Name (please print legibly!) _____

Gravitational Waves: Assignment 6
Bayesian Analyses and Parameter Estimation

1. Probability Review:

   (a) Derive Bayes' theorem from the Kolmogorov axioms.

   (b) Show that Bayes' theorem can be written as

   $$P(A|B) = \frac{P(B|A)P(A)}{\sum_i P(B|A_i)P(A_i)} \tag{1}$$

   for any B and A_i disjoints and such that $\cup_i A_i = S$.

2. A gravitational-wave signal from a binary system can be characterized by more than 15 parameters. We can separate the parameters θ such that the template waveform can be written as $h(t;\theta) = a h_a(t;\xi)$. Thus, θ has been separated into an amplitude term a and into $h_a(t;\xi)$ which has the remaining parameters ξ. If

   $$\log \Lambda(s|\theta_t) = (h_t|s) - \frac{1}{2}(h_t|h_t), \tag{2}$$

   then compute the following:

   (a) the maximum likelihood estimate $\hat{a}_{\rm ML}(s)$ by requiring $\partial \log \Lambda / \partial a = 0$.

   (b) $\log \Lambda(s|\xi)$.

3. Consider the future when the LIGO and Virgo detectors have achieved their design sensitivity and gravitational waves from binary black hole detections are occurring regularly. We want to be able to measure the astrophysical rate of these black holes mergers. To do so, we need to be able to count the number that we've detected in a given amount of time at this fixed sensitivity. Let's say that we've finished 50 half-year experiments. While we known the number of mergers can vary a bit, let's say that the true number of mergers that we should detect in a half-year experiment is 100. Because this is a counting experiment, a Poisson distribution is a good approximation to the measurement process.

   (a) Generate some data for these `N=50` experiments and see if you can make a plot like the one shown in Fig. 1. Measurements of the rates `R` from a Poisson distribution can be achieved with `R = scipy.stats.poisson(R_true).rvs(N)` where `R_true` is the true rate value of 100. Note that you can set numpy.random.seed() at the beginning of your code for reproducibility. Also include error bars on each rate number measured from the 50 experiments. Errors e_i on Poisson counts can be estimated via the square root of each measurement: $e_i = \sqrt{R_i}$.

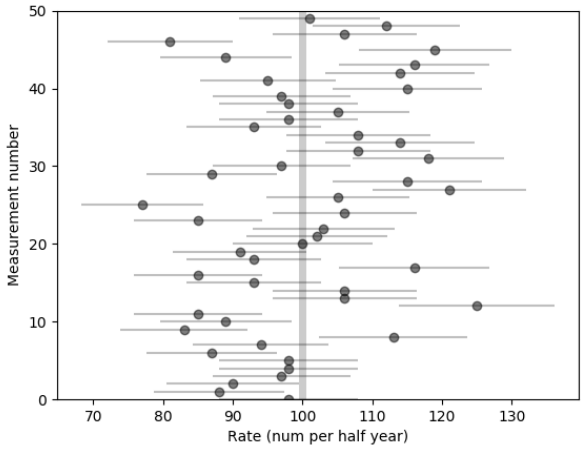

Figure 1: Repeated experimental results

(b) **Frequentist Maximum Likelihood Approach:** Given a single observation of the rate $D_i = (R_i, e_i)$, we can compute the probability distribution of the measurements given the true rate R_{true} and our assumption of Gaussian errors:

$$P(D_i|R_{\text{true}}) = \frac{1}{\sqrt{2\pi e_i^2}} \exp\left[\frac{-(R_i - R_{\text{true}})^2}{2e_i^2}\right] \tag{3}$$

which can be read as "the probability of data D_i given R_{true} equals...". In the frequentist sense, the likelihood function can be constructed by computing the product of probabilities for each data point:

$$\mathcal{L}(D|R_{\text{true}}) = \prod_{i=1}^{N} P(D_i|R_{\text{true}}). \tag{4}$$

Because the value of the likelihood can become very small, it is often more convenient to compute the log-likelihood.

i. Combine Eq. 3 and Eq. 4, compute the natural log, and then perform the maximization analytically by setting $d \ln \mathcal{L}/dR_{\text{true}} = 0$. You should find that the observed estimate of R_{true} is given by

$$R_{\text{estimated}} = \frac{\sum w_i R_i}{\sum w_i} \tag{5}$$

where w_i is a weight given by $w_i = 1/e_i^2$.

ii. It can be shown that the standard deviation of a Gaussian approximation to the likelihood curve at maximum is:

$$\sigma_{\text{estimated}} = \frac{1}{\sqrt{\sum_{i=1}^{N} w_i}}. \tag{6}$$

Evaluate Eq. 5 and Eq. 6 and report the result as $R_{\text{estimated}} \pm \sigma_{\text{estimated}}$. Is this consistent with $R_{\text{true}} = 100$?

(c) **Bayesian Approach:** With this method, what we really want to compute is $P(R_{\rm true}|D)$. To do this, Bayesian's apply Bayes' Theorem:

$$P(R_{\rm true}|D) = \frac{P(D|R_{\rm true})P(R_{\rm true})}{P(D)}. \tag{7}$$

For the one parameter problem considered here, we will compute $P(R_{\rm true}|D)$ as a function of $R_{\rm true}$ using the Markov Chain Monte Carlo (MCMC) sampling method. The solution below will be outlined using the Python `emcee` library (`http://dfm.io/emcee/current/`) but you are welcome to explore other sampling methods. To perform the MCMC, start by defining the following functions in your code in terms of an array of parameters θ, which for this case is just $\theta = [R_{\rm true}]$:

i. The log prior $P(R_{\rm true})$. Let's just use a flat prior here:

```
def log_prior(theta):
       return 1
```

ii. The log likelihood $P(D|R_{\rm true})$ as we saw in Eq. 4 as a function of our experimental results R_i and e_i:

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

iii. The log posterior $P(R_{\rm true}|D)$ which will simply be the sum of $\ln P(R_{\rm true})$ and $\ln P(D|R_{\rm true})$:

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

iv. Now we can set up the problem, including setting some initial parameters and starting guesses for 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 explore parameter
               space and  stabilize
nsteps = 2000  # number of MCMC steps to take

# Start with some random locations between 0 and 500
starting_guesses = 500 * numpy.random.rand(nwalkers, ndim)
```

v. Import the necessary module `emcee` which requires the logarithm of the probability density functions. This is why we defined `log_posterior(theta, R, e)` above. The first argument of `log_posterior` is the position of a single walker. The other

arguments come from the `args` parameter of the `emcee.EnsembleSampler`. Do the production run with `emcee.EnsembleSampler.run_mcmc` in 2000 steps:

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

vi. The sampler has a property `EnsembleSampler.chain` that is a numpy array with shape = (nwalkers, nsteps, ndim). You can use this to obtain a `sample` array that is reshaped into a flat list and that has the burn-in points discarded:

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

vii. Plot a histogram of `sample` with `bins=50` and overlay it with a best-fit Gaussian distribution. This can be achieved with `stats.norm(numpy.mean(sample), numpy.std(sample)).pdf(R_fit)` where `R_fit` is the range of points to fit the Gaussian. See if you can obtain a plot like Fig. 2.

viii. Report the result as `numpy.mean(sample)` $\pm$ `numpy.std(sample)`. Is this consistent with $R_{\rm true} = 100$? How does this compare to the Frequentist approach?

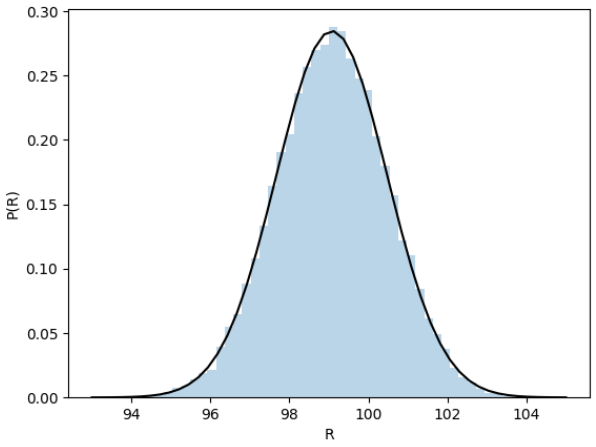

Figure 2: Samples drawn from the normal posterior distribution.