homework-03

July 8, 2025

1 Homework 3

1.1 References

- Lectures 8-12 (inclusive).

1.2 Instructions

- Type your name and email in the "Student details" section below.
- Develop the code and generate the figures you need to solve the problems using this notebook.
- For the answers that require a mathematical proof or derivation you should type them using latex. If you have never written latex before and you find it exceedingly difficult, we will likely accept handwritten solutions.
- The total homework points are 100. Please note that the problems are not weighed equally.

[ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
import seaborn as sns
sns.set_context("paper")
sns.set_style("ticks")

import scipy
import numpy as np
import scipy.stats as st
import urllib.request
import os

def download(
    url : str,
    local_filename : str = None
):
    """Download a file from a url.

    Arguments
    url            -- The url we want to download.
    local_filename -- The filemame to write on. If not
                      specified
    """
    if local_filename is None:
        local_filename = os.path.basename(url)
    urllib.request.urlretrieve(url, local_filename)
```

1.3 Student details

- **First Name:** Hannah
- **Last Name:** Moskios
- **Email:** hmoskios@purdue.edu

1.4 Problem 1 - Propagating uncertainty through a differential equation

This is a classic uncertainty propagation problem you must solve using Monte Carlo sampling. Consider the following stochastic harmonic oscillator:

$$\begin{array}{ccc}
\ddot{y} + 2\zeta\omega(X)\dot{y} + \omega^2(X)y & = & 0,\\
y(0) & = & y_0(X),\\
\dot{y}(0) & = & v_0(X),
\end{array}$$

where: + $X = (X_1, X_2, X_3)$, + $X_i \sim N(0,1)$, + $\omega(X) = 2\pi + X_1$, + $\zeta = 0.01$, + $y_0(X) = 1 + 0.1X_2$, and + $v_0 = 0.1X_3$.

In other words, this stochastic harmonic oscillator has an uncertain natural frequency and uncertain initial conditions.

Our goal is to propagate uncertainty through this dynamical system, i.e., estimate the mean and variance of its solution. A solver for this dynamical system is given below:

```
[ ]: class Solver(object):
         def __init__(
             self,
             nt=100,
             T= 5
         ):
             """This is the initializer of the class.

             Arguments:
                 nt -- The number of timesteps.
                 T  -- The final time.
             """
             self.nt = nt
             self.T = T
             # The timesteps on which we will get the solution
             self.t = np.linspace(0, T, nt)
             # The number of inputs the class accepts
             self.num_input = 3
             # The number of outputs the class returns
```

```
        self.num_output = nt

    def __call__(self, x):
        """This special class method emulates a function call.

        Arguments:
            x -- A 1D numpy array with 3 elements.
                 This represents the stochastic input x = (x1, x2, x3).

        Returns the solution to the differential equation evaluated
        at discrete timesteps.
        """
        # uncertain quantities
        x1 = x[0]
        x2 = x[1]
        x3 = x[2]

        # ODE parameters
        omega = 2*np.pi + x1
        y10 = 1 + 0.1*x2
        y20 = 0.1*x3
        # initial conditions
        y0 = np.array([y10, y20])

        # coefficient matrix
        zeta = 0.01
        # spring constant
        k = omega**2
        # damping coeff
        c = 2*zeta*omega
        C = np.array([[0, 1],[-k, -c]])

        #RHS of the ODE system
        def rhs(y, t):
            return np.dot(C, y)

        y = scipy.integrate.odeint(rhs, y0, self.t)

        return y
```

First, let's demonstrate how the solver works:

[ ]:
```
solver = Solver()

x = np.random.randn(solver.num_input)

y = solver(x)
```

```
print(y)
```

```
[[ 9.42266867e-01  2.30322192e-02]
 [ 9.09367642e-01 -1.31671642e+00]
 [ 8.10991489e-01 -2.55418964e+00]
 [ 6.54593374e-01 -3.60059553e+00]
 [ 4.51777182e-01 -4.38141477e+00]
 [ 2.17435611e-01 -4.84170393e+00]
 [-3.13427891e-02 -4.94995681e+00]
 [-2.76519151e-01 -4.70025270e+00]
 [-5.00411760e-01 -4.11254653e+00]
 [-6.86970710e-01 -3.23108862e+00]
 [-8.22927882e-01 -2.12109924e+00]
 [-8.98739906e-01 -8.63946985e-01]
 [-9.09257111e-01  4.48812797e-01]
 [-8.54071660e-01  1.72210287e+00]
 [-7.37521481e-01  2.86420740e+00]
 [-5.68351524e-01  3.79337553e+00]
 [-3.59058683e-01  4.44366992e+00]
 [-1.24969346e-01  4.76964218e+00]
 [ 1.16882351e-01  4.74950376e+00]
 [ 3.48993570e-01  4.38657174e+00]
 [ 5.54658890e-01  3.70889410e+00]
 [ 7.19172160e-01  2.76709171e+00]
 [ 8.30877464e-01  1.63058208e+00]
 [ 8.81994231e-01  3.82466407e-01]
 [ 8.69158108e-01 -8.86543773e-01]
 [ 7.93640087e-01 -2.08471670e+00]
 [ 6.61229792e-01 -3.12592865e+00]
 [ 4.81793155e-01 -3.93585171e+00]
 [ 2.68538193e-01 -4.45725650e+00]
 [ 3.70434142e-02 -4.65405206e+00]
 [-1.95879668e-01 -4.51377761e+00]
 [-4.13406919e-01 -4.04837508e+00]
 [-5.99916311e-01 -3.29319546e+00]
 [-7.42109294e-01 -2.30432067e+00]
 [-8.29958459e-01 -1.15440291e+00]
 [-8.57413851e-01  7.26697973e-02]
 [-8.22818361e-01  1.28789061e+00]
 [-7.29003716e-01  2.40358748e+00]
 [-5.83061868e-01  3.33974569e+00]
 [-3.95810046e-01  4.02974346e+00]
 [-1.80989731e-01  4.42508978e+00]
 [ 4.57412026e-02  4.49882780e+00]
 [ 2.67949245e-01  4.24736502e+00]
 [ 4.69615903e-01  3.69060698e+00]
 [ 6.36291921e-01  2.87039356e+00]
 [ 7.56132072e-01  1.84736026e+00]
 [ 8.20736320e-01  6.96459247e-01]
 [ 8.25737738e-01 -4.98529055e-01]
 [ 7.71096079e-01 -1.65109222e+00]
 [ 6.61077388e-01 -2.67824651e+00]
 [ 5.03922829e-01 -3.50650907e+00]
 [ 3.11232293e-01 -4.07715210e+00]
 [ 9.71088347e-02 -4.35036341e+00]
 [-1.22873004e-01 -4.30801890e+00]
 [-3.32799578e-01 -3.95487441e+00]
 [-5.17568837e-01 -3.31809904e+00]
 [-6.63976337e-01 -2.44519316e+00]
 [-7.61658331e-01 -1.40044943e+00]
 [-8.03824720e-01 -2.60220661e-01]
 [-7.87730037e-01  8.92657535e-01]
 [-7.14849896e-01  1.97488221e+00]
 [-5.90751730e-01  2.90869983e+00]
 [-4.24670816e-01  3.62749002e+00]
 [-2.28823771e-01  4.08051545e+00]
 [-1.75104776e-02  4.23649946e+00]
 [ 1.93929524e-01  4.08577938e+00]
 [ 3.90230143e-01  3.64088782e+00]
 [ 5.57300456e-01  2.93552830e+00]
 [ 6.83235760e-01  2.02202807e+00]
 [ 7.59165279e-01  9.67459948e-01]
 [ 7.79876277e-01 -1.51279584e-01]
 [ 7.44170713e-01 -1.25307495e+00]
 [ 6.54930127e-01 -2.25847056e+00]
 [ 5.18885640e-01 -3.09539832e+00]
 [ 3.46111264e-01 -3.70434104e+00]
 [ 1.49278584e-01 -4.04256188e+00]
 [-5.72720743e-02 -4.08709944e+00]
 [-2.58575987e-01 -3.83631915e+00]
 [-4.40127754e-01 -3.30991636e+00]
 [-5.88926130e-01 -2.54737968e+00]
 [-6.94404564e-01 -1.60503342e+00]
 [-7.49180974e-01 -5.51880490e-01]
 [-7.49573572e-01  5.35447696e-01]
 [-6.95846759e-01  1.57826485e+00]
 [-5.92170799e-01  2.50152185e+00]
 [-4.46299705e-01  3.23920547e+00]
 [-2.68992137e-01  3.73905578e+00]
 [-7.32184706e-02  3.96626513e+00]
 [ 1.26787529e-01  3.90589721e+00]
 [ 3.16563174e-01  3.56385848e+00]
 [ 4.82462204e-01  2.96635944e+00]
 [ 6.12635661e-01  2.15791317e+00]
```

```
 [ 6.97877666e-01  1.19802293e+00]
 [ 7.32275830e-01  1.56805581e-01]
 [ 7.13620385e-01 -8.90127872e-01]
 [ 6.43543794e-01 -1.86716074e+00]
 [ 5.27382206e-01 -2.70413000e+00]
 [ 3.73770267e-01 -3.34136259e+00]
 [ 1.93999963e-01 -3.73392712e+00]
 [ 1.19106005e-03 -3.85479848e+00]]
```

Notice the dimension of y:

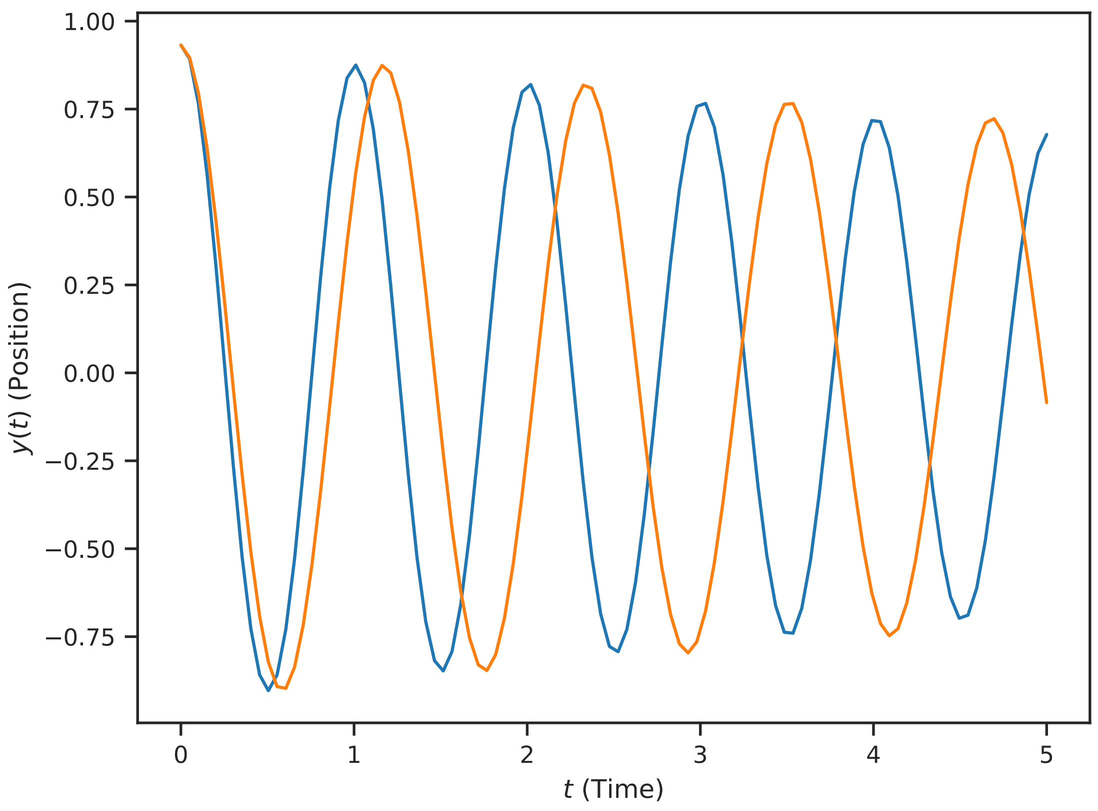
```
[ ]: y.shape
```

```
[ ]: (100, 2)
```

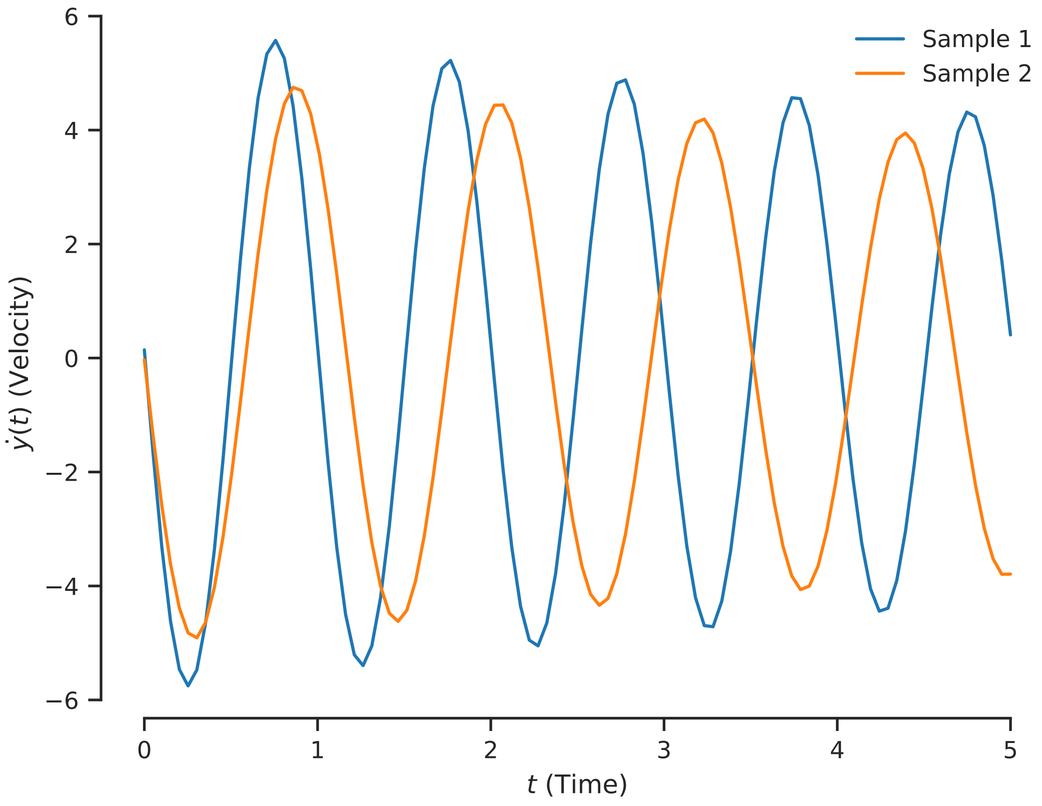
The 100 rows corresponds to timesteps. The 2 columns correspond to position and velocity.

Let's plot a few samples:

```
[ ]: fig1, ax1 = plt.subplots()
     ax1.set_xlabel('$t$ (Time)')
     ax1.set_ylabel('$y(t)$ (Position)')

     fig2, ax2 = plt.subplots()
     ax2.set_xlabel('$t$ (Time)')
     ax2.set_ylabel('$\dot{y}(t)$ (Velocity)')

     for i in range(2):
         x = np.random.randn(solver.num_input)
         y = solver(x)

         ax1.plot(solver.t, y[:, 0])
         ax2.plot(
             solver.t, y[:, 1],
             label=f'Sample {i+1:d}')
     plt.legend(loc="best", frameon=False)
     sns.despine(trim=True);
```



1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
$y(t)$ (Position)
0
1
2
3
4
5
t (Time)



For your convenience, here is code that takes many samples of the solver at once:

```
[ ]: def take_samples_from_solver(num_samples):
         """Takes ``num_samples`` from the ODE solver.

         Returns them in an array of the form:
         ``num_samples x 100 x 2``
         (100 timesteps, 2 states (position, velocity))
         """
         samples = np.ndarray((num_samples, 100, 2))
         for i in range(num_samples):
             samples[i, :, :] = solver(
                 np.random.randn(solver.num_input)
             )
         return samples
```

It works like this:

```
[ ]: samples = take_samples_from_solver(50)
     print(samples.shape)
```

```
(50, 100, 2)
```

Here, the first dimension corresponds to different samples. Then we have timesteps. And finally, we have either position or velocity.

As an example, the velocity of the 25th sample at the first ten timesteps is:

```
[ ]: samples[24, :10, 1]
```

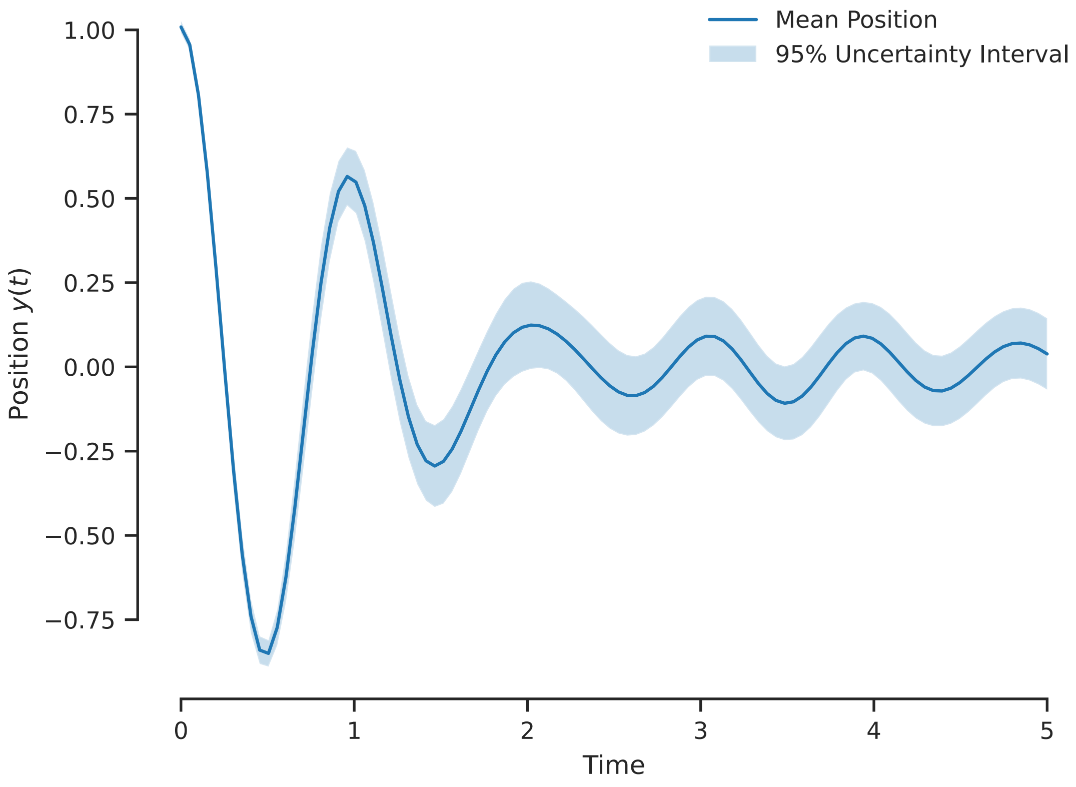
```
[ ]: array([-2.12558319e-03, -1.08466686e+00, -2.10717170e+00, -3.01792217e+00,
            -3.77111592e+00, -4.32916831e+00, -4.66458627e+00, -4.76132084e+00,
            -4.61553079e+00, -4.23572369e+00])
```

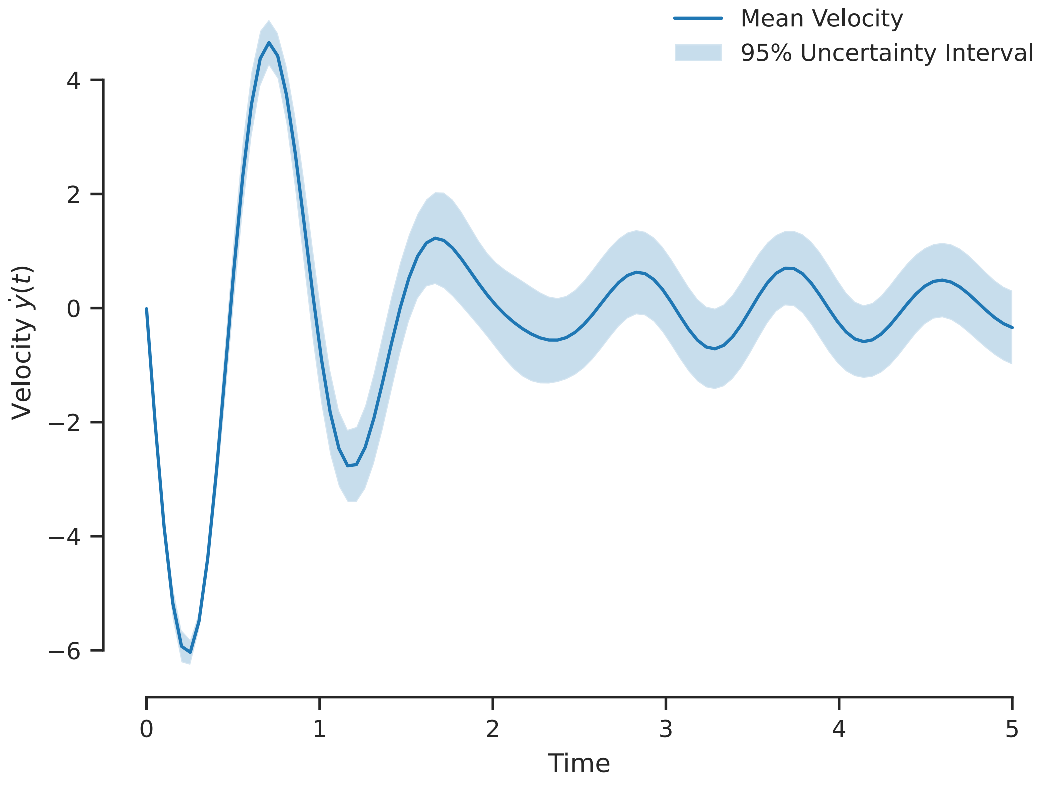
1.4.1 Part A

Take 100 samples of the solver output and plot the estimated mean position and velocity as a function of time along with a 95% epistemic uncertainty interval around it. This interval captures how sure you are about the mean response when using only 100 Monte Carlo samples. You need to use the central limit theorem to find it (see the lecture notes).

```
[ ]: samples = take_samples_from_solver(100)
     # Sampled positions are: samples[:, :, 0]
     # Sampled velocities are: samples[:, :, 1]
     # Sampled position at the 10th timestep is: samples[:, 9, 0]
     # etc.

     # Your code here
     sh = samples.shape
     N = sh[0]

     # Mean position and velocity
     mean_pos = np.mean(samples[:, :, 0], axis=0)
     mean_vel = np.mean(samples[:, :, 1], axis=0)

     # Variance of position and velocity
     var_pos = np.var(samples[:, :, 0], axis=0)
     var_vel = np.var(samples[:, :, 1], axis=0)

     # Use CLT to find the upper/lower bounds for the 95% uncertainty interval
     lb_pos = mean_pos - 2 * np.sqrt(var_pos / N)
     ub_pos = mean_pos + 2 * np.sqrt(var_pos / N)
     lb_vel = mean_vel - 2 * np.sqrt(var_vel / N)
     ub_vel = mean_vel + 2 * np.sqrt(var_vel / N)

     # Plot average position vs. time
     fig1, ax1 = plt.subplots()
     ax1.plot(solver.t, mean_pos, label='Mean Position')
     ax1.fill_between(solver.t, lb_pos, ub_pos, alpha=0.25,
                      label='95% Uncertainty Interval')
     ax1.set_xlabel('Time')
```

```
ax1.set_ylabel('Position $y(t)$')
ax1.legend(loc="best", frameon=False)
sns.despine(trim=True)

# Plot average velocity vs. time
fig2, ax2 = plt.subplots()
ax2.plot(solver.t, mean_vel, label='Mean Velocity')
ax2.fill_between(solver.t, lb_vel, ub_vel, alpha=0.25,
                 label='95% Uncertainty Interval')
ax2.set_xlabel('Time')
ax2.set_ylabel('Velocity $\dot{y}(t)$')
ax2.legend(loc="best", frameon=False)
sns.despine(trim=True)
```





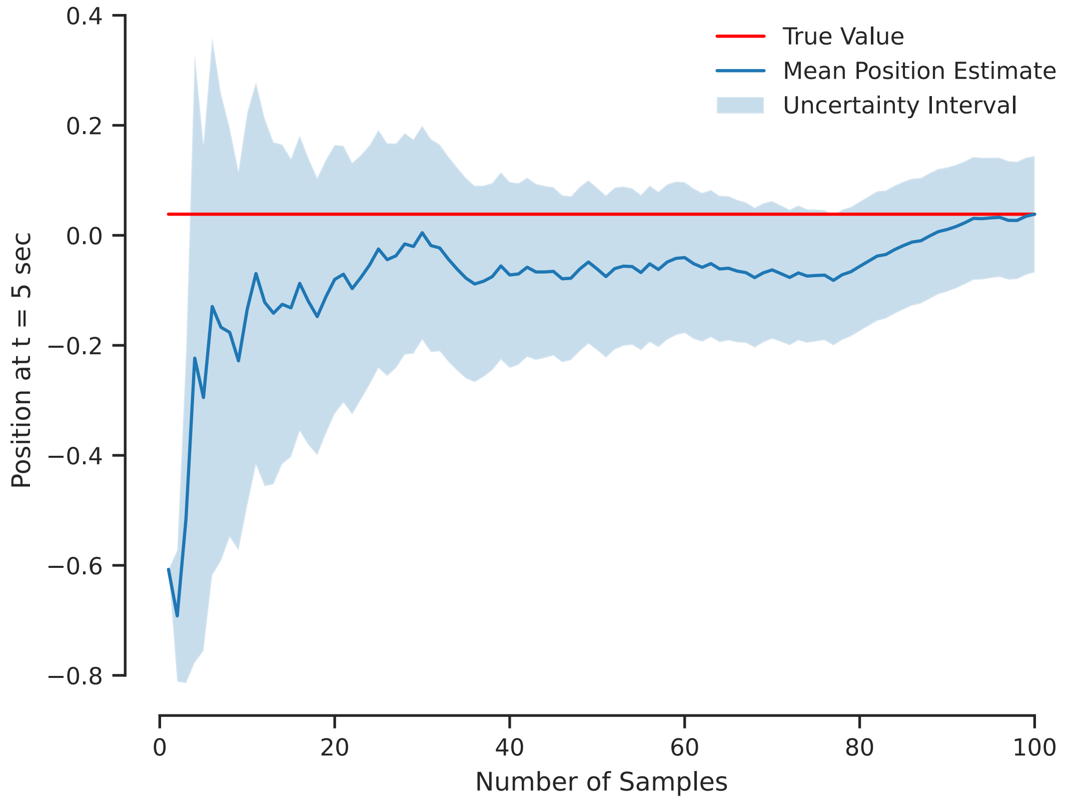
1.4.2 Part B

Plot the epistemic uncertainty about the mean position at $t = 5$s as a function of the number of samples.

Solution:

```
[ ]: # Your code here
idx = np.where(solver.t == 5)[0]

# Find the positions at t = 5 seconds
pos_t5 = samples[:, idx, 0]

# Find the positions for all sample sizes
pos_running = np.cumsum(pos_t5) / np.arange(1, N + 1)

# Find the square of the positions for all sample sizes
pos2_running = np.cumsum(pos_t5 ** 2) / np.arange(1, N + 1)

# Find the variance of the positions for all sample sizes
var_pos_running = pos2_running - pos_running ** 2
```

```python
# Find the lower/upper bounds for the epistemic uncertainty interval
lb_pos_running = pos_running - 2 * np.sqrt(var_pos_running / np.arange(1, N+1))
ub_pos_running = pos_running + 2 * np.sqrt(var_pos_running / np.arange(1, N+1))

# Plot the mean position vs. the number of samples for t = 5 seconds
fig1, ax1 = plt.subplots()
ax1.plot(np.arange(1, N+1), mean_pos[idx] * np.ones(N), 'r', label='True Value')
ax1.plot(np.arange(1, N+1), pos_running, label='Mean Position Estimate')
ax1.fill_between(np.arange(1, N + 1), lb_pos_running, ub_pos_running,
                 alpha=0.25, label='Uncertainty Interval')
ax1.set_xlabel('Number of Samples')
ax1.set_ylabel('Position at t = 5 sec')
ax1.legend(loc="best", frameon=False)
sns.despine(trim=True)
```



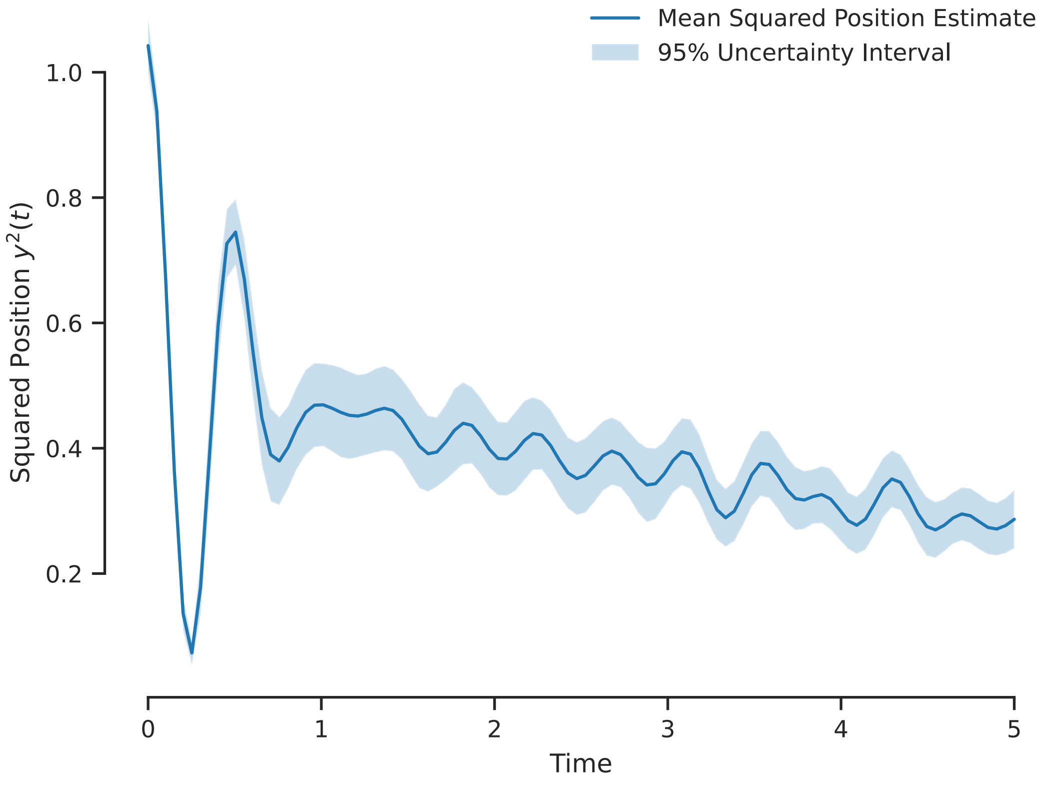
1.4.3 Part C

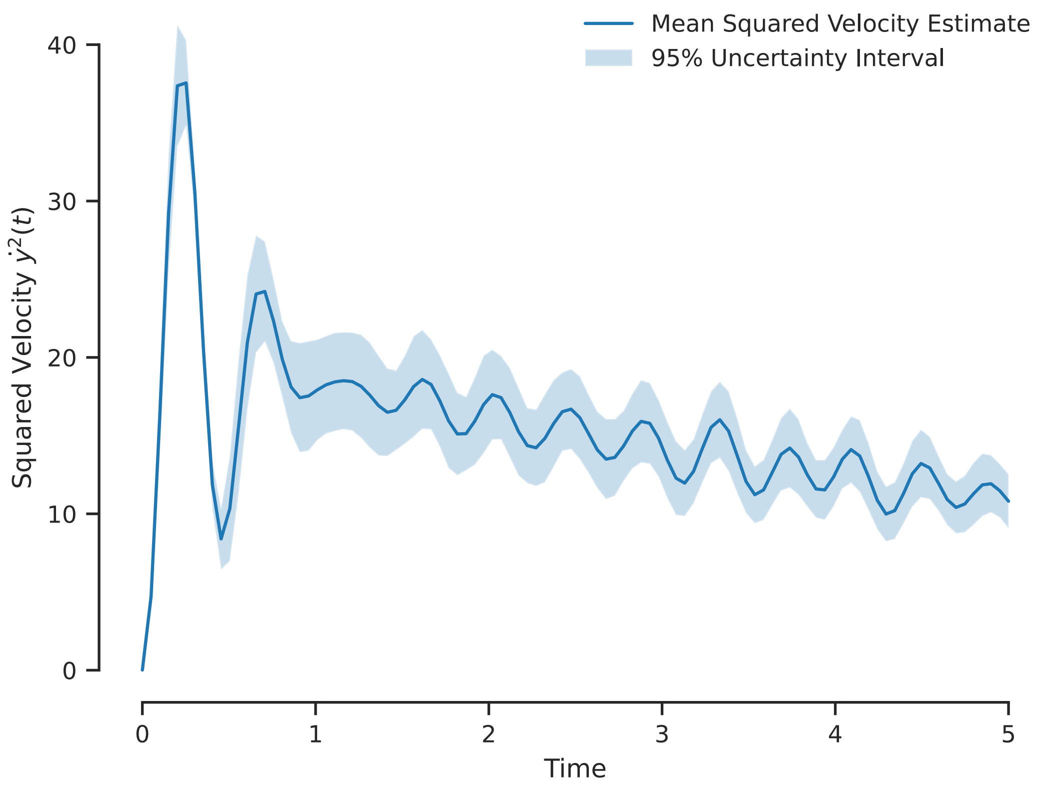
Repeat parts A and B for the squared response. That is, do the same thing as above, but consider $y^2(t)$ and $\dot{y}^2(t)$ instead of $y(t)$ and $\dot{y}(t)$. How many samples do you need to estimate the mean squared response at $t = 5$s with negligible epistemic uncertainty?

Solution:

Repeat Part A for the Squared Response:

```
[ ]: # Your code here
     samples = take_samples_from_solver(100)
     sh = samples.shape
     N = sh[0]

     # Part A for the squared response
     # Squared position and velocity
     sq_pos = samples[:, :, 0] ** 2
     sq_vel = samples[:, :, 1] ** 2

     # Mean squared position and velocity
     mean_sq_pos = np.mean(sq_pos, axis=0)
     mean_sq_vel = np.mean(sq_vel, axis=0)

     # Variance of the squared position and velocity
     var_sq_pos = np.var(sq_pos, axis=0)
     var_sq_vel = np.var(sq_vel, axis=0)

     # Use CLT to find the upper/lower bounds for the 95% uncertainty interval
     lb_sq_pos = mean_sq_pos - 2 * np.sqrt(var_sq_pos / N)
     ub_sq_pos = mean_sq_pos + 2 * np.sqrt(var_sq_pos / N)
     lb_sq_vel = mean_sq_vel - 2 * np.sqrt(var_sq_vel / N)
     ub_sq_vel = mean_sq_vel + 2 * np.sqrt(var_sq_vel / N)

     # Plot average position vs. time
     fig1, ax1 = plt.subplots()
     ax1.plot(solver.t, mean_sq_pos, label='Mean Squared Position Estimate')
     ax1.fill_between(solver.t, lb_sq_pos, ub_sq_pos,
                      alpha=0.25, label='95% Uncertainty Interval')
     ax1.set_xlabel('Time')
     ax1.set_ylabel('Squared Position $y^2(t)$')
     ax1.legend(loc="best", frameon=False)
     sns.despine(trim=True)

     # Plot average velocity vs. time
     fig2, ax2 = plt.subplots()
     ax2.plot(solver.t, mean_sq_vel, label='Mean Squared Velocity Estimate')
     ax2.fill_between(solver.t, lb_sq_vel, ub_sq_vel,
                      alpha=0.25, label='95% Uncertainty Interval')
     ax2.set_xlabel('Time')
     ax2.set_ylabel('Squared Velocity $\dot{y}^2(t)$')
     ax2.legend(loc="best", frameon=False)
     sns.despine(trim=True)
```

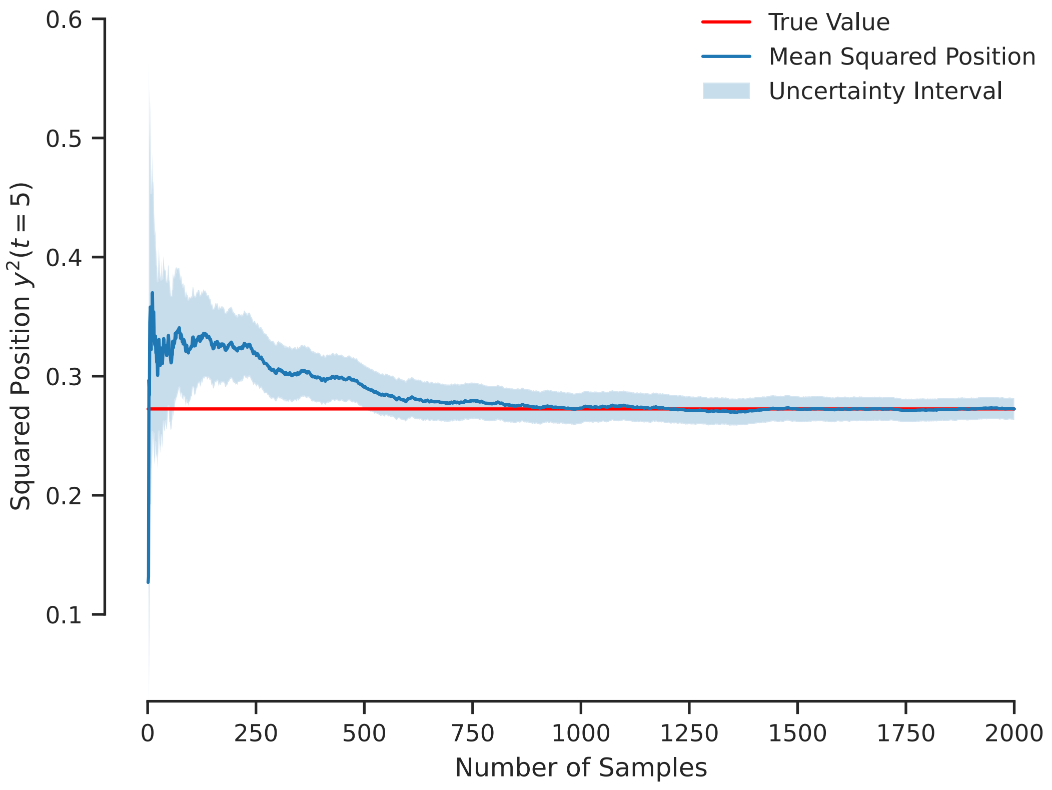

Mean Squared Position Estimate
95% Uncertainty Interval
Squared Position $y^2(t)$
1.0
0.8
0.6
0.4
0.2
0
1
2
3
4
5
Time



Repeat Part B for the Squared Response:

```
[ ]: # Part B for the squared response at t = 5 seconds
     # Use 2000 samples and find the index where t = 5 seconds
     samples = take_samples_from_solver(2000)
     sh = samples.shape
     N = sh[0]
     idx = np.where(solver.t == 5)[0]

     # Find the squared positions at t = 5 seconds
     sq_pos_t5 = np.square(samples[:, idx, 0])

     # Find the squared positions for all sample sizes
     sq_pos_running = np.cumsum(sq_pos_t5) / np.arange(1, N + 1)

     # Find the square of the squared positions for all sample sizes
     sq_pos2_running = np.cumsum(sq_pos_t5 ** 2) / np.arange(1, N + 1)

     # Find the variance of the squared positions for all sample sizes
     var_sq_pos_running = sq_pos2_running - sq_pos_running ** 2
```

```python
# Find the lower/upper bounds for the epistemic uncertainty interval
lb_sq_pos_running = sq_pos_running - 2 * np.sqrt(var_sq_pos_running / np.
 ↪arange(1, N + 1))
ub_sq_pos_running = sq_pos_running + 2 * np.sqrt(var_sq_pos_running / np.
 ↪arange(1, N + 1))

# Find the true values
mean_sq_pos_true = np.mean(samples[:, :, 0] ** 2, axis = 0)

# Plot the mean squared position vs. the number of samples for t = 5 seconds
fig3, ax3 = plt.subplots()
x_vals = np.arange(1, N + 1)
ax3.plot(x_vals, mean_sq_pos_true[idx] * np.ones(N), 'r', label='True Value')
ax3.plot(x_vals, sq_pos_running, label='Mean Squared Position')
ax3.fill_between(x_vals, lb_sq_pos_running, ub_sq_pos_running,
                 alpha=0.25, label='Uncertainty Interval')
ax3.set_xlabel('Number of Samples')
ax3.set_ylabel('Squared Position $y^2(t = 5)$')
ax3.set_ylim(np.min(lb_sq_pos_running), 1.1 * np.max(ub_sq_pos_running))
ax3.legend(loc="best", frameon=False)
sns.despine(trim=True)
```



[ ]:
```python
# Samples needed to estimate squared mean @ t = 5 sec w/ negligible uncertainty
interval_width = ub_sq_pos_running - lb_sq_pos_running
threshold = 0.02

# Find indices where the condition is met
# Note: The first two points on the plot (N = 1 and N = 2) meet the condition.
# These two points will be ignored when finding the valid indices.
valid_indices = np.where(interval_width[2:] < threshold)[0]

# Get the first index that meets the condition & the corresponding # of samples
if valid_indices.size > 0:
    N_negligible = valid_indices[0]
    width_negligible = interval_width[N_negligible]
    print(f"Number of samples for negligible epistemic uncertainty =␣
 ↪{N_negligible}")
    print(f"Uncertainty interval width for {N_negligible} samples =␣
 ↪{width_negligible:.2f}")
else:
    print("No values found below the threshold.")
```

```
Number of samples for negligible epistemic uncertainty = 1695
Uncertainty interval width for 1695 samples = 0.02
```

We need approximately 1695 samples to estimate the mean squared response at $t = 5$ seconds with negligible epistemic uncertainty.

This value was be determined by evaluating the width of the uncertainty interval throughout the plot. I chose 0.02 to be the uncertainty interval width that we want to be below. The first instance where the uncertainty interval is below this threshold is when $N = 1695$.

1.4.4 Part D

Now that you know how many samples you need to estimate the mean of the response and the square response, use the formula:

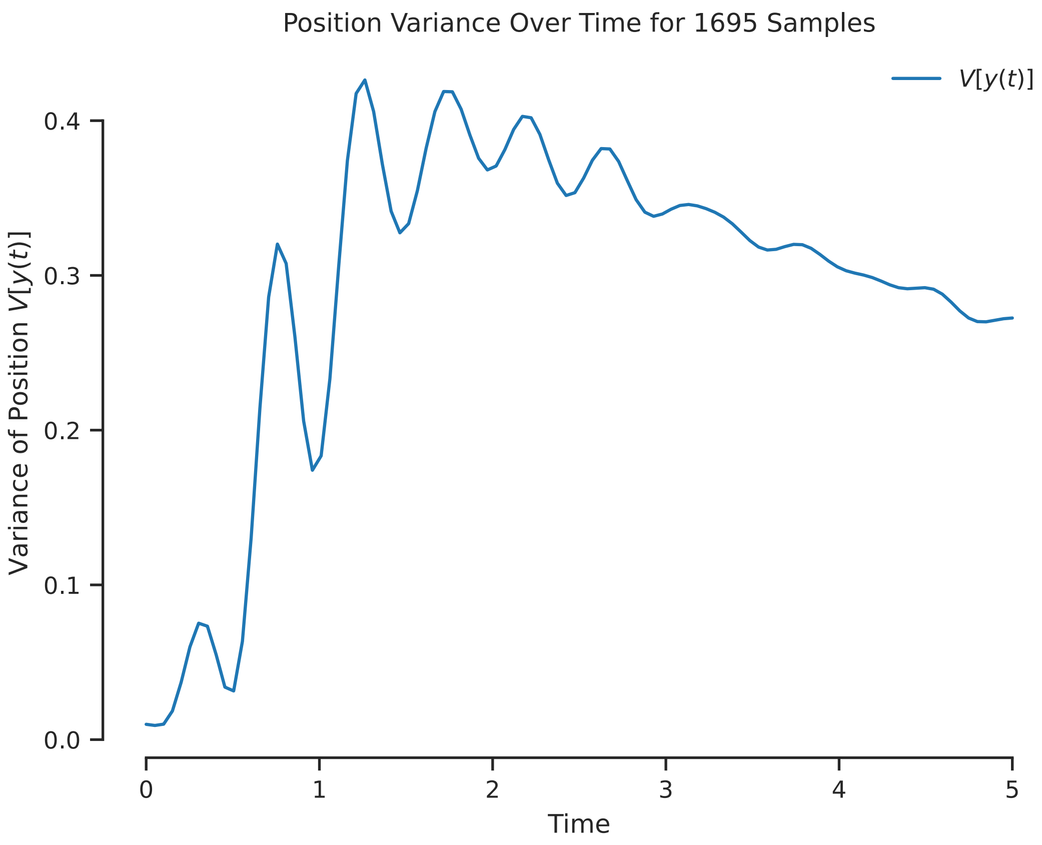
$$\mathbb{V}[y(t)] = \mathbb{E}[y^2(t)] - \left(\mathbb{E}[y(t)]\right)^2,$$

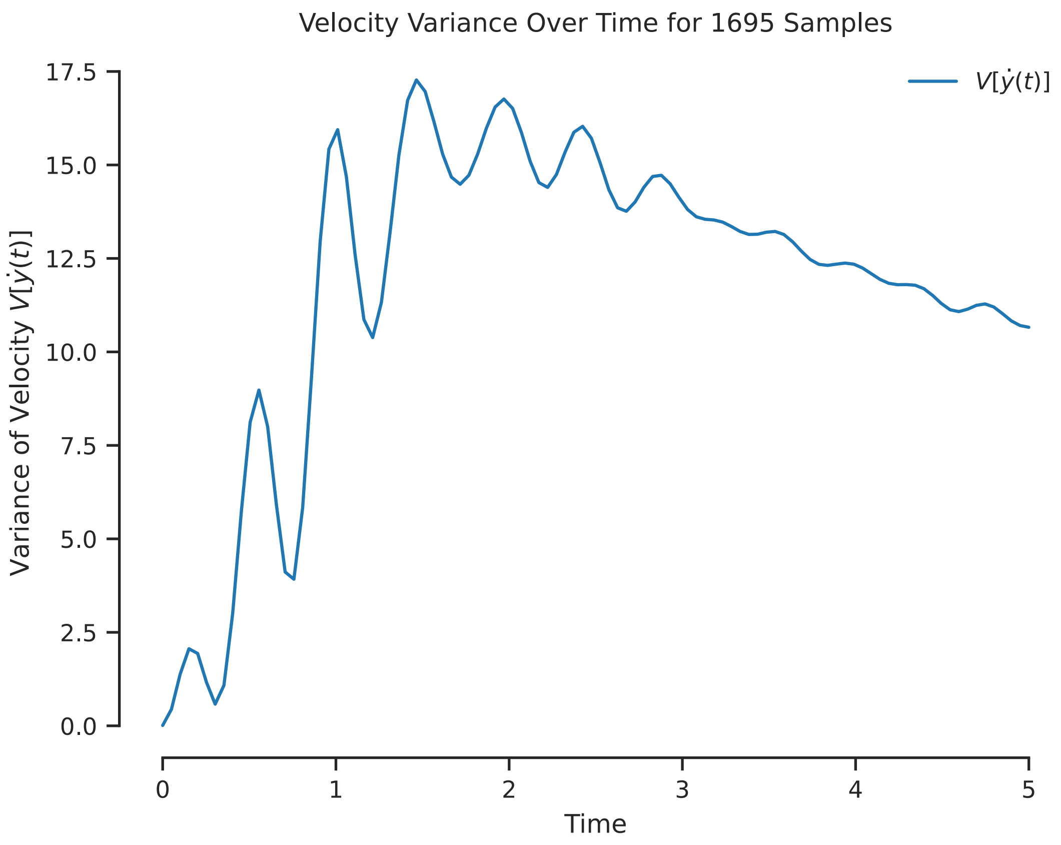
and similarly, for $\dot{y}(t)$, to estimate the position and velocity variance with negligible epistemic uncertainty. Plot both quantities as a function of time.

Solution:

[ ]:
```python
# Your code here
# Number of samples needed for negligible epistemic uncertainty
N_negligible = 1695
samples = take_samples_from_solver(N_negligible)

# Mean position and velocity
mean_pos = np.mean(samples[:, :, 0], axis=0)
```

```
mean_vel = np.mean(samples[:, :, 1], axis=0)

# Mean squared position and velocity
mean_sq_pos = np.mean(samples[:, :, 0] ** 2, axis=0)
mean_sq_vel = np.mean(samples[:, :, 1] ** 2, axis=0)

# Variance of position and velocity
var_pos = mean_sq_pos - mean_pos ** 2
var_vel = mean_sq_vel - mean_vel ** 2

# Plot variance of position vs. time
fig1, ax1 = plt.subplots()
ax1.plot(solver.t, var_pos, label='$V[y(t)]$')
ax1.set_xlabel('Time')
ax1.set_ylabel('Variance of Position $V[y(t)]$')
ax1.set_title('Position Variance Over Time for 1695 Samples')
ax1.legend(loc="best", frameon=False)
sns.despine(trim=True)

# Plot variance of velocity vs. time
fig2, ax2 = plt.subplots()
ax2.plot(solver.t, var_vel, label='$V[\dot{y}(t)]$')
ax2.set_xlabel('Time')
ax2.set_ylabel('Variance of Velocity $V[\dot{y}(t)]$')
ax2.set_title('Velocity Variance Over Time for 1695 Samples')
ax2.legend(loc="best", frameon=False)
sns.despine(trim=True)
```


Position Variance Over Time for 1695 Samples
$V[y(t)]$
Variance of Position $V[y(t)]$
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
Time



1.4.5 Part E

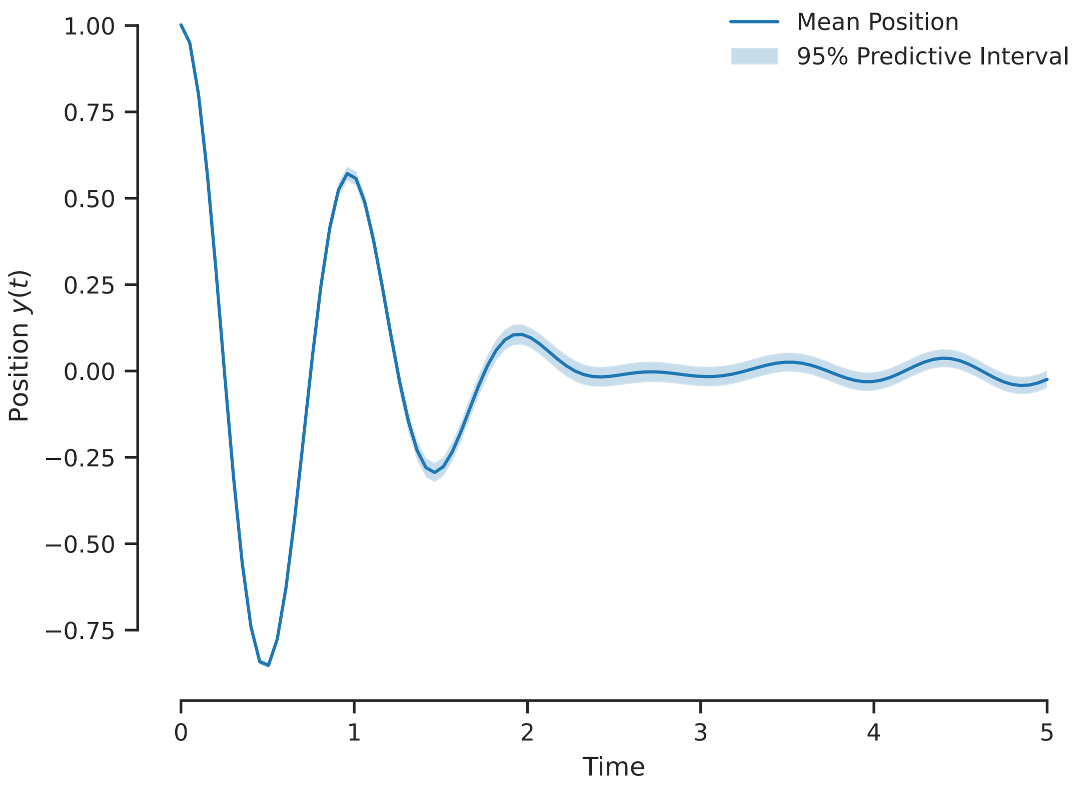
Put together the estimated mean and variance to plot a 95% predictive interval for the position and the velocity as functions of time.

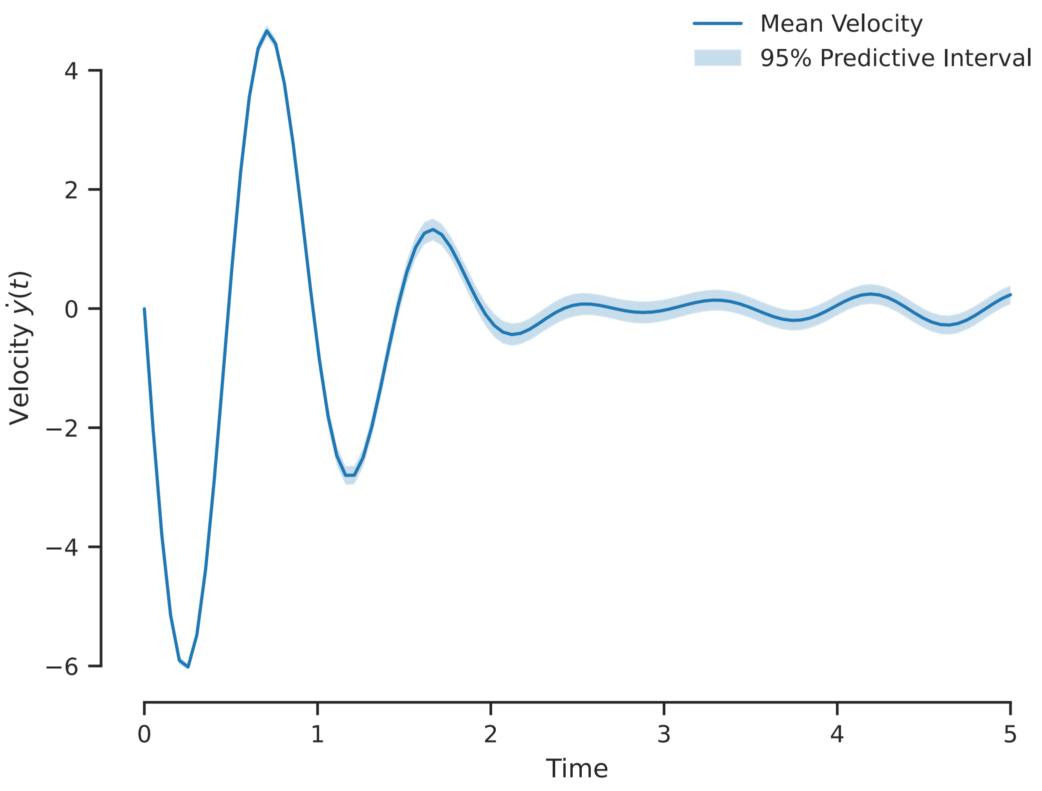
Hint: You need to use the Central Limit Theorem. Check out the corresponding textbook example.

Solution:

[ ]:
```
# Your code here
# Compute the lower and upper bounds
lb_pos = mean_pos - 2 * np.sqrt(var_pos / N_negligible)
ub_pos = mean_pos + 2 * np.sqrt(var_pos / N_negligible)
lb_vel = mean_vel - 2 * np.sqrt(var_vel / N_negligible)
ub_vel = mean_vel + 2 * np.sqrt(var_vel / N_negligible)

# Plot average position vs. time
fig1, ax1 = plt.subplots()
ax1.plot(solver.t, mean_pos, label='Mean Position')
ax1.fill_between(solver.t, lb_pos, ub_pos, alpha=0.25,
                 label='95% Predictive Interval')
```

```python
ax1.set_xlabel('Time')
ax1.set_ylabel('Position $y(t)$')
ax1.legend(loc="best", frameon=False)
sns.despine(trim=True)

# Plot average velocity vs. time
fig2, ax2 = plt.subplots()
ax2.plot(solver.t, mean_vel, label='Mean Velocity')
ax2.fill_between(solver.t, lb_vel, ub_vel, alpha=0.25,
                 label='95% Predictive Interval')
ax2.set_xlabel('Time')
ax2.set_ylabel('Velocity $\dot{y}(t)$')
ax2.legend(loc="best", frameon=False)
sns.despine(trim=True)
```





1.5 Problem 2 - Earthquakes again

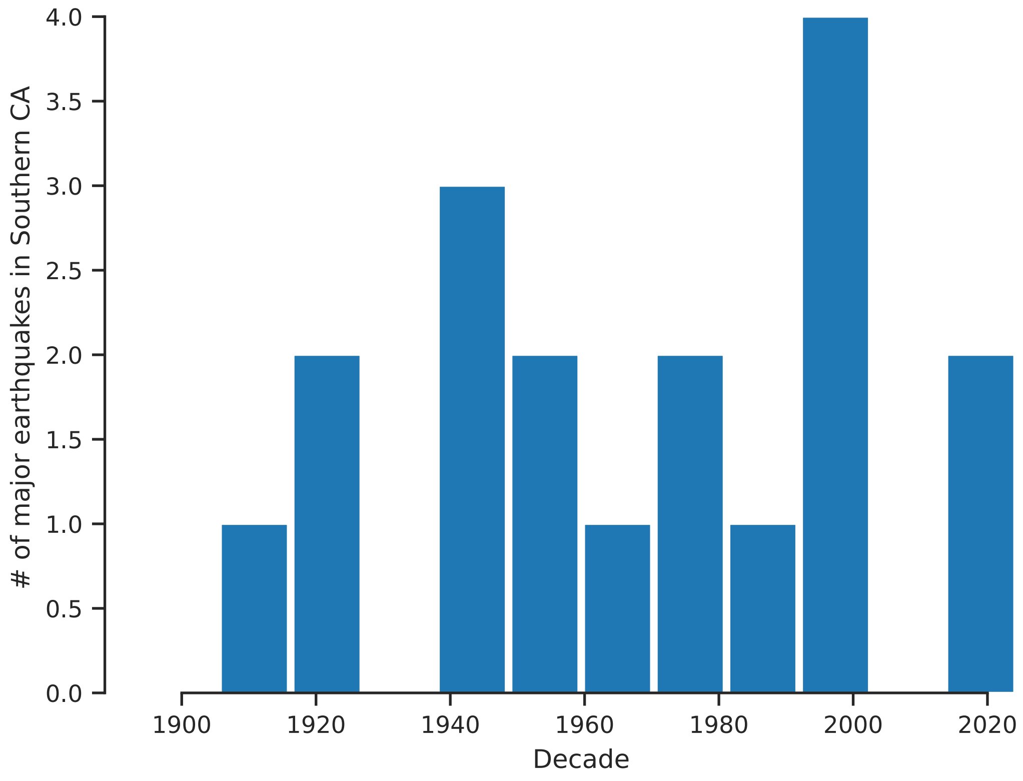
The San Andreas fault extends through California, forming the boundary between the Pacific and the North American tectonic plates. It has caused some of the most significant earthquakes on Earth. We are going to focus on Southern California, and we would like to assess the probability of a significant earthquake, defined as an earthquake of magnitude 6.5 or greater, during the next ten years.

A. The first thing we will do is review a database of past earthquakes that have occurred in Southern California and collect the relevant data. We will start at 1900 because data before that time may be unreliable. Go over each decade and count the occurrence of a significant earthquake (i.e., count the number of orange and red colors in each decade). We have done this for you.

[ ]:
```
eq_data = np.array([
    0, # 1900-1909
    1, # 1910-1919
    2, # 1920-1929
    0, # 1930-1939
    3, # 1940-1949
    2, # 1950-1959
    1, # 1960-1969
    2, # 1970-1979
    1, # 1980-1989
    4, # 1990-1999
    0, # 2000-2009
    2 # 2010-2019
])
```

Let's visualize them:

```
[ ]: fig, ax = plt.subplots()
     ax.bar(
         np.linspace(1900, 2019, eq_data.shape[0]),
         eq_data,
         width=10
     )
     ax.set_xlabel('Decade')
     ax.set_ylabel('# of major earthquakes in Southern CA')
     sns.despine(trim=True);
```



A. The right way to model the number of earthquakes X_n in a decade n is using a Poisson distribution with unknown rate parameter λ, i.e.,

$$X_n|\lambda \sim \text{Poisson}(\lambda).$$

The probability mass function is:

$$p(x_n|\lambda) \equiv p(X_n = x_n|\lambda) = \frac{\lambda^{x_n}}{x_n!}e^{-\lambda}.$$

Here we have $N = 12$ observations, say $x_{1:N} = (x_1, \dots, x_N)$ (stored in `eq_data` above). Find the *joint probability mass function* (otherwise known as the likelihood) $p(x_{1:N}|\lambda)$ of these random variables.

Hint: Assume that all measurements are independent. Then, their joint PMF is the product of the individual PMFs. You should be able to simplify the expression.

Answer: According to the problem statement, the PMF of the Poisson distribution is:

$$p(x_n|\lambda) \equiv p(X_n = x_n|\lambda) = \frac{\lambda^{x_n}}{x_n!}e^{-\lambda}$$

Since all 12 earthquake measurements are independent, the joint PMF is the product of the individual PMFs:

$$p(x_{1:N}|\lambda) = \prod_{n=1}^{N} p(x_n|\lambda)$$

$$p(x_{1:N}|\lambda) = \prod_{n=1}^{N} \frac{\lambda^{x_n} e^{-\lambda}}{x_n!}$$

$$p(x_{1:N}|\lambda) = (\prod_{n=1}^{N} \lambda^{x_n})(\prod_{n=1}^{N} e^{-\lambda})(\prod_{n=1}^{N} \frac{1}{x_n!})$$

$$p(x_{1:N}|\lambda) = (\lambda^{\sum_{n=1}^{N} x_n})(e^{-N\lambda})(\frac{1}{\prod_{n=1}^{N} x_n!})$$

First, we must list out the $N = 12$ x-values:

$$x_{1:N} = (0, 1, 2, 0, 3, 2, 1, 2, 1, 4, 0, 2)$$

Next, we must compute the sum of the x-values:

$$\sum_{n=1}^{N} x_n = 0 + 1 + 2 + 0 + 3 + 2 + 1 + 2 + 1 + 4 + 0 + 2$$

$$\sum_{n=1}^{N} x_n = 18$$

Then we must compute the product of the factorial of the x-values:

$$\prod x_n! = 0! \cdot 1! \cdot 2! \cdot 0! \cdot 3! \cdot 2! \cdot 1! \cdot 2! \cdot 1! \cdot 4! \cdot 0! \cdot 2!$$

$$\prod x_n! = 1 \cdot 1 \cdot 2 \cdot 1 \cdot 6 \cdot 2 \cdot 1 \cdot 2 \cdot 1 \cdot 24 \cdot 1 \cdot 2$$

$$\prod x_n! = 2304$$

Plugging these values into the joint PMF equation gives the following result:

$$\boxed{p(x_{1:N}|\lambda) = \frac{\lambda^{18} e^{-12\lambda}}{2304}}$$

B. The rate parameter λ (number of significant earthquakes per ten years) is positive. What prior distribution should we assign to it if we expect it to be around 2? A convenient choice is to pick a Gamma. See also the scipy.stats page for the Gamma because it results in an analytical posterior. We write:

$$\lambda \sim \text{Gamma}(\alpha, \beta),$$

where α and β are positive *hyper-parameters* that we must set to represent our prior state of knowledge. The PDF is:

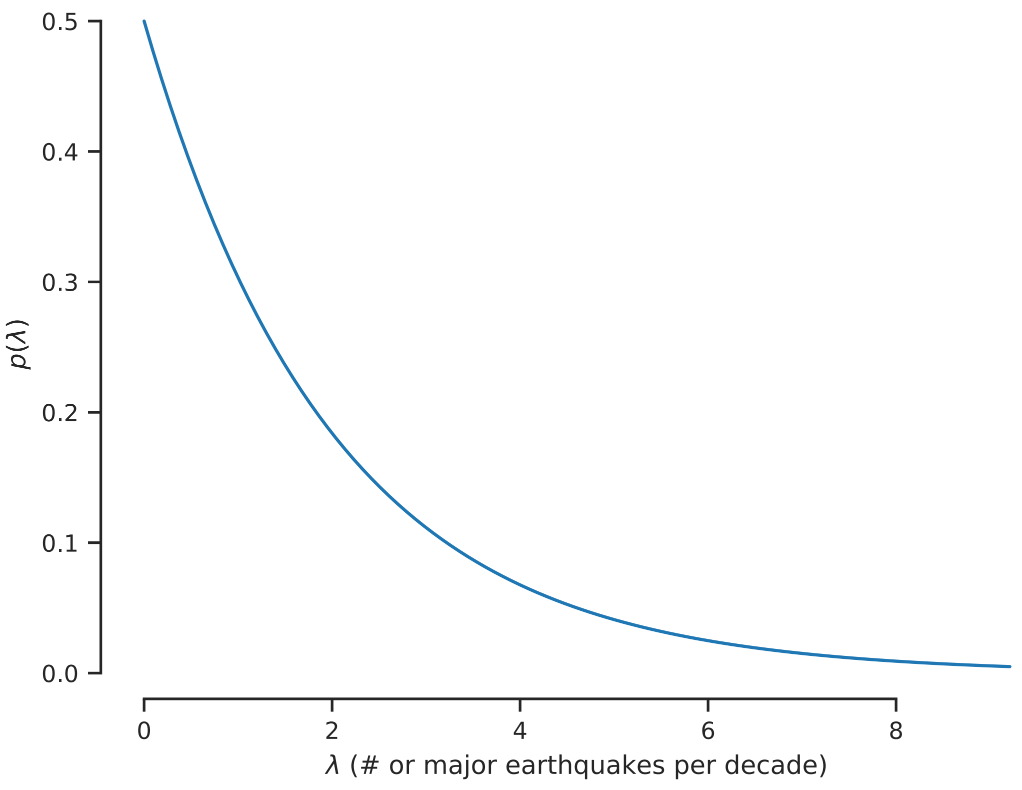
$$p(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)},$$

where we are not conditioning on α and β because they should be fixed numbers. Use the code below to pick reasonable values for α and β. **Just enter your choice of α and β in the code block below.**

Hint: Notice that the maximum entropy distribution for a positive parameter with known expectation is the Exponential, e.g., see the Table in this wiki page. Then, notice that the Exponential is a particular case of the Gamma (set $\alpha = 1$).

[ ]:
```python
import scipy.stats as st

# Expectation of the gamma distribution is alpha/beta --> E[lambda] = alpha/beta
# We know that E[lambda] = 2 and alpha = 1 --> 2 = 1/beta --> beta = 0.5

# You have to pick an alpha:
alpha = 1.0
```

```
# And you have to pick a beta:
beta = 0.5

# This is the prior on lambda:
lambda_prior = st.gamma(alpha, scale=1.0 / beta)

# Let's plot it:
lambdas = np.linspace(0, lambda_prior.ppf(0.99), 100)
fig, ax = plt.subplots()
ax.plot(lambdas, lambda_prior.pdf(lambdas))
ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
ax.set_ylabel('$p(\lambda)$')
sns.despine(trim=True);
```



C. Show that the posterior of $\lambda$ conditioned on $x_{1:N}$ is also a Gamma, but with updated hyperparameters. **Hint:** When you write down the posterior of $\lambda$ you can drop any multiplicative term that does not depend on it as it will be absorbed in the normalization constant. This will simplify the notation a little bit. **Answer:**

The prior probability is as follows:

$$p(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}$$

From part A, we know the likelihood is:

$$p(x_{1:N}|\lambda) = (\lambda^{\sum\limits_{n=1}^{N} x_n})(e^{-N\lambda})(\frac{1}{\prod\limits_{n=1}^{N} x_n!})$$

Now we can equate the posterior with the prior and likelihood using the following proportionality relation:

$$\text{Posterior} \propto \text{Likelihood} \cdot \text{Prior}$$

$$p(\lambda|x_{1:N}) \propto p(x_{1:N}|\lambda) \cdot p(\lambda)$$

$$p(\lambda|x_{1:N}) \propto (\lambda^{\sum\limits_{n=1}^{N} x_n})(e^{-N\lambda})(\frac{1}{\prod\limits_{n=1}^{N} x_n!})(\frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)})$$

We can drop the constant terms from the proportionality relation:

$$p(\lambda|x_{1:N}) \propto (\lambda^{\sum\limits_{n=1}^{N} x_n})(e^{-N\lambda})(\lambda^{\alpha-1} e^{-\beta\lambda})$$

Combine the λ terms and exponential terms:

$$p(\lambda|x_{1:N}) \propto (\lambda^{\alpha - 1 + \sum\limits_{n=1}^{N} x_n})(e^{-N\lambda-\beta\lambda})$$

$$p(\lambda|x_{1:N}) \propto (\lambda^{\alpha - 1 + \sum\limits_{n=1}^{N} x_n})(e^{-\lambda(\beta+N)})$$

From the above relation, we can see that:

$$\boxed{\lambda|x_{1:N} \sim \text{Gamma}(\alpha + \sum_{n=1}^{N} x_n, \beta + N)}$$

Therefore, the posterior of $\lambda$ conditioned on $x_{1:N}$ also follows a Gamma distribution, but with different parameters. We can solve for the new α and β by comparing the prior and posterior parameters:

$$\alpha_{posterior} = \alpha_{prior} + \sum_{n=1}^{N} x_n = 1 + 18 = 19$$

$$\beta_{posterior} = \beta_{prior} + N = 0.5 + 12 = 12.5$$

Thus, the updated hyperparameters are:

$$\boxed{\alpha_{posterior} = 19}$$

$$\boxed{\beta_{posterior} = 12.5}$$

D. Prior-likelihood pairs that result in a posterior with the same form as the prior are known as conjugate distributions. Conjugate distributions are your only hope for analytical Bayesian inference. As a verification check, look at the Wikipedia page for conjugate priors, locate the Poisson-Gamma pair, and verify your answer above. *Nothing to report here. Just do it as a verification check.*

The prior hyperparameters are $\alpha = 1$ and $\beta = 0.5$.

According to the Wikipedia page, the posterior hyperparameters are:

$\alpha + \sum_{n=1}^{N} x_n = 1 + 18 = 19$

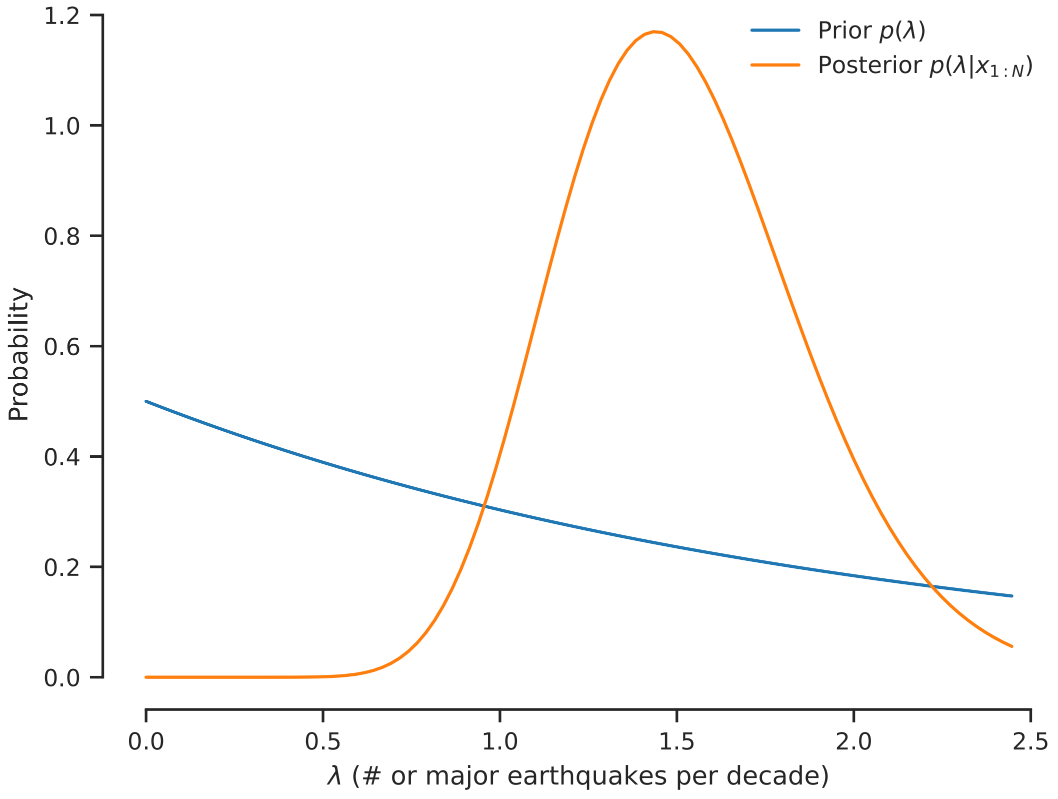
$\beta + N = 0.5 + 12 = 12.5$

The verifies that the answers in Part C are correct.

E. Plot the prior and the posterior of $\lambda$ on the same plot.

```
[ ]: # Your expression for alpha posterior here:
     alpha_post = alpha + eq_data.sum()  # Equals 19
     # Your expression for beta posterior here:
     beta_post = beta + eq_data.shape[0]  # Equals 12.5
     # The posterior
     lambda_post = st.gamma(alpha_post, scale=1.0 / beta_post)

     # Plot it
     lambdas = np.linspace(0, lambda_post.ppf(0.99), 100)
     fig, ax = plt.subplots()
     ax.plot(lambdas,lambda_prior.pdf(lambdas),label='Prior $p(\lambda)$')
     ax.plot(lambdas,lambda_post.pdf(lambdas),label='Posterior $p(\lambda|x_{1:N})$')
     ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
     ax.set_ylabel('Probability')
     ax.legend(loc="best", frameon=False)
     sns.despine(trim=True);

     print(f"alpha_post = {alpha_post}")
     print(f"beta_post = {beta_post}")
```

```
alpha_post = 19.0
beta_post = 12.5
```



F. Let's determine the predictive distribution for the number of significant earthquakes during the next decade. This is something we did not do in class, but it will reappear in future lectures. Let X be the random variable corresponding to the number of significant earthquakes during the next decade. We need to calculate:

$$p(x|x_{1:N}) = \text{our state of knowledge about } X \text{ after seeing the data.}$$

How do we do this? We use the sum rule:

$$p(x|x_{1:N}) = \int_0^\infty p(x|\lambda, x_{1:N})p(\lambda|x_{1:N})d\lambda = \int_0^\infty p(x|\lambda)p(\lambda|x_{1:N})d\lambda,$$

where going from the middle step to the rightmost one, we assumed that the number of earthquakes occurring in each decade is independent. You can carry out this integration analytically (it gives a negative Binomial distribution), but we are not going to bother with it.

Below, you will write code to characterize it using Monte Carlo sampling. You can take a sample from the posterior predictive by:

- sampling a λ from its posterior $p(\lambda|x_{1:N})$.
- sampling an x from the likelihood $p(x|\lambda)$.

This is the same procedure we used for replicated experiments.

Complete the code below:

```
[ ]: def sample_posterior_predictive(n, lambda_post):
         """Sample from the posterior predictive.

         Arguments
         n           -- The number of samples to take.
         lambda_post -- The posterior for lambda.

         Returns n samples from the posterior
         """
         samples = np.empty((n,), dtype="i")
         for i in range(n):
             # WRITE ME (SAMPLE FROM POSTERIOR)
             lambda_sample = lambda_post.rvs()
             # WRITE ME (SAMPLE FROM POISSON GIVEN LAMBDA_SAMPLE)
             samples[i] = st.poisson.rvs(mu=lambda_sample)
         return samples
```

Test your code here:

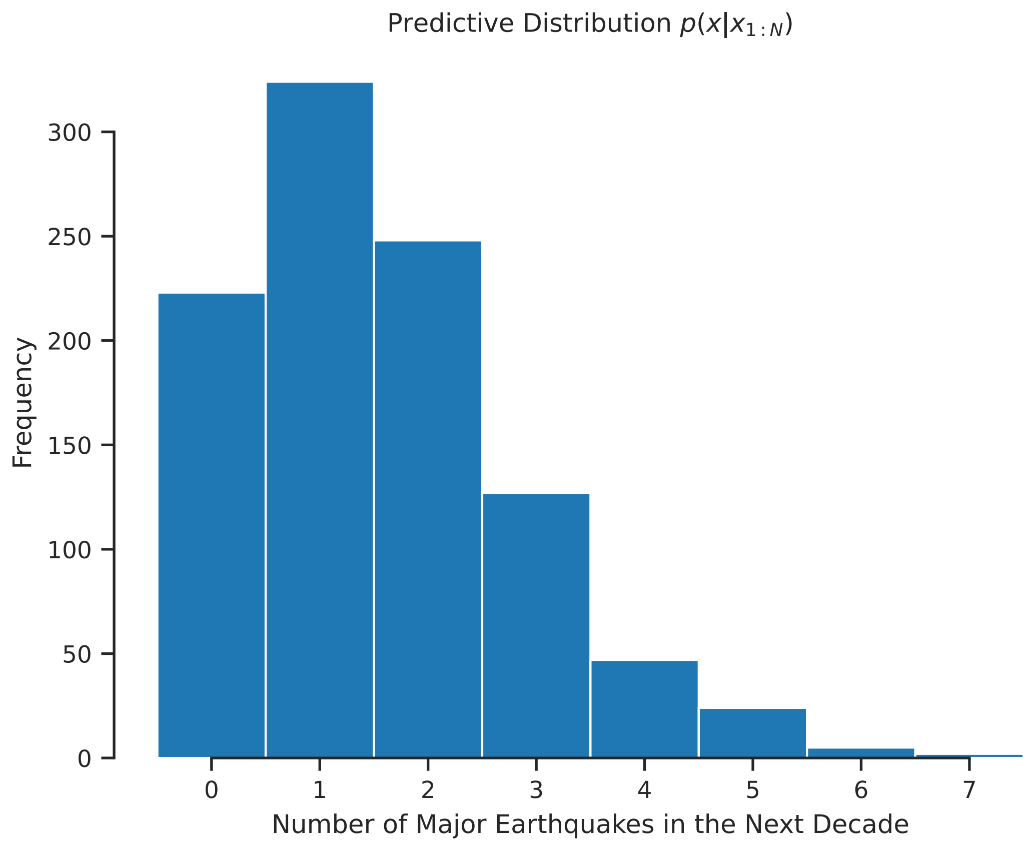
```
[ ]: samples = sample_posterior_predictive(10, lambda_post)
     samples
```

```
[ ]: array([1, 4, 0, 2, 2, 0, 1, 1, 3, 0], dtype=int32)
```

G. Plot the predictive distribution $p(x|x_{1:N})$.

Hint: Draw 1,000 samples using `sample_posterior_predictive` and then draw a histogram.

```
[ ]: # Your code here
     samps = sample_posterior_predictive(1000, lambda_post)
     bins = np.arange(samps.max() + 2) - 0.5
     plt.hist(samps, bins=bins)
     plt.xlabel('Number of Major Earthquakes in the Next Decade')
     plt.ylabel('Frequency')
     plt.title('Predictive Distribution $p(x|x_{1:N})$')
     sns.despine(trim=True)
```



H. What is the probability that at least one major earthquake will occur during the next decade?

Hint: You may use a Monte Carlo estimate of the probability. Ignore the uncertainty in the estimate.

```
[ ]: num_samples = 10000
     samples = sample_posterior_predictive(num_samples, lambda_post)

     # Count how many major earthquakes occured:
     count = 0
     for i in range(num_samples):
         if samples[i] >=1:
             count += 1

     prob_of_major_eq = count / num_samples  # YOUR ESTIMATE HERE

     print(f"p(X >= 1 | data) = {prob_of_major_eq}")
```

```
p(X >= 1 | data) = 0.7654
```

I. Find a 95% credible interval for λ.

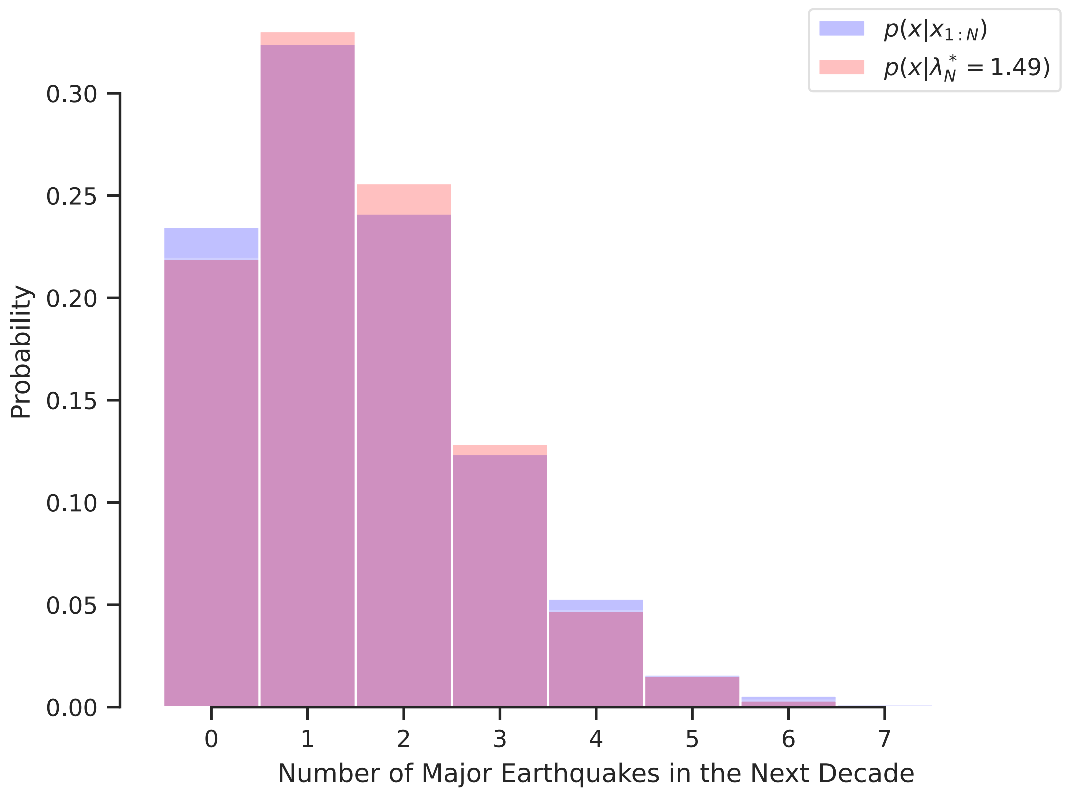
```
[ ]: # Write your code here and print() your answer
     p_0025 = lambda_post.ppf(0.025)
     p_0975 = lambda_post.ppf(0.975)
     print(f"A 95% credible interval for lambda is [{p_0025:.3f}, {p_0975:.3f}].")
```

```
A 95% credible interval for lambda is [0.915, 2.276].
```

J. Find the λ that minimizes the absolute loss (see lecture), and call it λ_N^*. Then, plot the fully Bayesian predictive $p(x|x_{1:N})$ in the same figure as $p(x|\lambda_N^*)$.

```
[ ]: # Write your code here and print() your answer
     # Median lambda minimizes absolute loss: p(lambda <= lambda_N* | x_1:N) = 0.5
     lambda_N_star = lambda_post.median()  # Note: could also do lambda_post.ppf(0.5)
     print(f"The lambda_N_star that minimizes absolute loss is {lambda_N_star:.2f}")

     # Use the posterior predictive samples from part H for p(x | x_1:N)
     x_bayesian = samples

     # p(x | lambda_N_star) = Poisson(x | lambda_N_star)
     x_N_star = st.poisson(mu=lambda_N_star).rvs(size=num_samples)

     # Plot the Bayesian predictive p(x | x_1:N) and p(x | lambda_N_star)
     fig, ax = plt.subplots()
     bins = np.arange(samples.min() - 0.5, samples.max() + 1.5, 1)
     ax.hist(x_bayesian, bins=bins, alpha=0.25, color='blue',
             density=True, label=r'$p(x|x_{1:N})$')
     ax.hist(x_N_star, bins=bins, alpha=0.25, color='red',
             density=True, label=r'$p(x|\lambda_N^*=1.49)$')
     ax.set_xlabel('Number of Major Earthquakes in the Next Decade')
     ax.set_ylabel('Probability')
     ax.set_xticks(np.arange(0, 8))
     ax.legend(loc="best", frameon=True, framealpha=0.6)
     sns.despine(trim=True)
     plt.show()
```

```
The lambda_N_star that minimizes absolute loss is 1.49
```



L. Draw replicated data from the model and compare them to the observed data.

Hint: Complete the missing code at the places indicated below.

```
[ ]: def replicate_experiment(post_rv, n=len(eq_data), n_rep=9):
         """Replicate the experiment.

         Arguments
         post_rv -- The random variable object corresponding to
                    the posterior from which to sample.
         n       -- The number of observations.
         nrep    -- The number of repetitions.

         Returns:
         A numpy array of size n_rep x n.
         """
         x_rep = np.empty((n_rep, n), dtype="i")
         for i in range(n_rep):
             x_rep[i, :] = st.poisson.rvs(size=n, mu=post_rv.rvs()) # Your code here
         return x_rep
```

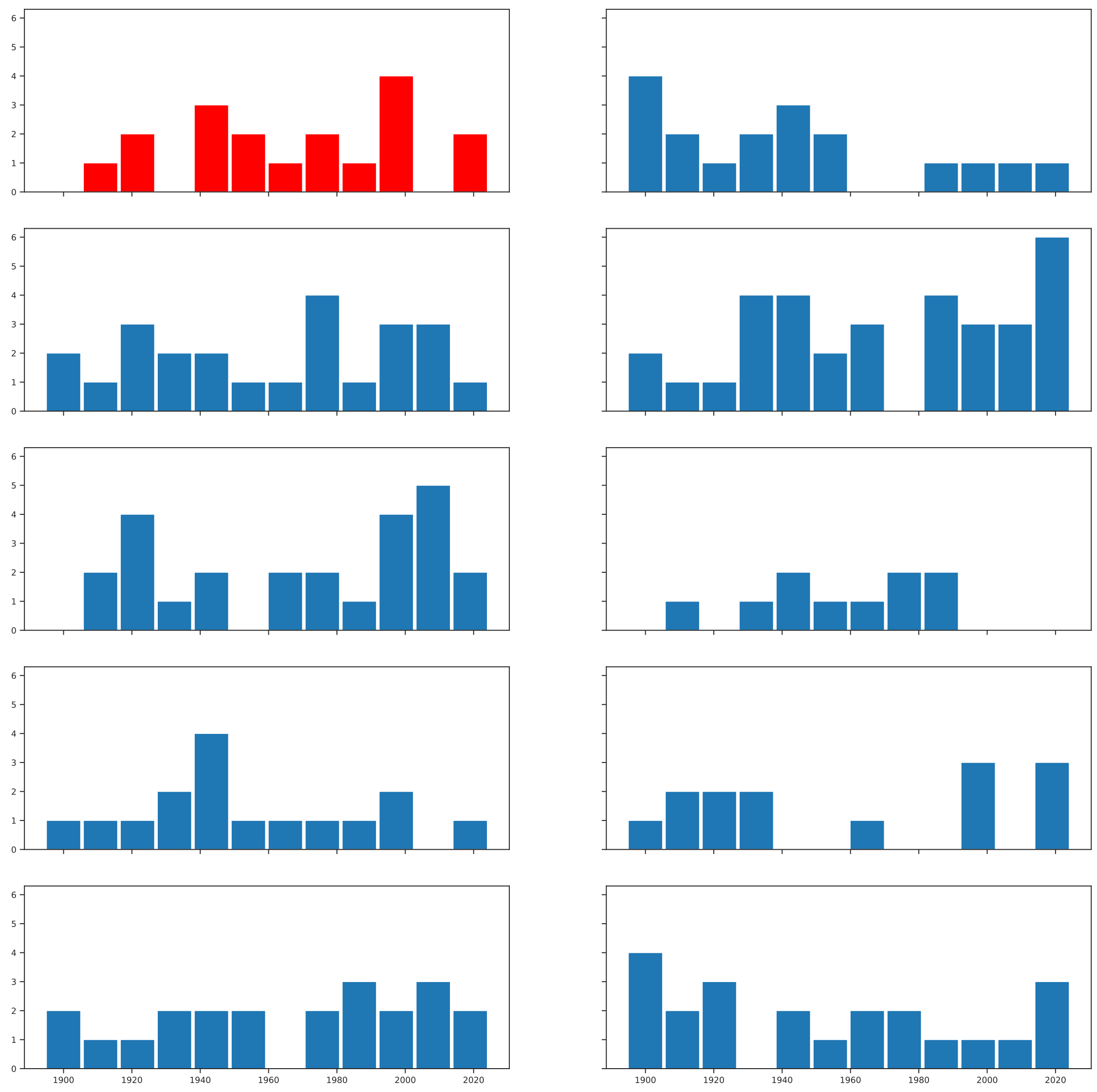
Try your code here:

```
[ ]: x_rep = replicate_experiment(lambda_post)
     x_rep
```

```
[ ]: array([[4, 2, 1, 2, 3, 2, 0, 0, 1, 1, 1, 1],
            [2, 1, 3, 2, 2, 1, 1, 4, 1, 3, 3, 1],
            [2, 1, 1, 4, 4, 2, 3, 0, 4, 3, 3, 6],
            [0, 2, 4, 1, 2, 0, 2, 2, 1, 4, 5, 2],
            [0, 1, 0, 1, 2, 1, 1, 2, 2, 0, 0, 0],
            [1, 1, 1, 2, 4, 1, 1, 1, 1, 2, 0, 1],
            [1, 2, 2, 2, 0, 0, 1, 0, 0, 3, 0, 3],
            [2, 1, 1, 2, 2, 2, 0, 2, 3, 2, 3, 2],
            [4, 2, 3, 0, 2, 1, 2, 2, 1, 1, 1, 3]], dtype=int32)
```

If it works, then try the following visualization:

```
[ ]: fig, ax = plt.subplots(
         5,
         2,
         sharex='all',
         sharey='all',
         figsize=(20, 20)
     )
     ax[0, 0].bar(
         np.linspace(1900, 2019, eq_data.shape[0]),
         eq_data,
         width=10,
         color='red'
     )
     n_rep = 9
     for i in range(1, n_rep + 1):
         ax[int(i / 2), i % 2].bar(
             np.linspace(1900, 2019, eq_data.shape[0]),
             x_rep[i-1],
             width=10
         )
```



M. Plot the histograms and calculate the Bayesian p-values of the following test quantities:

- Maximum number of consecutive decades with no earthquakes.
- Maximum number of successive decades with earthquakes.

Hint: You may reuse the code from Posterior Predictive Checking.

```
[ ]: def perform_diagnostics(post_rv, data, test_func, n_rep=1000):
         """Calculate Bayesian p-values.

         Arguments
         post_rv    -- The random variable object corresponding to
                       the posterior from which to sample.
         data       -- The training data.
```

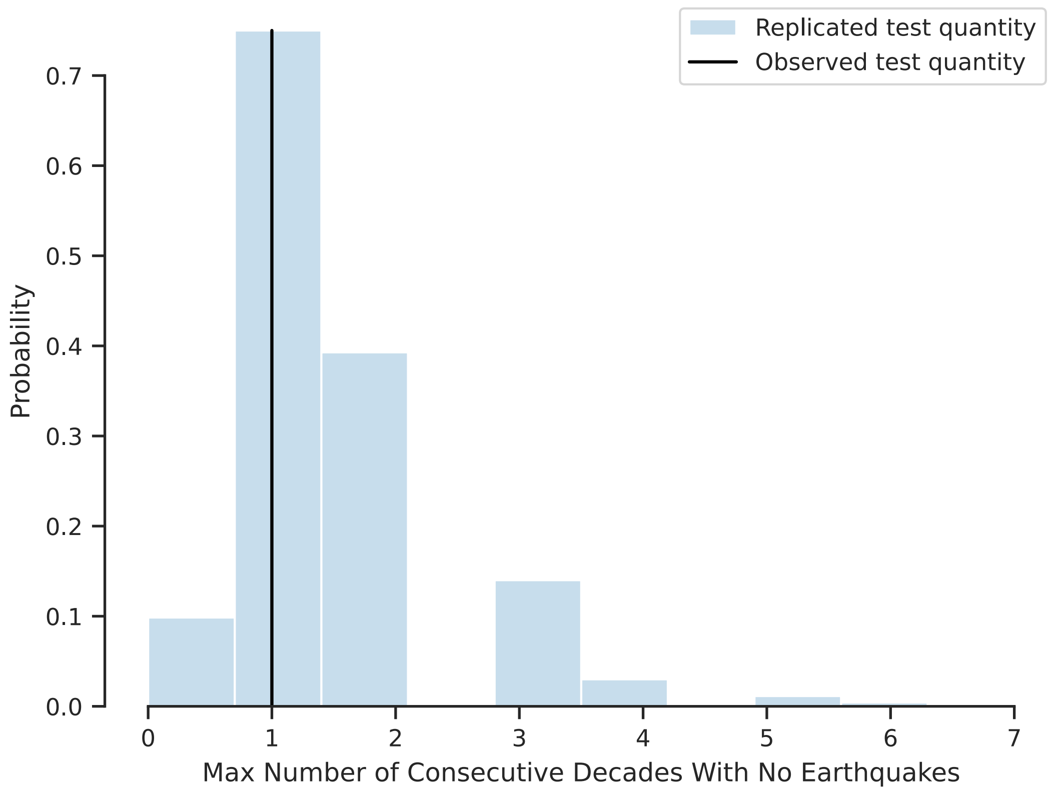
```
    test_func -- The test function.
    n         -- The number of observations.
    nrep      -- The number of repetitions.

    Returns a dictionary that includes the observed value of
    the test function (T_obs), the Bayesian p-value (p_val),
    the replicated test statistic (T_rep),
    and all the replicated data (data_rep).
    """
    T_obs = test_func(data)
    n = data.shape[0]
    data_rep = replicate_experiment(post_rv, n_rep=n_rep)
    T_rep = np.array(
        tuple(
            test_func(x)
            for x in data_rep
        )
    )
    p_val = (
        np.sum(np.ones((n_rep,))[T_rep > T_obs]) / n_rep
    )
    return dict(
        T_obs=T_obs,
        p_val=p_val,
        T_rep=T_rep,
        data_rep=data_rep
    )


def plot_diagnostics(diagnostics):
    """Make the diagnostics plot.

    Arguments:
    diagnostics -- The dictionary returned by perform_diagnostics()
    """
    fig, ax = plt.subplots()
    tmp = ax.hist(
        diagnostics["T_rep"],
        density=True,
        alpha=0.25,
        label='Replicated test quantity'
    )[0]
    ax.plot(
        diagnostics["T_obs"] * np.ones((50,)),
        np.linspace(0, tmp.max(), 50),
        'k',
        label='Observed test quantity'
    )
    plt.legend(loc='best');


def do_diagnostics(post_rv, data, test_func, n_rep=1000):
    """Calculate Bayesian p-values and make the corresponding
    diagnostic plot.

    Arguments
    post_rv   -- The random variable object corresponding to
                 the posterior from which to sample.
    data      -- The training data.
    test_func -- The test function.
    n         -- The number of observations.
    nrep      -- The number of repetitions.

    Returns a dictionary that includes the observed value of
    the test function (T_obs), the Bayesian p-value (p_val),
    and the replicated experiment (data_rep).
    """
    res = perform_diagnostics(
        post_rv,
        data,
        test_func,
        n_rep=n_rep
    )

    T_obs = res["T_obs"]
    p_val = res["p_val"]

    print(f'The observed test quantity is {T_obs}')
    print(f'The Bayesian p_value is {p_val:.4f}')

    plot_diagnostics(res)
```

[ ]:
```
# Here is the first test function for you
def T_eq_max_neq(x):
    """Return the maximum number of consecutive decades
    with no earthquakes."""
    count = 0
    result = 0
    for i in range(x.shape[0]):
        if x[i] != 0:
            count = 0
        else:
            count += 1
            result = max(result, count)
    return result

# Consult the textbook (Lecture 12) to figure out
# how to use do_diagnostics().
```

[ ]:
```
# Write your code here for the second test quantity
# (maximum number of consecutive decades with earthquakes)
# Hint: copy paste your code from the previous cell
# and make the necessary modifications
def T_eq_max_eq(x):
    """Return the maximum number of consecutive decades
    with earthquakes."""
    count = 0
    result = 0
    for i in range(x.shape[0]):
        if x[i] == 0:
            count = 0
        else:
            count += 1
            result = max(result, count)
    return result
```

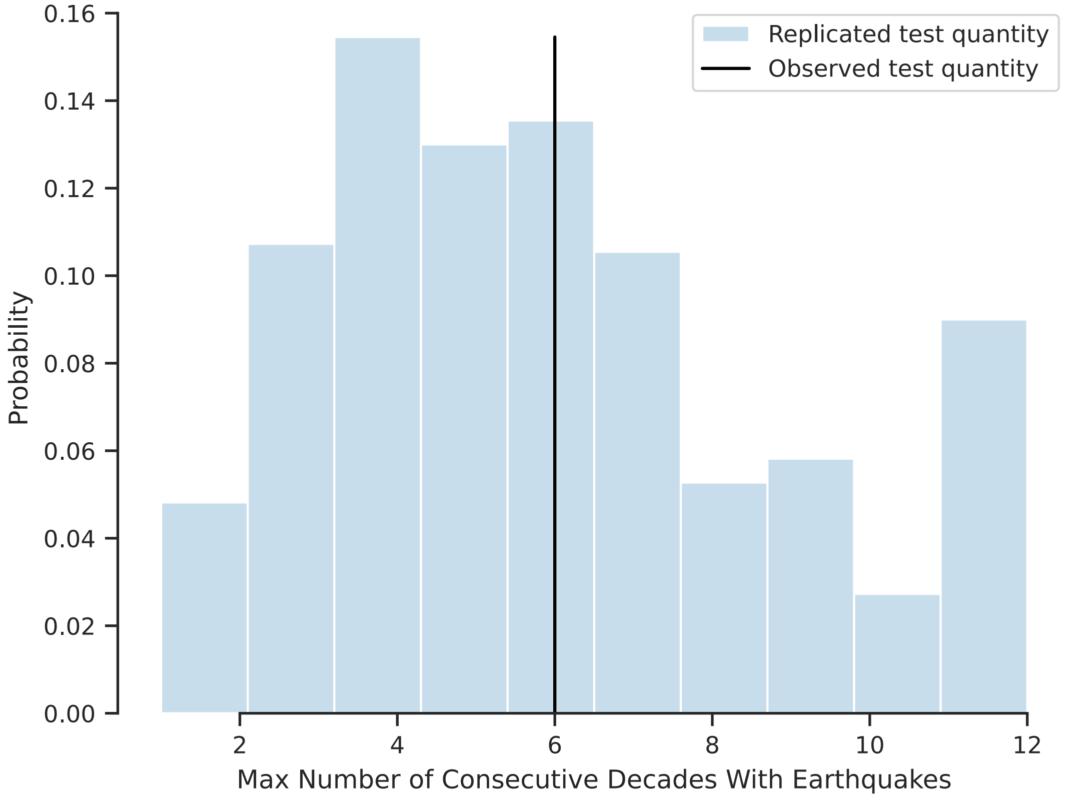
[ ]:
```
# Number of consecutive decades with no earthquakes
res1 = do_diagnostics(lambda_post, eq_data, T_eq_max_neq)
plt.gca().set_xlabel("Max Number of Consecutive Decades With No Earthquakes")
plt.gca().set_ylabel("Probability")
sns.despine(trim=True)
```

```
The observed test quantity is 1
The Bayesian p_value is 0.4060
```



```
[ ]: # Number of consecutive decades with earthquakes
     res2 = do_diagnostics(lambda_post, eq_data, T_eq_max_eq)
     plt.gca().set_xlabel("Max Number of Consecutive Decades With Earthquakes")
     plt.gca().set_ylabel("Probability")
     sns.despine(trim=True)
```

```
The observed test quantity is 6
The Bayesian p_value is 0.3670
```


Replicated test quantity
Observed test quantity
Probability
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2
4
6
8
10
12
Max Number of Consecutive Decades With Earthquakes