Imperial College London

COURSEWORK 1

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Stochastic Simulation

Author:
Richard Fu (CID: 01852979)

Date: November 5, 2022

1 QUESTION 1

1.1 Compute M_λ by finding optimal x^*

$$M_\lambda = \sup_x \frac{p_\nu(x)}{q_\lambda(x)}. \tag{1}$$

First we evaluate:

$$\begin{aligned}\frac{p_\nu(x)}{q_\lambda(x)} &= \frac{1}{2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)} x^{\frac{\nu}{2}-1}\exp\left(-\frac{x}{2}\right)\cdot\frac{1}{\lambda\exp(-\lambda x)},\\ &= \frac{1}{\lambda 2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)} x^{\frac{\nu}{2}-1}\exp\left(\left(\lambda-\frac{1}{2}\right)x\right).\end{aligned} \tag{2}$$

Since log is an increasing function, we can take the log to help compute x^*:

$$\begin{aligned}\log\left(\frac{p_\nu(x)}{q_\lambda(x)}\right) &= \log\left(\frac{1}{\lambda 2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)} x^{\frac{\nu}{2}-1}\exp\left(\left(\lambda-\frac{1}{2}\right)x\right)\right),\\ &= -\log\left(\lambda 2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)\right)+\left(\frac{\nu}{2}-1\right)\log(x)+\left(\lambda-\frac{1}{2}\right)x.\end{aligned} \tag{3}$$

Differentiating with respect to x and setting to 0:

$$\frac{d\log\left(\frac{p_\nu(x)}{q_\lambda(x)}\right)}{dx} = \left(\frac{\nu}{2}-1\right)\frac{1}{x}+\left(\lambda-\frac{1}{2}\right), \tag{4}$$

$$\frac{d\log\left(\frac{p_\nu(x)}{q_\lambda(x)}\right)}{dx} = 0 \implies \nu-2 = x(1-2\lambda), \tag{5}$$

$$\implies x^* = \frac{\nu-2}{1-2\lambda}. \tag{6}$$

Verify x^* is a maximiser by checking second derivative with respect to x:

$$\frac{d^2\log\left(\frac{p_\nu(x)}{q_\lambda(x)}\right)}{dx^2} = \left(1-\frac{\nu}{2}\right)\frac{1}{x^2}, \tag{7}$$

$$\left.\frac{d^2\log\left(\frac{p_\nu(x)}{q_\lambda(x)}\right)}{dx^2}\right|_{x=x^*} = \frac{(1-2\lambda)^2}{2(2-\nu)} < 0, \text{ since } 0<\lambda<\frac{1}{2} \text{ and } \nu>2. \tag{8}$$

We have shown x^* is indeed a maximiser. Compute M_λ by substituting $x = x^*$ into (2):

$$M_\lambda = \frac{p_\nu(x^*)}{q_\lambda(x^*)} = \frac{1}{\lambda 2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)}\left(\frac{1-2\lambda}{\nu-2}\right)^{1-\frac{\nu}{2}}\exp\left(1-\frac{\nu}{2}\right). \tag{9}$$

1.2 Find optimal λ^* in terms of ν

In order to optimise the acceptance rate $\hat{a} = \frac{1}{M_\lambda}$, we need to minimize M_λ over λ. Again since log is an increasing function, we can take the log to help compute λ^*:

$$\log(M_\lambda) = \log\left(\frac{1}{\lambda 2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)}\left(\frac{1-2\lambda}{\nu-2}\right)^{1-\frac{\nu}{2}}\exp\left(1-\frac{\nu}{2}\right)\right), \tag{10}$$

$$= -\log(\lambda) - \frac{\nu}{2}\log(2) - log\left(\Gamma\left(\frac{\nu}{2}\right)\right) + \left(1-\frac{\nu}{2}\right)(\log(1-2\lambda) - \log(\nu-2)) + \left(1-\frac{\nu}{2}\right). \tag{11}$$

Differentiating with respect to λ and setting to 0:

$$\frac{d(\log(M_\lambda))}{d\lambda} = -\frac{1}{\lambda} + \left(1-\frac{\nu}{2}\right)\left(-\frac{2}{1-2\lambda}\right) = -\frac{1}{\lambda} + \frac{\nu-2}{1-2\lambda}, \tag{12}$$

$$\frac{d(\log(M_\lambda))}{d\lambda} = 0 \implies \frac{1}{\lambda} = \frac{\nu-2}{1-2\lambda}, \tag{13}$$

$$\implies \lambda^* = \frac{1}{\nu}. \tag{14}$$

Verify that M_{λ^*} is a minimiser by checking second derivative with respect to λ

$$\frac{d^2(\log(M_\lambda))}{d^2\lambda} = \frac{1}{\lambda^2} + \frac{2(\nu-2)}{(1-2\lambda)^2}, \tag{15}$$

$$\frac{d^2(\log(M_\lambda))}{d^2\lambda}|_{\lambda=\lambda^*} = \nu^2 + \frac{2(\nu-2)}{\left(1-\frac{2}{\nu}\right)^2} > 0, \text{ since } \nu > 2. \tag{16}$$

So computing M_{λ^*}:

$$M_{\lambda^*} = \frac{\nu^{\frac{\nu}{2}}}{2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)}\exp\left(1-\frac{\nu}{2}\right). \tag{17}$$

1.3 Implement Rejection Sampler for $\nu = 4$

To implement the rejection sampler for $\nu = 4$, we take corresponding $\lambda^* = \frac{1}{\nu} = \frac{1}{4}$.
We first sample $U_i \sim Uniform(0,1)$, and then use the inversion method to sample $X_i^{'} \sim Exponential(\lambda^*)$, and then we calculate an an acceptance probability $a(X_i^{'}) = \frac{p_\nu(X_i^{'})}{M_{\lambda^*}q_{\lambda^*}(X_i^{'})}$. We then generate another $U \sim Uniform(0,1)$, and accept $X_i^{'}$ if $U \leq a(X_i^{'})$.
Using the code in Code Listing 1, we produce the plot in Figure 1. We can see that our sample histogram resembles the probability density $p_\nu(x)$, and so we have successfully sampled from a $\chi^2(4)$ distribution.
Our calculated acceptance rate of 0.6793 is very close to the theoretical
$\hat{a} = \frac{1}{M_{\lambda^*}} = 0.6796$, giving an error of only 0.0003.

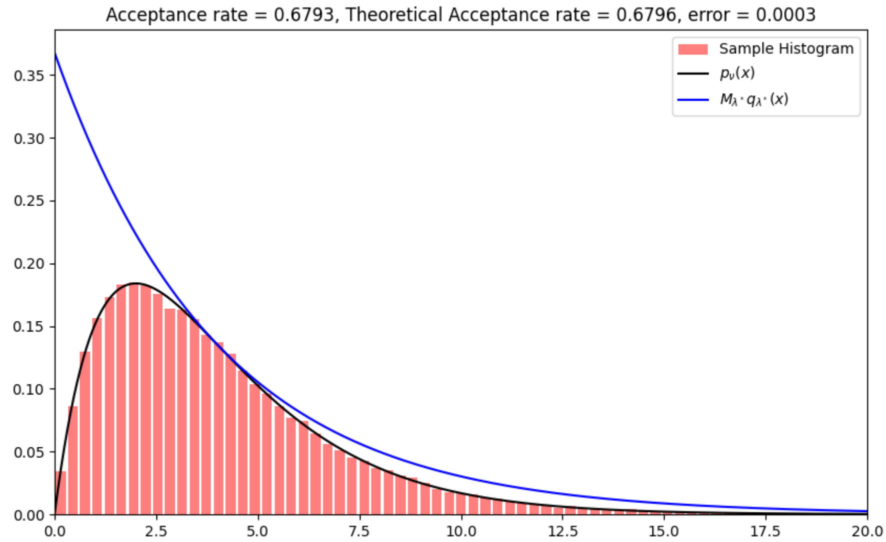

Figure 1

2 QUESTION 2

To sample from $p(x)$, we must first sample an index $i \in \{1, 2, 3\}$ from a discrete distribution with probabilities $[w_1, w_2, w_3]$ to determine which $p_{vi}(x)$ to sample from. We can then use our rejection sampler from Q1) to sample this $p_{v_i}(x)$. Using the code in Code Listing 2, we produce the plot in Figure 2. We see that our sample histogram resembles the probability density $p(x)$, and so we have succesfully sampled from a mixed χ^2 distribution.

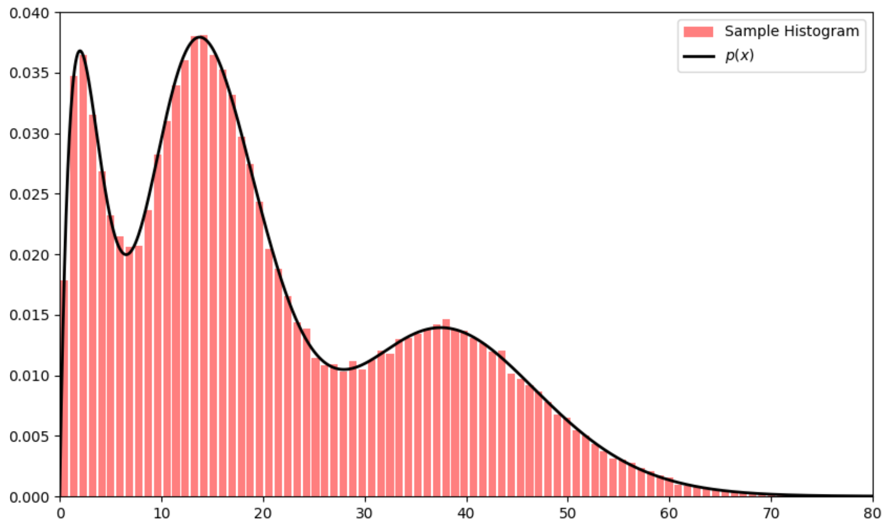

Figure 2

Appendix A

Code Listing 1: Code for question 1

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

### DEFINE FUNCTIONS AND PARAMETERS
def sample_exponential(l):
    u = np.random.uniform(0, 1) # generate uniform number
    inverse = lambda x: (-1/l) * np.log(1-x) # define inverse of
                                              exponential pdf
    sample = inverse(u) # samples from exponential distribution using
                          inversion method
    return sample

n = 100000 # desired size of sample
nu = 4
l = 1 / nu # optimal lambda to minimise M
M = (nu ** (nu / 2)) * np.exp(1 - nu / 2) / ((2 ** (nu / 2)) * np.
                                   math.factorial(int(nu / 2) - 1)) #
                                    calculate optimal M using nu and
                                   lmbda as above
p = lambda x, nu: x ** (nu / 2 - 1) * np.exp(- x / 2) / (2 ** (nu / 2
                                   ) * np.math.factorial(int(nu / 2)
                                   - 1)) # chi-squared pdf function
q = lambda x, l : l * np.exp(-l * x) # exponential pdf function

### SAMPLING
accepted_sample = np.array([]) # initialize array
for i in range(n):
    q_samp = sample_exponential(l) # sample from the exponential
                                     distribution
    acceptance_prob = p(q_samp, nu) / (M * q(q_samp, l)) # calculate
                                     acceptance probability
    u = np.random.uniform(0, 1) # sample u
    if u <= acceptance_prob: # acceptance condition
        accepted_sample = np.append(accepted_sample, q_samp) # append
                                          to accepted array

### PLOTTING
plt.figure(figsize=(10,6))
xx = np.linspace(0, 20, 1000)
plt.xlim(0, 20)
plt.hist(accepted_sample, bins=100, density=True, rwidth=0.8, color="
                                   r", alpha=0.5, label="Sample
                                   Histogram")
plt.plot(xx, p(xx, nu), "k-", label="$p_{\\nu}(x)$")
plt.plot(xx, M * q(xx, l), "b-", label="$M_{\lambda^*}q_{\lambda^*}(x
                                   )$")
plt.legend(loc="upper right")

### ACCEPTANCE RATE
acceptance_rate = len(accepted_sample) / n # number of accepted
                                   samples / total samples considered
```

```
theoretical_rate = 1 / M # theoretical acceptance rate
# put these rates and the error in the plot title, to 4 decimal
                                   places
plt.title(f"Acceptance rate = {np.around(acceptance_rate, 4)},
                                   Theoretical Acceptance rate = {np.
                                   around(theoretical_rate, 4)},
                                   error = {np.around(np.abs(
                                   acceptance_rate - theoretical_rate
                                   ), 4)}")
plt.show()
```

Appendix B

Code Listing 2: Code for question 2

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

def sample_discrete(w):
    cw = np.cumsum(w) # cdf of weights
    u = np.random.uniform(0, 1) # random uniform
    for k in range(len(cw)): # samples the index from the discrete
                                       distribution
        if u <= cw[k]:
            return k

def sample_exponential(l):
    u = np.random.uniform(0, 1) # generate uniform number
    inverse = lambda x: (-1/l) * np.log(1-x) # define inverse of
                                       exponential pdf
    sample = inverse(u) # samples from exponential distribution using
                                        inversion method

    return sample

def sample_chi(p, q, M, nu, l):
    while True:
        q_samp = sample_exponential(l) # sample from the exponential
                                           distribution
        acceptance_prob = p(q_samp, nu) / (M * q(q_samp, l)) #
                                           calculate acceptance
                                           probability
        u = np.random.uniform(0, 1) # sample u
        if u <= acceptance_prob: # acceptance condition
            return q_samp

n = 100000
w_array = np.array([0.2, 0.5, 0.3])
nu_array = np.array([4, 16, 40]) # initialize array of nu values
l_array = 1 / nu_array # initialize array of optimal lambda values
M_array = np.array([(nu ** (nu / 2)) * np.exp(1 - nu / 2) / ((2 ** (
                                   nu / 2)) * np.math.factorial(int(
```

```
                                                       nu / 2) - 1)) for nu in nu_array])
                                                        # calculate array of optimal M's

p = lambda x, nu: x ** (nu / 2 - 1) * np.exp(- x / 2) / (2 ** (nu / 2
                                                       ) * np.math.factorial(int(nu / 2)
                                                       - 1)) # chi-squared pdf function
q = lambda x, l : l * np.exp(-l * x) # exponential pdf function
mixture_density = lambda x, w_array, nu_array: sum([w_array[i]*p(x,
                                                       nu_array[i]) for i in range(len(
                                                       w_array))])

mixture_sample = np.array([])
for i in range(n):
    idx = sample_discrete(w_array)
    chi = sample_chi(p, q, M_array[idx], nu_array[idx], l_array[idx])
    mixture_sample = np.append(mixture_sample, chi)

### PLOTTING
plt.figure(figsize=(10,6))
xx = np.linspace(0, 80, 1000)
plt.xlim(0, 80)
plt.hist(mixture_sample, bins=100, density=True, rwidth=0.8, color="r
                                                       ", alpha=0.5, label="Sample
                                                       Histogram")
plt.plot(xx, mixture_density(xx, w_array, nu_array), color="k",
                                                       linewidth=2, label="$p(x)$")
plt.legend(loc="upper right")
plt.show()
```