Bayesian Statistics

Contents

3.1. Frequentism and Bayesianism

This section follows the MacKay 3.1 and the notes from Jake VanderPlas

Here we discuss the fundamentals of Bayesian statistics and what differs it from frequentist statistics. So, the most important question is why do we need to learn 'Bayes' principle?

The Disagreement: The Definition of Probability

Fundamentally, the disagreement between frequentists and Bayesians concerns the definition of probability.

For frequentists, probability only has meaning in terms of a *limiting case of repeated measurements*. If a hydrologist measures the flow rate of a river at a particular location, then measures it again, and repeats the process, each measurement will vary slightly due to statistical error of the measuring device. In the limit of many measurements, the frequency of any given value indicates the probability of measuring that value. For frequentists, *probabilities are fundamentally related to frequencies of events*. This means, for example, that in a strict frequentist view, it is meaningless to talk about the probability of the true flow rate: the true flow rate, by definition, a single fixed value, and to talk about an extended frequency distribution for a fixed value is nonsense.

For Bayesians, the concept of probability is extended to cover *degrees of certainty about statements*. A Bayesian might claim to know the flow rate of a river with some probability P(F): that probability can certainly be estimated from frequencies in the limit of a large number of repeated experiments, but this is not fundamental. The probability is a statement of the researcher's knowledge of what the flow rate is. For Bayesians, *probabilities are fundamentally related to their own knowledge about an event*. This means, for example, that in a Bayesian view, we can meaningfully talk about the probability that the true flow rate lies in a given range. That probability codifies our knowledge of the value based on prior information and available data.

Let’s see the following example to better understand the above text.

Let’s imagine a hydrologist using a device measures the flow rate at the downstream of a dam. Let’s assume that the true flow rate is constant with time, i.e., that is it has a fixed value F. We will assume that a series of N measurements are performed, where the ith measurement reports the observed flow rate F_i and error e_i. The question is, given this set of measurements $D = \{F_i, e_i\}$, what is our best estimate of the true flow rate F?

Let’s draw 50 samples F_i with a mean of 1000 (in arbitrary units) and a (known) error e_i:

```
np.random.seed(2) # for reproducibility
e = np.random.normal(30, 3, 50)
F = np.random.normal(1000, e)
```

```
%matplotlib inline
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.errorbar(F, np.arange(50), xerr=e, fmt='ok', ecolor='gray', alpha=0.5)
ax.vlines([1000], 0, 50, linewidth=5, alpha=0.2)
ax.set_xlabel("Flow rate");ax.set_ylabel("measurement number");
plt.savefig('/content/drive/MyDrive/PSU/ColabNotebooks/Example3_1.png',
dpi=300)
```

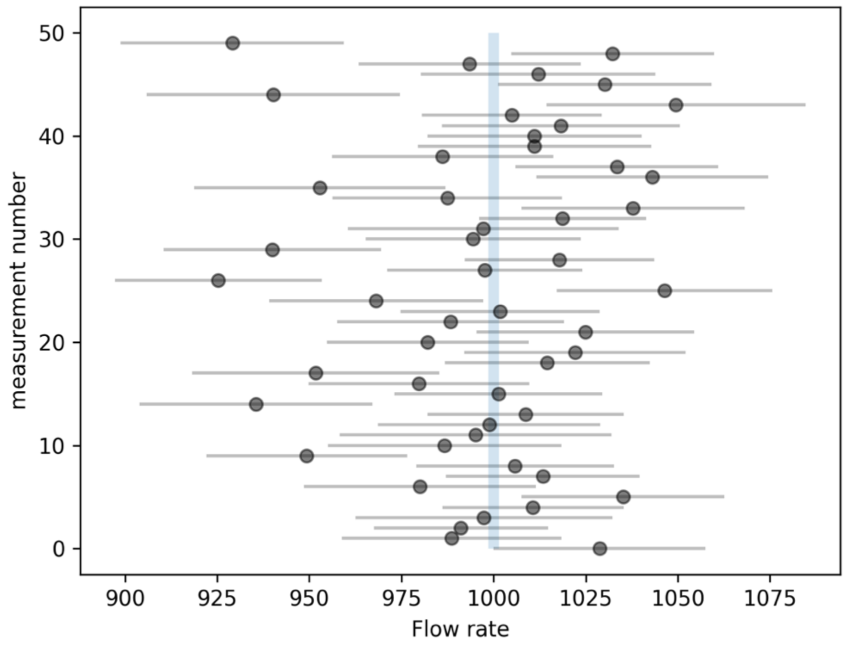


Figure 3.1

In this toy example we already know the true F, but the question is this: given our measurements and errors, what is our best point estimate of the true flow rate? Let's look at a frequentist and a Bayesian approach to solving this.

Frequentist Approach

We will start with the classical frequentist maximum likelihood approach. Given a single observation $D_i = \{F_i, e_i\}$, we can compute the probability distribution of the measurement given the true F with the assumption of Gaussian errors:

$$P(D_i|F) = (2\pi e_i^2)^{-1/2} \exp\left(\frac{-(F_i - F)^2}{2e_i^2}\right)$$

You should recognize this as a normal distribution with mean F and standard deviation e_i. We construct the likelihood by computing the product of the probabilities for each data point:

$$\mathcal{L}(D|F) = \prod_{i=1}^{N} P(D_i|F)$$

Here $D \ = \ \{D_i\}$ represents the entire set of measurements. For reasons of both analytic simplicity and numerical accuracy, it is often more convenient to instead consider the log-likelihood; combining the previous two equations gives:

$$\text{Log}\,\mathcal{L}(D|F) = -\frac{1}{2}\sum_{i=1}^{N}\left[\text{Log}\,(2\pi e_i^2) + \frac{-(F_i - F)^2}{e_i^2}\right]$$

We would like to determine the value of F which maximizes the likelihood. For this simple problem, the maximization can be computed analytically (e.g. by setting $d\ [\text{Log}\,\mathcal{L}(D|F)]/dF|_{\hat{F}} = 0$), which results in the following point estimate of F:

$$\hat{F} = \frac{\sum w_i F_i}{\sum w_i}, \qquad w_i = {}^{1}\!/_{e_i^2}$$

The result is a simple weighted mean of the observed values. Notice that in the case of equal errors e_i, the weights cancel and $\hat{F}$ is simply the mean of the observed data.

We can go further and ask what the uncertainty of our estimate is. One way this can be accomplished in the frequentist approach is to construct a Gaussian approximation to the peak likelihood; in this simple case the fit can be solved analytically to give:

$$\sigma_{\hat{F}} = \left(\sum_{i=1}^{N} w_i\right)^{-1/2}$$

```
w = 1. / e ** 2
F_hat = np.sum(w * F) / np.sum(w)
sigma_F = w.sum() ** -0.5
print(F_hat,sigma_F)
```

```
998.6496963757094 4.113743971231106
```

For our data, the result is $\hat{F}$ = 998 ± 4.

Bayesian Approach

The Bayesian approach, as you might expect, begins and ends with probabilities. The fundamental result of interest is our knowledge of the parameters in question, codified by the probability $P(F|D)$. To compute this result, we next apply Bayes' theorem, a fundamental law of probability:

$$P(F|D) = \frac{P(D|F)P(F)}{P(D)}$$

Though Bayes' theorem is where Bayesians get their name, it is important to note that it is not this theorem itself that is controversial, but the Bayesian interpretation of probability implied by the term $P(F|D)$. While the above formulation makes sense given the Bayesian view of probability, the setup is fundamentally contrary to the frequentist philosophy, which says that probabilities have no meaning for fixed model parameters like F.

$P(F|D)$: The *posterior*, which is the probability of the model parameters given the data.
$P(D|F)$: The *likelihood*, which is proportional to the $\mathcal{L}(D|F)$ used in the frequentist approach.
$P(F)$: The *model prior*, which is what we knew about the model before considering the data D.
$P(D)$: The *model evidence*, which in practice amounts to simply a normalization term.

The interesting point is we set the prior $P(F) \propto 1$ (a *flat prior*), the Bayesian posterior is maximized at precisely the same value as the frequentist result! So, despite the philosophical differences, we see that the Bayesian and frequentist point estimates are equivalent for this simple problem.

Please note that a *flat prior* is considered an uninformative prior, typically used when we lack any prior knowledge about the event. In such cases, the Bayesian formalism often leads to results similar to those of the frequentist formulation. Frequentism can often be viewed as simply a special case of the Bayesian approach for some (implicit) choice of the prior.

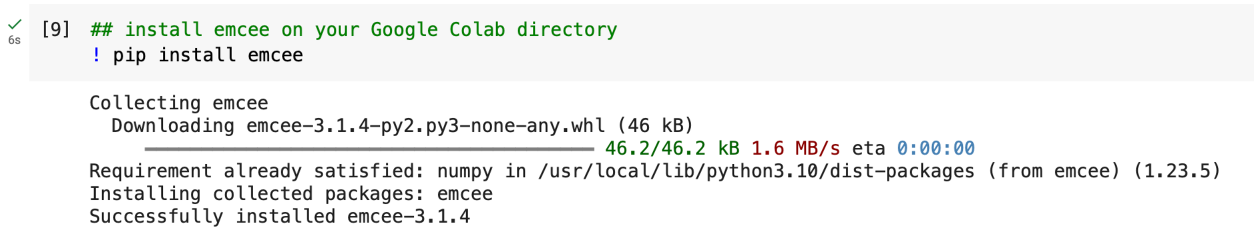
Let's see how Bayesian results are generally computed in practice. For a one parameter problem like the one considered here, it's as simple as computing the posterior probability $P(F|D)$ as a function of F: this is the distribution reflecting our knowledge of the parameter F. But as the dimension of the model grows, this direct approach becomes increasingly intractable.

For this reason, Bayesian calculations often depend on sampling methods such as Markov Chain Monte Carlo (MCMC). I won't go into the details of the theory of MCMC here.

Keep in mind here that the goal is to *generate a set of points drawn from the posterior probability distribution*, and to use those points to determine the answer we seek.

To perform this MCMC, we start by defining Python functions for the prior $P(F)$, the likelihood $P(D|F)$, and the posterior $P(F|D)$, noting that none of these need be properly normalized. Our model here is one-dimensional, but to handle multi-dimensional models we'll define the model in terms of an array of parameters θ, which in this case is $\theta = [F]$:

```
def log_prior(theta):
    return 1  # flat prior

def log_likelihood(theta, F, e):
    return -0.5 * np.sum(np.log(2 * np.pi * e ** 2)
                         + (F - theta[0]) ** 2 / e ** 2)

def log_posterior(theta, F, e):
    return log_prior(theta) + log_likelihood(theta, F, e)
```



```
## install emcee on your Google Colab directory
! pip install emcee
```

```
Collecting emcee
  Downloading emcee-3.1.4-py2.py3-none-any.whl (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.2/46.2 kB 1.6 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from emcee) (1.23.5)
Installing collected packages: emcee
Successfully installed emcee-3.1.4
```

Now we set up the problem, including generating some random starting guesses for the multiple chains of points.

```
ndim = 1  # number of parameters in the model
nwalkers = 50  # number of MCMC walkers
nburn = 1000  # "burn-in" period to let chains stabilize
nsteps = 2000  # number of MCMC steps to take
```

```
# we'll start at random locations between 0 and 2000
starting_guesses = 2000 * np.random.rand(nwalkers, ndim)

import emcee
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[F,
e])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].ravel()  # discard burn-in points
```

If this all worked correctly, the array *sample* should contain a series of 50000 points drawn from the posterior. Let's plot them and check:

```
# plot a histogram of the sample
from scipy import stats
plt.hist(sample, bins=50, histtype="stepfilled", alpha=0.3)

# plot a best-fit Gaussian
F_fit = np.linspace(975, 1025)
pdf = stats.norm(np.mean(sample), np.std(sample)).pdf(F_fit)

plt.plot(F_fit, pdf, '-k')
plt.xlabel("F"); plt.ylabel("P(F)")
plt.savefig('/content/drive/MyDrive/PSU/ColabNotebooks/Example3_2.png',
dpi=300)
```

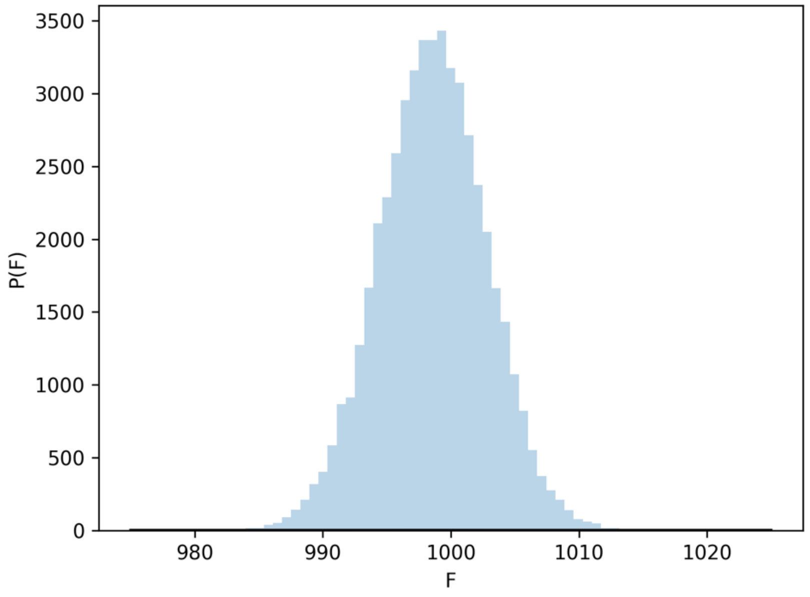


Figure 3.2

We end up with a sample of points drawn from the (normal) posterior distribution. The mean and standard deviation of this posterior are the corollary of the frequentist maximum likelihood estimate above:

```
print("""
      F_true = {0}
      F_est  = {1:.0f} +/- {2:.0f} (based on {3} measurements)
      """.format(1000, np.mean(sample), np.std(sample), 50))
```

```
F_true = 1000
F_est  = 999 +/- 4 (based on 50 measurements)
```

We see that as expected for this simple problem, the Bayesian approach yields the same result as the frequentist approach!

Now, you may think that the Bayesian method is unnecessarily complicated, and in this case it certainly is. Using a Markov Chain Monte Carlo Ensemble sampler to characterize a one-dimensional normal distribution is a bit like using the Death Star to destroy a beach ball, but I did this here because it demonstrates an approach that can scale to complicated posteriors in many, many dimensions, and can provide nice results in more complicated situations where an analytic likelihood approach is not possible.

As a side note, you might also have noticed one little sleight of hand: at the end, we use a frequentist approach to characterize our posterior samples! When we computed the sample mean and standard deviation above, we were employing a distinctly frequentist technique to characterize the posterior distribution. The pure Bayesian result for a problem like this would be to report the posterior distribution itself (i.e. its representative sample) and leave it at that. That is, in pure Bayesianism the answer to a question is not a single number with error bars; the answer is the posterior distribution over the model parameters!

3.2. Exploring a more sophisticated model

Let's briefly take a look at a more complicated situation and compare the frequentist and Bayesian results yet again. Above we assumed that the flow rate was static: now let's assume that we're looking at an object which we suspect has some stochastic variation — that is, it varies with time, but in an unpredictable way.

We'll propose a simple 2-parameter Gaussian model for this object: $\theta = [\mu, \sigma]$ where μ is the mean value, and σ is the standard deviation of the variability. Thus, our model for the probability of the true flow rate at the time of each observation looks like this:

$$F \sim \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[\frac{-(F-\mu)^2}{2\sigma^2}\right]$$

Now, we'll again consider N observations each with their own error. We can generate them this way:

```
np.random.seed(42)  # for reproducibility
N = 100  # we'll use more samples for the more complicated model
mu_true, sigma_true = 1000, 15  # stochastic flow rate model

F_true = stats.norm(mu_true, sigma_true).rvs(N)  # (unknown) true flow
rate
F = stats.poisson(F_true).rvs()  # observed flow rate: true flow rate plus
Poisson errors.
e = np.sqrt(F)  # root-N error, as above
```

Frequentist Approach

The resulting likelihood is the convolution of the intrinsic distribution with the error distribution, so we have:

$$\mathcal{L}(D|\theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi(\sigma^2 + e_i^2)}} exp\left[\frac{-(F_i-\mu)^2}{2(\sigma^2 + e_i^2)}\right]$$

Analogously to above, we can analytically maximize this likelihood to find the best estimate for μ:

$$\mu_{est} = \frac{\sum w_i F_i}{\sum w_i}; \qquad w_i = \frac{1}{(\sigma^2 + e_i^2)}$$

And here we have a problem: the optimal value of μ *depends on* the optimal value of σ. The results are correlated, so we can no longer use straightforward analytic methods to arrive at the frequentist result.

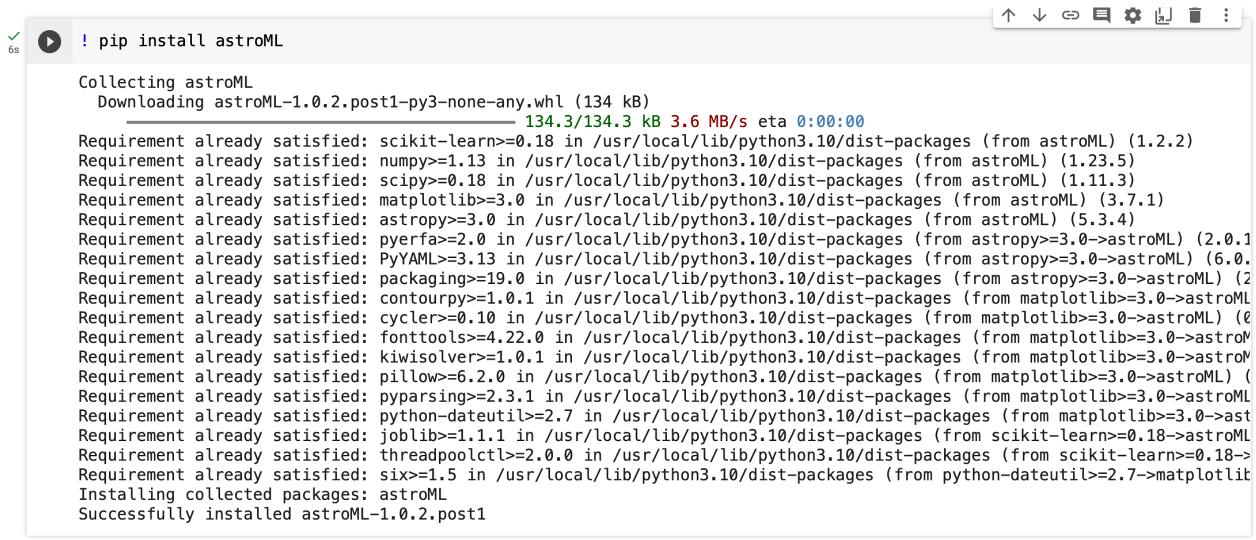
Nevertheless, we can use numerical optimization techniques to determine the maximum likelihood value. Here we'll use the optimization routines available within Scipy's optimize submodule:

```
def log_likelihood(theta, F, e):
    return -0.5 * np.sum(np.log(2 * np.pi * (theta[1] ** 2 + e ** 2))
                         + (F - theta[0]) ** 2 / (theta[1] ** 2 + e ** 2))

# maximize likelihood <--> minimize negative likelihood
def neg_log_likelihood(theta, F, e):
    return -log_likelihood(theta, F, e)

from scipy import optimize
theta_guess = [900, 5]
theta_est = optimize.fmin(neg_log_likelihood, theta_guess, args=(F, e))
print("""
      Maximum likelihood estimate for {0} data points:
          mu={theta[0]:.0f}, sigma={theta[1]:.0f}
      """.format(N, theta=theta_est))
```

```
Optimization terminated successfully.
         Current function value: 502.839505
         Iterations: 58
         Function evaluations: 114

      Maximum likelihood estimate for 100 data points:
          mu=999, sigma=19
```

This maximum likelihood value gives our best estimate of the parameters μ and σ governing our model of the source. But this is only half the answer: we need to determine how confident we are in this answer, that is, we need to compute the error bars on μ and σ.

There are several approaches to determining errors in a frequentist paradigm. We could, as above, fit a normal approximation to the maximum likelihood and report the covariance matrix (here we'd have to do this numerically rather than analytically). Alternatively, we can compute statistics like χ^2 and χ^2_{dof} to and use standard tests to determine confidence limits, which also depends on strong assumptions about the Gaussianity of the likelihood. We might alternatively use randomized sampling approaches such as Jackknife or Bootstrap, which maximize the likelihood for randomized samples of the input data in order to explore the degree of certainty in the result. All of these would be valid techniques to use, but each comes with its own assumptions and subtleties. Here, for simplicity, we'll use the basic bootstrap resampler found in the astroML package:



```
! pip install astroML
```

```
Collecting astroML
  Downloading astroML-1.0.2.post1-py3-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.3/134.3 kB 3.6 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from astroML) (1.2.2)
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.10/dist-packages (from astroML) (1.23.5)
Requirement already satisfied: scipy>=0.18 in /usr/local/lib/python3.10/dist-packages (from astroML) (1.11.3)
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.10/dist-packages (from astroML) (3.7.1)
Requirement already satisfied: astropy>=3.0 in /usr/local/lib/python3.10/dist-packages (from astroML) (5.3.4)
Requirement already satisfied: pyerfa>=2.0 in /usr/local/lib/python3.10/dist-packages (from astropy>=3.0->astroML) (2.0.1
Requirement already satisfied: PyYAML>=3.13 in /usr/local/lib/python3.10/dist-packages (from astropy>=3.0->astroML) (6.0.
Requirement already satisfied: packaging>=19.0 in /usr/local/lib/python3.10/dist-packages (from astropy>=3.0->astroML) (2
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->astroML
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->astroML) (0
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->astroM
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->astroM
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->astroML) (
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->astroML
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->ast
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->astroML
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib
Installing collected packages: astroML
Successfully installed astroML-1.0.2.post1
```

```python
from astroML.resample import bootstrap

def fit_samples(sample):
    # sample is an array of size [n_bootstraps, n_samples]
    # compute the maximum likelihood for each bootstrap.
    return np.array([optimize.fmin(neg_log_likelihood, theta_guess,
                                   args=(F, np.sqrt(F)), disp=0)
                     for F in sample])

samples = bootstrap(F, 1000, fit_samples)  # 1000 bootstrap resamplings
```

Now in a similar manner to what we did above for the MCMC Bayesian posterior, we'll compute the sample mean and standard deviation to determine the errors on the parameters.

```python
mu_samp = samples[:, 0]
sig_samp = abs(samples[:, 1])

print (" mu    = {0:.0f} +/- {1:.0f}".format(mu_samp.mean(),
mu_samp.std()))
print (" sigma = {0:.0f} +/- {1:.0f}".format(sig_samp.mean(),
sig_samp.std()))
```

```
mu    = 999 +/- 4
sigma = 18 +/- 5
```

I should note that there is a huge literature on the details of bootstrap resampling, and there are definitely some subtleties of the approach that I am glossing over here. One obvious piece is that there is potential for errors to be correlated or non-Gaussian, neither of which is reflected by simply finding the mean and standard deviation of each model parameter. Nevertheless, I trust that this gives the basic idea of the frequentist approach to this problem.

Bayesian Approach

The Bayesian approach to this problem is almost exactly the same as it was in the previous problem, and we can set it up by slightly modifying the above code.

```python
def log_prior(theta):
    # sigma needs to be positive.
    if theta[1] <= 0:
        return -np.inf
    else:
        return 0

def log_posterior(theta, F, e):
    return log_prior(theta) + log_likelihood(theta, F, e)

# same setup as above:
ndim, nwalkers = 2, 50
nsteps, nburn = 2000, 1000

starting_guesses = np.random.rand(nwalkers, ndim)
starting_guesses[:, 0] *= 2000  # start mu between 0 and 2000
starting_guesses[:, 1] *= 20    # start sigma between 0 and 20

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[F,
e])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, 2)
```

Now that we have the samples, we'll use a convenience routine from astroML to plot the traces and the contours representing one and two standard deviations:

```python
from astroML.plotting import plot_mcmc
fig = plt.figure()
ax = plot_mcmc(sample.T, fig=fig, labels=[r'$\mu$', r'$\sigma$'],
colors='k')
ax[0].plot(sample[:, 0], sample[:, 1], ',k', alpha=0.8)
```

```
ax[0].plot([mu_true], [sigma_true], 'o', color='red', ms=10);
plt.savefig('/content/drive/MyDrive/PSU/ColabNotebooks/Example3_3.png',
dpi=300)
```

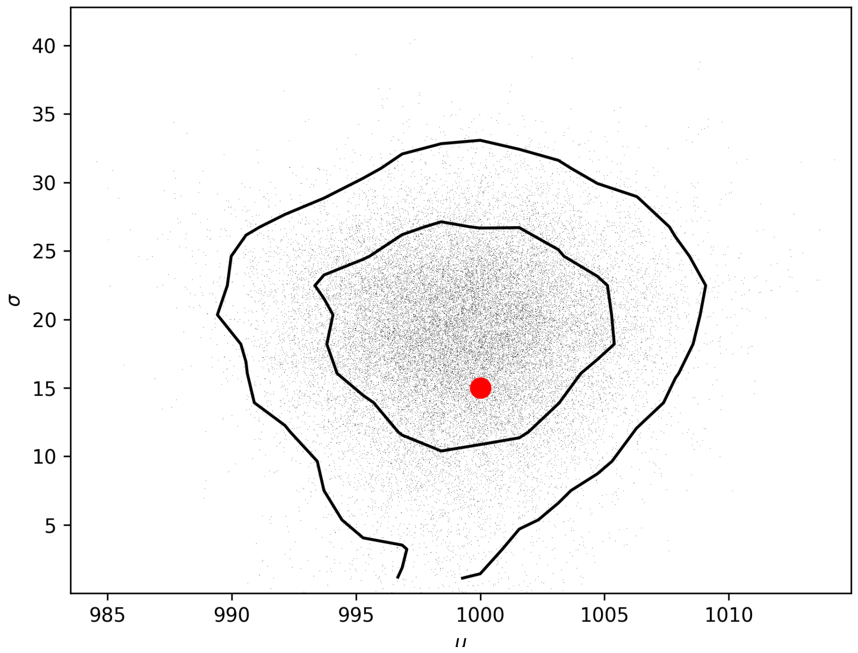


Figure 3.3

The red point serves as an indicator of ground truth, as per our problem setup. Meanwhile, the contours depict one and two standard deviations, corresponding to 68% and 95% confidence levels. In simpler terms, this analysis lends us 68% confidence that the model resides within the inner contour and 95% confidence that it falls within the outer contour.

It is worth noting that when σ=0, our data remains consistent within two standard deviations. In essence, depending on the level of certainty you seek, our data lacks the strength to definitively dismiss the possibility of a constant source.

Another point of interest is the unmistakable non-Gaussian nature of this posterior distribution, as evidenced by the absence of vertical symmetry. This implies that the Gaussian approximation commonly employed in the frequentist approach may not faithfully reflect the genuine uncertainties in the results. It's essential to recognize that this isn't a flaw in frequentism itself; there are indeed methods to accommodate non-Gaussianity within the frequentist framework. However, it's crucial to acknowledge that most widely used frequentist techniques explicitly or implicitly assume a Gaussian distribution. In contrast, Bayesian approaches generally do not necessitate such assumptions.

3.3. Non-Gaussian Likelihood Function

Likelihood: As we saw in the previous chapter, when you're flipping coins, and you want to know the probability of getting k heads out of n tosses, with each toss having a probability θ, you can use the Binomial distribution.

$$P(k \mid n, \theta) = \text{Binomial}(k \mid n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

Prior: We need a prior that covers the range from 0 to 1. We mentioned earlier that the Beta distribution is a good fit for this, and it's especially great for the Binomial because it has this shape:

$$\text{Beta}(\theta \mid a, b) \propto \theta^{a-1}(1-\theta)^{b-1}$$

So, it basically matches the shape of the likelihood. To find the posterior, we can just add up the exponents:

$$P(\theta \mid D) \propto P(D \mid \theta) P(\theta) \propto \theta^k (1-\theta)^{n-k} \theta^{a-1}(1-\theta)^{b-1} = \theta^{k+a-1}(1-\theta)^{n-k+b-1}$$

If you can select a prior that makes the posterior look the same, it's called a "conjugate prior" for that likelihood. Conjugate priors are cool because they make the math easier, as we just saw.

Now, the prior has its own set of parameters known as "*hyper-parameters*." These parameters express any prior beliefs we might hold. If we're in the dark about θ, we might go for a uniform prior, which is like a no-preference, neutral choice. As you can tell from its definition, the uniform distribution can be thought of as a Beta with $a = b = 1$. On the flip side, if we've tossed this coin before and found θ to have an average of 0.3 and a standard deviation of 0.1, that translates to $a = 6,\ b = 14$.

Posterior: the posterior can be express as:

$$P(\theta \mid D) \propto \text{Binomial}(k \mid n, \theta). \text{Beta}(\theta \mid a, b) = \text{Beta}(\theta \mid \text{k} + \text{a}, \text{n} - \text{k} + \text{b})$$

Notice that the -1 terms vanish for a and b because they're baked into the Beta definition. What's fascinating about the Binomial-Beta combo is that we can consider all these terms as counts in an experiment. n and k are actual observed counts, but the prior acts like we've also seen some counts of our own. In the context of this statistical model, they're dubbed "pseudo counts." So, the prior has its own set of (pseudo) sample size $c = a + b$, which is similar to n.

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

n, k = 8, 5

# Create a figure for the plots
fig = plt.figure(figsize=(10, 8))

# Plot 1: Flat prior
a, b = 1, 1
ax = fig.add_subplot(221)
theta = np.linspace(1e-3, 1 - 1e-3, 100)

# Likelihood, prior, and posterior plots
ax.plot(theta, beta.pdf(theta, k + 1, n - k + 1), label='Likelihood',
color='blue')
ax.plot(theta, beta.pdf(theta, a, b), ls='--', label='Prior',
color='green')
ax.plot(theta, beta.pdf(theta, k + a, n - k + b), label='Posterior',
color='red')

# Set axis label
ax.set_xlabel(r'$\theta$')

# Add a legend for the plots
ax.legend()

# Set the title for the subplot
ax.set_title('Flat Prior')

# Plot 2: Low prior
a, b = 6, 14
ax = fig.add_subplot(222)

# Likelihood, prior, and posterior plots
ax.plot(theta, beta.pdf(theta, k + 1, n - k + 1), label='Likelihood',
color='blue')
ax.plot(theta, beta.pdf(theta, a, b), ls='--', label='Prior',
color='green')
ax.plot(theta, beta.pdf(theta, k + a, n - k + b), label='Posterior',
color='red')

# Set axis label
ax.set_xlabel(r'$\theta$')
```

```
# Add a legend for the plots
ax.legend()

# Set the title for the subplot
ax.set_title('Low Prior')

# Show the figure
plt.tight_layout()
plt.savefig('/content/drive/MyDrive/PSU/ColabNotebooks/Example3_4.png',
dpi=300)
plt.show()
```

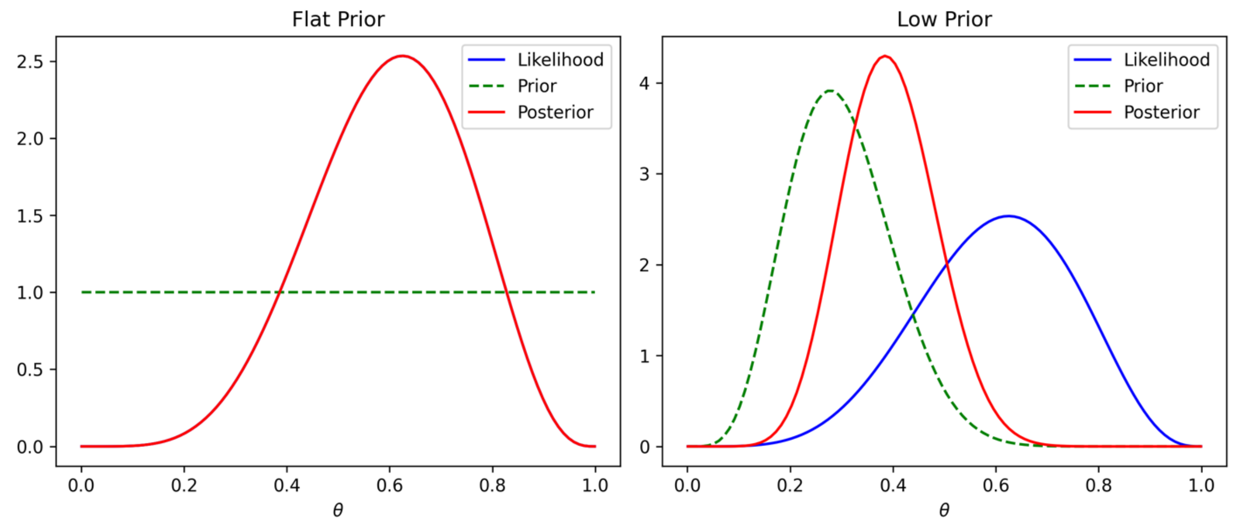


Figure 3.4

You can observe from the indistinguishable curves on the left that a flat prior has no impact on the posterior, in contrast to the assumption of a strongly biased coin on the right.

The mean of the posterior and Maximum a Posterior (MAP) estimate are expressed as follows:

$$\bar{\theta} = \frac{k+a}{n+c} \qquad \theta_{MAP} = \frac{k+a-1}{n+c-2}$$

In this case, we notice the count plus pseudo-count terms once more. If we opt for a uniform prior, with $a = b = 1$, then the MAP estimate simplifies to the Maximum Likelihood Estimate (MLE), representing solely the empirical fraction of heads:

$$\theta_{MLE} = \frac{k}{n}$$

The posterior mean can be expressed as a convex combination of both the prior mean and the Maximum Likelihood Estimate (MLE):

$$\mathbb{E}(\theta \mid D) = \lambda \frac{a}{c} + (1 - \lambda)\theta_{MLE}$$

Here, λ is defined as $c/(n + c)$**.** In simpler terms, the posterior represents a compromise between our initial beliefs and the information conveyed by the data, with the influence of the data growing stronger as n increases, assuming c remains constant.

The variance of the (Beta) posterior is:

$$\mathrm{var}(\theta \mid D) = \frac{(a + k)(b + n - k)}{(a + k + b + n - k)^2(a + k + b + n - k + 1)}$$

For high counts, $n \gg a, b$, that simplifies to

$$\mathrm{var}(\theta \mid D) \approx \frac{k(n - k)}{n^3} = \frac{\theta(1 - \theta)}{n}$$

The uncertainty of our estimate decreases proportionally to $1/\sqrt{n}$ which bears a resemblance to the characteristics of the Normal distribution. This convergence is not a fortuity; the Beta distribution approaches the Normal distribution as the sample size increases. This phenomenon was not only noted but also rigorously established through a specific experiment – repeated coin tosses conducted without prior assumptions. This experiment, conducted by Abraham de Moivre in 1733, led to the discovery of the Normal approximation to the Binomial/Beta distribution. Subsequently, in 1810, Pierre Simon Laplace extended this finding, leading to the formulation of the concept we now recognize as the Central Limit Theorem.

Furthermore, an intriguing observation is that the variance reaches its maximum at $\theta = 1/2$ and converges to zero as θ approaches 0 or 1, independently of the sample size n. This aligns with the insights derived from information theory, where we analyze the entropy of a single coin toss, representing a scenario with only two possible outcomes in the ensemble.

Example 1:

One classic example where frequentist and Bayesian statistics lead to the same result is estimating the probability of success in a binary trial (success/failure). Let's say you have conducted a coin flip experiment, and you want to estimate the probability of getting heads.

Frequentist Approach:

In the frequentist approach, you would estimate the probability of getting heads by calculating the relative frequency of heads in a large number of coin flips. For example, if you flipped the coin 100 times and got 55 heads, the frequentist estimate of the probability of getting heads would be 55/100, which is 0.55.

Bayesian Approach:

In the Bayesian approach, you start with a prior belief about the probability of getting heads before seeing any data. A commonly used prior is the Beta distribution, which is a conjugate prior for the binomial distribution (the distribution of a binary trial). You can use a $Beta(1,1)$ prior, which is a uniform (flat) prior over the range [0, 1]. After observing data, you update your belief using Bayes' theorem to obtain a posterior distribution.

Suppose you observed 55 heads in 100 flips, as in the frequentist example. In Bayesian terms, your posterior distribution would be $Beta(55 + 1, 100 - 55 + 1) = Beta(56, 46)$. The mode of this posterior distribution is (56-1)/(46+56-2) = 0.55.

So, in this specific example, the frequentist estimate (0.55) and the Bayesian estimate (0.55) are very close and essentially lead to the same result.

Please note that the choice of prior in the Bayesian approach can influence the result, and with a different prior, the Bayesian estimate may differ from the frequentist estimate. However, as more data is collected, the influence of the prior diminishes, and the two approaches tend to converge.

Example 2:

Estimating Annual Flood Risk for a Region: Suppose civil engineers want to estimate the annual flood risk for a specific region prone to flooding. They have historical flood data, and they want to estimate the probability of a flood exceeding a certain level each year.

Frequentist Approach:

Based on historical flood data for the region, they find that out of the last 100 years, there have been 10 years with floods exceeding the threshold. Using the frequentist approach, they calculate the annual flood risk as follows:

Annual Flood Risk (Frequentist) = (Number of Years with Floods > Threshold) / (Total Number of Years)

Annual Flood Risk (Frequentist) = 10 / 100 = 0.10 or 10%

So, the frequentist estimate of the annual flood risk is 10%.

Bayesian Approach:

Engineers have a prior belief about the flood risk, which is represented by a Beta distribution. Their prior belief is $Beta(8, 92)$, where 8 represents past "successes" (floods above the threshold) and 92 represents past "failures" (floods below the threshold).

This year, they observe a flood exceeding the threshold. To update their prior with this new data and calculate the Bayesian estimate, they can use Bayes' theorem. The posterior distribution will also be a Beta distribution with updated parameters.

Posterior = Prior * Likelihood

Posterior = $Beta(8, 92) * Binomial(1, p)$, where p is the unknown flood risk.

Now, you want to calculate the mode (most probable value) of the posterior distribution, which represents the Bayesian estimate of the annual flood risk given the observed data.

To calculate the mode, you can use the formula for the mode of a Beta distribution with parameters α and β:

Mode (Posterior) = (α - 1) / (α + β - 2)

In this case:

α (alpha) = 8 (prior successes)

β (beta) = 92 (prior failures)

Mode (Posterior) = (8) / (8 + 92 - 2) = 8 / 100 = 0.081 or 8.1%

So, the Bayesian estimate of the annual flood risk, after updating with the observed flood, is approximately 8%, and you can calculate this mode using the formula provided. This result differs from the frequentist estimate of 10% due to the influence of the prior belief and the observed data.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta, binom

# Define the prior parameters (Beta distribution)
alpha_prior = 8
beta_prior = 92
```

```
# Define the likelihood parameters (Binomial distribution)
n_observed = 1  # One observed flood event exceeding the threshold
p_values = np.linspace(0, 1, 1001)  # A range of possible flood risk
values

# Calculate the prior, likelihood, and posterior
prior = beta.pdf(p_values, alpha_prior, beta_prior)
likelihood = binom.pmf(n_observed, 1, p_values)  # Probability of
observing 1 flood event given p
posterior = prior * likelihood

# Normalize the posterior to make it a valid probability distribution
posterior /= np.trapz(posterior, p_values)

# Calculate the mode of the posterior (Bayesian estimate)
mode_posterior = p_values[np.argmax(posterior)]

# Plot the prior, likelihood, and posterior distributions
plt.figure(figsize=(10, 6))
plt.plot(p_values, prior, label='Prior (Beta)')
plt.plot(p_values, likelihood, label='Likelihood (Binomial)')
plt.plot(p_values, posterior, label='Posterior')
plt.axvline(mode_posterior, color='red', linestyle='--', label='Mode
(Bayesian Estimate)')
plt.xlabel('Flood Risk')
plt.ylabel('Probability Density')
plt.legend()
plt.title('Prior, Likelihood, and Posterior Distributions')
plt.grid()
plt.show()

print(f'Bayesian estimate of flood risk (mode of posterior):
{mode_posterior:.2%}')
```

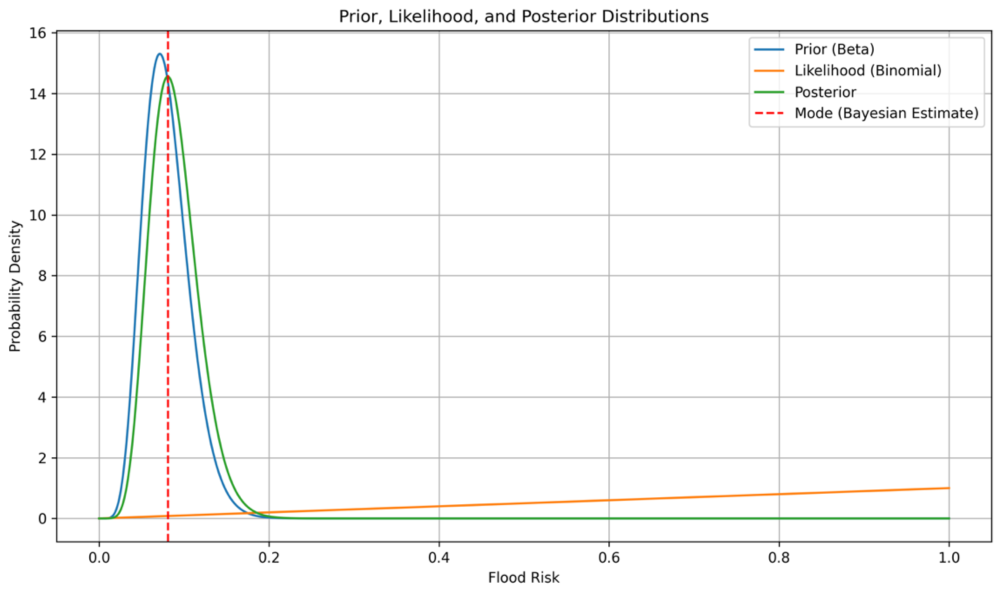


Figure 3.5

```
Bayesian estimate of flood risk (mode of posterior): 8.10%
```

3.4 Homework #3 (10 points)

You have two options: either question 1 (with a bonus question) or question 3. Please choose one.

1- Assuming the following likelihood function:

$$P(x \mid \lambda) = \frac{1}{Z(\lambda)}\left[\frac{1}{\lambda}\right] e^{-x/\lambda} \qquad \text{if} \quad x \,\epsilon\, [1,20]$$

where

$$Z(\lambda) = \int_{1}^{20} dx \frac{1}{\lambda} e^{-x/\lambda} = e^{-1/\lambda} - e^{-20/\lambda}$$

(a) Write a Python script and visualize $P(x \mid \lambda)$ for $\lambda = 2$, $\lambda = 5$ and $\lambda = 10$ ***(3 points)***

(b) Write a Python script and visualize $P(x \mid \lambda)$ for $x = 2$, $x = 5$ and $x = 10$ ***(3 points)***

(c) Assume following is the posterior for the observations $D = \{3, 7, 2, 12, 5\}$

$$P(\lambda \mid D) = \frac{P(D \mid \lambda)P(\lambda)}{P(D)} \propto \frac{1}{\left(\lambda(Z(\lambda))\right)^N} \exp\,(-\sum_i \frac{x_i}{\lambda})P(\lambda)$$

Write a Python script and visualize $P(\lambda \mid D)$ for i) a flat prior assumption and ii) a Gaussian prior assumption, $P(\lambda) = N(10, 5^2)$

(4 points)

2- Bonus- analytically solve the following log-likelihood: ***(2 points)***

$$\mathrm{Log}\,\mathcal{L}(D|F) = -\frac{1}{2}\sum_{i=1}^{N}\left[\mathrm{Log}\,(2\pi e_i^2) + \frac{-(F_i - F)^2}{e_i^2}\right]$$

and proof that it results in the following relationships:

$$\hat{F} = \frac{\sum w_i F_i}{\sum w_i}, \qquad w_i = {}^1\!/_{e_i^2}$$

3- Coin toss problem: Drive the posterior distribution of parameter θ, which is the probability that a coin toss results in heads. Suppose that the prior distribution is an uninformative (or flat) Beta distribution with parameters of $a = 1$ and $b = 1$. We use Binomial likelihood function to quantify the data from our experiment which resulted in 4 heads out of 10 tosses.

a) Use Monte Carlo (MC) approach. Visualize the MC evolution and the density plot for θ based on 1000 iterations. Use np.random.seed(5) and the proposal distribution that follows a Gaussian distribution with a mean and standard deviation of 0.5 and 0.05, respectively. Please discuss what the results would be if we increased the iterations to 10000. ***(2 points)***

b) Use Markov Chain Monte Carlo (MCMC) approach. Visualize the MCMC evolution and the density plot for θ based on 1000 iterations. Use np.random.seed(5) and the proposal distribution that follows a Gaussian distribution with a mean and standard deviation of 0.5 and 0.05, respectively. Please discuss what the results would be if we increased the iterations to 10000. ***(2 points)***

c) Use Markov Chain Monte Carlo (MCMC) with Metropolis-Hasting (MH) approach. Visualize the MCMC evolution and the density plot for θ based on 1000 iterations. Use np.random.seed(5) and the proposal distribution that follows a Gaussian distribution with a mean and standard deviation of 0.5 and

0.05, respectively. Please discuss what the results would be if we increased the iterations to 10000 with a burn-in period of 200. ***(6 points)***

i) Compare the density plot of θ provided by MCMC-MH with the theoretical posterior distribution, and discuss the results.