Imperial College London

COURSEWORK 2

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Stochastic Simulation

Author: 01844579

Date: November 29, 2022

1 Question 1

1.1 Question 1.1

Given a prior $p(x)$ and likelihood $p(y|x)$, we can compute the marginal likelihood as $p(y) = \int p(y|x)p(x)dx$. The analytical expression of $p(y)$ as derived in the lecture is $p(y) = N(y; 0, 2)$. Assuming $y = 9$ we have,

$$p(y=9) = \frac{1}{\sqrt{2}\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{9-0}{\sqrt{2}}\right)^2} = 4.52826474 \times 10^{-10}.$$

1.2 Question 1.2

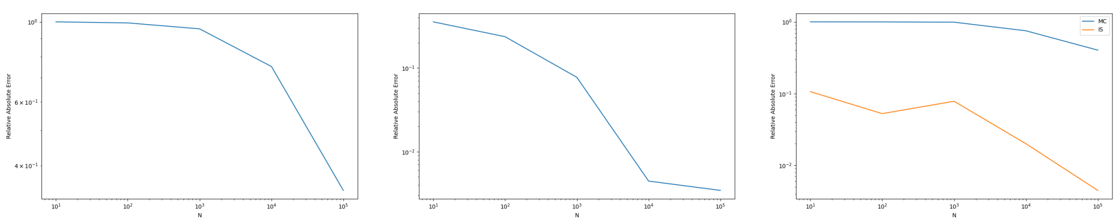
Since $p(y=9) = \int p(y=9|x)p(x)dx$, we can set our test function $\varphi(x) = p(y=9|x)$. We can then compute the integral using the MC estimation procedure as $\hat{\varphi}^N_{MC} = \hat{p}(y = 9) = \frac{1}{N}\sum_{i=1}^N p(y=9|X_i)$ where $X_i \sim p(x)$. Figure 1(a) shows the plot of RAE w.r.t N.

1.3 Question 1.3

By using the 'identity trick' and following the method in the notes we obtain $\hat{\varphi}^N_{IS} = \frac{1}{N}\sum_{i=1}^N w_i p(y=9|X_i)$ where $w_i = \frac{p(X_i)}{q(X_i)}$ and $X_i \sim q(x)$. Figure 1(b) shows the the plot of RAE w.r.t N.

1.4 Question 1.4

Figure 1(c) demonstrates the RAE of our MC and IS estimators. We can see that the RAE for our IS estimator decreases at a faster rate than the MC estimator. This implies that the IS estimator has better accuracy at computing $p(y = 9)$ than the MC estimator and does not require as many samples as the MC estimator to reach an acceptable estimate.



(a) MC Estimator (b) IS Estimator (c) Comparison of Estimators

Figure 1: Relative Absolute Errors

2 Question 2

2.1 Question 2.1

The set-up is similar to the Source Localisation example in the lectures. The model can be interpreted as having three sensors with three noisy observations coming from the likelihood, $p(y_i|x,s_i) = N(y_i;||x-s_i||,\sigma_y^2)$ trying to locate an object, where $s_i \in \mathbb{R}$ is the location of the ith sensor. Our aim is to locate this object so we set a prior, $p(x) = N(x;\mu_x,\sigma_x^2)$ and assume independence of the observations and that the noise is independent of the object's location. So we are interested in the posterior density of x, $p(x|y_i,s_i)$, the distribution over the location of the object. In order to implement the MH algorithm we choose a symmetric random walk proposal, $q(x'|x) = N(x';x,\sigma_q^2)$ and run our algorithm. Noticing that our proposal is symmetric, $r(x,x')$ simplifies to,

$$
\begin{aligned}
r(x,x') &= \frac{p(x')p(y_0|x',-1)p(y_1|x',2)p(y_2|x',5)}{p(x)p(y_0|x,-1)p(y_1|x,2)p(y_2|x,5)} \\
&= \frac{p(x')}{p(x)}\prod_{i=0}^{2}\frac{p(y_i|x',s_i)}{p(y_i|x,s_i)} \\
&= \frac{\frac{1}{\sigma_x\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x'-\mu_x}{\sigma_x}\right)^2}}{\frac{1}{\sigma_x\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)^2}} \cdot \prod_{i=0}^{2}\frac{\frac{1}{\sigma_y\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{y_i-||x'-s_i||}{\sigma_y}\right)^2}}{\frac{1}{\sigma_y\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{y_i-||x-s_i||}{\sigma_y}\right)^2}} \\
&= e^{-\frac{1}{2\sigma_x{}^2}\left[(x'-\mu_x)^2-(x-\mu_x)^2\right]} \cdot \prod_{i=0}^{2} e^{-\frac{1}{2\sigma_y{}^2}\left[(y_i-||x'-s_i||)^2-(y_i-||x-s_i||)^2\right]}.
\end{aligned}
$$

The Metropolis-Hastings algorithm works by defining transition kernels defined via the algorithm so that the stationary distribution is our target distribution. We have a local proposal $q(x'|x)$ which we sample X' from, that we accept with an acceptance probability and set $X_n = X'$. The acceptance probability, defined as $\alpha(X_{n-1}|X') = \min\{1, r(x,x')\}$, is designed so that our samples form a Markov chain that leaves p_* invariant. However if a sample is rejected we do not sample again and set $X_n = X_{n-1}$ Once we finish taking our samples, we discard the first *burn-in* samples and return the rest of our samples.

2.2 Question 2.2

We see in figure 3(a) that our histogram seems to resemble a normal distribution albeit not exactly centered at the true value. On the other hand, figure 3(b) can be said to very loosely resemble a normal distribution or possess a 'bell' shape. Figure 2 demonstrates realisations of our algorithm and by analysing the convergence of the realisations to $x_{true} = 4$ we can appropriately set our burn-in iterations. We see that for $\sigma_q = 0.1$ we have quite fast 'convergence' towards $x_{true} = 4$ and so we could set the burn-in level to 10000. However for $\sigma_q = 0.01$ we see that convergence is slower

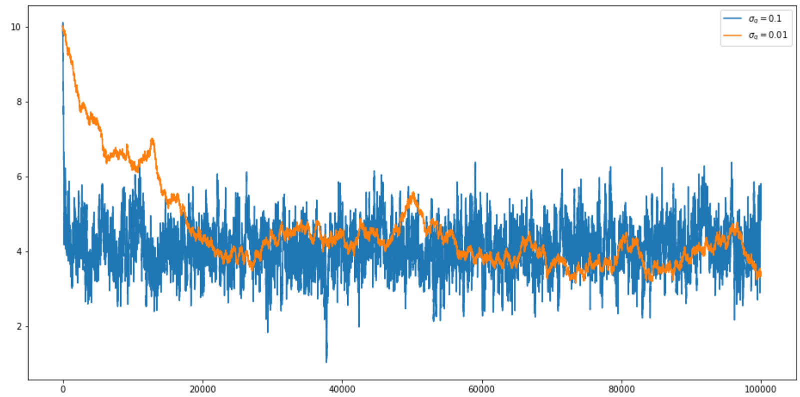
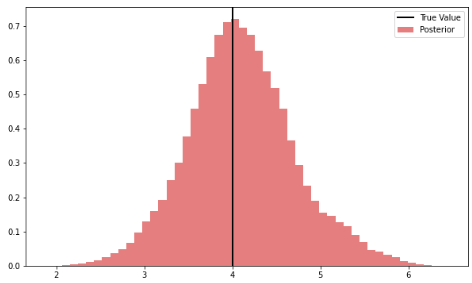
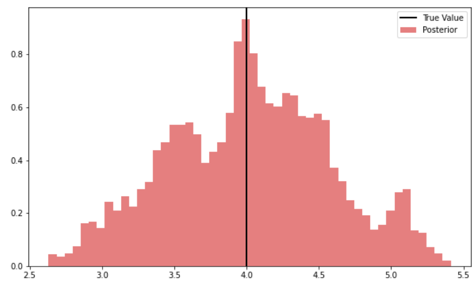


Figure 2: A realisation of our Markov chain



(a) $\sigma_q = 0.1$



(b) $\sigma_q = 0.01$

Figure 3: Histogram of posteriors

and that an appropriate burn-in level would be around 60000. Our results suggest that there's a variance/burn-in trade-off, that is by decreasing the variance we have to increase our burn-in iterations due to some of the initial values not reaching the stationary distribution, therefore the histogram obtains readings far from x_{true}.

2.3 Question 2.3

Figure 4 shows the histogram for our new values of y and σ_y. We notice that the histogram resembles a normal distribution however not centered at the true value, in-fact it appears to be right-skewed. A suggestion for this result could be due to the change of our variance or due to the new location of our sensors.

A Code for Question 1.2

```
def normalPDF(x, mu, sigma): # function for probability density function of normal distribution
```

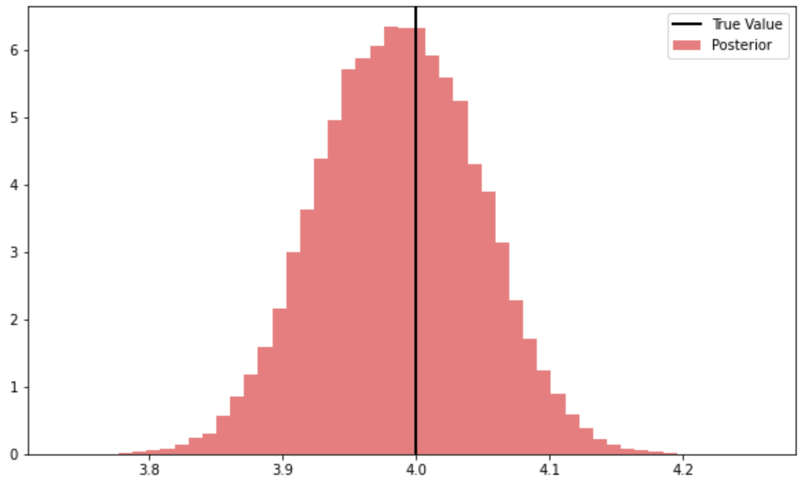


Figure 4: Histogram of posterior

```
    return 1/(np.sqrt(2*np.pi)*sigma) * np.exp(-1/2 * ((x - mu)/sigma
    )**2)

N = [10, 100, 1000, 10000, 100000] # set values of N for RAE

def MC(n, y): # function for MC estimator

    samples = [] # empty list to append samples

    for i in range(n): # for loop over N

        x = np.random.normal(0,1) # iid samples from p(x)
        p = normalPDF(y, x, 1) # sample from p(y|x)
        samples = np.append(samples, p) # append samples

    return 1/n * sum(samples) # return MC estimate

a = [abs(MC(i,9) - normalPDF(9, 0, np.sqrt(2)))/normalPDF(9, 0, np.
    sqrt(2)) for i in N] # comprehension to compute RAE

# code to plot RAE vs N
plt.figure(figsize=(10,6), dpi = 100)
plt.xlabel('N')
plt.ylabel('Relative Absolute Error')
plt.plot(N, a)
```

B Code for Question 1.3

```
def IS(n, y): # function for  importance sampling estimator

```

```
    samples = [] # empty list to append samples

    for i in range(n): # for loop over N

        x = np.random.normal(0,1) # iid samples from p(x)
        p = normalPDF(y, x, 1) # sample from p(y|x)
        w = normalPDF(x, 0, 1)/normalPDF(x, 6, 1) # computing weights
        samples = np.append(samples, p * w) # append samples

    return 1/n * sum(samples) # return IS estimate

b = [abs(IS(i,9) - normalPDF(9, 0, np.sqrt(2)))/normalPDF(9, 0, np.sqrt(2)) for i in N] # comprehension to compute RAE

# code to plot RAE vs N
plt.figure(figsize=(10,6), dpi = 100)
plt.xlabel('N')
plt.ylabel('Relative Absolute Error')
plt.plot(N, b)
```

C Code for Question 1.4

```

STILL NEED TO DO
plt.figure(figsize=(10,6), dpi = 100)
plt.xlabel('N')
plt.ylabel('Relative Absolute Error')

plt.loglog(N, a, label = 'MC')
plt.loglog(N, b, label = 'IS')
plt.legend()
```

D Code for Question 2.2

```
def acceptance(y, s, x, xprime, xsigma, ysigma, xmu): # function to compute acceptance ratio

    product = 1

    for i in range(len(y)):

        product *= np.exp((-1/(2*ysigma**2)) * ((y[i] - abs(xprime - s[i]))**2 - (y[i] - abs(x - s[i]))**2))

    return np.exp((-1/(2 * xsigma**2)) * (xprime**2 - x**2)) * product

# setting parameters
y = [4.44, 2.51, 0.73]
s = [-1, 2, 5]
```

```python

ysigma = 1
xmu, xsigma = 0, 10
qsigma1, qsigma2 = 0.1, 0.01

x0 = 10
xtrue = 4

N = 100000
burnin1, burnin2 = 10000, 60000

def MHAlgo(x0, qsigma, N, y, s, xsigma, ysigma, xmu): # MH algorithm

    samples = [] # empty list to append samples
    index = [] # empty list to append index of accepted samples

    for i in range(N): # for loop over N

        xprime = np.random.normal(x0, qsigma) # sample from q(x'|x_{n-1})
        u = np.random.uniform(0,1) # generate probability to accept

        if u < min(1, acceptance(y, s, x0, xprime, xsigma, ysigma, xmu)): # if statement to determine acceptance

            x0 = xprime # accepted so set x' = x_n
            samples = np.append(samples, xprime) # append new sample
            index = np.append(index, [i]) # track index of accepted sample to help choose burn-in value

        else:
            x0 = x0 # reject sample and set x_n = x_{n-1}

    return samples, index # return our samples, index

X, nx = MHAlgo(x0, qsigma1, N, y, s, xsigma, ysigma, xmu) # input parameters to obtain samples

Y, ny = MHAlgo(x0, qsigma2, N, y, s, xsigma, ysigma, xmu) # input parameters to obtain samples

# plot of a realisation to determine appropiate burn-in value
plt.figure(figsize=(16,8))
plt.plot(nx, X, label = '$\sigma_q = 0.1$')
plt.plot(ny, Y, label = '$\sigma_q = 0.01$')
plt.legend()
plt.show()

# code from coursework to plot posterior
plt.clf()
plt.axvline(xtrue, color='k', label='True Value', linewidth=2)
plt.hist(X[burnin1:N], bins=50 , density=True , label='Posterior',
    alpha=0.5, color=[0.8, 0, 0])
plt.legend()
plt.show()
```

```

plt.clf()
plt.axvline(xtrue, color='k', label='True Value', linewidth=2)
plt.hist(Y[burnin2:N], bins=50 , density=True , label='Posterior',
    alpha=0.5, color=[0.8, 0, 0])
plt.legend()
plt.show()
```

E Code for Question 2.3

```
# setting new parameters
ysigma2 = 0.1
y1 = [5.01, 1.97, 1.02]

Z, nz = MHAlgo(x0, qsigma1, N, y1, s, xsigma, ysigma2, xmu) ## input
    parameters to obtain samples

# code from coursework to plot posterior
plt.clf()
plt.axvline(xtrue, color='k', label='True Value', linewidth=2)
plt.hist(Z[burnin1:N], bins=50 , density=True , label='Posterior',
    alpha=0.5, color=[0.8, 0, 0])
plt.legend()
plt.show()
```