MATH60047: Stochastic Simulation Coursework 1

CID: 01938572

1 Q1: Sampling from Chi-Squared using Rejection Sampling

1.1 Compute $M_\lambda = \underset{x}{sup}\frac{p_\nu(x)}{q_\lambda(x)}$

$$M_\lambda = \underset{x}{sup}\frac{p_\nu(x)}{q_\lambda(x)}$$

$$\frac{p_\nu(x)}{q_\lambda(x)} = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})} \cdot \frac{x^{\frac{\nu}{2}-1}e^{-\frac{x}{2}}}{\lambda e^{-\lambda x}} = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})\lambda} x^{\frac{\nu}{2}-1}e^{-\frac{x}{2}+\lambda x}$$

$$\log(\frac{p_\nu(x)}{q_\lambda(x)}) = -\log(2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})\lambda) + (\frac{\nu}{2}-1)\log(x) - \frac{x}{2} + \lambda x$$

Taking the derivative of log with respect to x and setting it to 0, we obtain

$$\frac{d}{dx}(\log(\frac{p_\nu(x)}{q_\lambda(x)})) = (\frac{\nu}{2}-1)\frac{1}{x} - \frac{1}{2} + \lambda x = 0$$

$$(\frac{\nu-2}{2})\frac{1}{x} = \frac{1-2\lambda}{2}$$

$$x^* = \frac{\nu-2}{1-2\lambda}$$

To show that x^* is a maximum, we take the second derivative

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

$$x^* = \frac{\nu-2}{1-2\lambda} \implies \frac{d^2}{dx^2}\log(\frac{p_\nu(x)}{q_\lambda(x)}) = -(\frac{\nu-2}{2})(\frac{1-2\lambda}{\nu-2})^2 = -\frac{(1-2\lambda)^2}{2(\nu-2)} < 0$$

for $\nu > 2$ and $0 < \lambda < \frac{1}{2}$.
Placing $x = x^*$ in the ratio $\frac{p_\nu(x)}{q_\lambda(x)}$, we obtain

$$M_\lambda = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})\lambda}(\frac{\nu-2}{1-2\lambda})^{\frac{\nu}{2}-1}e^{-(\frac{1-2\lambda}{2})(\frac{\nu-2}{1-2\lambda})} = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})\lambda}(\frac{\nu-2}{1-2\lambda})^{\frac{\nu}{2}-1}e^{-\frac{\nu}{2}+1}$$

1.2 Find the optimal λ^* in terms of ν

$$\log(M_\lambda) = -\log(2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})) - \log(\lambda) + (\frac{\nu}{2}-1)(\log(\nu-2) - \log(1-2\lambda)) - \frac{\nu}{2} + 1$$

Taking the derivative of log with respect to λ and setting it to 0, we obtain

$$\frac{d}{d\lambda}(\log(M_\lambda)) = -\frac{1}{\lambda} + (\frac{\nu}{2}-1)(\frac{2}{1-2\lambda}) = 0$$

$$-\frac{1}{\lambda} + (\frac{\nu-2}{2})(\frac{2}{1-2\lambda}) = 0$$

$$-\frac{1}{\lambda} + \frac{\nu-2}{1-2\lambda} = 0$$

$$\frac{-1+2\lambda+\nu\lambda-2\lambda}{\lambda(1-2\lambda)} = 0$$

$$\nu\lambda = 1$$
$$\lambda^* = \frac{1}{\nu}$$

To show that λ^* is a minimum, we take the second derivative

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

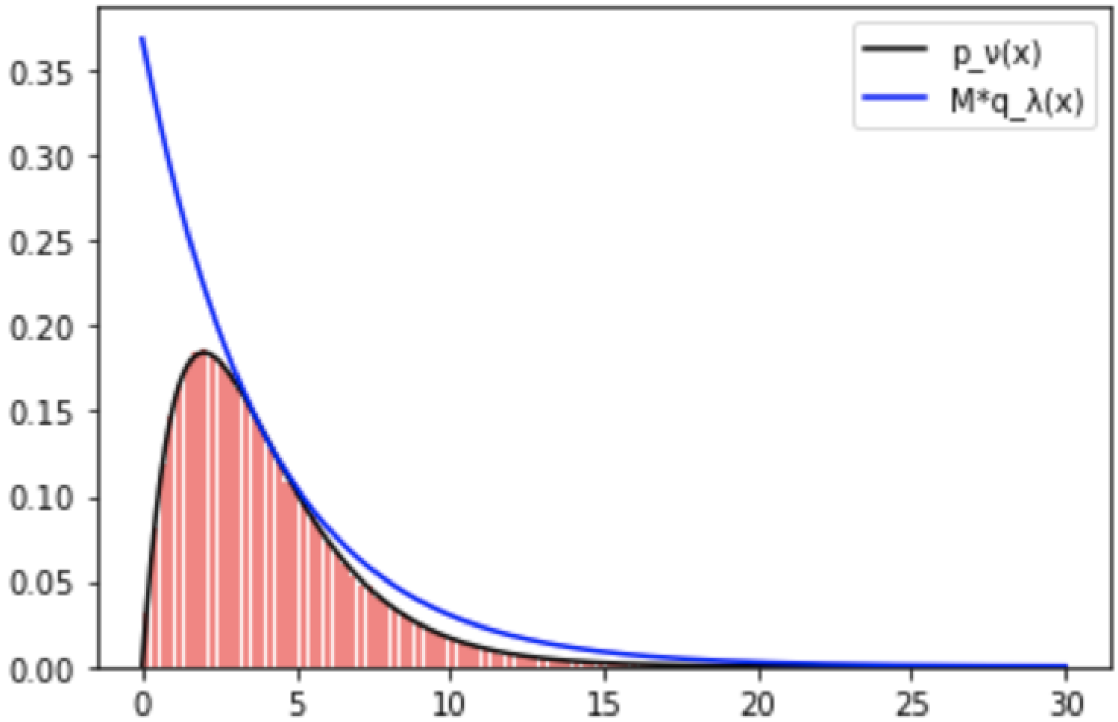
$$\lambda^* = \frac{1}{\nu} \implies \frac{d^2}{d\lambda^2}(\log(M_\lambda)) = \nu^2 + (\nu-2)\frac{2}{(1-\frac{2}{\nu})^2} = \nu^2 + (\nu-2)\frac{2}{(\frac{\nu-2}{\nu})^2} = \nu^2 + \frac{2\nu^2}{\nu-2} > 0$$

for $\nu > 2$.
Thus,

$$M_{\lambda^*} = \frac{\nu}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})}(\frac{\nu-2}{1-\frac{2}{\nu}})^{\frac{\nu}{2}-1}e^{-\frac{\nu}{2}+1} = \frac{\nu}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})}\nu^{\frac{\nu}{2}-1}e^{-\frac{\nu}{2}+1} = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})}\nu^{\frac{\nu}{2}}e^{-\frac{\nu}{2}+1}$$

1.3 Implement the rejection sampler for $\nu = 4$

1.3.1 Plot histogram, $p_\nu(x)$, and $M_{\lambda^*}q_{\lambda^*}(x)$



1.3.2 Compute the acceptance rate and compare it to the theoretical acceptance rate

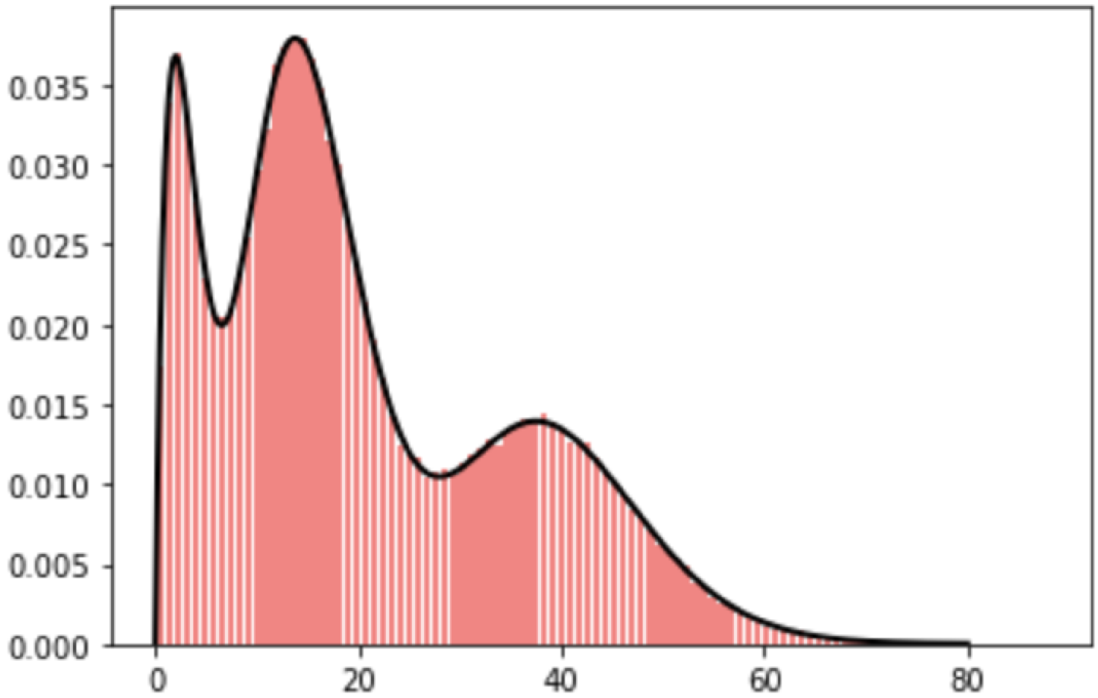
acceptance rate, $a = \frac{\text{number of accepted samples}}{\text{total number of samples}} = 0.67917$
theoretical acceptance rate, $\hat{a} = \frac{1}{M_{\lambda^*}} = 0.6795704571147613$

$$a - \hat{a} = -0.00040045711476122126$$

Thus, the computed acceptance rate and theoretical acceptance rate have very close values.

2 Q2: Sample from a Mixture of Chi-Squared

2.1 Plot the histogram and the density



3 Appendix

3.1 Code for Q1

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

def p(x, nu):
    return x ** (nu / 2 - 1) * np.exp(-x / 2) / (2 ** (nu / 2) * np.math.factorial(
                                                int(nu / 2) - 1))

def q(x, lam):
    return lam * np.exp(-lam * x)

nu = 4
lam = 1 / nu # optimal lambda derived in 1.2
M = nu ** (nu / 2) * np.exp(-nu / 2 + 1) / (2 ** (nu / 2) * np.math.factorial(int(nu
                                            / 2) - 1)) # optimal M derived in 1.2

# Rejection sampler for nu=4
n = 100000
x_accepted = np.array([])
count = 0

for i in range(n):
    # sample from exponential using inversion method
    u_1 = np.random.uniform(0, 1)
    x_proposed = -(1/lam) * np.log(1 - u_1)
    a = p(x_proposed, nu) / (M * q(x_proposed, lam))
    u_2 = np.random.uniform(0, 1)
    if u_2 <= a:
        x_accepted = np.append(x_accepted, x_proposed)
        count += 1 # count accepted samples

# plot histogram, p(x) and Mq(x)
```

```
xx = np.linspace(0, 30, 1000)
chi_squared_density = xx ** (nu / 2 - 1) * np.exp(-xx / 2) / (2 ** (nu / 2) * np.math
                                            .factorial(int(nu / 2) - 1))
M_q_density = M * lam * np.exp(-lam * xx)
plt.hist(x_accepted, bins=100, density=True, rwidth=0.8, color='r', alpha=0.5)
plt.plot(xx, chi_squared_density, 'k-', label='p_ (x)')
plt.plot(xx, M_q_density, 'b-', label='M*q_ (x)')
plt.legend()
plt.show()

a_hat = 1 / M # theoretical acceptance rate
a = count / n # computed acceptance rate
a - a_hat
```

3.2 Code for Q2

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

# function for rejection sampler
def chi_squared_sample(nu, n):
    lam = 1 / nu
    M = nu ** (nu / 2) * np.exp(-nu / 2 + 1) / (2 ** (nu / 2) * np.math.factorial(int
                                                (nu / 2) - 1))
    x_accepted = np.array([])
    while len(x_accepted) < n:
        u_1 = np.random.uniform(0, 1)
        x_proposed = -(1/lam) * np.log(1 - u_1)
        a = p(x_proposed, nu) / (M * q(x_proposed, lam))
        u_2 = np.random.uniform(0, 1)
        if u_2 <= a:
            x_accepted = np.append(x_accepted, x_proposed)
    return x_accepted

# function for sampling from discrete distribution using inversion method
def discrete(w, s):
    cw = np.cumsum(w)
    u = np.random.uniform(0, 1)
    for k in range(len(cw)):
        if cw[k] > u:
            discrete_sample = s[k]
            break
    return discrete_sample

w = np.array([0.2, 0.5, 0.3])
s = np.array([0, 1, 2])
nu = np.array([4, 16, 40])

n = 100000
sample = np.array([])

# mixture sampling
for i in range(n):
    discrete_sample = discrete(w, s)
    x = chi_squared_sample(nu[discrete_sample], 1)
    sample = np.append(sample, x)

# plot histogram and density
def mixture_density(x, w, nu):
    return w[0] * p(x, nu[0]) + w[1] * p(x, nu[1]) + w[2] * p(x, nu[2])

xx = np.linspace(0, 80, 1000)
plt.plot(xx, mixture_density(xx, w, nu), color='k', linewidth=2)
plt.hist(sample, bins=100, density=True, rwidth=0.8, color='r', alpha=0.5)
plt.show()
```