Exercises 4

Monte Carlo inference

4.1 Monte Carlo preliminaries

Exercise 4.1 Random number generation
Familiarize yourself with the random number generator on your computer.

a) Draw $L = 100$ independent normal random numbers (mean 0, variance 1) $x_1, x_2, \dots, x_{300}$ and plot them in a histogram together with the pdf for the normal distribution. Does your random number generator appear to be working OK?

b) Use the samples to estimate the mean (true value: 0) and variance (true value: 1) of the distribution. What happens with the quality of your estimates if you increase or decrease L?

4.2 Importance sampling

Exercise 4.2 A first importance sampler
Assume that you want samples from a Johnson's S_U distribution with pdf $p(x) = \frac{\sqrt{2}}{\sqrt{\pi\left(1+(x-1)^2\right)}} e^{-\frac{1}{2}\left(3+2\sinh^{-1}(x-1)\right)^2}$. In Python (with numpy imported as `np`), this pdf can be written as

```
np.sqrt(2)/np.sqrt(np.pi*(1+(x-1)**2))*np.exp(-.5*(3+2*np.arcsinh(x-1))**2)
```

a) In order to generate (weighted) samples from $p(x)$, implement an importance sampler with a standard normal distribution (mean = 0, variance = 1) as proposal. Draw $L = 1\,000$ samples.

b) To inspect your obtained Monte Carlo approximation, plot a histogram of the samples (note that you have to take the weights into account!) together with the target pdf and the proposal pdf. What can you say about the quality of the approximation you have obtained? How does it relate to the shape of the proposal?

c) Try using more samples L. Does the Monte Carlo approximation improve?

d) Limit yourself to $L = 1\,000$ samples again, and try to instead adjust the proposal to improve the approximation.

e) Use the samples to estimate the mean and variance of the target. (One can analytically show that the mean is -1.41 and the variance 1.98). Explore how the quality of these estimates changes with different L.

f) Change the proposal to a uniform distribution on the interval $[-4, 1]$. Is it still a valid importance sampler?

Exercise 4.3 Baysian linear regression as a Bayesian network: importance sampling
Bayesian linear regression with scalar x_i and y_i

$$y_i = wx_i + e_i, \quad e_i \sim \mathcal{N}(0, \sigma^2)$$

with unknown w (scalar) *and* σ^2 (scalar) can be formulated using the Normal-Inverse-Gamma distribution,

$$p(\sigma^2, w) = \begin{cases} p(\sigma_2) = \mathcal{IG}(\sigma^2; a, b) = \frac{b^a}{\Gamma(a)}(\sigma^2)^{-a-1}\exp\left(-\frac{b}{\sigma^2}\right) \\ p(w \mid \sigma^2) = \mathcal{N}(w; m, \sigma^2/\ell) = \frac{1}{\sqrt{(2\pi\sigma^2/\ell)}}\exp\left(-\frac{(x-m)^2}{\sigma^2/\ell}\right) \end{cases} \tag{20}$$
$$p(y \mid \sigma^2, w) = \mathcal{N}(y; wx, \sigma^2)$$

The corresponding Bayesian network is shown in Fig 4.1. Throughout the problem, use the hyperparameters $a = 2, b = 3, m = 0, \ell = 1$. Our goal is to infer the posterior $p(\sigma^2, w \mid y)$.

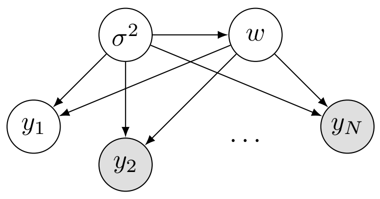


Figure 4.1: Bayesian networks in Exercise 4.5

a) Generate some data using the following code:

```python
import numpy as np

N = 100
X = np.random.normal(size=N)
y = -3*X + np.random.normal(size=N)*2
```

What are the true w and σ^2?

b) Design an importance sampler using the prior (20) as proposal.

Hint:
To draw L samples of σ^2 from $p(\sigma^2)$, import `invgamma` from `scipy.stats` and call `invgamma.rvs(size=L,a=a,scale=b)`. Furthermore, to draw L samples of w from the prior $p(w \mid \sigma^2)$, call `np.random.normal(size=L)*np.sqrt(s2/l)+m`, where `s2` is a vector of the σ^2-samples. Finally, to evaluate $p(y_{1:N} \mid \sigma^2, w)$, you may call

```
1/np.sqrt(2*np.pi*s2)**N*np.exp(-1/(2*s2)*np.sum((X[:,np.newaxis]*w[np.newaxis,:]-y[:,np.newaxis])**2,axis=0))
```

where `w` is the L-dimensional vector with samples of w.

c) Plot histograms of the marginal prior and posterior distribution for σ^2 and w, respectively. How do they change as the number of data points N changes?

Hint:

This problem actually has an analytical solution,

$$
\begin{aligned}
p(\sigma^2 \,|\, y_{1:N}) &= \mathcal{IG}(sigma^2; \bar{a}, \bar{b}), \\
p(w \,|\, y_{1:N}) &= \mathcal{T}_{2\bar{a}}\left(\sqrt{\frac{\bar{\ell}\bar{a}}{\bar{b}}}(w - \bar{m})\right) \\
\bar{a} &= a + \frac{N}{2}, \\
\bar{b} &= b + \frac{1}{2}(\sum_{i=1}^{N} y_i^2 + m^2\ell - \frac{(m\ell + \sum_{i=1}^{N} c_i y_i)^2}{\ell + \sum_{i=1}^{N} x_i^2)} \\
\bar{m} &= \frac{m\ell + \sum_{i=1}^{N} c_i y_i}{\ell + \sum_{i=1}^{N} x_i^2)} \\
\bar{\ell} &= \ell + \sum_{i=1}^{N} x_i^2
\end{aligned}
$$

This analytical solutions is implemented by the following code:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import invgamma, t

# define data and hyperparameters here!

mpost = (m*l+np.sum(X*y))/(l+np.sum(X**2))
lpost = l + np.sum(X**2)
apost = a + N/2
bpost = b+.5*(np.sum(y**2)+m**2*l-(m*l+np.sum(X*y))**2/(l+np.sum(X**2)))

s2v = np.linspace(0,10,1000)
plt.plot(s2v,invgamma.pdf(x=s2v,a=apost,scale=bpost))
plt.show()

wv = np.linspace(-5,5,1000)
plt.plot(wv,t.pdf(x=wv,df=2*apost,loc=mpost,scale=np.sqrt(bpost/(lpost*apost))))
plt.show()
```

4.3 Gibbs sampling

Exercise 4.4 A first Markov chain

A more complicated way than Problem 4.1 to sample from a standard normal distribution is to construct a Markov chain whose stationary distribution is a standard normal distribution. Such an example is

$$x[k+1] = 0.9x[k] + v[k], \quad v[k] \sim \mathcal{N}(0, 0.19). \tag{21}$$

a) Eq. (21) is a Markov chain. Why?

b) Implement (21). Start in $x[1] = 5$ and simulate it for $k = 2, 3, \ldots, 300$. Since this is a Markov chain that is simulated, it is a Markov chain Monte Carlo (MCMC) method. Plot a histogram of x and compare it to the pdf of a standard normal distribution.

c) Plot your obtained $x[1], x[2], \ldots, x[300]$. How long appears your *burn-in* period to be, for this example?

d) Prove that the stationary distribution of (21) is $\mathcal{N}(0, 1)$.

Exercise 4.5 Gibbs sampler for a directed graph with discrete variables
Consider the Bayesian network in Figure 4.2 with binary random variables A, B, and

$$\begin{aligned} &p(A) = 0.1 \quad \left(\Rightarrow p(\bar{A}) = 0.9\right), \\ &p(B \mid A) = 0.7, \\ &p(B \mid \bar{A}) = 0.1, \end{aligned}$$

Implement the following Gibbs sampler for $p(A, B)$:

1. Sample $A[k] \sim p(A \mid B[k-1])$
2. Sample $B[k] \sim p(B \mid A[k])$

Draw $K = 300$ samples and plot the trace as well as the obtained empirical distribution for $p(A)$ and $p(B)$, respectively.

$$\begin{tikzcd} \boxed{A} \arrow[r] & \boxed{B} \end{tikzcd}$$

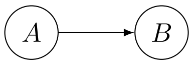


Figure 4.2: Bayesian networks in Exercise 4.5

Exercise 4.6 Baysian linear regression as a Bayesian network: Gibbs sampling
Now consider Problem 4.3 again, but this time using a Gibbs sampler. Plot histograms and compare to your previous solution.

Hints:

- $p(\sigma^2 \mid y_{1:N}, w)$ is an inverse Gamma distribution $\mathcal{IG}(\tilde{a}, \tilde{b})$ with $\tilde{a} = a + N/2 + 1/2$ and $\tilde{b} = b + \frac{1}{2}\sum_{i=1}^{N}(y_i - wx_i)^2 + \frac{\ell}{2}(w - m)^2$. To draw a sample from that distribution, you may use

```
invgamma.rvs(a=a+N/2+1/2,scale=b+1/2*np.sum((X*w-y)**2)+l/2*(w-m)**2)
```

- $p(w \mid \sigma^2, y_{1:N})$ is a normal distribution $\mathcal{N}(\frac{m\ell + \sum_{i=1}^{N} x_i y_i}{\ell + \sum_{i=1}^{N} x_i^2}, \frac{\sigma^2}{\ell + \sum_{i=1}^{N} x_i^2})$. To draw a sample from that distribution, you may use

```
np.random.normal()*np.sqrt(s2/(l+np.sum(X**2)))+(m*l+np.sum(X*y))/(l+np.sum(X**2))
```

Solutions 4

Monte Carlo inference

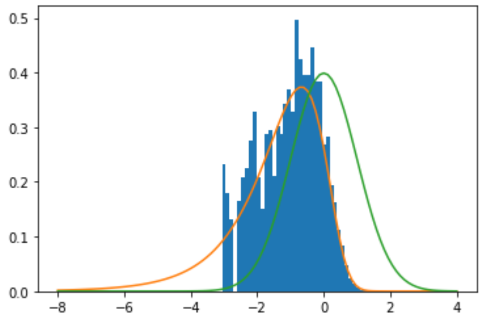
Solution to Exercise 4.1

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

L = 300
x = np.random.normal(size=L)

xv = np.linspace(-4,4,100)
plt.hist(x,bins=10,density=True)
plt.plot(xv,1/np.sqrt(2*np.pi)*np.exp(-.5*xv**2))
plt.show()

np.mean(x)
np.var(x)
```

Solution to Exercise 4.2

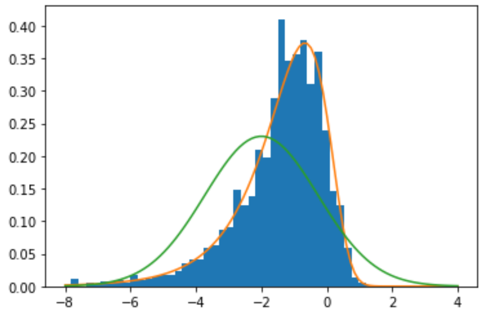
```python
import numpy as np
import matplotlib.pyplot as plt

# target
def p(x):
    return np.sqrt(2)/np.sqrt(np.pi*(1+(x-1)**2))*np.exp(-.5*(3+2*np.arcsinh(x-1))**2)

# proposal
m = 0
s2 = 1
def q(x):
    return 1/np.sqrt(2*np.pi*s2)*np.exp(-((x-m)**2)/2/s2)

L = 1000 # number of samples
x = np.random.normal(size=L)*np.sqrt(s2)+m #draw from proposal
w = p(x)/q(x) # compute weights

# plot a weighted histogram
plt.hist(x,weights=w,bins=50,density=True)

# plot the target distribution and the proposal
xv = np.linspace(-8,4,100)
plt.plot(xv,p(xv))
plt.plot(xv,q(xv))

# Estimate mean and variance
est_mean = np.sum(x*w)/L
est_var = np.sum(w*(est_mean-x)**2)/L
```

b) The approximation is better in areas where the proposal has about the same density, or higher, than the target (for x larger than -0.5 or so), and poorer in areas where the target has much higher density than the proposal (for x smaller than 0.5 or so).

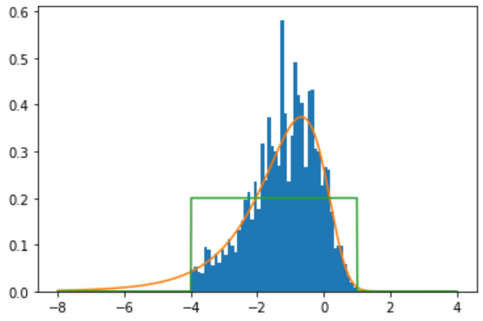


c) Using more samples improves the approximation, but requires (obviously) more computational effort.

d) With, for example, a proposal with mean -2 and variance 4, the approximation appears better.



e) No, this is not a valid importance sampler, since the proposal does not have support $q(x) > 0$ for all x where the target has support $p(x) > 0$ (such as for $x < -4$).



Solution to Exercise 4.3

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import invgamma, t

m = 0
l = 1
b = 3
a = 2

N = 100
w0 = -3
X = np.random.normal(size=N)
y = X*w0 + np.random.normal(size=N)*2

# Analytical solution
s2v = np.linspace(0,10,1000)
wv = np.linspace(-5,5,1000)
mpost = (m*l+np.sum(X*y))/(l+np.sum(X**2))
lpost = l + np.sum(X**2)
apost = a + N/2
bpost = b+.5*(np.sum(y**2)+m**2*l-(m*l+np.sum(X*y))**2/(l+np.sum(X**2)))


# Importance sampling
L = 10000

s2is = invgamma.rvs(size=L,a=a,scale=b)
plt.hist(s2is,density=True,bins=np.linspace(0,10,50))
plt.plot(s2v,invgamma.pdf(x=s2v,a=a,scale=b))
plt.xlim((0,10))
plt.title('Prior')
plt.show()

wis = np.random.normal(size=L)*np.sqrt(s2is/l)+m
plt.hist(wis,density=True,bins=np.linspace(-5,5,50))
plt.plot(wv,t.pdf(x=wv,df=2*a,loc=m,scale=np.sqrt(b/(a*l))))
plt.xlim((-5,5))
plt.title('Prior')
plt.show()

weightis = 1/np.sqrt(2*np.pi*s2is)**N*np.exp(-1/(2*s2is)*np.sum((X[:,np.newaxis]*wis[np.newaxis,:]-y
    [:,np.newaxis])**2,axis=0))

plt.hist(s2is,density=True,bins=np.linspace(0,10,50),weights=weightis)
plt.plot(s2v,invgamma.pdf(x=s2v,a=apost,scale=bpost))
plt.xlim((0,10))
plt.title('Posterior')
plt.show()

plt.hist(wis,density=True,bins=np.linspace(-5,5,50),weights=weightis)
plt.plot(wv,t.pdf(x=wv,df=2*apost,loc=mpost,scale=np.sqrt(bpost/(lpost*apost))))
plt.xlim((-5,5))
plt.title('Posterior')
plt.show()
```

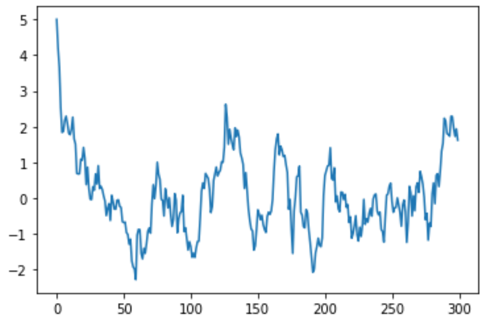
Solution to Exercise 4.4

a) Since $p(x[k+1]) \mid x[k], x[k-1], \ldots, x[1]) = p(x[k+1]) \mid x[k])$.

b)-c)

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

K = 300
x = np.zeros(K)
x[0] = 5
for k in range(K-1):
    x[k+1] = 0.9*x[k] + np.random.normal()*np.sqrt(0.19)

xv = np.linspace(-4,4,100)
plt.hist(x,bins=10,density=True)
plt.plot(xv,1/np.sqrt(2*np.pi)*np.exp(-.5*xv**2))
plt.show()

np.mean(x)
np.var(x)

plt.plot(x)
plt.show()
```

d) In the example below, the burn-in period (the time it takes for the chain to 'forget' the initial state and reach its stationary distribution) appears to be something like 25 iterations.



e) In stationarity, the mean and variance of $x[k]$ and $x[k+1]$ are (by definition of stationarity) the same. Hence,

$$\mathbb{E}[x[k]] = \underbrace{\mathbb{E}[0.9x[k]]}_{0.9\mathbb{E}[x[k]]} + \underbrace{\mathbb{E}[v[k]]}_{0} \Rightarrow \mathbb{E}[x[k]] = 0,$$

$$\text{Var}[x[k]] = \underbrace{\text{Var}[0.9x[k]]}_{0.9^2\text{Var}[x[k]]} + \underbrace{\text{Var}[v[k]]}_{0.19} \Rightarrow \text{Var}[x[k]] = \frac{0.19}{1-0.9^2} = 1.$$

Solution to Exercise 4.5

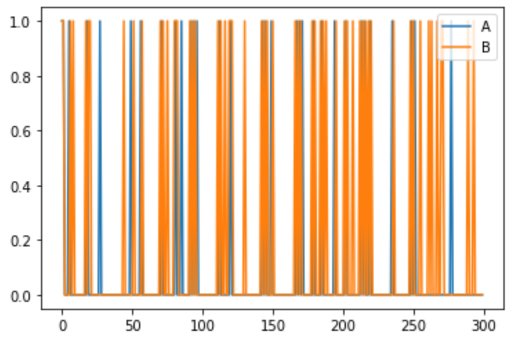
The conditional distribution $p(B \mid A)$ is given in the problem, but we also have to find the conditional distribution

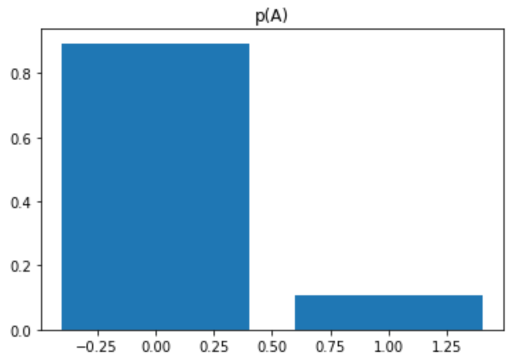
$$\begin{aligned} p(A \mid B) &= \frac{p(B \mid A)p(A)}{p(B)} = \frac{0.7 \cdot 0.1}{p(B)} \\ p(\bar{A} \mid B) &= \frac{p(B \mid \bar{A})p(\bar{A})}{p(B)} = \frac{0.1 \cdot 0.9}{p(B)} \\ &\Rightarrow \Big/ p(A \mid B) + p(\bar{A} \mid B) = 1 \Big/ \Rightarrow p(B) = 0.07 + 0.09 = 0.16, \end{aligned}$$

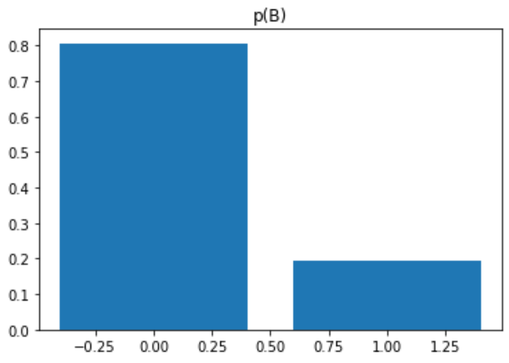
and similarly

$$p(A \mid \bar{B}) = \frac{0.3 \cdot 0.1}{p(\bar{B})}, \quad p(\bar{A} \mid \bar{B}) = \frac{0.9 \cdot 0.9}{p(\bar{B})} \quad \text{with } p(\bar{B}) = 0.84.$$

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

K = 300

A = np.zeros(K)
B = np.zeros(K)
A[0] = 1 # user-chosen initialization
B[0] = 1 # user-chosen initialization

for k in range(K-1):
    # Sample A given B
    if B[k]==0:
        A[k+1] = np.random.binomial(1,0.03/0.84)
    else:
        A[k+1] = np.random.binomial(1,0.07/0.16)

    # Sample B given A
    if A[k]==0:
        B[k+1] = np.random.binomial(1,0.1)
    else:
        B[k+1] = np.random.binomial(1,0.7)

plt.plot(A,label='A')
plt.plot(B,label='B')
plt.legend(loc='upper right')
plt.show()

plt.bar(x=(0,1),height=(sum(1-A)/K,sum(A)/K))
plt.title('p(A)')
plt.show()

plt.bar(x=(0,1),height=(sum(1-B)/K,sum(B)/K))
plt.title('p(B)')
plt.show()
```







Solution to Exercise 4.6

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import invgamma, t

m = 0
l = 1
b = 3
a = 2

N = 100
w0 = -3
X = np.random.normal(size=N)
y = X*w0 + np.random.normal(size=N)*2

# Analytical solution
s2v = np.linspace(0,10,1000)
wv = np.linspace(-5,5,1000)
mpost = (m*l+np.sum(X*y))/(l+np.sum(X**2))
lpost = l + np.sum(X**2)
apost = a + N/2
bpost = b+.5*(np.sum(y**2)+m**2*l-(m*l+np.sum(X*y))**2/(l+np.sum(X**2)))

# Gibbs sampling
K = 10000
s2g = np.zeros(K)
wg = np.zeros(K)
wg[0] = 0
s2g[0] = 1
for k in range(K-1):
    wg[k+1] = np.random.normal()*np.sqrt(s2g[k]/(l+np.sum(X**2)))+(m*l+np.sum(X*y))/(l+np.sum(X**2))
    s2g[k+1] = invgamma.rvs(a=a+N/2+1/2,scale=b+1/2*np.sum((X*wg[k+1]-y)**2)+l/2*(wg[k+1]-m)**2)


plt.hist(s2g,density=True,bins=np.linspace(0,10,50))
plt.plot(s2v,invgamma.pdf(x=s2v,a=apost,scale=bpost))
plt.xlim((0,10))
plt.title('Posterior')
plt.show()

plt.hist(wg,density=True,bins=np.linspace(-5,5,50))
plt.plot(wv,t.pdf(x=wv,df=2*apost,loc=mpost,scale=np.sqrt(bpost/(lpost*apost))))
plt.xlim((-5,5))
plt.title('Posterior')
plt.show()
```

Bibliography

[1] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.