# Coursework 1

CID 02358380

# 1 Main part

## 1.1 Q1

**1.** We must find $M_\lambda = \sup_x \frac{p_\nu(x)}{q_\lambda(x)}$. For this we use the ansatz

$$\frac{d}{dx}\frac{p_\nu(x)}{q_\lambda(x)} = 0.$$

Now

$$0 = \frac{d}{dx}\frac{1}{2^{\nu/2}\Gamma(\nu/2)\lambda}x^{\nu/2-1}e^{(\lambda-1/2)x}$$

is equivalent to

$$\begin{aligned}
0 &= \frac{d}{dx}x^{\nu/2-1}e^{(\lambda-1/2)x}\\
&= \left(\frac{\nu}{2}-1\right)x^{\nu/2-2}e^{(\lambda-1/2)x} + \left(\lambda-\frac{1}{2}\right)x^{\frac{\nu}{2}-1}e^{(\lambda-1/2)x}\\
&= x^{\nu/2-1}e^{(\lambda-1/2)x}\left(\frac{\frac{\nu}{2}-1}{x}+\lambda-\frac{1}{2}\right) \qquad (1)
\end{aligned}$$

Of course, the product (1) is zero if and only if one of its factors is zero. Hence its follows that $x^{\nu/2-1} = 0$, i.e. $x = 0$ (assuming $\nu > 2$), or

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

implying $x = \frac{\nu/2-1}{1/2-\lambda}$. It is obvious looking at the graph of $\frac{p_\nu}{q_\lambda}$ (see on page 2) that the maximum here is attained at $x^* = \frac{\nu/2-1}{1/2-\lambda}$. Thus, we find

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

**2.** Now we want to optimise $M_\lambda$ over $\lambda$. First, we rewrite $M_\lambda$ as the product of two terms: one constant in $\lambda$, one explicitly depending on $\lambda$.

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

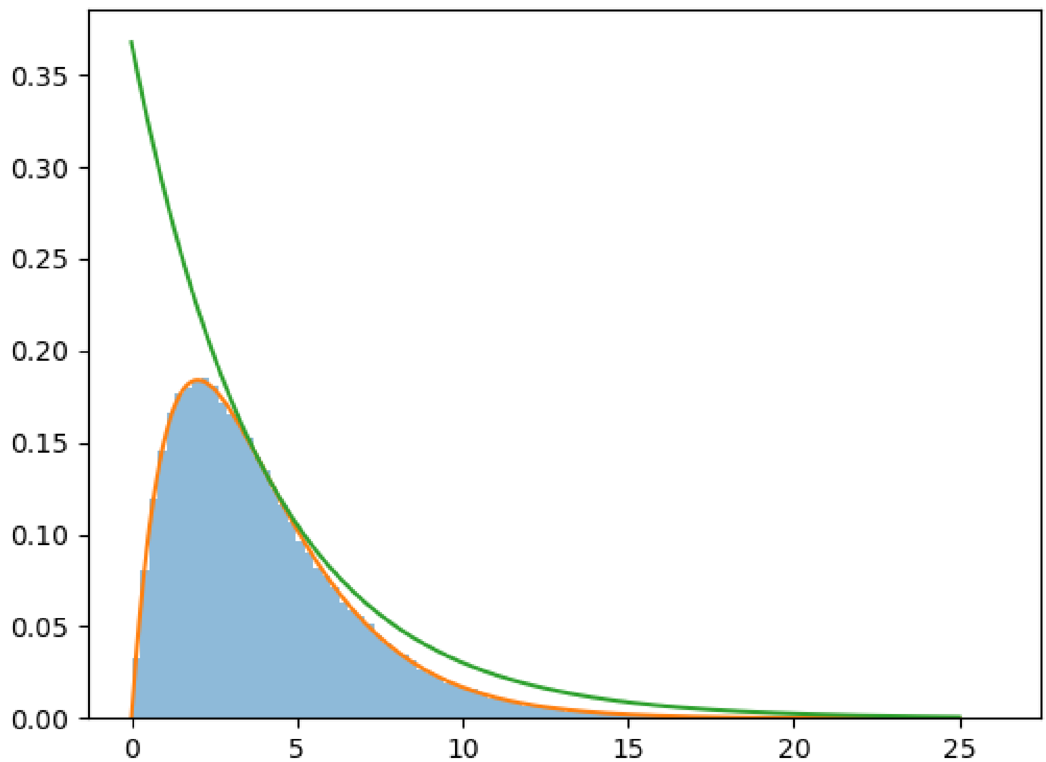
Then $\frac{d}{d\lambda}M_\lambda = 0$ is equivalent to

$$\begin{aligned}
0 &= \frac{d}{d\lambda}\frac{1}{\lambda(\frac{1}{2}-\lambda)^{\nu/2-1}}\\
&= \frac{-1}{\lambda^2(\frac{1}{2}-\lambda)^{\nu/2-1}} + \frac{\frac{\nu}{2}-1}{\lambda(\frac{1}{2}-\lambda)^{\nu/2}}\\
&= \frac{1}{\lambda(\frac{1}{2}-\lambda)^{\nu/2-1}}\left(\frac{-1}{\lambda}+\frac{\frac{\nu}{2}-1}{\frac{1}{2}-\lambda}\right). \qquad (2)
\end{aligned}$$

Again, a product is zero if and only if one of the factors is zero. In 2 the first factor cannot vanish. Thus the second factor is zero and we find

$$\frac{\frac{\nu}{2}-1}{\frac{1}{2}-\lambda^*}=\frac{1}{\lambda^*} \qquad \Longleftrightarrow \qquad \lambda^*=\frac{1}{\nu}.$$

**3.** A rejection sampler of the Chi-squared distribution was implemented and tested in Python. The code can be found in the appendix. Using the `matplotlib` library, a histogram of 100,000 samples was plotted along with the density $p_4$ and the scaled density of the proposal distribution, $M_{\lambda^*}q_{\lambda^*}$.



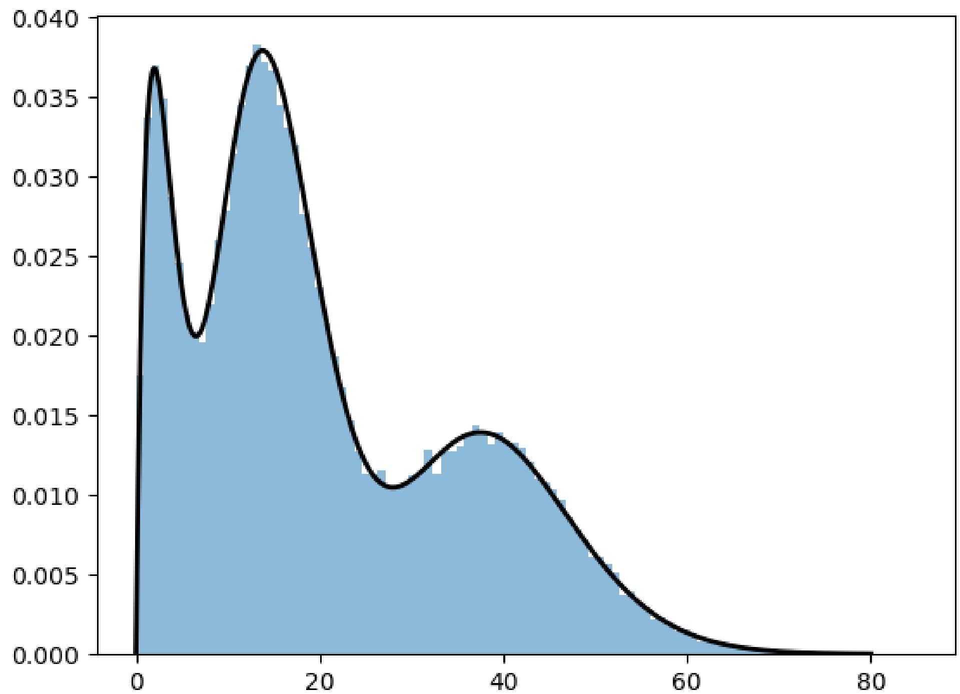

Over 100,000 samples the program computed acceptance rate and theoretical acceptance rate $\frac{1}{M_{\lambda^*}}$ with results

```
acceptance rate    0.6794724575839318
theor. acc. rate   0.6795704571147613
```

We see that these are very close (differ by less than 0.0001).

### 1.2 Q2

I implemented a sampler of the mixed density in Python. First, a sampler of the indices for the different weights ($w_1 = 0.2$, $w_2 = 0.5$, $w_3 = 0.3$) had to be implemented. This was done using the inverse transform method. Here is a plot of the histogram of 100,000 samples of the mixed density and the probability density function.




## 2 Appendix with code

### 2.1 Code for Q1

```
######
# Q1 #
######

import numpy as np
import matplotlib.pyplot as plt

def sample_exponential(lam):
    # This function uses the inverse transform to sample an exponential
                                distribution
    u = np.random.uniform()
    x = -1/lam * np.log(1-u)
```

```
    return x

def p(x, nu):
    return x ** (nu / 2 - 1) * np.exp(-x / 2) / (2 ** (nu / 2) * np.math.
                                        factorial(int(nu / 2+0.1) - 1))

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

def sample_chi_squared(nu, return_tries=False):
  """
      Sample the chi-squared distribution once by rejection sampling.
      The proposal distribution used is the exponential distribution with
                                          parameter lambda = 1/nu.
      The while-loop repeats until one sample is accepted instead of
                                          rejected.
  """
  lam = 1/nu
  M = np.power((nu/2-1)/(1/2-lam), nu/2-1) * np.exp(1-nu/2) / (np.power(2
                                       ,nu/2) * np.math.factorial(int(nu/
                                       2+0.1-1)) * lam)

  tries = 0
  while True:
    tries += 1
    x = sample_exponential(lam)
    y = np.random.uniform()

    if y <= p(x,nu)/(M*q(x,lam)):
      if return_tries:
        return x, tries
      else:
        return x


# generating 100,000 samples of chi-squared (nu = 4) distribution
samples = []
total_tries = 0
for i in range(100000):
  (x, tries) = sample_chi_squared(4, return_tries=True)
  samples.append(x)
  total_tries += tries


# plot of histogram, p and M*q
xx = np.linspace(0,25,1000)
yp = p(xx, 4)
nu = 4
lam = 1/4
M = np.power((nu/2-1)/(1/2-lam), nu/2-1) * np.exp(1-nu/2) / (np.power(2,
                                     nu/2) * np.math.factorial(int(nu/2+0
                                     .1-1)) * lam)
```

```
yMq = M*q(xx, 1/4)

plt.hist(samples, bins=100, density=True, alpha=0.5)
plt.plot(xx,yp)
plt.plot(xx,yMq)
plt.show()


# compute acceptance rate and compare with theoretical acceptance rate
a = len(samples)/total_tries
M = np.power((nu/2-1)/(1/2-lam), nu/2-1) * np.exp(1-nu/2) / (np.power(2,
                                    nu/2) * np.math.factorial(int(nu/2+0
                                    .1-1)) * lam)
a_hat = 1/M

print("acceptance rate  ", a)
print("theor. acc. rate ", a_hat)
```

## 2.2 Code for Q2

```
######
# Q2 #
######

import numpy as np
import matplotlib.pyplot as plt

def sample_exponential(lam):
    # This function uses the inverse transform to sample an exponential
                                          distribution
    u = np.random.uniform()
    x = -1/lam * np.log(1-u)
    return x

def p(x, nu):
    return x ** (nu / 2 - 1) * np.exp(-x / 2) / (2 ** (nu / 2) * np.math.
                                          factorial(int(nu / 2+0.1) - 1))

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

def sample_chi_squared(nu, return_tries=False):
  """
      Sample the chi-squared distribution once by rejection sampling.
      The proposal distribution used is the exponential distribution with
                                            parameter lambda = 1/nu.
      The while-loop repeats until one sample is accepted instead of
                                          rejected.
  """
  lam = 1/nu
```

```
  M = np.power((nu/2-1)/(1/2-lam), nu/2-1) * np.exp(1-nu/2) / (np.power(2
                                               ,nu/2) * np.math.factorial(int(nu/
                                               2+0.1-1)) * lam)

  tries = 0
  while True:
    tries += 1
    x = sample_exponential(lam)
    y = np.random.uniform()

    if y <= p(x,nu)/(M*q(x,lam)):
      if return_tries:
        return x, tries
      else:
        return x

w = [0.2, 0.5, 0.3]
nu = [4, 16, 40]


# drawing 100,000 samples
samples = []
for i in range(100000):
    """
        The indices (0,1,2) are sampled using the inversion method. This
                                        means,
        we first sample the uniform distribution on the interval [0,1]
                                        and then
        apply the inverse transform.
    """
    u = np.random.uniform(0,1)
    if u <= w[0]:
        samples.append(sample_chi_squared(nu[0]))
    elif u <= w[0]+w[1]:
        samples.append(sample_chi_squared(nu[1]))
    else:
        samples.append(sample_chi_squared(nu[2]))

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])


# plot the mixture density and the histogram of the samples
xx = np.linspace(0, 80 , 1000)
plt.plot(xx , mixture_density(xx , w, nu), color='k', linewidth=2)


plt.hist(samples, bins=100, density=True, alpha=0.5)
plt.show()
```