CODING SESSION SOLUTIONS 25TH OCTOBER 2022

SOLUTION Q1

:

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

def p(x, mu, sig):
    return 1/np.sqrt(2 * np.pi * sig**2) * np.exp(-0.5 * (x - mu)**2 / (sig**2))

def q(x, alpha):
    return (alpha/2) * np.exp(-alpha*np.abs(x))

alpha = 1 # optimal derived in Ex. 2.5
M = np.sqrt(2 * np.e / np.pi) # optimal derived in Ex. 2.5

n = 100000

x_samples = np.array([])

acc = 0

for i in range(n):
    x = np.random.laplace(0, 1/alpha) # proposal
    u = np.random.uniform(0, 1) # uniform

    if u < p(x, 0, 1)/(M * q(x, alpha)): # accept - reject
        x_samples = np.append(x_samples, x) # store sample if accepted
        acc += 1 # count accepted samples (for acceptance rate)


xx = np.linspace(-3, 3, 1000)

print(1/M) # theoretical acceptance rate
print(acc/n) # empirical acceptance rate

plt.hist(x_samples, bins = 100, density=True)
plt.plot(xx, p(xx, 0, 1), 'r-')
plt.show()
```

SOLUTION Q2

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

w_1 = 0.8
w_2 = 0.2

mu_1 = 2
mu_2 = -2

sigma_1 = 0.2
sigma_2 = 0.2

```

```
a_1 = 0.5
a_2 = 0.5

w = np.array([w_1, w_2])


def sample_discrete(w):  # draws a single index (0,...,K-1) from a discrete distribution with probabilities w where K is length of CDF
    cw = np.cumsum(w)
    sample = []

    u = np.random.uniform(0, 1)

    for k in range(len(cw)):
        if cw[k] > u:
            sample = k
            break

    return sample


def sample_truncated_gauss(mu, sigma, a, n):
    x_samples = np.array([])

    while len(x_samples) < n:  # keep sampling until we have n samples
        x_prop = np.random.normal(mu, sigma, 1)  # sample from the proposal distribution
        if mu - a < x_prop < mu + a:  # check if the sample is in the support of the truncated Gaussian
            x_samples = np.append(x_samples, x_prop)  # if it is, add it to the array of samples

    return x_samples


N = 10000

x_samples = np.array([])

# mixture sampling below (can be made a function)

for i in range(N):
    samp = sample_discrete(w)  # sample an index from the discrete distribution

    if samp == 0:  # if the index is 0, sample from the first truncated Gaussian
        x = sample_truncated_gauss(mu_1, sigma_1, a_1, 1)
    else:  # if the index is 1, sample from the second truncated Gaussian
        x = sample_truncated_gauss(mu_2, sigma_2, a_2, 1)

    x_samples = np.append(x_samples, x)  # add the sample to the array of samples

```

```
plt.hist(x_samples, bins=100, density=True, rwidth=0.8, color='r',
                                alpha=0.5)
plt.show()
```

SOLUTION Q3

Marginal distribution on the circle can be derived as

$$p_{x_1}(x_1) = \int_{-\sqrt{1-x_1^2}}^{\sqrt{1-x_1^2}} p_{x_1,x_2}(x_1, x_2)\mathrm{d}x_2.$$

This will result in

$$\begin{aligned} p_{x_1}(x_1) &= \left[\frac{1}{\pi}\right]_{-\sqrt{1-x_1^2}}^{\sqrt{1-x_1^2}}, \\ &= \frac{2}{\pi}\sqrt{1-x_1^2}, \qquad \text{for } x_1^2 < 1. \end{aligned}$$

Verify that this is a probability density

$$\int p_{x_1}(x_1)\mathrm{d}x_1 = \frac{2}{\pi}\int_{-1}^{1}\sqrt{1-x_1^2}\mathrm{d}x_1.$$

This is indeed one (the integral is arcsin. To compute marginal, sample from the circle and plot one axis of it:

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

# sample uniformly within a circle
def sample_circle(n):
    x_1 = np.zeros(n)
    x_2 = np.zeros(n)
    for i in range(n):
        while True:
            x_1[i] = np.random.uniform(-1, 1)
            x_2[i] = np.random.uniform(-1, 1)
            if x_1[i]**2 + x_2[i]**2 <= 1:
                break
    return x_1, x_2

# plot the circle and samples
def plot_circle(x_1, x_2):
    fig = plt.figure(figsize=(7, 7))
    plt.plot(x_1, x_2, 'k.')
    t = np.linspace(0, 2 * np.pi, 100)
    plt.plot(np.cos(t), np.sin(t), 'r-')
    plt.xlim([-1, 1])
    plt.ylim([-1, 1])
    plt.show()

n = 100000
x_1, x_2 = sample_circle(n)
# plot_circle(x, y)

# marginal of x
```

```python
def marginal(x):
    return (2/np.pi) * np.sqrt(1 - x**2)

# plot the marginal of x and histogram of x
xx = np.linspace(-1, 1, 1000)

fig, axs = plt.subplots(1, 1, figsize=(7, 7))
axs.hist(x_1, bins=50, density=True, color='k', alpha=1)
axs.plot(xx, marginal(xx), color=[0.8, 0, 0], linewidth=2)
plt.show()
```