CID: 01857602

1 Question 1

1.1 Part 1

To find the supremum, we take log and differentiate.

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

for some $c > 0$. Taking log gives

$$f(x) \coloneqq \ln\left(\frac{p_\nu(x)}{q_\lambda(x)}\right) = \ln(c) + \left(\frac{\nu}{2} - 1\right)\ln x - \frac{x}{2} + \lambda x$$

Differentiating RHS w.r.t x and setting the derivative to 0 gives

$$\begin{aligned} f'(x) = \frac{\frac{\nu}{2} - 1}{x} - \frac{1}{2} + \lambda = 0 \\ x = \frac{\nu - 2}{1 - 2\lambda} \end{aligned}$$

Denoting the above value by x_0, we notice that $f'(x) < 0$ for $x > x_0$ so f decreases in this range and $f'(x) > 0$ for $x \in (0, x_0)$ so f increases in the range. Hence, x_0 is a global maximum for f and also $\frac{p_\nu(x)}{q_\lambda(x)}$ because of monotonicity.

At $x^* = x_0$,

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

1.2 Part 2

Again, we take log and differentiate.

$$\begin{aligned} g(\lambda) \coloneqq \ln(M_\lambda) &= \left(\frac{\nu}{2} - 1\right)\ln\left(\frac{\nu-2}{1-2\lambda}\right) - \left(\frac{\nu-2}{1-2\lambda}\right)\left(-\frac{1}{2} + \lambda\right) - \ln(\lambda) - C \\ &= \left(1 - \frac{\nu}{2}\right)\ln(1 - 2\lambda) - \left(\frac{\nu-2}{1-2\lambda}\right)\left(-\frac{1}{2} + \lambda\right) - \ln(\lambda) - C_1 \end{aligned}$$

Differentiating RHS w.r.t λ and setting the derivative to 0 gives

$$
\begin{aligned}
g'(\lambda) = \frac{\nu-2}{1-2\lambda} - \left(\frac{\nu-2}{1-2\lambda}\right) - \left(-\frac{1}{2}+\lambda\right)\frac{-(\nu-2)(-2)}{(1-2\lambda)^2} - \frac{1}{\lambda} &= 0 \\
-\left(\lambda-\frac{1}{2}\right)\frac{2(\nu-2)}{(1-2\lambda)^2} - \frac{1}{\lambda} &= 0 \\
\lambda(\nu-2) &= 1-2\lambda \\
\lambda &= \frac{1}{\nu}
\end{aligned}
$$

Note that $g'(\lambda) = \frac{\nu\lambda-1}{(1-2\lambda)\lambda}$, so for $\frac{1}{\nu} < \lambda < \frac{1}{2}$, $g'(\lambda) > 0$ and g is increasing; for $0 < \lambda < \frac{1}{\nu}$, $g'(\lambda) < 0$ and g is decreasing, so using the same argument as above, g and hence M_λ attains minimum at $\lambda = \frac{1}{\nu}$.

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

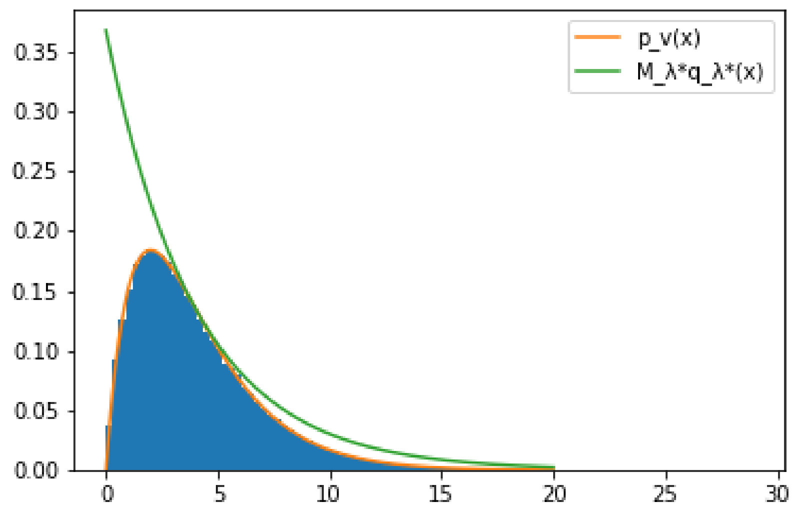
1.3 Part 3

First, we use inversion method to sample from exponential. The inversion method says that if $U \sim Unif(0,1)$ then $F_X^{-1}(U)$ has cdf F_X. So we need to invert the cdf of the exponential.

It is elementary to show that cdf of exponential is $1 - e^{-\lambda x}$. Next we find its inverse.

$$
\begin{aligned}
y &= 1 - e^{-\lambda x} \\
e^{-\lambda x} &= 1 - y \\
x &= -\frac{\ln(1-y)}{\lambda}
\end{aligned}
$$

We can use this and the inversion method to sample from an exponential distribution. Next, we use rejection sampling to sample from the desired distribution. First, we sample X from the exponential distribution, then accept it with probability $\frac{p_\nu(X)}{M_{\lambda^*} q_\lambda(X)}$. If we rejected the sample, we sample a new X and repeat the aforementioned process.



One trial gave an acceptance rate of 0.6806, which is indeed very close to the theoretical rate of 0.6796.

2 Question 2

Firstly, we sample from the discrete distribution. We do this using the inversion method and noting that the CDF is an increasing staircase function whose spacing in y axis reflects probabilities, as indicated in the lecture notes. We sample $U \sim Unif(0,1)$ and choose index i if $U \in [\sum_{j=1}^{i-1} w_j, \sum_{j=1}^{i} w_j)$.

Next, we sample from the desired distribution by noting that, as stated in the notes, one can sample from distribution p_{ν_i} with probability w_i, which is precisely what we do below.

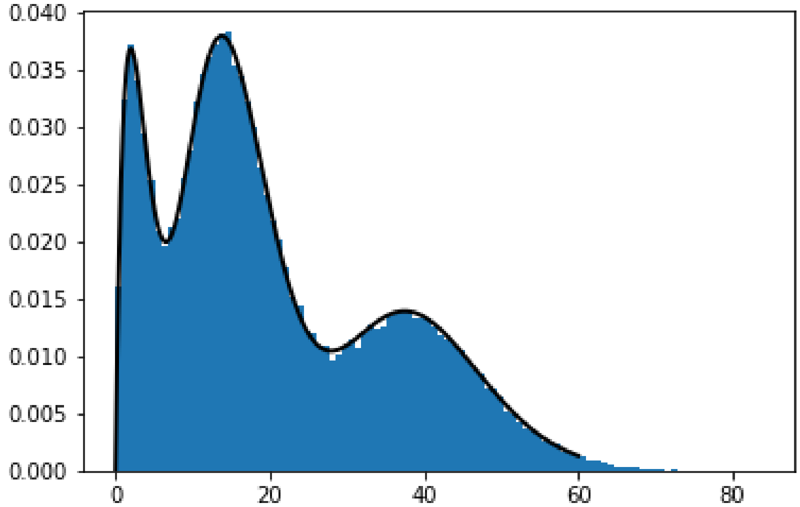


Figure 1: Histogram and density

A Preliminary

```
#  preliminary stuff, feel free to ignore

from sympy import *
init_printing(use_unicode = True)
x, v, lam = symbols('x v lam')
import sympy
e = sympy.E
from math import log
import numpy as np
from math import e
import matplotlib.pyplot as plt

p = x**(v/2 - sympy.Integer(1)) * e**(-x/2) / (2 ** (v/2) *
                                   factorial(v/2 - sympy.Integer(1))
                                   )
q = lam*e ** (-lam*x)
exp = p / q
df= diff(exp, x)
x_0 = solve(df,x)[0]
M_lambda = exp.subs(x, x_0)
```

B Code for Q1

```
#  returns a sample from Exp(lam)
def exponential_sampler(lam):
    # f is inverse of cdf of Exp(lam)

    f = lambda x: -log(1-x)/lam
    return f(np.random.uniform())

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

#  evaluate pdf of q
def eval_q(x, lam):
    return lam*e ** (-lam*x)

#  returns a sample from the chi square distriburion using
                                   parameters v, lam, M
def rejection_sampler(v, lam, M):
    while True:
        global sample_count
        global accept_count
        sample_count += 1
        x_proposed = exponential_sampler(lam)
        a = eval_p(x_proposed, v) / (M * eval_q(x_proposed, lam))
        u = np.random.uniform()
        if u < a:
            accept_count += 1
```

```
            return x_proposed

#  get a tuple of parameters
def get_params(v_cur):
    lam_cur = 1 / v_cur
    M_cur = float(M_lambda.subs([(v,v_cur), (lam, lam_cur)]))
    return v_cur, lam_cur, M_cur

#  sample repeatedly
count = 100000
sample_count = 0
accept_count = 0
samples = [rejection_sampler(*get_params(4)) for i in range(count)]
print(f"Acceptance rate: {accept_count / sample_count}")
print(f"Theoretical acceptance rate: {1/get_params(4)[-1]}")

def plot(samples, v, lam, M):
    plt.hist(samples, density = True, bins=100)
    p_density = lambda a:eval_p(a, v)
    p_density = np.vectorize(p_density)
    x_arr = np.linspace(0,5/lam,10000)
    plt.plot(x_arr,p_density(x_arr), label="p_v(x)")
    q = lambda a:M * eval_q(a, lam)
    plt.plot(x_arr, q(x_arr), label="M_lam*q_lam*(x)")
    plt.legend()
    plt.savefig("Q plot")

#  plot graph
plot(samples, *get_params(4))
```

C Code for Q2

```
#  output a sample from 1 to n inclusive such that number i has
                                probability p[i - 1]
def discrete_sampler(p):
    cumsump = np.cumsum(p)
    n = len(p)
    sample = np.random.uniform()

    for i in range(n):
        if sample < cumsump[i]:
            return i + 1
    return n


#  return a sample from the mixture
def mixture_chi_squared():
    choice = discrete_sampler([0.2, 0.5, 0.3])

    v = [4, 16, 40]
    return rejection_sampler(*get_params(v[choice - 1]))

sample_no = 100000
mixture_samples = [mixture_chi_squared() for i in range(sample_no)]
```

```python
#  plot the histogram and density
def plot_mixture(mixture_samples):
    plt.hist(mixture_samples , density = True , bins=100)

    w = [0.2, 0.5, 0.3]
    nu = [4, 16, 40]
    def mixture_density (x, w, nu):
        return w[0]*eval_p(x, nu[0]) + w[1]*eval_p(x, nu[1]) + w[2]
                                          *eval_p(x, nu[2])
    xx = np.linspace(0, 60 , 1000)
    plt.plot(xx , mixture_density (xx , w, nu), color='k',
                                      linewidth=2)
    plt.savefig("Q2 plot")


plot_mixture(mixture_samples)
```