * np.sum(x)
y
```

Output –

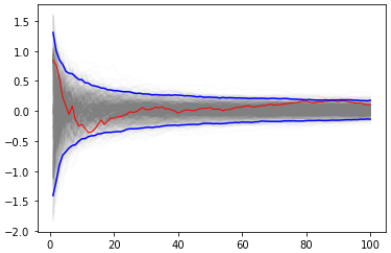
```
-0.09402249509278582
```

Using multiple independent sequences to monitor convergence

Code –

```
n = 100
reps = 1000
x = f(np.random.random((n, reps)))
y = 1/np.arange(1, n+1)[:, None] * np.cumsum(x, axis=0)
upper, lower = np.percentile(y, [2.5, 97.5], axis=1)
plt.plot(np.arange(1, n+1), y, c='grey', alpha=0.02)
plt.plot(np.arange(1, n+1), y[:, 0], c='red', linewidth=1);
plt.plot(np.arange(1, n+1), upper, 'b', np.arange(1, n+1), lower, 'b')
pass
```

Plot –



Variance Reduction and change of variables.

Code –

```
import scipy.stats as stats
h_true = 1 - stats.cauchy().cdf(3)
h_true
```

Output –

0.10241638234956674

Code –

```
n = 100
x = stats.cauchy().rvs(n)
h_mc = 1.0/n * np.sum(x > 3)
h_mc, np.abs(h_mc - h_true)/h_true
```

Output –

(0.18, 0.7575313233153023)

A change of variables lets us use 100% of draws

Code –

```
y = stats.uniform().rvs(n)
h_cv = 1.0/n * np.sum(3.0/(np.pi * (9 + y**2)))
h_cv, np.abs(h_cv - h_true)/h_true
```

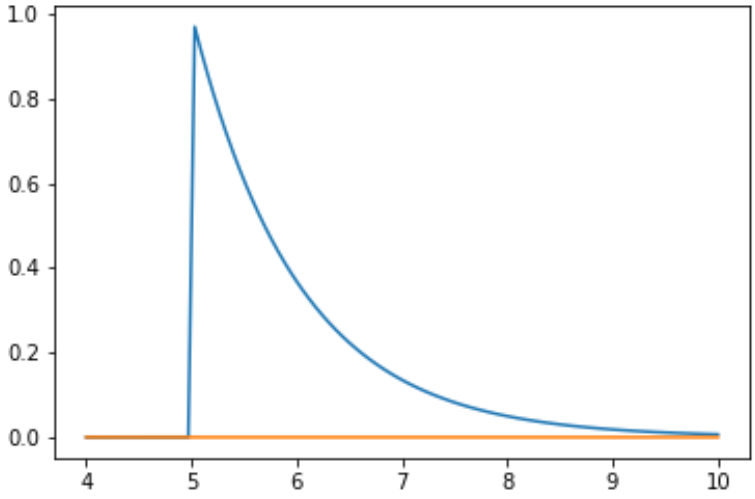
Output –

(0.1026423327183608, 0.0022061936148345426)

Code –

```
x = np.linspace(4, 10, 100)
plt.plot(x, stats.expon(5).pdf(x))
plt.plot(x, stats.norm().pdf(x))
pass
```

Plot –



Expected answer

Code –

```
v_true = 1 - stats.norm().cdf(5)
v_true
```

Output –

2.866515719235352e-07

Using direct Monte Carlo integration

Code –

```
n = 10000
y = stats.norm().rvs(n)
v_mc = 1.0/n * np.sum(y > 5)
# estimate and relative error
v_mc, np.abs(v_mc - v_true)/v_true
```

Output –

(0.0, 1.0)

Using importance sampling

Code –

```
n = 10000
y = stats.expon(loc=5).rvs(n)
v_is = 1.0/n * np.sum(stats.norm().pdf(y)/stats.expon(loc=5).pdf(y))
# estimate and relative error
v_is, np.abs(v_is- v_true)/v_true
```

Output –

(2.861214879846362e-07, 0.001849227392481794)