MATH60047: Stochastic Simulation Coursework 2

CID: 01938572

1 Q1: Importance Sampling for Marginal Likelihoods

1.1 Analytical Expression of $p(y)$ and $p(y = 9)$

$p(y) = \int p(y|x)p(x)dx = \int \mathcal{N}(y; x, 1)\mathcal{N}(x; 0, 1) = \mathcal{N}(y; 0, 1^2 + 1^2) = \mathcal{N}(y; 0, 2) = \frac{1}{\sqrt{4\pi}} e^{-\frac{x^2}{4}}$
$p(y = 9) = \frac{1}{\sqrt{4\pi}} e^{-\frac{81}{4}} = 4.528264739771741 \times 10^{-10}$

```
def p(x, mu, sigma):
    return np.exp(-(x - mu)**2/(2*sigma**2))/(np.sqrt(2*np.pi)*sigma)

true_value = p(9, 0, np.sqrt(2))
print(true_value)
```

1.2 Implement the MC estimator to estimate $p(y)$

For fixed $p(y = 9)$,

$$p(y = 9) = \int p(y = 9|x)p(x)dx$$

The test function is

$$\varphi(x) = p(y = 9|x) = \mathcal{N}(9; x, 1)$$

The MC estimator is

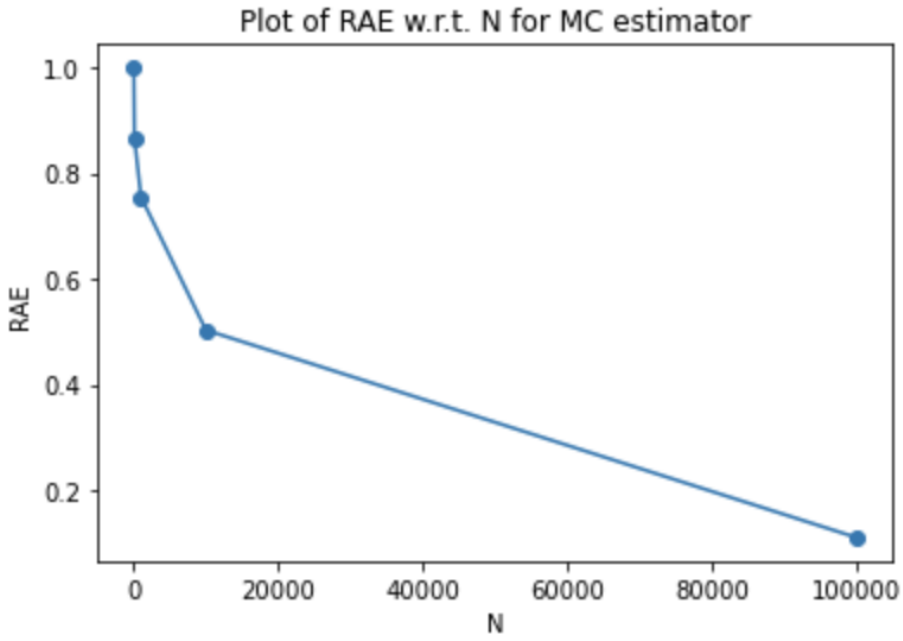
$$\hat{p}(y = 9)_{\text{MC}} = \frac{1}{N} \sum_{i=1}^{N} p(y = 9|X_i)$$

where $X_1, ..., X_N \sim p(x)$.
We calculate the relative absolute error (RAE) using the formula:

$$\text{RAE}(\hat{p}(y = 9)_{\text{MC}}) = \frac{|\hat{p}(y = 9)_{\text{MC}} - p(y = 9)|}{|p(y = 9)|}$$

N	10	100	1000	10000	100000
RAE (MC)	0.99994381	0.86733961	0.75540382	0.50288201	0.11010107

```
N_array = np.array([10, 100, 1000, 10000, 100000])
RAE_array_MC = np.array([])

# Implementation of MC estimator
for i in range(len(N_array)):
    N = N_array[i]
    x = np.random.normal(0, 1, N)  # Sample X_i ~ p(x)
    I_est_mc = (1/N) * np.sum(p(9, x, 1))  # Calculate the MC estimate
    RAE = abs(I_est_mc - true_value) / abs(true_value)  # Calculate RAE
    RAE_array_MC = np.append(RAE_array_MC, RAE)

# Plot of RAE w.r.t. N for MC estimator
plt.plot(N_array, RAE_array_MC, 'o-')
plt.xlabel('N')
plt.ylabel('RAE')
plt.title('Plot of RAE w.r.t. N for MC estimator')
```



1.3 Implement the IS estimator to estimate $p(y)$

The IS estimator is

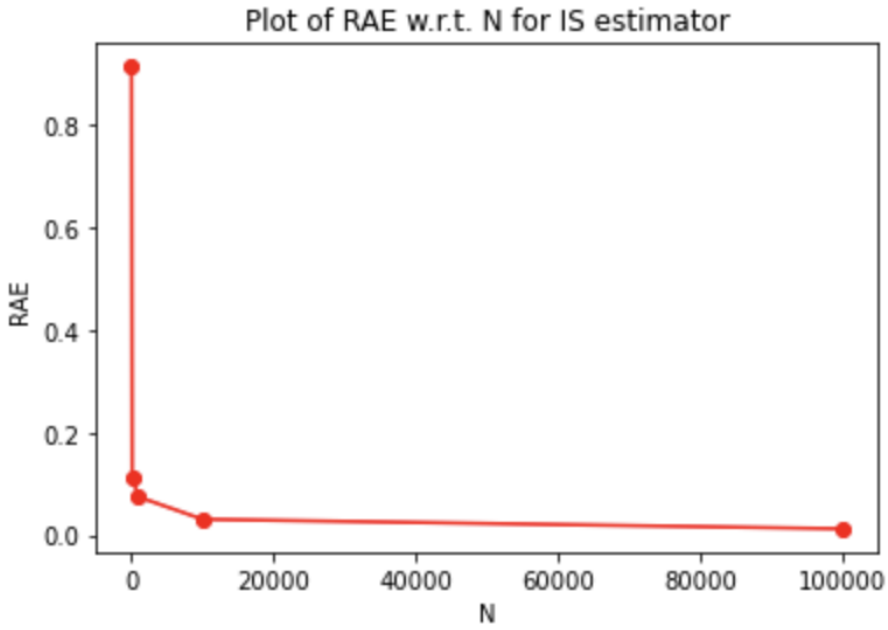
$$\hat{p}(y=9)_{IS} = \frac{1}{N}\sum_{i=1}^{N} w_i p(y=9|X_i)$$

where $X_i \sim q(x) = \mathcal{N}(x; 6, 1)$ and $w_i = \frac{p(X_i)}{q(X_i)}$ for $i = 1, ..., N$.
We calculate the relative absolute error (RAE) using the formula:

$$\text{RAE}(\hat{p}(y=9)_{\text{IS}}) = \frac{|\hat{p}(y=9)_{\text{IS}} - p(y=9)|}{|p(y=9)|}$$

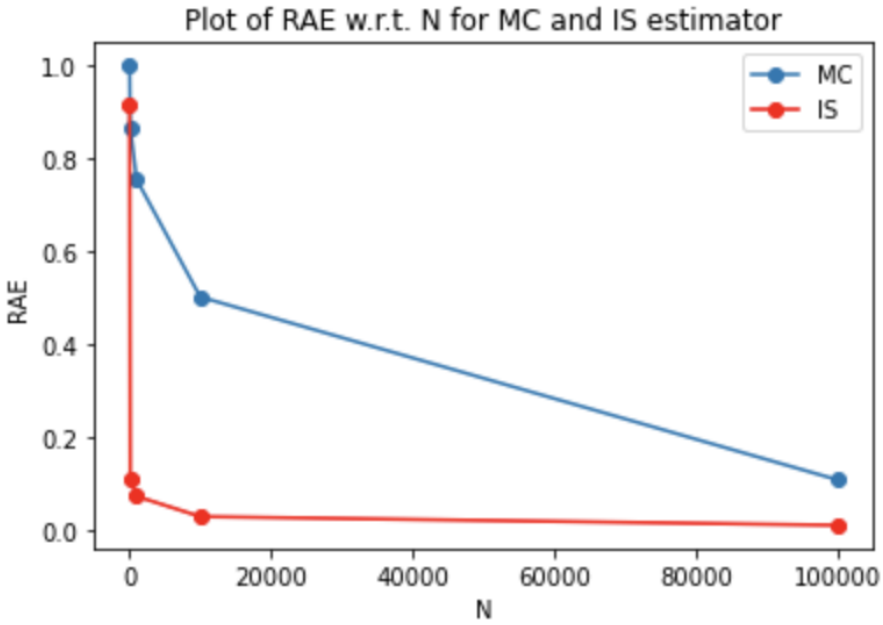
N	10	100	1000	10000	100000
RAE (IS)	0.91507674	0.11375072	0.07455063	0.03089048	0.01237603

```
def q(x):
    return np.exp(-(x - 6)**2/2)/(np.sqrt(2*np.pi))

def w(x):
    return p(x, 0, 1) / q(x)

RAE_array_IS = np.array([])

# Implementation of IS estimator
for i in range(len(N_array)):
    N = N_array[i]
    x = np.zeros(N)
    weights = np.zeros(N)
    for j in range(N):
        x[j] = np.random.normal(6, 1, 1)  # sample X_i ~ q(x)
        weights[j] = w(x[j])  # calculate importance weights
    I_est_IS = (1/N) * np.sum(weights * p(9, x, 1))  # calculate IS estimate
    RAE = abs(I_est_IS - true_value) / abs(true_value)  # calculate RAE
    RAE_array_IS = np.append(RAE_array_IS, RAE)

# Plot of RAE w.r.t. N for IS estimator
plt.plot(N_array, RAE_array_IS, 'o-', color='red')
plt.xlabel('N')
plt.ylabel('RAE')
plt.title('Plot of RAE w.r.t. N for IS estimator')
```



1.4 Compare the RAE for MC and IS

```
plt.plot(N_array, RAE_array_MC, 'o-', label='MC')
plt.plot(N_array, RAE_array_IS, 'o-', color='red', label='IS')
plt.xlabel('N')
plt.ylabel('RAE')
plt.legend()
plt.title('Plot of RAE w.r.t. N for MC and IS estimator')
```



N	10	100	1000	10000	100000
RAE (MC)	0.99994381	0.86733961	0.75540382	0.50288201	0.11010107
RAE (IS)	0.91507674	0.11375072	0.07455063	0.03089048	0.01237603

As N increases, RAE for both MC and IS decreases. The accuracy of IS estimator is higher than the accuracy of MC estimator for each N. The RAE for IS estimator converges to 0 much faster (for lower values of N) than MC estimator.

2 Q2: Metropolis-Hastings for 1D Source Localisation

2.1 Describe the MH algorithm

By Bayes theorem and given conditional independence, we have

$$p(x|y_{1:3}, s_{1:3}) = \frac{p(y_{1:3}|x, s_{1:3})p(x)}{p(y_{1:3})} = \frac{\prod_{i=1}^{3} p(y_i|x, s_i)p(x)}{p(y_{1:3})}$$

We write

$$p(x|y_{1:3}, s_{1:3}) \propto \prod_{i=1}^{3} p(y_i|x, s_i)p(x)$$

and set

$$\bar{p}(x|y_{1:3}, s_{1:3}) = \prod_{i=1}^{3} p(y_i|x, s_i)p(x)$$

which is the unnormalised posterior density.
The random walk proposal is symmetric as

$$q(x'|x) = \mathcal{N}(x'; x, \sigma_q^2) = \frac{1}{\sqrt{2\pi\sigma_q^2}} e^{-\frac{(x'-x)^2}{2\sigma_q^2}} = \frac{1}{\sqrt{2\pi\sigma_q^2}} e^{-\frac{(x-x')^2}{2\sigma_q^2}} = \mathcal{N}(x; x', \sigma_q^2) = q(x|x')$$

Therefore, the acceptance ratio is

$$r(x, x') = \frac{\bar{p}(x'|y_{1:3}, s_{1:3})q(x|x')}{\bar{p}(x|y_{1:3}, s_{1:3})q(x'|x)} = \frac{\bar{p}(x'|y_{1:3}, s_{1:3})}{\bar{p}(x|y_{1:3}, s_{1:3})} = \frac{\prod_{i=1}^{3} p(y_i|x', s_i)p(x')}{\prod_{i=1}^{3} p(y_i|x, s_i)p(x)}$$

where $p(y_i|x, s_i) = \mathcal{N}(y_i; ||x - s_i||, \sigma_y^2)$ for i=0,1,2 and $p(x) = \mathcal{N}(x; \mu_x, \sigma_x^2)$.
To implement the MH algorithm, we first sample from the proposal

$$X' \sim q(x'|X_{n-1})$$

We then draw $U \sim \text{Unif}(0, 1)$ and check if $U <= \alpha(X_{n-1}, X')$ where

$$\alpha(X_{n-1}, X') = \min\{1, r(X_{n-1}, X')\}$$

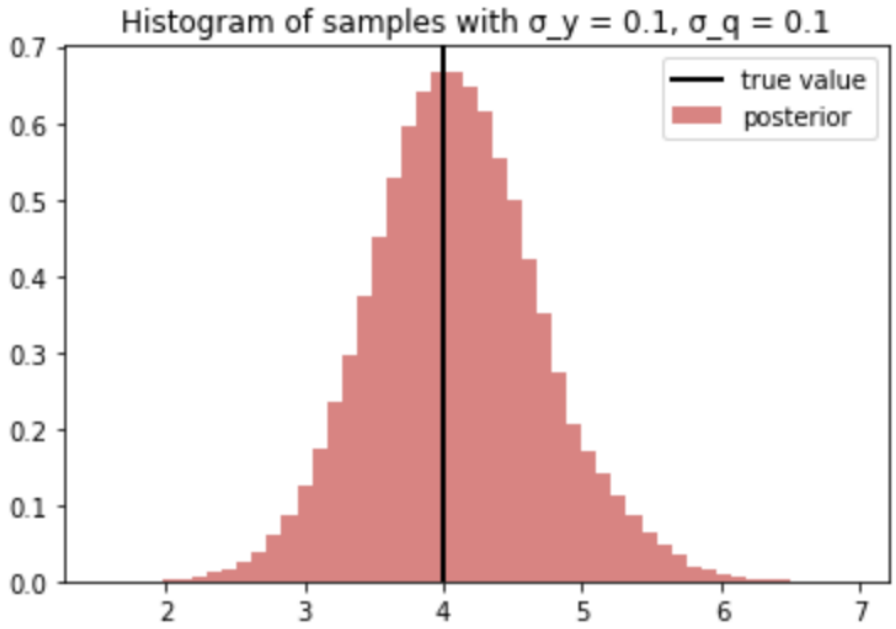
If $U <= \alpha(X_{n-1}, X')$, we accept the sample. Otherwise, we reject the sample and set $X_n = X_{n-1}$. We repeat this procedure for N times to obtain N samples. We discard the initial burnin amount of samples.

2.2 Implement the MH algorithm

```
s = np.array([-1, 2, 5])
y = np.array([4.44, 2.51, 0.73])

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

def p_bar(x, sigma_y):
    return p(y[0], abs(x - s[0]), sigma_y) * p(y[1], abs(x - s[1]), sigma_y) * p(y[2]
                                                , abs(x - s[2]), sigma_y)

# Implementation of MH algorithm
def MH(N, X_0, sigma_y, sigma_q):
    sample = np.array([X_0])
    for i in range(N):
        X_1 = np.random.normal(X_0, sigma_q, 1)  # sample X' ~ q(X_1'|X_0)
        u = np.random.uniform(0, 1)  # sample U ~  Unif(0,1)
        a = (p(X_1, 0, 10) * p_bar(X_1, sigma_y)) / (p(X_0, 0, 10) * p_bar(X_0,
                                                sigma_y))  # calculate acceptance
                                                ratio

        if u <= a:
            X_0 = X_1
        sample = np.append(sample, X_0)
    return sample
```

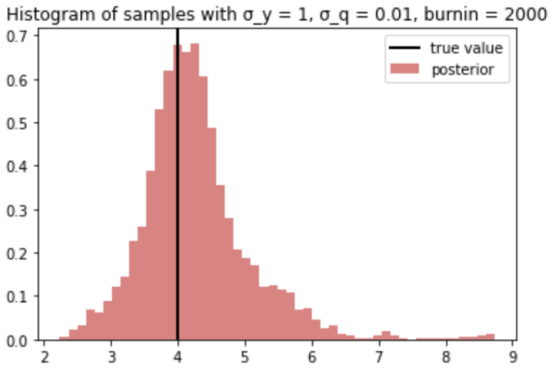
2.2.1 Implement the MH algorithm with $\sigma_y = 1, \sigma_q = 0.1$

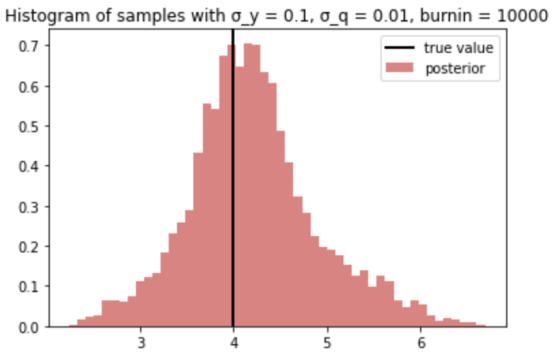
```
# sigma_y = 1, sigma_q = 0.1
sample = MH(400000, 10, 1, 0.1)
burnin = 2000
plt.clf()
plt.axvline(x = 4, color='k', label='true value', linewidth=2)
plt.hist(sample[burnin:], bins=50, density=True, label='posterior', alpha=0.5, color=
                                                [0.8, 0, 0])
plt.title('Histogram of samples with  _y  = 0.1,  _q  = 0.1')
plt.legend()
plt.show()
```



2.2.2 Implement the MH algorithm with $\sigma_y = 1, \sigma_q = 0.01$

```
# sigma_y = 1, sigma_q = 0.01
# Discard initial 2000 samples
sample = MH(400000, 10, 1, 0.01)
burnin = 2000
plt.clf()
plt.axvline(x = 4, color='k', label='true value', linewidth=2)
plt.hist(sample[burnin:], bins=50, density=True, label='posterior', alpha=0.5, color=
                                                [0.8, 0, 0])
plt.title('Histogram of samples with  _y  = 1,  _q  = 0.01, burnin = 2000')
plt.legend()
plt.show()

# Discard initial 10000 samples
burnin = 10000
plt.clf()
plt.axvline(x = 4, color='k', label='true value', linewidth=2)
plt.hist(sample[burnin:], bins=50, density=True, label='posterior', alpha=0.5, color=
                                                [0.8, 0, 0])
plt.title('Histogram of samples with  _y  = 1,  _q  = 0.01, burnin = 10000')
plt.legend()
plt.show()
```

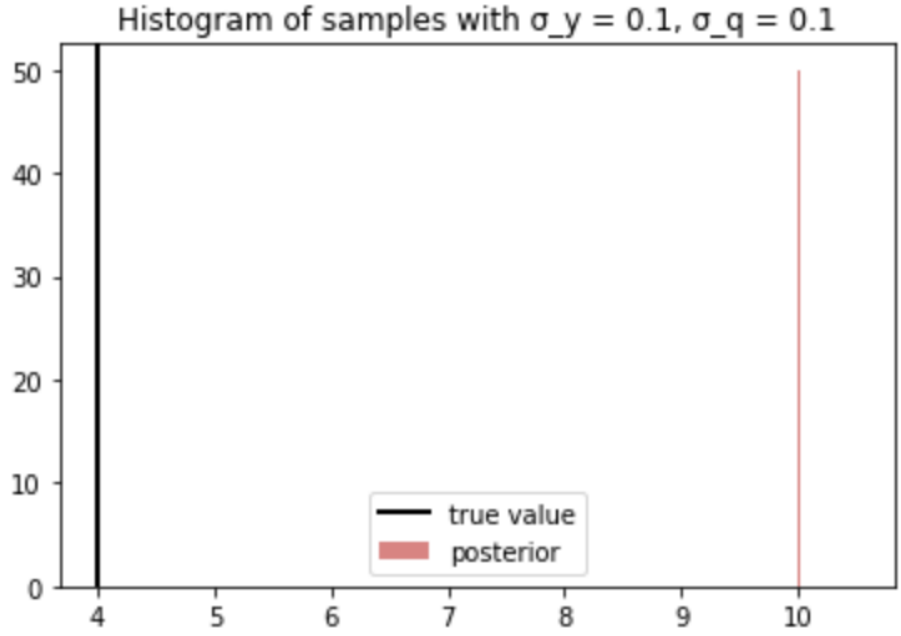




For the same number of burnin samples of 2000, the histogram of samples for $\sigma_q = 0.01$ has longer tails near the initial value of 10 as compared to the histogram of samples for $\sigma_q = 0.1$. A greater number of burnin samples is required to center the histogram at the true value of 4 as shown in the plot on the right hand side with 10000 burnin samples. Thus, the lower the value of σ_q, the greater the number of burnin samples required.

2.3 Implement the MH algorithm with $\sigma_y = 0.1, \sigma_q = 0.1$

```
# sigma_y = 0.1, sigma_q = 0.1
y = np.array([5.01, 1.97, 1.02])
sample = MH(400000, 10, 0.1, 0.1)
burnin = 2000
plt.clf()
plt.axvline(x = 4, color='k', label='true value', linewidth=2)
plt.hist(sample[burnin:], bins=50, density=True, label='posterior', alpha=0.5, color=
                                                [0.8, 0, 0])
plt.title('Histogram of samples with  _y  = 0.1,  _q  = 0.1')
plt.legend()
plt.show()
```



The code above produces "RuntimeWarning: invalid value encountered in true_divide" for the calculation of acceptance ratio. The histogram shows that the chain gets stuck at the initial value of 10 and is unable to converge to the true value of 4. This is because the variance of the likelihood is too small, the chain is unable to escape the initial value.