Assessed Coursework 3

CID 01252821

Question 1

Part a

We have $Y \sim Weibull(\theta_1, \theta_2)$ with a probability density function

$$p_{Y|\theta}(y|\theta) = \mathbb{1}_{[0,\infty)}(y)\theta_1\theta_2(\theta_1 y)^{\theta_2-1}e^{-(\theta_1 y)^{\theta_2}},\ \theta_1, \theta_2 \geq 0.$$

or for $\theta_1, \theta_2 \geq 0$:

$$p_{Y|\theta}(y|\theta) = \begin{cases} \theta_1\theta_2(\theta_1 y)^{\theta_2-1}e^{-(\theta_1 y)^{\theta_2}},\ y \geq 0 \\ 0, \text{otherwise.} \end{cases}$$

Therefore, for $y \geq 0$ it follows

$$F_{Y|\theta}(y) = P(Y \leq y|\theta) = \int_0^y \theta_1\theta_2(\theta_1 t)^{\theta_2-1}e^{-(\theta_1 t)^{\theta_2}}dt =$$

$$= \theta_1^{\theta_2}\theta_2 \int_0^y t^{\theta_2-1}e^{-(\theta_1 t)^{\theta_2}}dt = -\int_0^y e^{-(\theta_1 t)^{\theta_2}}d(-(\theta_1 t)^{\theta_2}).$$

Let's substitute $u = -(\theta_1 t)^{\theta_2}$, therefore

$$F_{Y|\theta}(y) = -\int_0^{-(\theta_1 y)^{\theta_2}} e^u du = -e^u|_0^{-(\theta_1 y)^{\theta_2}} = 1 - e^{-(\theta_1 y)^{\theta_2}},\ y \geq 0.$$

Finally,

$$F_{Y|\theta}(y) = \begin{cases} 1 - e^{-(\theta_1 y)^{\theta_2}},\ y \geq 0 \\ 0, \text{otherwise.} \end{cases}$$

For the survivor function we have

$$S_{Y|\theta}(y) = P(Y > y|\theta) = 1 - P(Y \leq y|\theta) = \begin{cases} e^{-(\theta_1 y)^{\theta_2}},\ y \geq 0 \\ 1, \text{otherwise.} \end{cases}$$

Part b

$$E(Y|\theta) = \int_0^\infty S_{Y|\theta}(y)dy = \int_0^\infty e^{-(\theta_1 y)^{\theta_2}}dy.$$

Let's substitute $u = (\theta_1 y)^{\theta_2}$, leading to $y = \frac{1}{\theta_1}u^{\frac{1}{\theta_2}}$, and $dy = \frac{1}{\theta_1}\frac{1}{\theta_2}u^{\frac{1}{\theta_2}-1}dy$. For the integral limits when $y = 0$ it follows $u = 0$, and $\lim_{y\to\infty} u = \infty$, therefore for $y \geq 0$:

$$E(Y|\theta) = \frac{1}{\theta_1}\frac{1}{\theta_2}\int_0^\infty u^{\frac{1}{\theta_2}-1}e^{-u}du = \frac{1}{\theta_1}\frac{1}{\theta_2}\Gamma(\frac{1}{\theta_2}) = \frac{1}{\theta_1}\Gamma(\frac{1}{\theta_2}+1),$$

where $\Gamma(x)$ is the Gamma function.

Part c

Stan model code:

```
stan_model_code = """
data {
  int<lower=0> n;
  vector<lower=0>[n] y;
}

parameters {
  real<lower=0> theta_1;
  real<lower=0> theta_2;
}

transformed parameters {
  real<lower=0> sigma;
  real<lower=0> alpha;
  sigma = 1 / theta_1;
  alpha = theta_2;
}

model {
  theta_1 ~ gamma(1, 0.1);
  theta_2 ~ gamma(1, 0.1);

  for (i in 1:n)
    y[i] ~ weibull(alpha, sigma);
}

generated quantities {
  real posterior_mean;
  posterior_mean = sigma * tgamma(1 + 1 / alpha);
}
"""
```

Part d

```
import numpy as np
import stan
import nest_asyncio
nest_asyncio.apply()
import matplotlib.pyplot as plt


def sample_weibull(n, theta_1 = 2.0, theta_2 = 1.5):
  alpha = theta_2
  sigma = 1 / theta_1
  y = sigma * np.random.default_rng(seed=0).weibull(alpha, n)

  return y, alpha, sigma


n = 1000
y, alpha, sigma = sample_weibull(n)

sm_data = {"n": n, "y": y}
sm = stan.build(stan_model_code, data=sm_data, random_seed=1)
```

```
## Building...
##
##
## Building: found in cache, done.
```

```
chains, samples, burn = 1, 10000, 1000
fit = sm.sample(num_chains=chains, num_samples=samples, num_warmup=burn, save_warmup=False)
```

```
## Sampling:   0%
## Sampling:   0% (1/11000)
## Sampling:   2% (200/11000)
## Sampling:   4% (400/11000)
## Sampling:   5% (600/11000)
## Sampling:   7% (800/11000)
## Sampling: 100% (11000/11000)
## Sampling: 100% (11000/11000), done.
## Messages received during sampling:
##   Gradient evaluation took 0.000176 seconds
##   1000 transitions using 10 leapfrog steps per transition would take 1.76 seconds.
##   Adjust your expectations accordingly!
##   Informational Message: The current Metropolis proposal is about to be rejected because of the follo
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
```

```
fit.to_frame()
```

```
## parameters        lp__  accept_stat__  ...     alpha  posterior_mean
## draws                                  ...
## 0          -102.534108       0.999422  ...  1.549086        0.474744
```

```
## 1          -101.630607       0.993946  ...  1.547200        0.453307
## 2          -101.558822       1.000000  ...  1.542784        0.453658
## 3          -101.482226       1.000000  ...  1.544096        0.455029
## 4          -101.412122       0.933492  ...  1.540670        0.463569
## ...                ...            ...  ...       ...             ...
## 9995       -101.499866       0.999105  ...  1.547005        0.464966
## 9996       -105.185495       0.720552  ...  1.473722        0.435600
## 9997       -106.741993       0.747533  ...  1.485287        0.429294
## 9998       -111.713584       0.793914  ...  1.496196        0.417612
## 9999       -107.559043       1.000000  ...  1.474880        0.427526
##
## [10000 rows x 12 columns]
```

```
mn_theta1 = np.mean(fit["theta_1"])
mn_theta1
```

```
## 1.9600836139572275
```

```
mn_theta2 = np.mean(fit["theta_2"])
mn_theta2
```

```
## 1.53466268476054
```

```
mn_posterior_mean = np.mean(fit["posterior_mean"])
mn_posterior_mean
```

```
## 0.45963801362364404
```

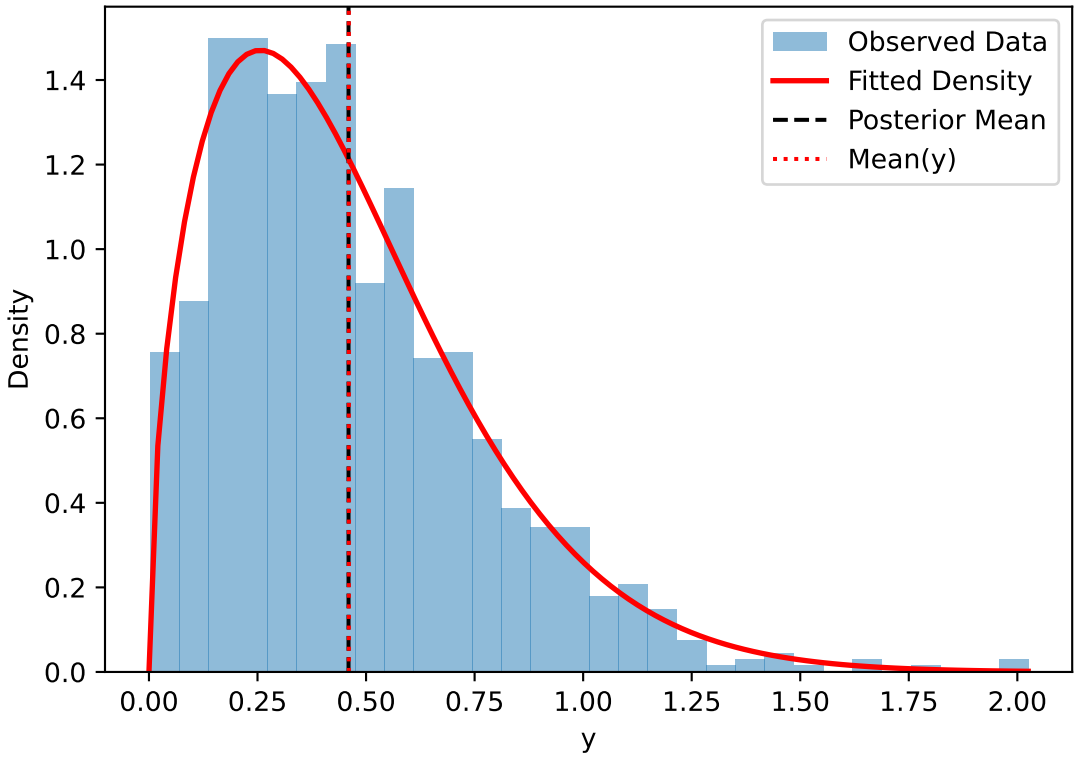
Part e

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import weibull_min

# histogram of observed data
plt.hist(y, bins=30, density=True, alpha=0.5, label='Observed Data');

# fitted density curve using the estimated parameters
x = np.linspace(0, np.max(y), 100)
plt.plot(x, weibull_min.pdf(x, mn_theta2, scale=1/mn_theta1), 'r-', lw=2, label='Fitted Density')
# plot posterior mean
plt.axvline(mn_posterior_mean, color="black", linestyle="--", label="Posterior Mean")
# add mean(y)
plt.axvline(np.mean(y), color="red", linestyle="dotted", label="Mean(y)")


plt.xlabel('y')
plt.ylabel('Density')
plt.legend()
plt.show()
```



Visually the fitted curve seems to be following the observed data really well. There are neither significant underfitting parts, nor overfitting parts. Also, the posterior mean seems to be really close to the observed data mean ($mean(y)$).

Part f

The calculated density $p_{Y|\theta}(y|\hat{m}_n)$ is not the correct way to estimate the predictive distribution $p_{Y|\curvearrowright}(y|\curvearrowright)$ for a new event time from the same population as $\curvearrowright$, which is based on the estimated parameters $\hat{m}_n$. The estimated density $p_{Y|\theta}(y|\hat{m}_n))$ represents the density of the Weibull distribution based on the estimated parameters θ_1 and θ_2, which does not take into account the additional uncertainty arising from the prediction of a new event time.

One alternative procedure is to use posterior predictive simulation, which involves the following steps:

- Sample parameter values from the estimated posterior distribution based on the observed data
- For each sample, generate a new event time from the Weibull distribution using the sampled parameters
- Collect the generated new event times from all the parameter sets to obtain a set of samples representing the predictive distribution.
- Use the set of simulated event times to estimate the predictive distribution

Question 2

```
import pandas as pd

df = pd.read_csv("cc.csv")
df.head()
```

```
##    id  study       rx  sex  age  ...  extent  surg  node4  time  etype
## 0   1      1  Lev+5FU    1   43  ...       3     0      1  1521      2
## 1   1      1  Lev+5FU    1   43  ...       3     0      1   968      1
## 2   2      1  Lev+5FU    1   63  ...       3     0      0  3087      2
## 3   2      1  Lev+5FU    1   63  ...       3     0      0  3087      1
## 4   3      1      Obs    0   71  ...       2     0      1   963      2
##
## [5 rows x 16 columns]
```

Part a

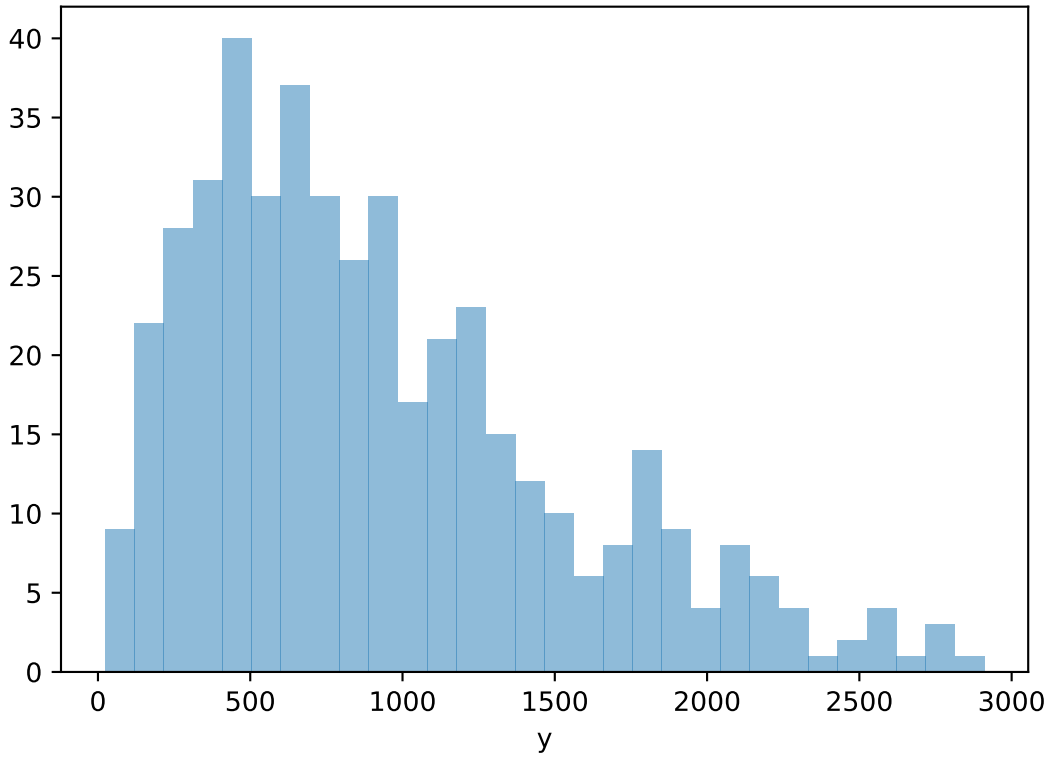
```
uncensored_death_df = df[(df["status"] == 1) & (df["etype"] == 2)]
uncensored_death_df.head()
```

```
##     id  study       rx  sex  age  ...  extent  surg  node4  time  etype
## 0    1      1  Lev+5FU    1   43  ...       3     0      1  1521      2
## 4    3      1      Obs    0   71  ...       2     0      1   963      2
## 6    4      1  Lev+5FU    0   66  ...       3     1      1   293      2
## 8    5      1      Obs    1   69  ...       3     1      1   659      2
## 10   6      1  Lev+5FU    0   57  ...       3     0      1  1767      2
##
## [5 rows x 16 columns]
```

```
y = uncensored_death_df["time"].values
len(y)
```

```
## 452
```

```
plt.hist(y, bins=30, density=False, alpha=0.5);
plt.xlabel('y')
plt.show()
```



Part b

```
new_sm_data = {"n": len(y), "y": y}
new_sm = stan.build(stan_model_code, data=new_sm_data, random_seed=0)
```

```
## Building...
##
##
## Building: found in cache, done.
```

```
chains, samples, burn = 1, 10_000, 10_000
new_fit = new_sm.sample(num_chains=chains, num_samples=samples, num_warmup=burn, save_warmup=False)
```

```
## Sampling:   0%
## Sampling:   0% (1/20000)
## Sampling:   2% (300/20000)
## Sampling:   4% (700/20000)
## Sampling:   6% (1100/20000)
## Sampling:   8% (1600/20000)
## Sampling:  10% (2000/20000)
## Sampling:  12% (2500/20000)
## Sampling:  16% (3100/20000)
## Sampling:  18% (3500/20000)
```

```
## Sampling:  20% (4000/20000)
## Sampling:  22% (4500/20000)
## Sampling:  25% (5000/20000)
## Sampling:  28% (5500/20000)
## Sampling:  30% (6000/20000)
## Sampling:  32% (6500/20000)
## Sampling:  36% (7100/20000)
## Sampling:  38% (7700/20000)
## Sampling:  42% (8300/20000)
## Sampling:  44% (8800/20000)
## Sampling:  46% (9300/20000)
## Sampling:  49% (9800/20000)
## Sampling: 100% (20000/20000)
## Sampling: 100% (20000/20000), done.
## Messages received during sampling:
##   Gradient evaluation took 7.4e-05 seconds
##   1000 transitions using 10 leapfrog steps per transition would take 0.74 seconds.
##   Adjust your expectations accordingly!
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
```

```
##   Informational Message: The current Metropolis proposal is about to be rejected because of the follo
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
```

```
new_fit_df = new_fit.to_frame()
new_fit_df[["theta_1", "theta_2", "sigma", "alpha"]]
```

```
## parameters   theta_1   theta_2        sigma     alpha
## draws
## 0           0.000948  1.594670  1054.880790  1.594670
## 1           0.000945  1.598795  1057.822186  1.598795
## 2           0.000945  1.598795  1057.822186  1.598795
## 3           0.000941  1.605179  1062.789497  1.605179
## 4           0.000941  1.619678  1062.300106  1.619678
## ...              ...       ...          ...       ...
## 9995        0.000978  1.592189  1022.545425  1.592189
## 9996        0.000978  1.592189  1022.545425  1.592189
## 9997        0.000973  1.656674  1027.476569  1.656674
## 9998        0.000900  1.646108  1110.519916  1.646108
## 9999        0.000895  1.630942  1117.612124  1.630942
##
## [10000 rows x 4 columns]
```

```
new_mn_theta1 = np.mean(new_fit["theta_1"])
new_mn_theta1
```

```
## 0.0009674112431744903
```

```
new_mn_theta2 = np.mean(new_fit["theta_2"])
new_mn_theta2
```

```
## 1.5804859206434394
```

```
new_mn_posterior_mean = np.mean(new_fit["posterior_mean"])
new_mn_posterior_mean
```

```
## 929.1010219047613
```

Part c

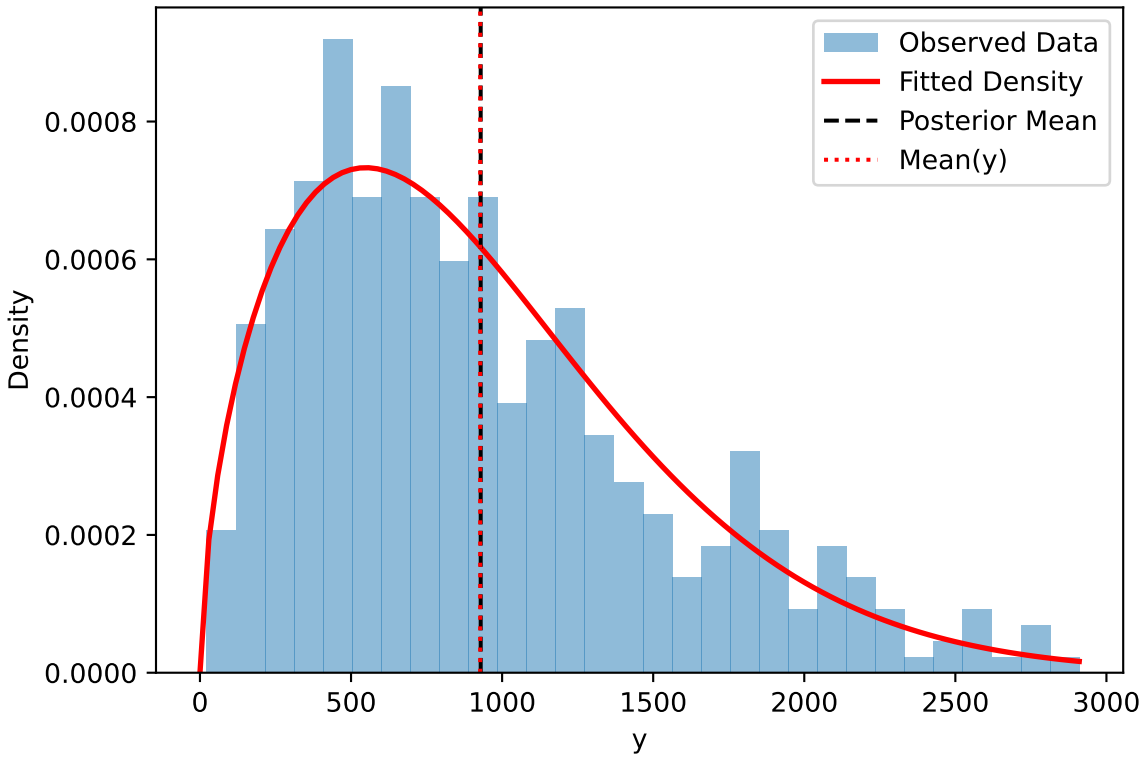
```
# histogram of observed data
plt.hist(y, bins=30, density=True, alpha=0.5, label='Observed Data');

# ffitted density curve using the estimated parameters
x = np.linspace(0, np.max(y), 100)
plt.plot(x, weibull_min.pdf(x, new_mn_theta2, scale=1/new_mn_theta1), 'r-', lw=2, label='Fitted Density
# add the posterior mean
plt.axvline(x=new_mn_posterior_mean, color="black", linestyle="--", label="Posterior Mean")
```

```
# add mean(y)
plt.axvline(x=np.mean(y), color="red", linestyle="dotted", label="Mean(y)")


# Set plot labels and legend
plt.xlabel('y')
plt.ylabel('Density')
plt.legend()

# Show the plot
plt.show()
```



Visually the fitted curve seems to be following the observed data, but around the peak we can see there is slightly underfitting for two of the bins, and there is overfitting in the range for $y \in (1000, 2000)$. The posterior mean seems to be really close to the observed data mean.

Part d

```
new_stan_model_code = """
data {
  int<lower=0> n;
  vector<lower=0>[n] y;
}
```

```
parameters {
  real<lower=0> theta_1;
  real<lower=0> theta_2;
}

transformed parameters {
  real<lower=0> sigma;
  real<lower=0> alpha;
  sigma = 1 / theta_1;
  alpha = theta_2;
}

model {
  theta_1 ~ gamma(1, 0.1);
  theta_2 ~ gamma(1, 0.1);

  for (i in 1:n)
    y[i] ~ weibull(alpha, sigma);
}

generated quantities {
  vector<lower=0>[n] y_rep;
  real posterior_mean;
  real test_statistic;
  real test_statistic_rep;
  real ppp;

  for (i in 1:n)
    y_rep[i] = weibull_rng(alpha, sigma);

  posterior_mean = sigma * tgamma(1 + 1 / alpha);
  test_statistic = pow(mean(y) - posterior_mean, 2);
  test_statistic_rep = pow(mean(y_rep) - posterior_mean, 2);

  ppp = test_statistic_rep >= test_statistic ? 1 : 0;
}
"""
```

```
new_sm_2 = stan.build(new_stan_model_code, data=new_sm_data, random_seed=0)
```

```
## Building...
##
##
## Building: found in cache, done.
```

```
chains, samples, burn = 1, 10_000, 10_000
new_fit_2 = new_sm_2.sample(num_chains=chains, num_samples=samples, num_warmup=burn, save_warmup=False)
```

```
## Sampling:   0%
## Sampling:   1% (200/20000)
## Sampling:   3% (600/20000)
## Sampling:   5% (1000/20000)
```

```
## Sampling:   8% (1500/20000)
## Sampling:  10% (1900/20000)
## Sampling:  12% (2400/20000)
## Sampling:  15% (3000/20000)
## Sampling:  17% (3400/20000)
## Sampling:  19% (3800/20000)
## Sampling:  22% (4300/20000)
## Sampling:  24% (4900/20000)
## Sampling:  27% (5400/20000)
## Sampling:  30% (5900/20000)
## Sampling:  32% (6500/20000)
## Sampling:  35% (7000/20000)
## Sampling:  38% (7500/20000)
## Sampling:  40% (8100/20000)
## Sampling:  43% (8600/20000)
## Sampling:  46% (9200/20000)
## Sampling:  48% (9700/20000)
## Sampling: 100% (20000/20000)
## Sampling: 100% (20000/20000), done.
## Messages received during sampling:
##   Gradient evaluation took 9.5e-05 seconds
##   1000 transitions using 10 leapfrog steps per transition would take 0.95 seconds.
##   Adjust your expectations accordingly!
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspec
##   Informational Message: The current Metropolis proposal is about to be rejected because of the foll
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
```

```
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the follo
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
##   Informational Message: The current Metropolis proposal is about to be rejected because of the follo
##   Exception: gamma_lpdf: Random variable is 0, but must be positive finite! (in '/var/folders/r_/15x
##   If this warning occurs sporadically, such as for highly constrained variable types like covariance
##   but if this warning occurs often then your model may be either severely ill-conditioned or misspeci
```

```
np.mean(new_fit_2["ppp"])
```

```
## 0.4977
```

The posterior predictive p-value is ~ 0.4977 implying the observed data is not significantly different from the model predictions. We don't expect to have predictions which are too high (observed for a value > 0.5), and given that $0.4977 \sim 0.5$, there is not high possibility of predictions on the lower side (observed for a value < 0.5).

Part e

```
theta_samples = np.vstack((new_fit_2.to_frame()["theta_1"].values, new_fit_2.to_frame()["theta_2"].valu
mu_hat = np.mean(theta_samples, axis=1)
print(f"Sample means of (theta_1, theta_2): {mu_hat}")
```

```
## Sample means of (theta_1, theta_2): [9.67945641e-04 1.57990656e+00]
```

```
sigma_hat = np.cov(theta_samples)
print(f"Covariance:\n {sigma_hat}")
```

```
## Covariance:
##  [[ 9.31624279e-10 -5.88269568e-07]
##  [-5.88269568e-07  3.36606272e-03]]
```

Part f

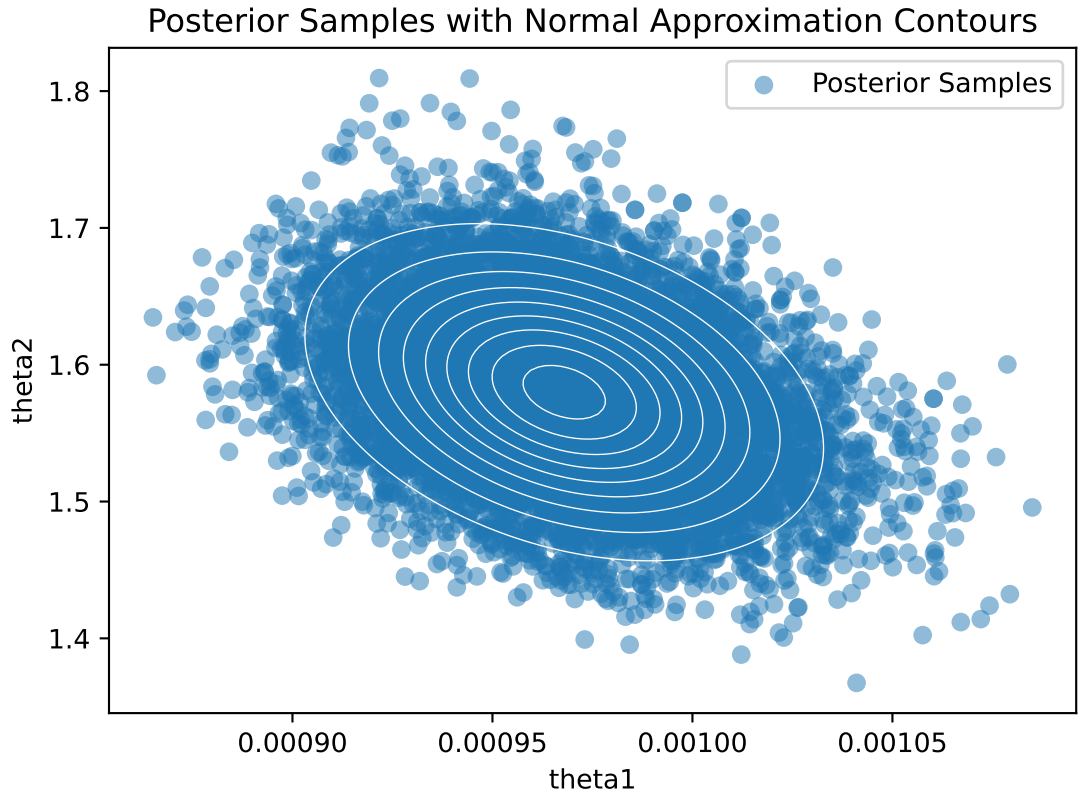
```
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# scatter plot of posterior samples
plt.scatter(theta_samples[0], theta_samples[1], alpha=0.5, label='Posterior Samples');

# grid of points for contour plot
theta1_range = np.linspace(mu_hat[0] - 3 * np.sqrt(sigma_hat[0, 0]), mu_hat[0] + 3 * np.sqrt(sigma_hat[
theta2_range = np.linspace(mu_hat[1] - 3 * np.sqrt(sigma_hat[1, 1]), mu_hat[1] + 3 * np.sqrt(sigma_hat[
theta1_grid, theta2_grid = np.meshgrid(theta1_range, theta2_range)
pos = np.dstack((theta1_grid, theta2_grid))
```

```
# contour plot
pdf_values = multivariate_normal.pdf(pos, mean=mu_hat, cov=sigma_hat)
plt.contour(theta1_grid, theta2_grid, pdf_values, levels=10, colors='white', linewidths=0.5);


plt.xlabel('theta1')
plt.ylabel('theta2')
plt.legend()
plt.title('Posterior Samples with Normal Approximation Contours')
plt.show()
```



We have eliptical contours, with the contours being elongated along the second diagonal (the line $\theta_2 = -\theta_1$), and a negative correlation. Visually the fit seems to be fitting the data points very well, the shape is following the collective data points shape, and also the contours are more concentrated in the middle where the data points seem to be more dense.

Part g

The likelihood is

$$p_M(y|\theta_1,\theta_2) = \prod_{i=1}^{n} \theta_1\theta_2(\theta_1 y_i)^{\theta_2-1} e^{-(\theta_1 y_i)^{\theta_2}} = \theta_1^{n\theta_2}\theta_2^n e^{-\theta_1^{\theta_2}\sum_{i=1}^n y_i^{\theta_2}} \prod_{i=1}^{n} y_i^{\theta_2-1},\ y \geq 0,$$

$p_M(\theta_M)$ is the joint distribution of θ_1 and θ_2, which are iid with

$$p_{\theta_1}(x) = p_{\theta_2}(x) = 0.1e^{-0.1x},$$

$$p_M(\theta_M) = p_{\theta_1}(\theta_M)p_{\theta_2}(\theta_M) = 0.01e^{-0.2\theta_M}.$$

```
import decimal
from scipy.stats import multivariate_normal, loggamma

num_samples = 1000

def log_likelihood(theta_1, theta_2, y):
    n = len(y)

    term_1 = n * theta_2 * np.log(theta_1)
    term_2 = n * np.log(theta_2)
    term_3 = -np.power(theta_1, theta_2)*(np.sum(np.power(y, theta_2)))
    term_4 = (theta_2 - 1) * np.sum(np.log(y))

    return term_1 + term_2 + term_3 + term_4

log_weights = []
for _ in range(num_samples):
    sampled_theta = multivariate_normal.rvs(mean=mu_hat, cov=sigma_hat)
    sampling_density = multivariate_normal.pdf(sampled_theta, mean=mu_hat, cov=sigma_hat)

    curr_res = log_likelihood(sampled_theta[0], sampled_theta[1], y)
    curr_res += loggamma.pdf(sampled_theta[0], 1, scale = 1/0.1)
    curr_res += loggamma.pdf(sampled_theta[1], 1, scale = 1/0.1)
    curr_res -= np.log(sampling_density)

    log_weights.append(curr_res)

np.mean(log_weights)
```

```
## -3488.7135436724298
```

```
np.mean(np.exp(log_weights))
```

```
## 0.0
```

```
decimal.Decimal(np.mean(log_weights)).exp()
```

```
## Decimal('7.42949067471470105863085993 6E-1516')
```