Homework 3

References

- Lectures 8-12 (inclusive).

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.

In [ ]:
```python
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)
```

Student details

- **First Name:** Kyle
- **Last Name:** Illenden
- **Email:** killende@purdue.edu

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:

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

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:

In [ ]:
```
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:

In [ ]:
```
solver = Solver()

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

y = solver(x)

print(y)
```

```
[[ 0.98881179  0.01936212]
 [ 0.95211389 -1.46191366]
 [ 0.84316008 -2.82374574]
 [ 0.67064144 -3.96315475]
 [ 0.44803856 -4.79465776]
 [ 0.19256899 -5.25667669]
 [-0.07614359 -5.3160928 ]
 [-0.33757614 -4.9706097 ]
 [-0.57187308 -4.24875771]
 [-0.76135369 -3.20755231]
 [-0.89184564 -1.92799759]
 [-0.95374415 -0.50878899]
 [-0.9427177   0.94129551]
 [-0.86000851  2.31172939]
 [-0.71230722  3.49865874]
 [-0.5112136   4.41276889]
 [-0.27232658  4.98600128]
 [-0.01403512  5.17661535]
 [ 0.24389679  4.97221969]
 [ 0.48184173  4.39055278]
 [ 0.68180183  3.47796655]
 [ 0.82876948  2.30574072]
 [ 0.91185292  0.96452037]
 [ 0.92508234 -0.44268948]
 [ 0.86783707 -1.80843114]
 [ 0.74486317 -3.02899673]
 [ 0.56588264 -4.01229993]
 [ 0.34482602 -4.6848198 ]
 [ 0.09874897 -4.99709336]
 [-0.15348302 -4.92734691]
 [-0.3926403  -4.48299993]
 [-0.60059456 -3.69993912]
 [-0.7616923  -2.63962997]
 [-0.86393249 -1.3842977 ]
 [-0.89985967 -0.03055487]
 [-0.86710621  1.31803062]
 [-0.76854438  2.55885581]
 [-0.61203877  3.59808577]
 [-0.40982023  4.35774625]
 [-0.17753086  4.78157004]
 [ 0.06698519  4.8391592 ]
 [ 0.30505192  4.52815535]
 [ 0.5185874   3.87426418]
 [ 0.69147651  2.92914403]
 [ 0.81078653  1.76633116]
 [ 0.86773376  0.47552244]
 [ 0.85832904 -0.84434019]
 [ 0.78365486 -2.09265056]
 [ 0.64975491 -3.17480397]
 [ 0.46714676 -4.00936367]
 [ 0.24999667 -4.53418567]
 [ 0.01502135 -4.71104147]
 [-0.21979834 -4.52839516]
 [-0.43659277 -4.00213381]
 [-0.61896286 -3.17420676]
 [-0.75321997 -2.10928919]
 [-0.82941266 -0.88973453]
 [-0.84206338  0.39079069]
 [-0.79056059  1.63449759]
 [-0.67917841  2.7469398 ]
 [-0.51672434  3.64418265]
 [-0.31584394  4.2591327 ]
 [-0.09203719  4.54655075]
 [ 0.1375368   4.48637381]
 [ 0.35537487  4.08510284]
 [ 0.54496377  3.37516111]
 [ 0.69203134  2.41228348]
 [ 0.78562031  1.27114594]
 [ 0.8189039   0.03957785]
 [ 0.78968243 -1.18819489]
 [ 0.70052526 -2.318756  ]
 [ 0.55854879 -3.26660369]
 [ 0.37484981 -3.96061378]
 [ 0.16363901 -4.34937306]
 [-0.05885777 -4.40498302]
 [-0.27564548 -4.12505305]
 [-0.47025746 -3.53274127]
 [-0.62800572 -2.67485108]
 [-0.7370889  -1.61813849]
 [-0.78947444 -0.4441215 ]
```

```
[-0.78148901  0.75720524]
[-0.71407404  1.89426617]
[-0.59268849  2.88088272]
[-0.42686849  3.64280192]
[-0.22947905  4.1232814 ]
[-0.01571661  4.28731084]
[ 0.19806036  4.12415626]
[ 0.39558271  3.64804269]
[ 0.56190824  2.89693364]
[ 0.68455149  1.92951081]
[ 0.7544204   0.82059367]
[ 0.76648943 -0.34464374]
[ 0.72015958 -1.47721199]
[ 0.61927948 -2.49109978]
[ 0.47182793 -3.30980315]
[ 0.28928386 -3.87209514]
[ 0.08573341 -4.13660093]
[-0.12321651 -4.0848368 ]
[-0.32163434 -3.72249032]
[-0.49447808 -3.07885383]]
```

Notice the dimension of `y`:

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

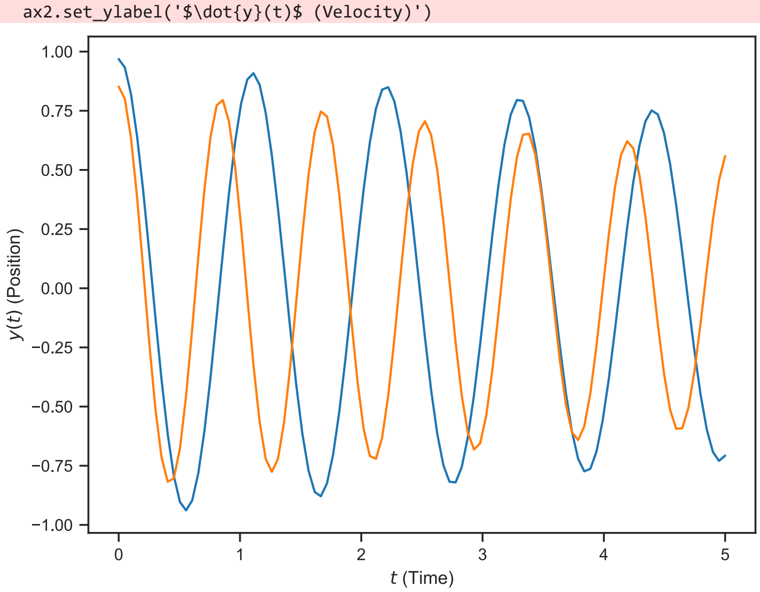
Out[ ]:
```
(100, 2)
```

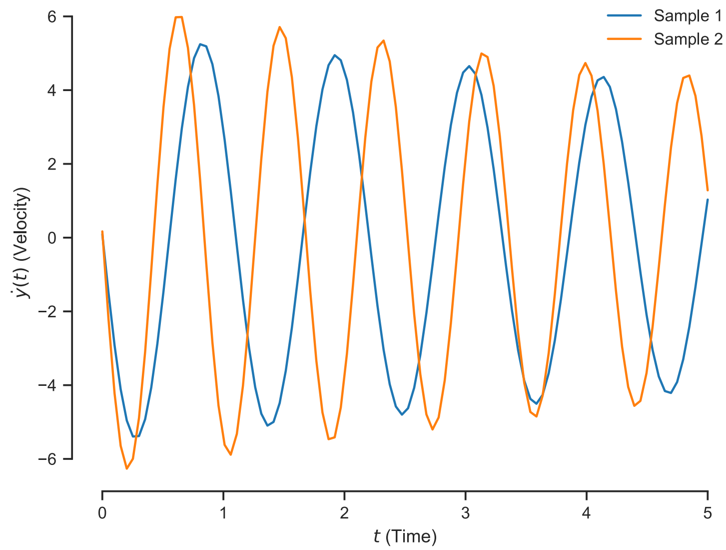
The 100 rows corresponds to timesteps. The 2 columns correspond to position and velocity.

Let's plot a few samples:

In [ ]:
```
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);
```

```
<>:7: SyntaxWarning: invalid escape sequence '\d'
<>:7: SyntaxWarning: invalid escape sequence '\d'
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\110166671.py:7: SyntaxWarning: invalid escape sequence '\d'
  ax2.set_ylabel('$\dot{y}(t)$ (Velocity)')
```





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

In [ ]:
```
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:

In [ ]:
```
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:

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

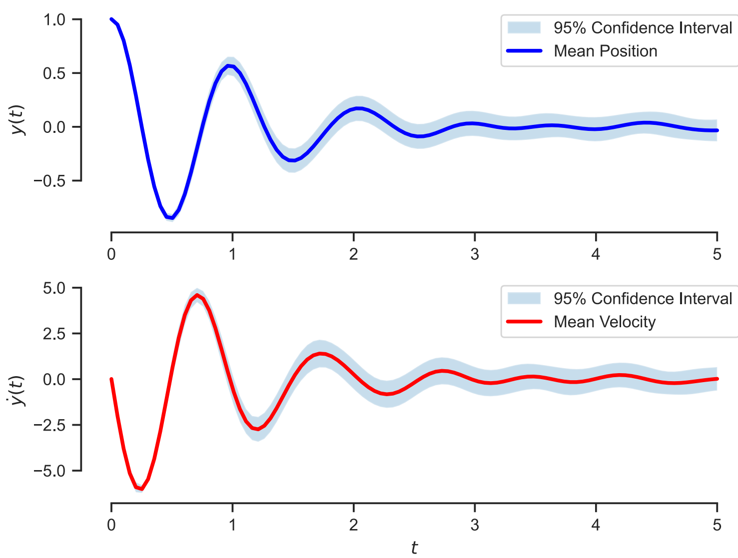
Out[ ]:
```
array([ 0.03221532, -2.05494003, -3.90391268, -5.31441866, -6.13537134,
       -6.28105021, -5.74002634, -4.57591771, -2.91994626, -0.9561636 ])
```

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).

In [ ]:
```
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

N=100

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

```python
# Find the standard deviation of position
d2_pos = abs(samples[:, :, 0] - mean_pos)**2
var_pos = np.sum(d2_pos,axis=0) / (N-1)
std_pos = var_pos**0.5

# Find the standard deviation of velocity
d2_vel = abs(samples[:, :, 1] - mean_vel)**2
var_vel = np.sum(d2_vel,axis=0) / (N-1)
std_vel = var_vel**0.5

# Using the Z score to get a 95% confidence interval
CI_pos = 1.96 * std_pos / np.sqrt(N)
CI_vel = 1.96 * std_vel / np.sqrt(N)

# Using the CI, find the bounds
pos_lower = mean_pos - CI_pos
pos_upper = mean_pos + CI_pos

vel_lower = mean_vel - CI_vel
vel_upper = mean_vel + CI_vel

# A common plot for all estimates
fig, axs = plt.subplots(2)
# Shaded area for the interval
axs[0].fill_between(
    np.linspace(0, 5, N),
    pos_lower,
    pos_upper,
    alpha=0.25
)

axs[0].plot(np.linspace(0, 5, N), mean_pos, 'b', lw=2)
# Shaded area for the interval
axs[1].fill_between(
    np.linspace(0, 5, N),
    vel_lower,
    vel_upper,
    alpha=0.25
)

axs[1].plot(np.linspace(0, 5, N), mean_vel, 'r', lw=2)

axs[1].set_xlabel('$t$')
axs[0].set_ylabel(r'$y(t)$')
axs[1].set_ylabel(r'$\dot{y}(t)$')
axs[0].legend(['95% Confidence Interval','Mean Position'])
axs[1].legend(['95% Confidence Interval','Mean Velocity'])
sns.despine(trim=True);
```

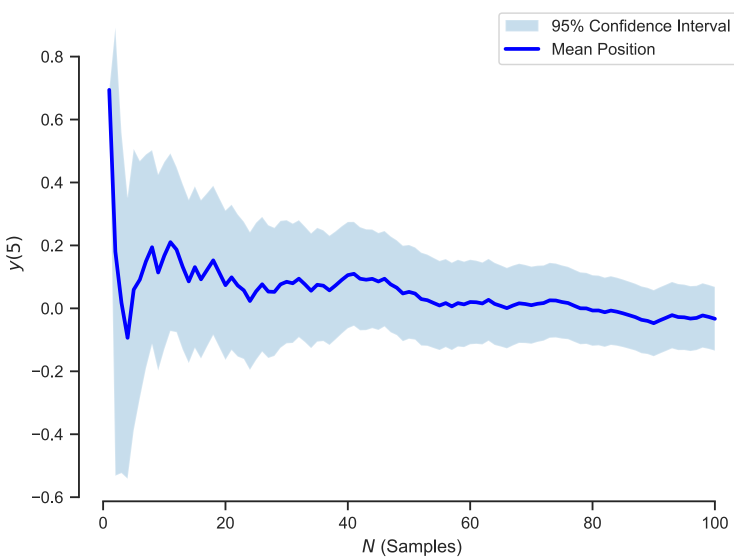


Part B

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

Solution:

In [ ]:
```
# Your code here
N=100

mean_pos = []
pos_lower = []
pos_upper = []
for i in range(0,N): # Running a for loop to execute this per number of samples
    if i == 0: # Ignoring the first sample to get past warnings
        mean_pos.append(samples[i, -1, 0])
        pos_lower.append(samples[i, -1, 0])
        pos_upper.append(samples[i, -1, 0])
        continue
    pos = samples[:i+1, -1, 0] # The +1 allows for the inclusion of i. So it is inclusive of i, but exclusive of i+1
    # Find the mean of the position
    mean_pos_value = np.mean(pos)
    # Find the standard deviation of position
    d2_pos = abs(pos - mean_pos_value)**2
    var_pos = np.sum(d2_pos,axis=0) / (i+1)
    std_pos = var_pos**0.5
    # Using the Z score to get a 95% confidence interval
    CI_pos = 1.96 * std_pos / np.sqrt(i+1)
    # Using the CI, find the bounds
    pos_lower_value = mean_pos_value - CI_pos
    pos_upper_value = mean_pos_value + CI_pos
    mean_pos.append(mean_pos_value)
    pos_lower.append(pos_lower_value)
    pos_upper.append(pos_upper_value)

# A common plot for all estimates
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    np.arange(1, N+1),
    pos_lower,
    pos_upper,
    alpha=0.25
)
ax.plot(np.arange(1, N+1), mean_pos, 'b', lw=2)
ax.set_xlabel('$N$ (Samples)')
ax.set_ylabel(r'$y(5)$')
ax.legend(['95% Confidence Interval','Mean Position'])
sns.despine(trim=True);
```

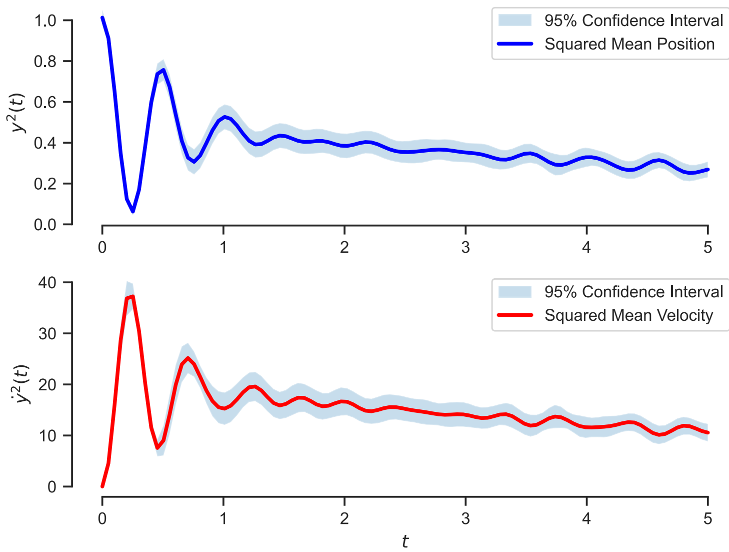


Part C

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:

In [ ]:
```
N=100

# Find the mean of the position and velocity
mean_pos = np.mean(samples[:, :, 0]**2, axis=0)
mean_vel = np.mean(samples[:, :, 1]**2, axis=0)

# Find the standard deviation of position
d2_pos = abs(samples[:, :, 0]**2 - mean_pos)**2
var_pos = np.sum(d2_pos,axis=0) / (N-1)
std_pos = var_pos**0.5

# Find the standard deviation of velocity
d2_vel = abs(samples[:, :, 1]**2 - mean_vel)**2
var_vel = np.sum(d2_vel,axis=0) / (N-1)
std_vel = var_vel**0.5

# Using the Z score to get a 95% confidence interval
CI_pos = 1.96 * std_pos / np.sqrt(N)
CI_vel = 1.96 * std_vel / np.sqrt(N)

# Using the CI, find the bounds
pos_lower = mean_pos - CI_pos
pos_upper = mean_pos + CI_pos

vel_lower = mean_vel - CI_vel
vel_upper = mean_vel + CI_vel


# A common plot for all estimates
fig, axs = plt.subplots(2)
# Shaded area for the interval
axs[0].fill_between(
    np.linspace(0, 5, N),
    pos_lower,
    pos_upper,
    alpha=0.25
)

axs[0].plot(np.linspace(0, 5, N), mean_pos, 'b', lw=2)
# Shaded area for the interval
axs[1].fill_between(
    np.linspace(0, 5, N),
    vel_lower,
    vel_upper,
    alpha=0.25
)

axs[1].plot(np.linspace(0, 5, N), mean_vel, 'r', lw=2)

axs[1].set_xlabel('$t$')
axs[0].set_ylabel(r'$y^2(t)$')
axs[1].set_ylabel(r'$\dot{y}^2(t)$')
axs[0].legend(['95% Confidence Interval','Squared Mean Position'])
axs[1].legend(['95% Confidence Interval','Squared Mean Velocity'])
sns.despine(trim=True);
```

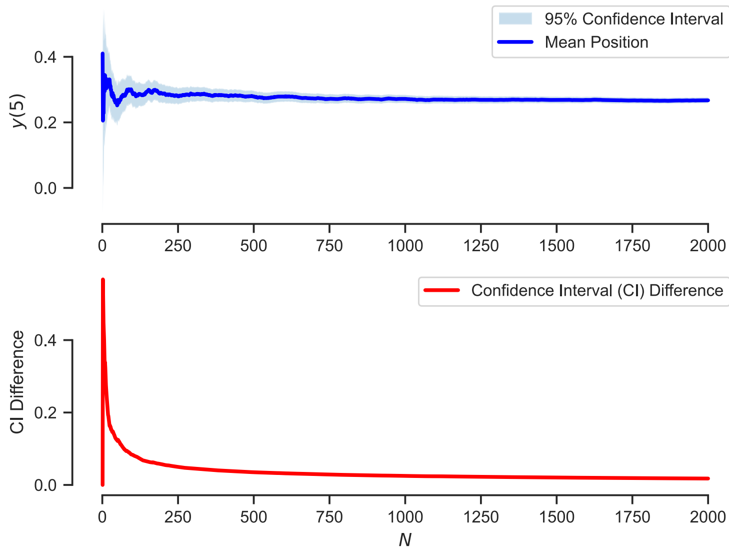


In [ ]:
```
samples = take_samples_from_solver(2000)
```

In [ ]:
```
# Your code here

N=2000

mean_pos = []
pos_lower = []
pos_upper = []
for i in range(0,N): # Running a for loop to execute this per number of samples
    if i == 0: # Ignoring the first sample to get past warnings
        mean_pos.append(samples[i, -1, 0]**2)
        pos_lower.append(samples[i, -1, 0]**2)
        pos_upper.append(samples[i, -1, 0]**2)
        continue
    pos = samples[:i+1, -1, 0]**2 # The +1 allows for the inclusion of i. So it is inclusive of i, but exclusive of i+1
    # Find the mean of the position
    mean_pos_value = np.mean(pos)
    # Find the standard deviation of position
    d2_pos = abs(pos - mean_pos_value)**2
    var_pos = np.sum(d2_pos,axis=0) / (i+1)
    std_pos = var_pos**0.5
    # Using the Z score to get a 95% confidence interval
    CI_pos = 1.96 * std_pos / np.sqrt(i+1)
    # Using the CI, find the bounds
    pos_lower_value = mean_pos_value - CI_pos
    pos_upper_value = mean_pos_value + CI_pos
    mean_pos.append(mean_pos_value)
    pos_lower.append(pos_lower_value)
    pos_upper.append(pos_upper_value)


mean_pos = np.asarray(mean_pos, dtype=np.float32)
pos_lower = np.asarray(pos_lower, dtype=np.float32)
pos_upper = np.asarray(pos_upper, dtype=np.float32)

for i in range(1,N):
    if (pos_upper[i]-pos_lower[i]) < 0.025:
        print(f"Confidence interval with N = {i} is {pos_upper[i]-pos_lower[i]}, which is within 0.025")
        future_N = i
        break
    if i == N:
        print(f"Not found = {pos_upper[-1]-pos_lower[-1]}")
        future_N = i


# A common plot for all estimates
# A common plot for all estimates
fig, axs = plt.subplots(2)
# Shaded area for the interval
axs[0].fill_between(
    np.arange(1, N + 1),
    pos_lower,
    pos_upper,
```

```
    alpha=0.25
)
# Here is the MC estimate:
axs[0].plot(np.arange(1, N+1), mean_pos, 'b', lw=2)
axs[1].plot(np.arange(1, N+1), (pos_upper-pos_lower), 'r', lw=2)
# The true value
#ax.plot(np.arange(1, N+1), [0.965] * N, color='r')
# and the labels
axs[1].set_xlabel('$N$')
axs[0].set_ylabel(r'$y(5)$')
axs[1].set_ylabel(r'CI Difference')
axs[0].legend(['95% Confidence Interval','Mean Position'])
axs[1].legend(['Confidence Interval (CI) Difference'])
sns.despine(trim=True);
```

```
Confidence interval with N = 997 is 0.024976342916488647, which is within 0.025
```



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:

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

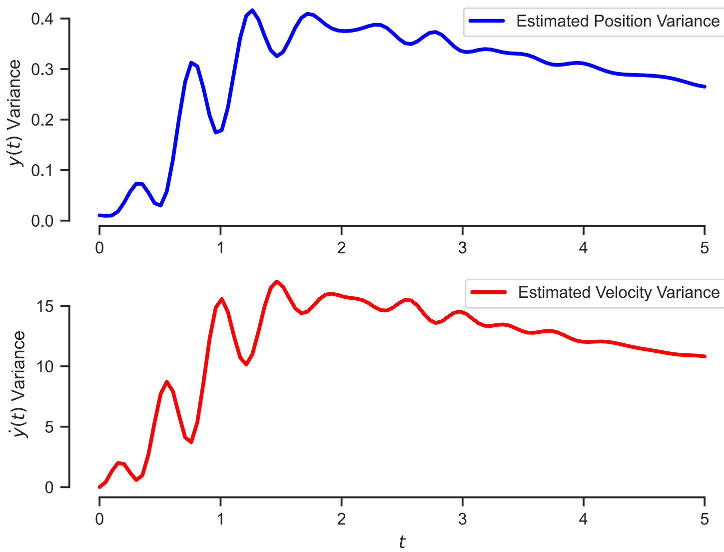
and similarly, for $\dot{y}(t)$, to estimate the position and velocity variance with negligible epistemic uncertainty. Plot both quantities as a function of time.

Solution:

In [ ]:

```
# Your code here

samples = take_samples_from_solver(future_N)
N=100


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

# Find the squared mean of the position and velocity
mean_2_pos_mean = np.mean((samples[:, :, 0]**2),axis=0)
mean_2_vel_mean = np.mean((samples[:, :, 1]**2),axis=0)

# Find the variance of the position and velocity
var_pos = mean_2_pos_mean-(mean_pos**2)
var_vel = mean_2_vel_mean-(mean_vel**2)

# A common plot for all estimates
fig, axs = plt.subplots(2)
axs[0].plot(np.linspace(0, 5, N), var_pos, 'b', lw=2)
axs[1].plot(np.linspace(0, 5, N), var_vel, 'r', lw=2)
axs[1].set_xlabel('$t$')
axs[0].set_ylabel(r'$y(t)$ Variance')
axs[1].set_ylabel(r'$\dot{y}(t)$ Variance')
axs[0].legend(['Estimated Position Variance'])
```

```
axs[1].legend(['Estimated Velocity Variance'])
sns.despine(trim=True);
```



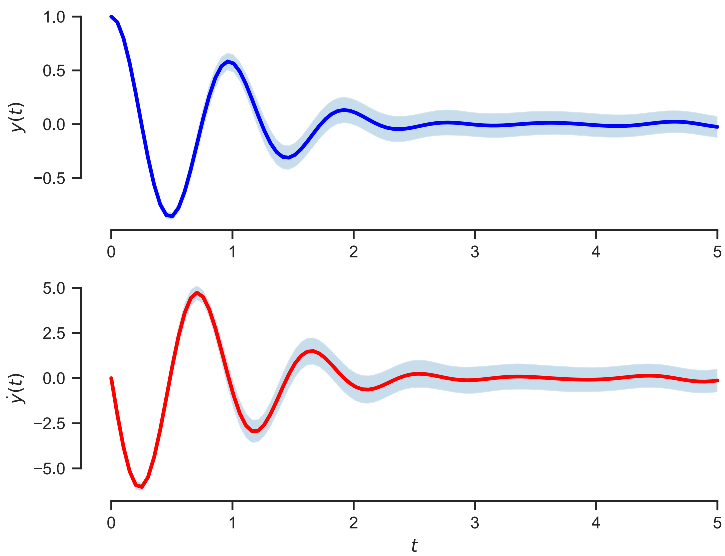
Part E

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:

In [ ]:
```
# Find the standard deviation of position and velocity
std_pos = var_pos**0.5
std_vel = var_vel**0.5

# Using the Z score to get a 95% confidence interval
CI_pos = 1.96 * std_pos / np.sqrt(N)
CI_vel = 1.96 * std_vel / np.sqrt(N)

# Using the CI, find the bounds
pos_lower = mean_pos - CI_pos
pos_upper = mean_pos + CI_pos

vel_lower = mean_vel - CI_vel
vel_upper = mean_vel + CI_vel

# A common plot for all estimates
fig, axs = plt.subplots(2)
# Shaded area for the interval
axs[0].fill_between(
    np.linspace(0, 5, N),
    pos_lower,
    pos_upper,
    alpha=0.25
)

axs[0].plot(np.linspace(0, 5, N), mean_pos, 'b', lw=2)
# Shaded area for the interval
axs[1].fill_between(
    np.linspace(0, 5, N),
    vel_lower,
    vel_upper,
    alpha=0.25
)

axs[1].plot(np.linspace(0, 5, N), mean_vel, 'r', lw=2)
axs[1].set_xlabel('$t$')
axs[0].set_ylabel(r'$y(t)$')
axs[1].set_ylabel(r'$\dot{y}(t)$')

sns.despine(trim=True);
```



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.

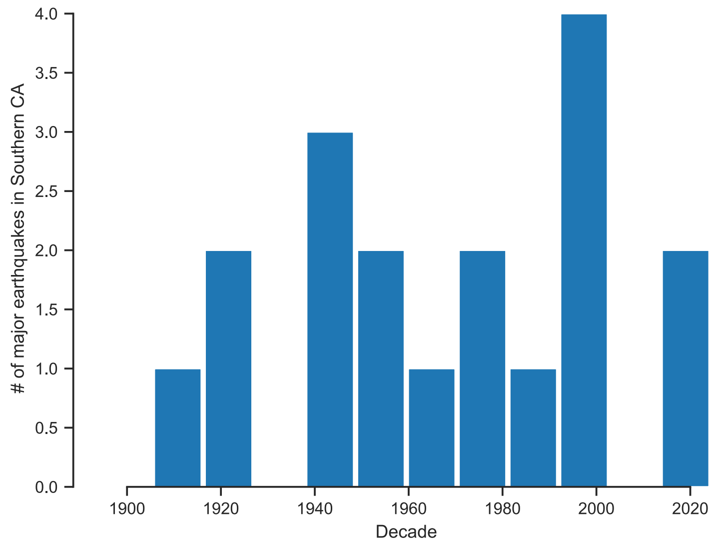
In [ ]:

```
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:

In [ ]:

```
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:

The probability mass function is:

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

Assuming all the measurements are independent, then the joint PMF is the product of the individual PMFs.

$$p(x_{1:N}|\lambda) = \frac{\lambda^{x_1}}{x_1!}e^{-\lambda} * \frac{\lambda^{x_2}}{x_2!}e^{-\lambda} * \ldots * \frac{\lambda^{x_N}}{x_N!}e^{-\lambda}$$

$$p(x_{1:N}|\lambda) = e^{-\lambda * N}(\frac{\lambda^{x_1}}{x_1!} * \frac{\lambda^{x_2}}{x_2!} * \ldots * \frac{\lambda^{x_N}}{x_N!}) = (\lambda^{\sum_{n=1}^{N} x_n})e^{-\lambda * N}(\frac{1}{x_1!} * \frac{1}{x_2!} * \ldots * \frac{1}{x_N!}) = (\lambda^{\sum_{n=1}^{N} x_n})\frac{e^{-\lambda * N}}{\prod_{n=1}^{N} x_n!}$$

Knowing

$$x_1 = 0, x_2 = 1, x_3 = 2, x_4 = 0, x_5 = 3, x_6 = 2, x_7 = 1, x_8 = 2, x_9 = 1, x_{10} = 4, x_{11} = 0, x_{12} = 2$$

and

$$\sum_{n=1}^{12} x_n = 0 + 1 + 2 + 0 + 3 + 2 + 1 + 2 + 1 + 4 + 0 + 2 = 18$$

and

$$\prod_{n=1}^{12} x_n! = 0 * 1 * 2 * 0 * 3 * 2 * 1 * 2 * 1 * 4 * 0 * 2 = 2304$$

$$p(x_{1:12}|\lambda) = (\lambda^{\sum_{n=1}^{12} x_n})\frac{e^{-\lambda * 12}}{\prod_{n=1}^{12} x_n!} = (\lambda^{18})\frac{e^{-\lambda * 12}}{2304}$$

In [ ]:
```
import math

print('Verifying the markdown above')

print('sum = '+str(np.sum(eq_data)))

product = 1
for i in range(len(eq_data)):
    product = product*math.factorial(eq_data[i])
print('product = '+str(product))
```

```
Verifying the markdown above
sum = 18
product = 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:

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

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

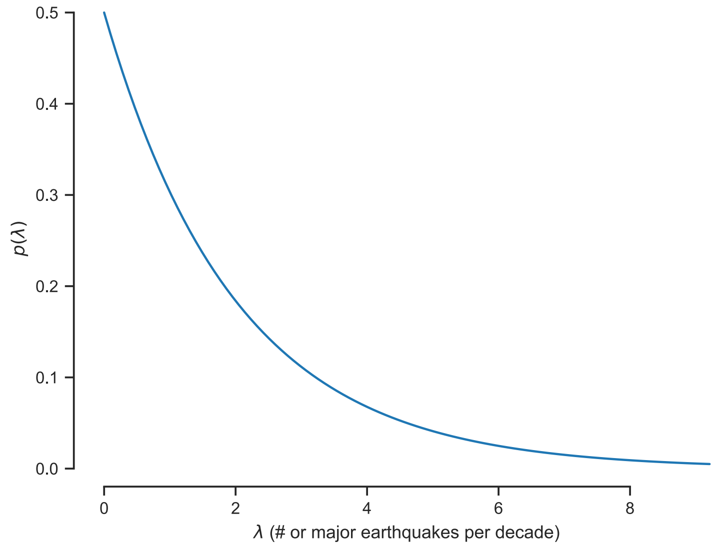
$$p(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)},$$

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 $\alpha = 1$).

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

# 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);
```

```
<>:15: SyntaxWarning: invalid escape sequence '\l'
<>:16: SyntaxWarning: invalid escape sequence '\l'
<>:15: SyntaxWarning: invalid escape sequence '\l'
<>:16: SyntaxWarning: invalid escape sequence '\l'
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\1634272363.py:15: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\1634272363.py:16: SyntaxWarning: invalid escape sequence '\l'
  ax.set_ylabel('$p(\lambda)$')
```



C. Show that the posterior of λ conditioned on $x_{1:N}$ is also a Gamma, but with updated hyperparameters.
Hint: When you write down the posterior of λ 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:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

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

$$\Rightarrow p(\lambda|x_{1:N}) \propto [(\lambda^{\sum_{n=1}^N x_n})\frac{e^{-\lambda N}}{\prod_{n=1}^N x_n!}][\frac{\beta^\alpha \lambda^{\alpha-1}e^{-\beta\lambda}}{\Gamma(\alpha)}]$$

$$\Rightarrow p(\lambda|x_{1:N}) \propto [(\lambda^{\sum_{n=1}^N x_n})e^{-\lambda N}][\lambda^{\alpha-1}e^{-\beta\lambda}]$$

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

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

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

so the updated values α and β in the Gamma distribution are:

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

$\beta \to N + \beta$

$$\Rightarrow \lambda|x_{1:N} \sim \mathrm{Gamma}\left(\alpha + \sum_{n=1}^N x_n, N+\beta\right)$$

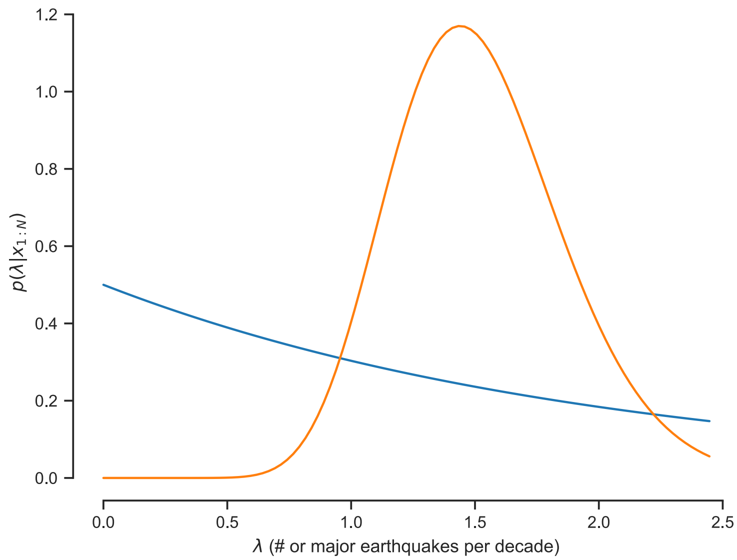
D. Prior-likelihood pairs that result in a posterior with the same form as the prior are known as conjugate distributions. Conjugate distributions are your only hope for analytical Bayesian inference. As a verification check, look at the Wikipedia page for conjugate priors, locate the Poisson-Gamma pair,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

E. Plot the prior and the posterior of λ on the same plot.

In [ ]:
```
# Your expression for alpha posterior here:
alpha_post = alpha + np.sum(eq_data)
# Your expression for beta posterior here:
beta_post = beta + len(eq_data)
# 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))
```

```
ax.plot(lambdas, lambda_post.pdf(lambdas))
ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
ax.set_ylabel('$p(\lambda|x_{1:N})$')
sns.despine(trim=True);
```

```
<>:13: SyntaxWarning: invalid escape sequence '\l'
<>:14: SyntaxWarning: invalid escape sequence '\l'
<>:13: SyntaxWarning: invalid escape sequence '\l'
<>:14: SyntaxWarning: invalid escape sequence '\l'
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\496041326.py:13: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\496041326.py:14: SyntaxWarning: invalid escape sequence '\l'
  ax.set_ylabel('$p(\lambda|x_{1:N})$')
```



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:

In [ ]:
```
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):
        lambda_sample = lambda_post.rvs()
        samples[i] = np.random.poisson(lambda_sample)
    return samples
```

Test your code here:

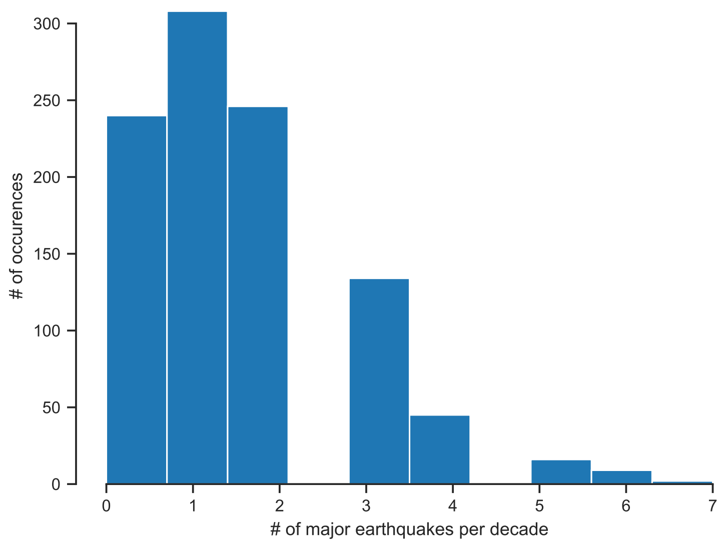
In [ ]:
```
samples = sample_posterior_predictive(10, lambda_post)
samples
```

Out[ ]:
```
array([1, 2, 1, 4, 1, 2, 2, 1, 2, 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.

In [ ]:
```
samples = sample_posterior_predictive(1000, lambda_post)
fig, ax = plt.subplots()
ax.hist(samples)
ax.set_xlabel('# of major earthquakes per decade')
ax.set_ylabel('# of occurences')
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.

In [ ]:
```
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 = (1 - np.mean(samples == 0))

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

```
p(X >= 1 | data) = 0.7574
```

I. Find a 95% credible interval for λ.

In [ ]:
```
# Calculate the 95% credible interval
lower_bound = lambda_post.ppf(0.025)
upper_bound = lambda_post.ppf(0.975)

# Output the result
print(f"95% credible interval for λ: ({lower_bound:.4f}, {upper_bound:.4f})")
```

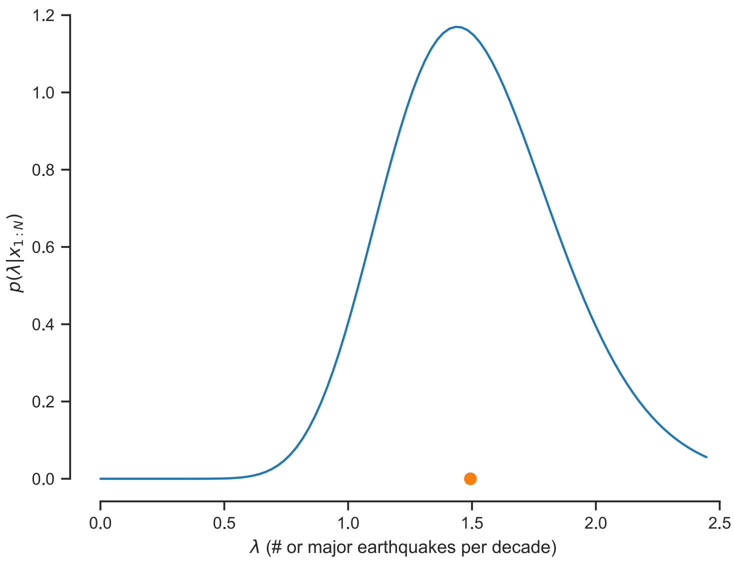
```
95% credible interval for λ: (0.9151, 2.2758)
```

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 $p(x|\lambda^*_N)$.

In [ ]:
```
lambda_star_N = lambda_post.median()
print(f'lambda_star_N = {lambda_star_N:.2f}')
```

```python
# Plot it
lambdas = np.linspace(0, lambda_post.ppf(0.99), 100)
fig, ax = plt.subplots()
ax.plot(lambdas, lambda_post.pdf(lambdas))
ax.plot(lambda_star_N, 0, 'o', markeredgewidth=2, label=r'$\theta^*_{01}$')
ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
ax.set_ylabel('$p(\lambda|x_{1:N})$')
sns.despine(trim=True);
```

```
lambda_star_N = 1.49
<>:9: SyntaxWarning: invalid escape sequence '\l'
<>:10: SyntaxWarning: invalid escape sequence '\l'
<>:9: SyntaxWarning: invalid escape sequence '\l'
<>:10: SyntaxWarning: invalid escape sequence '\l'
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\2008742213.py:9: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
C:\Users\socce\AppData\Local\Temp\ipykernel_4624\2008742213.py:10: SyntaxWarning: invalid escape sequence '\l'
  ax.set_ylabel('$p(\lambda|x_{1:N})$')
```



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

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

In [ ]:
```python
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, :] = np.random.poisson(post_rv.rvs(size=n))
    return x_rep
```

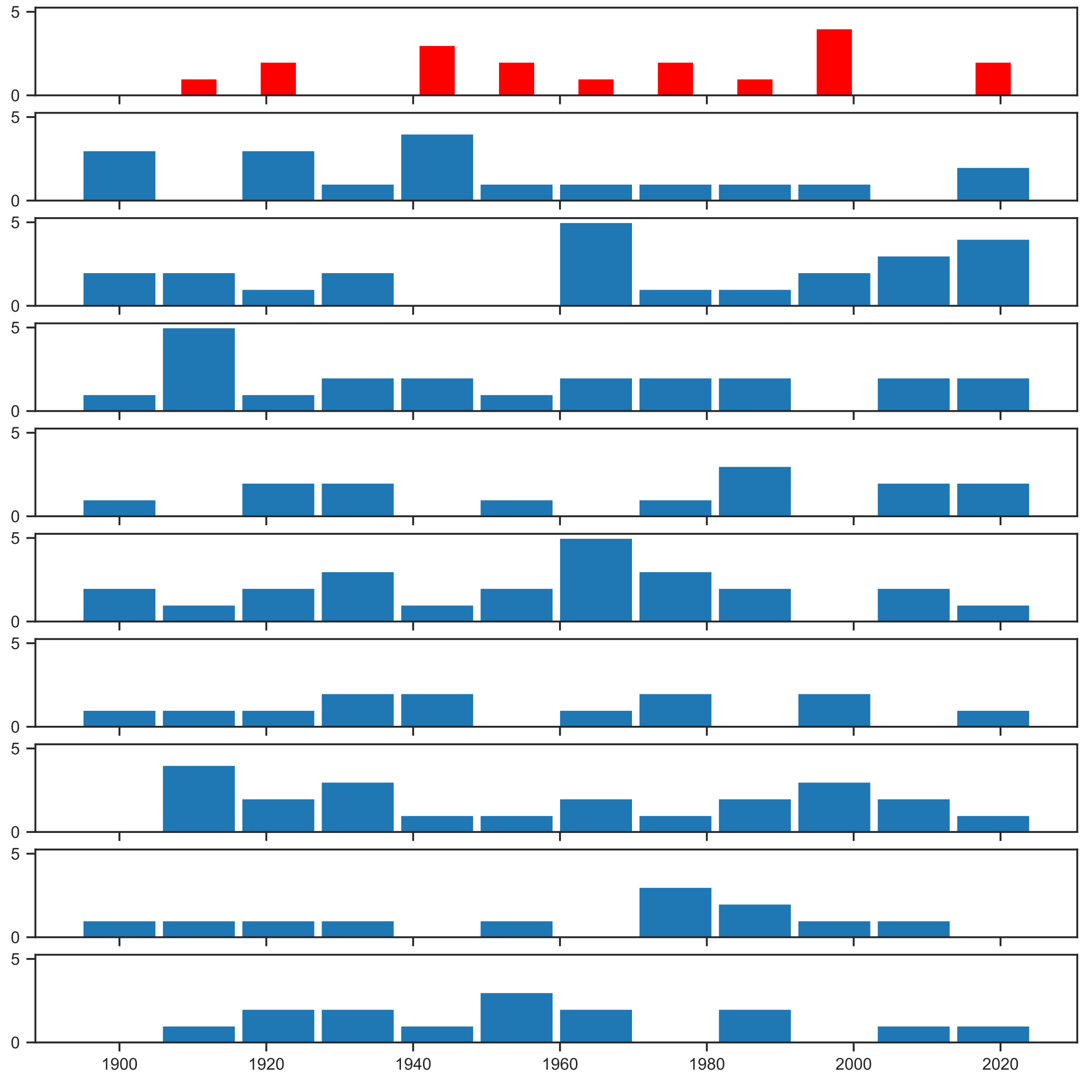
Try your code here:

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

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

If it works, then try the following visualization:

In [ ]:
```
#ADDED
n_rep = 9

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

In [ ]:
```
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
        )
    )
```

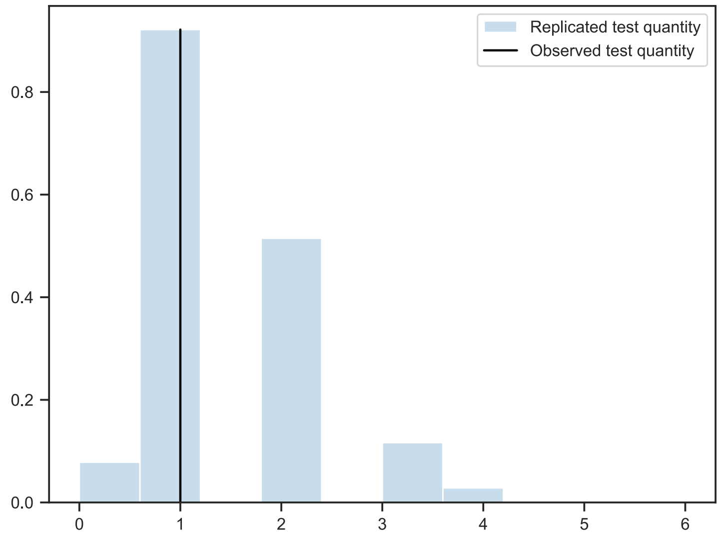
```python
    p_val = (
        np.sum(np.ones((n_rep,))[T_rep > T_obs]) / n_rep
    )
    return dict(
        T_obs=T_obs,
        p_val=p_val,
        T_rep=T_rep,
        data_rep=data_rep
    )


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)
```

In [ ]:

```python
# 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().
```

```python
do_diagnostics(
    lambda_post,
    eq_data,
    T_eq_max_neq
)
```

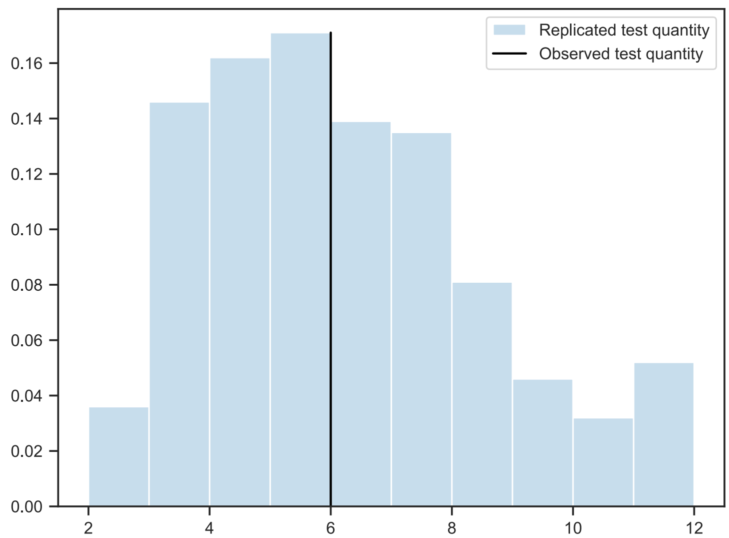
```
The observed test quantity is 1
The Bayesian p_value is 0.4000
```



In [ ]:

```python
# 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

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

```
The observed test quantity is 6
The Bayesian p_value is 0.3460
```


Replicated test quantity
Observed test quantity
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
2
4
6
8
10
12