Homework 3

References

- Lectures 7-12 (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 [95]:
```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, "savefig.dpi":300})
sns.set_context("notebook")
sns.set_style("ticks")

import scipy
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: Ben**

- **Last Name: McAteer**
- **Email: bmcateer@purdue.edu**

Problem 1 - Propagating uncertainty through a differential equation

This is a classic uncertainty propagation problem that you will have to 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 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 [96]:

```
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 [97]:

```
solver = Solver()

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

y = solver(x)

print(y)
```

```
[[ 1.153e+00  4.914e-03]
 [ 1.109e+00 -1.764e+00]
 [ 9.778e-01 -3.386e+00]
 [ 7.713e-01 -4.734e+00]
 [ 5.059e-01 -5.706e+00]
 [ 2.025e-01 -6.228e+00]
 [-1.149e-01 -6.261e+00]
 [-4.217e-01 -5.806e+00]
 [-6.938e-01 -4.901e+00]
 [-9.103e-01 -3.619e+00]
 [-1.055e+00 -2.062e+00]
 [-1.116e+00 -3.525e-01]
 [-1.090e+00  1.375e+00]
```

```
 [-9.791e-01  2.985e+00]
 [-7.926e-01  4.353e+00]
 [-5.453e-01  5.375e+00]
 [-2.568e-01  5.972e+00]
 [ 5.001e-02  6.099e+00]
 [ 3.512e-01  5.751e+00]
 [ 6.234e-01  4.956e+00]
 [ 8.454e-01  3.780e+00]
 [ 1.000e+00  2.315e+00]
 [ 1.076e+00  6.779e-01]
 [ 1.068e+00 -1.003e+00]
 [ 9.765e-01 -2.596e+00]
 [ 8.094e-01 -3.978e+00]
 [ 5.801e-01 -5.042e+00]
 [ 3.068e-01 -5.707e+00]
 [ 1.121e-02 -5.923e+00]
 [-2.836e-01 -5.677e+00]
 [-5.547e-01 -4.989e+00]
 [-7.811e-01 -3.916e+00]
 [-9.452e-01 -2.544e+00]
 [-1.035e+00 -9.811e-01]
 [-1.043e+00  6.495e-01]
 [-9.702e-01  2.220e+00]
 [-8.220e-01  3.609e+00]
 [-6.105e-01  4.709e+00]
 [-3.527e-01  5.435e+00]
 [-6.873e-02  5.734e+00]
 [ 2.190e-01  5.584e+00]
 [ 4.880e-01  5.000e+00]
 [ 7.175e-01  4.029e+00]
 [ 8.897e-01  2.749e+00]
 [ 9.916e-01  1.262e+00]
 [ 1.016e+00 -3.146e-01]
 [ 9.604e-01 -1.858e+00]
 [ 8.306e-01 -3.248e+00]
 [ 6.367e-01 -4.377e+00]
 [ 3.943e-01 -5.159e+00]
 [ 1.225e-01 -5.533e+00]
 [-1.573e-01 -5.475e+00]
 [-4.233e-01 -4.990e+00]
 [-6.548e-01 -4.119e+00]
 [-8.340e-01 -2.931e+00]
 [-9.471e-01 -1.522e+00]
 [-9.858e-01 -1.443e-03]
 [-9.474e-01  1.510e+00]
 [-8.354e-01  2.896e+00]
 [-6.588e-01  4.048e+00]
 [-4.319e-01  4.878e+00]
 [-1.726e-01  5.323e+00]
 [ 9.870e-02  5.351e+00]
 [ 3.608e-01  4.961e+00]
 [ 5.933e-01  4.187e+00]
 [ 7.783e-01  3.090e+00]
 [ 9.015e-01  1.759e+00]
 [ 9.538e-01  2.986e-01]
 [ 9.314e-01 -1.177e+00]
 [ 8.366e-01 -2.553e+00]
 [ 6.770e-01 -3.722e+00]
 [ 4.656e-01 -4.595e+00]
 [ 2.190e-01 -5.104e+00]
```

```
 [-4.321e-02 -5.213e+00]
 [-3.006e-01 -4.914e+00]
 [-5.332e-01 -4.234e+00]
 [-7.228e-01 -3.228e+00]
 [-8.551e-01 -1.976e+00]
 [-9.199e-01 -5.767e-01]
 [-9.127e-01  8.596e-01]
 [-8.344e-01  2.221e+00]
 [-6.914e-01  3.401e+00]
 [-4.954e-01  4.310e+00]
 [-2.618e-01  4.878e+00]
 [-9.125e-03  5.062e+00]
 [ 2.428e-01  4.851e+00]
 [ 4.745e-01  4.262e+00]
 [ 6.678e-01  3.345e+00]
 [ 8.080e-01  2.172e+00]
 [ 8.844e-01  8.360e-01]
 [ 8.915e-01 -5.575e-01]
 [ 8.290e-01 -1.900e+00]
 [ 7.022e-01 -3.086e+00]
 [ 5.214e-01 -4.026e+00]
 [ 3.010e-01 -4.646e+00]
 [ 5.829e-02 -4.901e+00]
 [-1.876e-01 -4.772e+00]
 [-4.174e-01 -4.272e+00]
 [-6.135e-01 -3.441e+00]
 [-7.606e-01 -2.347e+00]]
```

Notice the dimension of `y`:

In [98]:

```
y.shape
```

Out[98]:

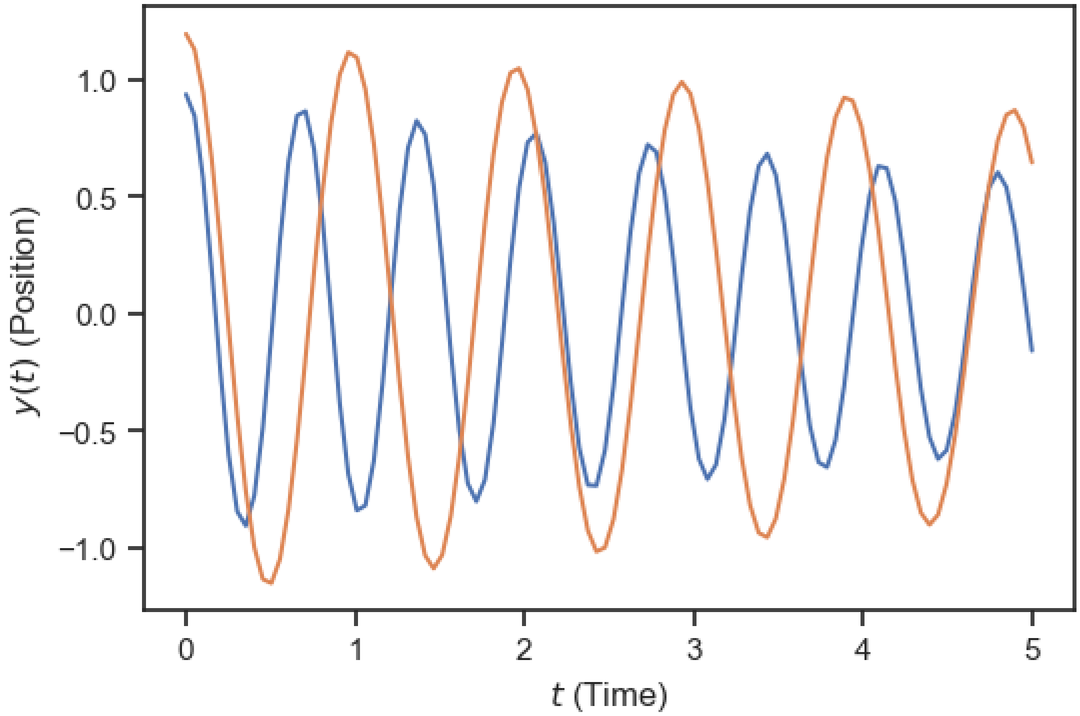
```
(100, 2)
```

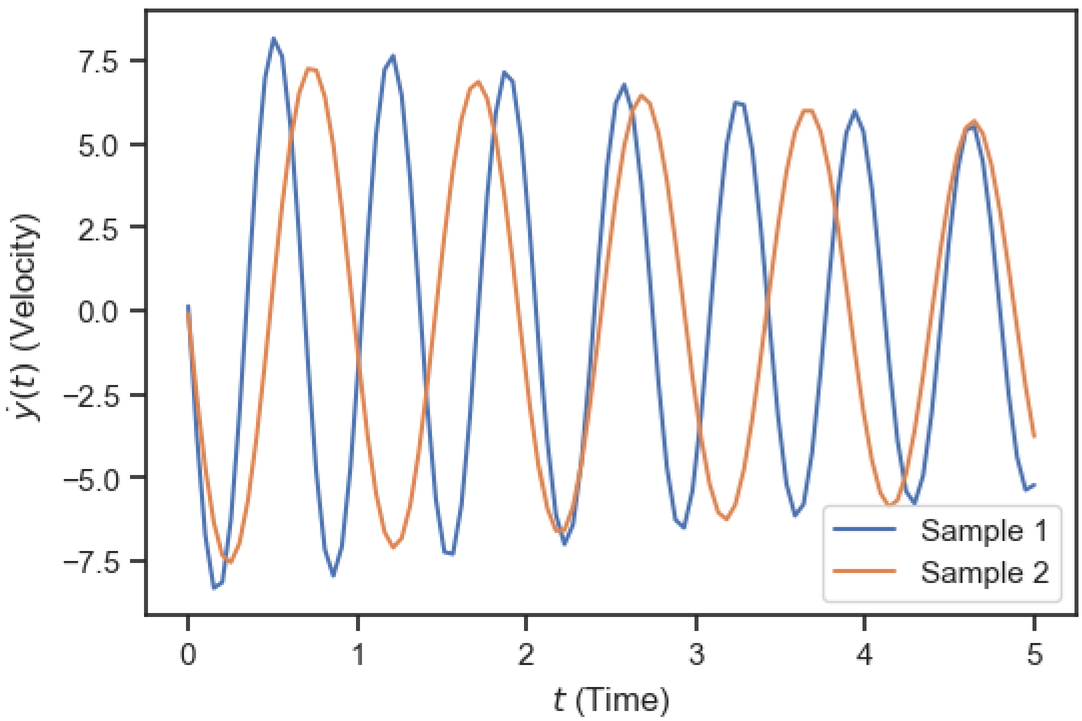
The 100 rows corresponds to timesteps. The 2 columns correspond to position and velocity.

Let's plot a few samples:

In [99]:

```
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");
```





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

In [100…

```
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 [101…

```
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 [102…

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

Out[102…

```
array([ 0.033, -1.787, -3.435, -4.761, -5.648, -6.017, -5.839, -5.133,
       -3.966, -2.446])
```

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 [103…

```
#Part A

N = 100 # Number of samples to take
samples = take_samples_from_solver(N)
# Sampled positions are: samples[:, :, 0]
# Sampled velocities are: samples[:, :, 1]
# Sampled position at the 10th timestep is: samples[:, 9, 0]
# etc.
pos = samples[:,:,0]
vel = samples[:,:,1]
time = solver.t

#need to create model with variance and mean and then graph both pos and velocity
posMean = np.mean(pos, axis = 0) #sample mean of position
velMean = np.mean(vel,axis = 0)#sample mean of velocity

posVar = np.var(pos, axis = 0)
velVar = np.var(vel,axis = 0)

# Evaluate the sample average for all sample sizes
I_running_pos = posMean
I_running_vel = velMean#np.cumsum(vel[0]) / np.arange(1, N + 1)

# Evaluate the sample average for the squared of Y
g2_running_pos = np.cumsum(pos[0] ** 2, axis = 0) / np.arange(1, N + 1)
g2_running_vel = np.cumsum(vel[0] ** 2, axis = 0) / np.arange(1, N + 1)
```

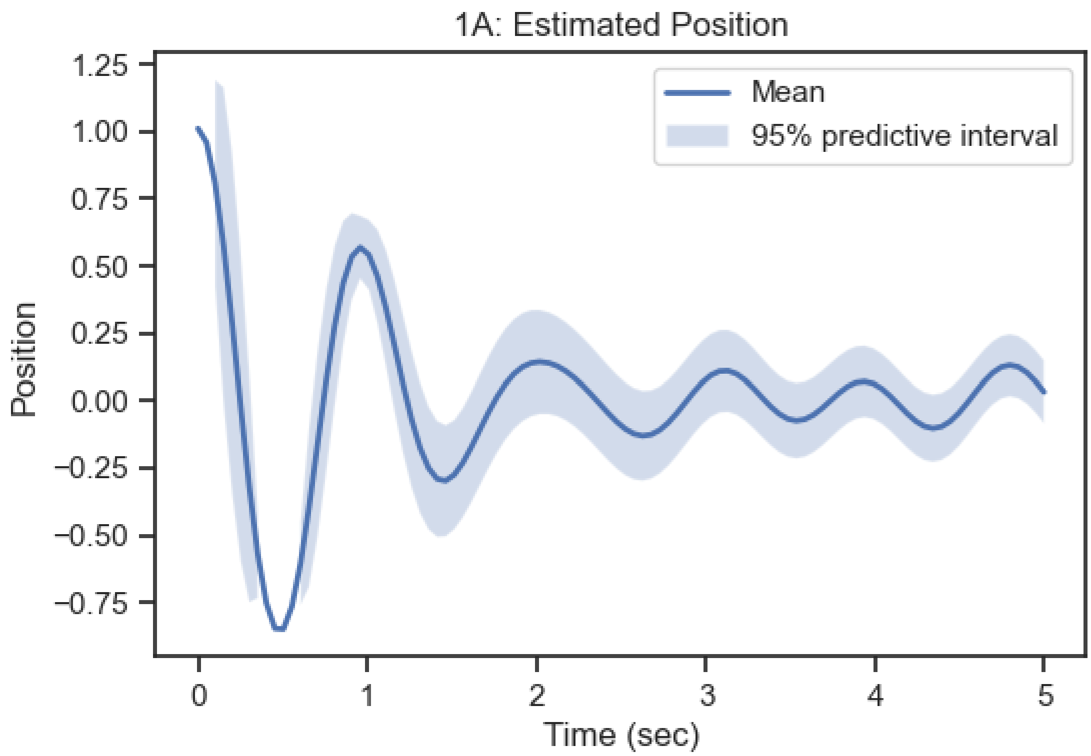
```python
# Evaluate the running average of the variance
sigma2_running_pos = g2_running_pos - I_running_pos ** 2
sigma2_running_vel = g2_running_vel - I_running_vel ** 2

# Alright, now we have quantified our uncertainty about I for every N
# from a single MC run. Let's plot a (about) 95% predictive interval
# Running lower bound for the predictive interval
I_lower_running_pos = (I_running_pos - 2.0 * np.sqrt(sigma2_running_pos / np.arange(1,
I_lower_running_vel = (I_running_vel - 2.0 * np.sqrt(sigma2_running_vel / np.arange(1,

# Running upper bound for the predictive interval
I_upper_running_pos = (I_running_pos + 2.0 * np.sqrt(sigma2_running_pos / np.arange(1,
I_upper_running_vel = (I_running_vel + 2.0 * np.sqrt(sigma2_running_vel / np.arange(1,
#1A Position Plot
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    time,
    I_lower_running_pos,
    I_upper_running_pos,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(time, posMean, 'b', lw=2, label='Mean')
ax.set_xlabel('Time (sec)')
ax.set_ylabel('Position');
plt.title('1A: Estimated Position')
plt.legend()

#1A Velocity Plot
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    time,
    I_lower_running_vel,
    I_upper_running_vel,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(time, velMean, 'b', lw=2, label='Mean')
ax.set_xlabel('Time (sec)')
ax.set_ylabel('Velocity');
plt.title('1A: Estimated Velocity')
plt.legend()
```

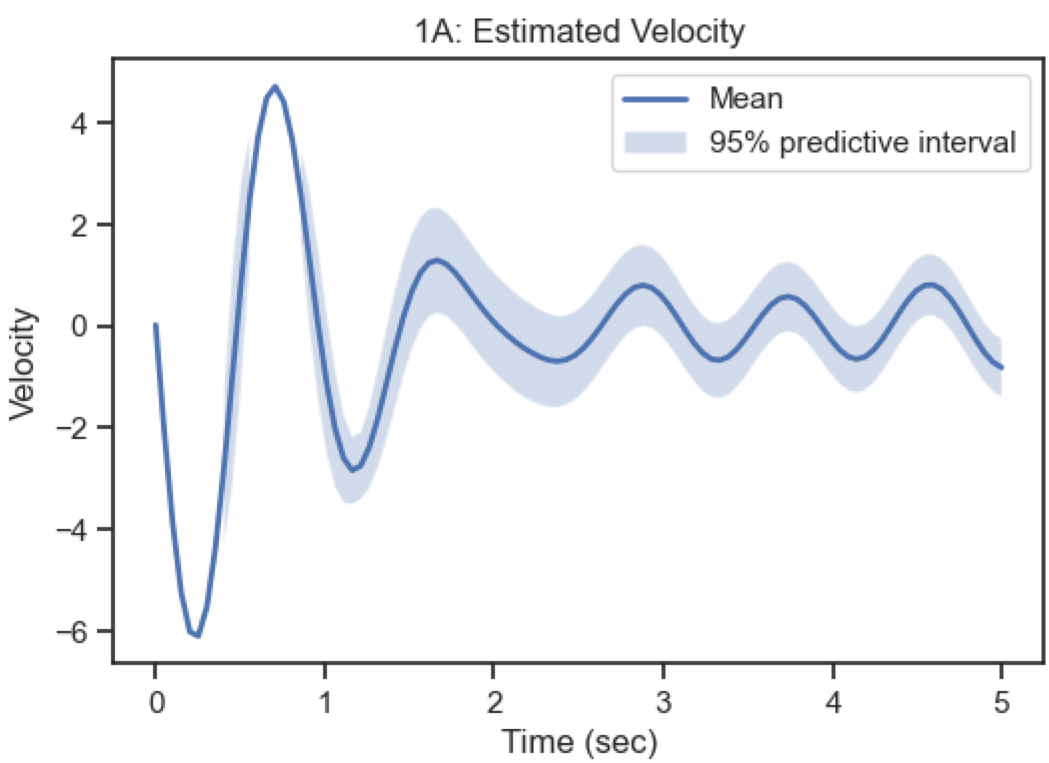
```
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3454814612.py:35: RuntimeWarning: invalid
value encountered in sqrt
  I_lower_running_pos = (I_running_pos - 2.0 * np.sqrt(sigma2_running_pos / np.arange(1,
N + 1)))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3454814612.py:36: RuntimeWarning: invalid
value encountered in sqrt
  I_lower_running_vel = (I_running_vel - 2.0 * np.sqrt(sigma2_running_vel / np.arange(1,
N + 1)))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3454814612.py:39: RuntimeWarning: invalid
value encountered in sqrt
  I_upper_running_pos = (I_running_pos + 2.0 * np.sqrt(sigma2_running_pos / np.arange(1,
N + 1)))
```

```
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3454814612.py:40: RuntimeWarning: invalid
value encountered in sqrt
  I_upper_running_vel = (I_running_vel + 2.0 * np.sqrt(sigma2_running_vel / np.arange(1,
N + 1)))
```

Out[103…

```
<matplotlib.legend.Legend at 0x130a11fefa0>
```





Part B

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

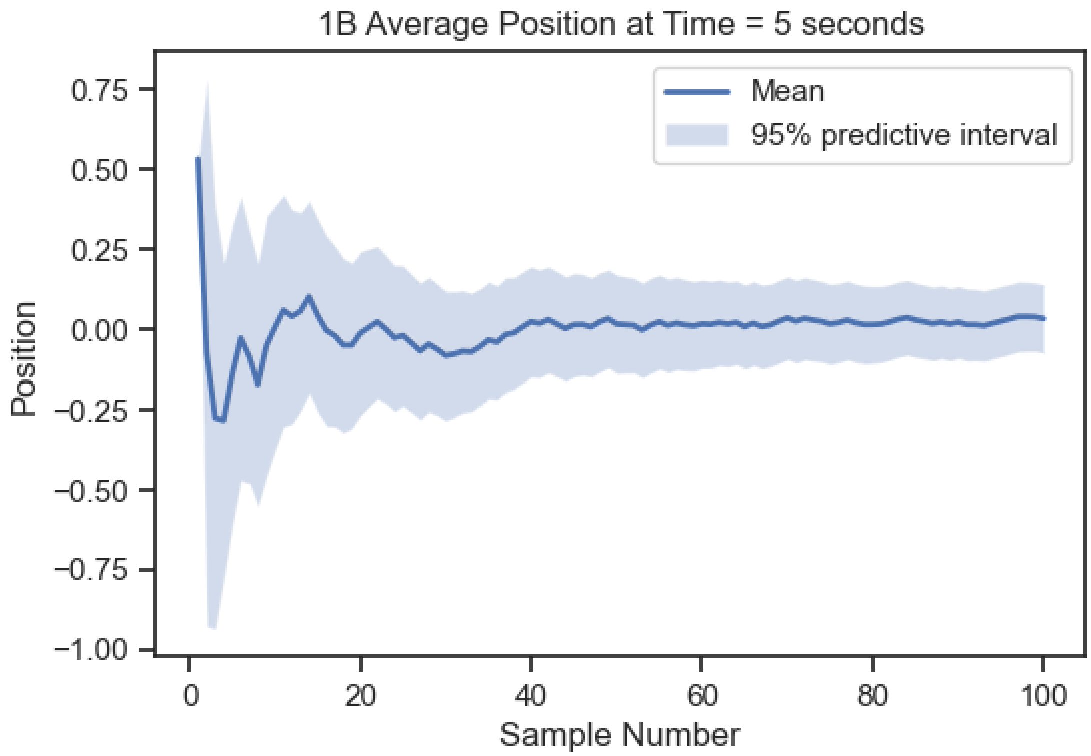
Solution:

In [104…

```
#Part B
posT = samples[:,99,0] #at five seconds
#velT = samples[:,99,1]

# Evaluate the sample average for all sample sizes
I_running_post = np.cumsum(posT) / np.arange(1, N + 1)
#I_running_velt = np.cumsum(velT) / np.arange(1, N + 1)

# Evaluate the sample average for the squared of Y
g2_running_post = np.cumsum(posT ** 2) / np.arange(1, N + 1)
#g2_running_velt = np.cumsum(velT ** 2)

# Evaluate the running average of the variance
sigma2_running_post = g2_running_post - I_running_post ** 2
#sigma2_running_velt = g2_running_velt - I_running_velt ** 2

# Alright, now we have quantified our uncertainty about I for every N
# from a single MC run. Let's plot a (about) 95% predictive interval
# Running lower bound for the predictive interval
I_lower_running_post = (I_running_post - 2.0 * np.sqrt(sigma2_running_post / np.arange(
#I_lower_running_velt = (I_running_velt - 2.0 * np.sqrt(sigma2_running_velt / np.arange

# Running upper bound for the predictive interval
I_upper_running_post = (I_running_post + 2.0 * np.sqrt(sigma2_running_post / np.arange(
#I_upper_running_velt = (I_running_velt + 2.0 * np.sqrt(sigma2_running_velt / np.arange


#1b Position Plot at t=5
fig, ax = plt.subplots()

ax.fill_between(
    np.arange(1, N + 1),
    I_lower_running_post,
    I_upper_running_post,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(np.arange(1, N + 1),I_running_post, 'b', lw=2, label = 'Mean')
ax.set_xlabel('Sample Number')
ax.set_ylabel('Position');
plt.title('1B Average Position at Time = 5 seconds')
plt.legend()
```

Out[104…

```
<matplotlib.legend.Legend at 0x130a11d40d0>
```



Part C

Repeat part A and B for the squared response. That is, do exactly 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: 5000 samples were used to estimate the mean squared response with negligible uncertainty

In [105…

```
#1C
NewN = 5000
SQsamples = take_samples_from_solver(NewN)


SQpos = SQsamples[:,:,0] ** 2
SQvel = SQsamples[:,:,1] ** 2

SQposMean = np.mean(SQpos, axis = 0) #sample mean of position
SQvelMean = np.mean(SQvel,axis = 0)#sample mean of velocity

SQposVar = np.var(SQpos, axis = 0)
SQvelVar = np.var(SQvel,axis = 0)

# Evaluate the sample average for all sample sizes
I_running_posSQ = SQposMean
I_running_velSQ = SQvelMean#np.cumsum(vel[0]) / np.arange(1, N + 1)

# Evaluate the sample average for the squared of Y
g2_running_posSQ = np.cumsum(SQpos[0] ** 2, axis = 0) / np.arange(1, N + 1)
g2_running_velSQ = np.cumsum(SQvel[0] ** 2, axis = 0) / np.arange(1, N + 1)

# Evaluate the running average of the variance
```

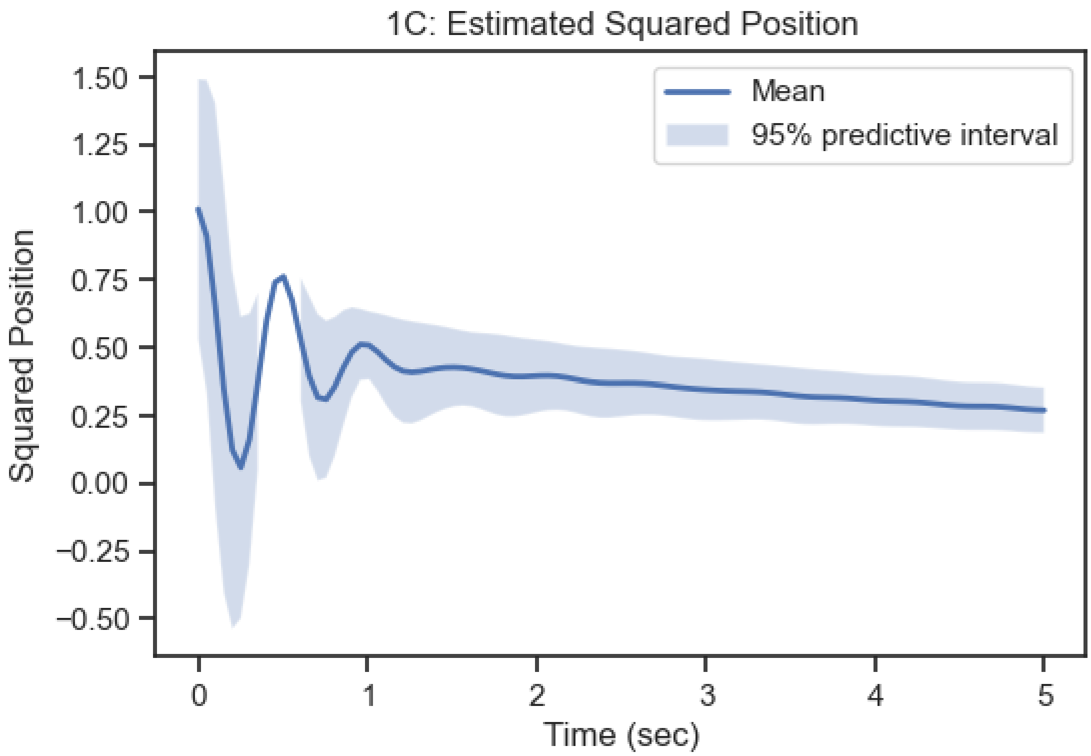
```
sigma2_running_posSQ = g2_running_posSQ - I_running_posSQ ** 2
sigma2_running_velSQ = g2_running_velSQ - I_running_velSQ ** 2

# Alright, now we have quantified our uncertainty about I for every N
# from a single MC run. Let's plot a (about) 95% predictive interval
# Running lower bound for the predictive interval
I_lower_running_posSQ = (I_running_posSQ - 2.0 * np.sqrt(sigma2_running_posSQ / np.aran
I_lower_running_velSQ = (I_running_velSQ - 2.0 * np.sqrt(sigma2_running_velSQ / np.aran

# Running upper bound for the predictive interval
I_upper_running_posSQ = (I_running_posSQ + 2.0 * np.sqrt(sigma2_running_posSQ / np.aran
I_upper_running_velSQ = (I_running_velSQ + 2.0 * np.sqrt(sigma2_running_velSQ / np.aran
#1C Position Plot
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    time,
    I_lower_running_posSQ,
    I_upper_running_posSQ,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(time, SQposMean, 'b', lw=2, label='Mean')
ax.set_xlabel('Time (sec)')
ax.set_ylabel('Squared Position');
plt.title('1C: Estimated Squared Position')
plt.legend()

#1C Velocity Plot
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    time,
    I_lower_running_velSQ,
    I_upper_running_velSQ,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(time, SQvelMean, 'b', lw=2, label='Mean')
ax.set_xlabel('Time (sec)')
ax.set_ylabel('Squared Velocity');
plt.title('1C: Estimated Squared Velocity')
plt.legend()


SQposT = SQpos[:,99]   #at five seconds
#velT = samples[:,99,1]

# Evaluate the sample average for all sample sizes
I_running_postSQ = np.cumsum(SQposT) / np.arange(1, NewN + 1)

# Evaluate the sample average for the squared of Y
g2_running_postSQ = np.cumsum(SQposT ** 2) / np.arange(1, NewN + 1)

# Evaluate the running average of the variance
sigma2_running_postSQ = g2_running_postSQ - I_running_postSQ ** 2

I_lower_running_postSQ = (I_running_postSQ - 2.0 * np.sqrt(sigma2_running_postSQ / np.a
```

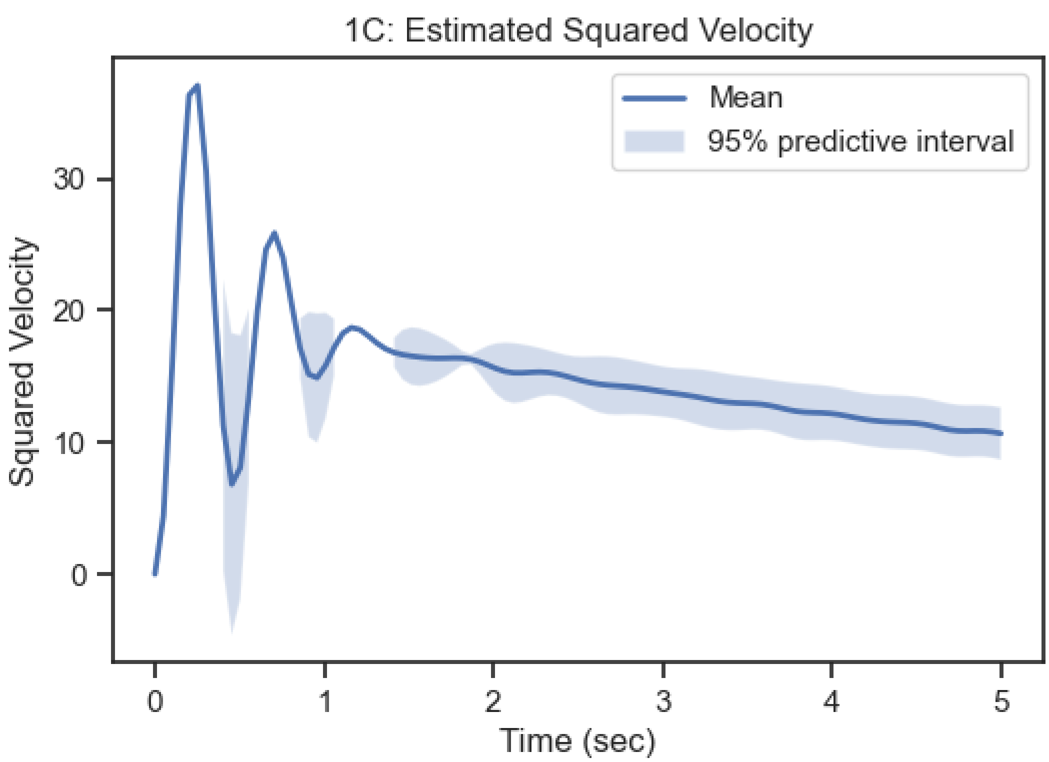
```python
# Running upper bound for the predictive interval
I_upper_running_postSQ = (I_running_postSQ + 2.0 * np.sqrt(sigma2_running_postSQ / np.a


#1b Position Plot at t=5
fig, ax = plt.subplots()

ax.fill_between(
    np.arange(1, NewN + 1),
    I_lower_running_postSQ,
    I_upper_running_postSQ,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(np.arange(1, NewN + 1),I_running_postSQ, 'b', lw=2, label = 'Mean')
ax.set_xlabel('Sample Number')
ax.set_ylabel('SQ Position');
plt.title('1C Average Position at Time = 5 seconds')
plt.legend()
```

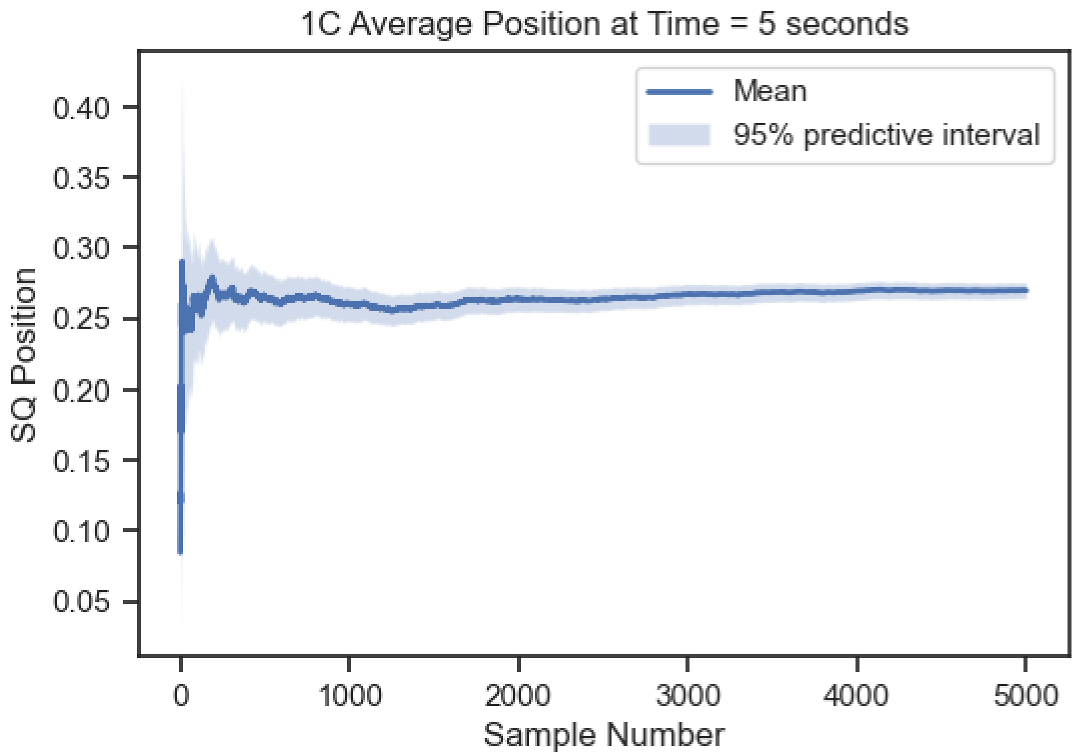
```
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3658096333.py:30: RuntimeWarning: invalid
value encountered in sqrt
  I_lower_running_posSQ = (I_running_posSQ - 2.0 * np.sqrt(sigma2_running_posSQ / np.ara
nge(1, N + 1)))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3658096333.py:31: RuntimeWarning: invalid
value encountered in sqrt
  I_lower_running_velSQ = (I_running_velSQ - 2.0 * np.sqrt(sigma2_running_velSQ / np.ara
nge(1, N + 1)))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3658096333.py:34: RuntimeWarning: invalid
value encountered in sqrt
  I_upper_running_posSQ = (I_running_posSQ + 2.0 * np.sqrt(sigma2_running_posSQ / np.ara
nge(1, N + 1)))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/3658096333.py:35: RuntimeWarning: invalid
value encountered in sqrt
  I_upper_running_velSQ = (I_running_velSQ + 2.0 * np.sqrt(sigma2_running_velSQ / np.ara
nge(1, N + 1)))
```

Out[105…

```
<matplotlib.legend.Legend at 0x1309db5cc40>
```


1C: Estimated Squared Position
Mean
95% predictive interval
Squared Position
Time (sec)
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0
1
2
3
4
5


1C: Estimated Squared Velocity
Mean
95% predictive interval
Squared Velocity
Time (sec)
30
20
10
0
0
1
2
3
4
5



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:

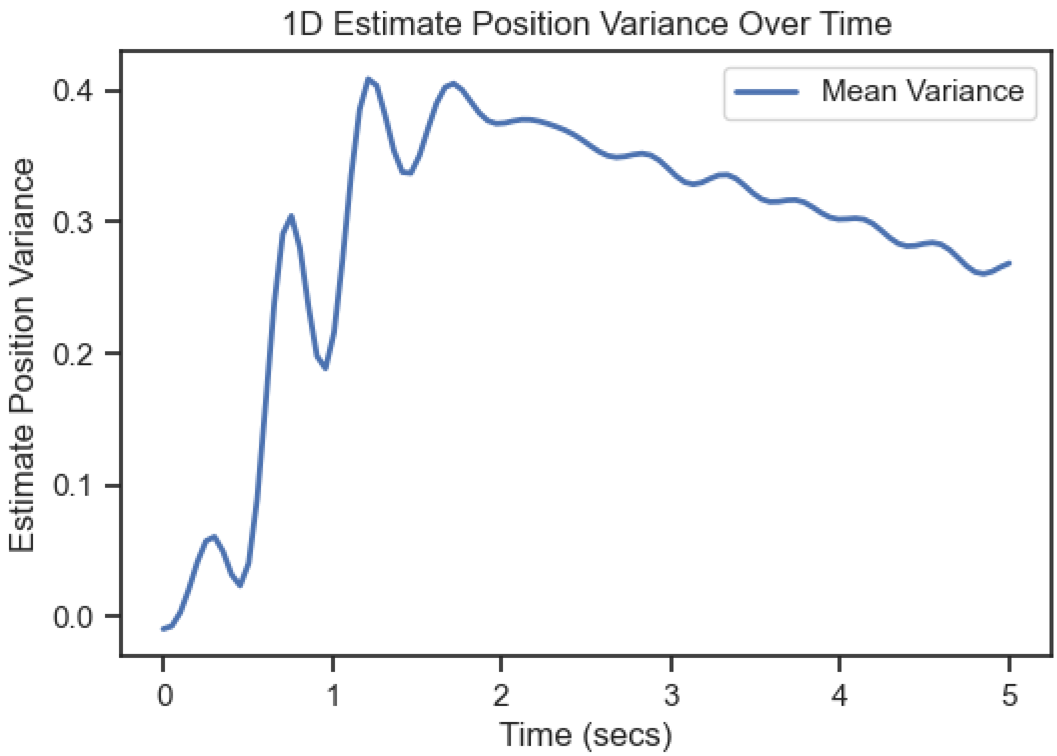
$$\mathbb{V}[y(t)] = \mathbb{E}[y^2(t)] - (\mathbb{E}[y(t)])^2,$$

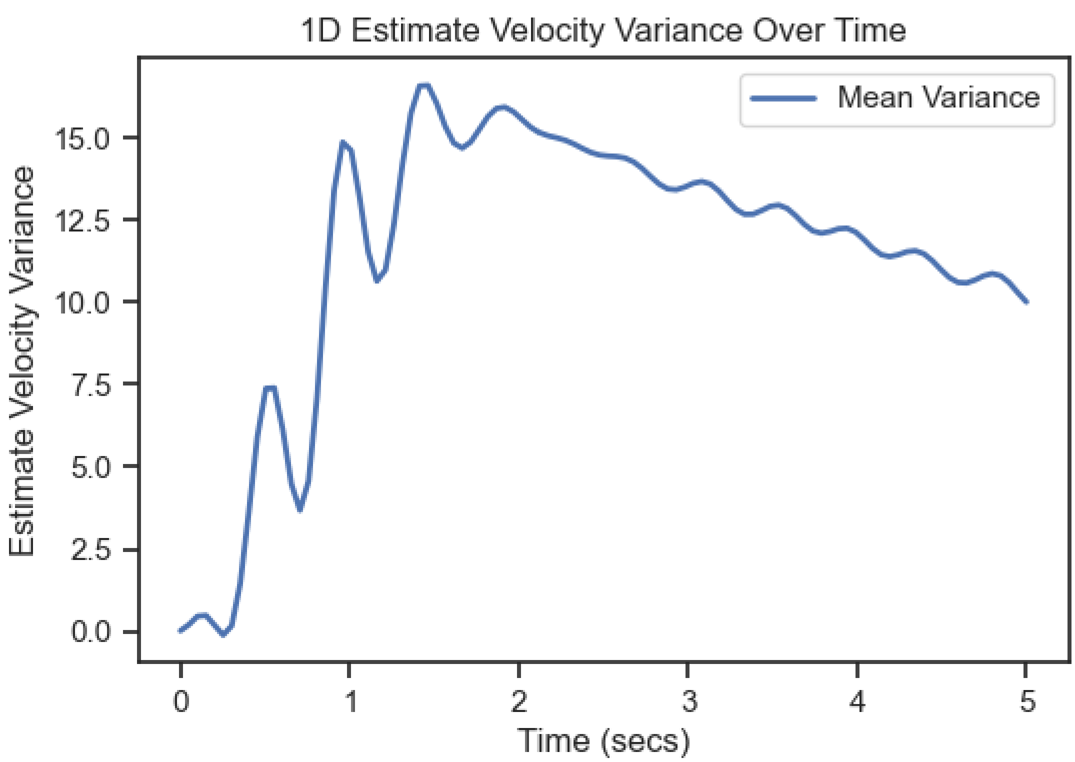
and similarly for $\dot{y}(t)$, to estimate the variance of the position and the velocity with negligible epistemic uncertainty. Plot both quantities as a function of time.

Solution:

In [106…

```
#1D

VarPos = I_running_posSQ - I_running_pos ** 2
VarVel = I_running_velSQ - I_running_vel ** 2

fig, ax = plt.subplots()
ax.plot(time,VarPos, 'b', lw=2, label = 'Mean Variance')
ax.set_xlabel('Time (secs)')
ax.set_ylabel('Estimate Position Variance');
plt.title('1D Estimate Position Variance Over Time')
plt.legend();

fig, ax = plt.subplots()
ax.plot(time,VarVel, 'b', lw=2, label = 'Mean Variance')
ax.set_xlabel('Time (secs)')
ax.set_ylabel('Estimate Velocity Variance');
plt.title('1D Estimate Velocity Variance Over Time')
plt.legend();
```





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 See Code Below:

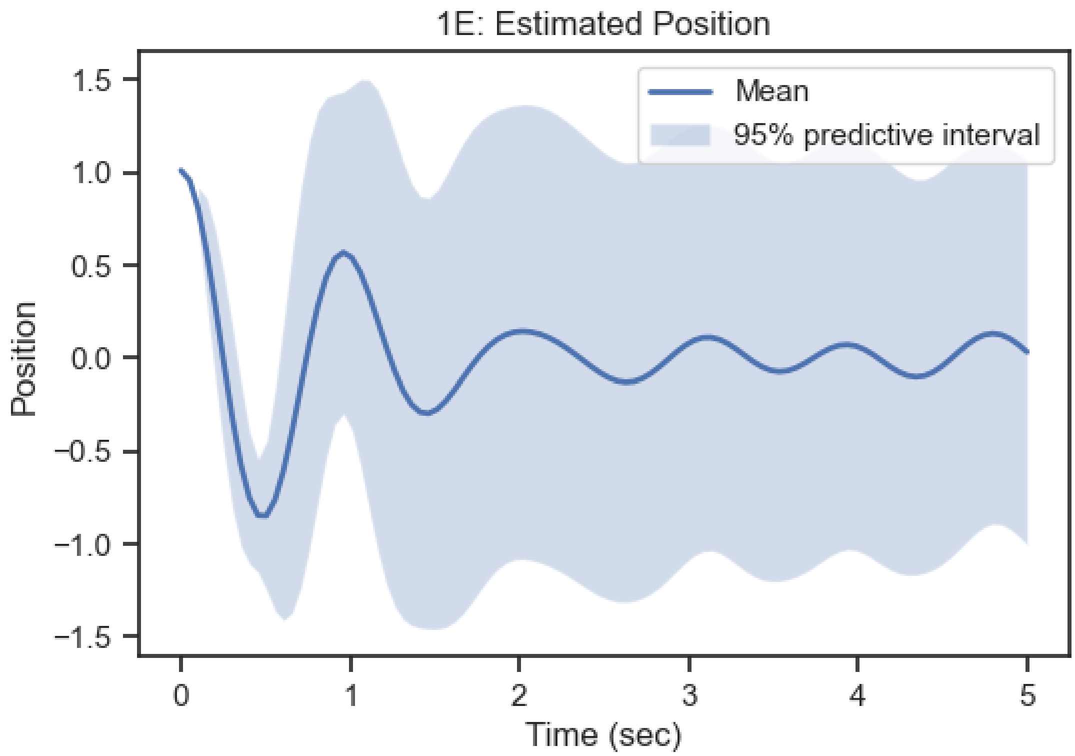
In [107…

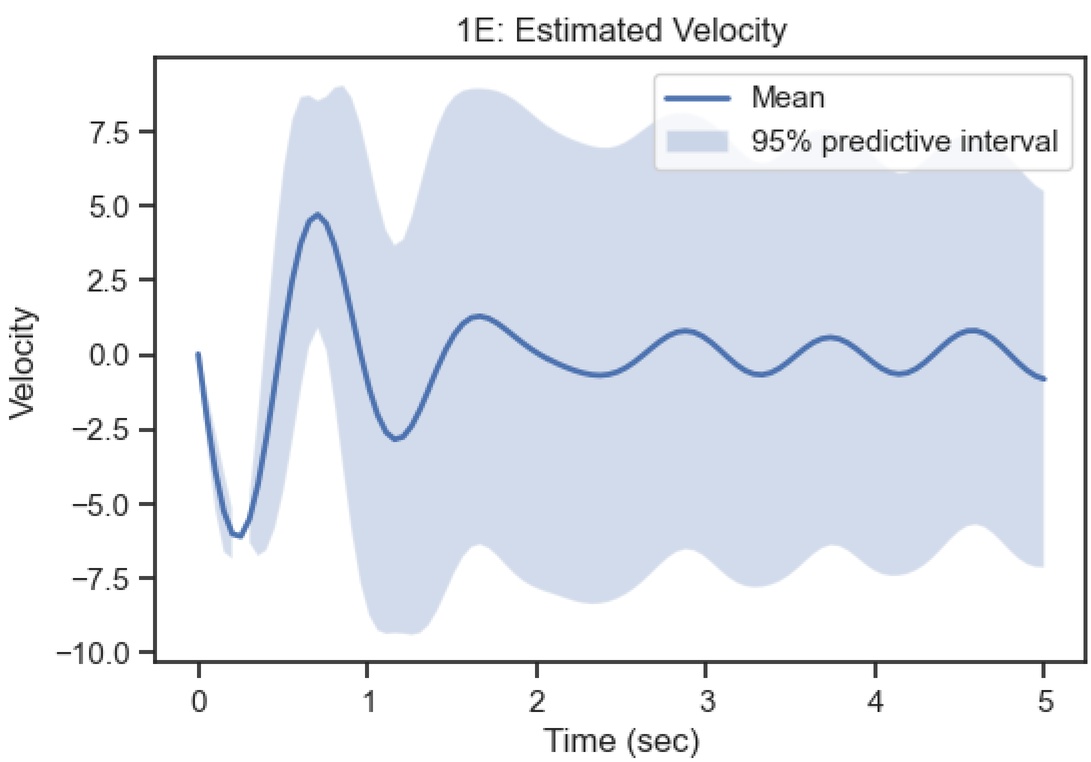
```
#Part 1E

I_lower_running_posE = (I_running_pos - 2.0 * np.sqrt(VarPos))
I_lower_running_velE = (I_running_vel - 2.0 * np.sqrt(VarVel))

I_upper_running_posE = (I_running_pos + 2.0 * np.sqrt(VarPos ))
I_upper_running_velE = (I_running_vel + 2.0 * np.sqrt(VarVel ))
#1E Position Plot
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    time,
    I_lower_running_posE,
    I_upper_running_posE,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(time, posMean, 'b', lw=2, label='Mean')
ax.set_xlabel('Time (sec)')
ax.set_ylabel('Position');
plt.title('1E: Estimated Position')
plt.legend()

#1A Velocity Plot
fig, ax = plt.subplots()
# Shaded area for the interval
ax.fill_between(
    time,
    I_lower_running_velE,
    I_upper_running_velE,
    alpha=0.25,
    label="95% predictive interval"
)
# Here is the MC estimate:
ax.plot(time, velMean, 'b', lw=2, label='Mean')
ax.set_xlabel('Time (sec)')
ax.set_ylabel('Velocity');
plt.title('1E: Estimated Velocity')
plt.legend()
```

```
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/1140402916.py:3: RuntimeWarning: invalid
value encountered in sqrt
  I_lower_running_posE = (I_running_pos - 2.0 * np.sqrt(VarPos))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/1140402916.py:4: RuntimeWarning: invalid
value encountered in sqrt
  I_lower_running_velE = (I_running_vel - 2.0 * np.sqrt(VarVel))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/1140402916.py:6: RuntimeWarning: invalid
value encountered in sqrt
  I_upper_running_posE = (I_running_pos + 2.0 * np.sqrt(VarPos ))
C:\Users\Ben\AppData\Local\Temp/ipykernel_2464/1140402916.py:7: RuntimeWarning: invalid
value encountered in sqrt
  I_upper_running_velE = (I_running_vel + 2.0 * np.sqrt(VarVel ))
```

Out[107…

```
<matplotlib.legend.Legend at 0x1309dc93280>
```





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 major earthquakes on Earth. We are going to focus on Southern California and we would like to assess the probability of a major earthquake, defined as an earthquake of magnitude 6.5 or greater, during the next ten years.

A. The first thing we are going to do is go over a database of past earthquakes that have occured in

Southern California and collect the relevant data. We are going to start at 1900 because data before that time may are unreliable. Go over each decade and count the occurence of a major earthquake (i.e., count the number of organge and red colors in each decade). We have done this for you.

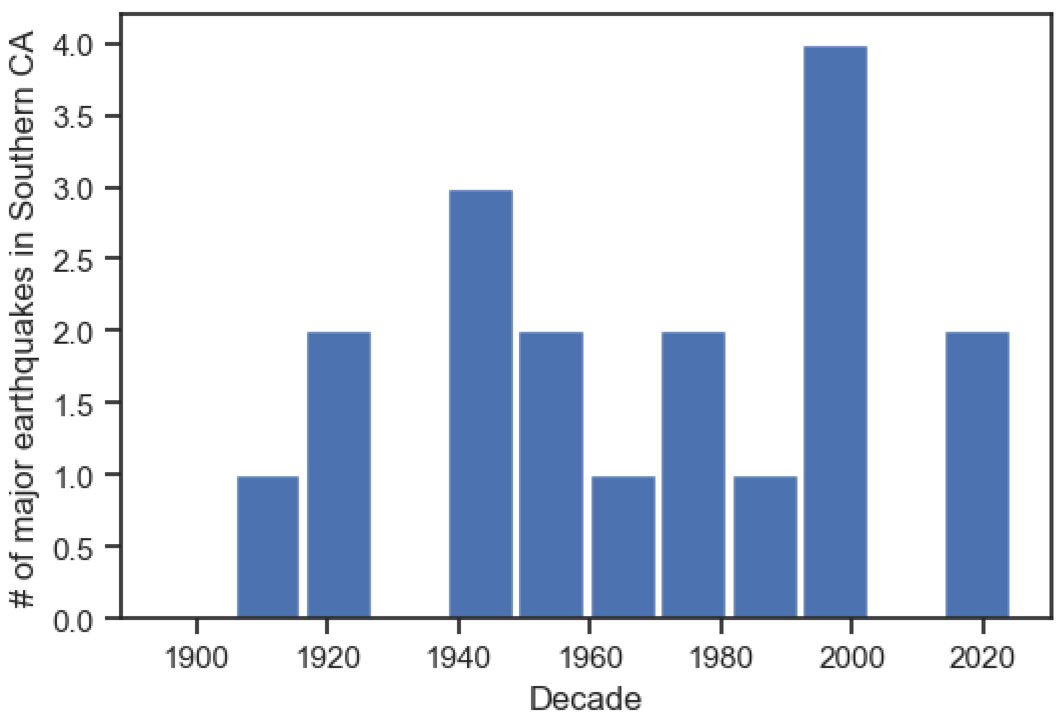
In [108…

```
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 [109…

```
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');
```



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 considerably.
Answer:

$$p(x_n|\lambda) = \frac{\lambda^{(0)}}{0!}e^{-\lambda} + \frac{\lambda^{(1)}}{1!}e^{-\lambda} + \frac{\lambda^{(2)}}{2!}e^{-\lambda} + \frac{\lambda^{(0)}}{0!}e^{-\lambda} + \frac{\lambda^{(3)}}{3!}e^{-\lambda} + \frac{\lambda^{(2)}}{2!}e^{-\lambda} + \frac{\lambda^{(1)}}{1!}e^{-\lambda} + \frac{\lambda^{(2)}}{2!}e^{-\lambda}$$

2A Answer:

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

B. The rate parameter λ (number of major 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 here 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 have to set to represent our prior state of knowledge. The PDF is:

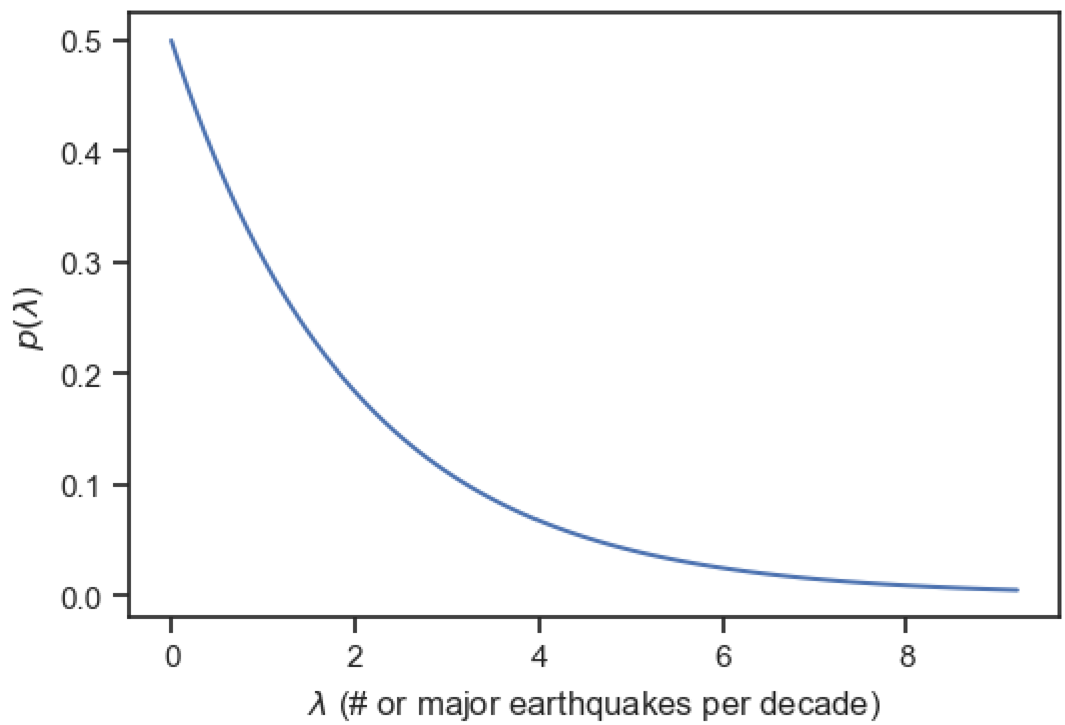
$$p(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1}e^{-\beta\lambda}}{\Gamma(\alpha)},$$

where we are not conditioning on α and β because they should be fixed numbers. Use the code below to pick some some 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 special case of the Gamma (set $\alpha = 1$).

In [110…

```
#2b
import scipy.stats as st

# You have to pick an alpha:
alpha = 1
# And you have to pick a beta:
beta = .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('$p(\lambda)$');
```



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:

$$p(\lambda|x_{1:N}) = p(x_{1:n}|\lambda) * p(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)} * \frac{\lambda^{(18)}}{2304} e^{-12\lambda} = \lambda^{\alpha-1} * e^{-\beta\lambda} * \frac{\lambda^{(18)}}{2304} e^{-12\lambda} = e^{-.5\lambda} * \cdot$$

$$p(\lambda|x_{1:N}) = \frac{\lambda^{(18)}}{2304} e^{-12.5\lambda}$$

D. Prior-likelihood pairs that result in a posterior with the same form as the prior as known as conjugate distributions. Conjugate distributions are your only hope for analytical Bayesian inference. As a sanity 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 sanity check.

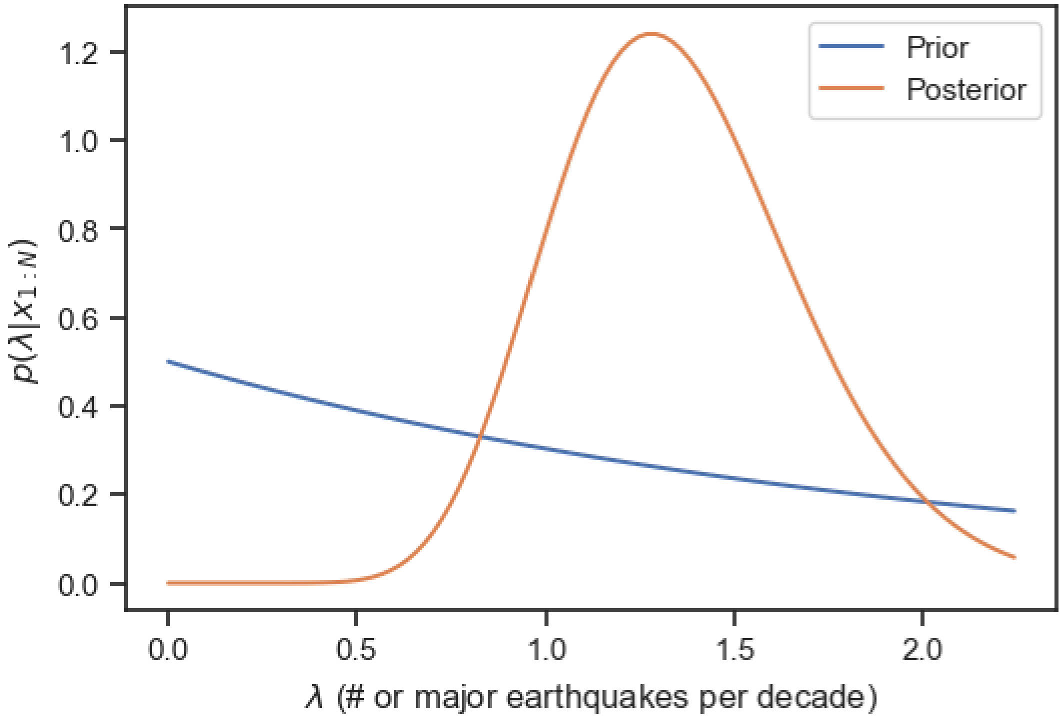
E. Plot the prior and the posterior of λ on the same plot.

In [111…

```
#Part E

N = 12
# Your expression for alpha posterior here:
alpha_post = 5 * alpha + N
# Your expression for beta posterior here:
beta_post = beta + N
# 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')
ax.plot(lambdas, lambda_post.pdf(lambdas), label = 'Posterior')
ax.set_xlabel('$\lambda$ (# or major earthquakes per decade)')
ax.set_ylabel('$p(\lambda|x_{1:N})$');
plt.legend()
```

Out[111…

```
<matplotlib.legend.Legend at 0x1309fc89b20>
```



F. Let's work out the predictive distribution for the number of major earthquakes during the next decade. This is something that we did not do in class, but it will appear again and again in future lectures. Let X be the random variable corresponding to the number of major eathquakes 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 just 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 used the assumption that the number of earthquakes occuring in each decade is independent. You can carray out this integration analytically (it gives a negative Binomial distribution) but we are not going to bother with it.

Below you are going to write code to characterize it using Monte Carlo sampling. Basically, 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 [112…

```
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()# WRITE ME (SAMPLE FROM POSTERIOR)
        samples[i] = st.poisson(lambda_sample).rvs()# WRITE ME (SAMPLE FROM POISSON GIV
    return samples
```

Test your code here:

In [113…

```
samples = sample_posterior_predictive(10, lambda_post)
samples
```

Out[113…

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

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

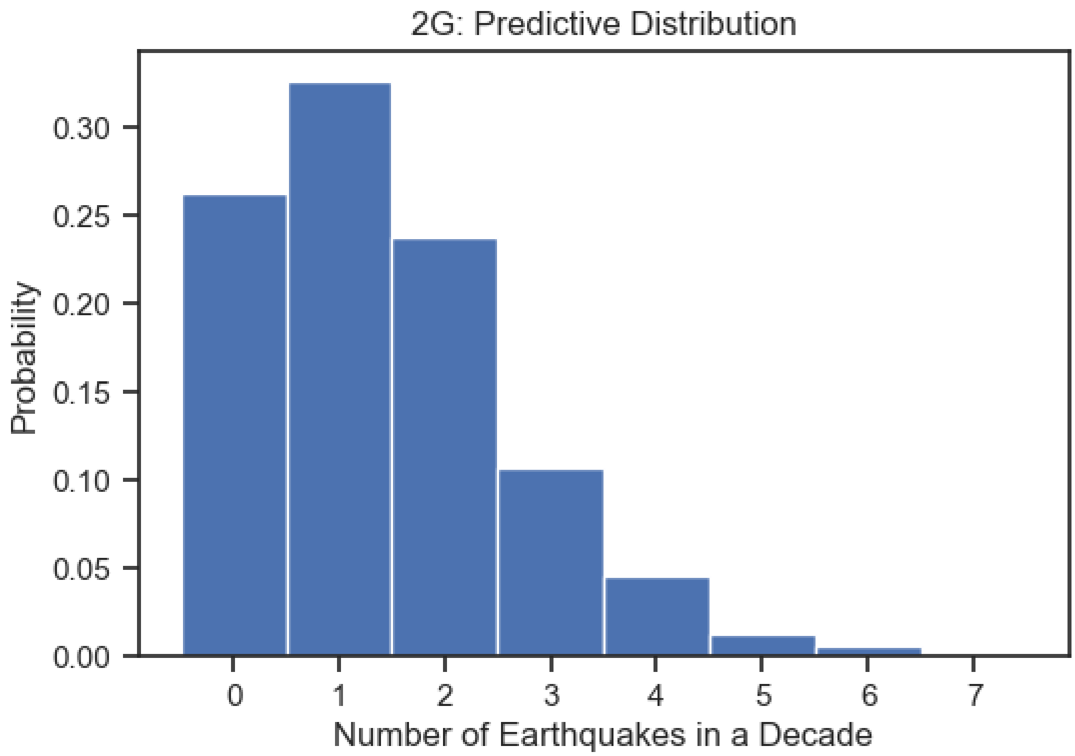
Hint: Draw 1,000 samples using `sample_posterior_predictive` *and then draw a histogram.*

In [114…

```
#partG

n= 1000 #number of samples
samplesg = sample_posterior_predictive(n, lambda_post)


plt.figure()
plt.hist(samplesg, bins = [0,1,2,3,4,5,6,7,8], align='left', density = True)
plt.xlabel('Number of Earthquakes in a Decade')
plt.ylabel('Probability')
plt.title('2G: Predictive Distribution')
```

Out[114…

```
Text(0.5, 1.0, '2G: Predictive Distribution')
```



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 [115…

```
#part H
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

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

```
p(X >= 1 | data) = 0.7303
```

I. Find a 95\% credible interval for λ.

In [116…

```
#part I
theta_low = lambda_post.ppf(0.025)
theta_up = lambda_post.ppf(0.975)
thetas = np.linspace(0, 1, num_samples)
print(f'Theta is in [{theta_low:.2f}, {theta_up:1.2f}] with 95% probability')
```

```
Theta is in [0.79, 2.08] with 95% probability
```

J. Find the λ that minimizes the absolute loss (see lecture), 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 [118…

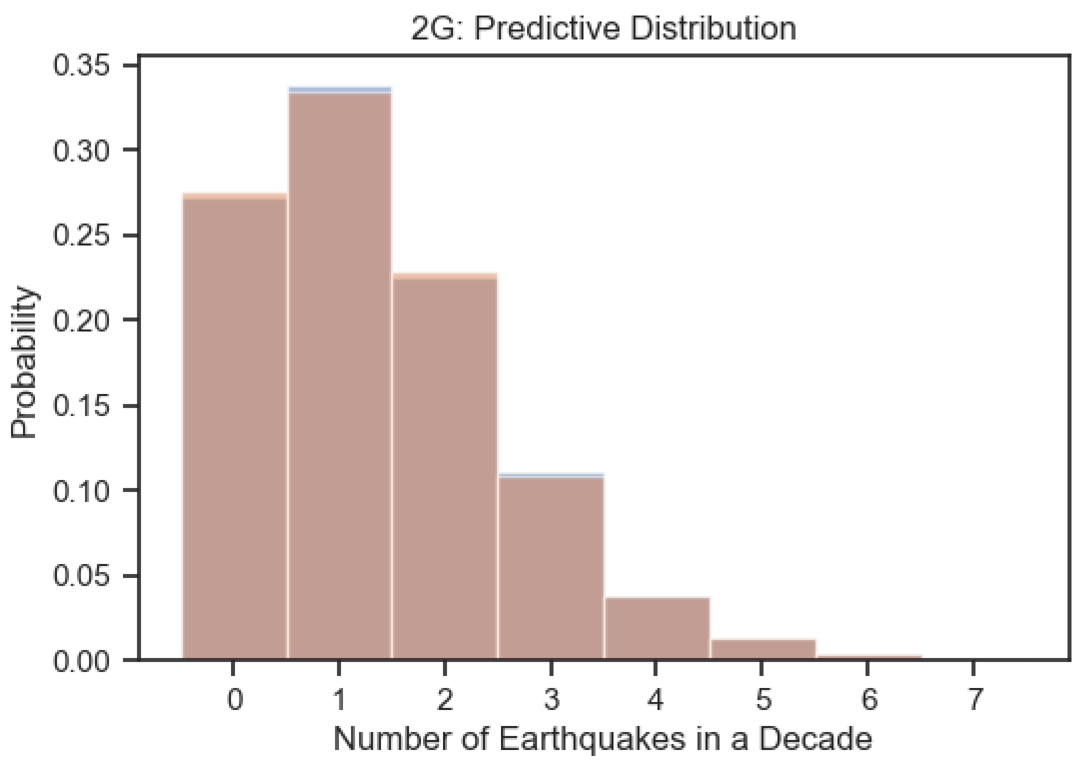
```
# part J
```

```
num_samples = 10000

theta_star_1 = lambda_post.median()
print(f'theta_star_1 = {theta_star_1:.2f}')
theta_predict = st.gamma(alpha_post, scale=1.0 / beta_post)

absolutesamples = sample_posterior_predictive(num_samples, theta_predict)
samples = sample_posterior_predictive(num_samples, lambda_post)

plt.figure()
plt.hist(samples, bins = [0,1,2,3,4,5,6,7,8], align='left', density = True, alpha = 0.5
plt.hist(absolutesamples , bins = [0,1,2,3,4,5,6,7,8], align='left', density = True, al
plt.xlabel('Number of Earthquakes in a Decade')
plt.ylabel('Probability')
plt.title('2G: Predictive Distribution')
```

```
theta_star_1 = 1.33
```

Out[118…

```
Text(0.5, 1.0, '2G: Predictive Distribution')
```



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 [119…

```
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):
        theta_post_sample = post_rv.rvs()
        x_rep[i, :] = st.poisson(theta_post_sample).rvs(size=n)
    return x_rep
```

Try your code here:

In [120…

```
x_rep = replicate_experiment(lambda_post)
x_rep
n_rep=9
```

If it works, then try the following visualization:

In [121…

```
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'
)
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 consecutive decades with earthquakes.

Hint: You may reuse the code from the textbook.

In [122…

```
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)
```

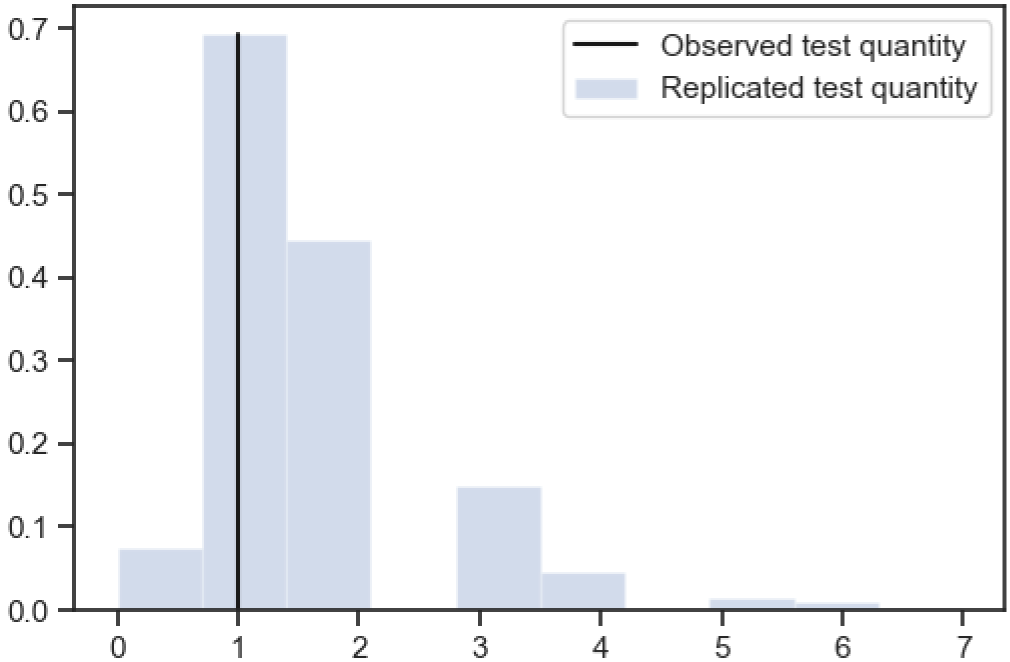
In [123…

```
# 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
```

In [125…

```
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

Maxconsec = T_eq_max_eq(samples)
MaxNoEq = T_eq_max_neq(samples)

diagnostics = do_diagnostics(lambda_post, thetas, T_eq_max_neq)

print(f'The maximum number of consecutive decades with no earthquakes is {MaxNoEq:.0f}'
print(f'The maximum number of consecutive decades with earthquakes is {Maxconsec:.0f}')
```

```
The observed test quantity is 1
The Bayesian p_value is 0.4630
The maximum number of consecutive decades with no earthquakes is 8
The maximum number of consecutive decades with earthquakes is 34
```


Observed test quantity
Replicated test quantity
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
2
3
4
5
6
7