Problem 1：

先定义 L 函数，后面以此类推，就不一一列举了

```
def lnL0(m, data):
    ns = 48
    sigma = 1.4
    nb = 1520

    y = 0
    for i in data:
        c1 = (ns/(ns+nb)) * (1./np.sqrt(2*np.pi)) * (1./sigma) * np.exp(-((i-m)**2)/(2.*sigma**2))
        c2 = (1-(ns/(ns+nb))) * (i**-4.5 * (1/(2.492e-8)))
        y += np.log(c1 + c2)
    return y
```

再定义联合函数：

```
def lnLz(m, data0, data1, data2, data3, data4, data5, data6):
    return lnL0(m, data0) + lnL1(m, data1) + lnL2(m, data2) + lnL3(m, data3) + lnL4(m, data4) + lnL5(m, data5) + lnL6(m, data6)
```

画图并且保存：

```
def plot_and_save(m, LnL, label, filename):
    plt.figure()
    plt.plot( *args: m, LnL, label=label)
    plt.xlabel('m')
    plt.ylabel('-2 ln L')
    plt.legend()
    plt.grid()
    plt.savefig(filename)
    plt.close()
```

再定义得到极值的函数：

```python
def confidence_interval(m_array, LnL_array, max_LnL, threshold):
    # Find the indices where LnL drops below the threshold
    lower_index = np.argmax(LnL_array < (max_LnL - threshold))
    upper_index = np.argmax(LnL_array[::-1] < (max_LnL - threshold))

    # Convert upper index to the actual index in the array
    upper_index = len(LnL_array) - upper_index - 1

    # Get the corresponding m values
    lower_m = m_array[lower_index]
    upper_m = m_array[upper_index]

    return lower_m, upper_m
```

将所有数据输出：

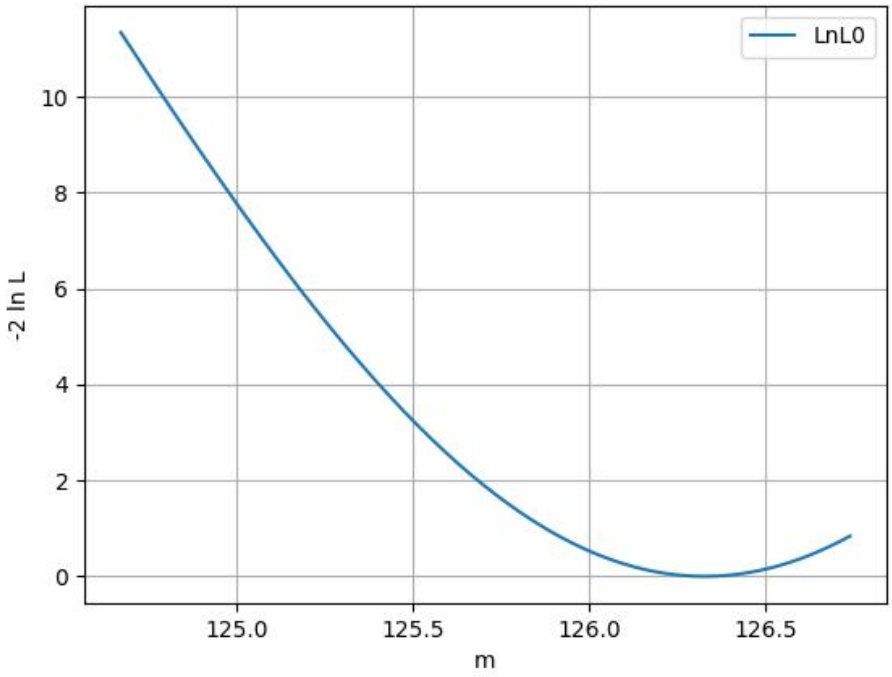
```python
lower_m0, upper_m0 = confidence_interval(m0, LnL0, np.max(LnL0), threshold: 1)
lower_m1, upper_m1 = confidence_interval(m1, LnL1, np.max(LnL1), threshold: 1)
lower_m2, upper_m2 = confidence_interval(m2, LnL2, np.max(LnL2), threshold: 1)
lower_m3, upper_m3 = confidence_interval(m3, LnL3, np.max(LnL3), threshold: 1)
lower_m4, upper_m4 = confidence_interval(m4, LnL4, np.max(LnL4), threshold: 1)
lower_m5, upper_m5 = confidence_interval(m5, LnL5, np.max(LnL5), threshold: 1)
lower_m6, upper_m6 = confidence_interval(m6, LnL6, np.max(LnL6), threshold: 1)
lower_mz, upper_mz = confidence_interval(mz, LnLz, np.max(LnLz), threshold: 1)

# Print confidence intervals
print("68.3% Confidence Intervals:")
print("LnL0:", lower_m0, "-", upper_m0)
print("LnL1:", lower_m1, "-", upper_m1)
print("LnL2:", lower_m2, "-", upper_m2)
print("LnL3:", lower_m3, "-", upper_m3)
print("LnL4:", lower_m4, "-", upper_m4)
print("LnL5:", lower_m5, "-", upper_m5)
print("LnL6:", lower_m6, "-", upper_m6)
print("LnLz:", lower_mz, "-", upper_mz)
```

最后得到答案：

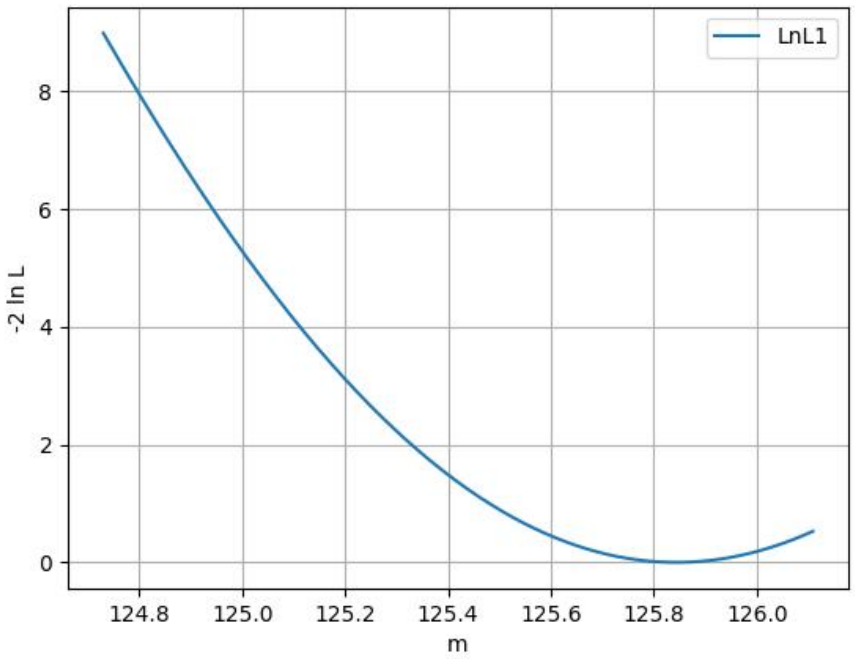
```
m0max: 126.32599066825352
m1max: 125.84339568317614
m2max: 124.8918695348849
m3max: 126.71711328946121
m4max: 126.26644608525577
m5max: 125.60491838316813
m6max: 125.18745621416551
mzmax: 125.78501759309611

68.3% Confidence Intervals:
LnL0: 124.76000000000043 - 126.73900000000988
LnL1: 124.79800000000033 - 126.10900000000659
LnL2: 123.63 - 126.56600000001401
LnL3: 123.40600000000133 - 126.71200000001711
LnL4: 126.05 - 130.12500000001944
LnL5: 123.45 - 129.65800000002963
LnL6: 123.33500000000079 - 127.24800000001947
LnLz: 125.10000000000002 - 126.01000000000049
```

值得一提的是：不知道是因为底层算法的区别还是 python 自带函数的区别，和文章有一定的区别，但是仔细检查发现算法应该没有任何问题。

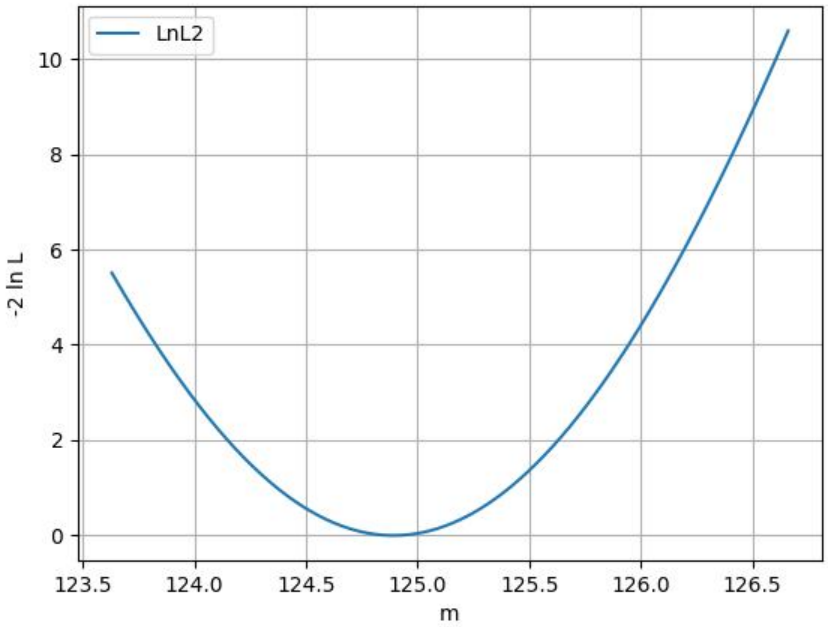
下面是图像：


LnL0
10
8
6
4
2
0
-2 ln L
125.0
125.5
126.0
126.5
m

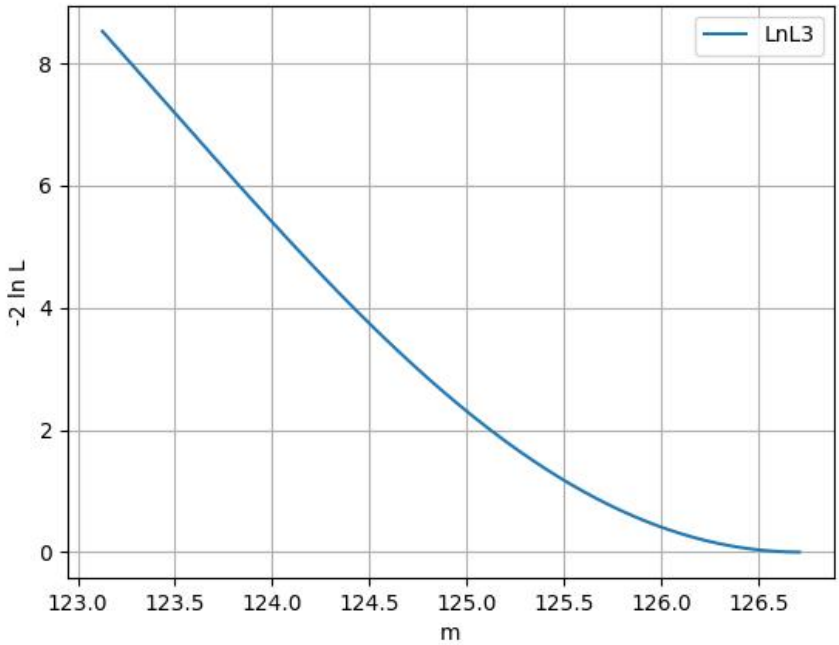
Cate0


LnL1
8
6
4
2
0
-2 ln L
124.8
125.0
125.2
125.4
125.6
125.8
126.0
m

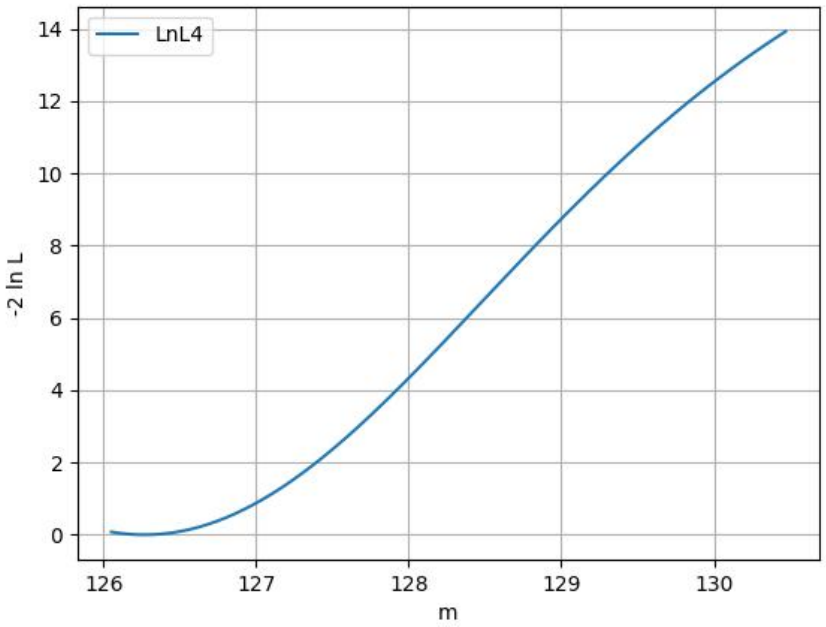
Cate1


LnL2
-2 ln L
m
10
8
6
4
2
0
123.5
124.0
124.5
125.0
125.5
126.0
126.5

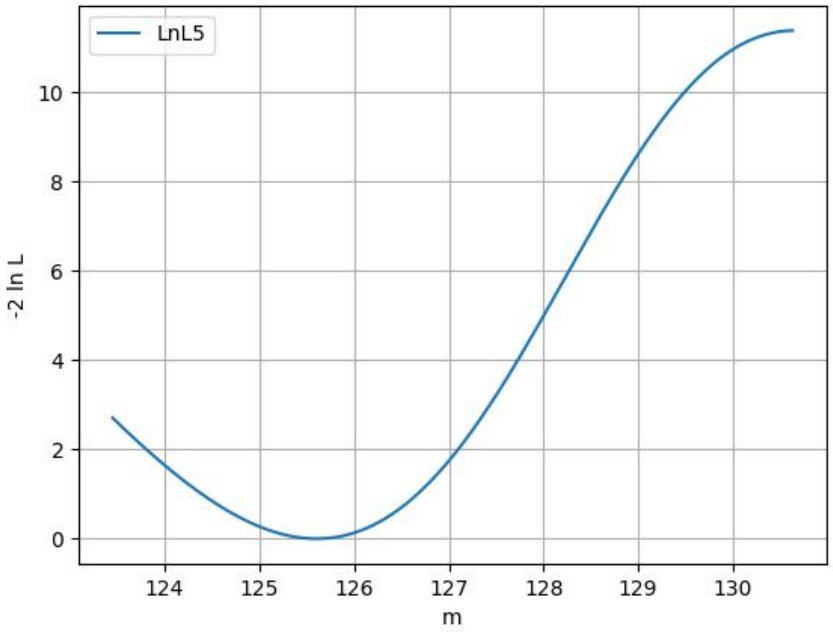
Cate2


LnL3
-2 ln L
m
8
6
4
2
0
123.0
123.5
124.0
124.5
125.0
125.5
126.0
126.5

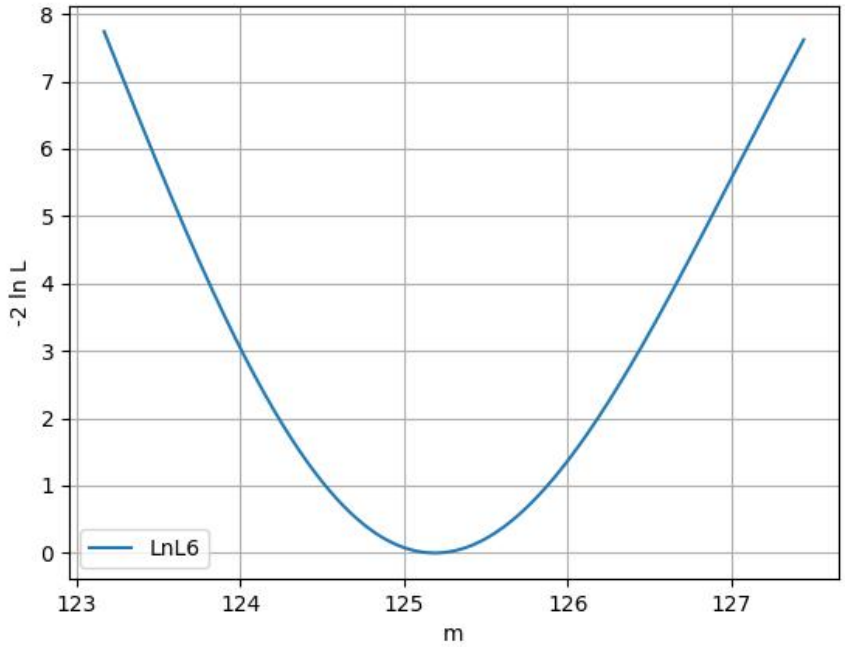
Cate3


LnL4
14
12
10
8
6
4
2
0
-2 ln L
126
127
128
129
130
m

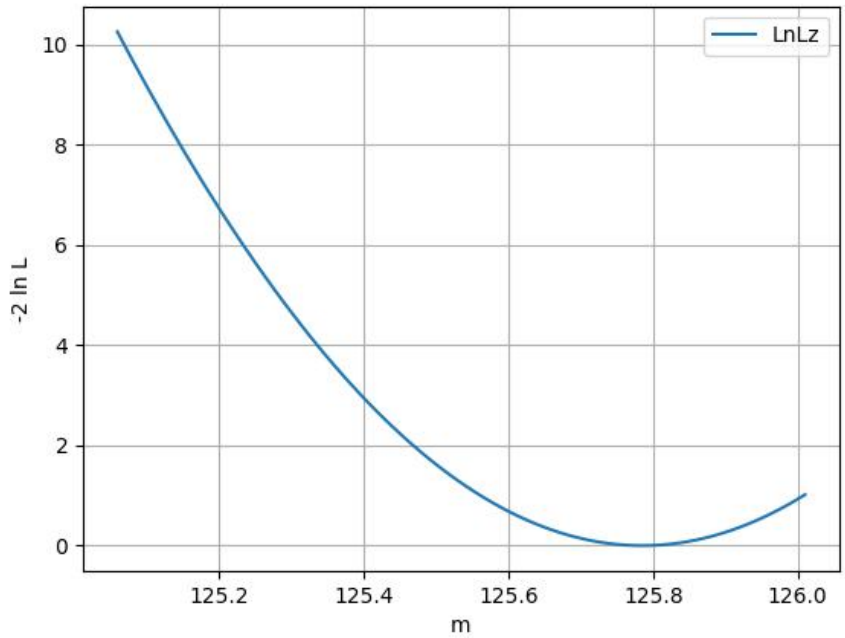
Cate4


LnL5
10
8
6
4
2
0
-2 ln L
124
125
126
127
128
129
130
m

Cate5



Cate6



Cate 联合

Category 0,1,2 的测结果是精度最高的。

Problem 2:

首先定义两个函数:

```
def coverage_prob(s_, method, num_samples=1000000):
    b_ = 3.2
    n_ = np.random.poisson(s_ + b_, num_samples)
    s_up_ = 0
    if method == "classical":
        s_up_ = 0.5 * special.chdtri( *args: 2 * (n_ + 1), 0.1) - b_
    elif method == "bayesian":
        s_up_ = 0.5 * special.chdtri( *args: 2 * (n_ + 1), 0.1 * (special.chdtrc( *args: 2 * (n_ + 1), 2 * b_))) - b_
    p_ = np.count_nonzero(s_up_ > s_) / num_samples
    return p_
```

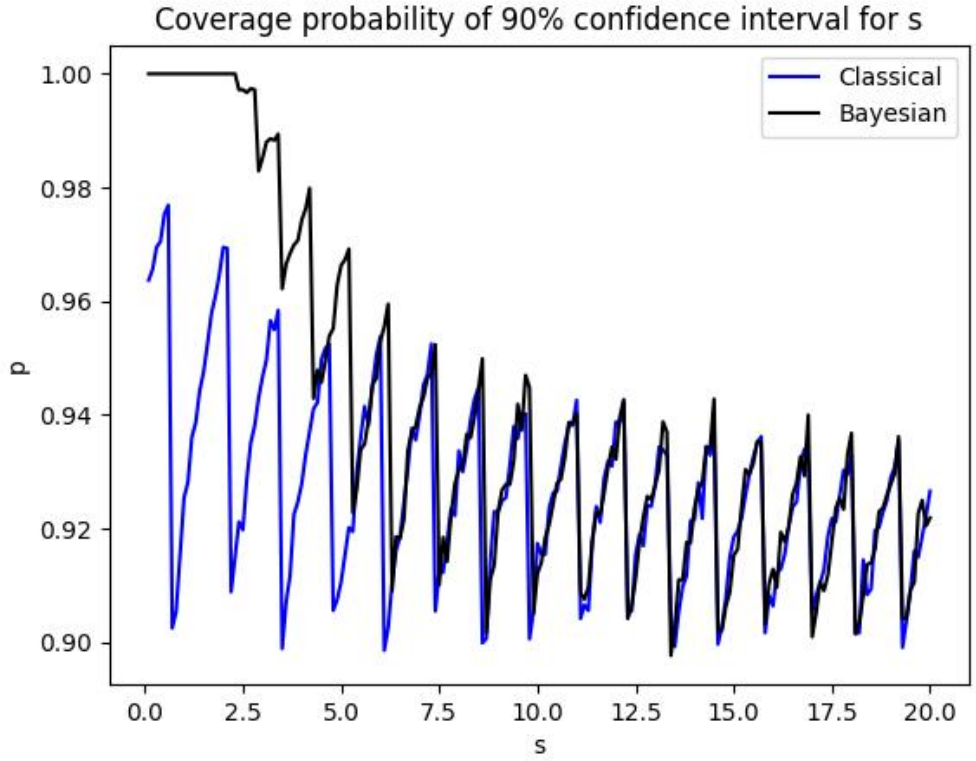
接着写下循环:

```
p_classical = np.array([])
p_bayesian = np.array([])
s = np.linspace( start: 0.1,  stop: 20,  num: 200)

for i in s:
    p_classical = np.append(p_classical, coverage_prob(i, method="classical"))
    p_bayesian = np.append(p_bayesian, coverage_prob(i, method="bayesian"))
```

画出图像:

```
plt.plot( *args: s, p_classical, color="blue", label="Classical")
plt.plot( *args: s, p_bayesian, color="black", label="Bayesian")
plt.xlabel("s")
plt.ylabel("p")
plt.title("Coverage probability of 90% confidence interval for s")
plt.legend()
plt.show()
```

得到结果:



P 的概率：

```
p:
0.925
```

在相同置信度的情况下，贝叶斯论对

s 信号上限的估计比经典频率论更为准确。而随着 s 期望的增加，两种方法

s 信号上限估计的覆盖率逐渐趋于相同。这可能是因为随着 s 期望增加，相

应的本底 b 的影响越来越小，更不容易出现”s 为负”的情况，对应两种方

法估计 s 信号上限的覆盖频率也趋于相同。