Solution Notes for Data Science Example Sheet 3

Question 1

(a) We start by writing out the likelihood function,

$$\text{lik}(x; \mu, \sigma) = \Pr(x_1, \ldots, x_n; \mu, \sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}.$$

We want to maximise this, which is equivalent to maximising $\log(\text{lik}(x; \mu, \sigma))$,

$$\log(\text{lik}(x; \mu, \sigma)) = \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x_i-\mu)^2}{2\sigma^2} = -\frac{n}{2}\log(2\pi) - n\log(\sigma) - \sum_{i=1}^{n} \frac{(x_i-\mu)^2}{2\sigma^2}.$$

Differentiating with respect to μ and setting to 0 we get the MLE estimate for μ,

$$\frac{\partial \log(\text{lik}(x; \mu, \sigma))}{\partial \mu} = \sum_{i=1}^{n} \frac{x_i - \mu}{\sigma^2} = 0 \Rightarrow \sum_{i=1}^{n}(x_i - \hat{\mu}_{\text{MLE}}) = 0 \Rightarrow \sum_{i=1}^{n} x_i = n \cdot \hat{\mu}_{\text{MLE}} \Rightarrow \hat{\mu}_{\text{MLE}} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Similarly, for σ,

$$\frac{\partial \log(\text{lik}(x; \mu, \sigma))}{\partial \sigma} = -\frac{n}{\sigma} + \sum_{i=1}^{n} \frac{(x_i-\mu)^2}{\sigma^3} = 0 \Rightarrow n\sigma_{\text{MLE}}^2 = \sum_{i=1}^{n}(x_i - \hat{\mu}_{\text{MLE}})^2 \Rightarrow \hat{\sigma}_{\text{MLE}} = \frac{\sum_{i=1}^{n}(x_i - \hat{\mu}_{\text{MLE}})^2}{n}.$$

Hence, we the MLE estimators are,

$$\hat{\mu}_{\text{MLE}} = \frac{\sum_{i=1}^{n} x_i}{n} \quad \text{and} \quad \hat{\sigma}_{\text{MLE}} = \frac{\sum_{i=1}^{n}(x_i - \hat{\mu}_{\text{MLE}})^2}{n}.$$

(b) To obtain a confidence interval using parametric resampling, we

1. Obtain estimates $\hat{\mu}$ and $\hat{\sigma}$ for the mean and st. deviation using the formulats from (a).
2. Generate a large number of synthetic datasets using the normal distribution with $\hat{\mu}$ and $\hat{\sigma}$. The size of these datasets is the same as the original dataset x.
3. For the i-th dataset, estimate $\hat{\sigma_i}$ and compute the 0.025 and 0.975 quantiles.

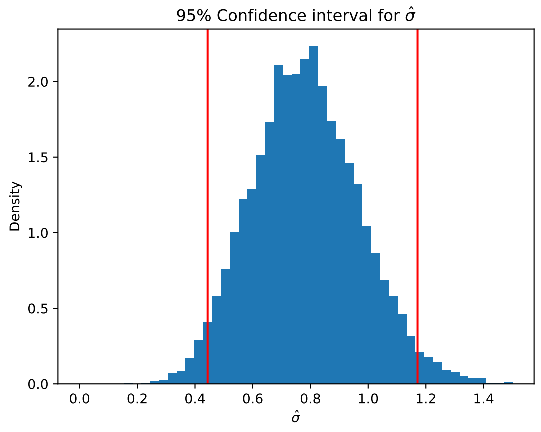
```python
import numpy as np

# The dataset. This is just an example generated with N(3.2, 1.1).
x = [3.98854554, 4.18610834, 3.79048213, 3.40307277, 2.52976334,
     3.52700584, 3.70533666, 3.75593724, 1.90099852, 5.21032427]

# 1. Readout statistic
def mu_hat(x):
    return np.mean(x)
def sigma_hat(x):
    return np.sqrt(np.mean((x-mu_hat(x))**2))

# 2. Function to generate dataset using parametric resampling.
def rx_par():
    return np.random.normal(loc=mu_hat(x), scale=sigma_hat(x), size=len(x))

# 3. (a) Generate datasets and estimate st. deviation in each.
sigma_hat_par = [sigma_hat(rx_par()) for _ in range(10000)]
#    (b) Find the confidence interval.
```

```
lo_par, hi_par = np.quantile(sigma_hat_par, [.025, .975])

print(f"Confidence interval: [{lo_par}, {hi_par}]")
# One possible output:
# Confidence interval: [0.4397..., 1.1721...]
```

Note: We could have first generated the datasets and then estimated the st. deviation in each of these. However, this would mean that we would store all of these datasets in memory. So, instead, we obtain an estimate for a dataset once it is created, and then it is no longer needed.

Optionally, we may plot the confidence interval,

```
# Optional: Plotting the confidence interval.
plt.hist(sigma_hat_par, density=True, bins=np.linspace(0, 1.5, 50))
plt.axvline(x=lo_par, color='r')
plt.axvline(x=hi_par, color='r')
plt.title("$95\\%$ Confidence interval for $\\hat{\\sigma}$")
plt.xlabel("$\\hat{\\sigma}$")
plt.ylabel("Density")
plt.savefig("ex1b_plot.pdf")
plt.show()
```



(c) To obtain a confidence interval using non-parametric resampling, we

1. Generate a large number of synthetic datasets by resampling with replacement dataset x. The size of these datasets is the same as the original dataset x.
2. For the i-th dataset, estimate $\hat{\sigma}_i$ and compute the 0.025 and 0.975 quantiles.

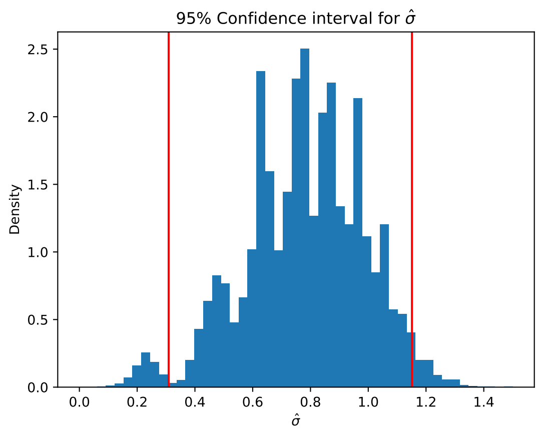
```
import numpy as np

# The dataset. This is just an example generated with N(3.2, 1.1).
x = [3.98854554, 4.18610834, 3.79048213, 3.40307277, 2.52976334,
     3.52700584, 3.70533666, 3.75593724, 1.90099852, 5.21032427]

# 0. Readout statistic
def mu_hat(x):
    return np.mean(x)
def sigma_hat(x):
    return np.sqrt(np.mean((x-mu_hat(x))**2))

# 1. Function to generate dataset using non-parametric resampling.
def rx_nonpar():
    return np.random.choice(x, size=len(x))
```

```
# 2. (a) Generate datasets and estimate st. deviation in each.
sigma_hat_nonpar = [sigma_hat(rx_nonpar()) for _ in range(10000)]
#    (b) Find the confidence interval.
lo_nonpar, hi_nonpar = np.quantile(sigma_hat_nonpar, [.025, .975])

print(f"Confidence interval: [{lo_nonpar}, {hi_nonpar}]")
# One possible output:
# Confidence interval: [0.2768..., 1.1611...]
```

Note: We did not assume anything about the distribution of x (other than the existence of the mean and st. deviation).

Optionally, we may plot the confidence interval,

```
# Optional: Plotting the confidence interval.
plt.hist(sigma_hat_nonpar, density=True, bins=np.linspace(0, 1.5, 50))
plt.axvline(x=lo_nonpar, color='r')
plt.axvline(x=hi_nonpar, color='r')
plt.title("$95\\%$ Confidence interval for $\\hat{\\sigma}$")
plt.xlabel("$\\hat{\\sigma}$")
plt.ylabel("Density")
plt.savefig("ex1c_plot.pdf")
plt.show()
```



Question 2

(a) To obtain a confidence interval, we:

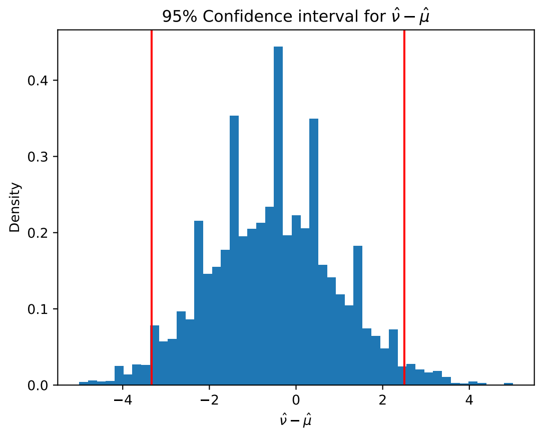
1. Estimate the parameters $\hat{\mu}$ and $\hat{\nu}$ using the MLE for Poisson distributions (which is just the mean, see E1Q2).
2. Parametrically resample a large number of datasets x^i with 3 samples from Po($\hat{\mu}$) and y^i with 2 samples from Po($\hat{\nu}$).
3. For the i-th dataset, estimate the difference of the means between x^i and y^i, and find the 0.025 and 0.975 quantiles for the differences.

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

# The two given datasets X_i ~ Po(mu) and Y_i ~ Po(nu).
x = [3, 1, 5]
```

```
y = [2, 3]

# MLE estimates for mu and nu.
mu_hat = np.mean(x)
nu_hat = np.mean(y)

# Estimating the quantity of interest.
def t(x,y):
    return np.mean(y) - np.mean(x)
# Resampling from the Poisson distributions.
def rxy():
    return (np.random.poisson(mu_hat, size=len(x)),
            np.random.poisson(nu_hat, size=len(y)))

t_ = np.array([t(*rxy()) for _ in range(10000)])
lo, hi = np.quantile(t_, [.025, .975])

print(f"Confidence interval: [{lo}, {hi}]")
# One possible output:
# Confidence interval: [-3.333..., 2.5].
```

Optionally, we can visualise the confidence interval on the empirical distribution of $\hat{\mu} - \hat{\nu}$.

```
# Optional: Plotting the confidence interval.
plt.hist(t_, density=True, bins=np.linspace(-5, 5, 50))
plt.axvline(x=lo, color='r')
plt.axvline(x=hi, color='r')
plt.title("$95\\%$ Confidence interval for $\\hat{\\nu} - \\hat{\\mu}$")
plt.xlabel("$\\hat{\\nu} - \\hat{\\mu}$")
plt.ylabel("Density")
plt.show()
```



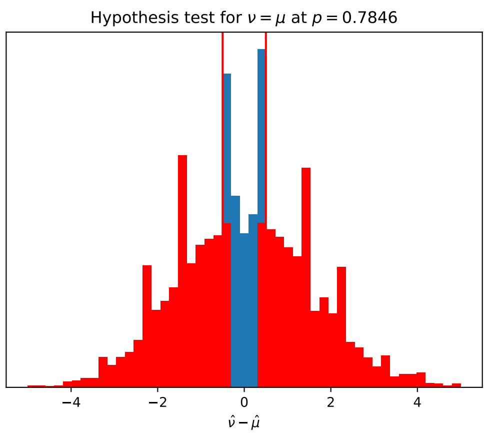
(b) Since we are asking "Is there a difference", we must consider the two-sided test in order to determine if there is an increase or a decrease. In order to compute the p-value for the hypothesis test, we:

1. Estimate the MLE for the Poisson distribution assuming all samples were generated using the same Poisson distribution Po(λ).
2. Parametrically resample a large number of datasets x^i and y^i, using the same underlying distribution Poi($\hat{\lambda}$), of the same length as the given x and y. (Note: this is the same as generating 5 samples from Poi($\hat{\lambda}$) and splitting into 3 and 2 samples).
3. For the i-th dataset, estimate the difference of the means between x^i and y^i, and find the proportion of samples that is smaller than the observed difference between the means of the two datasets x and y.

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

# The two given datasets, assuming both were generated from the same distribution,
# so X_i ~ Po(lambda) and Y_i ~ Po(lambda).
x = [3, 1, 5]
y = [2, 3]

# MLE estimate for lambda.
lambda_hat = np.mean(x + y)

# Estimating the quantity of interest.
def t(x,y):
    return np.mean(y) - np.mean(x)
# Resampling from the common Poisson distribution.
def rxy0():
    return (np.random.poisson(np.mean(x+y), size=len(x)),
            np.random.poisson(np.mean(x+y), size=len(y)))

t0_ = np.array([t(*rxy0()) for _ in range(10000)])
# Note: t0_ >= t(x, y) is a boolean. Taking the mean over the entire
# numpy array is equivalent to finding the proportion.
p = 2 * min(np.mean(t0_ >= t(x, y)), np.mean(t0_ <= t(x, y)))

print(f"p-value: {p}")
# One possible output:
# p-value: 0.7936
```

Optionally, we can visualise the acceptance region on the empirical distribution of $\hat{\mu} - \hat{\nu}$.

```
# Optional: Plotting the rejection region.
plt.hist(t0_, bins=np.linspace(-5, 5, 50))
l = min(t(x, y), -t(x, y))
r = max(t(x, y), -t(x, y))
plt.hist(t0_[t0_ <= l], bins=np.linspace(-5, 5, 50), color='r')
plt.hist(t0_[t0_ >= r], bins=np.linspace(-5, 5, 50), color='r')
plt.axvline(x=l, color='r')
plt.axvline(x=r, color='r')
plt.axes().get_yaxis().set_visible(False)
plt.title("Hypothesis test for $\\hat{\\nu} = \\hat{\\mu}$ at $p=" + str(p) + "$")
plt.xlabel("$\\hat{\\nu} - \\hat{\\mu}$")
plt.ylabel("Density")
plt.show()
```



(c) In part (b), we assume that the two sets x and y come from the same Poisson distribution and by resampling we estimate the probability of observing a difference of means smaller than the observed difference. In part (a), we assume that the two sets come from different distributions, resample the datasets and estimate the empirical distribution for the difference of means between the datasets. This allows us to find an interval containing 95% of the observations.

Some additional questions:

- Should we be doing data analysis when we have 5 datapoints?
- Why do we observe these spikes in the data?
- Is it strange that we observe some spikes near the boundaries of the hypothesis test?

Question 3

(a) There are three steps in calculating the confidence intervals:

(1) Find the MLEs for the parameters of the linear model and the st. deviation of the noise.

(2) Resample the dataset using the same values for t as in the original dataset. So, the randomness comes from the noise term. (Note: You could have also randomly sampled the t values in the range of t values given.)

(3) For the i-th resampled dataset, find the MLE $\hat{\gamma}_i$ for γ. Compute the 0.025 and 0.975 quantiles for $\hat{\gamma}$.

Some preliminary code:

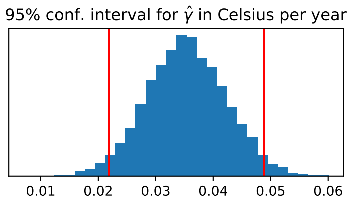
```python
import numpy as np
import pandas
import matplotlib.pyplot as plt
import sklearn.linear_model

# Load the dataset
url = 'https://www.cl.cam.ac.uk/teaching/2021/DataSci/data/climate.csv'
climate = pandas.read_csv(url)
climate = climate.loc[(climate.station == 'Cambridge') & (climate.yyyy >= 1985)].copy()
t = climate.yyyy + (climate.mm-1)/12
temp = (climate.tmin + climate.tmax)/2
```

The main part:

```python
# Fit a simple linear model.
X = np.column_stack([np.sin(2 * np.pi * t), np.cos(2 * np.pi * t), t-2000])
model = sklearn.linear_model.LinearRegression()
model.fit(X, temp)

# Step 1.
# Define a readout function that accepts a dataset and returns the statistic of
↪ interest.
def gamma_hat(temp):
    # Fit the model. Use the t values from the original climate dataset,
    # and assume that temp contains a new vector of temperature readings
    # that matches the times t in the dataset.
    # Return the gamma coefficient from the fitted model.
    new_model = sklearn.linear_model.LinearRegression()
    new_model.fit(X, temp)
    return new_model.coef_[-1]


# Define a function to generate a synthetic copy of the dataset,
# using parametric resampling. Here, parametric resampling means
# resampling the noise.
pred = model.predict(X)
sigma_hat = np.sqrt(np.mean((pred - temp) ** 2))
def rtemp():
    return np.random.normal(loc=pred, scale=sigma_hat)

# Step 3.
# Sample the readout statistic, and find the 0.25 and 0.975 quantiles.
gamma_hat_ = [gamma_hat(rtemp()) for _ in range(10000)]
lo, hi = np.quantile(gamma_hat_, [.025, .975])

print(f"gamma : {gamma_hat(temp)}")
print(f"Confidence interval: [{lo}, {hi}]")
# One possible output:
# gamma : 0.03537...
# Confidence interval: [0.02192..., 0.04881...]
```

Optionally, plotting the confidence interval,

```
_, ax = plt.subplots(figsize=(4.5, 2.0))
plt.title("$95\\%$ conf. interval for $\\hat{\\gamma}$ in Celsius per year")
plt.xlabel("$\\gamma$")
plt.ylabel("Density")
ax.get_yaxis().set_visible(False)
ax.hist(gamma_hat_, bins=30)
ax.axvline(lo, color='r')
ax.axvline(hi, color='r')
plt.show()
```



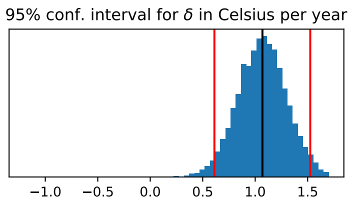
(b) **Confidence interval:** Computing the confidence interval is similar to the first part. However, we need to notice that the model will not be identifiable if we include an intercept term.

```
# Fit the model.
decade = np.floor(t/10).astype(int) * 10
decades = list(np.sort(np.unique(decade)))
X = np.column_stack([np.where(decade == d, 1, 0) for d in decades] +
↪  [np.sin(2*np.pi*t), np.cos(2*np.pi*t)])

model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, temp)
gamma_hat = [model.coef_[decades.index(d)] for d in decades]
pandas.Series(gamma_hat, index=decades)

# Step 1. Define a readout function
def delta(temp):
    model = sklearn.linear_model.LinearRegression(fit_intercept=False)
    model.fit(X, temp)
    return model.coef_[decades.index(2010)] - model.coef_[decades.index(1980)]

# Step 2. To generate a synthetic dataset ...
```

```
pred = model.predict(X)
def rtemp():
    sigma_hat = np.sqrt(np.mean((temp-pred)**2))
    return np.random.normal(loc=pred, scale=sigma_hat)

# Step 3. Sample the readout statistic, report a confidence interval
delta_ = np.array([delta(rtemp()) for _ in range(10000)])
lo, hi = np.quantile(delta_, [.025, .975])
print(f"delta : {delta(temp)}")
print(f"Confidence interval: [{lo}, {hi}]")
# One possible output:
# delta : 1.069...
# Confidence interval: [0.6114..., 1.524...]
```

Optionally, plotting the confidence interval,

```
# Depict a two-sided confidence interval
_, ax = plt.subplots(figsize=(4.5, 2))
plt.title("$95\\%$ conf. interval for $\\hat{\\delta}$ in Celsius per year")
plt.xlabel("$\\delta = \\gamma_{2010s} - \\gamma_{1980s}$")
plt.ylabel("Density")
ax.get_yaxis().set_visible(False)
ax.hist(delta_, bins=np.linspace(-1.2, 1.7, 60))
ax.axvline(lo, color='r')
ax.axvline(hi, color='r')
ax.axvline(delta(temp), color='black')
plt.show()
```



Hypothesis test: For the hypothesis test, we need to assume that $\gamma_{1980s} = \gamma_{2010s}$ and generate the data. Then, we fit the original model and then estimate the probability of observing the difference $\hat{\gamma}_{2010s} - \hat{\gamma}_{1980s}$ (where $\hat{\gamma}_{2010s}$ and $\hat{\gamma}_{1980s}$ are the MLE estimates for the original dataset). The model is then written as

$$\text{temp} = \beta_1 \sin(2\pi t) + \beta_2 \cos(2\pi t) + \gamma_{1990s}\mathbf{1}_{d=1990} + \gamma_{2000s}\mathbf{1}_{d=2000} + +\gamma_{2020s}\mathbf{1}_{d=2020s} + \alpha\mathbf{1}_{d=1990s \vee d=2010s}$$

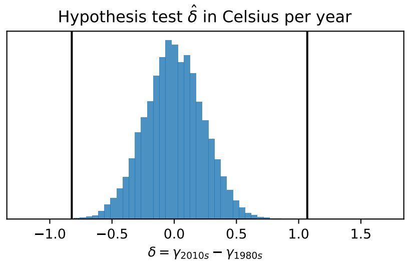
and we extend the previous code (since we need to reuse the readout function) as

```
# Step 1. Define a test statistic
# (A good choice is delta defined above)

# Step 2. To generate a synthetic dataset, assuming H0 is true
decades0 = [d for d in decades if d not in [1980,2010]]
X0 = np.column_stack([np.where(decade == d, 1, 0) for d in decades0] +
↪  [np.sin(2*np.pi*t), np.cos(2*np.pi*t)])
model = sklearn.linear_model.LinearRegression()
model.fit(X0, temp)
pred0 = model.predict(X0)
sigma_hat0 = np.sqrt(np.mean((temp-pred0)**2))
def rtemp0():
    return np.random.normal(loc=pred0, scale=sigma_hat0)

# Step 3. Sample the test statistic, report p-value
```

```
delta_0_ = np.array([delta(rtemp0()) for _ in range(10000)])
p = 2 * min(np.mean(delta_0_ >= delta(temp)), np.mean(delta_0_ <= delta(temp)))

print(f"p-value = {p}")
# One possible output:
# p-value = 0.0
```

Optionally, plotting the hypothesis test,

```
# Depict a two-sided hypothesis test.
# This code assumes that the observed test statistic is on the right hand tail,
# and finds an equally-extreme point u on the left hand tail.
_, ax = plt.subplots(figsize=(4.5, 3))
plt.title("Hypothesis test $\\hat{\\delta}$ in Celsius per year")
plt.xlabel("$\\delta = \\gamma_{2010s} - \\gamma_{1980s}$")
plt.ylabel("Density")
ax.get_yaxis().set_visible(False)
b = np.linspace(-1.2, 1.7, 60)

ax.hist(delta_0_, bins=b, alpha=.8)
u = np.sort(delta_0_)[round((p/2)*len(delta_0_))]
ax.hist(delta_0_[delta_0_ > delta(temp)], bins=b, color='red')
ax.hist(delta_0_[delta_0_ < u], bins=b, color='red')
ax.axvline(delta(temp), color='black')
ax.axvline(u, color='black', linestyle='dotted')
plt.show()
```



Some follow-up questions:

- What does this p-value mean?
- What about comparing different decades?

Question 4

(a) The likelihood for the i-th sample is

$$\Pr(x_i; \theta) = \theta^{x_i}(1-\theta)^{1-x_i}.$$

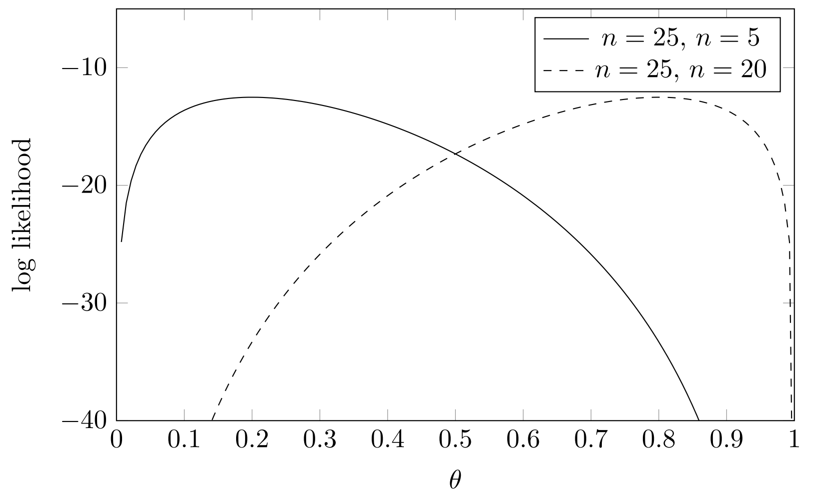
Since the samples are independent,

$$\text{lik}(\theta) \Pr(x_1, \dots, x_n; \theta) = \prod_{i=1}^{n} \Pr(x_i; \theta) = \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i} = \theta^{\sum_{i=1}^{n} x_i}(1-\theta)^{\sum_{i=1}^{n}(1-x_i)} = \theta^y(1-\theta)^{n-y}.$$

(b) Taking logs on both sides,

$$\log \Pr(x_1, \dots, x_n; \theta) = y \log \theta + (n-y) \log(1-\theta).$$

Let's fix $n = 25$ and choose $y = 5$ and $y = 20$. This gives the following plots,



(c) By differentiating,

$$\frac{\partial \log \text{lik}(x_1, \dots, x_n; \theta)}{\partial \theta} = \frac{y}{\theta} - \frac{n-y}{1-\theta} = \frac{y - \theta n}{\theta(1-\theta)},$$

which is zero for $\theta^* = y/n$. Since $y \leq n$, it means that $\theta^* \geq 0$ and $\theta \leq 1$, so the derivative is increasing on the left of θ^* and is decreasing on the right of θ^*, so θ^* is a maximum.

Assuming H_0, it means that $\theta \geq 1/2$, so if $\theta^* \geq 1/2$, the MLE is θ^*, but if not, it means that the log-likelihood is decreasing in that region so we must choose the left-most point, i.e. 1/2. In short,

$$\hat{\theta}_{\text{MLE}} = \max(1/2, y/n).$$

(d) The test statistic $Y = \sum_{i=1}^{n} X_i$ is a sum of Binomial random variables $X_i \sim \text{Bin}(1, \theta_{\text{MLE}})$, so it is also a Binomial random variable with distribution $\text{Bin}(n, \theta_{\text{MLE}})$.

(e) Let y be the value that we observed. All larger values, are in favour of the hypothesis, while smaller values are against it. So, if y is the observed statistic, then

$$p = \Pr(Y \leq y) = \Pr(\text{Bin}(n, \hat{\theta}_{\text{MLE}}) \leq y).$$

Note: You can estimate this using binomial.cdf or by writing a for-loop evaluating the sum of Binomial coefficients.

Question 5

This is the same setup as in the previous question, but we are also given that $y = 0$, so $\theta = 1/2$ and

$$p = \Pr(\text{Bin}(n, 1/2) \leq 0) = \Pr(\text{Bin}(n, 1/2) \leq 0) = \left(\frac{1}{2}\right)^n.$$

We are asked to find n such that $p < 5\%$, so

$$\left(\frac{1}{2}\right)^n < 0.05 \Leftrightarrow n > -\log_2(0.05) = 4.321...$$

Since n is integer, this is equivalent to $n \geq 5$.

Questions 6 and 7

The parametric and non-parametric resamplers are quite straightforward to implement:

```python
import numpy as np

theta = 2
num_samples = 1000

def parametric_conf_interval(samples):
    theta_mle = np.max(samples)

    thetas = np.array([np.max(theta_mle * np.random.random(len(samples))) for _ in
    ↪  range(10000)])
    return np.quantile(thetas, [.025, .975])


def non_parametric_conf_interval(samples):
    thetas = np.array([np.max(np.random.choice(samples, size=len(samples))) for _ in
    ↪  range(10000)])
    return np.quantile(thetas, [.025, .975])


samples = theta * np.random.random(num_samples)
print(f"Parametric confidence interval: [{parametric_conf_interval(samples)}]")
print(f"Non-parametric confidence interval: [{non_parametric_conf_interval(samples)}]")

# One possible output:
# Parametric confidence interval: [[1.992... 1.999...]]
# Non-parametric confidence interval: [[1.998... 1.999... ]]
```

The function np.random.random will never return the right boundary (according to the documentation). Hence, the initial samples will never contain θ. So, $\hat{\theta}_{\text{MLE}} < \theta$ and so the parametric resampling procedure will always generate values smaller than θ. Similarly, the non-parametric resampler will always resample values smaller than θ. Hence, neither will have θ or larger values, hence np.quantile will never return an interval that contains θ.

As stated in the note below the question, resampling is a heuristic that does not work well with extreme statistics like the maximum.

Question 8

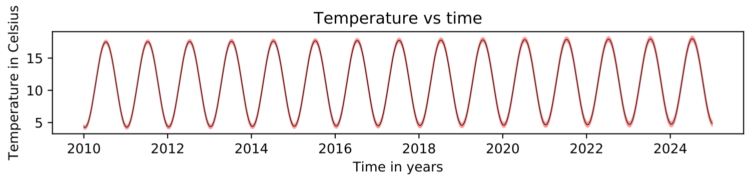
We need to modify our approach from Q3a, by keeping the entire fitted model (not just γ) and then storing all predictions for `Xnew`. In the end, we find the confidence interval for each `x` in `Xnew`. The code below implements the change in the code of Q3:

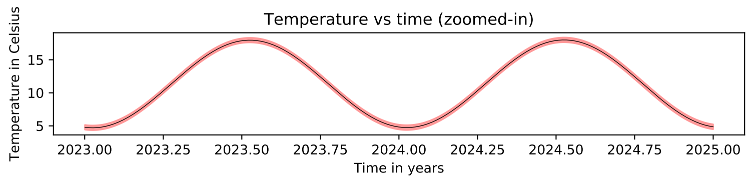
```python
import numpy as np
import pandas
import matplotlib.pyplot as plt
import sklearn.linear_model

# Load the dataset
url = 'https://www.cl.cam.ac.uk/teaching/2021/DataSci/data/climate.csv'
climate = pandas.read_csv(url)
climate = climate.loc[(climate.station == 'Cambridge') & (climate.yyyy >= 1985)].copy()
t = climate.yyyy + (climate.mm-1)/12
temp = (climate.tmin + climate.tmax)/2

# Fit a simple linear model.
X = np.column_stack([np.sin(2 * np.pi * t), np.cos(2 * np.pi * t), t-2000])
model = sklearn.linear_model.LinearRegression()
model.fit(X, temp)
```

```
# Extend the time scale.
tnew = np.linspace(2010, 2025, (2025-202)*12+1)
Xnew = np.column_stack([np.sin(2*np.pi*tnew), np.cos(2*np.pi*tnew), tnew-2000])
tempnew = model.predict(Xnew)

# Step 1. Define function that generates model from samples.
def get_model(temp):
    new_model = sklearn.linear_model.LinearRegression()
    new_model.fit(X, temp)
    return new_model


# Define a function to generate a synthetic copy of the dataset.
pred = model.predict(X)
sigma_hat = np.sqrt(np.mean((pred - temp) ** 2))
def rtemp():
    return np.random.normal(loc=pred, scale=sigma_hat)

# Step 3. Sample the models and for each model make predictions for Xnew.
models_ = [get_model(rtemp()) for _ in range(10000)]
preds = np.column_stack(np.array([cur_model.predict(Xnew) for cur_model in models_]))
# Step 4. For each x in Xnew, compute the quantile of interest.
# Note: It would be more intuitive to do this using a for-loop,
# but it would be slower.
conf_interval = np.quantile(preds, [0.025, 0.975], axis=1)
lows = conf_interval[0, :]
highs = conf_interval[1, :]
```

Optionally, we can also plot the confidence interval and see that it is barely visible.

```
_, ax = plt.subplots(figsize=(8, 2.0))
plt.title("Temperature vs time")
plt.xlabel("Time in years")
plt.ylabel("Temperature in Celsius")
ax.plot(tnew, tempnew, linewidth=0.5, color='black')
ax.fill_between(tnew, lows, highs, alpha=0.4, color='r')
plt.tight_layout()
plt.savefig('ex8_predictions_with_conf.pdf')
plt.show()

_, ax = plt.subplots(figsize=(8, 2.0))
plt.title("Temperature vs time (zoomed-in)")
plt.xlabel("Time in years")
plt.ylabel("Temperature in Celsius")
fltr = tnew > 2023
ax.plot(tnew[fltr], tempnew[fltr], linewidth=0.5, color='black')
ax.fill_between(tnew[fltr], lows[fltr], highs[fltr], alpha=0.4, color='r')
plt.tight_layout()
plt.savefig('ex8_predictions_with_conf_zoomed.pdf')
plt.show()
```





Question 9

There are many such tests. One way is to discretise the CDF and compute the L_1 distance between the empirical distribution and the theoretical Normal CDF. The code below implements this. Of course you can change the L_1 distance to any other distance and you can also change the binning.

```
import numpy as np
import scipy.stats

sigma = 0.23
num_samples = 1000
num_bins = 50
phi_values = [scipy.stats.norm.cdf(i) for i in np.linspace(-2.5, 2.5, num_bins)]
x = np.random.normal(loc=0, scale=sigma, size=num_samples) # np.random.poisson(0.4,
↪  size=num_samples).astype(float)

sigma_hat = np.sqrt(np.mean(x ** 2))


def ell1_distance(samples):
    # This computes how many values are below each i in np.linspace(..)
    # We do this efficiently by first sorting all samples and then
    # counting all values below each threshold.
    samples /= sigma_hat
    np.sort(samples)
    cur = 0
    acc = np.zeros(num_bins)
    sm = 0
    idx = 0
    for i in np.linspace(-2.5, 2.5, num_bins):
        while cur < len(samples) and samples[cur] < i:
            sm += samples[cur]
            cur += 1
        acc[idx] = sm
        idx += 1
    acc /= len(samples)
    return np.linalg.norm((phi_values - acc), ord=1)


distances = np.array([ell1_distance(np.random.normal(loc=0, scale=sigma_hat,
↪  size=num_samples)) for _ in range(1000)])
lo, hi = np.quantile(distances, [0.025, 0.975])
print(f"L1 distance received: {ell1_distance(x)}")
print(f"L1 distance conf interval: [{lo}, {hi}]")
# One possible output:
# L1 distance received: 24.992...
# L1 distance conf interval: [24.917..., 25.252...]
```

As a sanity check, when we set x to be generated from a Poisson distribution Po(0.4), the observed distance between the distributions is outside the confidence interval.

Note: You may want to read more about the Kolmogorov-Smirnov test and about the Glivenko–Cantelli theorem.

Question 10

This question is underspecified and you are required to make several assumptions:

- Where do you start?
- Does your speed change?
- Is it an infinite motorway? Do cars leave the motorway when they reach the end?
- How are the cars initially distributed? Note that the cars that are faster and are in front of you, you will never see. (Same for the slower cars before you)
- Does the count include over our entire stay on the motorway or up to a cut off? For example, if you are on the motorway for only a few seconds, then it might mean that you passed one car and one car passed you.

Note: You may find the following article interesting.