Final

Jinhong Du - 12243476

June 5, 2020

Contents

There are two questions. Question 1 is worth 30 points and Question 2 is worth 50 points.

Your answer should include all code as an appendix; however, do not expect graders to necessarily read your code. Thus, include in your main report a description of what you did and give analytic expressions for quantities you computed.

You may consult books and internet reference sources, but you must not seek help from others and must cite any sources you use. On page one of your report, include the statement: "I affirm the all of the work submitted herein is my own, and I have not offered or received any help from others on this exam."

I affirm the all of the work submitted herein is my own, and I have not offered or received any help from others on this exam.

Problem 1

Suppose you observe independent and identically distributed data

$$X_1, \dots, X_N \sim \mathrm{Beta}(a, b)$$

where the parameters a, b are to be estimated. Here the density of the Beta(a, b) distribution is $f(x) \propto x^{a-1}(1-x)^{b-1}$.

Recall that the mean, m, of the Beta(a, b) distribution is $m := \frac{a}{a+b}$, so one can also parameterize the Beta distribution in terms of m and $n := a + b$. That is, if I tell you (m, n) then you can work out (a, b) and vice-versa.

Suppose now that you want to do Bayesian inference for (m, n), with independent priors on m and n:

$$\begin{aligned} m &\sim U[0, 1] \\ n &\sim \Gamma(\text{shape} = 0.1, \text{rate} = 0.1). \end{aligned}$$

i) Outline and implement a Metropolis–Hastings algorithm, with random walk proposal, that samples from the posterior distribution for (m, n) given the observed data. Use the following `R` code as a template (filling in the ...).

```
# x is a vector of data of length N (X_1,\dots,X_N).
# niter is an integer;
# startval is a vector of length 2 containing starting values for m and n;
# proposal_sd is a vector of length 2, containing standard deviations
# for updating m and n respectively
mySampler = function(x, niter, startval, proposal_sd){
    m = rep(0,niter)
    m[1] = startval[1]
    n = rep(0,niter)
    n[1] = startval[2]
    for(i in 2:niter){
        current_m = m[i-1]
        current_n = n[i-1]
        proposed_m = rnorm(1,mean=current_m,sd=proposal_sd[1])
        proposed_n = rnorm(1,mean=current_n,sd=proposal_sd[2])
        ...
    }
    return(list(m=m,n=n))
}
```

The posterior distribution of m, n given data $\boldsymbol{X}$ is

$$p(m,n|\boldsymbol{X}) \propto p(X|m,n)p(m)p(n)$$
$$= \prod_{i=1}^{N}\left(\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}X_i^{a-1}(1-X_i)^{b-1}\right)\mathbb{1}_{[0,1]}(m)\frac{0.1^{0.1}n^{0.1-1}e^{-0.1n}}{\Gamma(0.1)}\mathbb{1}_{(0,\infty)}(n).$$

We choose the symmetric Gaussian kernels Q_m and Q_n with proposal standard deviations σ_m and σ_n for m and n respectively. The MH procedure is given by

1. Initialize, m_1 and n_1.
2. For $i = 2, \dots,$
   (a) Sample $m \sim Q_m(\cdot|m_{i-1})$ and $n \sim Q_n(\cdot|n_{i-1})$.
   (b) Compute the acceptance probabilty $A = \min\left\{1, \frac{p(X|m,n)p(m)p(n)}{p(X|m_{i-1},n_{i-1})p(m_{i-1})p(n_{i-1})}\right\}$.
   (c) With probabilty A accept the proposal values and set $m_i = m$ and $n_i = n$. Otherwise, set $m_i = m_{i-1}$ and $n_i = n_{i-1}$.

To avoid computational issues, we calculate the log-densities instead, e.g. `dbeta(x,a,b,log=True)`, `dgamma(n,0.1,0.1,log=True)`. And then we are able to compute $\log A$.

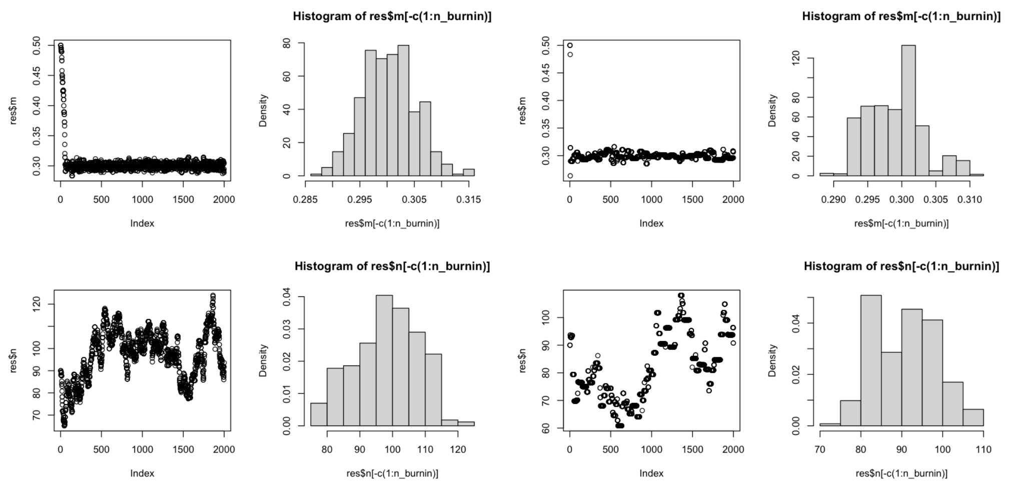
[1]:
```r
log_likelihood <- function(x, m, n){
    a <- m * n
    b <- n - a
    return(sum(dbeta(x, a, b, log=TRUE)))
}

mySampler <- function(x, niter, startval, proposal_sd){
    m <- rep(0,niter)
    m[1] <- startval[1]
    n <- rep(0,niter)
    n[1] <- startval[2]
    for(i in 2:niter){
        current_m <- m[i-1]
        current_n <- n[i-1]
        proposed_m <- rnorm(1,mean=current_m,sd=proposal_sd[1])
        proposed_n <- rnorm(1,mean=current_n,sd=proposal_sd[2])
        if(current_m<0 | current_m>1 | current_n<=0){
            logA <- Inf
        }
        else if(proposed_m<0 | proposed_m>1 | proposed_n<=0){
            logA <- -Inf
        }
        else{
            logdeno <- dgamma(current_n,shape=0.1,rate=0.1,log=TRUE) +␣
↪log_likelihood(x,current_m,current_n)
            logA <- dgamma(proposed_n,shape=0.1,rate=0.1,log=TRUE) +␣
↪log_likelihood(x,proposed_m,proposed_n) - logdeno
        }
        if(log(runif(1))<logA){
            m[i] <- proposed_m;n[i] <- proposed_n
        }else{
            m[i] <- current_m;n[i] <- current_n
        }
    }
    return(list(m=m,n=n))
}
```

ii) Describe how choice of the values in `proposal_sd` will affect i) the stationary distribution; ii) the mixing/convergence of the algorithm. Illustrate your answer by applying your algorithm to simulated data (simulated as you like), using different illustrative values for `proposal_sd`.

1. Larger proposal standard deviation produces a stationary distribution with heavier tail for the corresponding parameter. Also, it may not be unimode since the proposed values are varied very much.
2. Larger proposal standard deviation requires more iterations to converge.

For the following plots, we set $N = 100$, $m = 0.3, n = 100$ and initial values of m and n to be 0.5 and 80. First we use `proposal_sd`=(0.01,2) (the first two columns in the below plots).

Then we use `proposal_sd`=(0.1,5) (the last two columns in the below plots).



As we can see, with larger standard deviation, the convergence of n gets slower. The stationary distributions on the right hand side also look rugged.

[2]:
```r
generate_sample <- function(N, m, n){
    a <- m * n
    b <- n - a
    return(rbeta(N,a,b))
}

N <- 100
m <- 0.3
n <- 100
niter <- 2000
n_burnin <- 1000
set.seed(0)
x <- generate_sample(N,m,n)
startval <- c(0.5, 90)

proposal_sd <- c(0.01,2)
```

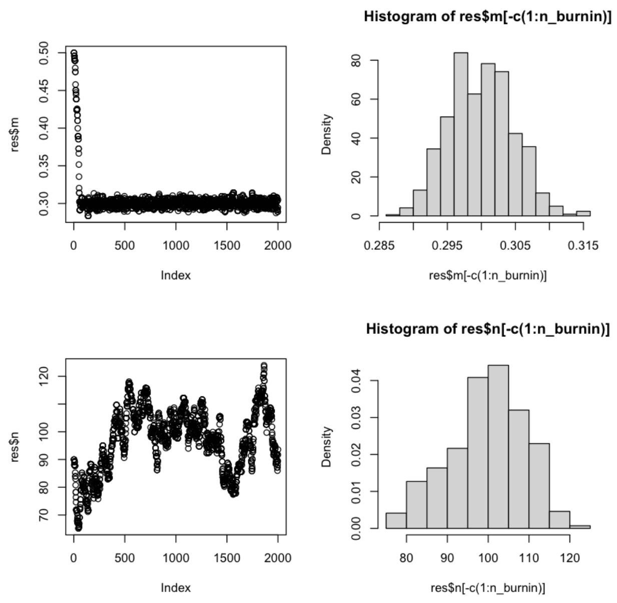
```r
res <- mySampler(x, niter, startval, proposal_sd)
par(mfrow=c(2,2))
plot(res$m)
hist(res$m[-c(1:n_burnin)], prob=T)
plot(res$n)
hist(res$n[-c(1:n_burnin)], prob=T)

proposal_sd <- c(0.1,5)
res <- mySampler(x, niter, startval, proposal_sd)
par(mfrow=c(2,2))
plot(res$m)
hist(res$m[-c(1:n_burnin)], prob=T)
plot(res$n)
hist(res$n[-c(1:n_burnin)], prob=T)
```

iii) Generate data under the model (1) with $N = 100$ and $m = 0.3, n = 100$, and apply your MCMC sampler to the simulated data. Perform appropriate convergence checks to select appropriate values for burn-in, number of iterations, and `proposal_sd` to ensure that the MCMC results are likely reliable. When you are comfortable with the results, use the sampler to obtain a a 90% posterior credible interval (CI) for each of the parameters m, n, a and b.

We set the initial values of m and n to be 0.5 and 80. First we choose suitable proposal standard deviations $(0.01, 2)$, as the magnitude of m is small and the one of n is large.

We can see that it converges very quickly after about 300 steps. So it is ok to have 2000 iterations and burnin the first 300 samples.

[3]:
```
N <- 100
m <- 0.3
n <- 100
niter <- 2000
n_burnin <- 300
set.seed(0)
x <- generate_sample(N,m,n)
startval <- c(0.5, 90)

proposal_sd <- c(0.01,2)
res <- mySampler(x, niter, startval, proposal_sd)
par(mfrow=c(2,2))
plot(res$m)
hist(res$m[-c(1:n_burnin)], prob=T)
plot(res$n)
hist(res$n[-c(1:n_burnin)], prob=T)
```



[4]:
```
est_m <- res$m[-c(1:n_burnin)]
est_n <- res$n[-c(1:n_burnin)]
est_a <- est_m*est_n
est_b <- est_n-est_a

mean(est_m)
quantile(est_m, c(0.05,0.95))
mean(est_n)
quantile(est_n, c(0.05,0.95))
mean(est_a)
quantile(est_a, c(0.05,0.95))
mean(est_b)
quantile(est_b, c(0.05,0.95))
```

0.299995459175625

5% 0.292887739407053 **95%** 0.307550566879464

99.8156630616193

5% 82.8803879813372 **95%** 113.926222526334

29.9424022940976

5% 24.9279495551082 **95%** 34.1778998592754

69.8732607675217

5% 58.1314938017207 **95%** 79.7493344386972

iv) Repeat the simulations and CI computation 100 times and check what proportion of these 100 simulations the posterior CIs contain the true values. (Assessing convergence separately for each of these 100 simulations is impractical, so use the number of iterations and burn-in you selected above). What does theory say, if anything, about what proportion of posterior CIs will cover the true values?

In theory, the coverage rate of posterior CI of each parameter should be 90%.

[5]:
```
for(N in c(100,1000)){
    a <- m*n
    b <- n-a
    n_simulation <- 100
    is_in_CI <- matrix(0, nrow = n_simulation, ncol = 4)
    set.seed(0)
    for(i in 1:n_simulation){
        x <- generate_sample(N,m,n)
        res <- mySampler(x, niter, startval, proposal_sd)
        est_m <- res$m[-c(1:n_burnin)]
        est_n <- res$n[-c(1:n_burnin)]
        est_a <- est_m*est_n
        est_b <- est_n-est_a

        if((m>=quantile(est_m, 0.05)) & (m<=quantile(est_m, 0.
 ↪95))){is_in_CI[i,1]<-1}
        if((n>=quantile(est_n, 0.05)) & (n<=quantile(est_n, 0.
 ↪95))){is_in_CI[i,2]<-1}
        if((a>=quantile(est_a, 0.05)) & (a<=quantile(est_a, 0.
 ↪95))){is_in_CI[i,3]<-1}
        if((b>=quantile(est_b, 0.05)) & (b<=quantile(est_b, 0.
 ↪95))){is_in_CI[i,4]<-1}
    }
    cat(apply(is_in_CI, 2, mean))
    cat(mean((is_in_CI[,1]==1)&(is_in_CI[,2]==1)), '\n')
}
```

```
0.98 0.48 0.51 0.48 0.46
0.93 0.87 0.88 0.87 0.82
```

As we can see when $N = 100$, the individual coverage rates of n, a, b are very low. While for $N = 1000$, all individual coverage rates are near 0.9.

Problem 2

Consider the data in the file `COVID-19_Cases_US.csv` which I downloaded from https://coronavirus-resources.esri.com/datasets/628578697fb24d8ea4c32fa0c5ae1843_0. The data set includes counts of confirmed cases, deaths, and incident rates of COVID-19 at a county level, together with location data (latitude and longitude) for each county. In this question we will conduct two main analyses, the first aimed at examining the evidence for variation in the death rate (deaths per confirmed case) among counties; and a second to examine the spatial correlation in incident rates across the US. There are many complications that a full analysis would have to take account of, not least the problem of under-reporting of cases, which likely varies among counties, and the fact that death rates lag behind cases. However, for the purposes of this exam you may largely ignore these practical issues. (However, you will have to deal somehow with a few practical issues, such as the fact that the data set has missing values.)

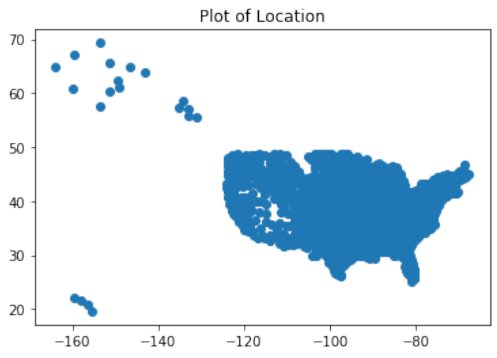
(a) Begin by reading the data into `R` (or your preferred statistical analysis environment), and plot some exploratory plots to familiarize yourself with the data (including features such as missing data) and check that they have been read in correctly. (Hint: Plot more plots than you include in your report!). As you complete the following questions you may want to filter out problematic data, e.g. because of missing-ness or anomalies. Describe any processing/filtering of the data you perform and your reasoning.

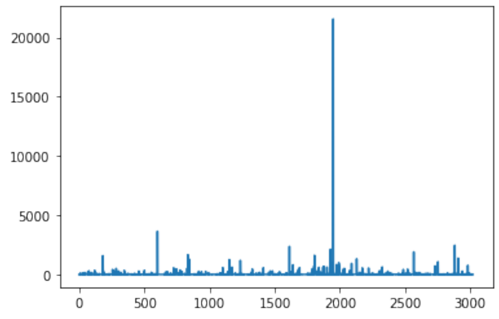
We drop the samples with missing values in X and Y. Also, we drop some anomalies whose confirmed cases are less than death cases.

After that, there are still some missing values of incident rate. Since this will not effect our analysi in (b) we just keep them, and we will drop them at the beginning of (c).

[6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('COVID-19_Cases_US.csv')

mean_death_rate = df['Deaths'].sum()/df['Confirmed'].sum()
len(df)

df = df[~(pd.isna(df['X'])&pd.isna(df['Y'])&(df['Confirmed']<df['Deaths']))]
print(len(df))
plt.scatter(*df[['X','Y']].values.T)
plt.title('Plot of Location')
plt.show()

df['Deaths'].plot()
plt.show()
```

```
3015
```



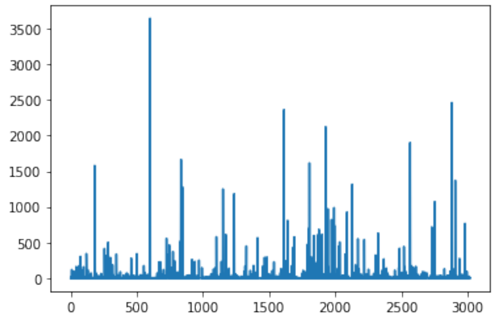


The first plot above is the locations of all counties. The locations are in the right range.

The second plot above is the number of death cases, there is one county with very large number of deaths, whicn is `New York`. Since it is an outlier, we would be better to remove it.

```
[7]: print(np.sum(df['Deaths']>10000))
     df = df[df['Deaths']<10000]
     df['Deaths'].plot()
     plt.show()
```

```
1
```



Now it is better.

(b) In this subquestion we will assess the variation in death rate per case among counties. Let i index county, and let d_i and c_i denote, respectively, the number of deaths and confirmed cases in county i. Assume the model

$$d_i|c_i, q_i \sim \text{Binomial}(c_i, q_i),$$

where we will refer to q_i as the "death rate" parameter for county i. I computed the average death rate across the whole country as $104245/1786780 \approx 0.0583$.

i. Use the Benjamini–Hochberg method to identify a set of counties whose death rate appears to be higher than the average (0.0583), controlling the FDR at 0.05. Repeat this procedure to find counties whose death rates appear lower than the average at the same FDR. Produce plots comparing the spatial locations of the "significant" high-death-rate and low-death-rate counties with those that are not significant.

Let $q_0 = \frac{104245}{1786780}$. Since $d_i|c_i, q_i \sim \text{Binomial}(c_i, q_i)$ for $i = 1, \dots, n$, for hypothesis test $H_0 : q_i = q_0$ versus $H_1 : q_i > q_0$, the p value of this test is given by

$$p_i = \mathbb{P}(d_i \geq D_i | c_i, q_0) = \sum_{j=D_i}^{c_i} \binom{c_i}{j} q_0^j (1 - q_0)^{c_i - j}$$

where D_i is the realization of d_i.

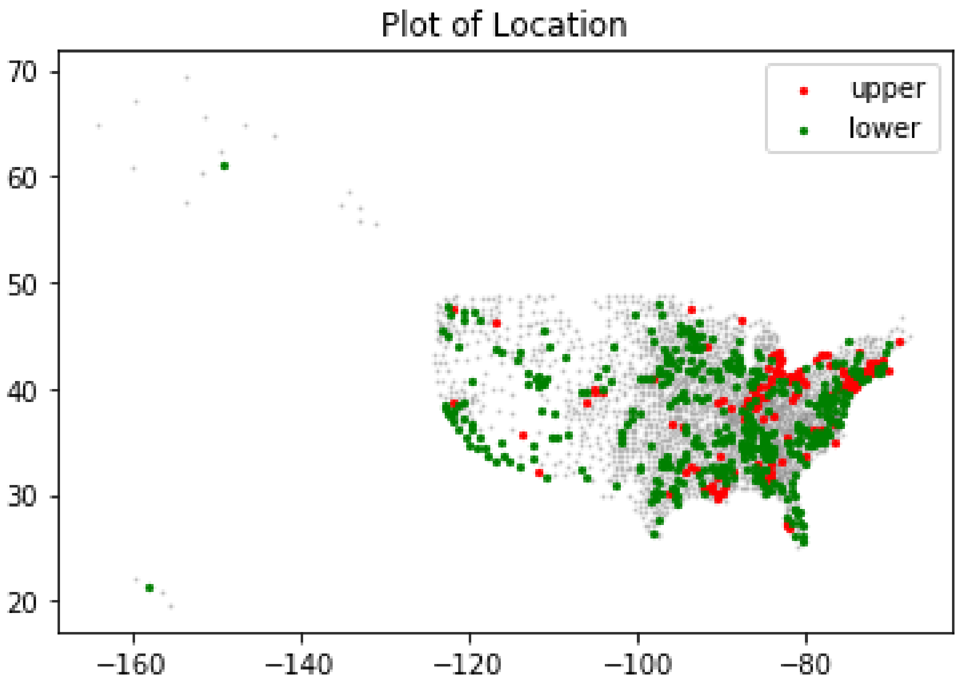
Analogously, the p value of hypothesis test $H_0 : q_i = q_0$ versus $H_1 : q_i < q_0$ is given by

$$p_i = \mathbb{P}(d_i \leq D_i | c_i, q_0) = \sum_{j=0}^{D_i} \binom{c_i}{j} q_0^j (1 - q_0)^{c_i - j}.$$

[8]:
```python
from scipy.stats import binom
c = df['Confirmed'].values
D = df['Deaths'].values
q0 = mean_death_rate

def BH(p, alpha):
    m = len(p)
    gamma = np.zeros(m, dtype=int)
    sort_id = np.argsort(p)
    sort_p = p[sort_id]

    k = np.where(sort_p <= (np.arange(m)+1) * alpha / m)[0]
    if len(k)>0:
        gamma[sort_id[:k[-1]+1]] = 1
    return gamma


p_upper = 1 - binom.cdf(D, c, q0) + binom.pmf(D, c, q0)
gamma_upper = BH(p_upper, 0.05)

p_lower = binom.cdf(D, c, q0)
```

```
gamma_lower = BH(p_lower, 0.05)

print(np.sum(gamma_upper), np.sum(gamma_lower))

plt.scatter(*df[['X','Y']].values.T, s=1, c='gray', alpha=0.3)
plt.scatter(*df[gamma_upper==1][['X','Y']].values.T, label='upper',c='r',s=5)
plt.scatter(*df[gamma_lower==1][['X','Y']].values.T, label='lower',c='g',s=5)
plt.title('Plot of Location')
plt.legend()
```

```
140 447
```



As we can see, the gray points are non-significant, while the red and green points are significant high-death-rate and low-death-rate counties. It seems the significant high-death-rate counties are near some big states and cities like IL, LA, New York, etc. Those non-significant counties are mainly in the west.

ii. Outline an Empirical Bayes approach to this problem, using a conjugate beta prior, $q_i \sim \text{Beta}(a, b)$, to model the variation in q_i. Implement this approach to estimate parameters a, b and the posterior distribution $p(q_i|d_i, c_i, \hat{a}, \hat{b})$. [You may fix the mean of the Beta distribution to the overall average 0.0583 if you like.] Use your implementation to identify counties that have a high probability (> 0.95) of having q_i exceed 0.0583. Similarly identify counties that have a high probability of having death rate less than 0.0583. Compare your results with those from the BH analysis. Comment on any differences and possible reasons for them.

Since $d_i|c_i, q_i \sim \text{Binomial}(c_i, q_i)$ and $q_i \sim \text{Beta}(a, b)$, we have

$$\begin{aligned}
p(d_i|c_i, a, b) &= \int p(d_i, q_i|c_i, a, b)\mathrm{d}q_i \\
&= \int p(d_i|c_i, q_i)p(q_i|a, b)\mathrm{d}q_i \\
&= \int \binom{c_i}{d_i} q_i^{d_i}(1-q_i)^{c_i-d_i}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}q_i^{a-1}(1-q_i)^{b-1}\mathrm{d}q_i \\
&= \frac{\Gamma(c_i+1)}{\Gamma(d_i+1)\Gamma(c_i-d_i+1)}\cdot\frac{\Gamma(d_i+a)\Gamma(c_i-d_i+b)}{\Gamma(c_i+a+b)}\cdot\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)},
\end{aligned}$$

which turns out to be the density of a Beta-Binomial(c_i, a, b) random variable (https://en.wikipedia.org/wiki/Beta-binomial_distribution). The MLE of a, b can be found numerically. If we fix the mean $\frac{a}{a+b} = 0.0583$, then we will only need to consider one unknown parameter $e = a + b$.

After solving $\hat{e}$ the MLE of e, we can obtain point estimate of q_i (e.g. posterior mean, mode, etc.) from its posterior distribution assuming its prior is $\text{Beta}(\hat{a}, \hat{b})$ where $\hat{a} = 0.0583\hat{e}$ and $\hat{b} = 0.9417\hat{e}$. The posterior distribution of q_i is given by

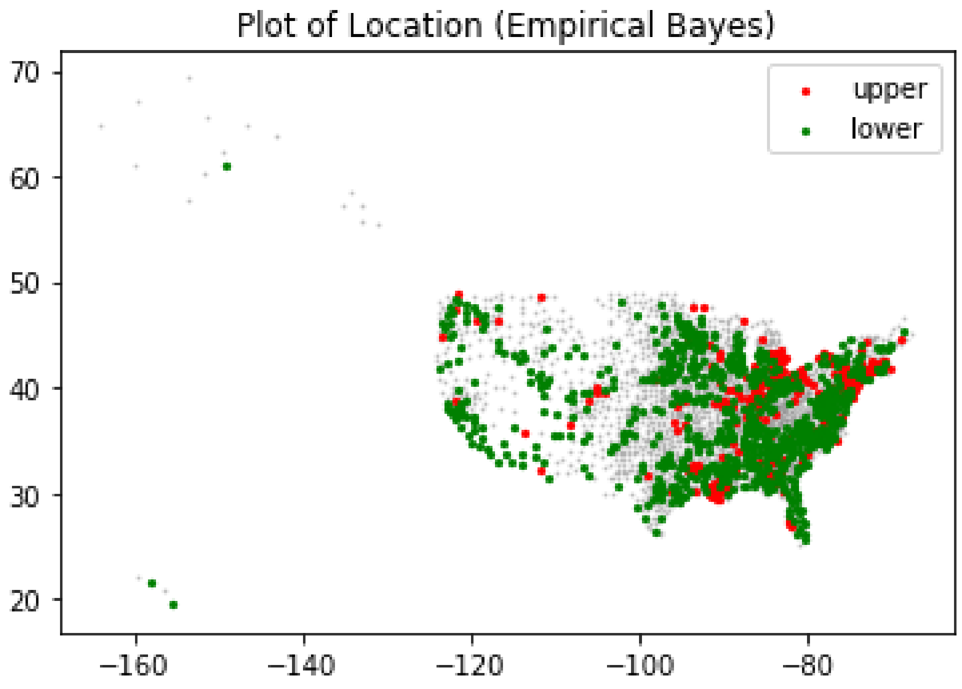
$$\begin{aligned}
p(q_i|d_i, c_i, \hat{a}, \hat{b}) &\propto p(d_i|c_i, q_i)p(q_i|\hat{a}, \hat{b}) \\
&= \binom{c_i}{d_i} q_i^{d_i}(1-q_i)^{c_i-d_i}\frac{\Gamma(\hat{a}+\hat{b})}{\Gamma(\hat{a})\Gamma(\hat{b})}q_i^{\hat{a}-1}(1-q_i)^{\hat{b}-1} \\
&\propto q_i^{d_i+\hat{a}-1}(1-q_i)^{c_i-d_i+\hat{b}-1},
\end{aligned}$$

i.e. $q_i|d_i, c_i, \hat{a}, \hat{b} \sim \text{Beta}(d_i + \hat{a}, c_i - d_i + \hat{b})$.

```
[9]: from scipy.stats import betabinom, beta
     from scipy.optimize import minimize

     def neg_log_likelihood(pars, d, c, m):
         a = np.exp(pars) * m
         b = np.exp(pars) - a
         L = np.sum(betabinom.logpmf(d, c, a, b))
         return -L


     def eb(d, c, m):
         res = minimize(neg_log_likelihood, np.zeros(1), args=(d,c,m),␣
      ↪method='L-BFGS-B')
```

```
    a = np.exp(res.x) * m
    b = np.exp(res.x) - a
    return a,b

a_hat, b_hat = eb(D,c,q0)
p_upper_eb = 1 - beta.cdf(q0, D+a_hat, c-D+b_hat)
gamma_upper_eb = (p_upper_eb>0.95)

p_lower_eb = beta.cdf(q0, D+a_hat, c-D+b_hat)
gamma_lower_eb = (p_lower_eb>0.95)

print(np.sum(gamma_upper_eb), np.sum(gamma_lower_eb))
plt.scatter(*df[['X','Y']].values.T, s=1, c='gray', alpha=0.3)
plt.scatter(*df[gamma_upper_eb][['X','Y']].values.T, label='upper',c='r',s=5)
plt.scatter(*df[gamma_lower_eb][['X','Y']].values.T, label='lower',c='g',s=5)
plt.title('Plot of Location (Empirical Bayes)')
plt.legend()
```

```
229 788
```



As we can see, the numbers of significant high-death-rate and low-death-rate counties computed by Empirical Bayes are larger than those computed by BH procedure.

(c) In this subquestion we will model the log of the incident rate in each county as a spatial Gaussian process. Note that the incident rate is a column in the data (and different from the death rate we looked at above). I suspect incident rate is computed by dividing the number of confirmed cases by the population of the county and scaling by 10^5 to give "cases per $100,000$ population". However, I have been unable to confirm this. Because fitting GPs can be quite computationally intensive, you may choose to fit the model to only a random subset of counties (eg try 200-500 counties to begin with and assess what is practical).

Let Y_i denote the log of the incident rate for county i. The goal is to fit the model

$$Y_i = f_i + e_i$$

where f denotes a spatial Gaussian process (GP) with squared-exponential kernel $K(d) = \lambda^2 e^{-\frac{d^2}{2l}}$ and constant mean μ; and the elements of e are independent and identically distributed $\mathcal{N}(0, \sigma^2)$. The intuition here is that f captures the "spatial" part of the variation, where e captures the "non-spatial" part. For example, if $\lambda = 0$ then $f_i = 0$ and we obtain an entirely "non-spatial" model.

i. Give an expression for the likelihood for the (hyper-)parameters μ, λ, l, σ. Write code to compute this likelihood, and to (numerically) obtain the maximum likelihood estimates. Give a brief summary of how the code works. (Do not simply use a package for fitting GPs here; however, it is fine to make use of existing routines to compute multivariate Gaussian densities, for linear algebra operations, and for numerical optimization.) [Hint: if optimizing over all four parameters numerically is too challenging, try fixing μ to the sample mean of Y_i. If it is still too challenging try fixing σ^2 to its mle under the non-spatial model.]

Since $f|\mu, \lambda, l \sim \mathcal{N}(\mu, K)$ and $e|\sigma \sim \mathcal{N}(0, \sigma^2 I)$ are independent, we have $Y|\mu, \lambda, l, \sigma \sim \mathcal{N}(\mu, K + \sigma^2 I)$ and the likelihood is given by

$$p(Y|\mu, \lambda, l, \sigma) = \frac{1}{(2\pi)^n |K + \sigma^2 I|^{\frac{1}{2}}} e^{-\frac{1}{2}(Y-\mu)^\top (K+\sigma^2 I)^{-1}(Y-\mu)}.$$

We define d as the geometry distance between each two counties with unit 10^3 km.

```
[10]: df = df[~pd.isna(df['Incident_Rate'])]

      print(np.sum(df['Incident_Rate']==0))
      df = df[df['Incident_Rate']>0]

      idx = np.random.choice(len(df), 500, replace=False)
      Y = np.log(df.iloc[idx]['Incident_Rate'].values)
      print(len(df))

      from sklearn.metrics.pairwise import haversine_distances
      from math import radians
      # geo distance (unit: 1e3 km)
      d = haversine_distances(df.iloc[idx][['Lat','Long_']].applymap(radians).values)␣
       ↪* 6371000/1e6
```

```
24
2940
```

There are a few counties having zero confirmed cases, we juset drop them. Then we randomly select 500 samples.

[11]:
```
from scipy.stats import multivariate_normal

def neg_log_likelihood(pars, Y, d2):
    n = len(Y)
    mu = pars[0]
    lamb, l, sigma = np.exp(pars[1:])
    K = lamb**2 * np.exp(-d2/2/l)
    L = np.sum(multivariate_normal.logpdf(Y, mu*np.ones(n), K+sigma**2*np.
 ↪eye(n)))
    return -L
```

ii. Demonstrate your code on data that you simulate from the model (3), to show that it can obtain reasonable parameter estimates. Compare the loglikelihood achieved by your fitted parameter values with the log-likelihood achieved by the true parameter values.

Here I set $(\mu, \lambda, l, \sigma) = (2, 0.5, 1.5, 1.5)$ and generate Y based on distance d computed from our sampled data. It turns out the estimated log-likelihood is similar to the true log-likelihood, and the estimated values of $(\mu, \lambda, l, \sigma)$ are also similar to the true values.

[12]:
```
mu, lamb, l, sigma = 2, 0.5, 1.5, 1.5
n = len(Y)
K = lamb**2 * np.exp(-d**2/2/l)
f = multivariate_normal.rvs(mean=mu*np.ones(n), cov=K+sigma**2*np.eye(n),␣
 ↪size=n)
e = multivariate_normal.rvs(size=n)
simu_Y = f+e
true_log_L = np.sum(multivariate_normal.logpdf(simu_Y, mu*np.ones(n),␣
 ↪K+sigma**2*np.eye(n)))

res = minimize(neg_log_likelihood, np.zeros(4), args=(simu_Y,d**2),␣
 ↪method='L-BFGS-B')
mu_hat = res.x[0]
lamb_hat, l_hat, sigma_hat = np.exp(res.x[1:])
print(true_log_L, -res.fun)
print(mu_hat, lamb_hat, l_hat, sigma_hat)
```

```
-520069.7394093586 -508698.2747959581
2.035047052215925 0.48383054848417045 1.2975097132258762 1.8325451682044804
```

iii. Apply your code to the COVID-19 incident rates to estimate the hyperparameters. Report the estimated values you obtain. *Again, to emphasize, because fitting GPs can be quite computationally intensive, you may choose to fit the model to only a random subset of counties.* Assess the importance of a spatial component by comparing the log-likelihood you achieve under the full model with the log-likelihood under the non-spatial model ($f_i = 0$).

The estimated values are $(\hat{\mu}, \hat{\lambda}, \hat{l}, \hat{\sigma}) \approx (4.91, 0.86, 0.026, 0.891)$ for sampled data. We can see that the log-likelihood of the full model is much larger than the non-spatial model.

[13]:
```
res = minimize(neg_log_likelihood, np.zeros(4), args=(Y,d**2),␣
 ↪method='L-BFGS-B')
mu_hat = res.x[0]
lamb_hat, l_hat, sigma_hat = np.exp(res.x[1:])

print(mu_hat, lamb_hat, l_hat, sigma_hat)
print(-res.fun)
print(np.sum(multivariate_normal.logpdf(simu_Y, mu_hat*np.ones(n),␣
 ↪sigma_hat**2*np.eye(n))))
```

```
4.905405958880524 0.857579657633926 0.025741037214198903 0.8912801440567765
-738.6933172684674
-2121693.0468437895
```

iv. Explain how to use your fitted GP to predict, for each county in your subset, its log incident rate given the data at all other counties. Plot the predicted values against the observed values and report the mean squared error (MSE) of your predictions. Compare the MSE you achieve with this "spatially-informed" predictor, with the MSE achieved by the "naive predictor" that simply uses the overall mean of Y_i. Under what circumstances would the naive predictor be expected to perform competitively with the spatially-informed predictor?

First note that $Y \sim \mathcal{N}(\mu \mathbf{1}_n, K + \sigma^2 I_n)$. From https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Conditional_distributions, we can able to compute $\mathbb{E}(Y_i|Y_{-i})$ where Y_{-i} denote the length-$(n-1)$ random vector formed by remove the ith entry of Y. More specifically,

$$\mathbb{E}(Y_i|Y_{-i} = a) = \mu + \Sigma_{i,-i}\Sigma_{-i,-i}^{-1}(a - \mu \mathbf{1}_{n-1})$$

where $\Sigma = K + \sigma^2 I_n$, $\Sigma_{i,-i} \in \mathbb{C}^{1\times n-1}$ is formed by remove the i th entry of the ith row of Σ, and $\Sigma_{-i,-i}$ is formed by remove the ith row and ith column of Σ.

[14]:
```
cond_mean = np.zeros(n)
K = lamb_hat**2 * np.exp(-d**2/2/l_hat)
Sigma = K + sigma_hat**2*np.eye(n)
for i in range(n):
    cond_mean[i] = mu_hat +  np.delete(Sigma[i,:], i, axis=-1) @ \
        np.linalg.inv(np.delete(np.delete(Sigma, i, axis=-1), i, axis=0)) @ \
        (np.delete(Y,i)-mu_hat)

print(np.mean((cond_mean - Y)**2), np.mean((np.mean(Y) - Y)**2))
```

```
0.9841266677633335 1.541768860039991
```

We can see that the MSE of the spatially-informed predictor is smaller than the naive predictor in the sampled data. When the spatial variations are small, i.e., a spatial component is not important, the two methods will perform competitively.