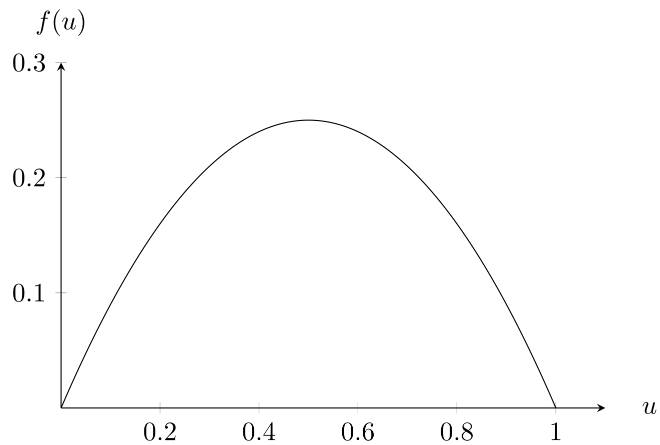
Solution Notes for Data Science Example Sheet 1

Question 1

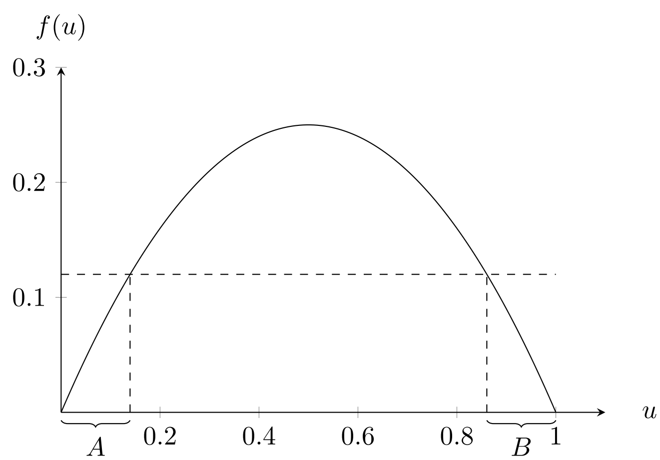
This question is asking to compute the PDF of $X = f(U)$, where $f(u) = u \cdot (1-u)$ is the transformation function. The plot for the transformation function is shown below:



We start by evaluating the CDF of X and then we will differentiate this to obtain the PDF.

$$F_X(x) = \Pr(X \le x) = \Pr(f(U) \le x).$$

Now we are searching for the regions of U where $f(U) \le x$ holds. These are shown below as regions A and B.



The crossing points for a fixed x, can be obtained by solving

$$f(u) = u \cdot (1-u) = x \Leftrightarrow u^2 - u + x = 0 \Leftrightarrow u = \frac{1-\sqrt{1-4x}}{2} \text{ or } u = \frac{1+\sqrt{1-4x}}{2}.$$

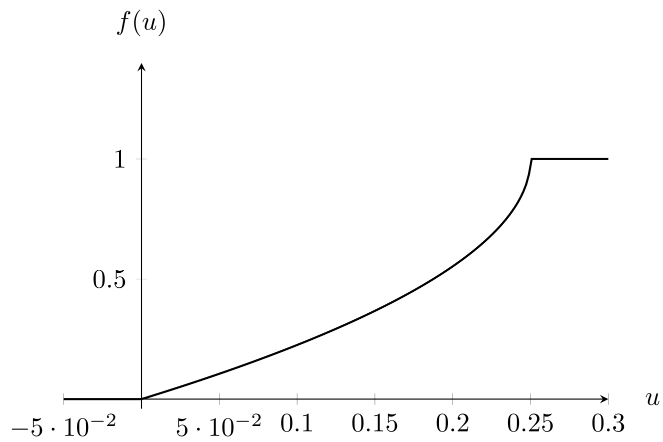
under the assumption that $x \le 1/4$ (otherwise there are no intersection points). Hence, region A is $[0, \frac{1-\sqrt{1-4x}}{2}]$ and region B is $[\frac{1+\sqrt{1-4x}}{2}, 1]$. Hence,

$$\begin{aligned} F_X(x) = \Pr(f(U) \le x) &= \Pr\left(0 \le U \le \frac{1-\sqrt{1-4x}}{2}\right) + \Pr\left(\frac{1+\sqrt{1-4x}}{2} \le U \le 1\right) \\ &= \frac{1-\sqrt{1-4x}}{2} + 1 - \frac{1+\sqrt{1-4x}}{2} \quad (\text{since } U \sim \mathcal{U}[0,1]) \\ &= 1 - \sqrt{1-4x}. \end{aligned}$$

Clearly, for $x \le 0$, $F_X(x) = 0$ and for $x \ge 1/4$, $F_X(x) = 1$. So, the CDF is given by,

$$F_X(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 - \sqrt{1-4x} & \text{for } 0 \le x \le 1/4 \\ 1 & \text{for } x > 1/4. \end{cases}$$

A sketch for which is shown below,



By differentiating, we get the PDF,

$$f_X(x) = \begin{cases} 0 & \text{for } x < 0 \\ \frac{2}{\sqrt{1-4x}} & \text{for } 0 \le x \le 1/4 \\ 1 & \text{for } x > 1/4 \end{cases}$$

Question 2

The likelihood for a single sample x_i is,

$$\Pr(x_i; \lambda) = \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}.$$

Assuming the samples are independent, the joint likelihood for the entire dataset is,

$$\text{lik}(\lambda) = \Pr(x_1, \ldots, x_n; \lambda) = \prod_{i=1}^{n} \Pr(x_i; \lambda) = \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!} = e^{-n\lambda} \cdot \lambda^{\sum_{i=1}^{n} x_i} \cdot \prod_{i=1}^{n} \frac{1}{x_i!}.$$

Maximising the likelihood is equivalent to maximising the log-likelihood (since log is an increasing function), so

$$\log \text{lik}(x_1, \ldots, x_n; \lambda) = -n\lambda + \sum_{i=1}^{n} x_i \log \lambda - \underbrace{\sum_{i=1}^{n} \log(x_i!)}_{\text{const}}.$$

By differentiating with respect to λ,

$$\frac{\partial \log \text{lik}(x_1, \ldots, x_n; \lambda)}{\partial \lambda} = -n + \frac{\sum_{i=1}^{n} x_i}{\lambda}.$$

Setting to zero, we get the MLE estimate $\hat{\lambda}_{\text{MLE}}$,

$$\frac{\partial \log \text{lik}(x_1, \ldots, x_n; \lambda)}{\partial \lambda} = 0 \Rightarrow \hat{\lambda}_{\text{MLE}} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

(Optional) We can argue that this is a maximum by noticing that the derivative left of $\hat{\lambda}$ is positive and on the right is is negative.

Question 3

The implementation is straightforward, we use the formula for the likelihood that we found in Question 2 and we try to maximise it (in scipy we need to minimise the negative of the log-likelihood which is equivalent to maximising the log-likelihood). We can verify the output is close to correct by comparing with the mean.

```python
import scipy.stats
import scipy.optimize
import numpy as np


def log_likelihood(x, l):
    lik = scipy.stats.poisson.logpmf(x, mu=l)
    return np.sum(lik)


x = [3, 2, 8, 1, 5, 0, 8]
initial_guess = 1
lambda_mle = scipy.optimize.fmin(lambda l: -log_likelihood(x, l), 1)

print(f"Lambda MLE: {lambda_mle}")
print(f"Mean : {np.mean(x)}")
# One possible output:
# Optimization terminated successfully.
#          Current function value: 19.033583
#          Iterations: 20
#          Function evaluations: 40
# Lambda MLE: [3.85712891]
# Mean : 3.857142857142857
```

Note 1: Instead of using the log-pmf we could have used the normal pmf, but the product is more prone to underflows (if we had more values in the dataset).

```python
def log_likelihood_2(x, l):
    lik = scipy.stats.poisson.pmf(x, mu=l)
    return np.prod(lik)


lambda_mle = scipy.optimize.fmin(lambda l: -log_likelihood_2(x, l), 1)

print(f"Lambda mle : {lambda_mle}")
# One possible output:
# Optimization terminated successfully.
#          Current function value: -0.000000
#          Iterations: 20
#          Function evaluations: 40
# Lambda mle : [3.85712891]
```

Note 2: Actually, the log-pmf still computes the factorial which we do not need (i.e. makes the computation slower) and also reduces the precision of the computation. So, we can instead compute,

```python
def log_likelihood_3(x, l):
    lik = np.sum(x) * np.log(l) - len(x) * l
    return np.sum(lik)


lambda_mle = scipy.optimize.fmin(lambda l: -log_likelihood_3(x, l), 1)

print(f"Lambda mle : {lambda_mle}")
```

```
# One possible output:
# Optimization terminated successfully.
#          Current function value: -9.448021
#          Iterations: 20
#          Function evaluations: 40
# Lambda mle : [3.85712891]
```

Question 4

The likelihood for a single sample is,

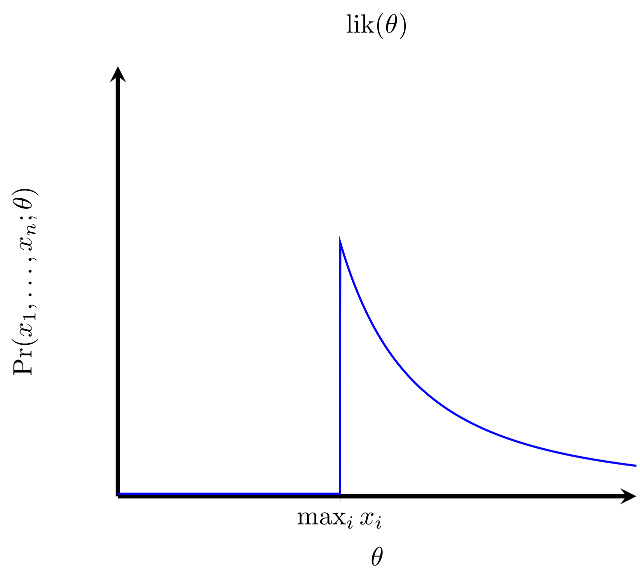
$$\Pr(x_i;\theta) = \frac{1}{\theta}\mathbf{1}_{0\le x_i\le\theta} = \frac{1}{\theta}\mathbf{1}_{0\le x_i}\cdot\mathbf{1}_{x_i\le\theta},$$

by using that $\mathbf{1}_{A \text{ and } B} = \mathbf{1}_A\cdot\mathbf{1}_B$. The joint likelihood for all samples, since they are independent, is given by,

$$\operatorname{lik}(\theta) = \Pr(x_1,\ldots,x_n;\theta) = \prod_{i=1}^{n}\frac{1}{\theta}\mathbf{1}_{0\le x_i}\cdot\mathbf{1}_{x_i\le\theta} = \frac{1}{\theta^n}\cdot\left(\prod_{i=1}^{n}\frac{1}{\theta}\mathbf{1}_{0\le x_i}\right)\cdot\left(\prod_{i=1}^{n}\frac{1}{\theta}\mathbf{1}_{x_i\le\theta}\right) = \frac{1}{\theta^n}\cdot\mathbf{1}_{0\le\min_i x_i}\cdot\mathbf{1}_{\max_i x_i\le\theta}.$$

Now we need to maximise this. Note that θ^{-n} is a decreasing as θ increases. Also note that $\mathbf{1}_{\max_i x_i\le\theta} = 0$ if $\theta < \max_i x_i$. Hence, this expression is maximised for the smallest possible value of θ that leads to non-zero likelihood, so $\hat{\theta}_{\text{MLE}} = \max_i x_i$.

The plot below validates our reasoning:



Question 5

We start by writing out the likelihood for μ,δ,σ,

$$\begin{aligned}
\operatorname{lik}(\mu,\delta,\sigma) &= \Pr(x_1,\ldots,x_m,y_1,\ldots,y_n;\mu,\delta,\sigma)\\
&= \left(\prod_{i=1}^{m}\Pr(x_i;\mu,\delta,\sigma)\right)\cdot\left(\prod_{j=1}^{n}\Pr(y_j;\mu,\delta,\sigma)\right) \quad \text{(since all samples are independent)}\\
&= \left(\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-(x_i-\mu)^2/(2\sigma^2)}\right)\cdot\left(\prod_{j=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-(y_j-\mu-\delta)^2/(2\sigma^2)}\right)
\end{aligned}$$

Maximising the likelihood function is equivalent to maximising the log-likelihood, so

$$\log\operatorname{lik}(\mu,\delta,\sigma) = -\frac{n+m}{2}\cdot\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left(\sum_{i=1}^{m}(x_i-\mu)^2 + \sum_{j=1}^{n}(y_j-\mu-\delta)^2\right).$$

Now, we set the partial derivatives equal to zero to get the MLEs for the parameters,

$$\frac{\partial \log \text{lik}(\mu,\delta,\sigma)}{\partial \delta} = 0 \Rightarrow \sum_{i=1}^{n}(y_i - \delta_{\text{MLE}} - \mu_{\text{MLE}}) = 0 \Rightarrow \overline{y} = \delta_{\text{MLE}} + \mu_{\text{MLE}}.$$

$$\frac{\partial \log \text{lik}(\mu,\delta,\sigma)}{\partial \mu} = 0 \Rightarrow \sum_{i=1}^{m}(x_i - \mu_{\text{MLE}}) + \sum_{j=1}^{n}(y_j - \mu_{\text{MLE}} - \delta_{\text{MLE}}) = 0 \Rightarrow m\cdot(\overline{x} - \mu_{\text{MLE}}) + n\cdot(\overline{y} - \mu_{\text{MLE}} - \delta_{\text{MLE}}) = 0.$$

By subtracting the second from the first, we get $\mu_{\text{MLE}} = \overline{x}$ and this gives $\delta_{\text{MLE}} = \overline{y} - \overline{x}$. **Note:** This is probably what we would have done if we did not use the MLE.

Finally, the MLE for σ is given by

$$\frac{\partial \log \text{lik}(\mu,\delta,\sigma)}{\partial \sigma} = 0 \Rightarrow -\frac{m+n}{\sigma_{\text{MLE}}} + \frac{1}{\sigma_{\text{MLE}}^3}\cdot\left(\sum_{i=1}^{m}(x_i - \mu_{\text{MLE}})^2 + \sum_{j=1}^{n}(y_j - \mu_{\text{MLE}} - \delta_{\text{MLE}})^2\right) \Rightarrow \sigma_{\text{MLE}}^2$$

$$\sigma_{\text{MLE}}^2 = \frac{1}{m+n}\cdot\left(\sum_{i=1}^{m}(x_i - \mu_{\text{MLE}})^2 + \sum_{j=1}^{n}(y_j - \mu_{\text{MLE}} - \delta_{\text{MLE}})^2\right)$$

$$\sigma_{\text{MLE}} = \sqrt{\frac{1}{m+n}\cdot\left(\sum_{i=1}^{m}(x_i - \mu_{\text{MLE}})^2 + \sum_{j=1}^{n}(y_j - \mu_{\text{MLE}} - \delta_{\text{MLE}})^2\right)}.$$

Question 6

The likelihood for a single sample is given by,

$$\Pr(y_i; x_i, \lambda) = \frac{(\lambda x_i)^{y_i} e^{-\lambda x_i}}{y_i!}.$$

Since the samples are assumed independent, the joint likelihood is given by,

$$\text{lik}(\lambda) = \Pr(y_1,\ldots,y_n; x_1,\ldots,x_n,\lambda) = \prod_{i=1}^{n}\frac{(\lambda x_i)^{y_i} e^{-\lambda x_i}}{y_i!} = \lambda^{\sum_{i=1}^{n} y_i} e^{-\lambda\sum_{i=1}^{n} x_i}\prod_{i=1}^{n}\frac{x_i^{y_i}}{y_i!} = (\text{const})\cdot\lambda^{\sum_{i=1}^{n} y_i} e^{-\lambda\sum_{i=1}^{n} x_i}.$$

Maximising the likelihood is equivalent to maximise the log-likelihood (note: the only parameter that we have control over is λ),

$$\log \text{lik}(\lambda) = (\text{const'}) + \sum_{i=1}^{n} y_i \log\lambda - \lambda\sum_{i=1}^{n} x_i.$$

By differentiation, this gives

$$\frac{\partial \log \text{lik}(\lambda; y_1,\ldots,y_n,x_1,\ldots x_n)}{\partial \lambda} = \frac{\sum_{i=1}^{n} y_i}{\lambda} - \sum_{i=1}^{n} x_i.$$

By setting the derivative to zero, we get the maximum likelihood estimate,

$$\frac{\partial \log \text{lik}(\lambda; y_1,\ldots,y_n,x_1,\ldots x_n)}{\partial \lambda} = 0 \Rightarrow \lambda_{\text{MLE}} = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i}.$$

(Optionally), we can argue that this is a maximum since the derivative is positive on the left of λ_{MLE} and it is negative on the right of λ_{MLE}.

Note (from official answers): The answer is intuitively reasonable. The mean of a $\text{Po}(\lambda x_i)$ is λx_i, so a sensible estimate for λ from a single city is y_i/x_i, and the estimate we've just computed corresponds to aggregating all the cities into one megopolis with total population $\sum_{i=1}^{n} x_i$. But why is this the right way to aggregate these per-city estimates? Is it obvious (without going through the algebra that we've just done) that it should be $(\sum_{i=1}^{n} y_i)/(\sum_{i=1}^{n} x_i)$ rather than just the average of the per-city estimates, $n^{-1}\sum_{i=1}^{n} y_i/x_i$?

Question 7

(Method 1): One way of seeing it is that we are picking a y value for the inflection point, say c and then we choose slopes m_1 and m_2 for the two lines. The equation of a line passing through (x_0, c) is given by $y = c + m_i(x - x_0)$. Hence, the entire function f is given by,

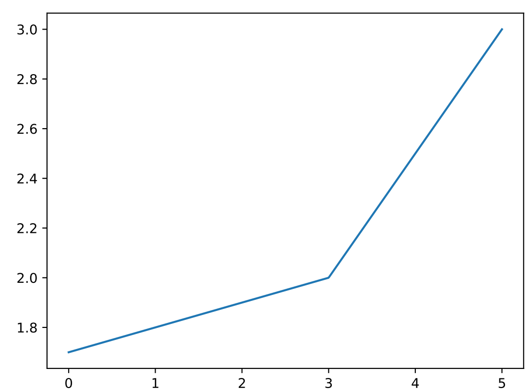
$$f(x) = \begin{cases} c + m_1(x - x_0) & \text{for } x < x_0 \\ c + m_2(x - x_0) & \text{otherwise.} \end{cases}$$

Using indicator functions, this can be written as,

$$f(x) = c + m_1(x - x_0) \cdot \mathbf{1}_{x<x_0} + m_2(x - x_0) \cdot \mathbf{1}_{x \geq x_0} = c + m_1(x - x_0) \cdot (1 - \mathbf{1}_{x \geq x_0}) + m_2(x - x_0) \cdot \mathbf{1}_{x \geq x_0}.$$

The implementation is given below,

```
import numpy
import matplotlib.pyplot as plt


def pred(x, m1, m2, c, inflection_x=3):
    e = numpy.where(x <= inflection_x, 1, 0)
    return e * (m1 * (x - inflection_x) + c) + (1-e) * (m2 * (x - inflection_x) + c)


x = numpy.linspace(0, 5, 1000)
plt.plot(x, pred(x, m1=0.1, m2=0.5, c=2))
plt.show()
```



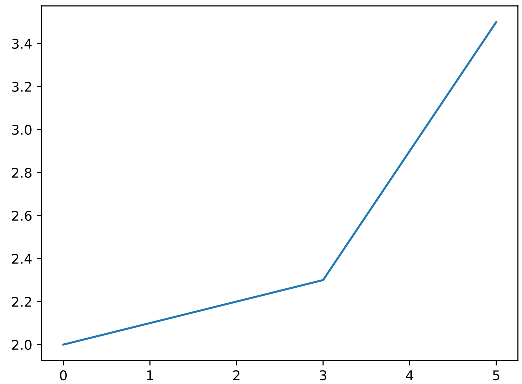
(Method 2): Another way of thinking about this is that in the first part there is a linear equation $y = m_1x + c_1$ and at the inflection point there is a an extra term that grows linearly with the distance from x_0. So this additive term can be represented as $m_2(x - x_0)\mathbf{1}_{x \geq x_0}$. Hence, the entire function f is given by,

$$f(x) = m_1x + c + m_2(x - x_0)\mathbf{1}_{x \geq x_0}.$$

The implementation is given below,

```
import numpy
import matplotlib.pyplot as plt


def pred(x, m1, m2, c, inflection_x=3):
    e = numpy.where(x <= inflection_x, 1, 0)
    return c + m1 * x + (1-e) * m2 * (x - inflection_x)
```

```
x = numpy.linspace(0, 5, 1000)
plt.plot(x, pred(x, m1=0.1, m2=0.5, c=2))
plt.show()
```



Some further questions:

- What if the inflection point was not specified?

Question 8

Using a one-hot encoding, the linear model becomes,

$$\text{temp} \approx \alpha + \beta_1 \sin(2\pi t) + \beta_2 \cos(2\pi t) + \sum_{d \in \text{decades}} \gamma_d \mathbf{1}_{u=d}.$$

In order to create a one-hot encoding for the decades, we search for all decades that appear in the dataset and we sort them uniquely. Then, we append a one-hot encoding of the decades to the features. There is one more point for which we must be careful. The one-hot encoded features of the decades are linearly dependent with the constant term. More formally,

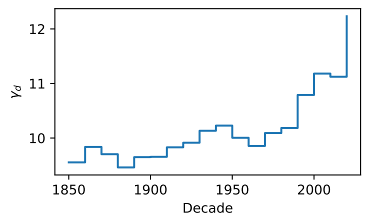
$$\sum_{d \in \text{decades}} \mathbf{1}_{u=d} = 1 \Rightarrow \sum_{d \in \text{decades}} \frac{1}{\gamma_d} \cdot \gamma_d \mathbf{1}_{u=d} - \frac{1}{\alpha} \cdot \alpha = 0.$$

This means that we should fit the linear model without the intercept term, i.e.

(Optionally,) below the a visualisation for the Oxford dataset which has more decades of data than Cambridge.

```
import pandas
import numpy as np
import sklearn.linear_model
import matplotlib.pyplot as plt


climate =
↪  pandas.read_csv('https://www.cl.cam.ac.uk/teaching/2021/DataSci/data/climate.csv')
climate['t'] = climate.yyyy + (climate.mm-1)/12
climate['temp'] = (climate.tmin + climate.tmax)/2

# Let's look at Oxford, which has longer records.
df = climate.loc[(climate.station=='Oxford') & (~pandas.isna(climate.temp))]
t,temp = df['t'], df['temp']
d = np.floor(t/10).astype(int) * 10
```

```
# Plotting is better if we work with integers for decades, not strings!
decades = np.sort(np.unique(d))
X = [np.sin(2 * np.pi * t), np.cos(2 * np.pi * t)] + [np.where(d==i, 1, 0) for i in
↪  decades]
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(np.column_stack(X), temp)

# Use Python's magic sequence-unpacking syntax: gamma is a LIST of remaining coefs
b1, b2, *gamma = model.coef_

_, ax = plt.subplots(figsize=(4, 2.5))
ax.step(decades, gamma, where='post')
ax.set_xlabel("Decade")
ax.set_ylabel("$\\gamma_d$")
plt.show()
```



Question 9

No, they are not linearly independent, since

$$g_1 + g_2 = e_1 + e_2 + e3 = \mathbf{1},$$

where $\mathbf{1}$ is the all ones vector.

We remove one of the vectors from the set and we will show that $\{g_1, e_1, e_2, e_3\}$ are linearly independent. There are several ways to do this.

Method 1: Start from the definition we need to show that

$$\forall a, b, c, d. \quad ag_1 + be_1 + ce_2 + de_3 = \mathbf{0} \Rightarrow a = b = c = d = 0.$$

We can expand the LHS,

$$a \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + b \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + c \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} + d \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a+b \\ a+b \\ a+c \\ a+d \\ b \\ c \\ d \end{bmatrix} = \mathbf{0}$$

The last three rows give $b = c = d = 0$ and the first row gives $a = 0$.

Method 2: In the previous method we just had to show that $a = b = c = d = 0$ is the unique solution to the system, so we want the rank of the matrix of coefficients to be 4. We can compute this using Gaussian Elimination or we can just use numpy,

```
import numpy as np

g1 = np.array([1, 1, 1, 1, 0, 0, 0])
e1 = np.array([1, 1, 0, 0, 1, 0, 0])
e2 = np.array([0, 0, 1, 0, 0, 1, 1])
e3 = np.array([0, 0, 0, 1, 0, 0, 0])
```

```python
print(f"Rank: {np.linalg.matrix_rank(np.column_stack([g1, e1, e2, e3]))}")
# Output: Rank 4
```

Question 10

The linear model can be written as,

$$\mathbf{1}_{\text{outcome=“find”}} \approx \sum_{g \in \text{genders}} \alpha_g \mathbf{1}_{\text{gender}=g} + \sum_{e \in \text{ethnicities}} \beta_e \mathbf{1}_{\text{eth}=e}.$$

The model is not identifiable since,

$$\sum_{g \in \text{genders}} \mathbf{1}_{\text{gender}=g} = \sum_{e \in \text{ethnicities}} \mathbf{1}_{\text{eth}=e} \Rightarrow \sum_{g \in \text{genders}} \frac{1}{\alpha_g} \cdot \alpha_g \mathbf{1}_{\text{gender}=g} - \sum_{e \in \text{ethnicities}} \frac{1}{\beta_e} \cdot \beta_e \mathbf{1}_{\text{eth}=e}.$$

In order to make the parameters of the model identifiable, we remove one of the indicator features, e.g. $\mathbf{1}_{\text{eth=asian}}$. This means that our model becomes,

$$\mathbf{1}_{\text{outcome=“find”}} \approx \sum_{g \in \text{genders}} \alpha_g \mathbf{1}_{\text{gender}=g} + \sum_{e \neq \text{asian}} \beta_e \mathbf{1}_{\text{eth}=e}.$$

In order to interpret the coefficients, we make a table of the possible indicator values,

Gender	Ethnicity	Prediction
Female	Asian	α_{Female}
Female	Black	$\alpha_{\text{Female}} + \beta_{\text{Black}}$
Female	x, $x \neq$ Asian	$\alpha_{\text{Female}} + \beta_x$
Male	Asian	α_{Male}
Male	Black	$\alpha_{\text{Male}} + \beta_{\text{Black}}$
Male	x, $x \neq$ Asian	$\alpha_{\text{Male}} + \beta_x$
Other	Asian	α_{Other}
Other	Black	$\alpha_{\text{Other}} + \beta_{\text{Black}}$
Other	x, $x \neq$ Asian	$\alpha_{\text{Other}} + \beta_x$

From this we see that α_g is the predicted probability that outcome=“find” for a eth=“Asian” person of gender=g, and β_e is the difference between an eth=e person and an eth=“Asian” person, the same difference assumed across all levels of gender.

Question 11

(a) For two random variables U and V, $\max(U, V) \leq x$ iff $U \leq x$ and $V \leq x$. Hence, $\Pr(\max(U, V) \leq x) = \Pr(U \leq x, V \leq x)$ and if the random variables are independent,

$$\Pr(\max(U, V) \leq x) = \Pr(U \leq x, V \leq x) = \Pr(U \leq x) \cdot \Pr(V \leq x).$$

Extending this argument, $\max(U_1, \dots, U_m) \leq x$ iff $U_1 \leq x, \dots, U_m \leq x$. Hence, for U_i being independent uniform random variables, we get for $t \in [0, 1]$,

$$F_T(t) = \Pr(\max(U_1, \dots, U_m) \leq t) = \Pr(U_1 \leq t, \dots, U_m \leq t) = \prod_{i=1}^{m} \Pr(U_i \leq t) = t^m.$$

The PDF is given by,

$$f_T(t) = \frac{d}{dt} F_T(t) = m t^{m-1}$$

(b) We proceed the usual way for computing the MLE for m, i.e. by writing out the likelihood and log-likelihood,

$$\text{lik}(m) = m t^{m-1} \Rightarrow \log \text{lik}(m) = \log m + (m - 1) \log t.$$

Then, differentiating,

$$\frac{\partial \text{lik}(m)}{\partial m} = \frac{1}{m} + \log t.$$

Finally, equating with zero, we find m_{MLE},

$$\frac{1}{m_{\text{MLE}}} + \log t = 0 \Rightarrow m_{\text{MLE}} = \frac{1}{-\log t}.$$

(Optionally,) we can verify that this is a maximum, since the derivative is positive for $m < m_{\text{MLE}}$ and negative for $m > m_{\text{MLE}}$.

Note 1: The following code implements this estimation (in practice the random sampling would be replaced with a "good" hashing function).

```python
import numpy as np

m = 100
samples = np.random.random(m)

m_mle = 1 / (- np.log(np.max(samples)))
print(f"m_MLE: {m_mle}")
# One possible output:
# m_MLE = 161.724...
```

Note 2: This algorithm has several advantages: (i) requires only constant RAM memory and (ii) it can be parallelised (since combining two maximums is easy).

Note 3: The following code extends the MLE estimate using multiple independent hash functions:

```python
import numpy as np

m = 10000
num_reps = 10
acc = 0
for _ in range(num_reps):
    samples = np.random.random(m)
    acc -= np.log(np.max(samples))

print(f"m_MLE: {num_reps / acc}")
# One possible output:
# m_MLE: 9522.075...
```

Question 12

We start by establishing the relation between X and Θ. By looking at the orthogonal triangle, formed by the ray of light, we have

$$\tan(\Theta) = \frac{X}{1} = X.$$

As usual, we start by estimating the CDF of X.

$$\begin{aligned} F_X(x) &= \Pr(X \leq x) = \Pr(\tan(\Theta) \leq x) = \Pr(\Theta \leq \tan^{-1}(x)) \text{ (since tan is monotonous)} \\ &= \frac{1}{\pi} \cdot \tan^{-1}(x) \text{ (Using the CDF of } \Theta \sim U[-\pi/2, \pi/2]). \end{aligned}$$

Hence, we can find the PDF by differentiating,

$$f_X(x) = \frac{d}{dx}\left(\frac{1}{\pi} \cdot \tan^{-1}(x)\right) = \frac{1}{\pi} \cdot \frac{1}{1+x^2}.$$

(Optionally,) we can check that this distribution (the Cauchy distribution) has no mean,

$$E[X] = \int_{-\infty}^{\infty} \frac{1}{\pi} \cdot \frac{1}{1+x^2} \cdot x\text{dx} = 2 \cdot \int_0^{\infty} \frac{1}{\pi} \cdot \frac{x}{1+x^2}\text{dx} = \frac{2}{\pi} \cdot \log(1+x^2)|_0^{\infty} \to \infty$$

Question 13

In this question we have to formulate the optimisation objective and use the `scipy.fmin` function. However, there are a few points that we need to be careful with:

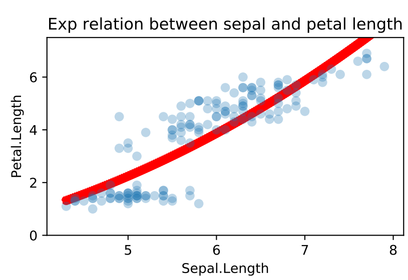
- The *initial point* should be chosen carefully. If $\gamma \geq 1$, then the method will converge at $\gamma \approx 0$.
- Changing the *optimisation method* might significantly improve the result.
- We should be using $\gamma = \exp(\gamma')$ to ensure that $\gamma > 0$.

One possible implementation is the following:

```
import numpy as np
import pandas
import scipy.optimize
import matplotlib.pyplot as plt

iris = pandas.read_csv('https://www.cl.cam.ac.uk/teaching/2021/DataSci/data/iris.csv')
sepal, petal = iris['Sepal.Length'], iris['Petal.Length']


def mse(a):
    predictions = a[0] - a[1] * (sepal ** np.exp(a[2]))
    return np.sum((predictions - petal) ** 2)


a, b, c = scipy.optimize.fmin(mse, np.array([1, 0.3, -0.4]))
newsepal = np.linspace(np.min(sepal), np.max(sepal), 150)
preds = a - b * (newsepal ** np.exp(c))
```

(Optionally,) we can plot the curve we just fitted.

```
fig, ax = plt.subplots(figsize=(4.5, 3))
ax.scatter(newsepal, preds, color='r', zorder=-1, linestyle='dashed')
ax.scatter(iris['Sepal.Length'], iris['Petal.Length'], alpha=0.3)
ax.set_ylim(0, 7.5)
ax.set_ylabel('Petal.Length')
ax.set_xlabel('Sepal.Length')
plt.title('Exp relation between sepal and petal length')
plt.tight_layout()
plt.savefig("ex13_exponential_plot.pdf")
plt.show()
```

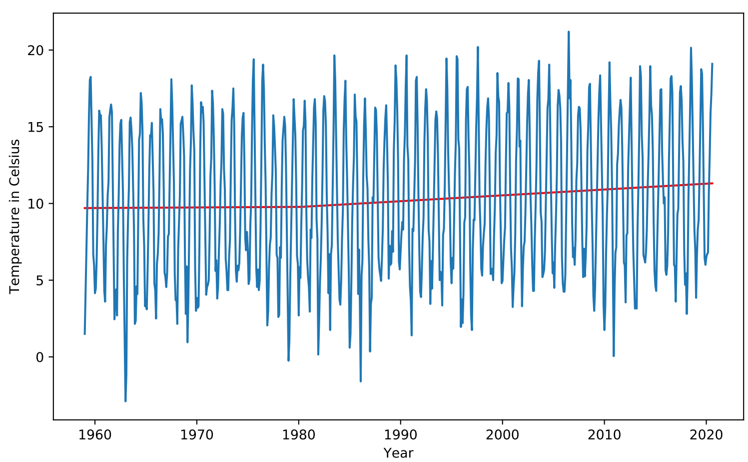


Question 14

There are (at least) three interpretations to this question:

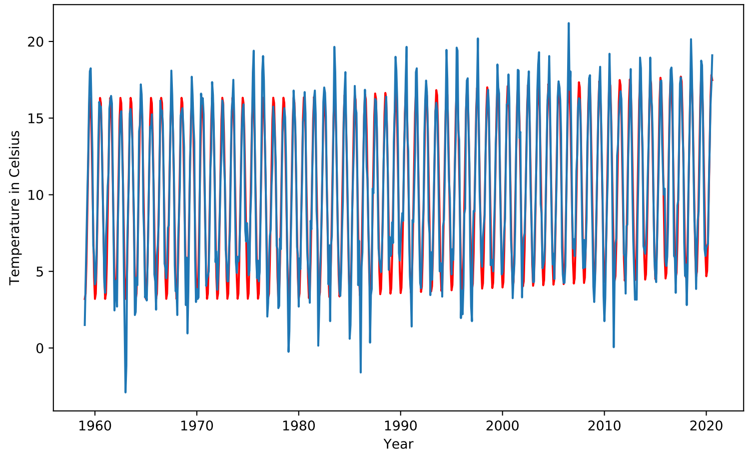
(Interpretation 1): Approximate the temperatures using two linear functions on the time.

```
import pandas
import numpy as np
import sklearn.linear_model
import matplotlib.pyplot as plt


climate =
↪  pandas.read_csv('https://www.cl.cam.ac.uk/teaching/2021/DataSci/data/climate.csv')
climate['t'] = climate.yyyy + (climate.mm-1)/12
climate['temp'] = (climate.tmin + climate.tmax)/2

df = climate.loc[(climate.station=='Cambridge') & (~pandas.isna(climate.temp))]
t, temp = df['t'], df['temp']

X = np.column_stack([
    (t - 1980) * np.where(t >= 1980, 1, 0),
    (t - 1980) * np.where(t < 1980, 1, 0)
])
model = sklearn.linear_model.LinearRegression(fit_intercept=True)
model.fit(X, temp)

_, ax = plt.subplots(figsize=(8, 5))
ax.plot(t, model.predict(X), color='r')
ax.plot(t, temp)
ax.set_xlabel("Year")
ax.set_ylabel("Temperature in Celsius")
plt.tight_layout()
plt.savefig('ex15_cam_plot.pdf')
plt.show()
```



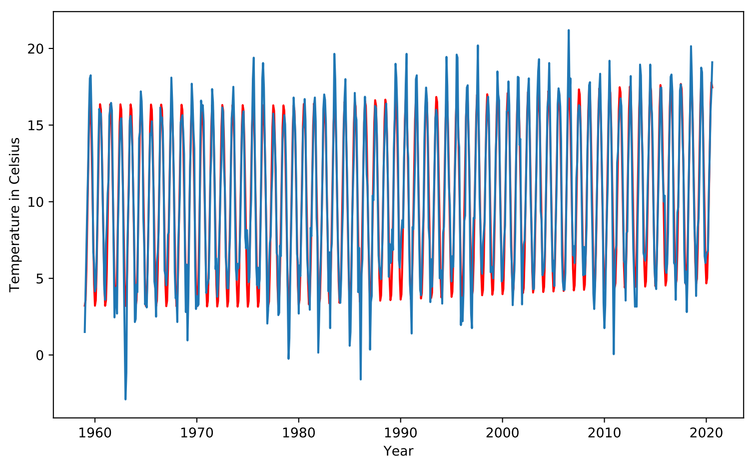
(Interpretation 2): Approximate the temperatures using two additional linear functions on time.

```
X = np.column_stack([
    np.sin(2 * np.pi * t),
    np.cos(2 * np.pi * t),
    (t - 1980) * np.where(t >= 1980, 1, 0),
    (t - 1980) * np.where(t < 1980, 1, 0)
])
model = sklearn.linear_model.LinearRegression(fit_intercept=True)
model.fit(X, temp)
```



(Interpretation 3): Use two different linear models for times before 1980s and after.

```
X = np.column_stack([
    np.sin(2 * np.pi * t) * np.where(t >= 1980, 1, 0),
    np.cos(2 * np.pi * t) * np.where(t >= 1980, 1, 0),
    np.sin(2 * np.pi * t) * np.where(t < 1980, 1, 0),
    np.cos(2 * np.pi * t) * np.where(t < 1980, 1, 0),
    np.where(t < 1980, 1, 0),
    (t - 1980) * np.where(t >= 1980, 1, 0),
    (t - 1980) * np.where(t < 1980, 1, 0)
])
model = sklearn.linear_model.LinearRegression(fit_intercept=True)
model.fit(X, temp)
```



Note: If you want to compare the three models, you need to look closer at the different regions and various metrics.

Question 15

(a) Under the linear model assumptions, we would expect the error to be a Normal distribution with zero mean and constant variance. However, in this case, it seems that the variance increases with x_i.

(b) We start by writing out the likelihood for $\alpha, \beta, \gamma, \sigma$.

$$\begin{aligned}
\operatorname{lik}(\alpha,\beta,\gamma,\sigma) &= \prod_{i=1}^{n} \Pr(y_1,\ldots,y_n;\alpha,\beta,\gamma,\sigma) \\
&= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi(\sigma x_i)^2}} \cdot e^{-(y_i-(\alpha+\beta x_i+\gamma x_i^2))^2/(2(\sigma x_i)^2)} \\
&= \prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi(\sigma x_i)^2)}} \cdot e^{-\sum_{i=1}^{n} \frac{(y_i-(\alpha+\beta x_i+\gamma x_i^2))^2}{2(\sigma x_i)^2}}
\end{aligned}$$

Taking the logarithm, we get,

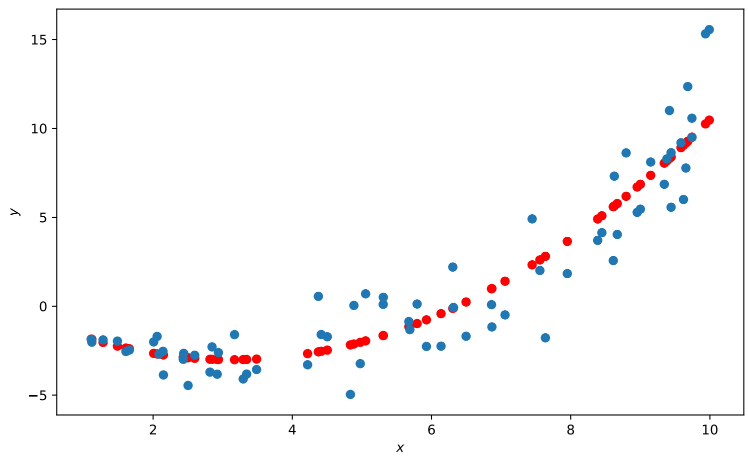
$$\log\operatorname{lik}(\alpha,\beta,\gamma,\sigma) = (\text{const}) - \sum_{i=1}^{n} \log(\sigma x_i) - \sum_{i=1}^{n} \frac{(y_i - (\alpha + \beta x_i + \gamma x_i^2))^2}{2(\sigma x_i)^2}.$$

We can optimise this using scipy.optimize.fmin. A possible implementation is shown below.

```
import pandas
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize


dataset =
↪  pandas.read_csv('https://www.cl.cam.ac.uk/teaching/2021/DataSci/data/heteroscedasticity.csv')
x, y = dataset['x'], dataset['y']
x2 = x ** 2


def predict(a, X, X2):
    return a[0] + a[1] * X + a[2] * X2


def likelihood(a):
    pred = predict(a, x, x2)
    mse = (pred - y) ** 2
    sigma = np.exp(a[3])
    return np.sum(+np.log(sigma * x) + mse / ((2.0 * sigma * sigma) * x2))


a = scipy.optimize.fmin(likelihood, [0.2, 0.2, 0.3, 0.3])
print(f"Parameters: {a}")
# One possible output:
# [-0.21591674 -1.78830743  0.28594986 -1.23517892]
```

Optionally, we can plot the predictions,

```
_, ax = plt.subplots(figsize=(8, 5))
ax.scatter(x, predict(a, x, x2), color='r')
ax.scatter(x, y)
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
plt.tight_layout()
plt.savefig('ex15_plot.pdf')
plt.show()
```



Question 16

Note that

$$f_1 + f_2 = \begin{bmatrix} F_3 \\ F_4 \\ F_5 \\ \vdots \end{bmatrix} + \begin{bmatrix} F_2 \\ F_3 \\ F_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} F_3 + F_2 \\ F_4 + F_3 \\ F_5 + F_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} F_4 \\ F_5 \\ F_6 \\ \vdots \end{bmatrix} = f.$$

So, one solution is $\alpha = 0$, $\beta_1 = 1$ and $\beta_2 = 1$. Using any of the methods we used in Q9, we get that this is the unique solution.

For the linear model $f \approx \alpha + \beta_1 f_1 + \beta_2 f_2 + \beta_3 f_3$, note that

$$f_2 + f_3 = \begin{bmatrix} F_2 \\ F_3 \\ F_4 \\ \vdots \end{bmatrix} + \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} F_2 + F_1 \\ F_3 + F_2 \\ F_4 + F_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} F_3 \\ F_4 \\ F_5 \\ \vdots \end{bmatrix} = f_1.$$

Hence, f_1, f_2, f_3 are linearly dependent. Hence, the linear model is not identifiable.

Question 17

We get the following values,

Ethnicity	**Value**
Asian	$\alpha + \beta_{Asian}$
Black	$\alpha + \beta_{Black}$
Mixed	$\alpha + \beta_{Mixed}$
Other	$\alpha + \beta_{Other}$
White	$\alpha - \beta_{Asian} - \beta_{Black} - \beta_{Mixed} - \beta_{Other}$

The average prediction is given by,

$$\frac{1}{5}\left((\alpha + \beta_{Asian}) + (\alpha + \beta_{Black}) + (\alpha + \beta_{Mixed}) + (\alpha + \beta_{Other}) + (\alpha - \beta_{Asian} - \beta_{Black} - \beta_{Mixed} - \beta_{Other})\right) = \frac{5\alpha}{5} = \alpha.$$

So, α is the average prediction and β_k (for $k \neq White$) is the difference from the average prediction, for ethnicity k.

Question 19

Yes, take a look at $B(0.5, 0.5)$.

The intuition for this is that like an infinite geometric series does converge, we can construct an integral with a point approaching ∞ that also converges. For the mean and variance to also converge we just need to make sure that $\int xf(x)\mathrm{dx}$ and $\int x^2 f(x)\mathrm{dx}$ also converges. One natural choice for $f(x)$ is $kx^{-0.5}$ in $[0,1]$, since $\int_0^1 kx^{-0.5} = 2k\sqrt{x}|_0^1 = 2k$ (Note that if we chose an exponent greater than 1, then this would not work). Now, $xf(x) = kx^{0.5}$ and $x^2 f(x) = kx^{1.5}$, which obviously converge over the interval $[0,1]$.